A Multi-Level Approach to NBTI Mitigation in Processors

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

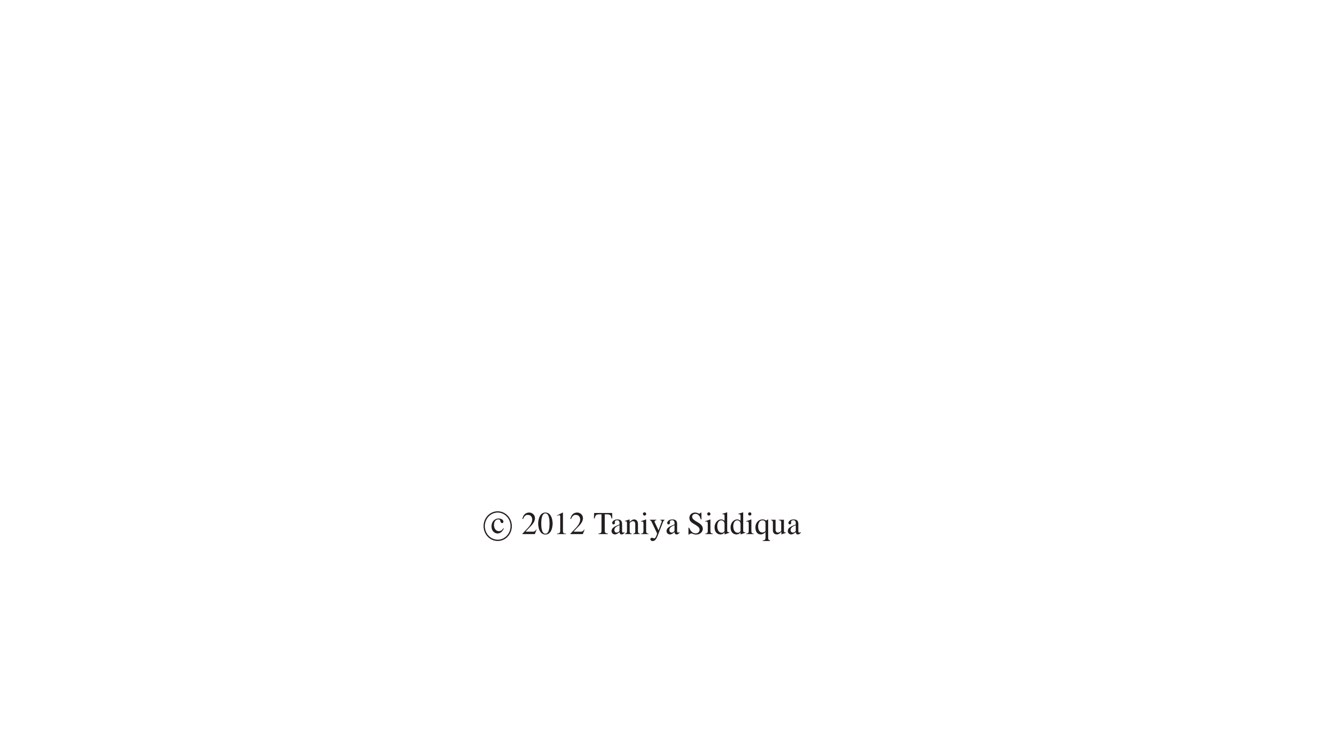
of the requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Taniya Siddiqua

December 2012



© 2012 Taniya Siddiqua

Abstract

We are in the era of multicore processors and it is expected that the number of the processing cores on a chip will steadily increase over the next decade, driven by Moore's Law. While technology scaling has benefitted high performance, the scaling has a dark side too: a degradation in the reliability of silicon devices. Processors have become highly susceptible to a variety of reliability problems in silicon, such as particle induced soft errors and hard errors. Therefore, processors have to be designed to provide adequate protection against these reliability problems while maintaining high performance and energy efficiency. Designing a reliable computer system is a large and complex multi-dimensional and multi-level problem, comprising of different hardware blocks, reliability phenomena, design layers, metrics, and optimization techniques. This dissertation considers a key emerging reliability phenomenon: Negative Bias Temperature Instability (NBTI). This dissertation develops NBTI mitigation techniques for the logic and memory structures in the processor that impose very little performance, power, and area overheads. This research also creates the foundation for understanding NBTI in the context of one other important processor reliability problem: process variations.

APPROVAL SHEET

The dissertation

is submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

TANIYA SIDDIQUA

AUTHOR

The dissertation has been read and approved by the examining committee:

Advisor

Accepted for the School of Engineering and Applied Science:

Dean, School of Engineering and Applied Science

December

2012

To my loving parents and siblings

Acknowledgements

It would not have been possible to write this dissertation without the help and support of many people around me and I owe my gratitude to all those people.

My deepest gratitude is to my advisor, Professor Sudhanva Gurumurthi. I have been extraordinarily fortunate to have him as my advisor for his sage guidance. He was always a source of inspiration. He gave me the freedom to explore on my own, and at the same time the advice to recover when I swayed away from the right direction. He taught me how to question thoughts and express ideas. No matter what, he was always accessible and despite my uncountable mistakes, he always encouraged me to do the proper thing with patience. I am also thankful to him for encouraging the use of correct grammar and consistent notation in my writings and for carefully reading and commenting on countless revisions of every manuscript I have written over my graduate life. It was always a pleasure to work with him.

I would like to thank Professor Mircea R. Stan who was always there to listen and give advice. I am deeply grateful to him for the long discussions that helped me sort out the technical details of my work. His insightful comments and constructive criticisms at different stages of my research were thought provoking and they helped me focus my ideas. I would like to thank my other committee members Professor Kevin Skadron,

Professor Joanne B. Dugan, Professor Mary L. Soffa for their guidance over the years. I would also like to take this opportunity to thank Dr. Athanasios Papathanasiou with whom I worked for a short duration as an intern, but his continuous encouragement and advice helped me boost my confidence and made me ready for the corporate world.

I would like to thank my best friends Anindita and Sonia-Imran who always stand by me not only during my happy moments but also during my tough moments, help me to recollect my enthusiasm about life and work. Thanks to them for showing enormous patience with me when I am complaining for countless hours. I would also like to thank Bushra and Aamer with whom I forged a special bond during my six-month internship. Since then they have been a constant source of support and strength.

I would like to thank all my friends at the University of Virginia without whom life would have been very difficult. Special thanks to Tanima, who made my stay at Charlottesville much easier. She was always there for me whenever I needed her, no matter how busy her schedule was. Whenever life seemed unruly, all the friendly banters exchanged with Enamul, Nirjon, Munir, Yan alleviated some of the pressures of graduate life and made me ready again to face more challenges. Special thanks to Munir for bringing lots of fun and joy in the tedious graduate life. He has been a continuous source of support through out my stay here. Even the newcomers Yamina, Anindya and Jisa whom I got to know for such a short time made my life so much more fun. It is because of them, my last days of graduate life were more tolerable. When I was practically homeless during the end of my graduate life, Yamina was kind enough to let me stay with her. I would also like to thank Tanveer, Samia, Anwica, Redwan, Sadi and Samee for their support. Gratitude to Kirti for his wise suggestions and assistances through out my stay at Charlottesville. Lastly, thanks to VB who helped

me stay sane through these difficult years in the lab. All the technical discussions as well as the lightweight conversations with him made the lab a brighter place to work. I greatly value the friendship of all these people and I deeply appreciate their belief in me. Their support and care helped me overcome setbacks and stay focused on my graduate study.

Finally, my family without whom I cannot think of my existence. I am so much lucky to have all my caring siblings Rashed, Arifa, Asma, Radwan and Saima. And of course thanks to my most adorable angles, my nephews and nieces Sameen, Safwan, Sahrish, Sara, Yaveen, Saihan and Sajid; even though they do not realize how they impacted my life. Every year just the thought of spending some quality time with them used to motivate me to work at a higher pace. Limitless gratitude and respect for my wonderful brother-in-law Rifat Bhai who is more than a brother to me. Without his moral support I would not be here. Lastly, my parents who are the source of all my motivation and optimism. At times I lose all my hopes but my parents never let me feel down. They make me realize what is life about and the fact that I should always keep going.

Last but not least, thanks to God for making my life bountiful and giving me the strength to survive the tests.

Contents

Contents **vi**
List of Tables . . . viii
List of Figures . . . ix

1 Introduction **1**

2 Overview of NBTI and Related Work **6**
2.1 Related Work . . . 9
2.1.1 Stress reduction techniques for NBTI: . . . 9
2.1.2 Recovery enhancement techniques for NBTI: . . . 11
2.1.3 Process Variation: . . . 12
2.1.4 Interaction of NBTI and Process Variation: . . . 13

3 Modeling NBTI at the Architecture-level **15**
3.1 Challenges Posed by Existing NBTI models for Architecture Simulation 16
3.2 Adapting the NBTI model for Circuit and Architecture Simulation . . 19
3.3 Summary . . . 22

4 Enhancing NBTI Recovery in SRAM Arrays through Recovery Boosting **23**
4.1 Basics of Recovery Boosting . . . 25
4.1.1 Fine-Grained Recovery Boosting . . . 30
4.1.2 Coarse-Grained Recovery Boosting . . . 32
4.1.3 Other Issues . . . 33
4.2 Designing Microarchitectural Structures that Support Recovery Boosting 35
4.2.1 Physical Register File . . . 36
4.2.2 Issue Queue . . . 38
4.2.3 Circuit-Level Simulation Results . . . 42
4.3 Experimental Methodology for the Architecture Level Analysis 50
4.4 Architecture-Level Simulation Results . . . 51

4.4.1 Physical Register File Results 53
4.4.2 Issue Queue Results 56
4.5 Summary 61

5 Mitigating the Impact of NBTI on Processor Functional Units 63
5.1 Approaches to NBTI Mitgation at the Circuit and Microarchitecture Levels 65
5.1.1 Circuit Level Techniques 66
5.1.2 Microarchitecture Level Techniques 69
5.2 Experimental Setup 73
5.3 Results 75
5.3.1 Circuit-Level Optimization 75
5.3.2 Microarchitecture-Level Optimization 78
5.3.3 Multi-Level Optimization 81
5.4 Summary 82

6 Modeling and Analyzing NBTI in the Presence of PV 84
6.1 Overview of NBTI and PV 86
6.2 An analytical model for NBTI and PV 88
6.2.1 Capturing the impact of Workload Variation, Temporal Variation, and PV 89
6.3 Experimental Setup 91
6.4 Results 92
6.4.1 RF Results 93
6.4.2 KSA Results 96
6.4.3 Implications of the Results 98
6.5 Summary 100

7 Conclusions and Future Work 101

Bibliography 106

List of Tables

4.1 Modified SRAM cell operation 28
4.2 Control signal truth-table for a register 37

List of Figures

2.1 NBTI Stress Phenomena. 7
2.2 NBTI Recovery Phenomena. 8

3.1 The existing circuit-level NBTI model breaks down with discretization. 17
3.2 The existing circuit-model NBTI model does not capture multiple sequences of stress and recovery events. 19

4.1 Conventional 6T SRAM cell 26
4.2 SRAM Cell Design for Recovery Boosting 27
4.3 PMOS gate voltages of an SRAM bitcell due to recovery boosting using the modified SRAM cell shown in Figure 4.2 (V_{dd}=0.9V, T=90C) 28
4.4 Modified SRAM cell with connection to the V_{dd} rail of an adjoining row 30
4.5 SRAM Array for Fine-Grained Recovery Boosting (N entries, M-wide) 31
4.6 PMOS gate voltages of an SRAM bitcell due to recovery boosting and power gating (V_{dd}=0.9V, T=90C) 33
4.7 Register states. The candidate states for recovery boosting are shown in dashed circles. 37
4.8 Control Logic for generating Control Signal CR (UM_x = 'unmapped' bit for register x and CM_x = 'completed' bit for register x) 37
4.9 An issue queue entry. 39
4.10 CAM structure of an issue queue entry (IW = issue width) 40
4.11 Modified CAM Structure (IW = issue width). MBC is the Modified Bit-Cell for recovery boosting. 41
4.12 Write delay of the modified bitcell. Node0 and Node1 are the node voltages of the bitcell (V_{dd}=0.9V, T=90C). 42
4.13 Transition between recovery and normal modes. Node0 and Node1 are the node voltages of the bitcell (V_{dd}=0.9V, T=90C). 44

4.14 Area of the register file and the issue queue for designs that use conventional 6T cells and cells modified to support recovery boosting. . . 46
4.15 Power consumption of a single register entry (V_{dd}=0.9V, T=90C). . . 48
4.16 Power consumption of a single issue queue entry (V_{dd}=0.9V, T=90C). 49
4.17 V_t and SNM degradation for the RF for the *Baseline*, *Recovery Boosting* and *Balancing* configurations (V_{dd}=0.9V, T=90C). 52
4.18 Breakdown of time spent by the registers in different states. The lowest part of each stacked bar is the *Unmapped* state. 54
4.19 Improvement in the Static Noise Margin for the RF over the *Baseline* processor configuration (V_{dd}=0.9V, T=90C). 55
4.20 V_t and SNM degradation for the IQ for the *Baseline*, *Recovery Boosting* and *Balancing* configurations (V_{dd}=0.9V, T=90C). 56
4.21 Breakdown of time spent by the IQ entries in the *Valid* and *Invalid* states.. 59
4.22 Improvement in the Static Noise Margin for the IQ over the baseline processor configuration (V_{dd}=0.9V, T=90C). 60

5.1 Partitioned Kogge-Stone Adder Design to Support NBTI Recovery. . 68
5.2 Utilization of the Integer ALUs in a 4-wide issue processor core using the PS policy. 71
5.3 Guardband reduction for the FUs with 2, 4 and 8 segments. 75
5.4 Percentage increase in area and delay for the FUs with 2, 4 and 8 segments wrt. the unpartitioned FU design. 77
5.5 Impact of the instruction scheduling policies. 79
5.6 Guardband reduction using FUs with 2 segments and the PR scheduling policy. 81

6.1 Different sources of V_t variation in PMOS devices. 87
6.2 V_t distributions of the RF due to RDF, temporal, workload and combined variation for the mcf benchmark. 94
6.3 Number of bits experiencing SNM below the minimum allowed value in a RF due to temporal, workload and the combined variation for the different benchmarks. 95
6.4 V_t distributions of the KSA due to RDF, temporal, workload and combined variation for the mcf benchmark. 97
6.5 Percentage increase in delay in a KSA due to temporal, workload and the combined variation for different benchmarks. 99

Chapter 1

Introduction

The key drivers in processor design are high performance and energy efficiency. Moore's Law has been key in enabling the design of processors with ever increasing performance, for example, by facilitating multicore design. Additionally, the processor also has to operate reliably and continue to do so over a long period of time, what is usually referred to as the *Service Life* of the processor. The service life target of a high performance processor is typically 7-10 years. While technology scaling has benefitted high performance, the scaling has a dark side too: degradation in the reliability of silicon devices. Processors have become highly susceptible to a variety of reliability problems in silicon, such as particle induced soft errors and hard errors. A key emerging hard error problem facing the microprocessor industry today is Negative Bias Temperature Instability (NBTI), which affects the lifetime of PMOS transistors. Providing protection for the processor against declining silicon reliability in order to meet service life guarantees can entail significant performance, power and area overheads. *The overall goals of this dissertation are to improve our understanding of NBTI and to develop mit-*

igation techniques that impose little performance, power, and area overheads to meet the service life target to combat this reliability problem.

NBTI occurs when a negative bias is applied at the gate of a PMOS transistor, which causes an increase in the threshold voltage of the device. NBTI affects both the cycle time and the stability of storage structures within the processor. In terms of its impact on microprocessor circuits, the increase in the threshold voltage degrades the speed of the transistors and therefore degrades the speed and the noise margin of the circuit in which they are used, eventually causing the circuit to violate timing constraints. Such timing violations due to NBTI will cause the circuit to behave incorrectly and cause the processor itself to fail. NBTI is typically addressed via guardbanding. Guardbanding accounts for the degradation in cycle time and the stability of the storage structures over the lifetime by reducing the operating frequency and increasing the minimum voltage of the storage elements (V_{min}). Typically, 20% of the cycle time is reserved as a guardband for the logic structures. Similarly, V_{min} is increased by 10% as a guardband to handle 10% increase in threshold voltage (V_t) for the storage structures. However, reducing the frequency and increasing V_{min} have a detrimental impact on performance and power respectively and therefore it is desirable to reduce the guardband via the use of NBTI mitigation techniques.

NBTI mitigation techniques can be implemented at different levels of the system stack (device, circuit, microarchitecture). There are pros and cons to providing the protection at each of these levels. A hierarchical approach that handles reliability at various levels across the hierarchy can enable the designer to get the best of each of the worlds by being able to optimally address issues at the level of the system stack where they are most naturally handled well. The main contribution of this disserta-

tion is to develop and characterize NBTI mitigation techniques at different levels and quantify their impact on metrics, such as, guardband reduction, area, delay, and application performance. While NBTI itself is well characterized, NBTI is just one of the physical phenomena that affect the reliability of the processor. It is also important to understand NBTI in the context of other key physical phenomena in order to arrive at optimal design decisions. This dissertation lays the foundation for such an understanding of NBTI by examining the interaction of this reliability phenomenon with Process Variation (PV). PV is the variation in the transistor attributes (length, width, oxide thickness) caused during the fabrication of the integrated circuits and manifests itself as threshold voltage variation, which results in variability in circuit performance and power. It is important to realize how NBTI is affected by PV in order to come up with energy-efficient mitigation techniques. Therefore, we need a deep and accurate understanding of how NBTI interacts with this physical phenomenon.

To summarize, this dissertation develops NBTI mitigation techniques for the microarchitectural structures in a microprocessor and creates the foundation for understanding NBTI in the context of other physical phenomena that affect the processor. This dissertation consists of tasks that involve modeling and optimization related to NBTI. The modeling tasks involve developing models for NBTI that are usable at the architecture level and capture the interaction between NBTI and PV described previously. The optimization tasks involve developing NBTI mitigation techniques for both the logic and memory structures within the processor and explore multiple levels in the design stack.

The specific contributions of this dissertation are:

1. **Modeling NBTI at the Architecture-level:** There have been several efforts in developing analytical models for NBTI at the circuit-level. However, these models are suitable only for analyzing NBTI effects over a very short time span and are not readily usable for architecture simulations. Since our research involves exploring NBTI mitigation at multiple levels of the design hierarchy, our first contribution is an analytical NBTI model that is suitable for microarchitecture and architecture-level evaluations. This work has been published in ISQED 2011 [1].

2. **NBTI Mitigation Techniques for Memory Structures:** With the architecture-level NBTI model, our next contribution is mitigation techniques that can combat NBTI to meet the service life guarantee with minimal performance, power, and area overheads. Modern processor cores are composed of several critical SRAM-based structures, such as the register file and the issue queue. We develop mitigation techniques for the memory structures in the processor core to maximize their lifetimes. We develop and evaluate mitigation techniques at both the circuit and microarchitecture levels. This work is published has been ISVLSI 2010 [2] and in the IEEE Transactions on VLSI 2011 [3].

3. **NBTI Mitigation Techniques for Functional Units:** Our next contribution is mitigation techniques that can combat NBTI in the logic structures in the processor with minimal performance, power, and area overheads. We present a quantitative analysis of NBTI recovery techniques at the circuit and microarchitecture levels for the functional units (FUs) in the cores of a high-performance

multicore processor that can facilitate reducing the device-level guardband requirements. This work has been published in GLSVLSI 2010 [4].

4. **Modeling and Analyzing NBTI in the Presence of PV:** Both NBTI and PV affect the threshold voltage of the devices and these two problems should not be addressed in isolation. Therefore, an analytical model is required which captures the impact of both NBTI and PV in a coherent way and which is suitable for use in architecture level analyses. Leveraging the prior research on NBTI and PV modeling from the circuits community as the starting point, we develop a model that captures the interaction between these two reliability phenomena. This work has been published in ISQED 2011 [1].

The organization of the rest of this dissertation is as follows. The next chapter provides a brief overview of NBTI and discusses the related work. Chapter 3 presents the NBTI model for architecture simulations. NBTI mitigation techniques for memory and logic structures are presented in the Chapter 4 and 5 respectively. Chapter 6 discusses the analysis of NBTI in the presence of PV. Lastly, Chapter 7 conludes this dissertation.

Chapter 2

Overview of NBTI and Related Work

Negative Bias Temperature Instability (NBTI) is a growing concern for CMOS technology and affects the lifetime of PMOS transistors. NBTI increases the threshold voltage of PMOS devices, which in turn degrades the speed of circuits. To better understand how to develop optimization techniques to combat NBTI, we first need to look at how NBTI occurs and how it affects a circuit. During the fabrication process, hydrogen atoms form a $Si - H$ bond along the Si/SiO_2 interface. Water molecules are often present during the contact and via formation of the IC fabrication process, which can increase the effects of NBTI by donating hydrogen to the silicon oxide interface. Although many engineers feel that this is a fabrication problem, until it can be resolved, it is important to be able to design around the issue.

When a logic input of “0” is applied to the gate of a PMOS transistor (V_{gs} = -V_{dd}), NBTI occurs due to the generation of interface traps at the Si/SiO_2 interface. When silicon is oxidized, most of the Si atoms at the surface of the wafer bond with oxygen while a few atoms bond with hydrogen. When the transistor is being stressed, an

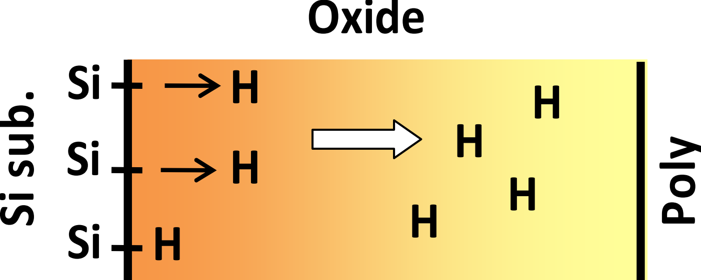


Figure 2.1: NBTI Stress Phenomena.

electric field is placed across the oxide layer. When a negative bias (i.e., a logic input of "0") is applied at the gate of a PMOS transistor, the relatively weak $Si - H$ bonds get disassociated as shown in Figure 2.1. These hydrogen atoms enter the oxide layer. The longer a hydrogen atom is in the electric field, the deeper into the oxide layer it will penetrate. Eventually the hydrogen atoms can reach the oxide/poly interface and cluster there. By introducing positively charged ions/interface traps (hydrogen atoms) into the oxide layer, the part of the field that inverts the transistor channel is weakened. Over time, this reduces the number of hydrogen atoms that can break free. These interface traps cause the threshold voltage (V_t) of the PMOS transistor to increase, which in turn degrades the speed of the device and the noise margin of the circuit. This is known as the Stress phase for the PMOS. The increase in V_t due to stress is given by the equation [5] :

$$\Delta V_{ts} = (\frac{qt_{ox}}{e_{ox}})^{\frac{3}{2}}.K_1.\sqrt{C_{ox}(V_{gs} - V_t)}.e^{\frac{-E_a}{4kT} + \frac{2(V_{GS} - V_t)}{t_{ox}E_{01}}}.T_0^{-0.25}.t_{stress}^{0.25} \tag{2.1}$$

where t_{stress} is the time under stress, t_{ox} is the oxide thickness and C_{ox} is the gate capacitance per unit area. K_1, E_a, T_0, E_{01} and k are constants equal to 7.5 $C^{-0.5}nm^{-2.5}$, 0.49 eV, 10^{-8} s/nm^2, $0.08V/nm$ and $8.6174 \ 10^{-5}eV/K$ respectively.

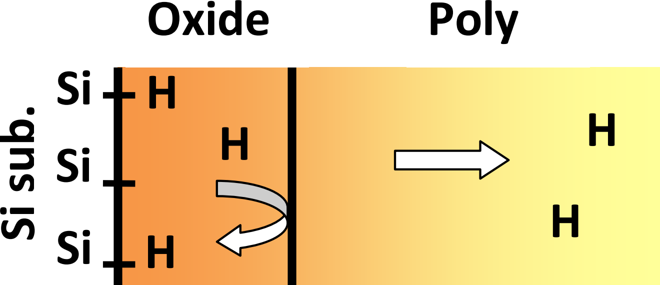


Figure 2.2: NBTI Recovery Phenomena.

