Cross-Layer Approaches for an Aging-Aware Design of Nanoscale Microprocessors

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

an der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Fabian Oboril

aus Karlsruhe

Tag der mündlichen Prüfung: 29.01.2015

Referent: Prof. Dr. Mehdi Baradaran Tahoori, KIT
Korreferent: Prof. Dr. Jörg Henkel, KIT

For my parents.

ACKNOWLEDGMENTS

Let us be grateful to people who make us happy, they are the charming gardeners who make our souls blossom. *(Marcel Proust)*

Without the help and support of some others, this work would have not emerged in this form. I am very grateful for all these people.

I want to sincerely thank my advisor, Prof. Dr. Mehdi Baradaran Tahoori for the intensive support and countless discussions throughout the last years. He shaped my research interests, challenged me, and also always encouraged me at the right time. In addition, he gave me complete freedom to explore my own directions, for which I am very thankful. His vision provided me a great opportunity for exceptional professional and personal growth. He also gave me the opportunity to mentor students as well as to assist in lectures, and by that means gave me the chance to improve my didactic and teaching skills. Furthermore, I want to thank Prof. Dr. Jörg Henkel for taking the Korreferat, the useful and helpful comments.

Furthermore, I would like to thank all of my colleagues and friends in the Chair of Dependable Computing at Karlsruhe Institute of Technology for their continuous support and encouragement. In particular, I would like to express my gratitude to Saman Kiamehr, who was always a great help and with whom I share many great personal and professinals experiences. I also want to thank Jos Ewert for his contributions to parts of the experimental platform which we developed.

I am also grateful to the DFG Priority Program SPP 1500 "Dependable Embedded Systems" for the financial support of my research work.

Last but not least, I would like to deeply thank my friends Viola Riess, Benedigt Galler and Jörg Schmid for proofreading as well as their continuous support and motivation in the past years. Furthermore, I want to express my gratitude to my girlfriend Sabine for her love and encouragement during my odyssey in the research world, and to my parents Ulrike and Peter as well as my sisters Anja and Saskia for their love and the strong cohesion that they gave me throughout my life.

Fabian Oboril
Hofäckerstr. 37a
76139 Karlsruhe

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben haben und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen - die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, Dezember 2014

Fabian Oboril

CONTENTS

Contents vii

Glossary xi

List of Figures xiii

List of Tables xv

List of Algorithms xvii

List of own Publications included in this Thesis xix

Abstract xxi

Zusammenfassung der Arbeit xxiii

1. Introduction 1

1.1. Accelerated Transistor Aging: Status Quo and Trends . . . 3
1.2. Contribution of this Thesis . . . 4
1.3. Outline . . . 6

I. Background 7

2. Background & State of the Art 9

2.1. Basic Terminology of CMOS Transistors . . . 9
2.2. Reliability: Basic Notions and Challenges . . . 10
2.2.1. Notion of Reliability . . . 11
2.2.2. Sources of Unreliability . . . 13
2.3. Background on Transistor Aging . . . 15
2.3.1. Bias Temperature Instability . . . 15
2.3.2. Hot Carrier Injection . . . 21
2.3.3. Summary . . . 24
2.4. State of the Art . . . 25
2.4.1. Modeling and Simulation . . . 25
2.4.2. Mitigation . . . 27
2.4.3. Error detection and correction schemes . . . 29

II. Frameworks for Aging Modeling and Evaluation 31

3. Cross-Layer Aging Modeling and Analysis Frameworks 33
3.1. Overview 33
3.2. ExtraTime: Modeling and Analysis of Transistor Aging at Architecture-Level 34
3.2.1. Overview and Introduction of the Basic Components 35
3.2.2. Aging Models at Architecture-Level 38
3.2.3. Accuracy Analysis 41
3.2.4. Comparison with State of the Art Approches 43
3.3. ExtraTime in a real system 44
3.3.1. Description and Explanation of the Hardware Platform 44
3.4. Modeling and Analysis of Transistor Aging at Register-Transfer-Level 47
3.4.1. Overview and Introduction of the Basic Components 47
3.4.2. Aging Analysis Tool 49
3.4.3. Enhancements 50
3.4.4. Comparison with other Low-Level Platforms 51
3.5. Summary 52

III. Aging Mitigation Techniques 53

4. Aging-Aware Design of Microprocessor Instruction Pipelines 57
4.1. Problem Introduction and Motivation 57
4.2. Main Idea 59
4.3. Aging-Aware Pipeline Design Methodology 60
4.3.1. Generation of an MTTF-balanced Pipeline Design 60
4.3.2. Modification (faster/slower) of a Pipeline Stage 62
4.3.3. Runtime Analysis and Improvements 62
4.4. Experimental Results 64
4.4.1. Optimization for FabScalar 67
4.4.2. Optimization for OpenSPARC T1 69
4.4.3. Comparison of FabScalar and OpenSPARC T1 71
4.5. Comparison with Related Work 71
4.6. Summary and Conclusion 72

5. Aging-aware Instruction Scheduling 73
5.1. Problem Introduction and Motivation 73
5.2. Aging-aware Instruction Scheduling Methodology 74
5.2.1. Instruction Classification 75
5.2.2. Aging-Aware Scheduling 76
5.2.3. Optimal Cut-off Line between TC and NTC Instructions 77
5.2.4. Further Extensions 78
5.3. Experimental Results 79
5.3.1. Power and Area Evaluation 80
5.3.2. Performance and Lifetime Evaluation 80
5.4. Comparison with Related Work 85
5.5. Summary and Conclusion 86

6. Aging-aware Instruction Set Encoding for Lifetime Improvement 87
6.1. Problem Introduction and Motivation . . . 87
6.2. ArISE: Aging-aware Instruction Set Encoding . . . 88
6.2.1. Optimization using Simulated Annealing . . . 89
6.2.2. Optimization using a Genetic Algorithm . . . 90
6.2.3. Hierarchical Optimization . . . 92
6.2.4. Co-Optimization of Lifetime and Memory Power Consumption . . . 94
6.2.5. Runtime Analysis and Further Improvements . . . 95
6.2.6. Applying Modified Instruction Encoding . . . 96
6.3. Experimental Results . . . 97
6.3.1. Evaluation of an Aging-aware Instruction Set Encoding . . . 98
6.3.2. Impact of an Aging-aware ISE on the entire Processor . . . 100
6.3.3. Impact of an Aging-aware ISE on the Memory Power Consumption . . . 101
6.3.4. Aging-aware ISE vs. Periodical Inversion . . . 102
6.4. Comparison with Related Work . . . 103
6.5. Summary and Conclusion . . . 104

7. Aging-aware Proactive Dynamic Runtime Adaptation 105
7.1. Problem Introduction and Motivation . . . 105
7.2. Dynamic Runtime Adaption Methodology . . . 107
7.2.1. Runtime Monitoring . . . 107
7.2.2. Expert System . . . 108
7.2.3. Dynamic Voltage and Frequency Scaling Policy . . . 108
7.2.4. Possible Implementation of the Expert System . . . 112
7.3. Experimental Results . . . 112
7.3.1. Experimental Setup . . . 112
7.3.2. Results . . . 113
7.4. Comparison with Related Work . . . 115
7.5. Summary and Conclusion . . . 116

IV. Summary, Conclusions and Outlook 117

8. Summary, Conclusions and Outlook 119
8.1. Summary and Conclusions . . . 119
8.2. Outlook . . . 120

Bibliography 123

GLOSSARY

Symbols | A | B | C | D | F | G | H | I | L | M | N | O | P | R | S | T | V

Symbols

V_{dd} Supply voltage of a transistor.

V_{gs} Gate-Source voltage of a transistor.

V_{th} Transistor threshold voltage.

α Transistor switching activity (→ Section 2.3.2).

δ Transistor duty cycle which is the ratio of ON to OFF time (→ Section 2.3.1).

A

Aging rate The aging rate of a device refers to its aging-induced relative delay increase (→ Section 2.3.3).

ALU Arithmetic Logic Unit.

B

BTI Bias Temperature Instability (→ Section 2.3.1).

C

CHC Channel Hot Carrier (→ Section 2.3.2).

CMOS Complementary Metal-Oxide-Semiconductor.

D

DTS Digital Thermal Sensor.

DUI Design Under Investigation.

Duty cycle The transistor duty cycle δ is the ratio of ON to OFF time (→ Section 2.3.1).

DVFS Dynamic Voltage and Frequency Scaling.

F

FabScalar Academic superscalar processor with configurable out-of-order instruction pipeline [1].

FPU Floating Point Unit.

G

GA Genetic Algorithm.

Gem5 Configurable performance simulator supporting various instruction set architectures [2] (→ Section 3.2.1).

Guardband Additional timing margin added by designers to avoid timing failures within a given lifetime.

H

HCI Hot Carrier Injection (→ Section 2.3.2).

HotSpot Compact thermal model [3] (→ Section 3.2.1).

I

IPC Instructions Per Cycle: Average number of instructions executed by a microprocessor for each clock cycle.

ISE Instruction Set Encoding.

ITRS International Technology Roadmap for Semiconductors.

IVC Input Vector Control (→ Section 2.4.2).

IVM Illinois Verilog Model of the Alpha 21264 microprocessor at Register-Transfer-Level [4].

L

LSU Load-Store Unit.

LTF Linear Trend Function (→ Section 7.2).

M

McPAT Power and area modeling framework [5] (→ Section 3.2.1).

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor (→ Section 2.1).

MTTF Mean Time To Failure.

N

NBTI Negative Bias Temperature Instability (→ Section 2.3.1).

NMOS n-type MOSFET (electrons are charge carriers).

NOP No-Operation Instruction.

NTC Non-Timing Critical.

O

OpenSPARC T1 Industrial processor with an in-order, 6-stage pipeline that features 4-way simultaneous multithreading (SMT) [6].

P

PBTI Positive Bias Temperature Instability (→ Section 2.3.1).

PG Power Gating.

PMOS p-type MOSFET (holes are charge carriers).

R

RTL Register-Transfer-Level.

S

SA Simulated Annealing.

SAIF Switching Activity Interchange Format.

SDF Standard Delay Format.

Signal probability The signal probability is the probability of a signal to have the logic value '1'.

Slack The slack is the time difference between the signal arrival time and the clock edge (→ Section 2.1).

SPR Special Purpose Register.

T

TC Timing Critical.

TR Toggle rate of a logic signal.

TTF Time To Failure.

V

VCD Value Change Dump.

VLSI Very Large Integration Scale.

LIST OF FIGURES

1.1. Microprocessor development and transistor count from 1970 to 2014 1
1.2. Failure rate projections for recent technology nodes 2
1.3. Acceleration of transistor aging based on projections from Intel [7] 3
1.4. Design stack and its impact on transistor aging 3

2.1. Basic illustration and terminology of a MOSFET 9
2.2. Illustration of the delay increase due to aging for a CMOS inverter 10
2.3. Illustration of the circuit delay increase due to aging for a simple circuit 10
2.4. Fundamental chain of reliability threats 11
2.5. Chain of fault, error and failure in case of transistor aging 11
2.6. Dependability tree 12
2.7. Important reliability challenges in nanoscale CMOS technology nodes 13
2.8. Guardband decomposition for the IBM Power7+ [8] 14
2.9. Illustration of the $Si - SiO_2$ interface showing dangling bonds and Si-H bonds 16
2.10. Illustration of the two BTI phases (stress and recovery) 17
2.11. Illustration of a sequence of stress and recovery phases and their impact on $|V_{th}|$ 17
2.12. Illustration of the duty cycle, supply voltage and temperature impact on BTI . 18
2.13. Physical mechanism of HCI 22
2.14. Illustration of the runtime, supply voltage and temperature impact on HCI . . 22

3.1. Abstraction levels covered by ExtraTime and the RTL-Platform 34
3.2. Gem5 performance simulator platform with a single core configuration 35
3.3. Overview of the McPAT power and area modeling framework 36
3.4. Overview of the HotSpot thermal modeling framework 37
3.5. Data flow in the proposed ExtraTime framework 37
3.6. Layout of an Intel Core i5 3rd gen microprocessor 44
3.7. Power consumption of the SPEC2006 gcc benchmark for two different processors 46
3.8. Power consumption of two different applications for the Intel Core i5-3450 . . . 46
3.9. Temperature trace for two applications for the Intel Core i5-3450 46
3.10. Basic components of the RTL aging evaluation platform 47
3.11. Illustration of inputs and intermediate outputs of the RTL-platform 48
3.12. Illustration of the employed aging modeling approach 50
3.13. Aging estimation inaccuracy due to pre-synthesis simulations compared to comprehensive post-synthesis simulations 51
3.14. Overview of the proposed aging mitigation techniques 55

4.1. Delay degradation and MTTF for the pipeline stages of two microprocessors . . 58
4.2. Effect of using an MTTF-balanced design on MTTF and performance 59

4.3. Pipeline stage modification required to generate a MTTF-balanced design . . . 62
4.4. Transformation processes from a delay-balanced to an MTTF-balanced design . 63
4.5. Aging rates of different pipeline stages of the FabScalar microprocessor for six SPEC2000 benchmarks 64
4.6. Aging rates of different pipeline stages of the FabScalar microprocessor for different datasets 65
4.7. Delay degradation of the delay-balanced design and MTTF-balanced design for the FabScalar microprocessor 67
4.8. Illustration of wearout affecting parameters 68
4.9. Delay degradation of the delay-balanced design and MTTF-balanced design for the OpenSPARC T1 microprocessor 69

5.1. Illustration of timing-critical (TC) and non-timing-critical (NTC) instructions . 74
5.2. Worst-case delay and occurrence rate of all instructions for the IVM [4] ALU . 75
5.3. Assignment of instructions to functional units for balanced and aging-aware scheduling policies 76
5.4. Instruction classification and its impact on MTTF and performance 77
5.5. Comparison of different ALU configurations using aging-aware IVC 81
5.6. MTTF for different input vectors and different ALU configurations 82
5.7. Time flow of power gating periods of an execution unit 83
5.8. Comparison of different ALU configurations using power gating 84

6.1. Worst-case delay change for different pipeline stages for different ISEs 88
6.2. Exemplary recombination of ISE_1 & ISE_2 to $ISE_{1\times2}$ and $ISE'_{1\times2}$ 92
6.3. Instruction classification and sequence of the hierarchical optimization process . 92
6.4. Improvements of various ISEs in terms of aging and switching activity of the opcode bits in the instruction buffer compared to the default ISE 94
6.5. ISE optimization flows including aging analysis 96
6.6. Using lower and upper bounds as optimization constraints 96
6.7. Delay degradation of the decoding stages for different benchmarks using the standard ISE and the best ISE in terms of lifetime 100

7.1. Frequency of an Intel Core i5-3450 while a Youtube video is played back using the DVFS ondemand governor of Linux [9] 106
7.2. Reactive vs. proactive adaptation philosophy 107
7.3. Organization of the expert system for dynamic runtime adaptation 108
7.4. Linear trend function used to predict future (critical) system states 110
7.5. Effect of parameter changes on wearout at time t_1 113
7.6. Effect of different DVFS policies on MTTF, energy, temperature and performance 114
7.7. Comparison of a DVFS technique with and without trend analysis 114
7.8. Influence of application on the efficiency of DVFS (hexa f + trend) 115

8.1. Technology roadmap based on ITRS [10] from 2013 until 2025 120

LIST OF TABLES

2.1. Summary of the BTI model employed in this thesis 21
2.2. Summary of the HCI model employed in this thesis 24
2.3. Impact of different parameters on BTI and HCI 25

3.1. Comparison of the two different cross-layer platforms 33
3.2. Temperature effect on the aging-induced delay change for an ALU 42
3.3. Inaccuracy of the proposed architectural aging models compared to detailed transistor-level models 42
3.4. Employed SPRs for the proposed power estimation (here for Intel processors) . 45
3.5. Tools employed in the RTL aging estimation framework 49

4.1. Architecture comparison of FabScalar and OpenSPARC 64
4.2. Comparison of a delay-balanced and MTTF-balanced design for FabScalar . . . 66
4.3. Comparison of a delay-balanced and MTTF-balanced design for OpenSPARC T1 70

5.1. Workloads and their instruction ratios using a single ALU 75
5.2. Configuration details for the experiments 79
5.3. Different ALU configurations and their impact on area and power 80
5.4. MTTF for several SPEC2000 benchmarks for the standard (balanced) scheduling and the proposed aging-aware scheduling using an aging-aware IVC for different ALU configurations 81
5.5. Summary for the standard (balanced) scheduling and the proposed aging-aware scheduling using an aging-aware IVC for different ALU configurations 81
5.6. MTTF for several SPEC2000 benchmarks for the standard (balanced) scheduling and the proposed aging-aware scheduling using power gating for different ALU configurations 84
5.7. Summary for the standard (balanced) scheduling and the proposed aging-aware scheduling using power gating for different ALU configurations 84

6.1. Architecture setup of the FabScalar microprocessor 97
6.2. Optimization parameters 98
6.3. Comparison of different optimization strategies to generate an aging-aware ISE for the decoding stages 99
6.4. Recombination of two “good” ISEs resulting in “worse” ISEs 99
6.5. Differences between the default ISE and the best ISE in terms of MTTF 100
6.6. Improvements of the best ISE in terms of MTTF and delay for the decoding stages 100
6.7. Comparison of the FabScalar’s standard ISE and the best obtained one 101

6.8. Comparison of optimization strategies with and without considering the toggle rate in memories 102
6.9. Comparison between our proposed technique and periodical ISE inversion [11] . 103

7.1. Intel Core i5-3450 P-States 106
7.2. Functionality of the local experts 108
7.3. Configuration details for the experiments 112
7.4. Effect of different sampling periods on performance and lifetime 113

LIST OF ALGORITHMS

4.1. Transformation of a delay-balanced design to an MTTF-balanced design 61
5.1. Flow to obtain instruction groups such that the overall MTTF is maximized . . 78
6.1. Simulated annealing approach to generate an aging-aware ISE 90
6.2. Genetic algorithm to generate an aging-aware ISE 91
6.3. Hierarchical approach to obtain an aging-aware ISE 93
7.1. Proactive DVFS methodology using trend analysis 109

LIST OF OWN PUBLICATIONS INCLUDED IN THIS THESIS

Transactions (peer-reviewed)

[12] F. Oboril and M. Tahoori, "Aging-Aware Design of Microprocessor Instruction Pipelines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 5, pp. 704–716, May 2014.

[13] F. Oboril and M. Tahoori, "Exploiting Instruction Set Encoding for Aging-Aware Microprocessor Design," *ACM Transactions on Design Automation of Electronic Systems*, pp. 1–22, 2014, submitted for review.

[14] F. Oboril, F. Firouzi, S. Kiamehr, and M. B. Tahoori, "Negative Bias Temperature Instability-Aware Instruction Scheduling: A Cross-Layer Approach," *Journal of Low Power Electronics*, vol. 9, no. 4, pp. 389–402, Dec. 2013.

Conferences (double-blind peer-reviewed)

[15] F. Oboril and M. B. Tahoori, "ExtraTime: Modeling and Analysis of Wearout due to Transistor Aging at Microarchitecture-Level," in *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 1–12, Jun. 2012.

[16] F. Oboril and M. Tahoori, "ExtraTime: A Framework for Exploration of Clock and Power Gating for BTI and HCI Aging Mitigation," in *Proceedings of Zuverlässigkeit und Entwurf*, pp. 62–69, Sep. 2011.

[17] F. Oboril and M. Tahoori, "Reducing Wearout in Embedded Processors using Proactive Fine-Grained Dynamic Runtime Adaptation," in *Proceedings of the 17th IEEE European Test Symposium*, pp. 68–73, May 2012.

[18] F. Oboril, F. Firouzi, S. Kiamehr, and M. Tahoori, "Reducing NBTI-induced Processor Wearout by Exploiting the Timing Slack of Instructions," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 443–452, Oct. 2012.

[19] F. Oboril and M. Tahoori, "MTTF-Balanced Pipeline Design," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 270–275, Mar. 2013.

[20] F. Oboril and M. Tahoori, "ArISE: Aging-Aware Instruction Set Encoding for Lifetime Improvement," in *Proceedings of the 2014 Asia and South Pacific Design Automation Conference*, pp. 1–6, Jan. 2014.

[21] F. Oboril, J. Ewert, and M. Tahoori, "High-Resolution Online Power Monitoring for Modern Microprocessors," in *Proceedings of the Conference on Design, Automation and Test in Europe*, Mar. 2014, to appear.

Invited papers

[22] F. Oboril and M. Tahoori, "ExtraTime: Eine Mikroarchitektur-Simulationsumgebung zur Modellierung, Analyse und Linderung von Alterungseffekten," *GMM MECHATRONIK 11/2012*, pp. 13–14, Nov. 2012.

[23] F. Oboril, M. Ebrahimi, S. Kiamehr, and M. Tahoori, "Cross-Layer Resilient System Design Flow," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 2015, to appear.

ABSTRACT

Thanks to the aggressive scaling of transistor dimensions in the past decades, computing systems have revolutionized our life. However, in the shade of the downscaling benefits such as increased microprocessor performance, more integrated features and improved energy/cost efficiency, a major threat for the future success of computing systems has grown: The unreliability of devices fabricated in nanoscale technology nodes. Because of that, with every new technology node, it becomes harder for the chip manufacturers to ensure the reliable operation of their chips, i.e. their correct functionality in the field. As a consequence, malfunctions during the operational mode, that can lead to erroneous program outputs or even system crashes, are more likely to occur.

Among various reliability challenges, accelerated transistor aging is of particular importance. It degrades the transistor switching speed, and thus leads to slower circuits over time. As a result, in synchronous digital systems such as microprocessors, timing failures due to the increased circuit delay can occur and cause incorrect system states. This considerably reduces the overall lifetime of digital systems. To make matters worse, it is projected that transistor aging will become worse with the next technology nodes, if no countermeasures are taken. Hence, it is a necessity to consider reliability, and in particular lifetime, as another design constraint, beside the traditional performance, power and cost parameters. However, due to the strong interdependencies among the different constraints, the co-optimization is very challenging.

To avoid aging-induced failures in the field, designers add timing margins to their designs, which, however, is very inefficient and costly. In addition, a great deal of efforts is spent on improvements at the lowest hardware layers (i.e. at transistor/gate-level), as these layers are very close to the physical origin of the problem. However, the influence of higher levels in the hardware-software design stack such as the architecture- or application-level is neglected in most state-of-the-art solutions, although these layers have a considerable effect on the system lifetime. Therefore, it is crucial to investigate the impact of these higher abstraction layers, to achieve cost-efficient resilient computer systems. This requires models and frameworks to allow an aging-aware design space exploration at higher abstraction layers to effectively co-optimize reliability with the traditional design parameters.

This thesis addresses this challenge for microprocessors and pushes the state-of-the-art forward by proposing novel methods to model, analyze and mitigate transistor aging at higher abstraction levels such as microarchitecture-level and above. For this purpose, unique cross-layer multi-objective frameworks are developed covering the entire abstraction stack from circuit-level up to application-level. These platforms include aging, power and thermal models to enable an effective design space exploration which treats lifetime as an additional design constraint. Using these platforms, several new cross-layer aging-mitigation techniques addressing different microprocessor components are developed. Among these, there are solutions that are employed during the microprocessor design process to improve the functional units, the decoding stages of the instruction pipeline and the pipeline design philosophy, respectively. Furthermore, an aging-aware proactive dynamic system adaptation approach is provided, that tackles transistor aging at runtime, and thus complements the other design time techniques. All of these approaches are cross-layer schemes which means that the combined knowledge of various abstraction levels is exploited, to achieve effective solutions. The effectiveness of the proposed schemes is demonstrated in extensive experiments that show significant improvements of the proposed methods upon state-of-the-art solutions, i.e. better lifetime results with lower costs in terms of performance, power and area.

ZUSAMMENFASSUNG DER ARBEIT

Computersysteme haben in den letzten Jahrzehnten unser alltägliches Leben revolutioniert. Dank immer kleiner werdenden Transistorabmessungen konnten die Chiphersteller einen größeren Funktionsumfang in ihre Computersysteme integrieren, deren Leistungsfähigkeit steigern und zugleich die Energie-/Kosteneffizienz verbessern. Als Folge dessen haben Computersysteme immer mehr Bereiche unseres Lebens erobert und erleichtert. Allerdings bringt die stete Verkleinerung der Bauteile auch einige Probleme mit sich. Eine der größten Herausforderungen ist hierbei die wachsende Unzuverlässigkeit von Bauteilen, die Strukturbreiten im Nanometerbereich aufweisen. Aus diesem Grund wird es für die Chiphersteller mit jedem neuen Verkleinerungsschritt immer schwieriger den zuverlässigen Betrieb der Chips zu gewährleisten, d.h. die korrekte Funktion der Computersysteme im normalen Betrieb sicherzustellen. Damit einhergehend steigt die Wahrscheinlichkeit für Fehlfunktionen während des Betriebs, die zu fehlerhaften Programmausgaben, falschen Rechenergebnissen oder Systemabstürzen führen können.

Neben zahlreichen Zuverlässigkeitsherausforderungen ist die beschleunigte Transistoralterung von besonderer Bedeutung. Sie beeinträchtigt die Schaltgeschwindigkeit der Transistoren im Laufe der Betriebsdauer und führt damit zu einer "langsamer" werdenden Schaltung. Deshalb steigen die Signallaufzeiten mit der Zeit an, was schwerwiegende Laufzeitfehler in synchronen Schaltungen (z.B. Mikroprozessoren) zur Folge haben kann. Diese machen sich in falschen Systemzuständen oder Rechenergebnissen bemerkbar, so dass die korrekte Funktionsweise nicht mehr gewährleistet werden kann. Folglich sinkt auch die zu erwartende Lebensdauer der Computersysteme. In Zukunft wird sich dieser Effekt sogar noch verstärken, wenn keine Gegenmaßnahmen ergriffen werden. Daher ist die Berücksichtigung der Zuverlässigkeit und insbesondere der Lebensdauer als weiterer Entwicklungsparameter neben den traditionellen Werten (Leistung, Energiebedarf und Kosten) von großer Bedeutung. Aufgrund der starken Wechselwirkungen zwischen den unterschiedlichen Randbedingungen ist jedoch die gleichzeitige Optimierung aller Parameter eine große Herausforderung.

Um alterungsbedingte Ausfälle während des Betriebs zu vermeiden, führen Designer üblicherweise zusätzliche Sicherheitsmargen ein, d.h. bei der Festlegung der Länge eines Taktzyklus wird der alterungsbedingte Anstieg der Signallaufzeiten miteinbezogen. Dies ist jedoch sehr ineffizient und teuer. Darüber hinaus wird ein großer Aufwand betrieben, um Verbesserungen auf den untersten Hardwareschichten (d.h. auf Transistor- oder Gatter-Ebene) zu erzielen, da diese Schichten nahe an der physikalischen Ursache des Problems liegen. Dagegen wird der Einfluss höherer Abstraktionsebenen im Hardware-Software-Design-Ablauf, wie z.B. die Architektur- oder Anwendungsebene, in den meisten aktuellen Lösungsansätzen vernachlässigt, obwohl diese Schichten einen erheblichen Einfluss auf die Lebensdauer und Alterungsgeschwindigkeit von Computersystemen haben. Daher ist es von entscheidender Bedeutung die Auswirkungen dieser höheren Abstraktionsschichten zu untersuchen und mit in das Schaltungsdesign einzubeziehen, um kostengünstige und zugleich zuverlässige Computersysteme zu ermöglichen. Dies erfordert allerdings geeignete Modelle und Plattformen, um eine Untersuchung des Parameterraums auf höheren Abstraktionsschichten unter Berücksichtigung der Lebensdauer und Alterungsgeschwindigkeit zu ermöglichen, die das Ziel hat die Lebensdauer und die traditionellen Design-Parametern zu optimieren.

Diese Doktorarbeit befasst sich mit der beschriebenen Herausforderung für Mikroprozessoren und stellt dafür neue Methoden zur Modellierung, Analyse und Bekämpfung der beschleunigten Transistoralterung auf höheren Abstraktionsebenen (Mikroarchitekturebene und höher) vor. Zu diesem Zweck werden neue Simulationsplattformen entwickelt, die den gesamten Abstraktionsraum von der Schaltkreisebene bis zur Anwendungsebene abdecken. Diese Plattformen beinhalten dabei Modelle zur

Analyse der Alterung, des Energiebedarfs und der Temperaturentwicklung. Damit ermöglichen sie eine effektive Untersuchung des gesamten Parameterraums unter Berücksichtigung aller Rahmenbedingungen, zu denen auch die Lebensdauer des Mikroprozessors zählt. Mit Hilfe dieser Plattformen werden mehrere neue Techniken entwickelt, die das Ziel haben die Mikroprozessorlebensdauer zu verbessern. Dazu zählen Methoden, die während des Entwurfsprozesses zum Einsatz kommen, um die Lebensdauer der Ausführungseinheiten sowie der Dekodierstufen in der Befehlspipeline zu verlängern, und um die Design-Philosophie der gesamten Befehlspipeline zu verbessern. Darüber hinaus stellen wir einen neuartigen, dynamischen und proaktiven Lösungsansatz vor, der während des normalen Betriebs die Systemkonfiguration dynamisch auf die jeweils aktuellen Zustände anpasst und dadurch die Alterung verlangsamt. Damit ergänzt diese Methode die vorher genannten Ansätze, die während der Entwurfsphase eingesetzt werden. Alle Vorgehensweisen haben dabei gemein, dass es sich um sogenannte Cross-Layer Ansätze handelt, d.h. sie nutzen die Informationen von verschiedenen Abstraktionsebenen aus, um Lösungen mit hoher Effizienz zu erzielen. Die Wirksamkeit aller vorgestellter Methoden wird in umfangreichen Experimenten analysiert, deren Ergebnisse signifikante Vorteile gegenüber dem aktuellen Stand der Technik aufzeigen, d.h. z.B. eine deutlich verbesserte Lebensdauer bei zugleich geringeren Kosten in Bezug auf Leistungsfähigkeit, Energiebedarf und Fertigungskosten.

CHAPTER

ONE

INTRODUCTION

Nowadays, computing systems are indispensable in our daily life, covering a wide range of different application areas from electronic control units, high performance computing labs, data centers, banking or health care to traditional computers, multimedia devices and smartphones. In all of these fields, the never ending demands for higher performance, more integrated features and increasing energy/cost efficiency drive the need for a continuous downscaling of Very Large Integration Scale (VLSI) technology. Therefore, the microelectronic industry shrinks the Complementary Metal-Oxide-Semiconductor (CMOS) device feature size (i.e. transistor dimensions) approximately every two years, which allows them to double the number of transistors in an integrated circuit in the same time period [28]. This paradigm is known as *Moore's Law* that was postulated by Gordon E. Moore[1] in 1965 [29]. The resulting abundance of transistor-integration capacity was deployed by designers to enhance their microprocessors [30] with more sophisticated architectures such as superscalar and out-of-order instruction pipelines, on-die caches, integrated graphics processing units (GPUs) or multi/many-core approaches, as depicted in Figure 1.1 [24–27]. In other words, the technology development was driving the architecture development.

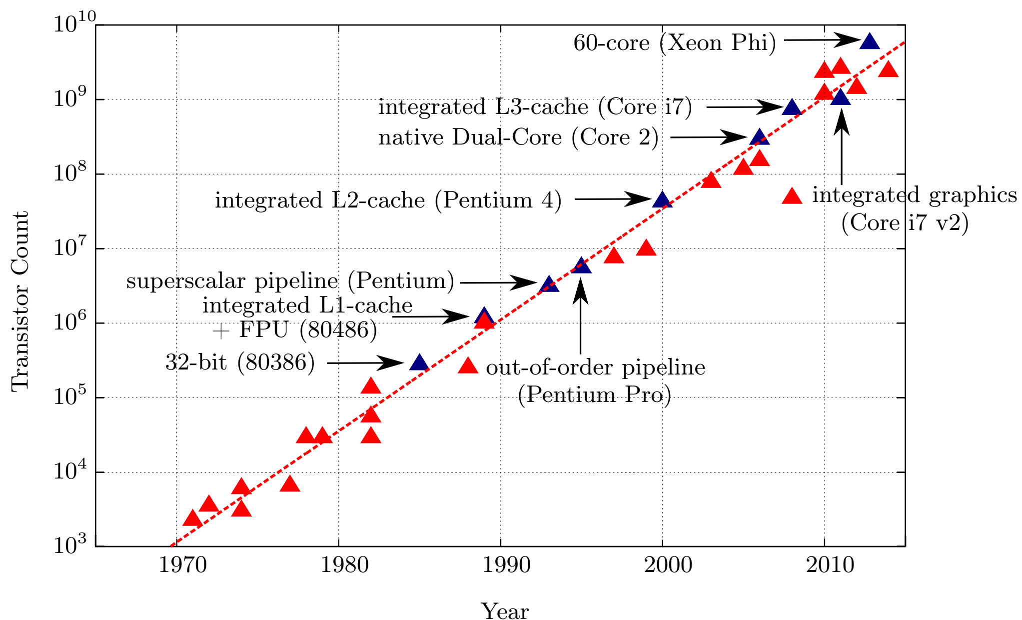
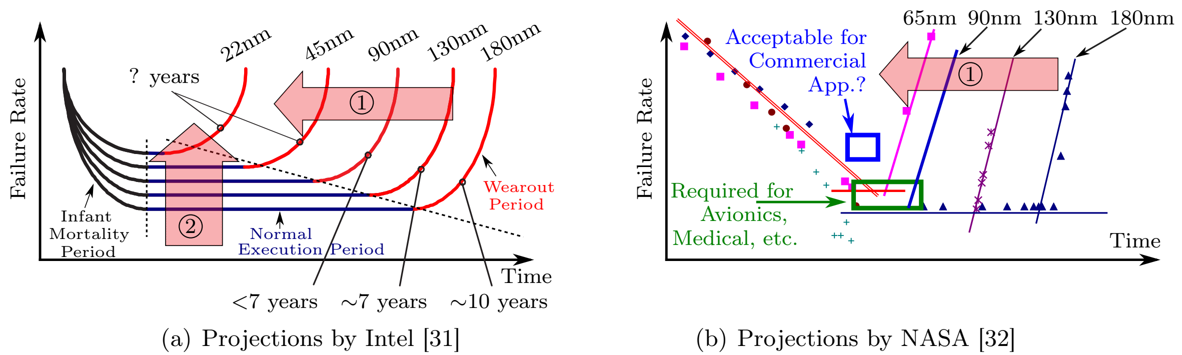


Figure 1.1.: Microprocessor development and transistor count from 1970 to 2014 based on Intel processors illustrating Moore's Law (data taken from [24–27])

[1]Gordon Earle Moore (born 1929) is a co-founder of Intel, a semiconductor chip maker corporation



(a) Projections by Intel [31]

(b) Projections by NASA [32]

Figure 1.2.: Failure rate projections from Intel and NASA for recent technology nodes considering accelerated transistor aging ① and susceptibility to noise as well as soft errors ②

Although the continuous downscaling of CMOS technology is very advantageous for the overall system performance, feature count and mobility, it also comes along with various severe design challenges, especially in the era of nanoscale technology nodes (i.e. below 130 nm). Beside the increasing power density [33, 34], which makes heat dissipation more and more difficult, the increasing *unreliability* (i.e. higher failure rates during the operational time) of devices fabricated in nanometer CMOS technology nodes is critical [10, 30, 35–39]. Because of that, with every new technology node, it becomes harder for the manufacturers to guarantee the reliable operation of their chips, i.e. their correct functionality in the field (see Figure 1.2). The reasons for the increasing reliability challenges are manifold [7, 10, 32, 40–43]. The increasing design complexity and process variability due to lithography issues lead to a rising number of bugs that escape to the field. In addition, transient errors in the field become more likely because of the higher sensitivity to electrical noise and radiation-induced soft errors. Moreover, transistors fabricated in nanoscale dimensions suffer from *accelerated aging*, which degrades the transistor characteristics (e.g. delay and current flowing through the transistor channel) over time such that the circuit lifetime is limited to a few years, as illustrated in Figure 1.2(a).

In summary, due to all of these reliability challenges, the failure rate increases in all phases of the chip lifetime, as illustrated by the bathtub curves in Figure 1.2 based on the data from Intel [31] and NASA [32]. As a consequence, malfunctions in computing systems ranging from annoying computer crashes, through data and financial losses, to loss of human life are more likely to occur [42]. On top of that, the period of "normal execution" (i.e. *lifetime*) shrinks due to accelerated transistor aging (see Figure 1.2), and thus the *Mean Time To Failure (MTTF)*[2] becomes shorter. In addition, it is projected that both unreliability aspects (higher failure rate and reduced MTTF) may increase exponentially in future [7]. As a result, it is a necessity to consider reliability as another design constraint, beside the traditional parameters performance, power and cost. However, due to the strong interdependencies among the different constraints, the co-optimization is very challenging. For example, a higher clock frequency improves the performance at the cost of a higher power consumption and reduced reliability (lifetime), and using redundant hardware to enhance the reliability (lifetime) increases the power consumption as well as the cost. *This thesis tackles this challenge for one of the most important reliability issues, i.e. reduced lifetime due to accelerated transistor aging. Therefore, several novel approaches to efficiently co-optimize aging-induced lifetime degradation, system-level performance, power consumption and cost for nanoscale microprocessors are proposed.*

[2]MTTF is the statistical expectation of the time to failure, i.e. it represents the expected lifetime value. However, in the field, devices may fail sooner or later, since MTTF is a statistical measure and does not stand for a guaranteed lifetime.

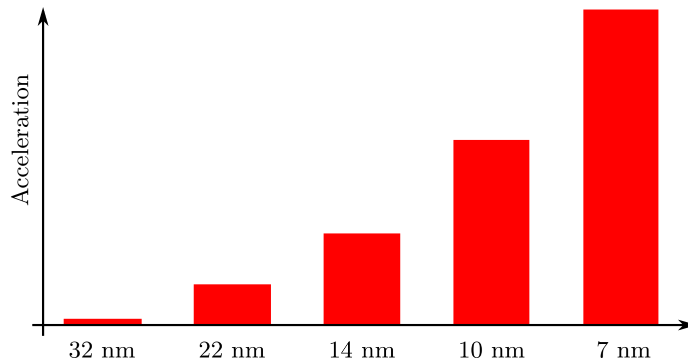


Figure 1.3.: Acceleration of transistor aging based on projections from Intel [7]

1.1. Accelerated Transistor Aging: Status Quo and Trends

Accelerated transistor aging is nowadays one of the most important reliability challenges [10], as it considerably reduces the operational lifetime of nanoscale microprocessors, as depicted in Figure 1.2. Among various physical phenomena that cause transistor aging, *Bias Temperature Instability (BTI)* [44–50] and *Hot Carrier Injection (HCI)* [46, 51–54] are of particular interest [10, 41]. Both lead to a degradation of the transistor's threshold voltage and drive current, where the degradation rate depends on various aspects such as workload, temperature, supply voltage or frequency. As a result of this transistor parameter shift, device, path and circuit delays increase over time, i.e they become "slower" over time. Consequently, erroneous data is eventually calculated and stored, as the correct signal values cannot be obtained in time (i.e. not before the clock edge). Hence, at system-level these problems manifest in corrupted data (bit errors), program as well as system crashes and a reduced MTTF.

In order to avoid such severe scenarios, designers and manufacturers add safety margins[3], so called *guardbands*, to their designs, to ensure the reliable operation for a certain amount of time. However, the guardbands to account for aging in a modern technology node can be 10 % of the clock period[4] for a lifetime of 3 years [55–57]. To make matters worse, the guardbands have to considerably increase in future, since transistor aging is accelerated by downscaling [7], as shown in Figure 1.3. Thus, guardbands are very costly, as expensive overdesigns are required [58].

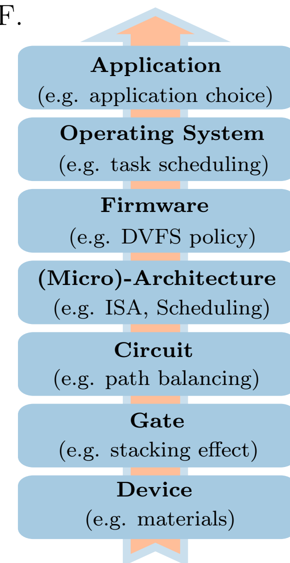


Figure 1.4.: Design stack and its impact on transistor aging

Therefore, techniques to co-optimize lifetime, performance, power and cost are a necessity [10, 35, 38, 39]. For this purpose, various approaches at the lowest hardware layers (i.e. device- to circuit-level) were proposed such as [49, 59–69], as these layers are very close to the physical origin of the problems. However, also higher levels in the design stack of a computing system have a considerable aging impact, as these influence important parameters such as temperature, workload, frequency or supply voltage, for instance via the instruction scheduling policy or the Dynamic Voltage and Frequency Scaling (DVFS) strategy. This important point is motivated in Figure 1.4. In

[3]Employing a safety margin means that the clock frequency of a microprocessor is set according to its slowest component plus an additional timing margin, i.e. clock period = delay of slowest component + margin.

[4]Please note that the actual timing margin strongly depends on the technology and application domain (e.g. low power, high performance, harsh environment, mission-critical).

fact, only very few approaches exploit higher-level knobs to alleviate the impact of accelerated transistor aging such as re-active techniques to adjust the frequency or supply voltage in a stepwise manner to compensate the delay degradation [70–75] as well as load/stress balancing approaches [11, 76–78]. In addition, most of the state-of-the-art techniques exploit only the information of a single abstraction layer and ignore the interdependencies among the different layers. However, if the information of several abstraction layers is taken into consideration to alleviate accelerated transistor aging, reliability, performance, power and cost can be better co-optimized [79–82]. Such approaches, that use the combined knowledge of various layers, are called *cross-layer* techniques.

In summary, due to the considerable impact of accelerated transistor aging on the system lifetime, efficient mitigation techniques are required. In this regard, cross-layer approaches are a great promise, as the combined knowledge of various abstraction layers is exploited, and thus a better tuning of the system with respect to lifetime, performance power and cost is possible. Moreover, also abstraction layers above circuit-level should be taken into account, as these have a great influence on wearout, as well. In other words, now the system-architecture needs to develop to allow further CMOS developments. This need has been also stated recently by the "International Technology Roadmap for Semiconductors" (ITRS) [10] as well as the ENIAC Strategic Research Agenda [39]. *Therefore, this thesis pushes the state-of-the-art in aging modeling and mitigation forward by proposing novel cross-layer modeling and analysis frameworks, as well as several new cross-layer aging mitigation techniques that take also higher abstraction levels of the design stack into account to efficiently co-optimize the lifetime, performance, energy consumption and cost of nanoscale microprocessors.*

1.2. Contribution of this Thesis

As motivated before, the objective of this thesis is to alleviate the impact of transistor aging using efficient means available at higher abstraction levels (i.e. microarchitecture and above). Therefore, device-level models were abstracted all the way to architecture-level to develop fast and accurate lifetime reliability analysis frameworks. Based on these cross-layer multi-objective frameworks, investigations of various wearout influencing parameters from microarchitecture-level all the way to application-level were performed. Furthermore, several novel mitigation techniques employed at runtime and during the design process, that co-optimize lifetime, performance, power and area, were designed. In this regard, cross-layer approaches are of special interest, as explained before.