When a logic input of "1" is applied to the gate (V_{gs} = 0), the H atoms released in the stress process can anneal the broken bonds, or the H atoms may diffuse (or drift) away from the interface toward the oxide/poly interface. This is illustrated in Figure 2.2. Therefore, this process helps in eliminating some of the traps. This is known as the Recovery phase. During the recovery phase a positive electric field is placed accross the oxide layer. The field removes the inverted channel and hydrogen is free to reconnect with the available silicon by annealing [6, 7, 8, 9, 10]. This process, much like stressing, can be exacerbated by increased temperature. As mentioned earlier, the hydrogen can move all the way to the Si/Poly interface making it possible that not all of the hydrogen can return to the Si/SiO_2 interface. This effect creates a state of hysteresis, leaving behind a residual ΔV_t after annealing. The effects on transistor instability around the Si/SiO_2 interface have been studied for many years and were recognized as early as the 1970's [11, 12]. Most research on NBTI was developed by device and reliability physics groups. Alam and Mahapatra developed one of the first comprehensive models to explain PMOS NBTI using a standard reaction-diffusion model [13]. Later, Wang et al. presented a compact model that could be used for circuit simulation [5]. The final increase of V_t after considering both the stress and recovery

phases is [5]:

$$\Delta V_t = \Delta V_{ts}.(1 - \frac{2\xi_1 t_{ox} + \sqrt{\xi_2 e^{\frac{-E_a}{kT}} T_0 t_{rec}}}{(1+\delta)t_{ox} + \sqrt{e^{\frac{-E_a}{kT}}(t_{stress} + t_{rec})}}) \quad (2.2)$$

where t_{rec} is the time under recovery, ξ_2, ξ_1 and δ are constants equal to 0.5, 0.9 and 0.5 respectively. From the equations, it is observed that NBTI is exponentially dependant on the difference between V_{gs} and V_t, temperature and stress/recovery time. A lower V_{gs}, temperature and stress time improves the lifetime whereas a higher V_t and recovery time improves it. A lower V_{gs} (which eventually lowers the temperature) will have an impact on performance.

2.1 Related Work

Several recent studies have proposed techniques for mitigating NBTI to improve processor lifetime. There are two basic approaches to mitigating NBTI: (i) reduce the stress on the PMOS transistors; (ii) enhance the recovery process. Stress reduction techniques aim to reduce the aging rate by controlling V_{dd}, V_t, and temperature, whereas recovery enhancement techniques aim to increase the recovery time for the PMOS devices.

2.1.1 Stress reduction techniques for NBTI:

Srinivasan et al. [14] propose the use of Dynamic Reliability Management (DRM) to stay within the reliability budget. They describe an architecture-level model RAMP,

that can dynamically track lifetime reliability, responding to changes in application behavior. RAMP is based on device models for different wear-out mechanisms including NBTI. Using RAMP, they show that dynamic voltage scaling is an effective response technique for DRM, and that dynamic thermal management neither subsumes nor is subsumed by DRM. Tiwari and Torrellas propose a technique called "Facelift" [15] to hide the effects of aging through temperature-based job-scheduling to individual cores of a multicore processor. Facelift hides the effects of aging by steering high-temperature jobs to the fast cores and low-temperature jobs to the slow cores. The chip appears to age less by keeping the slow cores cooler. Also, Facelift slows down aging by making chip-wide changes to V_{dd} or V_t at key times to balance the impact of the changes on the aging rate and on the critical path delays. Finally, Facelift configures a chip for a short service life by shifting performance from the unused lifetime portion to the used one. Basoglu et al. [16] propose a low-cost NBTI-Aware DVFS framework to reduce energy consumption and increase the lifetime of the processor. They utilize real-time degradation data and employ a technique that alleviate this problem through core-level DVFS control and OS-controlled workload mapping based on core status. If the OS is informed of the degradation status of each core, it could map threads so that sturdy cores (those with low V_t) work more than the weaker cores (those with high V_t). This equalizes core lifetimes, and thereby extends overall processor life. All these works aim to reduce the stress on the devices but make no attempt to leverage the recovery process to combat NBTI and therefore their effectiveness is limited. Our goal is to leverage the recovery phenomenon to extend the lifetime the processor. In general, the use of stress reduction techniques is orthogonal to recovery enhancement.

2.1.2 Recovery enhancement techniques for NBTI:

Abella et al. [17] propose to feed specific bit patterns into the devices to increase the recovery time for PMOS transistors in logic structures (e.g., adders) during idle periods. They also propose to balance the degradation of the PMOS devices in SRAM-based memory structures by storing appropriate data value into the SRAM cells when they hold invalid data. Kumar et al. [18] propose a similar technique to periodically flip the contents of SRAM cells to balance the wear on the PMOS transistors. The cost of such a technique comes from the extra XNOR gates required to invert/deinvert data with the invert bit (global signal indicating the current mode), which has an impact in cycle time. However, invertion is not a suitable solution for combinational blocks because inverted and non-inverted inputs may stress the same PMOS transistors. Shin et al. [19] propose a recovery enhancement technique for caches where SRAM cells are proactively put into the recovery mode via the use of a spare memory array. They reverse bias ($V_{gs} = V_{dd}$) the PMOS devices to put them into a deep recovery state. When an array is put into the recovery mode, the PMOS device in one of the inverters of each cell is put into the recovery mode followed by those in the other inverters and this recurring pattern is continued throughout the recovery period for the array. Gunadi et al. [20] proposes Colt, which balances the utilization of devices in a processor by equalizing the duty cycle ratio of the internal nodes of circuits and the usage frequency of devices. It relies on alternating true- and complement-mode operations to equalize the duty cycle ratio of signals. These approaches mitigate NBTI by balancing the degradation of the PMOS devices within the cell. They aim to achieve a 50% lifetime degradation for the PMOS devices. Our approach does not limit itself to achieving

a balance in the degradation of the PMOS devices. Our approach attempts to extend the recovery process of all the PMOS devices beyond 50% to extend the lifetime of the processor. For the SRAM structures, we propose a new SRAM cell design thats facilitates simultaneous recovery process for both the PMOS devices in the memory cell. Similarly, power-gating is utilized to introduce simultaneous recovery process for all the PMOS devices in the functional units along with new instruction scheduling techniques that extend their idle time.

2.1.3 Process Variation:

There are several studies whose goal is to design PV tolerant systems and do not look at the impact of NBTI in the presence of PV. Tiwari et al. [21] present an architectural framework that applies cycle-time stealing to the pipeline to tolerate PV. Chun et al. [22] propose a scheme of adjusting the clock speed of a processor based on the instruction-level parallelism of the program phases to achieve overall performance improvement to address PV. Sarangi et al. [23] propose a framework called EVAL (Environment for Variation-Afflicted Logic) to understand how processors can tolerate and mitigate variation-induced errors. They present a technique to maximize processor performance and minimize power in the presence of variation-induced timing errors by adapting the processor frequency, multiple voltages, and two processor structures. Even though these techniques provide effective solutions to tackle PV, the effectiveness of these techniques might reduce in the presence of NBTI.

2.1.4 Interaction of NBTI and Process Variation:

All the aforementioned studies concentrate on NBTI or PV without considering the interaction between them. There have been several studies on the combined effect of NBTI and PV. There are different sources of variation inherent in NBTI and PV that affect the PMOS threshold voltage. One source of variation in the threshold voltage due to NBTI is workload variation which is caused by executing different workloads on the processor. This variation is due to changing patterns of utilization of the microarchitectural structures and changes in the bit patterns within the structures. Another factor lies in the silicon process, known as the Random Charge Fluctuation (RCF), which causes a temporal variation in threshold voltage on top of the workload variation. Kang et al [24] propose a compact circuit-level V_t model that captures the impact of temporal NBTI variations in the presence of PV and shows how temporal V_t variations can affect the lifetime and performance of different circuit topologies. Basu et al. [25] present a methodology to develop PV and NBTI tolerant robust standard cells which can be used in timing critical sections of the circuits. They model the combined effect of PV and NBTI on intrinsic gate delay using a reduced dimension modeling technique to optimize the standard cells with a target lifetime of 10 years. Lu et al. [26] design a comprehensive IC reliability analysis framework with respect to NBTI and PV. This work is capable of characterizing the overall circuit lifetime reliability, as well as efficiently quantifying the vulnerabilities of individual circuit elements. This analysis framework has been integrated into an iterative design flow for circuit lifetime reliability analysis and optimization. Finally, Fu et al. [27] propose NBTI and PV tolerant micrarchitecture design techniques to improve processor lifetime. They show

that just combining PV mitigation techniques and NBTI recovery mechanisms lacks the capability of exploiting the opportunity to optimize their interaction. They propose microarchitecture designs that exploit the positive interplay between PV and NBTI that improve the trade-offs among different metrics. While all these prior works study some combinations of NBTI (static, or temporal variation, or workload variation) with PV, no prior work has holistically analyzed the combined effect of temporal and workload variations on top of static NBTI with process variation.

Chapter 3

Modeling NBTI at the Architecture-level

Analytical models provide guidance for assessing the reliability impact of a design decision. There have been several efforts in developing analytical models for NBTI at the circuit-level [5, 24]. However, these models are suitable only for analyzing NBTI effects over a very short time span and are not readily usable for architecture simulations. Architects, on the other hand, study microprocessor reliability by executing different program benchmarks and extrapolate the collected statistics over a much longer timescale (typically, 7-10 years). Throughout the benchmark execution, utilizations of the microarchitectural structures vary. Also, the interactions among the structures, the inputs to each structure, and bits stored within them change over the course of execution of a benchmark. An analytical model for NBTI should be able to factor-in these "variations" to be usable in architecture simulations to gain correct and holistic insight into these inter-related reliability problems in silicon. In this work, we aim to leverage

the prior research on NBTI modeling from the circuits community to develop a model that is usable at the architecture-level.

3.1 Challenges Posed by Existing NBTI models for Architecture Simulation

Existing circuit-level models cannot be directly used for architecture-level simulations. The existing NBTI models capture an analog process and assume continuous stress on the PMOS devices in a circuit. They do not capture scenarios where there are multiple sequences of varying stress/recovery times, which is the case when real workloads run on the processor. Typically architecture-level simulations capture the real workload behavior on the processor in a cycle-by-cycle basis and attempt to use the NBTI models in a quantized way. As a result of this quantized usage, existing NBTI models cannot be used directly in architecture-level simulations. To explain the problems with the existing NBTI models in details, we choose the model presented in Chapter 2 which is widely used in the circuit literature. However, the problems we discuss apply to the use of other circuit-level NBTI models too. This model has two limitations when used in an architecture simulation:

i) The problem of discretization of the continuous model:

Let us consider a hypothetical scenario where a PMOS device is stressed for t_1, t_2 and t_3 units of time and the degradations in threshold voltage due to these stress events are $V_t(t_1)$, $V_t(t_2)$ and $V_t(t_3)$ respectively. If the device is stressed for t_1 time units followed by t_2, and t_3 is equal to $(t_1 + t_2)$, then we expect $V_t(t_3) = V_t(t_1) + V_t(t_2)$. Figure 3.1

describes this behavior. The x-axis is the time and y-axis is the threshold voltage. In the figure, once stress time of t_1 is applied initially, the model computes the threshold voltage to reach A. At this point, if stress time of t_2 is applied, instead of reaching a value of B, the model computes it to be C, whereas the value should be B after (t_1 + t_2) units of time. The reason for this problem is because ΔV_{ts} has an exponential relationship with the stress time and we know that $(t_1 + t_2)^x \neq (t_1{}^x + t_2{}^x)$. This model is meant to represent the continuous process of stress and recovery phenomena and it breaks down whenever we discretize this continuous process. Therefore, using the model, we do not achieve the expected value of $V_t(t_3)$ which should be equal to [$V_t(t_1)$ + $V_t(t_2)$]. However, this limitation will exist in the circuit-level simulations as well if we want to use the model in a discretized way.

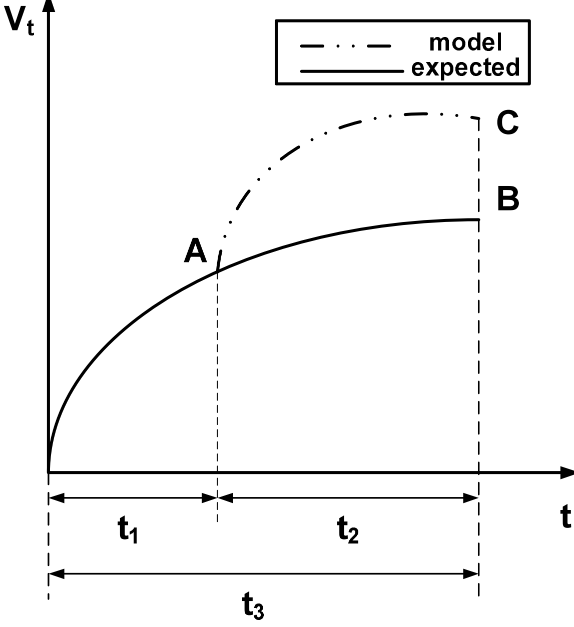


Figure 3.1: The existing circuit-level NBTI model breaks down with discretization.

ii) The model is not usable with multiple stress/recovery events:

To understand this problem, let us consider another hypothetical example where a PMOS device gets stressed followed by a single sequence of stress and recovery events. Figure 3.2 illustrates this situation. From the figure, with the first set of stress and recovery events, the V_t reaches a value of B and A respectively. At this point of time, with a subsequent stress event, the threshold voltage degradation pattern should follow the pattern of the first stress event starting from point A, since both instants have a V_t value of A. Therefore, after applying the second stress for t_1 time units, the final V_t value should be, B whereas the model provides a different value of C. Thus, the model is not able to capture multiple sequences of stress/recovery events properly. The reason for this problem is because the model uses the instantaneous V_t value as the history of degradation. From the stress phase equation, we can see that the value of ΔV_{ts} depends on the value of V_t. For a fixed stress time, the model would produce different ΔV_{ts} values for different V_t values. In this hypothetical scenario, the first stress event uses the nominal V_t value and the second stress event uses the degraded V_t value. Therefore, the two stress events of t_1 time unit starting from point A produce two different values.

Both these properties need to be modeled correctly for an architecture level analysis of NBTI degradation. Since the architecture simulations update the V_t values of the devices due to NBTI at different points of time throughout the execution of a workload, the lack of the ability to discretize in the model results in incorrect estimation of the V_t degradation. Also, real workloads show varying patterns of stress/recovery for different structures within the processor. Hence, the assumption of continuous stress on the PMOS devices does not capture the realistic scenario. The next section discusses how to modify the model to address these two limitations.

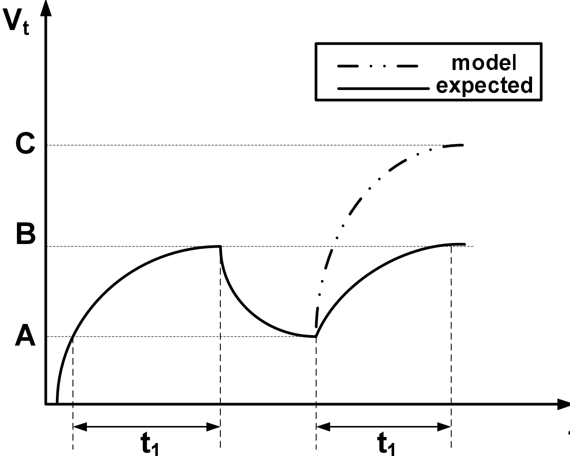


Figure 3.2: The existing circuit-model NBTI model does not capture multiple sequences of stress and recovery events.

3.2 Adapting the NBTI model for Circuit and Architecture Simulation

From the original model described in Chapter 2, we know that ΔV_{ts} is a function of voltage, temperature, instantaneous V_t, and t_{stress}, whereas, ΔV_t after recovery is a function of ΔV_{ts}, t_{stress} and t_{rec}. We can rewrite the original model as:

$$\Delta V_{ts} = f_{stress}(V, T, V_t).t_{stress}^{0.25} \tag{3.1}$$

$$\Delta V_t = \Delta V_{ts}.f_{rec}(t_{stress}, t_{rec}) \tag{3.2}$$

In this model, the V_t value represents the history of stress and recovery events (the total degradation). The main idea behind our proposed model is to represent the

degradation history in terms of the equivalent stress time experienced by the PMOS device. Since the existing model is applicable for a single stress/recovery event, we transform the previous multiple stress/recovery events into a single stress event and use that equivalent stress time with the new stress/recovery event. Note that in this case, we always use the nominal V_t value. After the aforementioned modification, we get the following model:

$$\Delta V_{ts} = f_{stress}(V,T).(t_{equi-stress} + t_{stress})^{0.25} \tag{3.3}$$

$$\Delta V_t = \Delta V_{ts}.f_{rec}[(t_{equi-stress} + t_{stress}), t_{rec}] \tag{3.4}$$

where $t_{equi-stress}$ is the equivalent stress time resulting from previous stress and recovery events and $t_{equi-stress} = 0$ at t = 0. Now we discuss how to calculate the value of $t_{equi-stress}$.

From equation 3.1, we get ΔV_{ts}, which is the increase in threshold voltage due to the stress time t_{stress}. If we reorganize equation 3.1, we find the following:

$$t_{stress} = [\frac{\Delta V_{ts}}{f_{stress}(V,T,V_t)}]^4 \tag{3.5}$$

This equation expresses the stress time experienced due to the previous stress/recovery events when the increase in threshold voltage is known. Using equation 3.5 with the nominal V_t and the given ΔV_t, which is a result of previous stress and recovery events,

we can calculate the $t_{equi-stress}$:

$$t_{equi-stress} = [\frac{\Delta V_t}{f_{stress}(V,T)}]^4 \tag{3.6}$$

After combining equations 3.3, 3.4 and 3.6, we get the following final model:

$$\Delta V_{ts} = f_{stress}(V,T).\{[\frac{\Delta V_t}{f_{stress}(V,T)}]^4 + t_{stress}\}^{0.25} \tag{3.7}$$

$$\Delta V_{tf} = \Delta V_{ts}.f_{rec}(\{[\frac{\Delta V_t}{f_{stress}(V,T)}]^4 + t_{stress}\}, t_{rec}) \tag{3.8}$$

where ΔV_{tf} is the final threshold voltage degradation and ΔV_t is the threshold voltage degradation due to previous stress and recovery events and ΔV_t = 0 at t = 0. Equations 3.7 and 3.8 together completes the model and they represent the stress and recovery phenomena respectively. This model can be readily used with any architectural simulator to characterize the NBTI degradation of a PMOS device.

Note that, this model captures the effect of voltage and temperature variation as well. Since the equivalent stress time is also a function of voltage and temperature, whenever there is a variation in either voltage or temperature or both, the equivalent stress time gets calculated under the new stress condition.

3.3 Summary

In this chapter, we develop an analytical model that captures NBTI for use in circuit and architecture simulations. Existing models cannot be directly used for architecture-level simulations. This is because these models assume continuous stress on the PMOS devices in a circuit lacking the additive property and do not capture scenarios where there are multiple sequences of varying stress/recovery times, which is the case when real workloads run on the processor. To address these problems, our proposed model represents the degradation history in terms of the equivalent stress time experienced by the PMOS device instead of the V_t value used by the existing models. Using this architecture-level NBTI model, we develop NBTI mitigation techniques for the microarchitectural structures in a microprocessor and create the foundation for understanding NBTI in the context of PV, which we discuss in the following chapters.

This chapter covers work published in ISQED 2011 [1].

Chapter 4

Enhancing NBTI Recovery in SRAM Arrays through Recovery Boosting

Memory arrays that use Static Random Access Memory (SRAM) cells are especially susceptible to NBTI. SRAM cells consist of cross-coupled inverters that contain PMOS devices. Since each memory cell stores either a '0' or a '1' at all times, one of the PMOS devices in each cell always has a logic input of '0'. Since modern processor cores are composed of several critical SRAM-based structures, such as the register file and the issue queue, it is important to mitigate the impact of NBTI on these structures to maximize their lifetimes. Previous work on applying recovery techniques to SRAM structures aim to balance the degradation of the two PMOS devices in a memory cell by attempting to keep the inputs to each device at a logic input of '0' exactly 50% of the time [17, 18, 19]. However, one of the devices is always in the negative bias condition at any given time. In this chapter, we propose a novel technique called *Recovery Boosting* that allows *both* PMOS devices in the memory cell to be put into the recovery

mode. The basic idea is to raise the ground voltage and the bitlines to V_{dd} when the cell does not contain valid data. In this chapter, we describe how SRAM cells can be modified to support recovery boosting and discuss several circuit and microarchitecture level design considerations when using such cells to build SRAM arrays. We present the circuit-level design of two large SRAM arrays in a 4-wide issue processor core - the physical register file and the issue queue - that use the modified cells to provide recovery boosting. We verify the functionality of these designs and quantify their area and power consumption through SPICE-level simulation using the Cadence Virtuoso Spectre circuit simulator [28] for the 32nm process technology. We show that the modified SRAM structures impose only a 3-4% area overhead over the baseline non-recovery boost designs and that their maximum power consumption is less than 2% over the baseline. We then evaluate the performance and reliability of area-neutral designs of these modified structures at the architecture-level via execution-driven simulation using the M5 simulator [29] and the SPEC CPU2000 benchmark suite [30] in nominal operating condition. We show that recovery boosting provides a 56% improvement in the static noise margin of the register file cells and a 48% improvement for the issue queue across the benchmark suite while having a negligible impact on performance.

The organization of the rest of this chapter is as follows. The next section discusses the recovery boosting technique. The circuit-level design and evaluation of the register file and issue queue are given in Section 4.2. The experimental methodology used for the architecture-level evaluation is given in Section 4.3 and the corresponding results in Section 4.4. Section 4.5 concludes this chapter.

4.1 Basics of Recovery Boosting

Before we discuss recovery boosting, we first review the design and operation of a conventional 6-transistor (6T) SRAM cell. The design of the 6T cell is given in Figure 4.1. The cell is composed of a wordline (WL), a pair of bitlines (BL, BLB), two cross-coupled inverters (I_0, I_1), and two access transistors (N_0, N_1). The cross-coupled inverters store one bit of data. There are three basic operations that one can perform on this SRAM cell: read, write and hold. To read and write data, the cell is selected by raising WL to high. This activates the access transistors and connects the inverters in the cell to the bitlines. During a read operation, both bitlines are first precharged high. Based on the data stored in the cell, one of the bitlines is discharged. This change is detected by a sense amplifier (which is not part of the cell) to determine the value stored in the cell. During a write operation, one of the bitlines is raised high and the other is lowered depending on the value to be written to the cell. When the cell is not selected (WL = 0) for read or write, it is expected to hold the data stored in it and is said to operate in the hold mode.

Since the SRAM cell has cross-coupled inverters, each inverter charges the gate of the PMOS or NMOS device of the other inverter. Therefore, at any given time, one PMOS device will always be in the stress mode. The goal of recovery enhancement is to put the PMOS devices into the recovery mode by feeding input values to the cell that will transition them into that mode. However, due to the cross-coupled nature of the inverters, only one of the PMOS devices can be put into the recovery mode. Therefore, previously proposed recovery enhancement techniques attempt to balance the wearout of the two PMOS devices by putting each PMOS into the recovery mode 50% of the

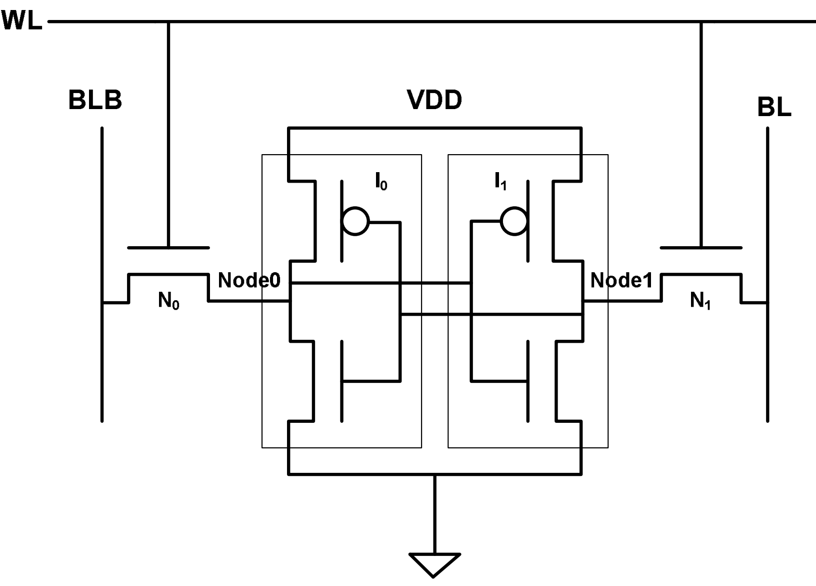


Figure 4.1: Conventional 6T SRAM cell

time by feeding appropriate input values [18, 17, 19]. We propose a 6T SRAM cell design shown in Figure 4.2 which is capable of normal operations (read, write, and hold) as well as providing an NBTI recovery mode (when the cell does not contain valid data) that we call the *recovery boost mode* where both PMOS devices within the cell undergo recovery at the same time. We refer to the period when the cell does not contain valid data that is never used by any other microarchitectural structure in the processor as “invalid period”.