In particular, the novel contributions of this thesis are:

- **Cross-Layer modeling and analysis platforms to evaluate aging, performance, power and cost:** For evaluating cross-layer aging mitigation techniques two different platforms for different purposes were developed. The first one is an architectural framework called *ExtraTime*, which can be used, when detailed circuit-level information is not required or not available. The second platform is a microarchitectural framework at Register-Transfer-Level, which is intended for investigations that rely on detailed gate-level knowledge. Both platforms include aging, power and thermal models, and are designed in such a way that the interplay of real-world applications, performance, power, temperature and aging is considered. As a result, these frameworks allow a very effective design space exploration for co-optimizing reliability (lifetime), performance, power and cost (die area). In addition, an actual experimental hardware-platform is built using the ExtraTime models to demonstrate their accuracy and capabilities. This platform can be used to evaluate the feasibility of various aging mitigation techniques not only by simulations, but also by employing them in a real system.

- **Aging-Aware Design of Microprocessor Instruction Pipelines:** Traditionally the delays of all instruction pipeline stages are balanced at design time. However, accelerated transistor aging causes a non-uniform delay degradation among all stages due to different usage patterns. Consequently, this design approach results in an imbalanced design after a short period of time. As a result, a single pipeline stage will become the bottleneck for the overall microprocessor lifetime. To alleviate this problem, we propose a novel design paradigm (*MTTF-Balanced pipeline design*) for the microprocessor instruction pipeline according to which all pipeline stage delays are balanced at the end of the desired lifetime. As a result, the overall microprocessor lifetime can be improved without affecting the performance.
- **Aging-aware instruction scheduling:** To alleviate transistor aging inside the execution units, which are among the most critical parts of a microprocessor, an aging-aware instruction scheduling technique was developed. The novelty of this cross-layer scheduling policy is the consideration of the timing-criticality of incoming instructions to increase the idle ratio of functional units executing the most critical instructions. This is exploited to considerably improve the lifetime of the functional units compared to a single-layer balanced scheduling approach that ignores the detailed timing information.
- **Aging-aware instruction set encoding (ArISE):** We observed that beside the execution units also the decoding stages of a microprocessor can become aging-critical and limit the microprocessor lifetime, especially when aging mitigation techniques targeting the functional units are employed. Hence, the decoding stages have to be considered for an aging-aware microprocessor design. Since the instruction set encoding, i.e. the mapping between instructions and opcodes, has a strong influence on the wearout of the decoding stages, we propose a novel aging-aware instruction set encoding methodology called *ArISE* to address the delay degradation in the decoding stages. The result is an optimization that yields significant lifetime improvements with negligible impact on other design parameters.
- **Aging-aware proactive dynamic runtime adaptation:** In order to detect and avoid potentially critical system conditions while the system is running, dynamic aging alleviation schemes employed at runtime (i.e. *runtime solutions*) have to complement mitigation solutions that are applied during the design phase of a microprocessor (i.e. *design time solutions*) [83]. However, the state-of-the-art dynamic runtime adaption techniques employ only reactive methodologies, which are inefficient due to the nature of "damage control"-type of techniques. In other words, these policies deal with already "aged" chips. In contrast, we provide a *proactive* and *preventive* runtime adaptation policy that tries to slow down the delay degradation in all phases of the chip lifetime, and hence can prolong the lifetime more efficiently than state-of-the-art techniques, i.e. with lower performance and power overheads.

All aforementioned aging mitigation techniques are cross-layer approaches that are very effective, as they combine the knowledge from different abstraction layers. We will demonstrate this fact in various parts of the thesis. Moreover, all approaches were evaluated using modern microprocessors such as FabScalar [1] (academic superscalar processor with configurable out-of-order instruction pipeline), OpenSPARC T1 [6] (industrial processor with an in-order, 6-stage pipeline that features 4-way simultaneous multithreading), or the out-of-order core of gem5 that is similar to the Alpha 21264 [84]. In addition, two real hardware-systems, one with an IBM Power7+ processor [8] and one based on recent Intel Core processors [85], are utilized to backup various data with real experimental results. Thus, the theoretical ideas have been analyzed and validated under realistic scenarios.

1.3. Outline

The rest of this thesis is organized in four main parts. The first one (Chapter 2) focuses on the background and state-of-the-art. In Part II (Chapter 3), the developed aging modeling and evaluation frameworks are presented, followed by the proposed aging mitigation techniques in Part III (Chapters 4-7). Finally, Part IV (Chapter 8) concludes this thesis.

After this introduction, the thesis continues with a broad overview of the background on reliability and in particular accelerated transistor aging in Chapter 2. In addition, this chapter also discusses the state-of-the-art techniques to model and alleviate transistor aging at various abstraction levels.

The proposed cross-layer aging modeling and evaluation frameworks are presented in Chapter 3. In the first part of this chapter, the architectural framework ExtraTime is described, and the novel architectural aging models are derived. Afterwards, a real experimental system is introduced that can be used to obtain very localized power and thermal information about various microprocessor components. The RTL-based evaluation platform is provided in the third part of the chapter.

The novel MTTF-balanced instruction pipeline design paradigm is proposed in Chapter 4, which starts with the problem introduction and motivation, followed by the new design methodology. Afterwards, two microprocessors are analyzed to evaluate the benefits of the MTTF-balanced design paradigm. The chapter ends with a discussion of related work and conclusions.

In Chapter 5 we present our envisioned aging-aware instruction scheduling approach. At the beginning, the need for a new scheduling approach is motivated, followed by the presentation of the scheduling technique itself. Next, the scheduling scheme is evaluated using various microprocessor configurations and compared to a state-of-the-art scheduling method. At the end, the related work is discussed followed by some concluding remarks.

Chapter 6 provides the aging-aware instruction set encoding methodology, starting with an introduction of the aging problem in instruction decoders and a motivation of the proposed technique. Afterwards, the scheme is introduced in detail, and two heuristic solutions are provided to solve the complex optimization problem. Next, the proposed encoding approach is analyzed and compared to a standard encoding. Then, a summary of related work is presented and finally the chapter ends with a conclusion.

The last aging-aware technique proposed in this thesis is presented in Chapter 7. In the first part of the chapter, the DVFS methodology is introduced, followed by our envisioned pro-active aging-aware implementation based on a two-level expert system. Afterwards, the benefits and drawbacks of the proposed DVFS technique are investigated and compared against other, non-proactive approaches. Also this chapter ends with a summary of related work and some concluding remarks.

Finally, Chapter 8 summarizes and concludes the thesis. In addition, an outlook into the future is provided in this chapter.

Part I.

Background

CHAPTER
TWO

BACKGROUND & STATE OF THE ART

As laid out in the previous chapter, this thesis tackles one of the most important design aspects of microprocessors fabricated in current and future CMOS technology nodes: The increasing unreliability, and in particular, the shrinking lifetime due to accelerated transistor aging. In order to provide a general background of these issues, some fundamental transistor information is provided in this chapter, and the basic notion of reliability as well as its most important threats are introduced with a special focus on accelerated transistor aging. Finally, a summary of the most relevant state of the art approaches to model and alleviate transistor aging is presented.

2.1. Basic Terminology of CMOS Transistors

Although this thesis tackles the problem of accelerated transistor aging at higher abstraction levels such as microarchitecture-level and above, it is fundamentally important to have a basic understanding of the functionality of a modern transistor. Therefore, in this section, a brief introduction on transistors and their important characteristics is given. Further details can be found in [86].

Nowadays, the manufacturing of microprocessors relies on CMOS: *Complementary Metal-Oxide-Semiconductor*. This means that a combination of p-type and n-type MOSFETs (*Metal-Oxide-Semiconductor Field-Effect Transistors*) is employed to implement logic gates, where p and n represent the charge carriers used in the transistor (n for electrons, p for holes). In this thesis, we refer to p-type MOSFETs as PMOS and NMOS for n-type MOSFETs.

The basic layout of a MOSFET is depicted in Figure 2.1. As the name indicates, it uses a metal gate which is attached to a few nanometer thin oxide layer (e.g. silicon oxide SiO_2) and a semiconducting substrate (typically silicon Si). If, for an NMOS (similar for PMOS), the voltage between gate and source (V_{gs}) is larger than the threshold voltage (V_{th}) of the transistor, a conducting path (i.e. channel) under the gate oxide will be formed between source and drain. In this case, the transistor is referred to as ON and current flows through the channel (on-current). Otherwise, it is not conducting (no channel), i.e. OFF.

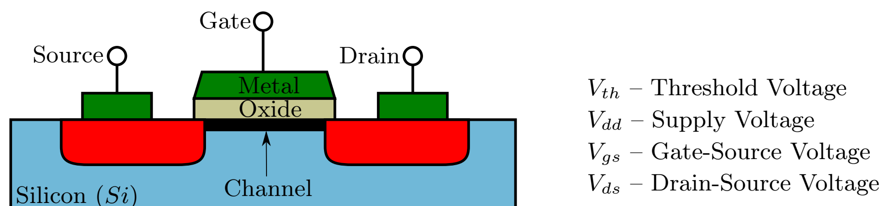


Figure 2.1.: Basic illustration and terminology of a MOSFET

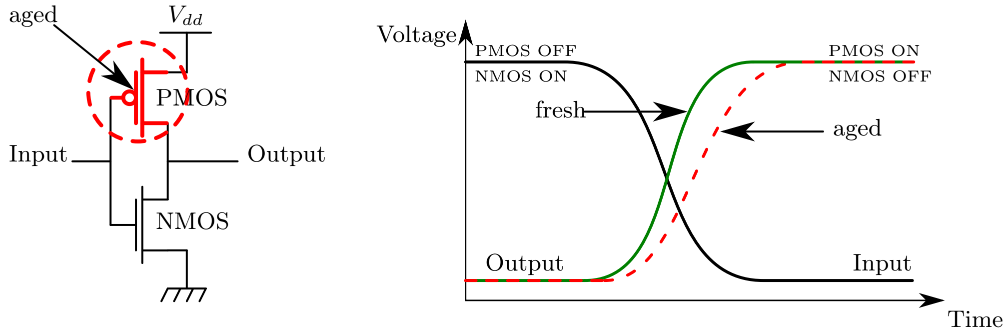


Figure 2.2.: Illustration of the delay increase (here: output rise delay) due to aging for a CMOS inverter (here: V_{th} of the PMOS is degraded due to aging)

A very important relation for transistors is the one between the threshold voltage and the switching speed (delay d) of a transistor [87]. This relation can be approximated using the following power model:

$$d \approx \frac{C}{(V_{dd} - V_{th})^{\sigma}} \sim (V_{dd} - V_{th})^{-\sigma}, \tag{2.1}$$

where C and σ are technology dependent constants with σ being around 1.3 in recent technology nodes [88]. Due to this relationship, an increase in threshold voltage (e.g. due to aging → Section 2.3) will impair the switching speed of the affected transistor, and thus also the gate delay. This circumstance is illustrated in Figure 2.2 for a simple CMOS inverter and its output rise delay. If the threshold voltage of the PMOS degrades, the PMOS becomes slower and thus the output rise delay increases, as the PMOS transistor is the responsible transistor to perform a rise transition of the output. Similarly, if the NMOS is affected, the output fall delay will increase, since in this case the NMOS transistor is responsible. At circuit-level, such increasing gate delays cause an increase in the path delays as depicted in Figure 2.3. Thus, the *slack*, i.e. the time difference between the arrival time of the correct signal values at the circuit outputs and the clock edge, decreases over time. Eventually, due to this process, some output signals may not switch to the correct values in time, i.e. before the clock edge arrives, which means that wrong data is captured in memories or released via primary outputs. Consequently, the correct functionality of the system is affected.

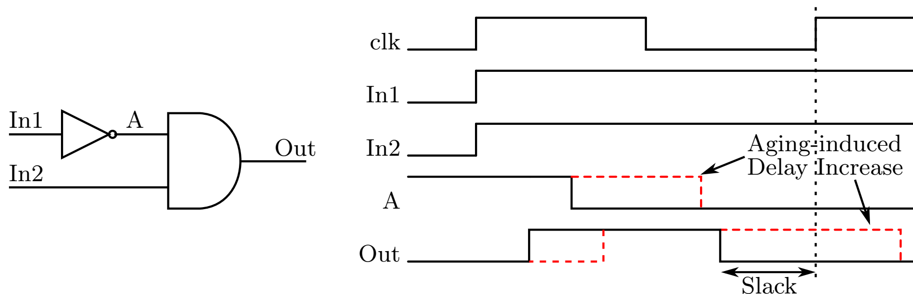


Figure 2.3.: Illustration of the circuit delay increase due to aging for a simple circuit

2.2. Reliability: Basic Notions and Challenges

If the correct functionality of a system is impaired, it also means that its reliability is affected. In order to quantify the reliability of a system, a statistical definition is commonly used. To understand this definition, it is important to first explain and differentiate various reliability threats, namely *faults*, *errors* and *failures* [89].

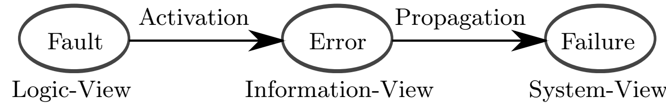


Figure 2.4.: Fundamental chain of reliability threats

While fault, error and failure have very similar meanings in the common speech, their notions are considerably different in the field of reliable computing. A *fault* is the logical representation of a defect. For instance, in case of transistor aging, the defect is the increased threshold voltage and its logic representation, i.e. the fault, is the delay increase of the affected gate (see Figure 2.2). In general, defects can occur in hardware (e.g. broken interconnects, crosstalk between wires, or a damaged gate oxide) as well as software (e.g bug, input mistakes), they can be permanent (e.g. broken interconnect) or transient (e.g. crosstalk), and can be caused by natural phenomena (e.g. crosstalk, transistor aging) or can be human made (design mistake).

Upon activation of a (previously dormant) fault (invoking the faulty component or uncovering the fault by an appropriate system state), an *error* is produced. Thus, an error is the manifestation of a fault in form of a wrong bit value, system state or information. In case of transistor aging, the error occurs if the delay increase (fault) leads to a timing violation and hence to a wrong bit value (error). In fact, not all faults lead to errors, as masking effects (e.g. a path with increased delay is not sensitized) can prevent faults to become active.

Eventually, when the error leads to a malfunction of the system (e.g. wrong program output), it is called *failure*. Similar to the activation of a fault, not every error propagates and becomes noticeable at system-level (i.e. becomes a failure) as illustrated in Figure 2.4, e.g. if a wrong intermediate result is not used anymore and thus does not affect the final program output. This causal chain is also depicted in Figure 2.5 for the case of transistor aging.

2.2.1. Notion of Reliability

Based on the definition of a failure, one can define reliability as a statistical measure. In this regard, the following definition of reliability is nowadays very common [89].

Definition 2.2.1 (Reliability)
The reliability $R(t)$ of a computing system is the probability that the system correctly performs its required function under the specified conditions for a given time period t, i.e. no failure occurs during this time period.

For instance, if the failure rate λ of a system, which is defined as $\lambda = -\frac{\mathrm{d}}{\mathrm{d}t}R(t) \cdot \frac{1}{R(t)}$, is constant over time, the system reliability $R(t)$ follows an exponential function, i.e. $R(t) = e^{-\lambda t}$. However, in case of transistor aging the failure rate is not constant over time, and thus the system reliability with respect to transistor aging cannot be obtained that easily.

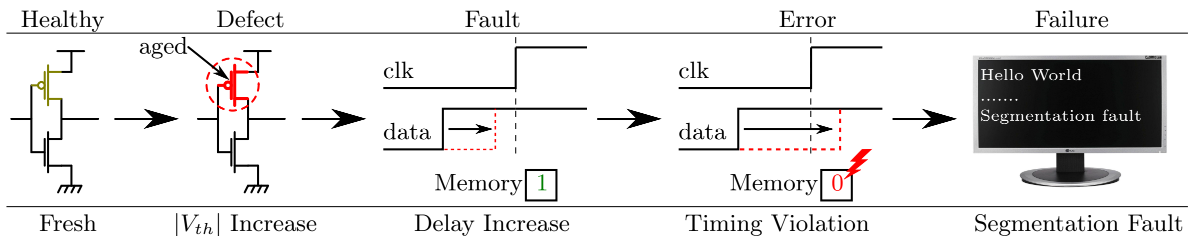


Figure 2.5.: Chain of fault, error and failure in case of transistor aging

In fact, the failure rate follows an exponential function (no failures in early life phases and exponentially increasing failure rate in late life phases) that has to be obtained experimentally or by simulation.

To quantify the reliability of a system, the *Mean Time To Failure (MTTF)* is often used. It stands for the length of time the system is expected to last in operation without having a failure [89]. Therefore, its mathematical definition is:

Definition 2.2.2 (Mean Time To Failure)
$\mathrm{MTTF} = \int_0^\infty R(t)\mathrm{d}t$

For example, if the failure rate λ is again constant, the MTTF of the system is $\frac{1}{\lambda}$.

In the scope of transistor aging the TTF (time to failure) is the time until a failure due to an aging-induced timing violation occurs. Assuming, that all timing errors result in a failure, the TTF can be approximated by the time until timing violations start to appear. This TTF estimation is also used in this thesis. Based on this TTF definition, the MTTF can be obtained as explained in Chapter 3.2.

The reliability of a computing system can be improved in various ways that can be categorized into four groups according to [89]: *Fault prevention*, *fault forecasting*, *fault removal*, and *fault tolerance*. During the design and manufacturing phase, fault prevention techniques are intended to prevent faults by using better materials or rigorous design rules. In fact, an absolute fault avoidance is impossible. Therefore, fault removal approaches aim at identifying and removing possible faults by employing verification and test techniques which can be used during the design, manufacturing and operational phase. The fault forecasting is applied to predict the occurrence of faults and their likelihood by analyzing a diversity of system states. Finally, when a fault becomes active (i.e. error) fault tolerance techniques are employed to first detect the resulting errors and then to correct the erroneous system state. In this thesis, four different approaches to mitigate the impact of accelerated transistor aging are presented that belong to the categories of fault prevention (longer lifetime by design modifications) as well as fault forecasting combined with fault removal (prediction indicates the likelihood of a timing violation and based on the result, countermeasures are activated).

Beside reliability there are also various other attributes namely availability, safety, confidentiality, integrity and maintainability that describe how “trustable” the usage and output of a computing system is. The root of all these attributes is called *dependability* which is the “ability of a system to deliver its intended behavior that can justifiably be trusted” [89]. Together with the already introduced categories for reliability (dependability) threats and means to improve reliability (dependability), these attributes form the dependability tree depicted in Figure 2.6.

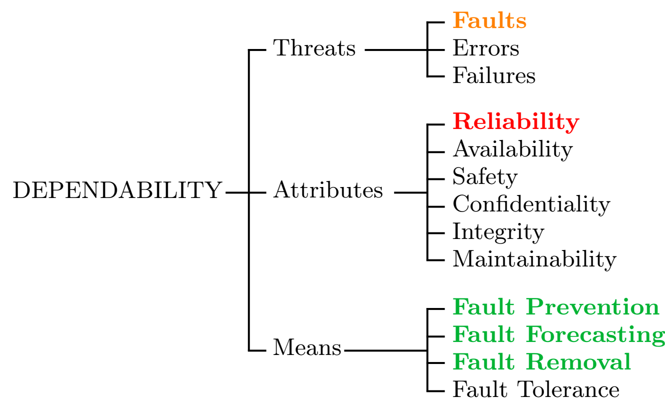


Figure 2.6.: Dependability tree [89] (highlighted parts indicate the field of this thesis)

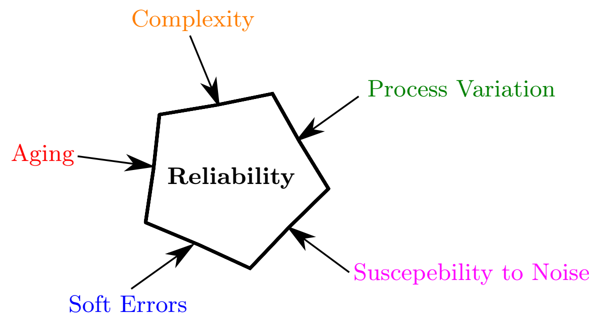


Figure 2.7.: Illustration of important reliability challenges in nanoscale CMOS technology nodes

2.2.2. Sources of Unreliability

The reliability of microprocessors fabricated in nanoscale CMOS technology nodes is threatened by various phenomena [10] as depicted in Figure 2.7. The first source for faults and defects is the *increasing complexity* at device-, circuit-, and architecture-level of modern microprocessors with billion transistors and a very rich feature set. The result is an increasing number of design flaws and electrical bugs in the design that are very hard to activate, isolate and debug. In current systems, the silicon debug already requires more than 50 % of the total time-to-market [90] while this portion was only 17 % in 2002 [91]. Moreover, on the way to manycore processors, even more integrated features and smaller device sizes, the complexity of silicon debug will further increase [92]. Consequently, bugs in the final product are almost inevitable, although rigorous verification and test procedures are performed. This is underlined by the errata documents of modern processors that often list more than 200 known bugs [93]. Very famous examples of such escaping bugs due to the increasing complexity are Intel's division bug[1] in the Pentium processor (1994) [94], AMD's bug in the Translation Lookaside Buffer (TLB)[2] in the K10 processor (2007) [95] and Intel's recent issue in the transactional memory extensions (TSX)[3] in 2014 [96].

Another growing reliability issue is *process variation*. Due to the ongoing transistor miniaturization an exact manufacturing process becomes increasingly challenging. As a result, in a microprocessor with billions of transistors, no transistor is exactly like the other. Due to lithography issues and dopant fluctuation, channel lengths differ from transistor to transistor, and variations of the gate oxide thickness as well as threshold voltage are on the agenda [10]. Such variations could be global (chip to chip) and local (core to core). At system-level these problems manifest in considerable performance and power consumption variations of different manufactured parts. For example, Intel reported for a 100 million transistor 80-core processor fabricated in a 65 nm technology node that the core-to-core variation leads to a frequency spread of 28 % at 1.2 V and 62 % at 0.8 V between the fastest and slowest cores on the die [97]. This means that some cores can be of high quality, making high clock frequencies or low power consumption possible, while others may even not be operational at all.

In addition to errors due to imperfect design, another major reliability threat is posed by *radiation-induced soft errors* at runtime [98, 99]. Cosmic radiation in form of neutrons and alpha particles from packaging material can cause a charge disturbance inside the transistor that was hit. If this charge disturbance was large enough, it can flip the signal state in form of a transient pulse which can lead to faulty data being latched or stored in a memory cell. Depending on how many transistors are affected, even multiple stored bits can be affected (Multi Bit Upset) [100]. Since the damage to the circuit is not permanent, these errors are called

[1] Errata 20 [94]: "Slight precision loss for floating-point divides on specific operand pairs"
[2] Errata 298 in [95]: "L2 Eviction May Occur During Processor Operation To Set Accessed or Dirty Bit"
[3] Errata HSW.136 in [96]: "Software Using Intel TSX May Result in Unpredictable System Behavior"

"soft". Nevertheless, these errors are a major challenge, in particular as the vulnerability of digital circuits to radiation-induced soft errors increases considerably with every new technology node (e.g. 8x higher error rate in 40 nm than in 130 nm) [98, 101, 102]. Therefore, hardening approaches and sophisticated error detection and correction techniques typically in form of parity bits and error correcting codes (ECC) are employed in modern microprocessors [37].

Beside the transient soft errors also the intermittent[4] errors due to the *increasing susceptibility to electric noise* are of growing importance. Crosstalk and coupling effects between wires can cause wrong signal states and the drop of the supply voltage caused by active circuits can lead to wrong computation output [10]. To alleviate these issues more conservative design rules need to be employed, and more sophisticated power delivery networks have to be designed. In particular, the power delivery network is very important as it is not uncommon for a modern microprocessor with a die area of 300 mm^2 to burn more than 150 Watts [103]. In fact, due to the end of the *Dennard* scaling model [104] (supply voltage and threshold voltage are lowered by about the same factor as the device feature size), the power density is increasing with every new technology generation. Because of that, the power density of a modern microprocessor exceeds already that of a common household iron [103]. This increasing power density makes thermal hotspots more likely to occur, which in turn necessitates more sophisticated cooling solutions. In addition, it considerably affects reliability, especially thermally accelerated effects such as transistor and interconnect wearout.

Finally, the *accelerated wearout* of interconnects and transistors in nanoscale CMOS technology nodes considerably reduces the operational lifetime of microprocessors, and thus is another important reliability challenge [31, 32, 105]. Physical effects such as Bias Temperature Instability (BTI) or Hot Carrier Injection (HCI) lead to an increasing switching delay of the transistors resulting in slower cores [10]. Moreover, Time Dependent Dielectric Breakdown (TDDB) and electromigration will cause sneaking deaths of transistors and wires [10]. Hence, aging is a time-dependent effect that will eventually lead to the death of the system. Furthermore, the system state changes over time, since not all transistors age at the same rate, as they are exposed to various temperature and activity conditions. Therefore, the nature of electrical bugs in the system, and how they result in (intermittent) failures, changes over time.

To deal with the aforementioned reliability challenges a common approach is to add timing margins in form of guardbands to the design to reduce the likelihood of failures (i.e. the clock frequency is reduced). The breakdown of such a timing margin for the different fault categories for the IBM Power7+ microprocessor [8] is depicted in Figure 2.8. Obviously a significant amount of the budget is put to tackle the impact of accelerated aging (around 15 %). Together with the fact that aging will become more severe in future technology nodes as shown in Figure 1.3, it is a necessity to find more efficient means than purely overdesigning the hardware (i.e. adding safety margins) to mitigate wearout. Therefore, this thesis proposes various novel cross-layer approaches that efficiently reduce the impact of transistor aging.

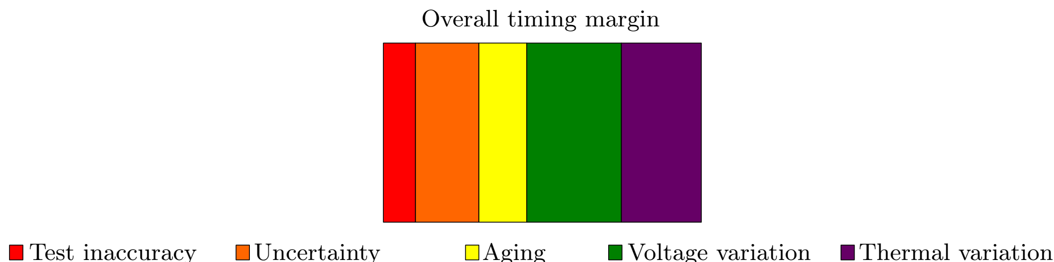


Figure 2.8.: Guardband decomposition for the IBM Power7+ [8]

[4]transient errors are non-permanent errors caused by environmental issues while intermittent errors are due to circuit-internal issues

2.3. Background on Transistor Aging

Transistor aging is a rather old phenomenon that was already under discussion in the 1960s and following decades [51, 52, 106–108]. However, at that time transistor aging did not reduce the microprocessor lifetime or its reliability to an alarmingly low value. In contrast, due to CMOS device feature sizes of 32 nm and less, transistor aging is nowadays a severe reliability issue, as it considerably reduces the microprocessor lifetime, i.e. microprocessors build in recent and future technology nodes start to fail sooner. Therefore, manufactures need to add greater timing margins to their designs, to guarantee the required lifetime. For instance, the experimental study performed in [55, 57] show that the delay degradation can be more than 10 % for a runtime of 3 years[5].

The three major physical phenomena that cause transistors to age are *Bias Temperature Instability (BTI)* , *Hot Carrier Injection (HCI)* and *Time Dependent Dielectric Breakdown (TDDB)* [10]. The first two effects cause a gradual increase of the transistor threshold voltage and on-current, which leads to an increasing circuit delay over time. In other words, BTI and HCI make a circuit, for example a microprocessor, slower over time. Because of that it could be necessary to reduce the clock frequency of a microprocessor after a certain amount of time to maintain a fault-free execution. Also TDDB can affect the circuit speed, in form of a “soft” breakdown. In this case, small defects are created inside the gate oxide which reduce the switching speed of the affected transistor. However, if a conducting path through the oxide is formed, the affected transistor will be no more functional, which is called “hard” breakdown [10].

This thesis addresses the problem of gradually decreasing circuit speed due to threshold voltage degradation and thus deals with BTI and HCI as accelerated transistor aging phenomena. TDDB in form of “soft” breakdowns is not the primary focus of this thesis, since its impact on the circuit delay is considerably smaller than that of the two other phenomena [109, 110]. In the following, BTI and HCI are introduced in detail and the fundamental models used in thesis are explained.

2.3.1. Bias Temperature Instability

Physical Mechanisms of BTI

BTI is a wearout phenomenon that, as the name tells, occurs under certain gate-source bias conditions and is accelerated under elevated temperature. It can be separated into the very similar phenomena *positive* and *negative* BTI (PBTI and NBTI), affecting NMOS and PMOS transistors, respectively [50, 111]. While NBTI is known since the 1960s [106] to cause a degradation of the switching speed of affected transistors, PBTI received only little attention [112]. This, however, changed considerably with the introduction of metal gates in combination with high-κ gate oxides for which stacks of Hafnium and other materials replace the classical SiO_2 compound for the gate oxide to improve the controllability of the channel via the gate[6] [50]. Using these high-κ metal gates the effect of PBTI became comparable to NBTI [50] and thus nowadays both phenomena need to be considered.

[5]Please note that these numbers can vary from one chip manufacturer to another or from one technology to another and are also dependent on the application domain (e.g. low power, long mission time, health care, aerospace, consumer devices) the microprocessor is designed for.

[6]The gate oxide acts as a dielectric in a capacitor formed by channel and gate. Thus, a high permittivity for this dielectric is required to achieve a good controllability of the channel (i.e. high capacity). Because of that the traditional SiO_2 gate oxide became thinner and thinner with every new generation. However, at some point it became too thin (less than 2 nm) and various parasitic effects became visible (e.g. very high leakage current through the gate oxide). To be able to increase the oxide thickness again, high-κ materials (with better permittivity than pure SiO_2) replaced the SiO_2 compound.

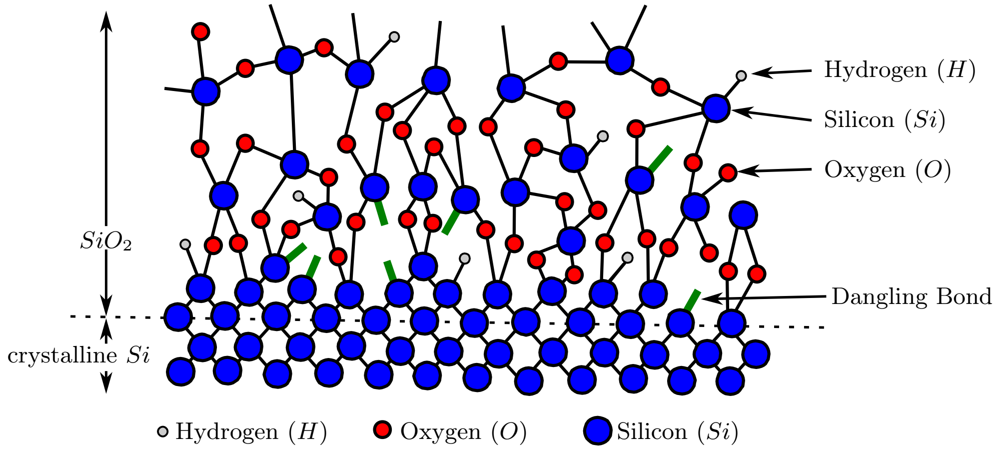


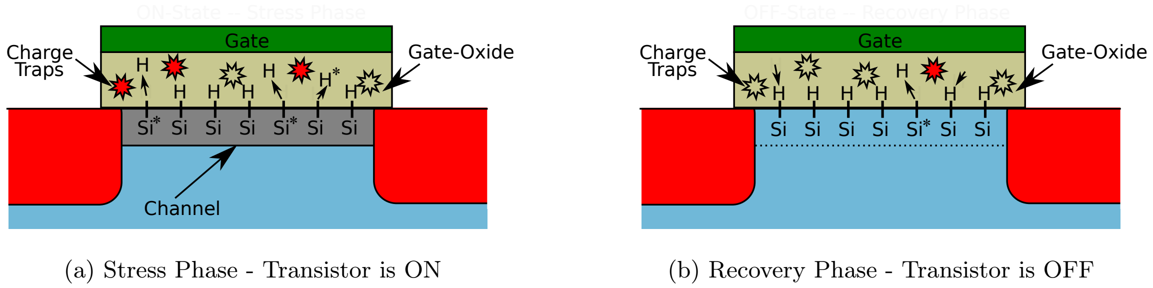
Figure 2.9.: Illustration of the $Si - SiO_2$ interface showing dangling bonds and *Si-H* bonds

Although, as mentioned before, BTI is a rather old phenomenon, the underlying physical and chemical processes are not yet fully understood and heavily debated. This is due to the fact that it cannot be observed directly, as it is happening inside transistors with nanometer dimensions. Thus, only indirect explanations based on measurements are possible. However, these are very challenging, as it is very hard to first provoke the BTI effect, and then to directly measure its consequences (e.g. threshold voltage) without changing the status of the transistor due to the fast dynamics of BTI [113, 114]. Nowadays, the most prevalent explanations are based on *trapping/detrapping* and *reaction/diffusion* processes [48, 114–116]. In both theories, the origin of the BTI effect is the manufacturing process of the gate oxide [43]. Therefore, this process is explained next for a conventional SiO_2 gate oxide. Nevertheless, the same principles apply also for a high-κ gate oxide and similar observations can be made.

The gate oxide is obtained using a dry oxidation process to let the amorphous SiO_2 grow thermally on top of the silicon (crystalline) substrate. However, there is a considerable lattice mismatch between the SiO_2 and the crystalline Si. Although a major part of this mismatch can be compensated by the very flexible *Si-O-Si* bonds, some unbound silicon atoms are remaining at the interface. These unbound atoms are called *dangling bonds*. In fact, at the gate oxide interface of a transistor with 100 nm gate length and a gate width of 1 μm the density of dangling bonds after the completion of the oxidation process is said to be around $10^{12}\,\text{cm}^{-2}\text{eV}^{-1}$ [117]. This translates to 1000 dangling bonds, which means that such a transistor cannot operate properly, as dangling bonds are active interface states (not all four valence electrons of Si are included in bonds) that degrade the important transistor parameters such as threshold voltage, on-current and carrier mobility. Therefore, to reduce the number of dangling bonds, hydrogen (H) is employed to passivated the dangling bonds by forming *Si-H* bonds [118]. As a result, the dangling bond density can be reduced to $10^{10}\,\text{cm}^{-2}\text{eV}^{-1}$, which is good enough. Hence, the interface between the gate oxide and the silicon substrate looks similar to the illustration in Figure 2.9 [119]. As it can be seen, beside *Si-O* and *Si-H* bonds, also other bonds such as *Si-Si* can occur in the interface between substrate and gate oxide, and even other materials may be deposited during the manufacturing process due to impurities.

The *Si-H* bonds as well as the remaining dangling bonds, which act as charge *traps*, are the source of the BTI effect. According to both BTI theories (trapping/detrapping and reaction/diffusion), this effect can be split into two phases depending on the gate bias, i.e. whether the transistor is ON or OFF (from there comes the “bias” in BTI).

- ***Stress Phase – Transistor is ON:*** When the transistor is ON, various things can happen as depicted in Figure 2.10(a). On the one hand, *Si-H* bonds can be torn apart due to the applied vertical electric field and their low binding energy. On the other hand,



(a) Stress Phase - Transistor is ON (b) Recovery Phase - Transistor is OFF

Figure 2.10.: Illustration of the two BTI phases (stress and recovery) with trapping/detrapping and reaction-diffusion (colored traps have captured a charge carrier)

charges that move through the channel can react with the Si-H bonds and break these. In both cases, the results are additional dangling bonds (not fully bound silicon atoms), and free hydrogen atoms as well as hydrogen molecules (created by combination of two hydrogen atoms) that diffuse into the gate oxide. In addition, also hydrogen ions H^* and silicon ions Si^* can be created. Beside these reaction and diffusion processes also trapping effects can occur, since the dangling bonds as well as the free hydrogen atoms act as traps and can capture charges creating Si^* or H^*. In this regard, depending on the Fermi-level[7] of these traps, either holes or electrons are collected. As a result of all these processes, the $|V_{th}|$ of the affected transistors increases.

- ***Recovery Phase – Transistor is OFF:*** When the transistor is OFF, some of the Si^* and H^* ions emit their captured charges (detrapping) and some of these even undergo a new bond with H atoms that diffuse back, as illustrated in Figure 2.10(b). Consequently, $|V_{th}|$ decreases again (see Figure 2.11). However, due to emission and back-diffusion times that range from fractions of a millisecond to hours or days [114, 122], the initial shift cannot be entirely compensated leading to a gradual increase of $|V_{th}|$ over time as shown in Figure 2.11. In this regard, the overall V_{th} shift, i.e. the wearout rate, depends on several aspects, such as technology parameters, supply voltage (i.e. the vertical electric field), temperature (from there comes the "temperature" in BTI) and *duty cycle* δ, i.e the ratio of stress to total time (see Figure 2.12).

Please note that the given explanations for the commonly observed behavior (Figure 2.11 and Figure 2.12) are still under discussion, since the exact physical mechanisms and electrochemical reactions are not yet fully understood. Thus, these may change with new technologies. For current technologies, it seems that the trapping/detrapping process is mainly responsible for the fast and large V_{th} changes at the beginning of each stress or recovery phase, whereas the

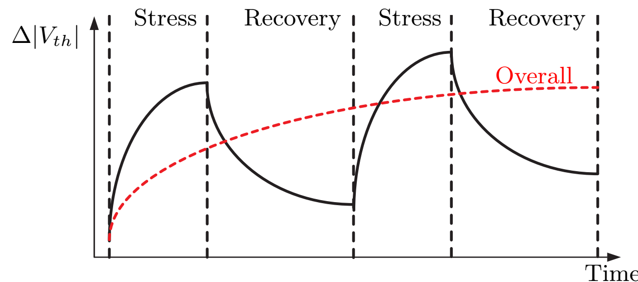
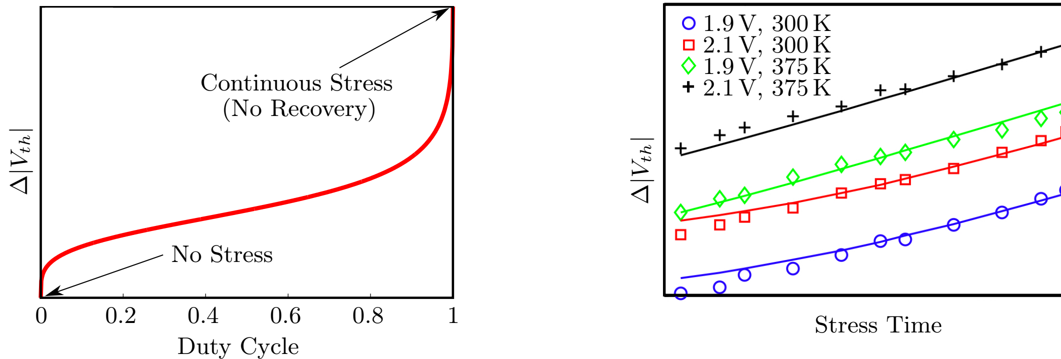


Figure 2.11.: Illustration of a sequence of stress and recovery phases and their impact on $|V_{th}|$ based on the experimental data in [122]

[7]Briefly spoken the Fermi-level of a trap describes the likelihood with which it captures a hole or an electron [120, 121]



(a) Illustration of the duty cycle impact based on the experimental data in [123, 124]

(b) Illustration of a V_{dd} and temperature impact based on the experimental data in [125]

Figure 2.12.: Illustration of the duty cycle, supply voltage (V_{dd}, i.e. vertical electric field) and temperature impact on BTI-induced V_{th} degradation

reaction-diffusion process is more important for the long-term behavior (over several weeks), as explained in [115].

Modeling the V_{th} shift due to BTI

Since the focus of this thesis is on the long-term (i.e. more than a few seconds) delay degradation of transistors due to BTI, a reaction-diffusion based model for the threshold voltage shift is applied, which is proven to accurately capture the device degradation for a wide range of measurements [126]. Although this model was only verified for NBTI, we also employ it to model the PBTI effect. This is reasonable, as both phenomena have a very similar impact on the affected transistors as laid out in [111]. In combination with the power law in Equation (2.1) this model allows us to model the delay degradation of single transistors, gates and at higher abstraction-levels even entire logic blocks.

In the following, we will now introduce the reaction-diffusion model which we use in this thesis. It is based on the model developed in [126] and for the matter of simplicity we focus now on PMOS transistors, i.e. NBTI.

Stress Phase:

According to [127] the generation rate of interface traps N_{IT} follows the equation:

$$\frac{\mathrm{d}N_{IT}}{\mathrm{d}t} = k_F(N_0 - N_{IT})P - k_B N_H N_{IT}, \tag{2.2}$$

where N_0 is the initial concentration of Si-H bonds, P is the charge (holes) concentration and N_H is the concentration of hydrogen at the interface between channel and gate oxide. Moreover, k_F and k_B are the forward and backward reaction rates, which means that k_F is the rate of trap generation (i.e. reaction of a hole and an Si-H bond) and k_B is the rate of trap passivation (i.e. recombination of Si and H).

At the beginning of a stress phase, the trap generation rate is very small as laid out in [127]. Thus, $\frac{\mathrm{d}N_{IT}}{\mathrm{d}t} \approx 0$ and $N_{IT} \ll N_0$ which simplifies Equation (2.2) to:

$$N_H N_{IT} \approx \frac{k_F}{k_B} P N_0. \tag{2.3}$$

With the continuation of the stress period, more and more free H atoms are created that diffuse into the gate oxide layer. In this phase, the reaction balance is governed by the diffusion process,

which itself is controlled by the gradient of the free hydrogen density N_H. According to *Fick's second law* [128] the diffusion process can be described as

$$\frac{\mathrm{d}N_H}{\mathrm{d}t} = D_H \frac{\mathrm{d}^2 N_H}{\mathrm{d}x^2}. \tag{2.4}$$

In this regard, D_H follows the *Arrhenius equation*, i.e. it has an exponential dependency on the temperature T and the activation energy E_a of the chemical reaction:

$$D_H = \gamma e^{-E_a/kT}, \tag{2.5}$$

where k is the *Boltzmann constant* and γ is the constant prefactor.