The basic idea behind recovery boosting is to raise the node voltages (Node0 and Node1 in Figure 4.1) of a memory cell in order to put both PMOS devices into the recovery mode. This can be achieved by raising the ground voltage to the nominal voltage through an external control signal. The modified SRAM cell has the ground connected to the output of an inverter, as shown in Figure 4.2. CR is the control signal to switch between the recovery boost mode and the normal operating mode. During

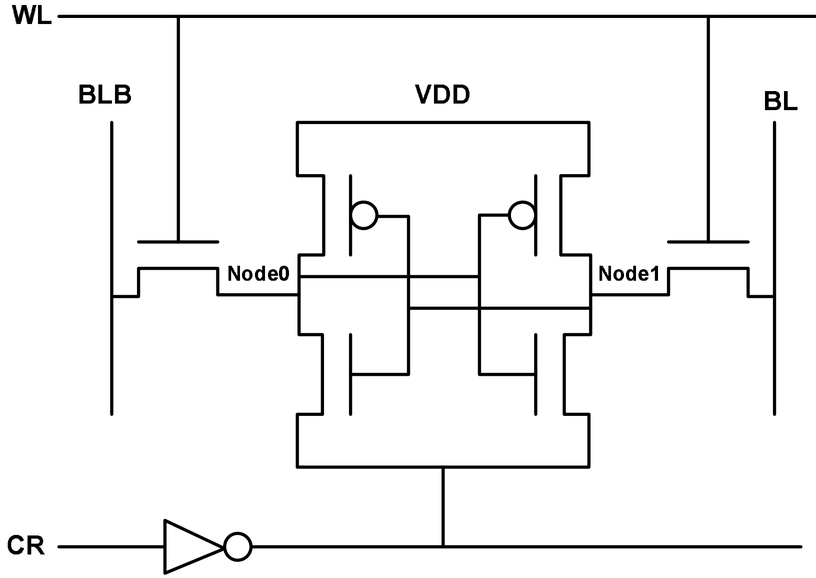


Figure 4.2: SRAM Cell Design for Recovery Boosting

the normal operating mode, CR has a value of '1' (V_{dd}), which in turn connects the ground of the SRAM cell to a value of '0'. With this connection, the SRAM cell can perform normal read, write, and hold operations. To apply recovery boosting, CR has to be changed to a '0' in order to raise the ground voltage of the SRAM cell to V_{dd}. This circuit configuration puts both PMOS devices in the SRAM cell into the recovery mode. A cell can be put into the recovery boost mode regardless of whether its wordline (WL) is high or low. Unlike read and write operations on a cell, putting a cell into the recovery boost mode does not require an access to its wordline. The operations of the modified SRAM cell are shown in Table 4.1.

However, the drawback of this approach is that it can take a long time to raise both the node voltages to V_{dd} in a high-performance processor that operates at a high clock frequency. This is illustrated in Figure 4.3, which presents the achieved PMOS gate voltages of a bitcell over time due to recovery boosting. The simulation is performed

CR	WL	BL	BLB	Node0	Node1	Operation
1	0	X	X	0/1	1/0	Hold
1	1	1	1	0/1	1/0	Read
1	1	1	0	0	1	Write ‘1’
1	1	0	1	1	0	Write ‘0’
0	X	X	X	1	1	Recovery Boost

Table 4.1: Modified SRAM cell operation

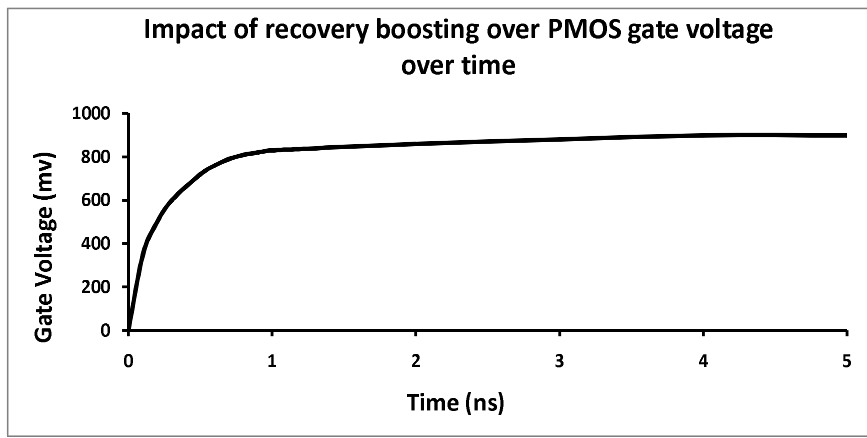


Figure 4.3: PMOS gate voltages of an SRAM bitcell due to recovery boosting using the modified SRAM cell shown in Figure 4.2 (V_{dd}=0.9V, T=90C)

using the Cadence Virtuoso Spectre circuit simulator [28] for the 32nm process using the Predictive Technology Model [31]. The operating temperature is 90C which is the average temperature in which high-performance processors operate [32]. We use this temperature value throughout the chapter for all the experiments. We can observe that this approach achieves the desired gate voltage (V_{dd}) within 3.33 ns. For a processor which operates at 3GHz frequency, it will take 10 cycles to switch to the recovery boost mode. Similarly, it takes around 10 cycles to go back to the normal operating mode from the recovery boost mode. However, our goal is to be able to switch between the recovery boost mode and the normal operating mode within a single cycle which is

critical for a high-speed SRAM structure, such as the issue queue, where instructions need to be woken up and selected within a single clock cycle, in order to expedite the execution of dependent instructions. As mentioned before, recovery boost mode is applied when an entry of the structure holds data that is considered "invalid" at the architecture-level. Entries in the high-speed structures change their status between valid and invalid very frequently. For example, we find from architecture simulations that an issue queue entry stays invalid for about 50 cycles before it changes its status to valid. In such scenario, the cell shown in Figure 4.2 will take 20 cycles of the 50 cycles (40% of the invalid period) to shift between modes, given that shifting to the normal operating mode takes place during the end of the invalid period. Thus, only 30 cycles could be utilized for the recovery process. On the other hand, if extra cycles are allocated to shift to the normal operating mode after the invalid period, that would have negative consequences on the processor performance. Therefore, single-cycle switching is required for the high-speed structures in the processor for the maximum utilization of the invalid states for the recovery process without any performance loss. Such single-cycle switching can be achieved by raising the bitlines along with the ground voltage to V_{dd}. There are various ways of incorporating such cells into SRAM arrays, which we will discuss shortly.

Recovery boosting can be provided at a fine granularity, such as for individual entries/rows of a memory array, or at a coarser granularity, such as for an entire array. We now discuss how the modified high-speed recovery boosting SRAM cells can be used in each of these scenarios and then discuss additional microarchitectural issues related to implementing recovery boosting.

4.1.1 Fine-Grained Recovery Boosting

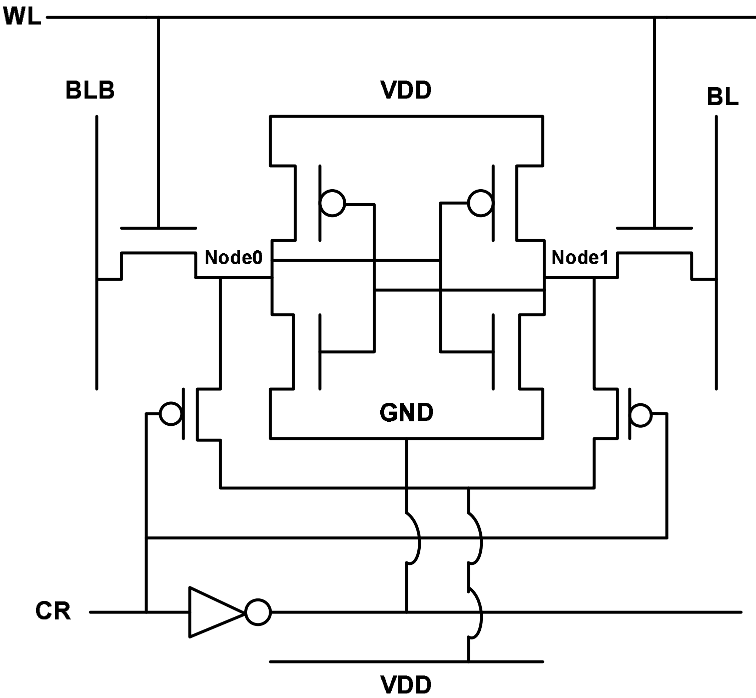


Figure 4.4: Modified SRAM cell with connection to the V_{dd} rail of an adjoining row

In the normal operating mode, the state of the bitlines change during read and write operations. Since a pair of bitlines is shared by all the memory cells in a given column in the array, even those memory cells that are not being read from or written to will have the voltage on their bitlines changing. In an ordinary SRAM array, these bitline transitions do not affect the normal operation of the cells. However, in order to perform recovery boosting of a memory cell, both bitlines of the cell need to be raised to V_{dd}. Therefore, we need to be able to isolate the bitlines of the memory cells that are in the recovery boost mode from the bitlines that are used for accessing other cells in the array. To provide this isolation, we extend the memory cell in Figure 4.2 with connections to the V_{dd} rail of an adjoining row or column via two PMOS access devices. The design of

the modified SRAM cell is shown in Figure 4.4 and an SRAM array that uses this cell for controlling individual entries to operate either in normal or recovery boost mode is shown in Figure 4.5.

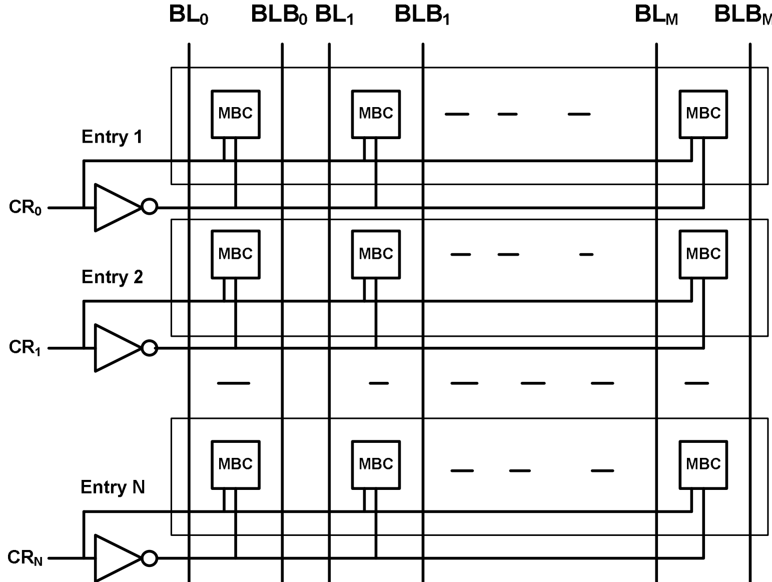


Figure 4.5: SRAM Array for Fine-Grained Recovery Boosting (N entries, M-wide)

In the memory cell design given in Figure 4.4, the CR signal serves the same purpose as before. When a value of '0' is input to the CR line to transition the cell into the recovery boost mode, in addition to raising the ground voltage, the two extra PMOS devices connected to the V_{dd} rail are also turned on. Therefore, by raising the ground and connecting the bitcell to V_{dd}, the cell can be transitioned into the recovery boost mode without affecting cells in other rows of the array.

We make the extra PMOS devices resilient against NBTI by using high-V_t transistors. Although high-V_t devices are slower, these devices are used only when transitioning the cell into the recovery boost mode and not when transitioning to the normal operating mode. Therefore, these devices do not impact performance but may delay the

transition into the recovery boost mode. Moreover, since these devices do not lie on the performance critical path, they are sized so as to minimize the overall area. However, the PMOS devices do consume leakage power. We quantify the power consumption in Section 4.2.

4.1.2 Coarse-Grained Recovery Boosting

In this approach, we use the SRAM cell design shown in Figure 4.2 instead of the one for fine-grained control. Here, a single control signal puts the entire array into the recovery boost mode. The control signal CR with a value of '0' raises the ground connection of each entry to V_{dd}. In this design, connections to the V_{dd} rail via the PMOS devices are not required. Instead we merely need to raise all the bitlines in the array to V_{dd} to transition all the cells in the array to the recovery boost mode.

Tradeoffs Between Fine-Grained and Coarse-Grained Recovery Boosting: Going in for the fine-grained approach entails an area overhead of having two additional PMOS devices for each memory cell which can be prohibitive for large SRAM arrays such as caches. On the other hand, the fine-grained approach provides single-cycle switching with greater flexibility in managing NBTI by exploiting the usage characteristics of individual entries in the structure. In this chapter, we evaluate the use of recovery boosting for the register file and issue queue. Due to the relatively small size of these structures (compared to caches), we use the fine-grained approach.

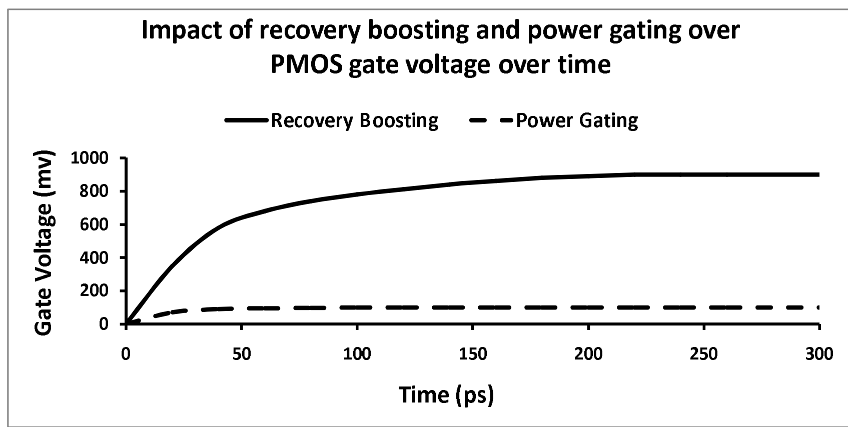


Figure 4.6: PMOS gate voltages of an SRAM bitcell due to recovery boosting and power gating (V_{dd}=0.9V, T=90C)

4.1.3 Other Issues

Difference Between Recovery Boosting and Power Gating: Similar to recovery boosting, power gating also involves a small change to the design of the SRAM cell and can also be used to combat NBTI [33]. As shown in Figure 4.6, we can observe that recovery boosting achieves the desired gate voltage (V_{dd}) within a very short interval of time (195 ps), whereas power gating achieves only about 11% of V_{dd}. Power gating requires several thousand nanoseconds to reach V_{dd} to provide recovery to the SRAM bitcell. Therefore, it can be used as a stress reduction approach for the high-speed structures since the duration of the invalid phases of these structures tend to be smaller than thousand nanoseconds. When a memory cell stores valid data, neither recovery boosting nor power gating can be applied and the PMOS devices in the cells will be stressed in a similar way. However, when the memory cell is idle and the data in the cell is no longer needed, it would be more beneficial to take advantage of recovery boosting.

Impact of Process Variation on Correct Functionality: In deep submicron technologies, intra-die process variation is an important issue. Different transistor parameters, including V_t, are affected by process variation and can impact circuit delay characteristics. V_t is affected by variations in the device geometry, random dopant number flucations, and mobile charges in the gate oxide [34]. Process variation will affect the delay characteristics of the 6T SRAM cell inherent in both the original and modified bitcells in a similar way. Process variation can also impact the V_t of the two V_{dd}-rail access transistors and the devices in the inverter connected to the CR line. If the V_t of the V_{dd} rail transistors is high, then transitioning the bitcell to the recovery boost mode may be slower. This could reduce the amount of time for which we can apply recovery boosting but will not affect the correctness of the SRAM cell operation. On the other hand, if their V_t is lower, the bitcell will transition into the recovery boost mode faster and the access devices will consume higher leakage power but will again not affect correctness. A delay in the PMOS device of the CR line inverter will again merely slow the transition into the recovery boost mode. However, a delay in the NMOS device in the inverter could affect the speed at which the cell transitions out of the recovery boost mode and into a normal operating mode, which can affect correctness. In order to handle this situation, we need to set the clock frequency such that this delay can be accomodated within a single cycle.

Recovery Boosting Does Not Exacerbate PBTI: Putting the PMOS devices into the recovery mode does not increase the stress on the NMOS devices in the memory cell. Stresses on the NMOS devices can lead to a phenomenon that is similar to NBTI called PBTI (Positive Bias Temperature Instability), which occurs when a positive bias (V_{gs} = V_{dd}) is applied to the NMOS device. As with NBTI, PBTI also generates interface

traps and increases the threshold voltage. PBTI is expected to become more important in future deep submicron technologies [35]. While in the recovery boost mode, both the ground and node voltages of the cell are raised to V_{dd}. Consequently, the V_{gs} of the NMOS devices in the inverters becomes zero and therefore these NMOS devices do not experience any positive bias. The access transistors are also not accessed during the recovery boost mode. Therefore recovery boosting does not exacerbate PBTI on the NMOS devices in the memory cell (and may in fact provide PBTI recovery [35]).

4.2 Designing Microarchitectural Structures that Support Recovery Boosting

Having discussed the basics of recovery boosting, we now turn our attention to designing SRAM-based microarchitectural structures that use this technique to provide resilience against NBTI. In this section, we present and evaluate the circuit-level design of two large SRAM-based structures within a 4-wide issue processor core, namely, the physical register file and the issue queue, which we modify to support recovery boosting. We study the design of a 128-entry multi-ported physical register file with 8 read-ports, 4 write-ports, and 64-bit entries. The issue queue uses a non-data-captured design [36] and consists of 64 entries with 4 read-ports, 4 write-ports, and 65 bits per-entry. The choice for the entry-size is based on the issue queue descriptions given in [17].

4.2.1 Physical Register File

Superscalar processors attempt to exploit Instruction-Level Parallelism (ILP) by fetching, decoding, executing, and retiring multiple instructions each clock cycle. In order to eliminate false dependences and support a large number of in-flight instructions, superscalar processors make use of register renaming. There are a number of microarchitectural options for implementing register renaming. In this chapter, we model a microarchitecture that uses a separate architected register file and a physical register file. Instructions whose source operand values are to be supplied by a physical register have their architected source register mapped to the appropriate physical register during renaming. These mappings from architected registers to physical registers are maintained in a Register Alias Table (RAT). The physical register is returned to the free list of registers when the next instruction that writes to the same architected register commits.

A physical register goes through a sequence of four states, shown in Figure 4.7: (i) it is not mapped to any producer instruction and is free (*Unmapped*), (ii) it is mapped to an instruction but it has not yet been written into by that instruction (*Mapped-Invalid*), (iii) it holds a valid value that has been written to it (*Mapped-Valid*), and (iv) it holds a valid value but the value is not read by any instruction before it is released to the pool of free registers (*Post Last-Read*). Once the register completes the *Post Last-Read* state, the register returns to the *Unmapped* state and remains in that state till the register-renaming logic chooses it again for a mapping.

There are three candidate states that one could use for recovery boosting: *Unmapped*, *Mapped-Invalid*, and *Post Last-Read*. When a physical register is in the *Un-*

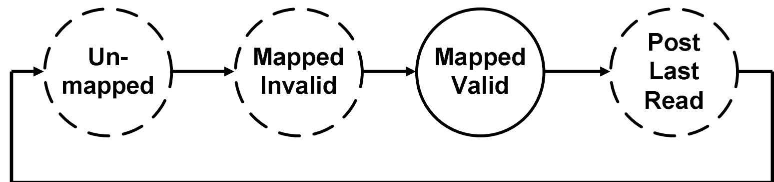


Figure 4.7: Register states. The candidate states for recovery boosting are shown in dashed circles.

Current State	**unmapped bit**	**completed bit**	**Control Signal (CR)**	**Switch to Recovery Boost?**
Mapped/invalid	0	0	0	Yes
Mapped/valid	0	1	1	No
Unmapped	1	0	0	Yes
Unmapped	1	1	0	Yes

Table 4.2: Control signal truth-table for a register

mapped and *Mapped-Invalid* states, it does not hold valid data and therefore we can put its cells into the recovery boost mode without affecting architectural correctness. The cells will need to be transitioned into the normal operating mode when moving from the *Mapped-Invalid* to the *Mapped-Valid* state, which occurs when the producer instruction has completed its execution and forwards the value to the register file. *Post Last-Read* is a more complex situation. Several cycles may elapse between the last

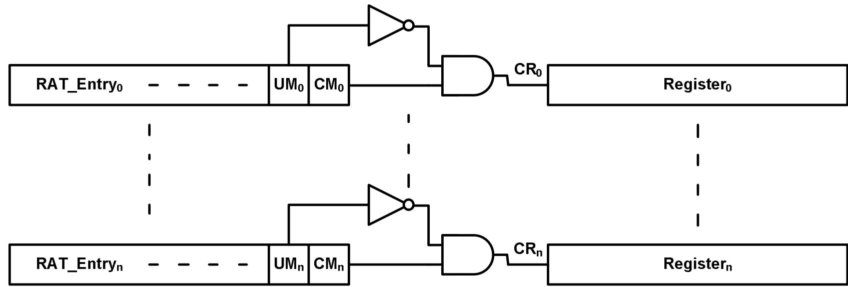


Figure 4.8: Control Logic for generating Control Signal CR (UM_x = 'unmapped' bit for register x and CM_x = 'completed' bit for register x)

time that the physical register is read and when the register is released. This time period could potentially be exploited for recovery boosting but it is challenging to precisely determine when the last-use of a register is complete. Although there have been proposals to exploit this time window to speculatively release registers early for the sake of performance and power [37, 38], these techniques entail additional complexities of tracking/predicting the last-use of registers and for maintaining and restoring the old state of the early-released registers if needed. To reduce the complexity of implementing recovery boosting, we put registers into the recovery boost mode only when they are in the *Unmapped* and *Mapped-Invalid* states. As we will show in Section 4.4, putting registers into the recovery boost mode just during these two states still provides substantial improvements in reliability.

Since the register renaming logic tracks whether a physical register is in the *Unmapped* or *Mapped-Invalid* state, the control signal for the recovery boost mode is set by the renaming logic. Each RAT entry has an 'unmapped' bit and a 'completed' bit that denotes whether the given physical register that the RAT entry points to has been mapped and whether it has been written to respectively. Using these two bits, we can implement the control signal for recovery boosting using the truth-table and the corresponding logic shown in Figures 4.2 and 4.8 respectively.

4.2.2 Issue Queue

The issue queue houses instructions that have been fetched, decoded, and renamed and are pending execution. Instructions are dynamically scheduled from the issue queue based on the availability of their source operands and functional units. Instruction is-

sue consists of two steps - wakeup and select - which both need to be completed within a single clock cycle for high performance. Instructions that have finished execution broadcast their result tags to all the instructions in the issue queue. Each instruction in the issue queue compares the broadcasted tags with its own source tags for a possible match. Once both the source operand tags of an instruction have matched, the instruction is ready to be issued to a functional unit (instruction wakeup). A subset of the ready instructions are then selected to be issued to the functional units (instruction select). These instructions obtain their source input operands from the register file or the bypass network and then proceed to use the functional units granted to them.

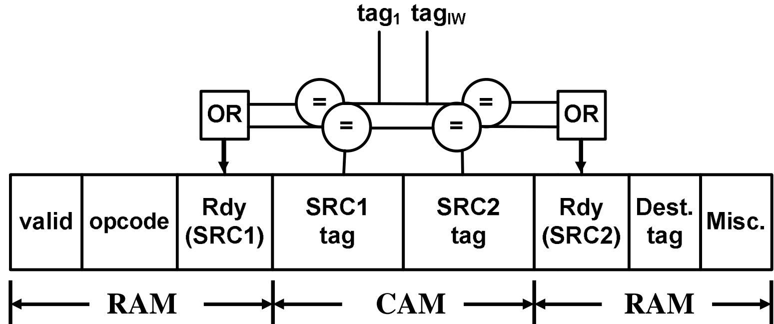


Figure 4.9: An issue queue entry.

We model a non-collapsing issue queue that is organized as a circular FIFO with head and tail pointers similar to the design proposed by Folegnani and Gonzalez [39]. We design the issue queue entry to be similar to the one described by Palacharla et al. [40]. Conventional issue queues have a CAM/RAM structure where the CAM holds the source operand tags and the RAM holds the remaining information. The structure of the issue queue entry is shown in Figure 4.9. Each entry has a valid-bit to indicate its status. The valid-bit is set when the entry is allocated for a dispatched instruction and is reset when the instruction is issued and leaves the issue queue. We put invalid entries

into the recovery boost mode. (The valid-bit itself is not put into the recovery boost mode to ensure correct operation of the instruction scheduler). The CAM performs tag-matching operations against all the broadcasted tags each clock cycle. In order to do this, each CAM entry has a set of comparators and the number of comparators required depends on the issue width of the processor. The design of the CAM part of the issue queue for a single bit is shown in Figure 4.10. In each cycle, each matchline is precharged. If there is a mismatch between the tag data in the memory cell and the broadcasted result tag in any of the CAM cells in the issue queue entry, then the corresponding matchline is discharged; otherwise the matchline stays high. If any of the matchlines for a given operand tag entry stays high, its corresponding ready signal (RDY) is asserted high via the OR-block shown in the figure.