Using the approximated diffusion profile [126] the total number of interface traps at time t can now be approximated:

$$N_{IT}(t) \approx \left((1+\epsilon)\, t_{ox} + \sqrt{D_H t} \right) N_H. \tag{2.6}$$

t_{ox} is the oxide thickness of the affected transistor and $\epsilon < 1$ is a fitting parameter that has to be determined experimentally. By combining the Equations (2.6) and (2.3) it follows that:

$$N_{IT}(t) \approx \left(\frac{k_F}{k_B} P N_0 \right)^{1/2} \left((1+\epsilon)\, t_{ox} + \sqrt{D_H t} \right)^{1/2}. \tag{2.7}$$

Knowing that $\Delta V_{th} = qN_{IT}/C_{ox}$ (q is the elementary charge and C_{ox} is the gate oxide capacitance), $P = C_{ox}(V_{gs} - V_{th})$ and that $N_0 k_F / k_B$ is a function of the vertical electric field E_{ox} in the gate, the threshold voltage shift during a single stress period that lasts for the time period t can be estimated [126]. It holds with $n = 1/4$ that:

$$\begin{aligned} \Delta V_{th}(t) &= A \left((1+\epsilon)\, t_{ox} + \sqrt{D_H(t)} \right)^{2n} \\ A &= \frac{q}{C_{ox}} \left(\left(K \exp\left(\frac{E_{ox}}{E_0} \right) \right)^2 C_{ox}(V_{gs} - V_{th}) \right)^{1/(2n)} \end{aligned} \tag{2.8}$$

Here, K and E_0 are technology dependent parameters that describe the relation between the charge density P and the electric field.

Accordingly, if at time t_0 a threshold voltage shift of $\Delta V_{th}(t_0)$ exists, the shift at time t can be calculated as follows:

$$\boxed{\Delta V_{th}(t) = \left(A^{1/(2n)} \sqrt{D_H} \left(\sqrt{t} - \sqrt{t_0} \right) + \Delta V_{th}(t_0)^{1/(2n)} \right)^{2n}}. \tag{2.9}$$

Recovery Phase:

During the recovery phase, as the transistor is OFF, no holes are present to create new interface traps. However, the hydrogen species can diffuse back to the interface and anneal existing traps. This back-diffusion process and with it the number of annealed traps N_{IT}^A can be mathematically expressed as:

$$N_{IT}^A(t) \approx \left(\xi_1 t_{ox} + \sqrt{\xi_2 D_H (t - t_1)} \right) N_H. \tag{2.10}$$

ξ_1 and ξ_2 are the back-diffusion constants, t_e is a technology dependent parameter and t_1 is the point in time at which the recovery phase started. Combining this equation with Equation (2.6) it follows that

$$N_{IT}^A(t) \approx N_{IT}(t) \left(\frac{\xi_1 t_{ox} + \sqrt{\xi_2 D_H (t - t_1)}}{(1+\epsilon) t_{ox} + \sqrt{D_H t}} \right). \tag{2.11}$$

By knowing that the total number of interface traps at time t is computed according to

$$N_{IT}(t) = N_{IT}(t_1) - N_{IT}^{A}, \quad (2.12)$$

the overall threshold voltage shift at time t after a recovery phase that started at time t_1 can be obtained as follows:

$$\boxed{\Delta V_{th}(t) \leq \Delta V_{th}(t_1)\left(1 - \frac{\xi_1 t_{ox} + \sqrt{\xi_2 D_H (t - t_1)}}{2t_{ox} + \sqrt{D_H t}}\right).} \quad (2.13)$$

Long-Term Dynamic Behavior:
Based on the models for the stress and recovery phases (Equation (2.9) and Equation (2.13)), which were validated in [126, 129], the long-term dynamic effect of BTI on the threshold voltage can be estimated. Such an estimation of the long-term behavior is required, as it is impossible to use the independent stress and recovery models for high performance processors in which the transistors undergo several billion stress and recovery periods within less than one hour. For this estimation it is assumed that the total time t can be divided into stress-recovery cycles with the length T_{cyc} (i.e. $t = mT_{cyc}$). For each cycle, the duty cycle is $\delta \leq 1$. Thus, after l cycles it holds:

$$\Delta V_{th,s,l} = \left(A^{1/(2n)}\sqrt{D_H}\left(\sqrt{\delta T_{cyc} + (l-1)T_{cyc}} - \sqrt{(l-1)T_{cyc}}\right) + \Delta V_{th,r,l-1}^{1/(2n)}\right)^{2n} \quad (2.14)$$

$$\Delta V_{th,r,l} \leq \Delta V_{th,s,l}\left(1 - \frac{\xi_1 t_{ox} + \sqrt{\xi_2 D_H (1-\delta) T_{cyc}}}{2t_{ox} + \sqrt{D_H l T_{cyc}}}\right) \quad (2.15)$$

In this regard, $V_{th,s,l}$ is the threshold voltage shift after the l-th stress cycle and similarly, $V_{th,r,l}$ is the corresponding V_{th} shift after the following recovery phase. Moreover, since $\delta \leq 1$ an upper bound for Equation (2.15) can be obtained using the inequality $\sqrt{\delta + (l-1)} - \sqrt{l-1} \leq \sqrt{\delta}$:

$$\Delta V_{th,s,l} \leq \left(A^{1/(2n)}\sqrt{D_H}\sqrt{\delta T_{cyc}} + \Delta V_{th,r,l-1}^{1/(2n)}\right)^{2n} \quad (2.16)$$

Using the Equation (2.16) and Equation (2.15) iteratively, it follows that:

$$\begin{aligned} \Delta V_{th,s,l+1} &\leq \left(A^{1/n} D_H \delta T_{cyc}\right)^n \left(\sum_{i=1}^{l}\left(\prod_{j=l-i+1}^{l} \beta_j^{1/(2n)}\right)\right)^{2n} \\ \beta_j &= 1 - \frac{\xi_1 t_{ox} + \sqrt{\xi_2 D_H (1-\delta) T_{cyc}}}{2t_{ox} + \sqrt{D_H j T_{cyc}}} \end{aligned} \quad (2.17)$$

For this equation an upper bound can be obtained by using the fact that $\beta_{j_1} \leq \beta_{j_2} \leq 1$ for $j_1 \leq j_2$. This upper bound is of the form:

$$\Delta V_{th,s,l+1} \leq \left(\sqrt{A^{1/n} D_H \delta T_{cyc}}\left(\sum_{i=1}^{l}\left(\beta_l^{1/(2n)}\right)^i\right)\right)^{2n} \quad (2.18)$$

Finally, by employing the inequality for the geometric progression $\sum_{i=1}^{l}\left(\beta_l^{1/(2n)}\right)^i$ a closed form to estimate the threshold voltage shift at time t can be obtained:

$$\boxed{\begin{aligned} \Delta V_{th}(t) &\leq A_{BTI}\left(\frac{\sqrt{D_H \delta T_{cyc}}}{1 - \beta^{1/(2n)}(t)}\right)^{2n} \\ \beta(t) &= 1 - \frac{\xi_1 t_{ox} + \sqrt{\xi_2 D_H (1-\delta) T_{cyc}}}{2t_{ox} + \sqrt{D_H t}}. \end{aligned}} \quad (2.19)$$

$\Delta V_{th}(t)$	$\leq$	$A_{BTI}\left(\frac{\sqrt{D_H \delta T_{cyc}}}{1-\beta^{1/(2n)}(t)}\right)^{2n}$	Long-term BTI model for the V_{th} shift
$\beta(t)$	$=$	$1-\frac{\xi_1 t_{ox}+\sqrt{\xi_2 D_H(1-\delta)T_{cyc}}}{2t_{ox}+\sqrt{D_H t}}$	
A_{BTI}	$=$	$\frac{q}{C_{ox}}\left(\left(K\exp\left(\frac{E_{ox}}{E_0}\right)\right)^2 C_{ox}(V_{gs}-V_{th})\right)^{1/(2n)}$	
n	$=$	0.25	
δ	$=$	stress time to total time	Duty Cycle
D_H	$=$	$\gamma e^{-E_a/kT}$	Diffusion constant
γ	$=$	10^8, $E_a = 0.13\,\text{eV}$, $k = 8.6174 \cdot 10^{-5}\,\text{eV/K}$	
ξ_1	$=$	1.8, $\xi_2 = 0.5$	Back-diffusion constants
t_{ox}	$=$	2.2 nm for PMOS, 2.0 nm for NMOS	Oxide thickness
q	$=$	$1.602 \cdot 10^{-19}\,\text{C}$	Elementary charge
C_{ox}	$=$	$t_{ox}^{-1} \cdot 3.45 \cdot 10^{-22}\,\text{F/nm}$	Oxide capacitance
E_{ox}	$=$	$\frac{V_{gs}-V_{th,0}}{t_{ox}}$	Electric field
E_0	$=$	0.335 V/nm	Constant
$V_{th,0}$	$=$	0.317 V for PMOS, 0.271 V for NMOS	Default threshold voltage
		T_{cyc}	stress-recovery cycle time (often clock frequency)
		T	Temperature
		K	Electric field influence

Table 2.1.: Summary of the BTI model employed in this thesis (parameters are obtained from the TSMC 65 nm general purpose library and [61, 126])

This upper bound for the long-term threshold voltage shift was validated against real data in [129] and is used in this thesis as BTI model, since it is capable to capture the duty cycle influence as well as the impact of temperature and supply voltage very well. Moreover, together with the power model in Equation (2.1), this BTI model allows to estimate the long-term delay degradation due to BTI. Therefore, Table 2.1 summarizes the most important facts and presents the constant parameter values based on the TSMC 65 nm general purpose library. This setup is used throughout this thesis. Please note that we used $n = 1/4$ which is valid for diffusion processes that are dominated by free hydrogen atoms. If H_2 is more important the final model and its derivation is still valid, however in this case $n = 1/6$ [126].

It is important to note that the influence of the electric field K is a kind of proportionality constant in the BTI model. Since its exact influence can only be determined experimentally, we set K in such a way, that under worst-case conditions, i.e. permanent stress and high temperature (100 °C), the delay degradation is at most 10 % after 3 years. As laid out in [55, 56] this assumption is very reasonable for recent technology nodes.

Furthermore, please note that due to changing technologies and new insights about BTI, the model may change in future. Nevertheless, the techniques and methodologies developed and presented in this thesis will still be valid and can be employed for aging mitigation. This is due to the fact that the proposed approaches are not based on the fundamental transistor-level models rather than on the observation, that transistors become slower over time due to stress, elevated temperature or high supply voltage. Thus, with new models, the exact numbers may change, but the overall approaches will still hold.

2.3.2. Hot Carrier Injection

Physical Mechanisms of HCI

Beside BTI also Hot Carrier Injection (HCI), which is often referred to as *Channel Hot Carrier (CHC)* effect, is a critical reliability threat in nanoscale CMOS technology nodes [54]. As the name suggests, the origin of HCI are "hot" charge carriers that travel through the transistor

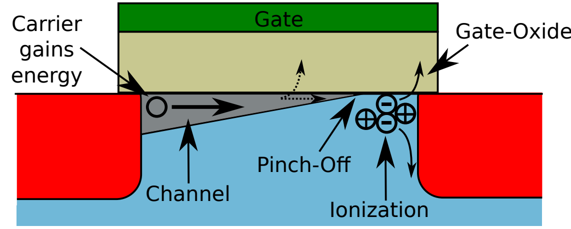


Figure 2.13.: Physical mechanism of HCI including impact ionization and charge injection when the transistor is in the saturation regime

channel. In this regard, "hot" refers to the high kinetic energy of the carriers that is built up due to the strong electric field in and around the channel. If these hot carriers gain enough energy, while traveling through the channel, they can be injected into the gate oxide, or can collide with silicon atoms which can result in an impact ionization creating secondary particles (electron-hole pairs) as illustrated in Figure 2.13. In both cases, charges can be injected into the gate oxide which degrades the important device characteristics, especially the threshold voltage V_{th} [43, 51, 53, 54]. Thus, the degradation is directly associated to the strength of the electric field, i.e. the stronger the electric field along the channel, the faster the transistor degrades. This accelerating electric field is related to the potential between drain and source (V_{DS}). When the transistor operates in its linear regime ($V_{DS} < V_{GS} - V_{th}$), the channel electric field is very small and thus hot carriers are not of importance. However, when the transistor operates in the saturation mode ($V_{DS} > V_{GS} - V_{th}$), i.e. the channel is pinched off (see Figure 2.13), the electric field is very strong and consequently carriers can be injected into the gate oxide or can ionize silicon atoms around the pinch-off point, causing an irreversible degradation of the affected transistor [43, 55, 130]. In this regard, the degradation rate strongly depends on the temperature and supply voltage, as depicted in Figure 2.14.

Although HCI affects both NMOS and PMOS transistors, its impact on NMOS transistors is observed to be considerably more severe [54, 132, 133]. In fact, HCI for PMOS transistors is negligible compared to the effect of BTI [133]. The exact reason for this huge difference between NMOS and PMOS transistors is still under discussion, however it seems that – maybe due to the higher mobility of electrons compared to holes – in NMOS transistors impact ionization occurs much more often than in PMOS transistors and also the affected region around the

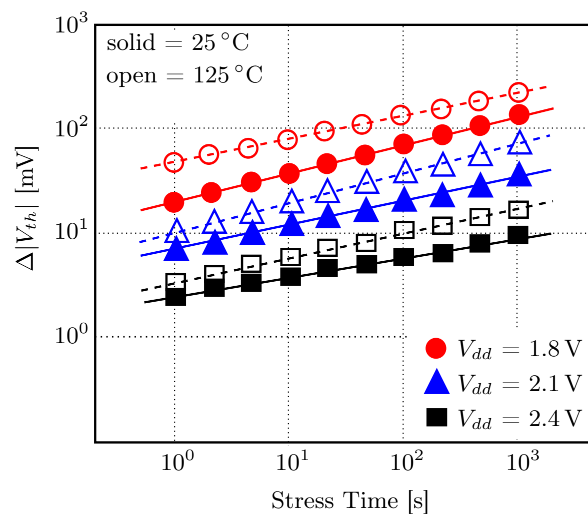


Figure 2.14.: Illustration of the runtime, supply voltage (V_{dd}, i.e. electric field) and temperature impact on HCI-induced V_{th} shift based on data presented in [131]

pinch-off point is much wider [133]. Thus, in NMOS more secondary particles are injected into the gate oxide, resulting in a stronger degradation.

It is important to note that HCI is not a new degradation mechanism that appeared recently. Instead, HCI was already observed for NMOS transistors in the 1980s [51, 52, 107]. The reason for this was the continuous downscaling of transistor dimensions, while the supply voltage was not reduced at the same rate. Thus, the electric field in the channel became very high which eventually resulted in hot charge carriers. From the nineties onwards, the supply voltage started to drop considerably due to arising power and reliability constraints, and thus HCI was less important. However, in recent technology nodes, supply voltage scaling came to an end, as the threshold voltage could not be lowered anymore due to the increasing importance of leakage power. Hence, the electric field is increasing again, and as a result HCI is becoming an important reliability threat [54].

Modeling the V_{th} shift due to HCI

Similar to the long-term BTI model, we will introduce in this section a long-term model for HCI that describes the induced threshold voltage shift over a long period of time (i.e. more than a few seconds).

As pointed out in the previous discussion about the physical processes of HCI, hot carriers are mostly created when the transistor is in the saturation region. In CMOS, this happens only during transitions of the corresponding gate output [130]. Thus, the number of charges trapped in the gate oxide and accordingly the induced threshold voltage shift, is a function of the number of transitions N_{trans}. As laid out in [51], this relation is sub-linear with an exponent of 0.5 for the long-term behavior. Hence, it follows:

$$\Delta V_{th}(t) \sim \sqrt{N_{trans}(t)} \tag{2.20}$$

The number of transitions in the time period t can be approximated by

$$N_{trans}(t) \approx \alpha \cdot f \cdot t, \tag{2.21}$$

where α is the average switching activity of the gate output per clock cycle and f is the clock frequency of the circuit.

Moreover, it is shown in [54] that the threshold voltage shift has an exponential relation to the temperature T, which – similar to BTI – originates from the diffusion process of the charges in the gate oxide (see Equation (2.5)). Thus,

$$\Delta V_{th}(t) \sim \exp\left(\frac{-E_a}{kT}\right), \tag{2.22}$$

where k is the Boltzmann constant and E_a the activation energy for the charge injection into the gate oxide. As a consequence of this relationship, a high temperature accelerates the HCI-induced threshold voltage degradation. In this regard it is important to note that in former technology nodes (above 90 nm), this intuitive temperature relation (higher temperature means more kinetic energy, and thus stronger degradation) was inverted, i.e. a high temperature reduced the impact of HCI. This was due to the fact, that in old technology nodes the mean free path[8] of the hot carriers was shorter than the channel, while nowadays the channel is shorter. Hence, by increasing the temperature, the mean free path increased and as a result fewer collisions between the hot carriers and the silicon atoms occurred. This in turn reduced the number of ionized atoms and free charges such that the threshold voltage shift at higher temperature was smaller than at a lower temperature [54, 134].

[8] mean free path is the average distance traveled by a moving particle between successive impacts

Finally, there is also an exponential dependency between the vertical electric field in the transistor and the threshold voltage shift [51]:

$$\Delta V_{th}(t) \sim exp\left(\frac{E_{ox}}{E_1}\right), \tag{2.23}$$

where E_1 is a technology dependent constant. The reason for this dependency is similar to BTI (see Equation (2.8)), i.e. ionization and injection rate are a function of the vertical electric field.

Putting all these dependencies together results in the V_{th} shift model used in this thesis:

$$\boxed{\Delta V_{th} \approx A_{HCI} \cdot \exp\left(\frac{E_{ox}}{E_1}\right) \cdot \exp\left(\frac{-E_a}{kT}\right) \cdot \sqrt{\alpha \cdot f \cdot t}.} \tag{2.24}$$

In the equation above, A_{HCI} is a technology dependent constant, that describes among other things the impact of the electric field along the channel [126]. However, as the exact value of this constant is not known and can only be determined experimentally, we set A_{HCI} in such a way, that under worst-case conditions, i.e. $\alpha = 1$ and high temperature (100 °C), the delay degradation is at most 10 % after 3 years. This assumption is very reasonable for recent technology nodes, as laid out in [55]. The other parameters of the model are based on [126] as well as the TSMC 65 nm general purpose library, and are summarized in Table 2.2. Together with the power law in Equation (2.1) this model allows us to model the HCI-induced delay degradation of single transistors, gates and at higher abstraction-levels even entire logic blocks.

Please note that the different components of this model were validated for various technology nodes [51, 52, 54, 126, 130, 131, 134]. However, due to rapidly changing technologies, it can happen in future that some relations may change, as it happened in the past with the temperature relation. Consequently, also the model has to be adjusted. Nevertheless, as long as high temperature, high supply voltage and high switching activity accelerate the threshold voltage shift due to HCI, all techniques proposed in this thesis can still be employed to alleviate the impact of HCI, as these are based on the overall HCI trends and not on particular numbers.

2.3.3. Summary

In summary, both HCI and BTI degrade the threshold voltage of the affected transistors depending on the transistor state (usage), supply voltage and temperature as summarized in Table 2.3. In order to mitigate accelerated transistor aging, these parameters have to be addressed. This can be done at various abstraction levels, as motivated in the Introduction (→

$\Delta V_{th}(t)$	$\approx$	$A_{HCI} \cdot \exp\left(\frac{E_{ox}}{E_1}\right) \cdot \exp\left(\frac{-E_a}{kT}\right) \cdot \sqrt{\alpha \cdot f \cdot t}$	Long-term HCI model for the V_{th} shift
E_a	$=$	0.1 eV, $k = 8.6174 \cdot 10^{-5}$ eV/K	Reaction/Diffusion constants
E_{ox}	$=$	$\frac{V_{gs}-V_{th,0}}{t_{ox}}$	Electric field
t_{ox}	$=$	2.0 nm for NMOS	Oxide thickness
$V_{th,0}$	$=$	0.271 V for NMOS	Default threshold voltage
E_1	$=$	0.8	Technology dependent constant
		T	Temperature
		α	Switching activity of gate output
		f	Clock frequency of the microprocessor
		A_{HCI}	Technology dependent constant

Table 2.2.: Summary of the HCI model employed in this thesis (parameters are obtained from the TSMC 65 nm general purpose library and [126])

	BTI	HCI
Temperature (T)	exponential	exponential
Frequency (f)	-	sublinear
Voltage (V_{dd})	exponential	exponential
Exec. Time (t)	sublinear	sublinear
Usage (δ, α)	sublinear	sublinear

Table 2.3.: Impact of different parameters on BTI and HCI

Figure 1.4). For instance, temperature and supply voltage can be adjusted at runtime by employing dynamic voltage and frequency scaling techniques. At (micro)-architecture-level the instruction scheduling can be adapted to reduce the stress time to alleviate BTI, or power and clock-gating approaches can be employed to reduce the impact of HCI (lower switching activity) and BTI (longer recovery time). Therefore, in the following section, a general introduction into the state of the art approaches is presented.

The overall *relative delay degradation* $\Delta^{rel}d$ at time t, which is referred to as *aging rate*, for a single transistor considering both BTI and HCI can be estimated using Equation (2.1). It follows that:

$$\Delta^{rel}d(t) = \frac{d(t)}{d(0)} = \frac{(V_{dd} - V_{th})^{\sigma}}{(V_{dd} - V_{th} - \Delta V_{th}(t))^{\sigma}} = \left(1 - \frac{\Delta V_{th}(t)}{V_{dd} - V_{th}}\right)^{-\sigma}, \quad (2.25)$$

where $\Delta V_{th}(t)$ is the superposition of the shift due to BTI and that due to HCI.