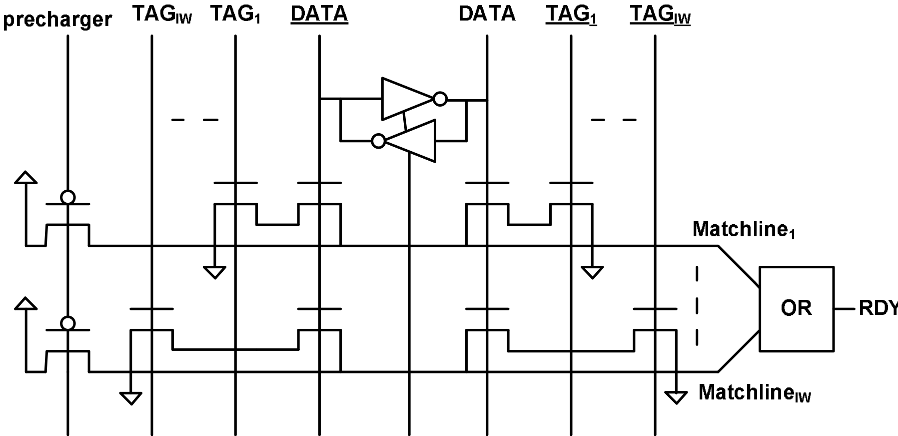


Figure 4.10: CAM structure of an issue queue entry (IW = issue width)

To provide recovery boosting, the memory cells of the RAM and CAM structures are composed of the modified SRAM cells shown in Figure 4.4. The modified issue queue entry is shown in Figure 4.11. The valid-bit works as the control signal (CR) for the entry. When the memory cells in the entry transition to the valid state, the CR

signal becomes high which pulls the ground down to low and the entry works in the normal operating mode. When the entry transitions to the invalid state, the CR signal becomes low and puts the memory cell into the recovery boost mode. When in the recovery boost mode, both nodes of the memory cells in both the RAM and the CAM parts are raised to V_{dd}. Due to the high node voltages, the comparators in the CAM will be triggered leading to a discharge of the matchline. To avoid this unnecessary precharging and discharging of the matchline (which wastes power), we further modify the issue queue entry so that the prechargers of the matchlines are connected to the CR signal. When CR is high in the normal operating mode, the matchlines will be precharged to V_{dd} and the tag-matching process will continue each cycle. When CR is low during the recovery boost mode, the matchlines stay low and therefore do not discharge.

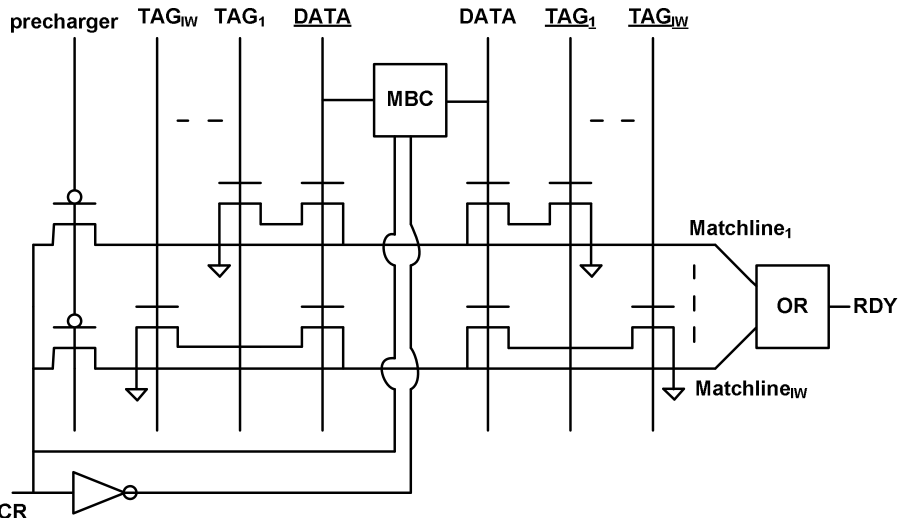


Figure 4.11: Modified CAM Structure (IW = issue width). MBC is the Modified Bit-Cell for recovery boosting.

4.2.3 Circuit-Level Simulation Results

We now present the results from a circuit-level analysis of the designs discussed in Sections 4.2.1 and 4.2.2. We perform SPICE-level simulation using the Cadence Virtuoso Spectre circuit simulator [28] to verify the functionality of our designs and determine their area and power consumption. Our experiments are carried out for the 32nm process using the Predictive Technology Model [31]. Our bitcell device sizes are: PMOS = 58nm × 33nm, NMOS = 87nm × 33nm, Access Transistor = 58nm × 41nm. The supply voltage (V_{dd}) and the operating temperature (T) are 0.9V and 90C respectively. For each structure, we simulate two designs: the baseline design that uses conventional 6T SRAM cells (which do not provide recovery boosting) and the design that uses the modified SRAM cells discussed in Section 4.1 to provide fine-grained recovery boosting.

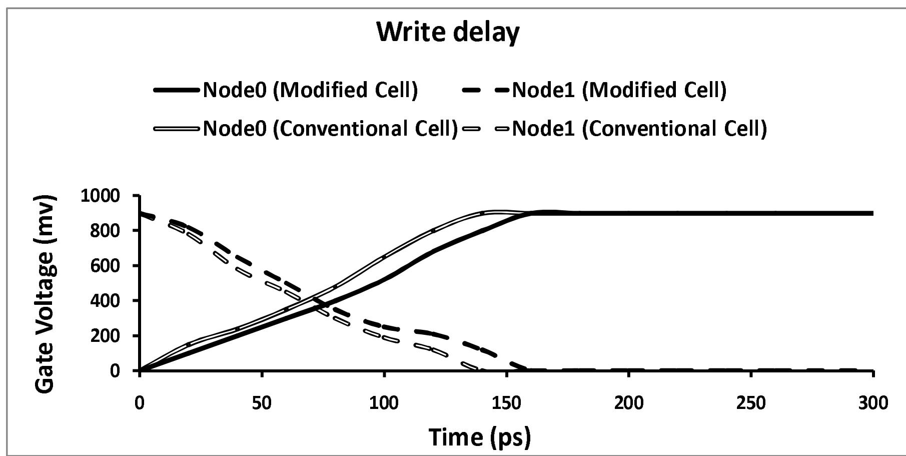


Figure 4.12: Write delay of the modified bitcell. Node0 and Node1 are the node voltages of the bitcell (V_{dd}=0.9V, T=90C).

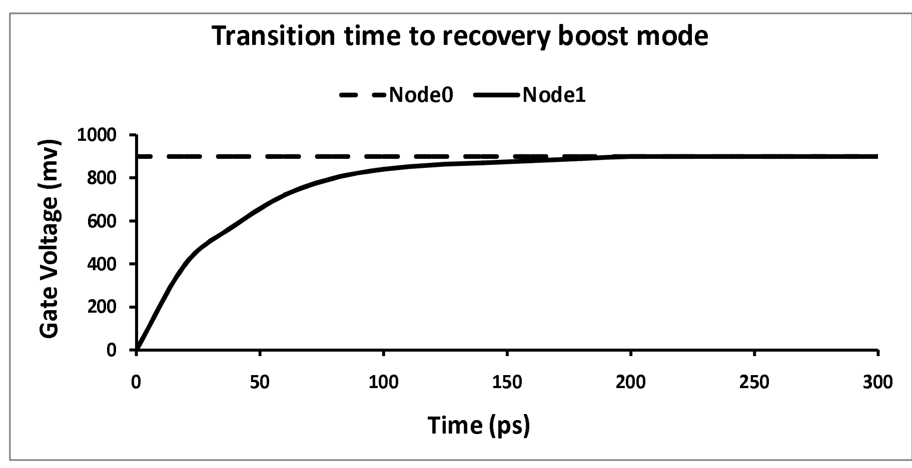
Functionality

We evaluate two aspects of the functionality: (i) whether we can perform read, write, and hold operations on the modified SRAM cells, and (ii) whether the modified cells correctly switch between the normal operating modes and the recovery boost mode. To evaluate these, we looked at the waveforms of the voltage variations at the nodes and bitlines of the cell in one clock cycle. We examine these waveforms for the read, write, and hold operations and also for the transitions between the recovery boost mode and the normal operating modes. We take into account the extra inverter delay required for changing the ground voltage of the cell during these transitions.

In Figure 4.12, we show that the node voltages (Node0 and Node1) of the modified cell for recovery boosting takes 160 ps to reach desired voltage values whereas, the conventional 6T cell takes 140 ps. Even though the write delay is increased by 20 ps because of the increased capacitance in the modified cell, this operation can be done within a single cycle of a high performance processor. Since the read and hold operations behave in a similar way for both modified and conventional 6T cells, we do not present the waveforms for these operations.

Figure 4.13(a) shows that the required time to switch to recovery boost mode is 190 ps. To switch to the recovery boost mode, both node voltages have to rise to V_{dd}. Node0 stays in V_{dd} and Node1 takes 190 ps to rise to V_{dd}. Since a bitcell transitions from recovery boost mode to normal operating mode on a write, we ran simulation to confirm that we can succesfully write to the cell after the transition within a cycle. Figure 4.13(b) shows the required time to switch to normal operating mode from recovery boost mode with a write operation. In recovery boost mode, both node voltages

(Node0 and Node1) stays in V_{dd}. Therefore, with a write operation, Node0 has to stay in V_{dd} and Node1 has to be pulled down to GND. As we can see from the figure, Node1 reaches the desired value (GND) within 140 ps. Therefore, shifting to the recovery boost mode and to come back to the normal operating mode from it take 190 ps and 140 ps respectively.



(a) Transition to recovery boost mode

Transition to normal operating mode with write operation

Node0 Node1

Gate Voltage (mv): 0, 200, 400, 600, 800, 1000

Time (ps): 0, 50, 100, 150, 200, 250, 300

(b) Transition to normal operating mode

Figure 4.13: Transition between recovery and normal modes. Node0 and Node1 are the node voltages of the bitcell (V_{dd}=0.9V, T=90C).

Our simulations indicate that we can correctly perform read, write, and hold operations on the registers and the issue queue entries that use the modified SRAM cells in the register file and the issue queue. The extra circuitry for recovery boosting is not active during the normal operating mode and therefore they do not interfere with normal operations on the cells. Since a register transitions from the recovery boost mode to the normal operating mode (i.e., *Mapped-Invalid* state to the *Mapped-Valid* state) on a write, our simulations confirm that we can successfully write to the cells after the transition. The same holds true for an issue queue entry. We also found that the matchlines for the CAM cells that are in the recovery boost mode are not precharged so that they never trigger a match for those cells.

Clock Frequency Setting: In our simulations, we found the smallest possible cycle-time for the modified SRAM cell to be 220ps (a clock frequency of 4.5 GHz). We choose a more conservative cycle-time of 333ps, which corresponds to a clock frequency of 3 GHz. We found the delay of the high-V_t access transistors that connect to the V_{dd} rail to be small enough to transition the cell into the recovery boost mode within a single cycle for the 3 GHz clock frequency.

Area

We designed our structures for both the baseline and recovery boosting cases to occupy the minimum area required to provide correct functionality. Care was taken to size the devices so that they are of minimal size while meeting the 3 GHz clock frequency requirement. We calculate the area of the structures based on the device sizes in their respective netlists. We assume that the area overhead due to any new routing or interconnect can be minimized by an optimized layout. In a typical SRAM array,

the cells are laid out in the array in a mirrored fashion so that the same interconnect could be shared by adjacent rows and columns. Under these assumptions, the extra PMOS devices and the inverters would dominate the area overhead and accounting for the extra devices would give a first order approximation of the SRAM array area with the modified cells. The overheads for the multi-ported register file and issue queue are given in Figure 4.14.

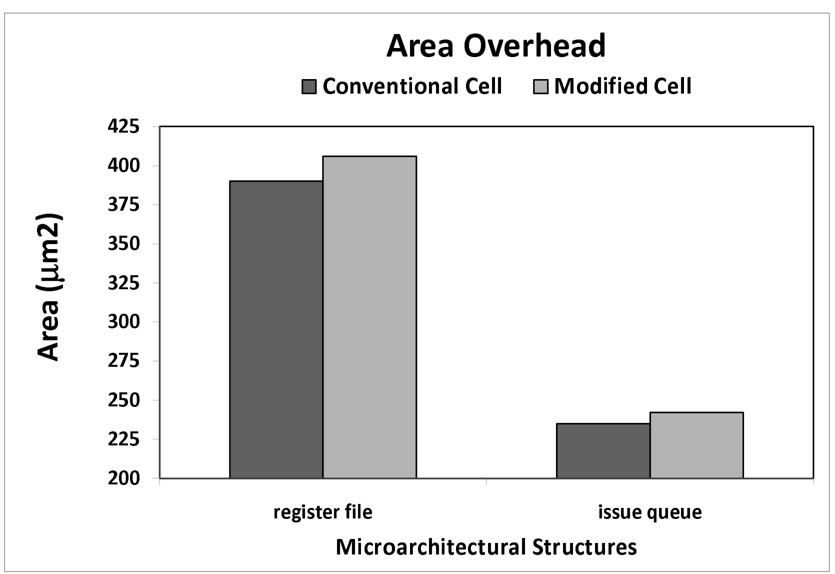


Figure 4.14: Area of the register file and the issue queue for designs that use conventional 6T cells and cells modified to support recovery boosting.

Since the register file has 8 read-ports and 4 write-ports, each bitcell has 20 transistors: 4 transistors for the inverter-pair and 8 transistors each for the write and read-ports (for supporting single-ended reads). Similarly, the issue queue has 4 read and write-ports respectively and has 16 transistors per bitcell. To support recovery boosting, we add 2 extra transistors of minimal size to each cell and one extra inverter for an entire row of 64 bitcells, in the case of the register file, or 65 bitcells for the issue

queue. Therefore, adding the extra transistors for recovery boosting to these heavily multi-ported structures is expected to add only a small amount of area. Indeed, we can see that the area of the physical register file and issue queue that use the modified cells are 4% and 3% respectively more than their baseline designs. This overhead is roughly equivalent to the area occupied by three registers in the modified register file and two entries in the modified issue queue. We can therefore design the register file and issue queue to be area-neutral with respect to the baseline (i.e., occupy the same area as the baseline design) by having their capacities reduced by three registers and two entries respectively. The rationale behind going in for area-neutral structures is to minimize the impact of designing structures that employ recovery boosting on the processor floorplan. Going for the area-neutral design of the structures could affect the performance of the processor. The performance impact of these area-neutral designs are evaluated in Section 4.4.

Dynamic and Leakage Power Consumption

Figure 4.15 gives the power consumption of a single register for both the baseline design and the one that uses the modified SRAM cell. Similarly, Figure 4.16 gives the power consumption of a single issue queue entry. For the register, we show the power consumed for the read, write, and hold operations as well as when the cells are in the recovery boost mode. For the issue queue entry, in addition to the power consumed in the recovery boost mode, we quantify the power consumed in each of the three normal operating modes. For each of these modes, we present the power consumption for two scenarios: (i) when both source tags of an entry mismatch with the ones broadcast down the issue queue in the same cycle, which is the highest power

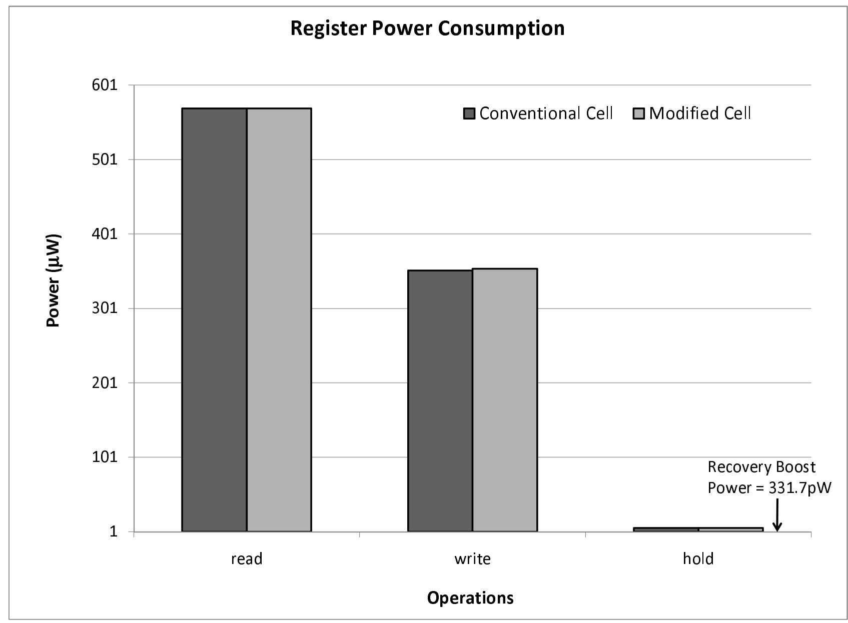


Figure 4.15: Power consumption of a single register entry (V_{dd}=0.9V, T=90C).

consumption scenario since all the matchlines discharge, and (ii) when both source tags match in the same cycle, which consumes the least amount of power.

We can see that the power consumed by the designs that use the modified SRAM cells for the read, write, and hold operations are nearly equal to those of the baseline designs. The maximum increase in power is less than 1% for the register and less than 2% for the issue queue entry. The power consumption of the issue queue entry is higher than the register because of its CAM/RAM structure. The slight increase in power for the recovery boost designs is due to leakage in one of the PMOS access transistors that connect to the V_{dd} rail. The sources of the PMOS access transistors are connected to V_{dd} and the drains are connected to the nodes. Therefore, based on whether a cell holds a '0' or a '1', one of the two PMOS devices will leak. Since we use high-V_t PMOS devices as the access transistors for the cells (to reduce the impact of NBTI),

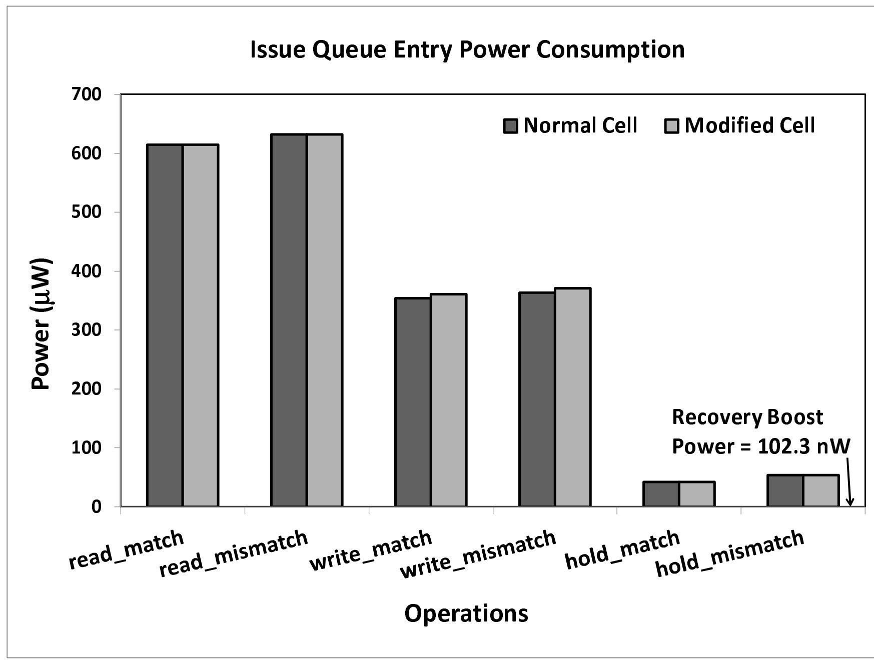


Figure 4.16: Power consumption of a single issue queue entry (V_{dd}=0.9V, T=90C).

the leakage power of these transistors is also reduced.

In memory arrays that use conventional SRAM cells, the cells will normally be operating in the hold mode when they house invalid data. However, when the modified cells are used, cells that hold invalid data can operate in the recovery boost mode. We can see that the power consumed in the recovery boost mode is orders of magnitude less than in the hold mode. This is because the recovery boost operation raises Node0 and Node1 (shown in Figure 4.4) and the ground to V_{dd}, which cuts off the path from V_{dd} to ground and significantly reduces the leakage currents. Finally, there is a small power benefit at the structure level since we use area-neutral designs for the physical register file and the issue queue that are slightly smaller than the baseline designs.

4.3 Experimental Methodology for the Architecture Level Analysis

Having seen the circuit-level design of the physical register file and the issue queue to support recovery boosting, we now evaluate the impact of using these techniques at the architecture level. We carry out our architecture-level evaluations via execution-driven simulation using the M5 simulator [29]. We use the system-call emulation mode of M5. Our workloads consist of all 26 benchmarks from the SPEC CPU2000 benchmark suite [30]. The benchmarks are compiled for the Alpha ISA and use the reference input set. We perform detailed simulation of the first 100-million instruction SimPoint for each benchmark [41]. We model a 4-wide issue core, which is similar to those in multicore processors. We assume the initial threshold voltage of the PMOS devices in the memory cells to be 0.2 V and the service life of the processor to be 7 years based on the work by Tiwari and Torrellas [15].

Reliability Metric - Read Static Noise Margin (SNM): NBTI causes an increase in the threshold voltage of the PMOS transistors. In the case of SRAM cells, this shift in V_t could increase the time needed for reading from and writing to the cells. NBTI could also decrease the read SNM (SNM) of the cells. The SNM is a measure of the stability of the cell and specifies the maximum amount of voltage noise that can be tolerated at the nodes of the memory cell before the contents of the cell get flipped [42]. Previous work [18] has shown that, of these three metrics, the SNM is the one that is most heavily affected by NBTI and therefore we use SNM as the reliability metric in this chapter.

Initially, before the processor is used for executing workloads, the bitcells in the

register file and the issue queue are designed such that their SNM is not limited by the strength of the PMOS devices. But after these structures are exercised by workloads, their SNM gets limited by the strength of the PMOS devices due the impact of NBTI on V_t. We capture this impact by tracking the stress and recovery cycles on all the PMOS devices in the register file and the issue queue (based on our circuit-level designs of these structures) over the course of an architecture simulation and extrapolate the statistics to calculate the degradation in V_t after the 7-year service life. We then feed the V_t values of these PMOS devices into the Cadence Virtuoso Spectre circuit simulator to calculate the SNM of all the cells in a structure at the end of the 7-year period and use the smallest value to denote the SNM for that structure.

4.4 Architecture-Level Simulation Results

We now study the impact of putting memory cells of registers and issue queue entries into the recovery boost mode when they hold invalid data. As discussed earlier, we put registers into the recovery boost mode when they are in the *Unmapped* and *Mapped-Invalid* states. We present results only for the integer register file for two reasons. First, the integer benchmarks have very few floating-point instructions and therefore rarely exercise the floating-point register file. Second, several of the floating-point benchmarks have a large number of integer instructions in their first 100-million instruction SimPoint and therefore significantly exercise the integer register file. (9 out of the 14 floating-point workloads have more integer than floating-point instructions in their first SimPoint). As a result, the integer register file is exercised to a greater extent than the

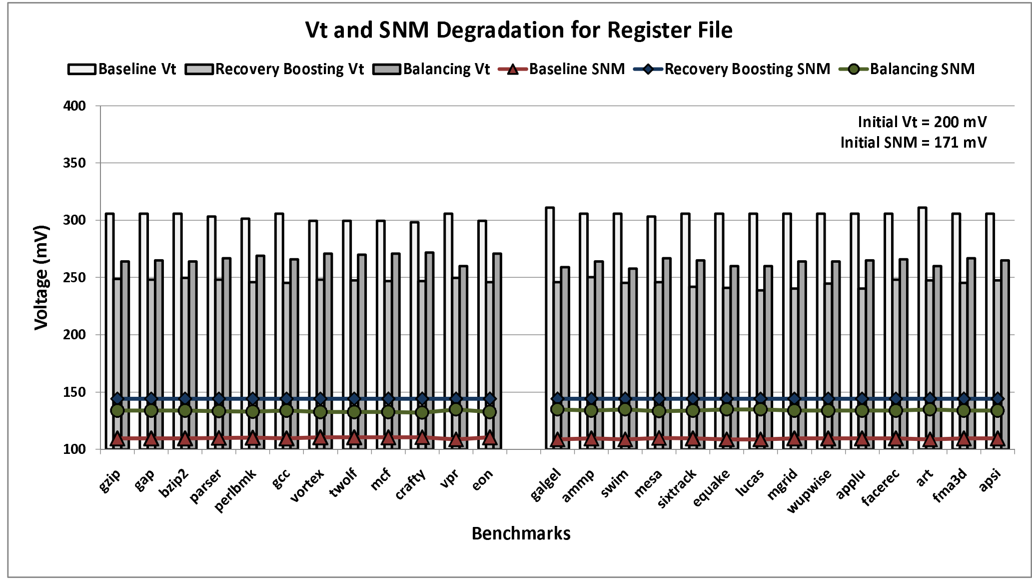


Figure 4.17: V_t and SNM degradation for the RF for the *Baseline*, *Recovery Boosting* and *Balancing* configurations (V_{dd}=0.9V, T=90C).

the floating-point register file for most of the benchmarks. Issue queue entries are put into the recovery boost mode when they no longer hold valid data.