2.4. State of the Art

In order to mitigate accelerated transistor aging, the microelectronic industry including Intel [45, 124, 135], IBM [46], TSMC [44] and Globalfoundries [55] spends a great deal of effort on finding new device technologies (e.g. material compounds) or improved manufacturing processes that result in lower aging rates. Nevertheless, as laid out in various studies such as [55–57], the gradual delay increase due to HCI and BTI in recent technology nodes can be more than 10 % in 3 years, and is likely to increase further with the next technology nodes [7] (see Figure 1.3). Therefore, it is an absolute necessity to develop aging modeling and simulation platforms to be able to analyze the long-term effect of accelerated transistor aging. Moreover, efficient aging mitigation techniques have to be designed to lower the overall aging rate, and thus, to allow reduced guardbands or longer lifetime results.

In this section, an overview of the state of the art on modeling and mitigation approaches for accelerated transistor aging due to HCI and BTI is presented. In addition, for every aging mitigation technique proposed in this thesis, a detailed comparison with related approaches can be found in the corresponding chapter.

2.4.1. Modeling and Simulation

BTI Modeling Traditionally, most BTI modeling approaches focus on NBTI, since PBTI was not an issue until the mid 2000s. In addition, PBTI in recent technology nodes is very similar to NBTI as explained in Section 2.3.1, and consequently most of the NBTI models are also applicable to evaluate the behavior of transistors under PBTI-induced degradation.

The simplest behavioral BTI model at transistor-level considers only static stress, which means that the transistors are assumed to be continuously ON or OFF [48, 136]. However, during the normal operation of a PMOS (or NMOS) transistor in a CMOS gate, the applied gate bias switches between high and low voltages, and consequently stress phases alternate with recovery periods. Because of that, static models tend to be too pessimistic and overestimate the

real transistor degradation. As a result, it is important to analyze BTI under realistic dynamic conditions (considering both stress and recovery phases). Therefore, various reaction-diffusion based predictive models similar to the one presented in Section 2.3.1 were proposed to model the long-term effect of NBTI [49, 129, 137, 138]. Although all of these models explain the BTI phenomenon with the same physical process, all of them have unique features. For example, some of these models can capture the influence of supply voltage and temperature as well as the relation of NBTI with the oxide thickness, the diffusing species and other key design parameters [49, 138], while others just provide a simplified model [129]. Based on such an holistic reaction-diffusion model, gate-level aging models were proposed in [139], which can accurately capture the effect of NBTI on complete CMOS gates taking the *stacking effect*[9] into account. Beside the reaction-diffusion-based models, also some trapping/detrapping models were presented in [115, 140, 141], which explain the NBTI effect with trapping/detrapping processes (see Section 2.3.1). Similar to the reaction-diffusion models also these models differ in the number of accurately captured impact factors, e.g. temperature and supply voltage. Furthermore, as the trapping/detrapping process is probabilistic, i.e. the capture and emission times of the different traps are randomly distributed, atomistic trap-based models were developed to account for this variability [142]. These models can be used to analyze the effect of different traps on the overall device degradation. However, due to the complexity of these stochastic models, no (probabilistic) trapping/detrapping model at gate-level has been proposed, yet.

HCI Modeling The model diversity for HCI is considerably smaller compared to BTI, as the physical origins are better understood. In [51, 130] the influence of runtime and switching activity were investigated and accurate models to capture this relation were proposed. Furthermore, the impact of the device temperature was described in [54, 55]. In addition, in [126] a model was presented that accurately captures the impact of the electric fields. Since, all of these models focus on particular aspects, a combined transistor-level is employed in this thesis to form the basis of all studies, as explained in Section 2.3.2. A similar model for the threshold voltage shift was also proposed in [78], however, an inaccurate switching activity dependency was used. Similar to BTI, also HCI models have been abstracted from transistor- to gate-level [139, 143].

A major challenge for modeling aging effects is the consideration of their interdependencies. Although various HCI and BTI models were already proposed, the interdependencies of these simultaneously occurring effects were mostly neglected. Instead, the different phenomena were considered independently. To overcome this problem, a first step towards a combined aging model for different physical processes was presented in [144]. In this work an approach from device- to system-level was proposed which takes NBTI, PBTI and HCI into consideration. Therefore, the device-level aging models were combined and abstracted to system-level failure models for the case of SRAM cells.

High-Level Modeling & Simulation In order to simulate the long-term behavior of a microprocessor, or in general a circuit, under accelerated transistor aging, pure gate-level simulations are not sufficient, as workload-dependent factors such as temperature and usage (switching activity and duty cycle) are not accurately captured. Therefore, circuit-level platforms were proposed in [145–147] which rely on gate-level models, but use more realistic input data. Because of that, the simulation effort is very high, and thus the circuit behavior can be only simulated for a very short period of time (typically a few million clock cycles). In addition,

[9] In a complex CMOS gate that consists of many serial and parallel transistors the aging rate of a single transistors is not only dependent on the state of the affected transistor, but also on the states of the surrounding transistors. This phenomenon is called stacking effect.

these platforms require the analysis of a huge amount of data for large size circuits to extract the degradation of each gate, which makes them often infeasible for the analysis of complete microprocessors (see Section 3.4.4 for more details). Moreover, the impact of dynamic runtime adaptation approaches can hardly be evaluated at this level. Therefore, cycle-accurate performance simulators at system- and architecture-level such as gem5 [2] have to be employed with which billions of clock cycles can be simulated. However, these platforms omit the low-level behavior of a circuit and consequently the aging estimation at these levels is based on simplified models [78, 148, 149]. For instance, at high abstraction levels (e.g. architecture-level) often the combined effect of HCI and BTI is represented by a simple lifetime model considering only the influence of temperature or supply voltage [71, 148–150] or a simple inverter chain is used to mimic the underlying circuit behavior [78]. However, due to the strong simplifications of these models, the results are often inaccurate [15] (see Section 3.2.4 for more details).

Summary In summary, a lot of research has been performed at transistor- and gate-level to develop accurate behavioral aging models. However, these models are hardly applicable at higher abstraction levels. Therefore, abstracted yet accurate models and cross-layer evaluation platforms are necessary, which however are very rare. To close this gap, two novel cross-layer analysis frameworks are proposed in this thesis. Moreover, accurate architectural aging models are designed to allow a fast and accurate aging estimation in early design phases when no circuit-level information is available.

2.4.2. Mitigation

To alleviate the effects of accelerated transistor aging such as a reduced operational lifetime, several schemes and design techniques have been proposed. Since NBTI was considered for a long time to be the most important issue [151], most of these techniques focus on NBTI. However, as said before, the same approaches can be used to alleviate PBTI as well, or can be extended to also address HCI. In general, all aging mitigation schemes can be split into two major categories: *Compensation* and *Reduction*. The first class of techniques, tries to compensate the delay degradation of logic circuits, while the second class of approaches tries to reduce the overall aging rate. Nevertheless, the result for the entire system is in both cases a better lifetime. In the following the most relevant approaches for each category are introduced.

Aging Compensation Techniques The key idea behind these approaches is not to primarily address the actual aging rate of a gate, path, circuit or entire microprocessor. Instead, these approaches aim at compensating the delay degradation by either adjusting the circuit design or by adapting system parameters (e.g. supply voltage or frequency) at runtime. Therefore, gate sizing techniques [59, 62] set a suitable size for each transistor proportional to its expected probability of being stressed. In other words, gates with high aging rates are designed faster (i.e. with larger transistors) and gates that wearout very slowly are designed with smaller transistors. A similar idea is followed by threshold voltage tuning approaches [61] that reduce V_{th} for aging-critical (i.e. high aging rate) transistors to make them faster. Another class of approaches also employs this idea to re-organization circuit paths based on their aging criticality [63]. Therefore, these techniques try to design aging-critical paths with more slack, while non-critical paths are designed with less slack. In all of these schemes the actual aging rate of a gate or path is not considerably affected, however, since the critical components have more timing slack, the overall lifetime is improved. Furthermore, there are body biasing [68] and supply voltage tuning [71, 75] approaches which compensate the aging-induced delay degradation by adjusting the body voltage or the supply voltage. For this purpose, these techniques are combined with monitoring circuits [58, 152] to continuously track the delay of the circuit or aging prediction

approaches. Based on the measured or predicted value, the operating voltage is adjusted in an reactive manner such that the circuit can meet the timing constraints for the next time period.

Aging Reduction Techniques In contrast to the aging compensation techniques, the aging reduction approaches aim at directly reducing the aging rate of gates, paths or entire processors, and by that means try to improve the overall lifetime. The authors in [64] proposed a circuit-level technique based on power gating to reduce the power consumption and the NBTI effect, since power gated transistors are in the recovery mode. However, power gating techniques become less effective when the idle time of a circuit is not long enough or could not be predicted, as a result of the long wake-up latency [15]. A second circuit-level approach named "Input Vector Control" (IVC) uses the fact that the input vector of a circuit can be used to control the status (stress, recovery) of a PMOS transistor. Hence, NBTI can be alleviated by assigning a suitable input vector, when the circuit does not perform any operation (standby mode) [65]. Similarly, "internal node control" can be employed to address NBTI [67]. The difference of this approach compared to IVC is that additional controlling elements are added to the circuit to be able to change to state of particular gates and transistors. The idea of controlling the circuit inputs was extended in [145] to microarchitecture-level, by designing an NBTI-aware NOP (no-operation) instruction[10] that replaces the default NOP, and thus reduces the wearout in functional units. Furthermore, a stacking-based pin reordering technique at circuit-level was proposed in [61] to reduce the overall circuit degradation due to NBTI. This approach uses the fact that transistors inside a gate can be re-ordered under certain constraints without changing the gate output. Moreover, as the transistor duty cycles inside a gate are not balanced, the transistors can be re-ordered such that the overall NBTI-induced degradation is reduced. This concept was extended to consider HCI-induced wearout as well in [143].

At architecture-level, it was proposed in [11] to balance the stress and recovery time of transistors by periodically inverting the instruction opcodes and operands. Consequently, the transistor duty cycles are likely to be around 0.5, and thus it is avoided that particular devices suffer from high NBTI-induced aging rates (i.e. high duty cycle). Similarly, various techniques address NBTI-induced wearout in memory elements, by employing low-cost cell-flipping schemes [153, 154]. Furthermore, aging-aware instruction scheduling techniques were proposed [77]. These try to balance the workloads among all available functional units, to balance their delay degradation. The authors in [78] suggested to even employ an aging-aware task scheduling in combination with voltage scaling to balance the aging rates due to NBTI and HCI of different cores in a multiprocessor environment. Beside these "static" techniques, where all decisions are taken during the design phase of the microprocessor, there are also dynamic aging mitigation techniques that adjust the system at runtime. For instance, in [155] a runtime methodology was proposed to adapt frequency, supply voltage and body bias at runtime to maximize the average performance or improve the energy efficiency, while maintaining a given lifetime target. Finally, also at application-level aging mitigation can be performed. Therefore, in [156] an NBTI-aware OpenMP-based programming approach was presented to increase the recovery time of the cores with the highest aging rates.

In addition to these "direct" aging mitigation schemes, also all methodologies that aim at reducing the overall temperature such as thermal and power management schemes improve the microprocessor lifetime, since BTI and HCI are thermally accelerated effects. In this regard, various techniques were proposed that address different microprocessor parts [157–162]. However, it is important to note that it is not mandatory to reduce temperature or power to improve the microprocessor lifetime. This is due to the fact, that also other parameters have a strong influence on the wearout rate.

[10] A NOP instruction is a no-operation instruction, i.e. an instruction that effectively does nothing at all

Summary In summary, similar to the aging modeling approaches, a lot of research has been performed at low abstraction levels. In contrast, only very few techniques make use of knobs available at microarchitecture-level and above. However, also these higher abstraction levels have a considerable impact on the overall aging rate, and thus on the microprocessor lifetime. Therefore, this thesis proposes four novel aging mitigation approaches that utilize such knobs. Moreover, all of the proposed solutions exploit information of several abstraction layers, i.e. these are cross-layer solution. As we will show later, this allows a more effective co-optimization of lifetime, performance, energy consumption and cost, compared to traditional design solutions that focus on a single layer only.

2.4.3. Error detection and correction schemes

Although this thesis aims at extending the lifetime of microprocessors by alleviating the issue of accelerated transistor aging, it is impossible to completely avoid that timing violations will occur at some point. As explained in Section 2.2, this eventually leads to erroneous data and system failures. To reduce the failure rate, classical fault tolerance approaches based on error detection and correction schemes can be employed.

Traditional fault tolerance approaches to protect computer systems employ temporal or spacial (in form of additional hardware) redundancy. Among these, Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) are very well known approaches [163]. Moreover, for memory components, parity bits or Error Correction Codes (ECC) can be employed to detect and correct erroneous bits. In addition, Razor-like [164, 165] flipflops can be used instead of traditional flipflops to detect and correct timing errors. Moreover, with DIVA [166] and Argus [167], two further error detection techniques for the instruction pipeline of a microprocessor were proposed. DIVA is an online checker component inserted into the commit stage of the instruction pipeline, which continuously validates the computation, communication and controls exercised in a complex microprocessor core. The Argus technique continuously checks invariants to detect execution errors. Another resilient processor is developed by Intel [168] which employs several resiliency features to detect and correct errors. These include error-detection techniques as well as tunable replica circuits. An instruction-replay technique is used for correction of erroneous instructions. In this regard it is important to note that classical replay mechanisms for error correction cannot be applied to cope with aging-induced errors. This is due to the fact that once timing violations lead to errors, these errors have to be considered as permanent, i.e. whenever the same input combination (e.g. instruction + operands) is applied under the same conditions, the circuit output will be erroneous. Hence, if a replay scheme is employed, the circuit conditions have to be adapted, for instance by lowering the clock frequency. Beside these hardware-based approaches, also software-based, or algorithm-based fault tolerance (ABFT) techniques can be employed [169]. The main idea behind ABFT is that algorithms are designed in such a way, that they can tolerate errors. Therefore, various works have been published up until now that are intended for mathematical operations such as matrix-matrix multiplications [169–172], complete mathematical solvers for partial differential equations [173] and even cryptographic algorithms [174].

Part II.

Frameworks for Aging Modeling and Evaluation

CHAPTER
THREE

CROSS-LAYER AGING MODELING AND ANALYSIS FRAMEWORKS

In this chapter, we introduce the two cross-layer aging modeling and analysis frameworks which we developed to evaluate aging and aging-influencing parameters at different abstraction levels. At first, we give an overview about these frameworks, their purposes and differences, and afterwards both multi-purpose platforms are discussed independently. In this regard, we put a special focus on the employed and abstracted transistor aging models. Furthermore, an experimental platform using real processors is introduced that enables us to demonstrate the feasibility of various solutions with experimental data.

3.1. Overview

In order to model and evaluate accelerated transistor aging at higher abstraction levels (above circuit-level), and to explore the design space with various constraints including aging mitigation approaches, a holistic cross-layer aging analysis framework is required, as only such a framework allows to accurately analyze performance, temperature, power and aging under the influence of realistic applications. For this purpose, two platforms for different purposes were developed in the scope of this thesis. The first one is an architectural framework, which can be used, when detailed circuit-level information is not required or not available. For instance, this is the case in early phases of the design process during which a good approximation of the impact of accelerated transistor aging is sufficient. The other platform is a microarchitectural framework at Register-Transfer-Level (RTL), which is intended for investigations that rely on circuit- and gate-level knowledge. As a result, this framework allows an accurate and detailed aging analysis.

The architectural framework called *ExtraTime* is based on the cycle-accurate performance simulator *gem5* [2], that allows to run a complete computing system including an operating system (OS). In addition, we integrated appropriate power and temperature models, as well as novel (micro)-architectural aging models. This enables a runtime analysis of power, tempera-

	ExtraTime	RTL-Platform
Cross-Layer	⊕⊕	⊕⊕
Simulation Speed	⊕⊕	⊖⊖
Flexibility	⊕⊕	⊕
Dynamic Runtime Adaptation	⊕	⊖
Aging Accuracy	⊖	⊕⊕
Area/Power Evaluation	⊖⊖	⊕⊕
Performance Evaluation	⊕⊕	⊕

Table 3.1.: Comparison of the two different cross-layer platforms developed in the scope of this thesis

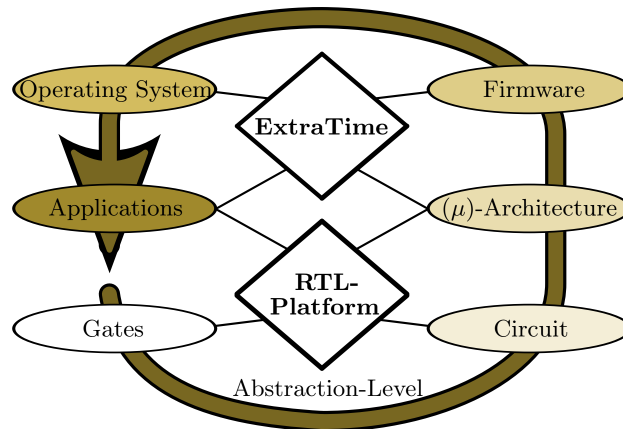


Figure 3.1.: Abstraction levels covered by ExtraTime and the RTL-Platform

ture and wearout, which makes the investigation of cross-layer and dynamic runtime adaptation techniques possible. Moreover, since the entire framework including the models for the microprocessor are written in C++, it is very flexible, i.e. design space exploration is easy. Also the simulation speed of this framework is very high as shown in Table 3.1, and because of that several billion clock cycles can be simulated and analyzed within a few hours.

The RTL-framework is based on standard industrial design tools from the electronic design automation (EDA) process such as synthesis tools[1] and circuit simulators[2] together with gate-level aging models. It is capable of analyzing all internal signals as well as gates, and it also considers the interplay of workloads, aging, power and temperature. Hence, this platform allows a very accurate analysis of aging, power and area cost. However, the flexibility compared to ExtraTime is significantly reduced and the simulation speed is rather low (only a few million clock cycles can be simulated within a few hours). In addition, the influence of the operating system and the firmware can hardly be captured as shown in Figure 3.1.

In summary, the ExtraTime platform is intended for design space exploration at architecture-level and above, that requires a fast and simplified aging estimation with reasonable accuracy. In contrast, the RTL-framework also accurately considers the circuit- and gate-level influence, and hence allows a very accurate aging estimation, however at the cost of a reduced flexibility. Thus, ExtraTime can be used in early design phases to allow the design of aging-aware microprocessors, while the RTL-framework is employed in later phases to fine-tune the design. In the following both platforms are introduced and discussed in detail. In Section 3.2, ExtraTime is presented and Section 3.4 focuses on the RTL-platform. In addition, in Section 3.3 our experimental platform based on the ExtraTime models, but using real processors, is presented.

3.2. ExtraTime: Modeling and Analysis of Transistor Aging at Architecture-Level

The objective of the ExtraTime framework is to model and analyze wearout due to transistor aging at architecture-level without requiring detailed transistor- or circuit-level information. Hence, the aging rate of each architectural component has to be estimated using only parameters known at this abstraction level. In this regard, architectural blocks are for example the instruction fetch unit, the different functional units such as ALUs and FPUs, the branch predictor, or the caches. Therefore, novel aging models are required. These were developed

[1] A synthesis tool transforms the behavioral RTL description of a circuit to a gate-level netlist.

[2] A circuit simulator is used to simulate the signal behavior inside a circuit under a sequence of input stimuli.

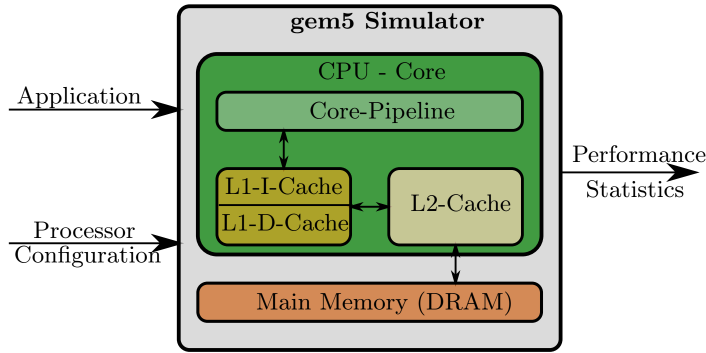


Figure 3.2.: Gem5 performance simulator platform with a single core configuration

in the scope of this thesis and are an essential part of ExtraTime. In this section we present this platform as well as the aging models in detail. Moreover, at the end a comparison with state-of-the-art modeling and simulation approaches is presented.

3.2.1. Overview and Introduction of the Basic Components

The basis of our ExtraTime framework is the *gem5* performance simulator [2] that includes configurable models for various microarchitectures ranging from simple 5-stage in-order pipelines to complex out-of-order architectures. Moreover, it supports a diversity of instruction sets (e.g. x86, alpha, power, ARM, SPARC), and it allows to run a Linux-based operating system on top of the modeled computing system (microprocessor, main memory, hard disk, network interface). In addition, almost every setup parameter such as memory sizes, memory latencies, number of functional units and their latencies, clock frequency and pipeline width[3] can be configured by the user. For this thesis, we picked the alpha instruction set and the cycle-accurate model for the 7-stage out-of-order, superscalar architecture, which is based on the Alpha 21264 core [84]. Nevertheless, the modeling and analysis infrastructure is not tight to this particular configuration, and can be used with any other setup requiring only small code modifications. Moreover, we modified gem5 to support various clock- and power gating strategies [15], several instruction scheduling policies [15] and dynamic voltage and frequency scaling (DVFS). Among the scheduling techniques is also the one presented in Chapter 5, and the DVFS approach is discussed in Chapter 7.