We evaluate three different processor configurations, which we denote as: *Baseline*, *Recovery Boosting*, and *Balancing*. *Baseline* models 4-wide issue core that do not use any NBTI mitigation technique. *Recovery Boosting* replaces the integer register file and the issue queue of the baseline configuration with their counterparts that support recovery boosting. We assume that the modified structures for *Recovery Boosting* are designed to be area-neutral with respect to *Baseline* by trading off a small amount of storage capacity to accommodate the extra area required to implement recovery boosting. Based on our area evaluations in Section 4.2, we assume that the modified integer register file and issue queue have 125 registers and 62 entries respectively. In all our simulations, we find that this reduction in capacity has a negligible impact on

performance and therefore we do not present detailed performance results. *Balancing* denotes a recovery enhancement scheme similar to the one proposed in [17] that uses the same time intervals that *Recovery Boosting* exploits to balance the degradation of the two PMOS devices in the memory cell. As pointed out by Abella et al. [17], flipping the contents of the memory cells only when they hold invalid data instead of when they hold both valid and invalid data, for which additional circuitry is required [18], is the preferable approach for high-speed SRAM structures in order to not increase their delay significantly. Since the access times of the physical register file and the issue queue have a strong impact on processor performance, the *Balancing* technique is applied only when the memory cells hold invalid data. We optimistically assume that *Balancing* does not impose any additional area overheads over the baseline design and that it can keep the inputs to each PMOS device at a logic '0' exactly 50% of the time whenever the cells are in this mode.

For the remainder of this chapter, we will refer to the integer register file and the issue queue as RF and IQ respectively.

4.4.1 Physical Register File Results

Figure 4.17 shows the V_t degradation and the resultant SNM for the RF across the benchmarks due to NBTI after the 7 year service life for the *Baseline*, *Recovery Boosting* and *Balancing* configurations. As mentioned earlier, the initial V_t is 200 mV and using this value, we get the initial SNM of the RF to be 171 mV before the RF is stressed by the benchmark. For the *Baseline*, on an average, the V_t degrades to 305 mV across the benchmarks, which leads to an average SNM of 109 mV. Therefore, *Base-*

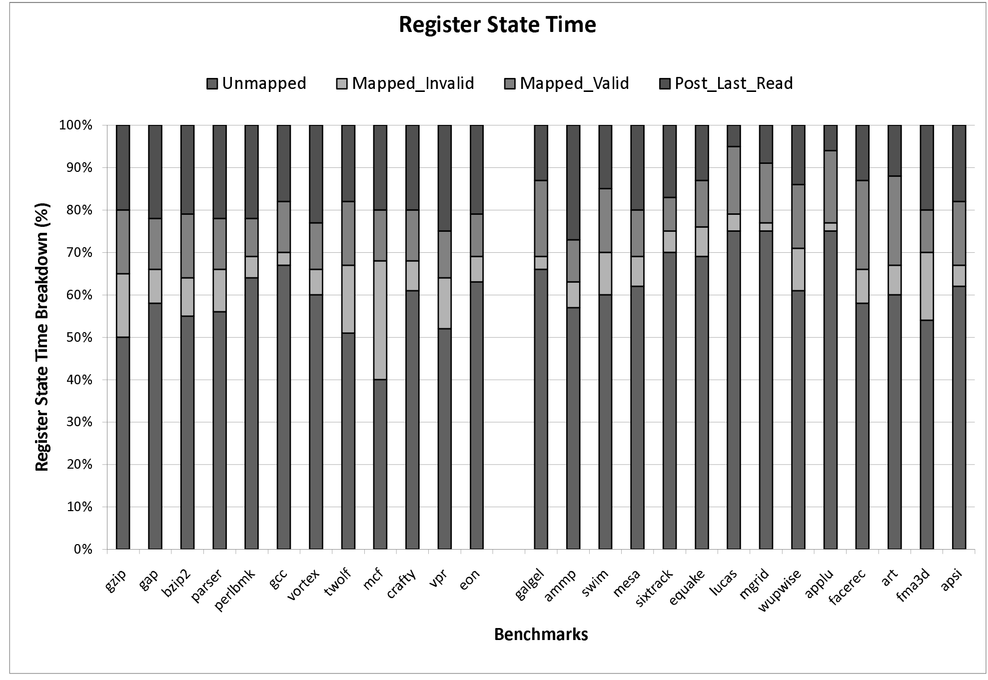


Figure 4.18: Breakdown of time spent by the registers in different states. The lowest part of each stacked bar is the *Unmapped* state.

line causes about 37% degradation in SNM. To improve the SNM over the *Baseline*, *Recovery Boosting* takes advantage of the invalid periods of the RF to apply recovery to the registers. A breakdown of the time spent by the registers in the four different states given in Figure 4.7 is shown in Figure 4.18. The values given in the graph are an average over all the registers and over the entire SimPoint for each benchmark. As we can see, the registers are in the *Unmapped* and *Mapped-Invalid* states for a large fraction of time for most of the benchmarks. Therefore, we have significant opportunities to apply recovery boosting for the RF. The impact on the SNM as a result of using recovery boosting is given in Figure 4.17. As Figure 4.17 shows, the average degraded V_t stays close to 245 mV because of the applied recovery to the RF bitcells. This causes the SNM to degrade to an average value of 144 mV, which is about 15%

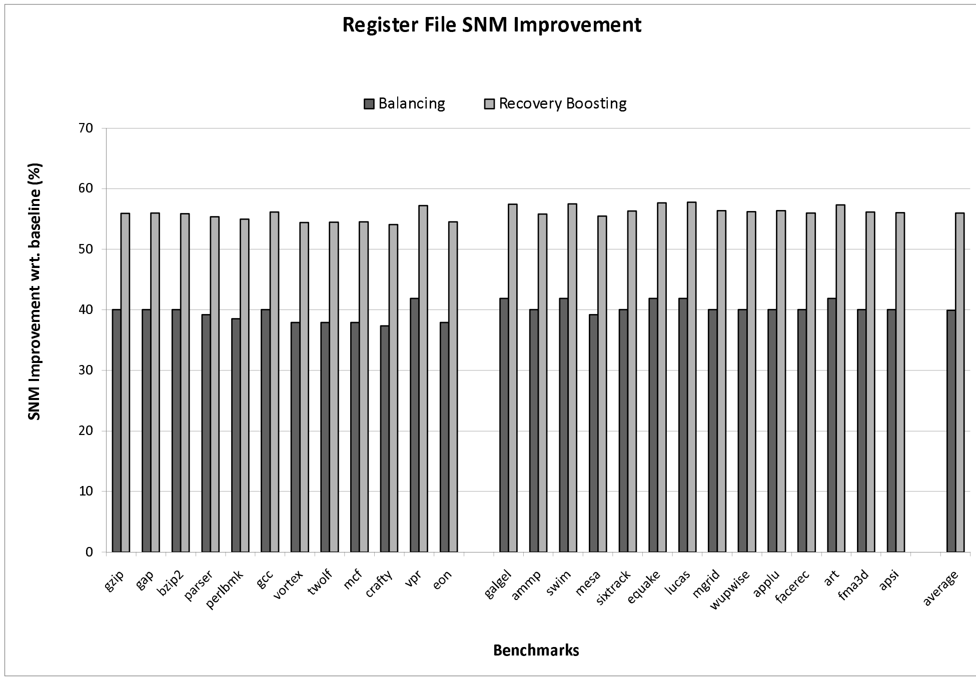


Figure 4.19: Improvement in the Static Noise Margin for the RF over the *Baseline* processor configuration (V_{dd}=0.9V, T=90C).

degradation in SNM over the initial condition. Similarly, for the *Balancing* approach, the average degraded V_t stays close to 265 mV and the SNM degrades to an average value of 133 mV.

In Figure 4.19, each pair of bars shows the improvement in the SNM over *Baseline* for *Recovery Boosting* and *Balancing* respectively. As Figure 4.19 shows, while *Balancing* provides a good improvement in the SNM, *Recovery Boosting* provides significantly higher reliability benefits by virtue of its ability to put both PMOS devices into the recovery mode. Across all the benchmarks, *Balancing* provides a 40% improvement in the SNM while *Recovery Boosting* provides a 56% improvement. These results clearly highlight the benefits of recovery boosting as a technique to mitigate NBTI in the RF.

4.4.2 Issue Queue Results

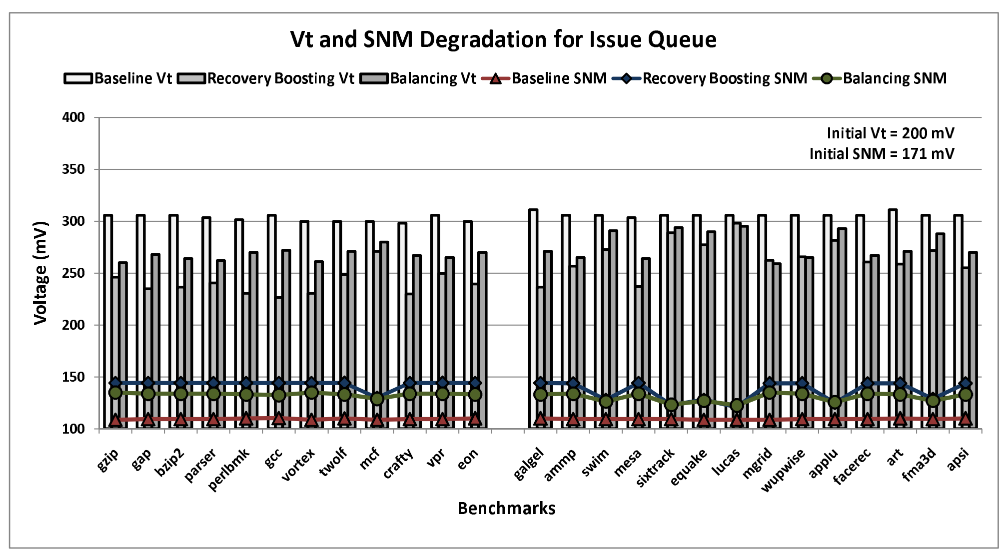


Figure 4.20: V_t and SNM degradation for the IQ for the *Baseline*, *Recovery Boosting* and *Balancing* configurations (V_{dd}=0.9V, T=90C).

Figure 4.20 shows the V_t degradation and the resultant SNM for the IQ across the benchmarks due to NBTI for the *Baseline*, *Recovery Boosting* and *Balancing* configurations. Similar to the RF, the initial V_t and the SNM are 200 mV and 171 mV respectively. On an average, the V_t degrades to 305 mV and the SNM degrades to 109 mV for the *Baseline*. Unlike the RF, the V_t and the SNM of the IQ varies across different benchmarks for the *Recovery Boosting* configuration. Specially, the V_t and the SNM varies a lot for the floating-point benchmakrs. The reason behind this is illustrated in Figure 4.21 where the breakdown of the time spent by the IQ entries in the *Valid* and *Invalid* states for each benchmark is shown. The time-breakdown is an average over all the entries in the IQ and over the entire SimPoint of each benchmark. The duration of the *Invalid* state is much shorter for some floating-point benchmarks as shown in the

figure. On an average, V_t degrades to 254 mV and the resultant average SNM becomes 140 mV for the *Recovery Boosting* configuration. Similarly, V_t degrades to 272 mV and the resultant average SNM becomes 131 mV for the *Balancing* configuration.

While the RF results show that *Recovery Boosting* is consistently superior to *Balancing*, the IQ shows a more varied trend. The reliability results are given in Figure 4.22. For the integer benchmarks, which are in the left-hand side group in the graphs in Figure 4.21, the IQ entries are in the *Invalid* state for a large fraction of the time (over 76% on average) and therefore enjoy significant benefits from recovery boosting. The exception to this is `mcf`, where the IQ entries spend only 34% of the time in the *Invalid* state. The difference in the SNM improvement between *Balancing* and *Recovery Boosting* is much smaller for this benchmark and so is the overall benefit over *Baseline*. This is because 37% of the instructions in `mcf` are memory instructions (33.5% are loads and 3.5% are stores). Given the large number of load-instructions and the fact that load values are consumed by subsequent instructions in its dependence-chain that wait in the IQ, they have a significant impact on the occupancy characteristics of the IQ. We find that the L1 data-cache miss rate for `mcf` is high (54.5%), which leads to these dependent instructions occupying IQ entries for a longer period of time. As a consequence, there are fewer opportunities to apply either *Balancing* or *Recovery Boosting* to the IQ for `mcf` compared to the other integer benchmarks. Note that although the IQ entries are in the *Valid* state while these instructions wait, the destination register of a producer instruction remains in the *Mapped-Invalid* state in the RF till the value is actually ready to be written to the register. We can see in Figure 4.18 that the registers spend a longer time in the *Mapped-Invalid* state for `mcf` than for the other integer benchmarks. Therefore, the RF still benefits from recovery boosting during

this period.

The floating-point workloads exhibit a range of behaviors. *Recovery Boosting* provides significant improvements in the SNM for more than half the benchmarks and the benefits provided are much better than *Balancing*. However, for certain workloads (`swim`, `equake`, `fma3d`) the benefits provided by *Recovery Boosting* and *Balancing* are nearly the same. Most interestingly, for `sixtrack`, `lucas`, and `applu`, *Balancing* provides slightly *better* improvement in the SNM than *Recovery Boosting*. We now explain why this happens.

First, as we expect, there is a relationship between the benefits that *Recovery Boosting* provides and the amount of time that the IQ entries spend in the *Invalid* state, as Figure 4.21 indicates. There are several factors that influence the amount of time that various workloads spend in the *Invalid* state, such as the percentage of high-latency instructions in the instruction mix of the benchmark and the memory system performance (both the hit-rates of the L1 instruction and data caches as well as the average miss latencies). For workloads that have a significant number of high-latency floating-point instructions, the instructions data-dependent on those high-latency instructions will occupy IQ entries for a longer time. `sixtrack`, `lucas`, and `applu` contain a significant number of floating-point instructions in their instruction-mix (65%, 64%, and 52% respectively). These three workloads also have the highest fraction of floating-point multiply instructions among all the floating-point workloads (36%, 23%, and 24% of the overall mix respectively). Since a floating-point multiply takes 4 cycles to execute, instructions on their dependence-chains wait in the IQ for a long time in the *Valid* state. Similarly, for workloads that have a high percentage of load instructions, the IQ entries occupied by the loads and their dependence-chains will remain in

the *Valid* state for a longer time if the loads frequently miss in the cache. In general, the IQ occupancy behavior depends on both the characteristics of the application and the exact microarchitectural configuration of the processor. However, since we do not optimize either of these factors for *Balancing* and *Recovery Boosting*, it is more important to understand when one scheme will be more beneficial than the other in terms of reliability.

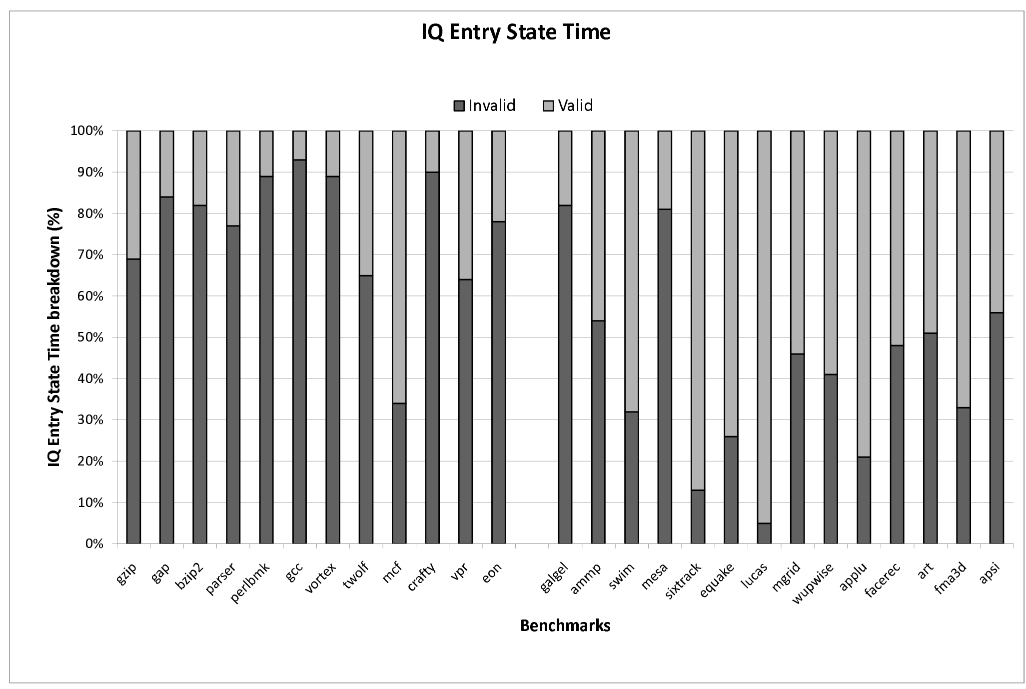


Figure 4.21: Breakdown of time spent by the IQ entries in the *Valid* and *Invalid* states..

To achieve a good SNM, it is important to minimize the increase in V_t due to NBTI of both PMOS devices in the memory cell. Additionally, it is also important to ensure that the *difference* in the threshold voltages between the two PMOS devices is kept as small as possible. *Recovery Boosting* addresses the first condition whereas *Balancing* addresses the second. When an IQ entry holds valid data for a long period of time, as

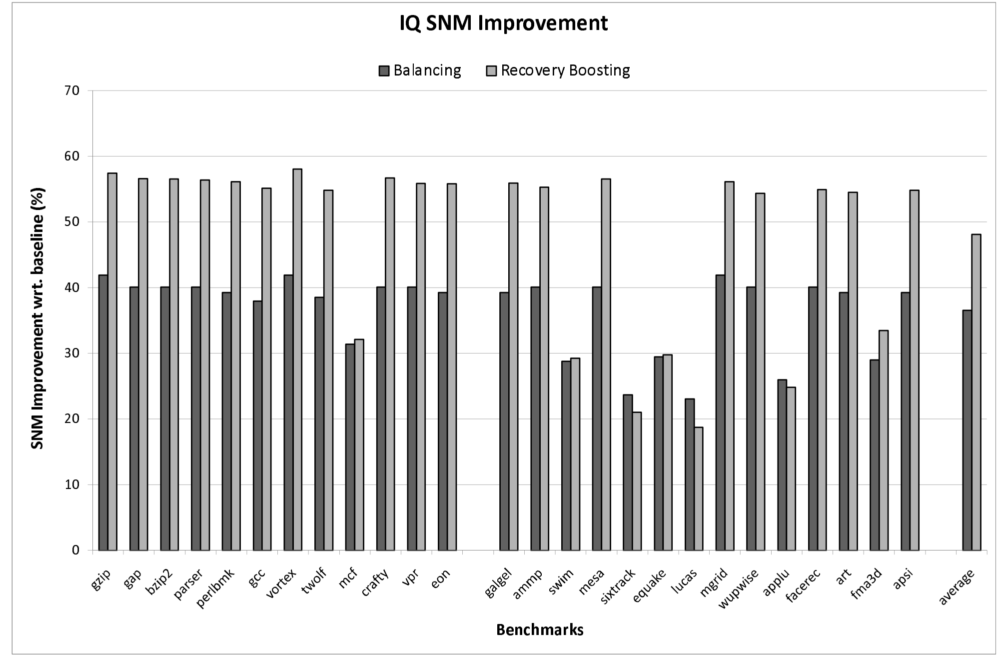


Figure 4.22: Improvement in the Static Noise Margin for the IQ over the baseline processor configuration (V_{dd}=0.9V, T=90C).

is the case for `sixtrack`, `lucas`, and `applu`, one of the PMOS devices in each memory cell stays in the stress phase while the other in the recovery phase. This increases the difference in the V_t between the two PMOS devices and therefore degrades the SNM. When the IQ entry is in the *Invalid* state for a short period of time, *Recovery Boosting* reduces the V_t of both PMOS devices all the memory cells. However, *Balancing* flips the bits stored in those cells so that the PMOS device that was in the recovery phase while the IQ entry was valid now enters the stress phase and vice-versa, which tends to reduce the difference in V_t between the two PMOS devices. As a result, for workloads where the IQ entries spend very little time in the *Invalid* state, the SNM for the *Balancing* scheme is better than for *Recovery Boosting*. The difference is especially evident for `lucas`, whose IQ entries spent the least amount of time in

the *Invalid* state (5% of the time) as shown in Figure 4.21. However, when the *Invalid* periods are even slightly longer, the difference between the two schemes becomes less evident and it starts becoming more beneficial to reduce V_t than to try maintaining a small threshold voltage difference for longer *Invalid* periods. For example, we can see in Figure 4.21 that `sixtrack` and `applu` spend successively longer periods in the *Invalid* state (13% and 21% respectively) and consequently the gap between *Balancing* and *Recovery Boosting* diminishes and *Recovery Boosting* provides more benefit than *Balancing* for workloads with longer *Invalid* times.

In general, we find that *Recovery Boosting* is more beneficial than *Balancing* for a wide range of IQ entry *Invalid*-state occupancy times. *Recovery Boosting* provides a 48% improvement in the SNM over *Baseline* whereas *Balancing* provides only a 37% improvement for the entire SPEC CPU2000 benchmark suite.

4.5 Summary

SRAM memory cells are especially vulnerable to NBTI since the input to one of the PMOS devices in the cell is always at a logic '0'. In this chapter, we propose recovery boosting, a technique that allows both PMOS devices in the cell to be put into the recovery mode by raising the ground voltage and the bitline to V_{dd}. We show how fine-grained recovery boosting can be used to design the physical register file and issue queue and evaluate their designs via SPICE-level simulations. We then show that area-neutral designs of these two structures can provide significant reliability benefits with very little impact on power consumption and negligible loss in performance.

This chapter covers work published in ISVLSI 2010 [2] and in the IEEE Transactions on VLSI 2011 [3].

Chapter 5

Mitigating the Impact of NBTI on Processor Functional Units

NBTI affects both the cycle time and the stability of storage structures within the processor. These problems are typically addressed via guardbanding. Guardbanding accounts for the degradation in cycle time and the stability of the storage structures over the lifetime by reducing the operating frequency and increasing the minimum voltage of the storage elements (V_{min}). Typically, 20% of the cycle time is reserved as a guardband for NBTI and a 10% increase in threshold voltage (V_t) can be handled with a 10% increase in V_{min} [17]. However, reducing the frequency and increasing V_{min} have a detrimental impact on performance and power respectively and therefore it is desirable to reduce the guardband via the use of NBTI mitigation techniques. In this chapter, we look into guardband reduction techniques to address the degradation in cycle time.

Techniques for putting PMOS devices into the recovery mode can be implemented at both the circuit and microarchitecture levels in the processor. The goal of circuit-

level techniques is to design the structures such that as many PMOS devices as possible in the structure can be put into the recovery mode whenever possible. Circuit-level techniques typically attempt to tackle NBTI at the granularity of a single structure. Microarchitecture level techniques, on the other hand, can manage NBTI for several structures within the processor core using techniques such as instruction fetch and scheduling policies. There are tradeoffs in implementing NBTI recovery at each of these levels in the system in terms of area, power, performance, complexity, and, their effectiveness in reducing the guardband.

In this work, we present a quantitative analysis of NBTI recovery techniques at the circuit and microarchitecture levels for the functional units (FUs) in the cores of a high-performance multicore processor. We choose to study FUs because of the general trend in multicore processor design, where more cores are integrated onto the die each sucessive generation but the cores themselves tend to be relatively simple and have only a small number of FUs. In this scenario, the failure of even one FU could seriously jeopardize the ability of that core to provide high performance. We characterize the effectiveness of NBTI mitigation at both the circuit and microarchitecture levels and quantify their impact on other important figures of merit, such as, area, delay, and application performance. The objective of this chracterization study is to identify those techniques that can effectively reduce the guardband but have the least amount of performance impact on applications, as well as impose minimal overheads in terms of area, power, and delay. We then show that lightweight optimizations at each level is more effective in reducing the guardband without adversely affecting the other figures of merit than applying more extensive changes at any one level. To the best of our knowledge, this is the first study to systematically analyze NBTI recovery techniques at each

of these levels for FUs and develop multi-level optimization techniques to tackle this important reliability problem. In this chapter, we propose and evaluate three different NBTI-aware FU designs in terms of guardband reduction, area and delay. We show that, on an average, the three different designs provide a guardband reduction of 42%, 46% and 47% over the baseline configuration. We then propose three different NBTI-aware scheduling policies and evaluate their impact on performance and guardband reduction. We show that, on average, the three policies provide a guardband reduction of 43%, 54% and 63% over the baseline FU configuration. Finally, we analyze the effectiveness of combining lightweight optimizations at each level and show that this approach provides a 56% guardband reduction with minimal impact on other figures of merit.

The outline of the rest of this chapter is as follows. The next section discusses the NBTI mitigation techniques we consider. The experimental methodology is described in Section 5.2. The results are presented in Section 5.3 and Section 5.4 concludes this chapter.

5.1 Approaches to NBTI Mitgation at the Circuit and Microarchitecture Levels

In this section, we describe how degradation due to NBTI can be reduced at the circuit and microarchitecture levels for the functional units (FUs) and describe the specific designs and policies we evaluate. While our circuit designs are optimized versions of a previously proposed technique [27], the microarchitecture level schemes we propose

are novel.

5.1.1 Circuit Level Techniques

Fu et al. [27] proposed a circuit-level technique where a FU is divided into multiple segments and instructions with narrow-width operands are steered into one of the segments based on the delay of each segment due to NBTI. The delay of a segment depends on the level of stress experienced by the PMOS transistors in that segment. Higher the stress, more would be the increase in V_t and hence higher would be its delay. The specific design considered in [27] is a 64-bit FU that is divided into four segments of 16 bits each. An instruction with 16-bit or smaller operands uses the segment with the smallest delay while the other segments are put into the recovery mode. Instructions with operand-widths greater than 16 bits make use of the entire FU.

There are two drawbacks to the design proposed by Fu et al. First, each segment in the FU is built as a Carry Lookahead Adder (CLA) and the segments are connected as a multi-level CLA to form a 64-bit FU. CLAs are seldom used in high performance microprocessors due to their low speed. Instead, most processors today use some form of a high-speed prefix adder. Second, to put PMOS devices in the idle segments into the recovery mode, they feed in special input vectors. Since an FU is a complex combinational circuit, a single vector of bits input to the FU cannot put all the PMOS devices in it into the recovery mode and instead a sequence of input vectors are necessary, which takes multiple clock cycles. Therefore, the PMOS devices cannot utilize the entire duration of an idle period to recover from NBTI. We address both these limitations as follows.

In this chapter, we model the FU to be a 64-bit Kogge-Stone adder (KSA) which is a high-speed prefix adder [43] for the purpose of estimating NBTI. KSA is a widely used FU design due to its regular structure and its speed. In order to put all the PMOS devices in the FU into the recovery mode during an idle period, we make use of power gating [44]. Power gating reduces leakage power in the circuit by using a header or footer device which connects the entire circuit to V_{dd} or ground. Whenever the circuit is idle, turning off this transistor disconnects the circuit from V_{dd} or ground. In our model, we use footer devices to connect the circuit to ground. During power gating, we find that the gate voltages of the PMOS devices in the FU stay reasonably close to a logic value of '1', thereby putting the FU into the recovery mode.

We design the KSA such that it consists of segments that are capable of processing narrow-width operands while the idle segments undergo recovery using power gating. We profiled the SPEC CPU2000 benchmark suite [30] to determine the distribution of narrow-width operands. We find that there are a large number of instructions with 8-bit, 16-bit, and 32-bit operands. Therefore, there is significant scope for NBTI recovery by partitioning the 64-bit FU into 2, 4, or 8 segments, where each segment is 32 bits, 16 bits, or 8 bits respectively. Each such segment can operate as an independent FU on operands of the given width and multiple such segments can be combined to operate on wider operands. For example, if the FU is partitioned into 4 parts, an instruction with two 16-bit input operands needs to utilize only one part while the other three can be put into the recovery mode. An instruction with 32-bit operands can use either the first two or last two consecutive segments while the other two are put into the recovery mode. Figure 5.1 shows an FU that has two segments. To ensure that all parts of the FU experience roughly an equal amount of wear, we schedule instructions to the segments

in a round-robin fashion.

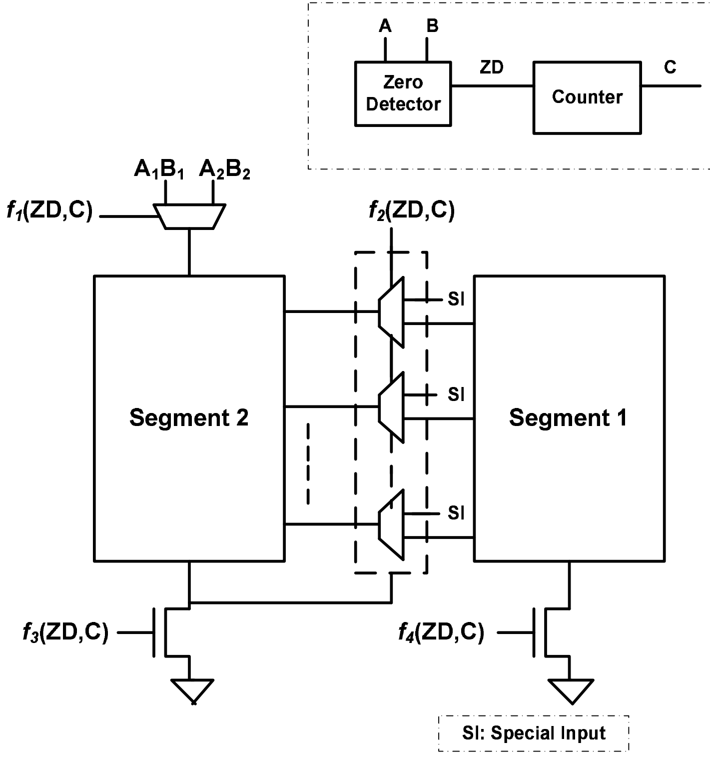


Figure 5.1: Partitioned Kogge-Stone Adder Design to Support NBTI Recovery.

It is important to note that the KSA is fast because the carries are computed in parallel, a feature that we retain in the partitioned design. The delay of the FU, when operating in the 64-bit mode, should not be significantly affected by the partitioned nature of the design. In order to achieve this property, we introduce a set of MUXes between adjacent segments, as shown in Figure 5.1. The selection input to these MUXes depend on the width at which the FU will be used. For example, in the case of Figure 5.1, the input to the MUXes will determine whether the FU will be used as a 64-bit FU or a 32-bit FU. Each segment is connected to ground via a footer device. The gates of the footer devices are controlled by zero detectors and a one-bit counter output. The

purpose of the zero detectors is to decide whether the instruction has narrow-width operands or not. The one-bit counter is incremented every time a narrow-width operation is performed. If the operands are narrow-width, they are steered towards a particular segment using a round-robin policy based on the value of the counter. The footer for the left segment is shared with the MUXes since the MUXes provide the input to the left segment.

There are tradeoffs in designing the FU in a partitioned manner to support recovery. The more segments that a FU has, greater are the opportunities for recovering the PMOS devices and reducing the guardband, since there will be a better matching between narrow-width operands and the number of segments required to operate on them. However, such a design comes at the cost of increased area, delay, and higher power consumption when it is used for operating on wide operands. We analyze these tradeoffs in Section 5.3.

5.1.2 Microarchitecture Level Techniques

There are opportunities to reduce the guardband by carefully managing the hardware resources within the core at the microarchitecture level. The usage characteristics of different FUs can be controlled through various instruction scheduling policies. The dynamic instruction scheduler decides which instructions are executed and at what times on a given set of FUs and therefore has a strong impact on the utilization characteristics of the FU. The scheduler consists of two key components: wakeup logic and select logic. The wakeup logic is responsible for asserting an instruction as being 'ready' in the issue window by updating the source dependences of instructions waiting

for their source operands to become available. Every time a result is produced by a FU, the tag of the result is broadcast to the waiting instructions in the issue window. Each waiting instruction compares the result tag with the tag of each of its source operands. Once both operand tags have matched, the instruction is ready to execute (instruction wakeup) and the ready instructions signal the select logic to request execution on a given type of FU. Once a FU becomes available, the select logic directs a suitable instruction to that unit for execution by asserting the corresponding grant signal (instruction select). Since multiple instructions could wakeup in a given cycle and the processor typically has multiple FUs of the same type, there needs to be a policy to select a subset of the ready instructions and assign them to specific FUs based on resource availability. Modern instruction schedulers typically use a form of prioritized scheduling where instructions are selected in an oldest-first order from the issue window and each instruction is issued to the lowest-numbered FU that is free. We call this policy **Prioritized Scheduling** (`PS`). In this approach, an instruction will be allocated to FU_0 if it is available; if FU_0 is busy, then FU_1 will be checked for availability and so on. This non-uniform assignment leads to the case where FUs with smaller sequence numbers get utilized more than those with higher sequence numbers and hence degrade faster. This is illustrated in Figure 5.2, which presents the utilization of the integer ALUs (in terms of the number of cycles that the FU is busy over the entire execution time of the workload) of a 4-wide issue processor core for a set of SPEC CPU2000 benchmarks. As we can observe from the figure, the lower-numbered functional units tend to be heavily utilized and therefore will wear out sooner than the higher-numbered ones. We propose instruction scheduling policies that attempt to extend the recovery times for the FUs so that their degradation due to NBTI can be minimized. We make

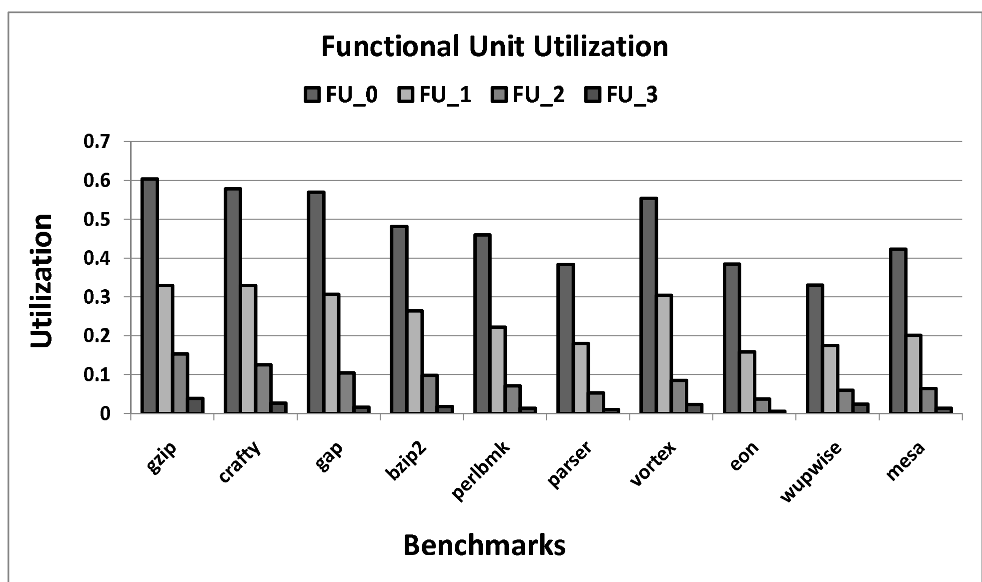


Figure 5.2: Utilization of the Integer ALUs in a 4-wide issue processor core using the PS policy.

use of power gating to put idle FUs into the recovery mode. We now present a brief overview of the instruction scheduling policies that we evaluate.

i) Priority Rotation Scheduling (PR): The PR policy is geared towards achieving a balanced utilization of the FUs in order to level the wear on them. This policy modifies the conventional PS scheduling policy so that the priorities of the FUs are changed, in a round-robin fashion, after a fixed number of cycles ($\texttt{Cycle}_{PR}$) have elapsed. In the PR policy, we start with FU_0 having the highest priority and assign lower priorities to the other FUs based on their sequence numbers (i.e., FU_0 initially has the highest priority whereas FU_{n-1} has the lowest). After $\texttt{Cycle}_{PR}$ cycles, FU_1 gets the highest priority, FU_2 gets the second highest priority and so on and FU_0 has the lowest priority. The select logic is similar to the PS policy but with added functionality to change the priorities of the FUs after $\texttt{Cycle}_{PR}$ cycles. A key advantage of the PR policy is that

it does not degrade performance in the sense that no FU is precluded from being used because it is in the recovery mode.

ii) Time-Dependent Scheduling (`TD`): Time-Dependent Scheduling extends the conventional policy to include an explicit fixed recovery period. In the `TD` policy, whenever the wakeup logic flags an instruction to be ready, the select logic allocates the lowest numbered FU that is available. After a FU is used, no other instruction is assigned to that particular FU for a fixed number of cycles ($\texttt{Cycle}_{TD}$). This policy allows the FU to undergo recovery for $\texttt{Cycle}_{TD}$ cycles after each stress phase. We can implement this in hardware by keeping the busy signal of the FU asserted for $\texttt{Cycle}_{TD}$ cycles after the FU is used. However, since the FU cannot be used during this time, there could be a detrimental impact on performance.

iii) Prioritized Time-Dependent Scheduling (`PTD`): Prioritized Time-Dependent Scheduling combines the `PR` and `TD` policies to include an explicit fixed recovery period to the highest priority FU. In this scheme, similar to `PR` policy, the priorities of the FUs are changed in a round-robin fashion after a fixed number of cycles ($\texttt{Cycle}_{PRC}$) have elapsed. Whenever an FU gains the highest priority, it also gains the priviledge of an extended recovery period after it is used. When this high priority FU is used, no other instruction is assigned to this particular FU for another fixed number of cycles ($\texttt{Cycle}_{TDC}$), allowing it to undergo recovery after each stress phase. Other FUs can be continued to be used as usual. Once $\texttt{Cycle}_{PRC}$ cycles have elapsed, the priorities are rotated. This transition in priorities does not affect the residual time for which an FU can remain in the recovery mode. Any FU that enters the recovery mode is guaranteed to be in that mode for $\texttt{Cycle}_{TDC}$ cycles. Since the `PTD` policy prevents an FU that is in the recovery mode from being used till the requisite number of cycles

elapse, this policy could also have a detrimental impact on performance. However, if $\mathtt{Cycle}_{PRC}$ is chosen to be significantly larger than $\mathtt{Cycle}_{TDC}$, only one FU would tend to be in the recovery mode at any one time and therefore the degradation in performance would tend to be less severe than the `TD` policy. We evaluate these policies in Section 5.3.

5.2 Experimental Setup

Our circuit-level modeling is performed via SPICE-level simulation using the Cadence Virtuoso Spectre circuit simulator [28] for the 32nm process using the Predictive Technology Model [31]. Our architecture-level evaluations are carried out via execution-driven simulation using the M5 simulator [29]. We simulate a 4-wide issue core, which is representative of cores used in multicore processors today, that runs at a 3 GHz clock frequency and has a supply voltage of 0.9V. We use all 26 benchmarks from the SPEC CPU2000 benchmark suite in our evaluations [30]. The benchmarks are compiled for the Alpha ISA and use the reference input set. We perform detailed simulation of the first 100-million instruction SimPoint for each benchmark [41].

We focus on the impact of NBTI on the integer ALUs only. Although we consider both integer and floating-point benchmarks in our evaluations, several of the floating-point benchmarks have a considerable number of integer instructions in their instruction mix and therefore make heavy use of the the integer ALUs [45]. We present our results for only the integer ALU with the lowest sequence number since this ALU tends to be the most heavily utilized of all the ALUs with conventional instruction scheduling, as explained in Section 5.1.2. Since NBTI affects the threshold voltages of PMOS

devices in the FUs, the delay of the FU hardware increases which causes a potential danger to meet the timing constraints. Guardbanding is used to protect a circuit from failure, which entails both performance and power penalties. In all our experiments, we assume that the baseline processor uses the unpartitioned FU design, the `PS` instruction scheduling policy, and a guardband of 20% [17].

In our evaluations, we use the percentage reduction in the guardband with respect to the baseline as the figure of merit to quantify the extent to which a particular design mitigates the impact of NBTI. To compute guardband reduction, we track the stress and recovery cycles of the FUs in the architectural simulations. Using these statistics, we estimate the degradation in V_t after a 7-year service life [15], using which we calculate the delay degradation in the structures and finally the guardband reduction. We also quantify the impact on other important figures of merit, such as, area, delay, and application performance (in terms of IPC), to study the effectiveness of NBTI mitigation at both the circuit and microarchitecture levels.

Our goal is to evaluate the extent to which optimizations at only the circuit or microarchitecture level can reduce the guardband and ascertain the costs incurred in applying these optimizations on the other figures of merit. These results are presented in Sections 5.3.1 and 5.3.2. Based on this analysis, in Section 5.3.3, we study whether less aggressive optimizations at each level, which impose less overheads, can be combined to provide an effective means to reduce the guardband.

5.3 Results

5.3.1 Circuit-Level Optimization

We now study the tradeoffs between different circuit-level techniques. We partition the FU into 2, 4 and 8 segments and analyze the guardband reductions, area overheads, and the increases in delay with respect to the baseline.

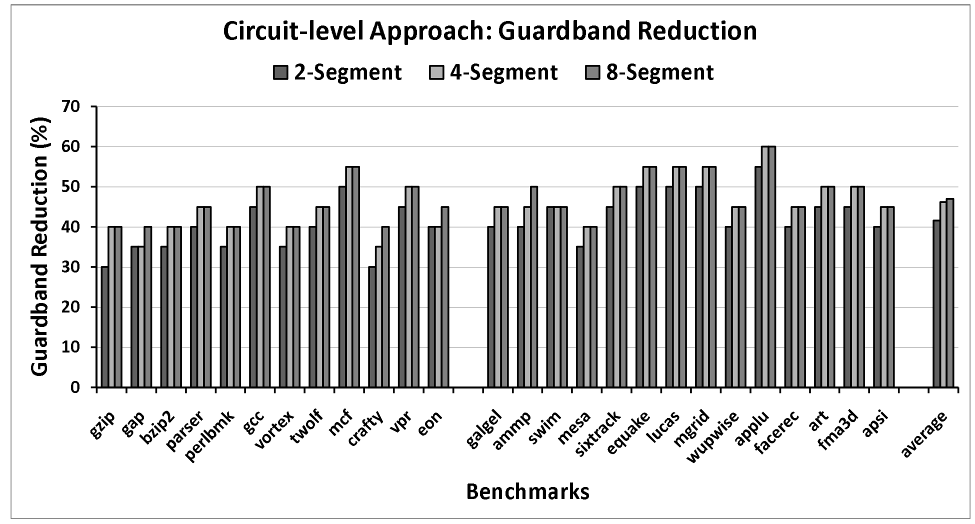


Figure 5.3: Guardband reduction for the FUs with 2, 4 and 8 segments.

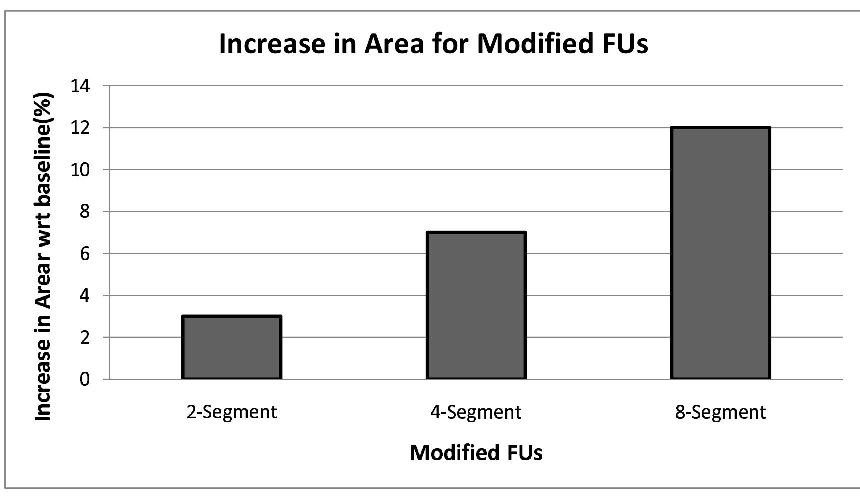
Guardband Reduction: Figure 5.3 presents the guardband reduction (as a percentage) due to the modified FU designs. The FU with 2 segments provides a guardband reduction of 30%-55% whereas a 4-Segment FU and a 8-Segment FU provide 35%-60% and 40%-60% guardband reductions respectively. This result shows that the more partitions that a FU has, the higher would be the guardband reduction. However, we see that the guardband reduction achieved by going from the baseline to the 2-segment design is much higher than going from the 2- to 4-segments and 4- to 8-segments. Also,

the integer benchmarks, which are the left-hand side group of bars in the figure 5.3, experience less guardband reductions than the floating-point benchmarks.

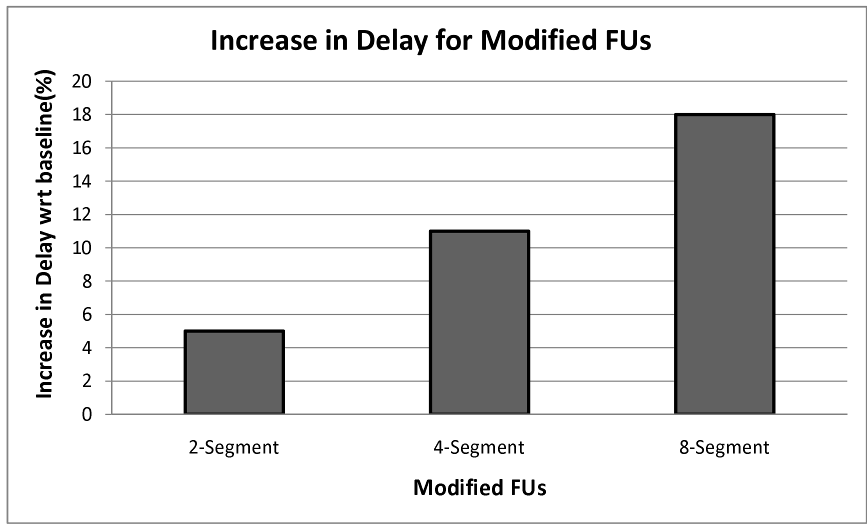
In order to understand how the modified FU design impacts NBTI, we need to analyze how the FUs are utilized and the distributions of the narrow-width operands. The instruction mix gives an indication of how frequently each type of FU gets accessed. For example, the higher the number of integer instructions, the higher is the probability of accessing integer FUs. A previous study by Siddiqua et al. [45] gives the breakdown of the instruction mix of these benchmarks. We find that the lowest guardband reductions are observed for those benchmarks which have a high percentage of integer instructions. Similarly, to understand the benefit of the segmented designs, it is important to look at the distribution of the narrow-width operands. We find that, on average, the percentage of 32-bit operands is about 48% whereas it is 8% for 16-bit operands and 27% for 8-bit operands across the benchmark suite. Since the 32-bit operand sizes occur most frequently, we get a higher guardband reduction when we go in for the 2-segment partition from the baseline, whereas the 4-segment to 8-segment designs provide diminishing returns.

Area and Power: We design the FU for the baseline case to occupy the minimum area required to provide correct functionality. The overheads for the partitioned FU designs are given in Figure 5.4 (a). We can see that the area overhead of the 2-segment, 4-segment and 8-segment FUs are 3%, 7% and 12% respectively. The increase in area is due to the number of sets of MUXes required for the three designs (1, 3 and 7 respectively). These MUXes increase the power consumption of the FU and therefore the designs with more segments will consume more power.

Delay: We evaluate the delay to measure the highest clock frequency at which the



(a) Increase in Area (%)



(b) Increase in Delay (%)

Figure 5.4: Percentage increase in area and delay for the FUs with 2, 4 and 8 segments wrt. the unpartitioned FU design.

FU can operate reliably. Figure 5.4 (b) shows the increase in delay with respect to the baseline design. In our simulations, we find that the increase in delay for the 2-segment, 4-segment and 8-segment FU designs due to the additional hardwares are 5%, 11% and 18% respectively. However, these increases in delay can be accomodated

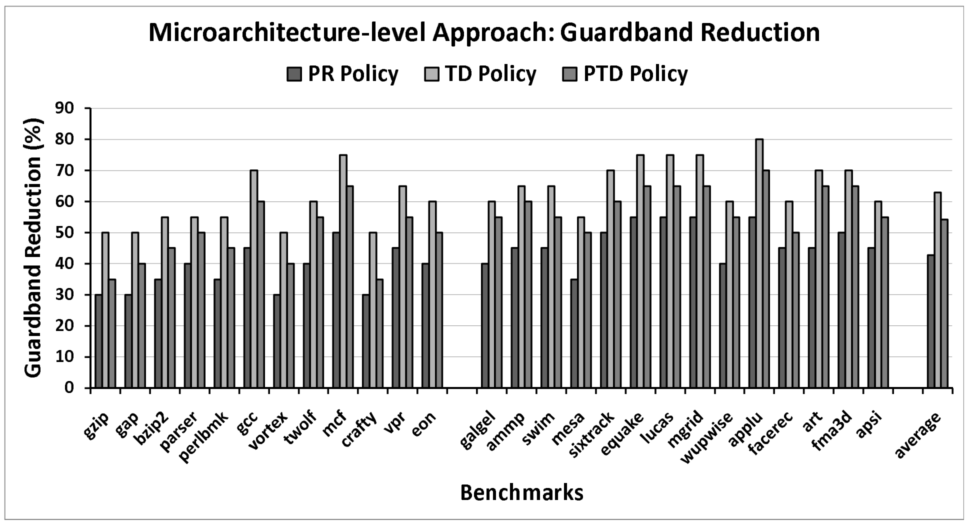
within a 333ps clock cycle time (which corresponds to the 3 GHz clock frequency) and therefore all these FU designs can provide a single-cycle access latency for instructions using the FUs.