As the illustration in Figure 3.2 shows, the gem5 simulator delivers detailed information about the performance statistics of the modeled processor, while executing an application (e.g. a SPEC benchmark). This output contains, among other information, data about the hit and miss rates of caches, the usage of functional units, or the number of committed instructions. However, this information alone is not enough to accurately estimate the aging rates of different architectural components, since accelerated transistor aging is strongly dependent on the temperature. Thus, sophisticated architectural temperature models are required. Moreover, since temperature is correlated with the power consumption, also detailed models for the power consumption are needed. In this thesis, we use a customized version of *McPAT* [5] to obtain the dynamic and static power consumption of all architectural components and the corresponding temperature model is based on *HotSpot* [3].

McPAT is a power and area modeling framework that was validated using several modern microprocessors [5] such as an ARM Cortex A9 with out-of-order execution or an Intel Atom with an in-order instruction pipeline. Also our own data, extracted with the experimental platform detailed in Section 3.3, shows the accuracy of McPAT compared to real measured

[3]The width of an instruction pipeline describes how many instructions can be fetched, decoded or committed in parallel.

data. As depicted in Figure 3.3, it uses a cross-layer abstraction approach from transistor-to architecture-level to obtain the size as well as the static and dynamic power consumption for each architectural component inside a microprocessor. Therefore, technology data (V_{dd}, V_{th}, feature size, etc.) based on the ITRS roadmap [10] for technology nodes ranging from 90 nm to 22 nm is employed to estimate the power consumption and size for various gate types (inverter, NAND, NOR, etc.). This information is then abstracted in several steps using architecture models for the basic microprocessor components (execution units, pipeline stages, caches, registers, etc.) to calculate the results for complete architectural blocks. Depending on the regularity of the block under investigation, this process is either performed analytically (e.g. for regular caches) or with the help of empirical data (e.g. for irregular functional units). Finally, the performance data delivered by a performance simulator is used to obtain the overall activity of a particular component for the last evaluation period. By that means, the average dynamic and static power consumption of this block for the corresponding time period can be estimated. In order to use McPAT as part of the ExtraTime framework and not as a standalone tool, several enhancements were made, such that the power model now supports dynamic adjustments of frequency and supply voltage at runtime as well as clock and power gating. Moreover, to integrate it into the gem5 simulator, we modified the code structure in such a way, that parameter changes do not require a time consuming reconfiguration of the cross-layer power models, which is the case for the original standalone version. Therefore, the models were updated according to the Equations (3.1) and (3.2) to take temperature T, frequency f or supply voltage V_{dd} changes into account [175] without requiring a re-initialization. As a result, the simulation time is improved by one to several orders of magnitude, depending on how often the power consumption is evaluated (in this thesis power is usually evaluated every 1 μs to 1 ms).

$$P_{dynamic} \sim f \cdot V_{dd}^2 \tag{3.1}$$

$$P_{static} \sim V_{dd} \cdot \left(aT^2 \cdot \exp(\frac{\alpha V_{dd} + \beta}{T}) + b \cdot \exp\left(\gamma V_{dd} + \delta\right) \right) \tag{3.2}$$

The (constant) fitting parameters $a, b, \alpha, \beta, \gamma, \delta$ that describe the temperature and voltage impact on the static power were extracted using McPAT's default static power model.

HotSpot is an accurate compact thermal model validated against various FPGA-based designs. Its accuracy is also underlined by our own experimental data obtained with the experimental platform described in Section 3.3. In HotSpot, the thermal behavior of architectural components and their thermal interaction is modeled with an equivalent circuit of resistances and capacitances. This is possible due to the duality between heat flow and electrical current [176], i.e. the heat flow passing through a thermal resistance is equivalent to an electrical current through an electrical resistance (R). Moreover, the potential (voltage) of a node corre-

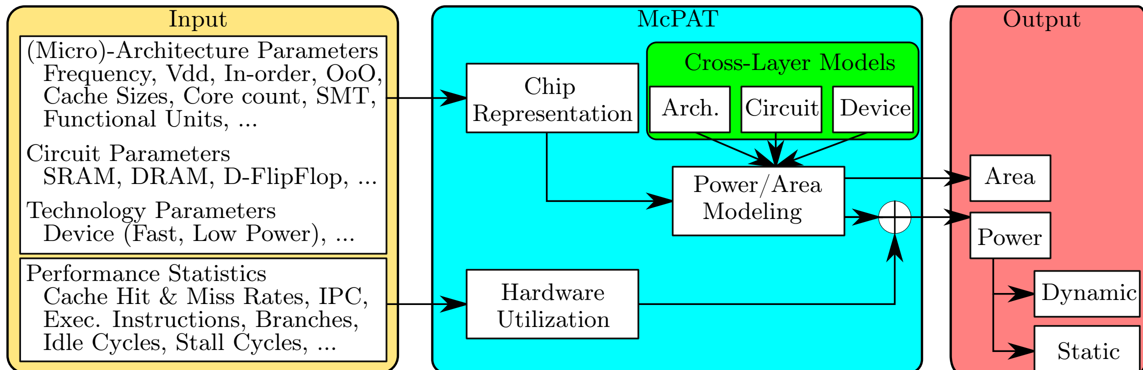


Figure 3.3.: Overview of the McPAT power and area modeling framework

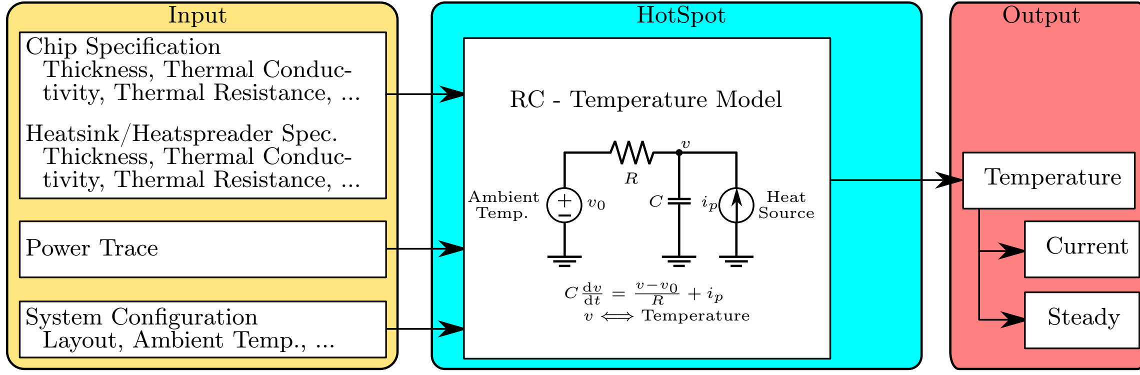


Figure 3.4.: Overview of the HotSpot thermal modeling framework

sponds to its temperature. In addition, heat absorbing materials (e.g. a cooler) are modeled as thermal capacitances which can be translated to electrical capacitances (C). By that means, the thermal modeling problem is translated to an electrical RC problem in which the heat source is replaced with a current source, as illustrated in Figure 3.4. Therefore, the current value for each component is computed using the power data delivered by a power model. Beside this power trace, HotSpot also requires a floorplan (layout) of the microprocessors as input, which contains the size of each architectural component and shows which components are neighbors to model the thermal interactions accurately. Moreover, HotSpot also considers the influence of essential aspects of the thermal package (thermal resistance of the package, heatspreader, PCB material, etc.) on the temperature distribution.

As part of the ExtraTime framework, the temperature model is used to obtain the maximum, minimum and average temperature within a given time interval for each architectural component. This temperature information in conjunction with information about the usage/activity of different architectural blocks is used by our proposed *architectural aging models*, to estimate the aging rates and lifetime for each microprocessor component. Therefore, the aging models (which are explained in the next subsection), the power model and the temperature model are integrated in one common framework (no more standalone tools) considering their interdependencies (e.g. power affects temperature, which in turn affects power), as shown in Figure 3.5. As a result of this design, ExtraTime is capable of analyzing the impact of various knobs at different abstraction levels ranging from architecture-level up to application-level on performance, power and in particular aging. In addition, it enables a runtime analysis of power, temperature and wearout, which makes the investigation of dynamic runtime adaptation techniques possible (see Section 3.2.4 for more details). Because of that, ExtraTime models a dependable

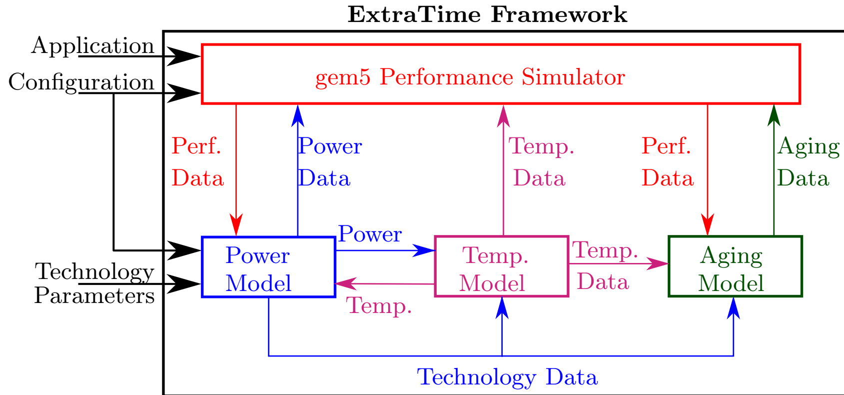


Figure 3.5.: Data flow in the proposed ExtraTime framework consisting of gem5, a power and temperature model and novel aging models

system, i.e. the microprocessor configuration can be adapted depending on the power, thermal or wearout status.

Please note that in a real processor the models for power and temperature can be realized by sensors. In this case, the aging rates can be either estimated using path delay sensors or by employing analytical approaches (i.e. aging models) in hardware or software, depending on the required accuracy, implementation overhead and runtime costs.

3.2.2. Aging Models at Architecture-Level

For our purpose, to model aging at architecture-level without having detailed transistor-level implementations, it is necessary to use special models. In the following we introduce these models and explain how these are derived from transistor-level models.

Definition of an Aging Metric

As the first step to build the architectural aging models, the definition of a suitable metric to estimate aging is necessary. For this purpose, we use the relative delay change $\Delta^{rel} d_B$ (see Section 2.3.3) of an architectural block B at time $t > t_0$ induced by aged transistors, which is defined as:

$$\Delta^{rel} d_B(t) = \frac{d_B(t)}{d_B(t_0)}. \tag{3.3}$$

The delay of a block always depends on its circuit layout (i.e. the number, size and type of transistors in the critical paths, the delay of the transistors, etc.). However, this information is usually not available at architecture-level, e.g. in early phases of the microprocessor development the RTL- or transistor-level circuit description is not available. Hence, in order to build architectural aging models some assumptions of the underlying hardware layout are necessary. Therefore, for a start, we assume that all transistors in one architectural block behave similar, i.e. age at the same rate. This means that all transistors inside one architectural block can be represented by one single transistor $\mathcal{T}$ (*representative transistor*). By that means, $\Delta^{rel} d_B$ of a block can be estimated by the relative delay change of the representative transistor $\Delta^{rel} d_{\mathcal{T}_B}$ of this block, i.e:

$$\Delta^{rel} d_B \approx \Delta^{rel} d_{\mathcal{T}_B}. \tag{3.4}$$

With the relation between $\Delta^{rel} d$ and ΔV_{th} given in Equation (2.25) the relative delay change $\Delta^{rel} d_{\mathcal{T}_B}$ of the representative transistor of block B is:

$$\Delta^{rel} d_{\mathcal{T}_B}(t) = \left(1 - \frac{\Delta V_{th}(t)}{V_{dd} - V_{th}(t_0)}\right)^{-\sigma} = f(\Delta V_{th}(t)). \tag{3.5}$$

Since everything in this equation is constant in time (and known) except ΔV_{th}, only ΔV_{th} has to be calculated to determine $\Delta^{rel} d_{\mathcal{T}_B}$. As it is explained in the following subsections, this can be done using *only* architectural information and known constants. Thus, $\Delta^{rel} d$ of each block can be obtained using *only* data which is known at architecture-level.

$$\begin{aligned} \Delta^{rel} d_B &\approx \Delta^{rel} d_{\mathcal{T}_B} = f(\Delta V_{th}) \\ &\approx f(g(\text{parameters known at architecture-level})) \end{aligned} \tag{3.6}$$

The last equation is also the reason, why we have chosen the relative delay change as the aging metric and not the total delay change $(d_B(t) - d_B(t_0))$. While the former one can be estimated using only architectural information, for the latter the original delay $d_B(t_0)$ has to be known, which is usually not the case at architecture-level. In addition, since $\Delta^{rel} d_B(t = \text{TTF})$ exceeds the guardbands due to the TTF definition used in this thesis (see Section 2.2.1), also

the TTF can be calculated as a function of $\Delta^{rel} d_B$. Therefore, since it is impossible to run a simulation for $t_{sim} = TTF$, the average degradation of the representative transistor during the simulation time is calculated and then used to estimate the degradation for $t > t_{sim}$ (i.e. extrapolated), by assuming that this average behavior is preserved for $t > t_{sim}$. Consequently, the average behavior of the block under investigation is considered to estimate TTF, and thus the resulting value can also be used to represent the MTTF of this block.

Bias Temperature Instability

The goal in the following is to estimate the V_{th} shift of the representative transistor $\mathcal{T}_B$ of an architectural block B due to BTI by using only architectural information.

Except the transistor dependent temperature T and duty cycle δ, all parameters of the transistor-level aging model (see Table 2.1) for BTI are known at architecture-level. Hence, only these two variables have to be attained. Since our first-order assumption is that all transistors in one architectural block behave similar, they have the same temperature (as the whole block) T_B, which is the maximum temperature of this particular block given by the temperature model of ExtraTime. By that means, also $\mathcal{T}_B$ has this temperature. In order to estimate the duty cycle δ of $\mathcal{T}_B$ (ratio between the time the transistor is under stress and total time) at architecture-level, the duty cycle δ_B of the block B is used, which represents the usage behavior of the block. Therefore, the stress time of block B is defined as the time in which at least one transistor inside this block can be under stress:

$$\delta_B = \frac{t_{stress,B}}{t_{total}} = \frac{\#cyc_{stress,B}}{\#cyc_{total}} \tag{3.7}$$

Thus, the newly defined architectural duty cycle δ_B of an entire block can be derived from parameters delivered by a performance simulator (total cycle count = $\#cyc_{total}$, number of stress cycles = $\#cyc_{stess,B}$). If power gating is used, the number of stress cycles can be easily calculated using the following relation (number of power gated cycles = $\#cyc_{pg,B}$):

$$\#cyc_{stess,B} = \#cyc_{total} - \#cyc_{pg,B}. \tag{3.8}$$

Otherwise, if power gating is not used, in all cycles at least one transistor can be under stress, and consequently $\delta_B = 1$ in this case.

Putting all this together and using δ_B as an estimation for the duty cycle of the representative transistor $\mathcal{T}_B$ leads to a first-order upper bound estimation at architecture-level for the V_{th} shift of $\mathcal{T}_B$ due to BTI:

$$\Delta^{1st} V_{th}(t) \leq A_{BTI} \left(\frac{\sqrt{D_H(T_B) \cdot \delta_B \cdot T_{cyc}}}{1 - \beta^{1/(2n)}(\delta_B, T_B, t)} \right)^{2n}, \tag{3.9}$$

whereby β, A_{BTI}, T_{cyc} and D_H are defined as in the transistor-level model (see Table 2.1), but using the architectural values T_B and δ_B instead of the transistor dependent ones (T, δ).

It is important to note that this BTI model delivers an upper bound for the real degradation. This is due to the definition of the block duty cycle in Equation (3.7) which was a consequence of the starting assumption that all transistors inside one block behave the same. In fact, in typical workloads, the duty cycle of each transistor inside a block will be far less than the duty cycle of the entire block. In other words, the assumption, that all transistors inside one block have the same duty cycle as the block, is too pessimistic. Hence, the real V_{th} shift will be smaller than the one calculated in Equation (3.9), which means that this equation is an overestimation. To improve this first-order estimation, it is necessary, to take a look at circuit-level. In a path inside a circuit, there are fast as well as slow gates, and there are gates with

high aging rates as well as low aging rates. Hence, the idea to approximate the aging rate of a path by averaging the aging rates of all gates is self-evident. This means:

$$\Delta^{rel} d_{\text{path}}(t) = \frac{d_{\text{path}}(t)}{d_{\text{path}}(0)} = \sum_{i=0}^{n} \frac{d_{\text{gate},i}(0)}{d_{\text{path}}(0)} \cdot \frac{d_{\text{gate},i}(t)}{d_{\text{gate},i}(0)} \approx \frac{1}{n} \sum_{i=0}^{n} \frac{d_{\text{gate},i}(t)}{d_{\text{gate},i}(0)} \tag{3.10}$$

Indeed, this is a good estimation (inaccuracy $\leq 3\,\%$), according to the simulations we performed for various circuits and stress conditions (i.e. input patterns, temperature, etc.) using accurate transistor-level models (see Section 3.2.3 for more details). However, at architecture-level the aging rates as well as delays of gates and paths are not available, and thus, additional simplifications are required to employ the approximation in Equation (3.10) at this abstraction level. Therefore, the duty cycle of a transistor inside a block B is represented by the product of the duty cycle of the block δ_B and the effective duty cycle δ_e. The latter is then assumed to be uniformly distributed between 0 and 1. As a result, one can obtain a second-order estimation for the V_{th} shift of the representative transistor, which corresponds to the "average" V_{th} shift considering all transistors.

$$\Delta^{2nd} V_{th}(t) \leq \int_0^1 A_{BTI} \left(\frac{\sqrt{D_H(T_B) \cdot \delta_B \cdot \delta_e \cdot T_{cyc}}}{1 - \beta^{1/(2n)}(\delta_B \cdot \delta_e, T_B, t)} \right)^{2n} \mathrm{d}\delta_e. \tag{3.11}$$

Of course, this estimation is too optimistic for particular transistors inside the block, but since we are interested in the V_{th} shift of the representative transistor, i.e. the delay change of the entire architectural block rather than in the change of a single transistor, the "average" V_{th} shift is a good estimation, as motivated before. We validated the accuracy of this high-level BTI modeling approach in Section 3.2.3 with the result, that this model can estimate the worst-case delay degradation very well, i.e. the relative inaccuracy is less than $5\,\%$ compared to a detailed transistor-level model. Hence, this equation forms our architectural BTI model to estimate the V_{th} shift of the representative transistor $\mathcal{T}_B$, and together with Equation (3.5) the aging rate of the component B.

Please note that the assumption of a uniform duty cycle distribution can be further optimized to be only partially uniform, if detailed knowledge about the underlying hardware implementation is available. In this case, a partially weighted mean (integral) is used, where the weights represent the occurrence probability of certain duty cycle ranges. As a result, also other distributions such as normal or W-shape distributions (similar to Figure 4.8(b)) can be modeled with this approach.

Hot Carrier Injection

As for BTI, the goal of the following part is to estimate the V_{th} shift of the representative transistor $\mathcal{T}_B$ due to HCI by using only architectural information.

The approach to transfer the transistor-level model for HCI in Equation (2.24) to architecture-level is quite similar to the one used for BTI. Again, the problem is that the temperature T corresponds to individual transistors and the activity factor α is also transistor-dependent. Hence, as for BTI, the temperature T_B of an entire architectural block B is used for all transistors inside this block. Since the activity factor of a transistor is the product of the activity factor of this transistor, while the complete architectural block B is active (effective activity factor α_{eff}), and the activity factor α_B of the block itself, Equation (2.24) can be written as follows:

$$\Delta V_{th}(t) = A_{HCI} \cdot \exp\left(\frac{E_{ox}}{E_1}\right) \cdot \exp\left(\frac{-E_a}{kT_B}\right) \cdot \sqrt{\alpha_B \cdot \alpha_{eff} \cdot f \cdot t}. \tag{3.12}$$

In this regard, a block is active, when at least one transistor inside this block is active (i.e it can make a transition):

$$\alpha_B = \frac{t_{active,B}}{t_{total}} = \frac{cyc_{active,B}}{\#cyc_{total}} \tag{3.13}$$

As the equation above illustrates, the activity factor α_B of a block can be calculated using only parameters delivered by the performance simulator (total cycle count = $\#cyc_{total}$, number of active cycles = $\#cyc_{active,B}$). In case clock or power gating is available, the number of active cycles of a block can be calculated using the following relation (number of power gated cycles = $\#cyc_{pg,B}$, number of clock gated cycles = $\#cyc_{cg,B}$):

$$\#cyc_{active,B} = \#cyc_{total} - \#cyc_{pg,B} - \#cyc_{cg,B}. \tag{3.14}$$

Otherwise, if neither clock nor power gating are employed, at least one transistor inside the block can switch and consequently, $\alpha_B = 1$.

With respect to the representative transistor, the effective activity factor α_{eff} has a specific meaning. It represents the *average switching activity* $\alpha_{avg,B}$ of all gates in the architectural block B. In worst case the average switching activity for the (near-)critical paths can be 1, according to our transistor-level investigations using SPEC2000 workloads. Hence, we use the value of 1 for $\alpha_{avg,B}$ to obtain the maximum V_{th} shift that can be induced by typical applications. If detailed transistor-level information is available, this value can also be adjusted to improve the accuracy of the estimation.

Thus, the HCI model at architecture-level and in this way the estimation of the V_{th} shift of the representative transistor $\mathcal{T}_B$ has the form:

$$\Delta V_{th}(t) \leq A_{HCI} \cdot \sqrt{\alpha_{avg,B}} \cdot \exp\left(\frac{E_{ox}}{E_1}\right) \cdot \exp\left(\frac{-E_a}{kT_B}\right) \cdot \sqrt{\alpha_B \cdot f \cdot t}, \tag{3.15}$$

where the parameters A_{HCI}, E_{ox}, E_1, E_a and k are defined in Table 2.2.