Summary: There are merits and demerits to using the aforementioned FU designs. The higher the segment count in the FU, the higher is the guardband reduction. However, the area, power, and delay overheads also increase significantly. Each segment introduces a set of extra circuitry which adds area and increases power consumption. For 64-bit operand computations, the FU would consume the highest power since all the MUXes will be active and this power consumption will be higher with more partitions. Overall, we find that we get higher guardband reduction with least area, power and delay overheads for the 2-segment design. Therefore, we choose the 2-segment FU design for use in the multi-level approach in Section 5.3.3

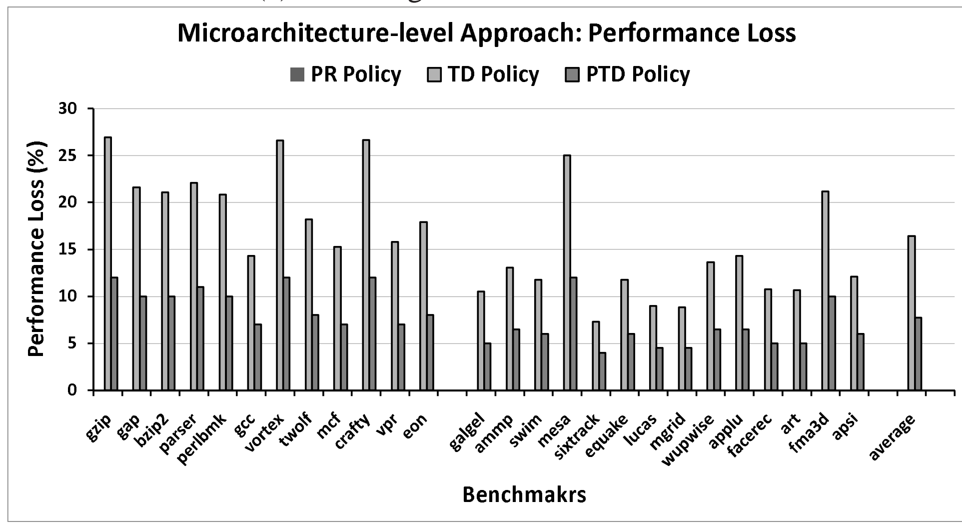
5.3.2 Microarchitecture-Level Optimization

We evaluate the guardband reduction and performance of the `PR`, `TD` and `PTD` policies. We use a value of 10K for $\texttt{Cycle}_{PR}$ and $\texttt{Cycle}_{PRC}$ and a value of 1 for $\texttt{Cycle}_{TD}$ and $\texttt{Cycle}_{TDC}$ for the different policies.

Guardband Reduction and Performance: Figure 5.5(a) presents the percentage guardband reduction for the various policies. The `PR` policy reduces the guardband by 30%-55% whereas the `TD` and `PTD` policies reduce it by 50%-80% and 35%-70% respectively. Since the scheduling policies aim to achieve improved guardband reduction by increasing idleness to provide NBTI recovery, they may impact performance, as discussed in Section 5.1.2. Figure 5.5(b) quantifies the performance loss. As we can see,



(a) Percentage Guardband reduction



(b) Percentage loss in performance

Figure 5.5: Impact of the instruction scheduling policies.

the `TD` and `PTD` policies achieve a greater guardband reduction at the cost of reduced performance whereas the `PR` policy experiences no performance loss. On average, the `TD` and `PTD` policies lead to a performance loss of 17% and 8% respectively. The `PTD` policy provides a guardband reduction and experiences a performance loss that is be-

tween PR and TD because the value of $\texttt{Cycle}_{PRC}$ is much higher than $\texttt{Cycle}_{TDC}$. Therefore, the policies that attempt to increase the recovery time of a FU cause a corresponding loss in performance for the benchmarks. Similar to the circuit-level results, the integer benchmarks have less guardband reduction than the floating-point benchmarks.

The impact of scheduling policies on NBTI is influenced by two factors: the instruction mix and instruction level parallelism. As mentioned before, the instruction mix is an indicator of how frequently each type of FU gets accessed. Instruction level parallelism also determines the frequency of use of the FUs. Also, it affects how groups of FUs get used. When the IPC is high, more instructions are executed per cycle. Consequently, more FUs will get utilized. Indeed we find the least guardband reductions are observed for those benchmarks which have a high percentage of integer instructions and higher IPC.

Summary: As the results indicate, there is a clear tradeoff between achieving higher guardband reductions and application performance by using instruction scheduling. Although the TD policy provides large benefits in terms of guardband reduction, it imposes large performance overheads as well, which is unattractive for use in high-performance processors. Although the PTD policy reduces the performance overheads significantly while still providing significant guardband reductions, the detrimental impact on application performance is still non-trivial. The PR policy, on the other hand, does not cause any performance loss while still providing a guardband reduction of 43% on average. Therefore, although TD and PTD provide large guardband reductions, we choose PR, since it is the most lightweight policy in terms of its performance impact, and use it in the multi-level approach.

5.3.3 Multi-Level Optimization

In the results presented thus far, we have observed that increasing the effectiveness of NBTI mitigation techniques at only the circuit or microarchitecture level to improve the guardband entails penalties in terms of area, power, delay, and application performance. On the other hand, lightweight optimizations at *each* level could provide a net increase in guardband reduction with less overhead. We now evaluate this hypothesis.

We choose the 2-segment FU design, which we found to have the smallest overhead in terms of area, delay, and power in Section 5.3.1 as our circuit-level optimization and the PR instruction scheduling policy as our microarchitecture-level optimization. The result of combining these two optimizations is given in Figure 5.6.

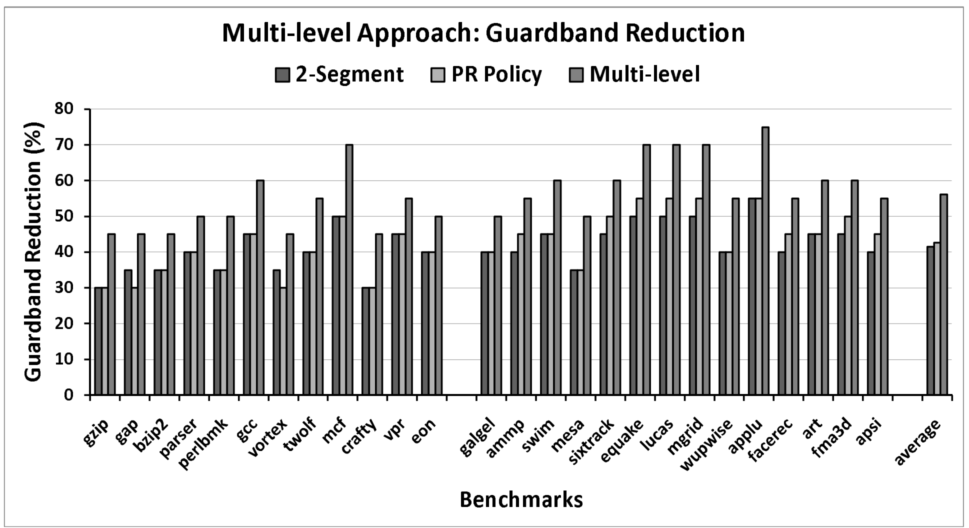


Figure 5.6: Guardband reduction using FUs with 2 segments and the PR scheduling policy.

From the figure, it is evident that we achieve a guardband reduction of 45%-75%, which is a much higher range than the 30%-55% for both circuit and microarchitecture levels. It is also important to note that the achieved guardband reduction due to

the multi-level approach is *not* merely additive from the individual optimizations at each level. This is due to the fact that, in the multi-level approach, the overall flow of bits through the FUs over the course of execution of the workload is different from the previous two sets of evaluations. The new FU design changes the stress and recovery characteristics on the PMOS devices in the FU, due to the partitioned nature of its design, compared to the unpartitioned FU design evaluated in Section 5.3.2 for the microarchitecture-only optimizations. Similarly, the `PR` policy alters the overall utilization of each FU and increases the idleness of all the FU segments, which allows for greater recovery than the `PS` policy used in Section 5.3.1 for the circuit-only optimizations.

Overall, we observe that the multi-level approach provides greater reductions in the NBTI guardband while retaining the low area, power, and delay benefits of the 2-segment FU design and the high performance of the `PR` policy.

5.4 Summary

In this chapter, we evaluate both circuit and microarchitecture level approaches to reduce the NBTI guardband for the FUs of a high-performance processor core. At the circuit-level, we use an optimized version of a partitioned FU design and evaluate several design points in terms of their effectiveness in reducing the guardband and also their area, delay, and power. At the microarchitecture-level, we propose and evaluate a set of NBTI-aware dynamic instruction scheduling policies and evaluate their impact in terms of guardband reduction and performance. Finally, we show that a multi-level optimization approach, which combines the benefits of both circuit and microarchitec-

ture level optimizations, is the most effective in reducing the guardband while imposing little overhead in terms of area, power, delay, and performance.

This chapter covers work published in GLSVLSI 2010 [4].

Chapter 6

Modeling and Analyzing NBTI in the Presence of PV

Process Variation (PV) is the variation in the transistor attributes (length, width, oxide thickness) caused during the fabrication of the integrated circuits and manifests itself as threshold voltage variations which results in variability in circuit performance and power. The impact of NBTI is exacerbated by PV. Processors have to be designed to provide adequate protection against both these problems. Both NBTI and PV have received attention in the architecture community in recent years and several mitigation techniques have been proposed for each [17, 15, 4, 2, 21, 22]. Since both NBTI and PV affect the threshold voltage of devices, these two problems should not be addressed in isolation. To come up with the appropriate mitigation techniques, it is important to accurately gauge the impact of both NBTI and PV and factor-in the impact of the workloads that run on the processor as well. For this purpose, an analytical model is required which captures the impact of both NBTI and PV in a coherent way and which

is suitable for use in architecture level analyses.

There have been several efforts in developing analytical models for NBTI and PV at the circuit-level. However, these models are suitable only for analyzing NBTI and PV effects over a very short time span and are not readily usable for architecture simulations. Architects, on the other hand, study microprocessor reliability by executing different program benchmarks and extrapolate the collected statistics over a much longer timescale (typically, 7-10 years). Throughout the benchmark execution, utilizations of the microarchitectural structures vary. Also, the interactions among the structures, the inputs to each structure, and bits stored within them change over the course of execution of a benchmark. The analytical model for NBTI and PV should be able to factor-in all these "variations" to be usable in architecture simulations to gain correct and holistic insight into these inter-related reliability problems in silicon. In this chapter, we leverage the prior research on NBTI and PV modeling from the circuits community to develop a model that captures the interactions between these two reliability phenomena and which is usable at the architecture-level.

There are different sources of variation inherent in NBTI and PV that affect the PMOS threshold voltage. One source of variation in the threshold voltage due to NBTI is *workload variation* which is caused by executing different workloads on the processor. This variation is due to changing patterns of utilization of the microarchitectural structures and changes in the bit patterns within the structures. Another factor lies in the silicon process, known as the Random Charge Fluctuation (RCF), which causes a *temporal variation* in threshold voltage on top of the workload variation. Along with the variations due to NBTI, each device also has Random Dopant Fluctuations (RDF) due to *process variation* (details of the sources of these variations are discussed in the

next section). The analytical model we have developed accounts for all these variations. In this chapter, we develop an analytical model to capture both NBTI and PV for use in architecture simulations. We use this model to analyze the combined impact of NBTI and PV on a memory structure (register file) and a logic structure (Kogge-Stone adder). We show that the impact of the threshold voltage variations due to NBTI and PV over the nominal degradation can hurt the yield of the structures. Due to the combined effect of NBTI and PV across different benchmarks, 26 to 117 bits fail in a 8Kb size register file and the execution delay increases by 18% to 28% in a kogge-stone adder. We then discuss the implications of these results for architecture-level reliability techniques.

The outline of the rest of this chapter is as follows. The next section gives a brief overview of the different sources of threshold voltage variation due to NBTI and PV. The analytical model for NBTI and PV is described in Section 6.2. The experimental methodology is described in Section 6.3. The results are presented in Section 6.4 and Section 6.5 concludes this chapter.

6.1 Overview of NBTI and PV

Figure 6.1 shows the overall picture of the different sources of variation in PMOS threshold voltage degradation due to NBTI and PV. We now describe how NBTI gets affected by workloads that run on the processor and the silicon process.

As shown in Figure 6.1, the impact of NBTI is affected by several factors. In a real processor, different microarchitctural structures exhibit different utilization patterns based on the characteristics of the workloads that exercise them. On top of the overall utilization of the structures, all the PMOS devices within each processor structure are

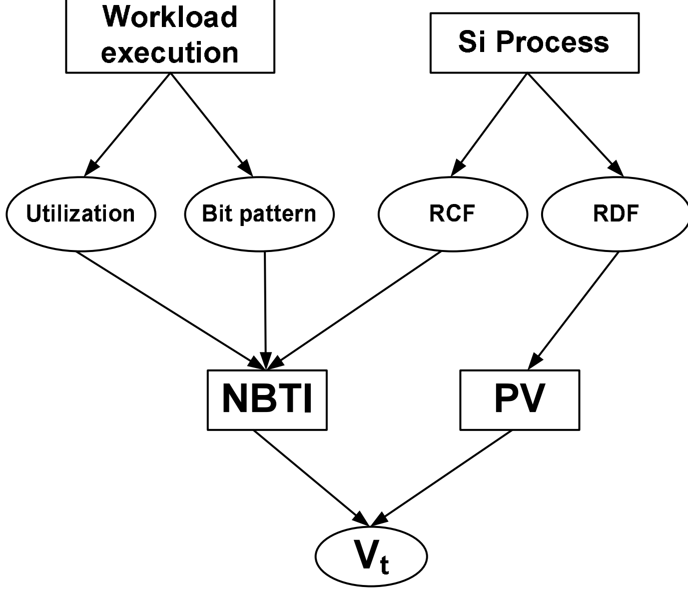


Figure 6.1: Different sources of V_t variation in PMOS devices.

stressed in different ways throughout the workload execution due to the varying data bit patterns (gate inputs of the devices) within them. Therefore, workload execution leads to a variation in the threshold voltage degradation, which we call *workload variation*. The third factor lies in the silicon process, known as Random Charge Fluctuation (RCF), which causes a *temporal variation* on top of the workload variation. Recent observations on PMOS devices with small gate areas show that the threshold voltage degradation is a subject to random fluctuations [24, 46]. These fluctuations increase as a function of stress time. The source of this behavior is the formation of a random number of trapped charges, which can occur at random locations across the gate. Such random fluctuations of trapped charges result in a variation in the threshold voltage degradation and needs to be considered when studying NBTI. We call the impact of NBTI which considers only the structure utilization and does not capture the effect of

the workload variation and temporal variation as *static NBTI*.

Furthermore, the degradation in processor lifetime due to NBTI is exacerbated by Process Variation (PV). Process variations can be broadly categorized into two groups: inter-die and intra-die variations [47]. Due to inter-die variations, the same device on a die can have different characteristics across various dies, whereas, due to intra-die variations, transistors can have different characteristics within a single die. There are two more subcategories of intra-die variation: systematic and random variations. Due to systematic variations, transistors close to each other are expected to have relatively similar parameters (channel length and oxide thickness) when compared to those farther away on the die. On the other hand, random variation is mostly caused by RDF. Due to RDF, transistors can have mutually independent V_t variation with respect to each other, regardless of their spatial location. We consider only the effect of RDF in this work, for two reasons. First, RDF is expected to be the major contributor to transistor threshold voltage variations in the sub-65nm technology [47]. Second, we look at individual processor microarchitectural structures where the devices within them are spatially proximate. The analytical model we develop accounts for the combined effect of workload and temporal variation due to NBTI in the presence of RDF.

6.2 An analytical model for NBTI and PV

As mentioned in Chapter 3, there have been several efforts in developing an analytical model for NBTI based on the reaction-diffusion model [5, 24]. These models have been extended to address dynamic temperature and voltage variations in [48, 16] and are suitable for use in circuit-level simulations. However, these models cannot be

directly used for architecture-level simulations. This is because these models assume continuous stress on the PMOS devices in a circuit and do not capture scenarios where there are multiple sequences of varying stress/recovery times, which is the case when real workloads run on the processor. We present a compact analytical model that is suitable for both circuit and architecture simulations and also takes into account the effect of PV. In order to consider the effect of PV, we use the analytical model for NBTI developed in Chapter 3 as our baseline.

6.2.1 Capturing the impact of Workload Variation, Temporal Variation, and PV

The NBTI V_t model (3.7 and 3.8) presented in Chapter 3 assumes the nominal or static degradation for each device without considering the workload variation or temporal variation. As described in the introduction, in a realistic scenario, the nominal NBTI for each structure is impacted by the workload execution due to the variation in the utilization of the structure and its bit patterns. While executing a workload, for a given structure, we track the stress/recovery patterns for each device within that structure. Using the model presented in the previous section, we get a V_t distribution (Standard Deviation = σ_{ARCH}). This results in multiple groups of devices where all the devices within each group experience similar stress/recovery patterns and have similar final V_t values.

Moreover, as mentioned in the introduction, the temporal variation in the underlying degradation process due to RCF causes additional variation on top of the workload variation. From [49], if a group of devices are stressed in a similar way, the variation

caused by RCF is:

$$\sigma_{RCF} = \sqrt{\frac{K.t_{ox}.\Delta V_{tf}}{A_g}}$$

where, σ_{RCF} is the standard deviation of the V_t distribution, A_g is the gate area of the device, t_{ox} is the oxide thickness, ΔV_t is the nominal degradation due to NBTI and K is a constant. Since workload variation results in multiple groups of devices experiencing similar kinds of stress patterns, temporal variation within each group of devices results in several V_t distributions. After combining all the distributions, we get a final V_t distribution which captures the effect of both workload and temporal variation (Standard Deviation = $\sigma_{(ARCH+RCF)}$).

Furthermore, to combine the effect of PV, we know from [49]:

$$\sigma_{RDF} = \frac{\alpha}{\sqrt{A_g}}$$

where, σ_{RDF} is the standard deviation of the V_t distribution due to RDF, A_g is the gate area of the device, and α is a constant.

Finally, combining the effect of NBTI (static, workload and temporal variation) and PV, we get the following standard deviation:

$$\sigma_{(PV+NBTI)} = \sqrt{\sigma_{(ARCH+RCF)}{}^2 + \sigma_{RDF}{}^2} \tag{6.1}$$

This completes the model. From the equations 3.7 and 3.8, we get the mean V_t degradation and equation 6.1 gives the V_t standard deviation.

6.3 Experimental Setup

To carry out the architecture simulations, we use the M5 simulator [29]. We simulate a 4-wide issue core, which runs at a 3 GHz clock frequency and is representative of cores that is used in multicore processors today. We use the 32nm process with a supply voltage of 0.9V. We assume the initial threshold voltage of the PMOS devices to be 0.2 V and the service life of the processor to be 7 years [15]. Our workloads consist of benchmarks from the SPEC CPU2000 benchmark suite [30]. We present simulation results for 8 representative benchmarks - 4 integer and 4 floating-point. The benchmarks are compiled for the Alpha ISA and use the reference input set. We perform detailed simulation of the first 100-million instruction SimPoint for each benchmark [41]. Our circuit-level simulations are performed using the Cadence Virtuoso Spectre circuit simulator [28] taking the technology parameters of 32nm process from the Predictive Technology Model [31]. In this chapter, we focus on the impact of NBTI and PV on one memory structure - the register file (RF) and one logic structure - the Kogge-Stone Adder (KSA). The RF is a 128x64 size SRAM array made up of 6T bitcells and the KSA is implemented for 64-bit inputs.

RF Reliability Metric: NBTI and PV affect the read and write delays and the read Static Noise Margin (SNM) of the SRAM cells. Previous work [18] has shown that the SNM is the one that is most heavily affected by NBTI. Therefore we use SNM as the reliability metric for the RF.

KSA Reliability Metric: Since NBTI affects the threshold voltages of PMOS devices in the KSA, the delay of the KSA increases, which could potentially cause a timing violation. Therefore we use delay as the reliability metric for the KSA.

Before exercising the RF and the KSA with workloads, the SNM and the delay of the RF and KSA respectively are already degraded because of PV. We calculate this degraded SNM distribution and delay by using the Spectre circuit simulator. The SNM and delay degrades further after the structures get exercised by the workloads, due to NBTI. We capture this impact by tracking the stress and recovery cycles on all the PMOS devices in the RF and the KSA over the course of the architecture simulation and extrapolate the statistics to calculate the final degradation in V_t after the 7-year service life. We calculate the mean V_t and the different standard deviation values due to temporal, workload, and the combined variations for both the RF and the KSA. We then feed these values into the Spectre circuit simulator, and calculate the degraded SNM distributions of the RF and delays of the KSA.

6.4 Results

We now quantitatively analyze the effect of NBTI in the presence of PV in RF and KSA. We evaluate four different conditions: i) *RDF*: considering only the impact of RDF without the effect of NBTI, ii) *RCF+RDF*: considering the impact of NBTI only with the temporal variation on top of the RDF effect, iii) *ARCH+RDF*: considering the impact of NBTI only with the workload variation on top of the RDF effect, and finally, iv) *ARCH+RCF+RDF*: considering the impact of NBTI with both the temporal and workload variation on top of the RDF effect.

6.4.1 RF Results

We now explain the impact of NBTI and PV on the RF by means of an example. We first show the V_t distributions under different conditions. From the simulations, we calculate the following standard deviations: (σ_{RDF}, $\sigma_{(RCF+RDF)}$, $\sigma_{(ARCH+RDF)}$ and $\sigma_{(PV+NBTI)}$). Figure 6.2 shows the V_t distributions of the RF for one of the benchmarks we evaluate - mcf. Initially, before the workload is executed, the V_t distribution is due to RDF (the leftmost distribution in the figure). But once the workload is executed and the stress/recovery statistics on the RF are extrapolated to 7 yrs, the V_t distribution shifts to the right due to NBTI. As the figure indicates, the effect of temporal variation in the presence of RDF merely causes a shift in the mean of the distribution, but once the workload variation is factored in, the distribution widens. In order to understand why the width increases, we need to understand how the V_t of the PMOS devices get affected by workload and temporal variation. As mentioned in Section 6.2.1, workload variation results in multiple groups of devices which experience similar stress patterns, leading to similar V_t values. However, because of the temporal variation, each group of devices ends up in a V_t distribution. Therefore, when we take into account all the V_t values in the structure, we get a wider distribution. It is important to note that without considering the effect of RDF, the distributions due to NBTI with temporal, workload, and the combined variations would be much narrower. Hence it is important to consider the effect of NBTI in the presence of PV along with temporal and workload variation to avoid any significant error in the lifetime estimation of the structure. Now we show how the V_t distributions affect the yield of the RF, using the RDF as the baseline.

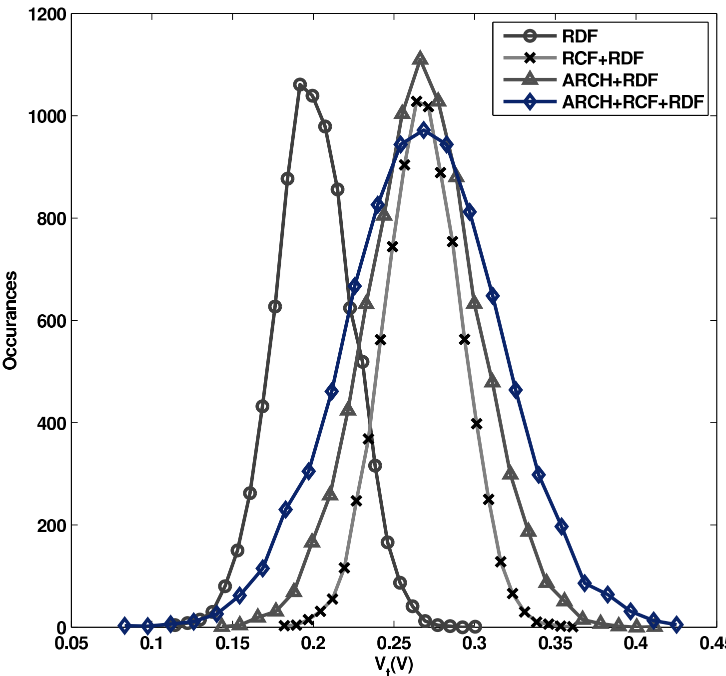


Figure 6.2: V_t distributions of the RF due to RDF, temporal, workload and combined variation for the mcf benchmark.

The required design coverage (N_σ) of a memory is a function of the target yield and the memory density and is expressed by the following equation [50]:

$$N_\sigma = \phi^{-1}(Y_{mem}{}^{\frac{1}{N_{bits}}})$$

where ϕ^{-1} is the inverse standard normal cumulative distribution, Y_{mem} is the yield of the memory, and N_{bits} is the total number of bitcells in the memory. Once the design coverage is calculated, from the expected SNM distribution (baseline: $\mu_{SNM-RDF}$, $\sigma_{SNM-RDF}$), the minimum allowed SNM can be calculated as:

$$SNM_{min} = \mu_{SNM-RDF} - N_\sigma * \sigma_{SNM-RDF}$$

Under each NBTI and PV condition, we count the number of bitcells whose SNM values are less than SNM_{min}. We denote this number as $\#bit_{fail}$.

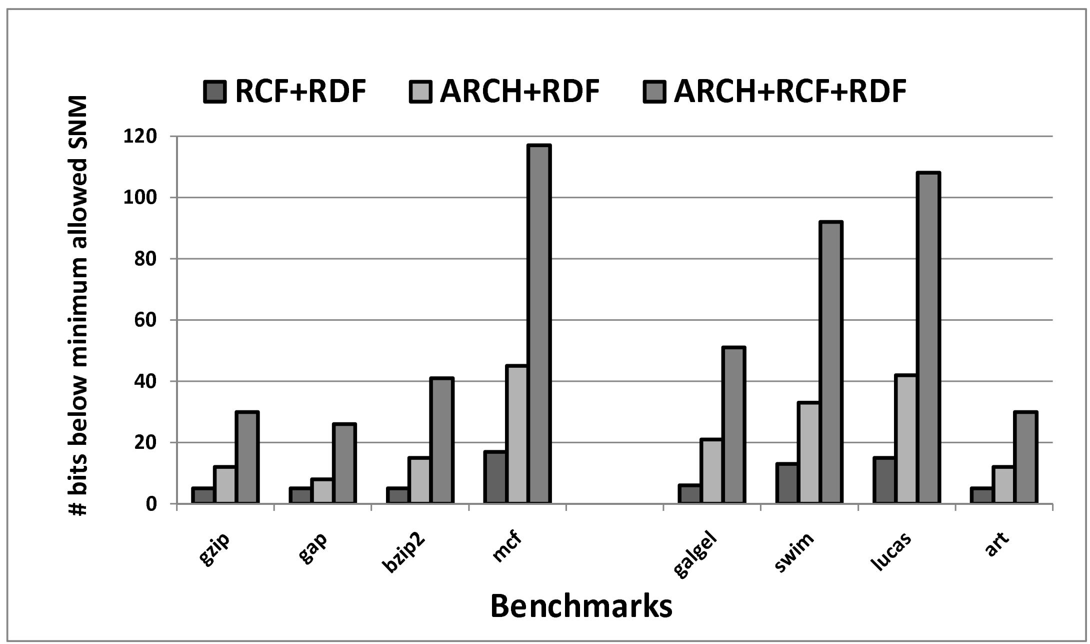


Figure 6.3: Number of bits experiencing SNM below the minimum allowed value in a RF due to temporal, workload and the combined variation for the different benchmarks.

Figure 6.3 shows the $\#bit_{fail}$ in the RF under three different conditions (*RCF+RDF*, *ARCH+RDF*, and *ARCH+RCF+RDF*) for different benchmarks. $\#bit_{fail}$ ranges from 5 to 17 for the *RCF+RDF* condition where only the temporal variation is considered in the presence of PV. It ranges from 8 to 45 for the *ARCH+RDF* condition, whereas it ranges from 26 to 117 for the *ARCH+RCF+RDF* condition. As expected from the V_t distributions, this result shows that the impact of the temporal variation alone is less than the impact of the workload variation, whereas the combined effect is much greater than the sum of the individual effects. This is due to the widening of the V_t distribution, as explained before. It is also important to note that the effects of the variations vary

significantly across the benchmarks. mcf, $lucas$ and $swim$ benchmarks have large $\#bit_{fail}$ values (117, 108 and 92 respectively) under the *ARCH+RCF+RDF* condition. The reason behind this is due to workload variations. mcf, $lucas$ and $swim$ experience much higher σ_{ARCH} as compared to the other benchmarks because of the bit patterns and the long residence times of the bits in each register. Generally, we find that most of the registers tend to have more 0's in the higher order bits and a random mix of 0's and 1's in the lower order bits, which contribute to the variability of the stress/recovery patterns of the register bits. Also, these benchmarks experience high L2 cache miss rate which causes stalls in the processor pipeline. Therefore, the contents of the register files do not get updated often. As a result, some bits tend to experience more stress whereas others experience less stress. Because of this, the bits in the RF experience high workload variation. The impact of workload variations, combined with temporal and process variations leads to a higher failure rate.

6.4.2 KSA Results

To explain the impact of NBTI and PV on the KSA, we again begin with the V_t distributions under different conditions. Figure 6.4 shows the V_t distributions of the KSA for mcf. The leftmost V_t distribution in the Figure is due to the RDF and this distribution gets shifted to the right because of NBTI. Similar to the RF, the effect of temporal variation and the workload variation in the presence of RDF is less than their combined impact. However, unlike the RF, the curves for the temporal variation and the workload variation are close to each other. The reason why the workload variation does not contribute to V_t changes significantly beyond the temporal variation is because of the

circuit design of the KSA. Based on the inputs to the KSA, bits propagate through the internal nodes of the circuit. The inherent design of the circuit generates internal node values of 0's and 1's within the structure in a balanced manner, which produces a comparatively smaller workload variation. Overall, the combined effect of NBTI and RDF is significant, similar to the RF. We now show the implication of the V_t distributions on the delay of the KSA.

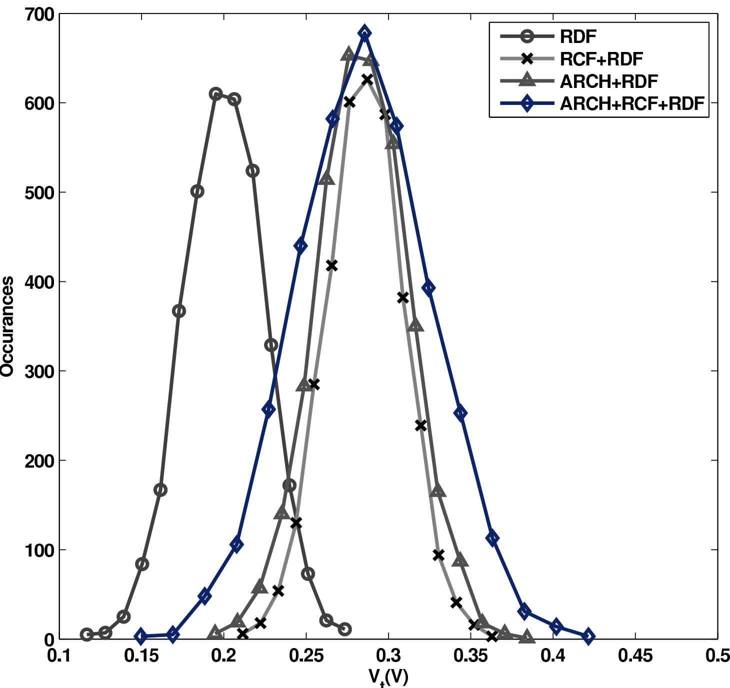


Figure 6.4: V_t distributions of the KSA due to RDF, temporal, workload and combined variation for the mcf benchmark.

As before, we use the *RDF* condition as our baseline. We calculate the percentage increase in delay with respect to the baseline for the other three conditions to analyze the impact of different variations due to NBTI.

Figure 6.5 shows the percentage increase in delay in the KSA with respect to the

baseline due to NBTI in the presence of PV for three different conditions for different benchmarks. The increase in delay ranges from 9% to 15% for the *RCF+RDF* condition, 11% to 20% for the *ARCH+RDF* condition, whereas it ranges from 18% to 28% for the *ARCH+RCF+RDF* condition. Just like the RF behavior, this result also shows that the impact of the temporal variation is less than the impact of the workload variation. Unlike RF, in this case the combined effect is not higher than the sum of the individual effects. This is because of the cancelling effect of the variations in the same timing paths of the logic structure. Each timing path of the structure consists of many PMOS devices which have different threshold voltages and the effect of the slower devices gets offset to some extent by the faster devices. Although, the combined effect of the workload and temporal variation causes an increase in the delay for each benchmark, this impact does not vary significantly across the benchmarks. Again, the reason behind this relates to the circuit design of the KSA which balances the values of 0's and 1's within the structure and reduces the impact of the variability in the utilization and bit patterns on the KSA across the different benchmarks.

6.4.3 Implications of the Results

- As the results indicate, both PV (RDF) and NBTI have a significant impact on V_t. More importantly, as Figures 6.2 and 6.4 show, if we consider only the impact of RDF or only Static NBTI (as is the case in a large number of architecture studies [17, 15, 4, 2, 21, 22]), then one does not get an accurate picture of the impact of these related reliability phenomena on the V_t distributions. For example, if only RDF is considered, then the shift in the mean of the V_t distribution due to NBTI is

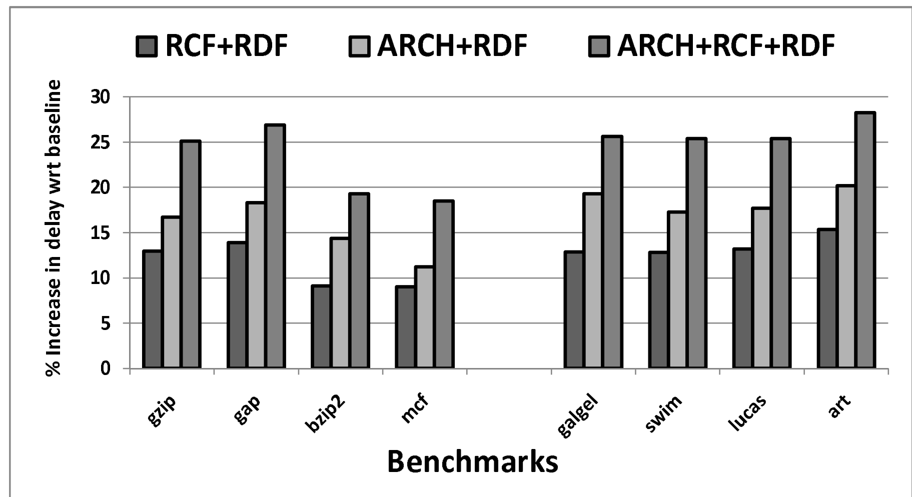


Figure 6.5: Percentage increase in delay in a KSA due to temporal, workload and the combined variation for different benchmarks.

not captured. Even within NBTI, unless both temporal and workload variations are accounted for, the widening of the V_t distribution will not be captured. It is important to capture these behaviors accurately in order to select appropriate guardbands and also develop effective mitigation techniques.

- While RDF and RCF depend on the underlying process, we can observe that the combined impact of RCF (temporal variation) and workload variation on lifetime reliability is significant. Since both temporal variation and workload variation strongly depend on the stress and recovery patterns on microarchitectural structures and also the bits that flow through them, there is large scope for NBTI mitigation at the architecture-level. However, it is important to develop and evaluate such mitigation techniques in way that is cognizant of the interaction between PV, temporal variation, and workload variation. The model that we have presented in Section 6.2 can be used to carry out such studies.

6.5 Summary

NBTI and PV are very important reliability problems in silicon facing processor designers. In this chapter, we develop an analytical model that captures both NBTI and PV for use in circuit and architecture simulations. We capture the following aspects in the model: i) variation in NBTI due to workloads, ii) temporal variation in NBTI and iii) process variation. We use this model to analyze the combined impact of NBTI and PV on a memory structure (register file) and a logic structure (Kogge-Stone adder). We show that the impact of the threshold voltage variations due to NBTI and PV both need to be captured in order to get an accurate view of silicon reliability.

This chapter covers work published in ISQED 2011 [1].

Chapter 7

Conclusions and Future Work

NBTI is one of the most important reliability problems in silicon devices facing processor designers. This dissertation looks at NBTI mitigation techniques for the microarchitectural structures in a microprocessor and creates the foundation for understanding NBTI in the context of other physical phenomena that affect the processor. Chapter 3 described an analytical model that captures NBTI for use in circuit and architecture simulations. Existing models cannot be directly used for architecture-level simulations. This is because these models assume continuous stress on the PMOS devices in a circuit and lack the additive property. Also these models do not capture scenarios where there are multiple sequences of varying stress/recovery times, which is the case when real workloads run on the processor. To address these problems, this chapter presented a model that represents the degradation history in terms of the equivalent stress time experienced by the PMOS device instead of the V_t value used by the existing models.

With the architecture-level NBTI model, our next research developed techniques that can combat NBTI to meet the service life guarantee with minimal performance,

power, and area overheads. Modern processor cores are composed of several critical SRAM-based structures, such as the register file and the issue queue. Chapter 4 described mitigation techniques for the memory structures in the processor core to maximize their lifetimes. SRAM memory cells are especially vulnerable to NBTI since the input to one of the PMOS devices in the cell is always at a logic '0'. In this chapter, we proposed recovery boosting, a technique that allows both PMOS devices in the cell to be put into the recovery mode by raising the ground voltage and the bitline to V_{dd}. We showed how fine-grained recovery boosting can be used to design the physical register file and issue queue and evaluated their designs via SPICE-level simulations. We then showed that area-neutral designs of these two structures can provide significant reliability benefits with very little impact on power consumption and negligible loss in performance.

The fine-grained recovery boosting approach that we evaluated in this chapter can be used for small SRAM arrays. This work can be extended to study the use of coarse-grained recovery boosting, which imposes less area overheads, for designing caches. Caches pose additional challenges, such as identifying when lines become valid to put them into the recovery boost mode. Use of techniques such as dead-block prediction [51] in conjunction with recovery boosting can be explored to mitigate the impact of NBTI on caches.

Chapter 5 evaluated both circuit and microarchitecture level approaches to reduce the NBTI guardband for the FUs of a high-performance processor core. At the circuit-level, an optimized version of a partitioned FU design is evaluated with several design points in terms of their effectiveness in reducing the guardband and also their area, delay, and power. At the microarchitecture-level, a set of NBTI-aware dynamic in-

struction scheduling policies are proposed and evaluated in terms of their impact in terms of guardband reduction and performance. Finally, this chapter showed that a multi-level optimization approach, which combines the benefits of both circuit and microarchitecture level optimizations, is the most effective in reducing the guardband while imposing little overhead in terms of area, power, delay, and performance.

However, as shown in Chapter 5, the mitigation technique results in a guardband reduction along with an increase in the critical path delay of the FU. Even though the mitigation technique allows for the guardband reduction which will result in cycle time reduction (increase in frequency), the increase in critical path delay also impacts the cycle time in a negative way. Therefore, it is not evident how to estimate the cycle time or guardband requirement from the results given in this chapter. In addition, not only the frequency or cycle time gets affected by the process of guardbanding and mitigation techniques, but also other metrics such as area, power, temperature might get altered. For example, increase in frequency due to the guardband reduction could lead to an increase in temperature which is not feasible for the core. Hence, it raises the question of what would be the ideal frequency given the reliability impacts of the problems and the benefits of the mitigation techniques and the core condition. Also, if the mitigation technique introduces power or area overheads, there are questions about how much overhead can be tolerated to achieve the target guardband reduction. Thus far, there is no systematic way of setting the guardband given all the metrics and mitigation techniques. Developing a systematic approach to analyzing these tradeoffs and deriving appropriate guardbands is future work.

Chapter 6 presented an analytical model that captures both NBTI and PV for use in circuit and architecture simulations. The following aspects are captured in the model:

i) variation in NBTI due to workloads, ii) temporal variation in NBTI and iii) process variation. This model is used to analyze the combined impact of NBTI and PV on a memory structure (register file) and a logic structure (Kogge-Stone adder). We show that the impact of the threshold voltage variations due to NBTI and PV both need to be captured in order to get an accurate view of silicon reliability.

A number of studies have been conducted to investigate the effect of NBTI on both digital and analog circuits. However, certain device-level aspects of NBTI have not been well characterized and modeled. It is important to have an holistic understanding of NBTI by examining the interaction between this reliability phenomenon with process variation, leakage current, and overall power consumption. There are several key unresolved questions, which must be answered, in order to broaden our understanding of the problems and to offer solutions to mitigate them. For example, previous efforts focus on the negative bias caused by the gate-to-source connection (V_{gs}) of the PMOS device. Other kinds of negative bias caused by the gate-to-drain (V_{gd}) or gate-to-body (V_{gb}) connections are still unexplored. Also, the effect of temperature on the NBTI recovery is not investigated yet. Secondly, since NBTI affects the V_t and leakage current is dependant on V_t, it is important to understand the impact of NBTI on the leakage current. Leakage current causes the processor to consume more power and generates heat which degrades the processor performance. The leakage current increases with lower V_t and decreases with higher V_t of the device. With continuous technology scaling, transistors end up having thinner insulating layers which translates to lower V_t, causing more leakage current. On the other hand, NBTI increases the V_t of the devices in a detrimental way which affects the speed of the devices. If NBTI facilitates leakage power reduction, the effect of NBTI could be utilized as a power management knob

and balance between reliability and power consumption. However, if NBTI exacerbates the leakage current condition, then it is needed to cope with both reliability and power, making NBTI mitigation even more critical.

The current practice in handling NBTI is to employ guardbanding. However, guardbanding needs to cover the worst case from both PV and NBTI, and can lead to large area and power overhead. An alternative solution to this problem is to embed on-chip sensors to dynamically track NBTI [52, 53] and use mitigation techniques to handle the problem before it manifests as system level failures. Recent efforts propose dedicated sensors for this purpose which comes with the cost of extra area or performance [52, 53]. In order to reduce this overhead, one could investigate if power consumption of the chip (or components of the chip) changes with the degradation due to NBTI and if power could be used as a sensor for tracking NBTI. If there is any correlation between these two metrics, just by monitoring the power consumption, it should be possible to realize the degradation of the chip (or components of the chip) by using the correlation. In this case, the extra dedicated NBTI sensors would not be necessary, rather existing power sensors [54, 55] could be used for this purpose.

Bibliography

[1] T. Siddiqua, S. Gurumurthi, and Mircea Stan. Modeling and Analyzing NBTI in the Presence of Process Variation. In *International Symposium on Quality Electronic Design*, March 2011.

[2] T. Siddiqua and S. Gurumurthi. Recovery Boosting: A Technique to Enhance NBTI Recovery in SRAM Arrays. In *IEEE Computer Society Annual Symposium on VLSI*, July 2010.

[3] T. Siddiqua and S. Gurumurthi. Enhancing NBTI Recovery in SRAM Arrays through Recovery Boosting. In *IEEE Transactions on Very Large Scale Integration Systems*, 2011.

[4] T. Siddiqua and S. Gurumurthi. A Multi-Level Approach to Reduce the Impact of NBTI on Processor Functional Units. In *Great Lakes Symposium on VLSI*, May 2010.

[5] W. Wang, V. Reddy, A.T. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao. Compact modeling and simulation of circuit reliability for 65-nm cmos technology. In *IEEE Transactions on Device and Materials and Reliability*, 2007.

[6] G. Chen et al. Dynamic nbti of pmos transistors and its impact on device lifetime. In *Reliability Physics Symposium Proceedings*, 2003.

[7] M. Denais et al. New perspectives on nbti in advanced technologies: modelling characterization. In *Solid-State Device Research Conference*, 2005.

[8] V. Huard and M. Denais. Hole trapping effect on methodology for dc and ac negative bias temperature instability measurements in pmos transistors. In *Reliability Physics Symposium Proceedings*, 2004.

[9] C. Shen et al. Characterization and physical origin of fast vth transient in nbti of pmosfets with sion dielectric. In *IEDM*, 2006.

[10] R. Vattikonda et al. Modeling and minimization of pmos nbti effect for robust nanometer design. In *DAC*, 2006.

[11] B. E. Deal et al. Characteristics of the surface state charge (q[sub ss]) of thermally oxidized silicon. In *Journal of The Electrochemical Society*, 1967.

[12] A. Goetzberger et al. On the formation of surface states during stress aging of thermal si-sio[sub 2] interfaces. In *Journal of The Electrochemical Society*, 1973.

[13] M. A. Alam and S. Mahapatra. A comprehensive model of pmos nbti degradation. In *Microelectronics Reliability*, 2005.

[14] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 276–287, June 2004.

[15] A. Tiwari and J. Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2008.

[16] M. Basoglu et al. NBTI-Aware DVFS: A New Approach to Saving Energy and Increasing Processor Lifetime. In *ISPLED*, 2010.

[17] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-Aware Processor. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture*, 2007.

[18] S.V. Kumar, C.H. Kim, and S.S. Sapatnekar. Impact of NBTI on SRAM Read Stability and Design for Reliability. In *Proceedings of the International Symposium on Quality Electronic Design*, 2006.

[19] J. Shin, V. Zyuban, P. Bose, and T.M. Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache sram lifetime. In *Proceedings of the International Symposium on Computer Architecture*, 2008.

[20] E. Gunadi, A.A. Sinkar, N.S. Kim, and M.H. Lipasti. Combating Aging with the Colt Duty Cycle Equalizer. In *International Symposium on Microarchitecture*, 2010.

[21] A. Tiwari et al. ReCycle: Pipeline Adaptation to Tolerate Process Variation. In *ISCA*, 2007.

[22] E. Chun et al. Shapeshifter: Dynamically Changing Pipeline Width and Speed to Address Process Variations. In *MICRO*, 2008.

[23] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. Utilizing Processors with Variation-Induced Timing Errors. In *International Symposium on Microarchitecture*, 2008.

[24] K. Kang et al. Estimation of Statistical Variation in Temporal NBTI Degradation and its Impact on Lifetime Circuit Performance. In *ICCAD*, 2007.

[25] S. Basu and R. Vemuri. Process Variation and NBTI Tolerant Standard Cells to Improve Parametric Yield and Lifetimes of ICs. In *ISVLSI*, 2007.

[26] Y. Lu et al. Statistical Reliability Analysis Under Process Variation and Aging Effects. In *DAC*, 2009.

[27] X. Fu, T. Li, and J. Fortes. NBTI Tolerant Microarchitecture Design in the Presence of Process Variation. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2008.

[28] Cadence Virtuoso Spectre Circuit Simulator. http://www.cadence.com/products/cic/spectre_circuit/.

[29] N.L. Binkert and et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, July 2006.

[30] SPEC CPU2000. http://www.spec.org/cpu2000/.

[31] Predictive Technology Model. http://www.eas.asu.edu/˜ptm/.

[32] Y. Li, D. Brooks, Z. hu, and K. Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2005.

[33] X. Yang, E. Weglarz, and K. Saluja. On NBTI Degradation Process in Digital Logic Circuits. In *Proceedings of the International Conference on VLSI Design*, pages 723–730, January 2007.

[34] A. Sil, S. Ghosh, N. Gogineni, and M. Bayoumi. A Novel High Write Speed, Low Power, Read-SNM-Free 6T SRAM Cell. In *Proceedings of the Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 771–774, August 2008.

[35] G. Reimbold and et al. Initial and PBTI-induced traps and charges in Hf-based oxides/TiN stacks. *Microelectronics Reliability*, 47(4-5):489–496, April 2007.

[36] J.P. Shen and M.H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors (Beta Edition)*. McGraw Hill, 2003.

[37] H. Akkary, R. Rajwar, and S.T. Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 423–434, December 2003.

[38] O. Ergin, D. Balkan, D. Ponomarev, and K. Ghose. Increasing Processor Performance Through Early Register Release. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 480–487, October 2004.

[39] D. Folegnani and A. Gonzalez. Energy-Effective Issue Logic. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 230–239, June 2001.

[40] S. Palacharla. *Complexity-Effective Superscalar Processors*. PhD thesis, University of Wisconsin - Madison, 1998.

[41] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.

[42] E. Seevinck, F.J. List, and J. Lohstroh. Static-Noise Margin Analysis of MOS SRAM Cells. *IEEE Journal of Solid-State Circuits*, 22(5), October 1987.

[43] P. Kogge and H. Stone. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. In *IEEE Transactions on Computers*, 1973.

[44] M. Powell, S-H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In *Proceedings of the International Symposium on Lower Power Electronics and Design (ISLPED)*, pages 90–95, July 2000.

[45] T. Siddiqua and S. Gurumurthi. Balancing Soft Error Coverage with Lifetime Reliability in Redundantly Multithreaded Processors. In *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, September 2009.

[46] M. Agostinelli et al. Random Charge Effects for PMOS NBTI in Ultra-Small Gate Area Devices. In *IRPS*, 2005.

[47] K. Kang et al. Statistical Timing Analysis Using Levelized Covariance Propagation Considering Systematic and Random Variations of Process Parameters. In *TODAES*, 2006.

[48] B. Zhang and M. Orshansky. Modeling of NBTI-Induced PMOS Degradation under Arbitrary Dynamic Temperature Variation. In *ISQED*, 2008.

[49] S. Pae et al. Effect of BTI Degradation on Transistor Variability in Advanced Semiconductor Technologies. In *TDMR*, 2008.

[50] M. Rahma et al. Reducing SRAM Power Using Fine-Grained Wordline Pulsewidth Control. In *TVLSI*, 2009.

[51] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 240–251, June 2001.

[52] A.C. Cabe et al. Small embeddable NBTI sensors (SENS) for tracking on-chip performance decay. In *International Symposium on Quality Electronic Design*, 2009.

[53] J. Keane et al. An On-Chip NBTI Sensor for Measuring pMOS Threshold Voltage Degradation. In *IEEE Transaction on Very Large Scale Integration*, 2010.

[54] M. Ware et al. Architecting for power management: The IBM POWER7 approach. In *International Symposium on Computer Architecture*, 2010.

[55] V. Sylvester et al. ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon. In *IEEE Design and Test of Computers*, 2006.