Similar to BTI-induced aging, Equation (3.5) and (3.15) enable us to also calculate the $\Delta^{rel}d$ due to HCI during runtime to determine the actual aging status. Since, the inaccuracy of this high-level HCI model is less than 5 % compared to a detailed transistor-level model, as explained in the Section 3.2.3, we employ this model in ExtraTime.

3.2.3. Accuracy Analysis

As mentioned earlier, the abstraction of the aging models from transistor-level to architecture-level has an impact on the estimation accuracy. The purpose in the following is to quantify the amount of inaccuracy due to this abstraction.

Temperature

One major assumption of our proposed architectural aging models is that all transistors within a block have the same temperature, which is equal to the maximum temperature of the block obtained by the thermal model. Hence, the temperature gradient within a block is neglected. Using ExtraTime we observed that the maximum temperature difference within one architectural block is at most 6 °C for various SPEC2000 benchmarks simulated with the out-of-order processor model of gem5. This is supported by experimental results using an IBM Power7+ system[4] which features five digital temperature sensors per core [177]. For this machine the maximum measured temperature difference for an entire core was 9 °C, while running the SPEC2006 benchmark suite.

[4]System Configuration: IBM Power 720 with 4 Power7+-cores [8] running at 3.6 GHz with 1.08 Volt, manufactured in a 32 nm technology; 16 GByte DDR3-RAM; Debian Wheezy Linux 3.2; SPEC2006 benchmark suite compiled with GCC 4.6.3; Sensor readout performed with IBM's Amester software.

	Temperature	$\Delta^{rel} d_{ALU}$ after 3 years	
		due to BTI	due to HCI
Minimum	71 °C	9.4%	8.5%
Maximum	77 °C	9.6%	8.9%
Relative Difference		2.1%	4.5%

Table 3.2.: Temperature effect on the aging-induced delay change for an ALU

Based on accurate transistor-level aging models for an ALU, the observed temperature gradient translates to a difference in total delay of less than 1 %, which is negligible. Also the difference in terms of $\Delta^{rel} d$ is considerably small ($< 5\,\%$), as shown Table 3.2. Hence, the deviation of our architectural aging models, which use the maximum temperature per block, compared to accurate transistor-level models is less than 5 %, which is very good given the level of abstraction.

Usage

Beside the assumption that all transistors have the same temperature, we also made an assumption regarding the "usage" of the transistors in an architectural block, i.e. the time a transistor is under stress (BTI) or the amount of transitions that occur (HCI).

For BTI we derived the "average" V_{th} shift given in Equation (3.11), to take into account that different transistors in one block will have different stress times. We have validated this approach using a detailed transistor-level model of an ALU by considering the influence of different input vectors (i.e. different instruction opcodes and different operands) [65] and the stacking effect [61]. Therefore, the ALU of the Illinois Verilog Model (IVM) [4], which is a Verilog model of the Alpha 21264 core, was synthesized using the Synopsys Design Compiler and mapped to the SAED 90 nm standard cell library. Afterwards, for each instruction 10,000 random operands together with the instruction opcode were applied at the inputs of the ALU to find out the duty cycle of each transistor in the 10 % most critical paths. In the next step, the delay change for each considered transistor was estimated using the transistor-level model given in Table 2.1. Using these results, the overall delay change for the entire ALU was calculated. For a runtime of three years under a constant temperature of 70 °C, this accurate transistor-level approach estimates the maximum $\Delta^{rel} d_{ALU}$ (maximum over all input vectors) to be 9.7 %. Using our architectural BTI model the $\Delta^{rel} d_{ALU}$ is estimated to be 9.4 % under the same conditions (i.e. temperature, supply voltage, etc.). Hence, as given in Table 3.3, our model is just 3 % off. This shows that our architectural aging model for BTI comes with a reasonable accuracy given the level of abstraction, but in addition architectural investigations are possible.

Please note that investigating the maximum $\Delta^{rel} d_{ALU}$ is sufficient here, since the maximum delay degradation that can occur matters most, i.e. it does not matter how many input combinations are working, as long as one can lead to a timing failure, the entire system can fail. Furthermore, as mentioned in Section 3.2.2, the accuracy of our architectural model for BTI can be improved, if detailed knowledge about the underlying hardware implementation is available.

	$\Delta^{rel} d_{ALU}$ after 3 years	
	BTI	HCI
Detailed Transistor-Level Model (Max)	9.7 %	8.4 %
Proposed Architectural Model	9.4 %	8.4 %
Relative Difference	3.1 %	0.0 %

Table 3.3.: Difference between a detailed transistor-level model and the proposed architectural models regarding the estimate delay for an ALU after 3 years

For HCI the accuracy investigation is very similar. The only difference is, that pairs of input combinations have to be investigated, in order to obtain the switching behavior of the circuit. The results are even better, showing that also the model for HCI comes with a satisfying accuracy for the estimation of the maximum degradation with respect to the level of abstraction.

In summary, both architectural models are reasonable accurate and are based on simplifications that are typical for this abstraction level (e.g. McPAT uses similar simplifications). Thus, these models allow a fast, yet accurate aging estimation without requiring detailed circuit- or even transistor-level knowledge. Because of that, these models can be employed in early design phases and allow comprehensive aging-aware microprocessor designs.

3.2.4. Comparison with State of the Art Approches

Beside the accurate aging models, ExtraTime possesses several other advantages compared to state-of-the-art approaches at (micro)-architecture-level. In the following part the advantages are explained and underlined with simulation data.

The major advantage of ExtraTime compared to other platforms such as [78, 146, 148, 149] is the usage of realistic high-level aging models that do not require detailed circuit-level information. As a result, it can be used already in early design phases to allow a first order aging analysis and design space exploration. In contrast, in [146] a circuit-level description of the microprocessor is required, and in [78] an inverter chain is modeled to estimate the delay degradation leading to non-realistic aging conditions. In addition, the aging models are often too simplistic. For example, in [148, 149] the applied models do not consider usage at all, meaning an architectural block has always the same aging rate, no matter if the block is used, clock gated or power gated. Hence, these frameworks are not capable of accurately modeling aging of modern microprocessors under varying applications. In [78] usage is taken into account, but for HCI the usage dependency is considered to be linear. In fact, this relationship is sublinear [51], which leads to overestimations. Another problem of some techniques is that the temperature is estimated on a too coarse level, e.g. in [148] the highest temperature of the entire microprocessor is used to estimate aging. Our experimental analysis shows that this would lead to around 30 % wearout-overestimation for the execution units of the processor under investigation.

To increase the accuracy of the aging estimation we have extended the power model (based on McPAT) in various areas. Our customized version uses the actual temperature of each architectural block to estimate leakage power. Compared to the original version, in which all blocks were assumed to have the same temperature, this is a huge benefit. This is also a major lack in current solutions such as [78, 146, 148], where usually first the performance simulation is accomplished, afterwards a power trace is created followed by a modeling of the temperature behavior. Hence, an accurate coupling between temperature and power is missing in most existing solutions. Thereby, our results obtained with the adjusted version of McPAT show that a temperature difference of 10 degrees can impact leakage power by up to 50 %, if temperature increases by 30 degrees, leakage power can even increase by a factor of 3. Due to the coupling between temperature and power, the increase in leakage power leads to an increase in temperature, which can be up to 23 degrees (analysis based on HotSpot) in the latter case. If power and temperature are considered independently, the increase in leakage power and its impact on temperature is neglected. For this reason, it is very important, to consider the tight coupling between temperature and power, in order to obtain accurate power, temperature results, which are necessary to accurately estimate aging.

Another advantage of our approach is the integration of all models in one common framework. In order to allow dynamic runtime adaption based on the current state (temperature, power, aging, performance) of the microprocessor, it is necessary to calculate the power con-

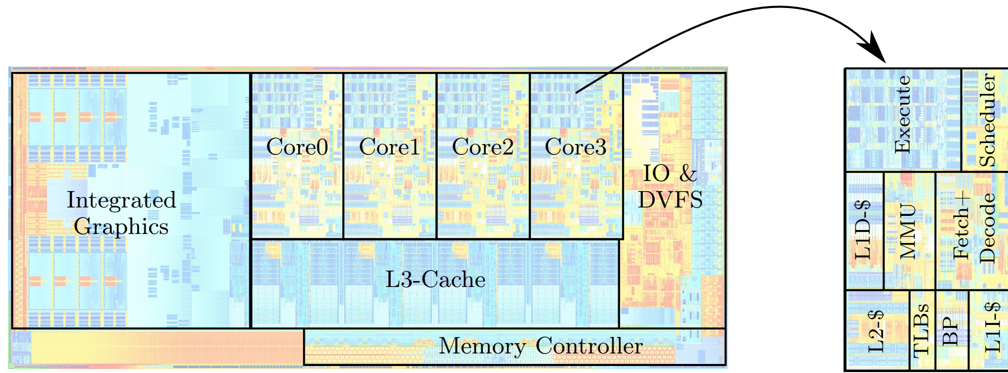


Figure 3.6.: Layout of an Intel Core i5 3rd gen (complete processor and single core) taken from [24, 178]

sumption, temperature or wearout every X cycles while the simulation is running (online) rather than only once after the simulation is done (offline). Transferring the necessary data every X cycles among standalone (offline) solutions can result in considerable performance overhead (up to 25 %) and huge data storage overhead (up to 25x). In summary, the only viable solution to accurately investigate wearout, temperature, and power including the runtime behavior of the executed applications is the integration of the necessary models in one holistic framework. In fact, this is missing in all previously published techniques, which have their focus on (micro)-architectural solutions. ExtraTime closes this gap.

However, the advantage that ExtraTime omits detailed low-level information makes it hard to evaluate the circuit-level implications of potential aging mitigation techniques with this framework. For this purpose, we developed the already mentioned RTL-platform, which is presented in Section 3.4.

3.3. ExtraTime in a real system

Beside the pure simulation framework ExtraTime, we also built a real hardware platform based on Intel's Core-i-microprocessors. For this purpose, this hardware platform reuses ExtraTime's power and thermal models which are calibrated using a single available on-chip sensor for power and temperature, respectively. As a result, this framework allows us to extract power and thermal information with a high spacial and temporal resolution, while the microprocessor executes real workloads, although the available sensors only report aggregated values (e.g. combined power consumption of all cores). Hence, this platform enables us to demonstrate the feasibility of various approaches with experimental data (performance, power and temperature).

3.3.1. Description and Explanation of the Hardware Platform

The hardware platform is built around Intel's Core-i-processors [85] (e.g. Intel Core i5-2400 or Core i5-3450) and as operating system Ubuntu Linux 64-bit is employed. The processors feature up to four superscalar out-of-order cores that can execute up to two threads per core, an integrated DVFS controller, memory controller and graphics, as depicted in Figure 3.6. Furthermore, these processors have built-in sensors to monitor the average power consumption of the last millisecond as well as a diversity of Digital Thermal Sensors (DTSs) that can be accessed via Special Purpose Registers (SPRs) [179]. However, the power sensors can only obtain the combined power consumption of all cores as well as the complete package, and as a user one has also only access to the maximum temperature value of each core. Hence, the available information is rather coarse-grained, which does not allow an accurate power, thermal or aging estimation for different (micro)-architectural components.

Special Purpose Registers (SPRs)	Functionality	Monitored Component
L2_TRANS_ALL_X	L2 operations with X=Read/Write/Total	L2Cache, DCache
UOPS_DISPATCHED_PORT_X	Dispatched instructions at port X	Exec, LSU, Reg, Scheduler
INSTS_RETIRED.ANY	Retired instructions	Fetch, Retire
CPU_CLK_UNHALTED_*	Active cycles of the entire core	Overall core
BR_*_ALL_BRANCHES	Executed/Mispredicted branches	Branch Predictor
MEM_UOPS_RETIRED_X	Retired operations with X=Load/Store	LSU, TLB, DCache
IDQ_MITE_ALL_UOPS	Decoded micro-instructions	ICache, Fetch, Decode, ROB
FP_COMP_OPS_EXE.*	Computational scalar FP operations	Exec
SIMD_FP_256.*	Vector operations	Exec

Table 3.4.: Employed SPRs for the proposed power estimation (here for Intel processors)

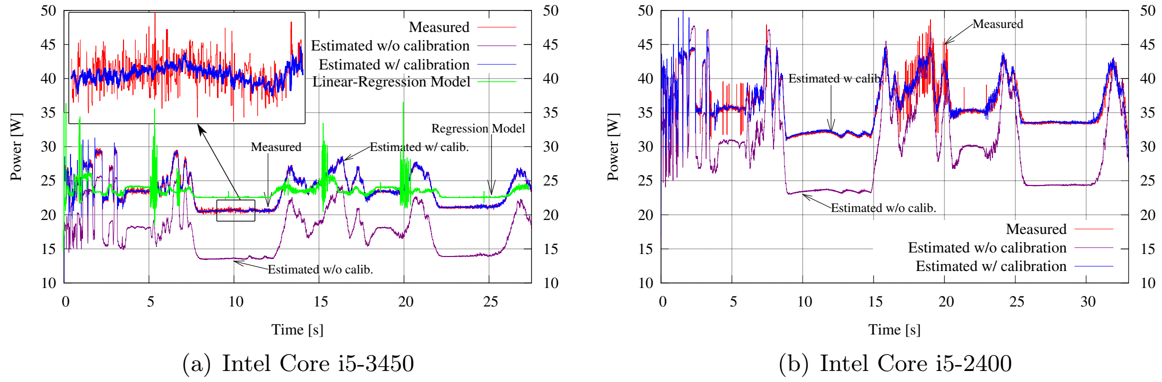
Therefore, we implemented ExtraTime's power and thermal model in the form of kernel modules to have a higher spacial resolution, i.e. information for each (micro)-architectural block of the employed microprocessor. The input to the power model is performance data (like in ExtraTime), which is gained from the available performance counters and SPRs [179]. For the Intel-based system we use, the SPRs detailed in Table 3.4 are accessed to extract the necessary information. The access is performed by self-written C++-routines that use the `/dev/cpu/CPUNUM/msr` interface of modern Linux kernels to read the SPR data. As a result, the time required to gather all data and estimate the power consumption of all components within a single core takes less than $50\,\mu$s for an Intel Core i5-2400 in one estimation step.

The power estimation results are directly fed to the thermal model, which uses HotSpot's block-model to extract the temperature for each microarchitectural component based on the layout shown in Figure 3.6. The block-model is employed, since it allows a very fast estimation (less than 1 ms) and is almost as accurate as the much slower grid-model (temperature difference is less than 1 °C). As a result, the power and thermal model allow to obtain high-resolution results very frequently, i.e. the access frequency can be up to 1 kHz.

In addition, both models will feature a self-calibration mechanism, i.e. the available power and thermal sensors are employed to improve the accuracy of the power and temperature models. This is of great importance, as nowadays, due to extensive process variation, every manufactured processors behaves slightly different [180]. However, the software-based power and thermal models only reflect the "average" behavior. Therefore, a calibration is required to enhance the model accuracy. In case of our Intel-based experimental platform the access to the sensors can be performed using the SPRs `MSR_PP0_ENERGY_STATUS` and `IA32_THERM_STATUS`. The first one delivers the combined power consumption of all cores including their L1- and L2-caches, while the second register contains the maximum per-core temperature.

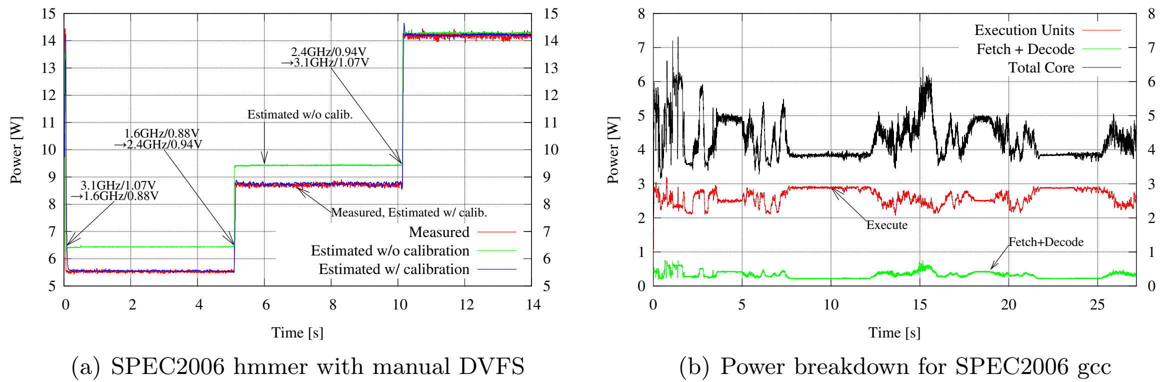
The first observations made with this setup for an Intel Core i5-3540[5] and Core i5-2400[6] are very interesting. First of all, we already implemented the calibration feature for the power model, and as a result the estimation inaccuracy for the combined power consumption is less than 1 % (over the SPEC2006 benchmark suite) w.r.t the readout of the available power sensor, as depicted in Figure 3.7. Moreover, even without calibration the power model shows the correct power trend during the execution of an application. In addition, frequency and voltage changes due to the DVFS feature of these processors are accurately captured as shown in Figure 3.8(a). As a result, the high resolution of ExtaTime's power model allows deep insights in the power consumption behavior of a microprocessor (see Figure 3.8(b)), which cannot be obtained with a single sensor or with simple linear regression models that are often proposed in literature [177, 181–187] (see Figure 3.7(a)). In addition, this information can be exploited to predict thermal hotspots more accurately compared to only a few DTSs that monitor only a few components. For instance, as depicted in Figure 3.9 which was obtained with the thermal model using fine-grained power data, for some application sequences the ALUs have a higher temperature than

[5]Intel Core i5-3450: 22 nm, 1.6 GHz–3.1 GHz, 0.88 V–1.07 V, DVFS and all sleep modes activated
[6]Intel Core i5-2400: 32 nm, 1.6 GHz–3.1 GHz, 1.06 V–1.22 V, DVFS and all sleep modes activated



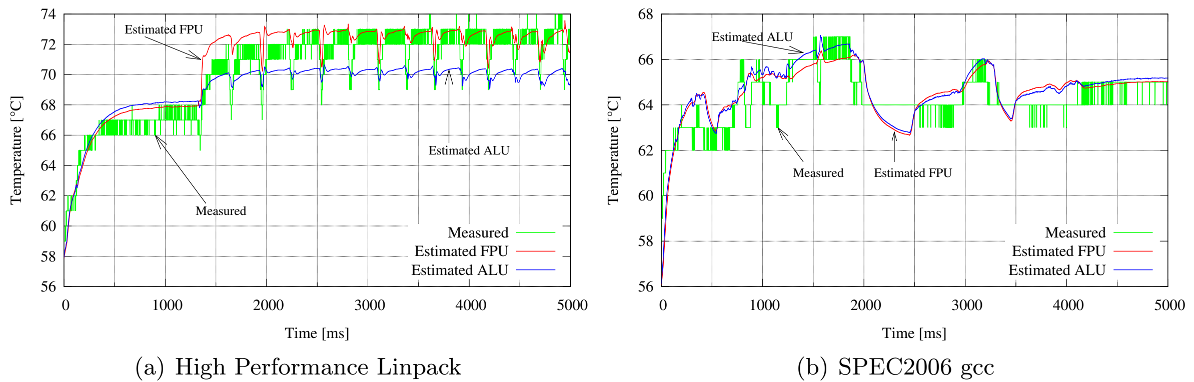
(a) Intel Core i5-3450 (b) Intel Core i5-2400

Figure 3.7.: Power consumption of the SPEC2006 gcc benchmark running on all cores for two different processors (measured=sensor readout)



(a) SPEC2006 hmmer with manual DVFS (b) Power breakdown for SPEC2006 gcc

Figure 3.8.: Power consumption of two different applications for the Intel Core i5-3450 (measured=sensor readout)



(a) High Performance Linpack (b) SPEC2006 gcc

Figure 3.9.: Temperature trace for two applications running on all cores for the Intel Core i5-3450 (measure=sensor readout representing maximum temperature)

the FPUs (Floating Point Units), while it is the other way around for other sequences. Thus, if only one sensor is placed in the execution units (as it is typically the case [188, 189]), the hottest component might not be monitored, which can affect reliability. Consequently, this fine-grained power and thermal information helps to improve the microprocessor reliability. Furthermore, Figure 3.9 shows also that the thermal model of ExtraTime can capture the temperature behavior of a real processor very well. Hence, in summary, the power and thermal model of ExtraTime are capable to accurately reflect the behavior of a real processor executing real workloads, and consequently these models are a good choice for our simulation setup that is used to evaluate various aging mitigation approaches.

3.4. Modeling and Analysis of Transistor Aging at Register-Transfer-Level

While the objective of ExtraTime is to allow a fast “first-order” aging prediction at high abstraction levels, the objective of the RTL-platform is to obtain a very accurate aging estimation taking also circuit- and gate-level information into account. In addition, this framework should also be able to capture the application and (micro)-architecture influence on aging to achieve a realistic aging estimation. Therefore, we developed a novel RTL-based aging analysis framework that relies on standard industrial design tools in combination with accurate gate-level aging models. In this section, we present this RTL-platform including its aging models in detail. Furthermore, the differences to other low-level frameworks are discussed.

3.4.1. Overview and Introduction of the Basic Components

As discussed in Section 2.3 the aging rates of transistors strongly depend on their duty cycles and switching activities. Moreover, these values influence the power consumption of the microprocessor and hence the temperature distribution, which in turn also affects the aging rate

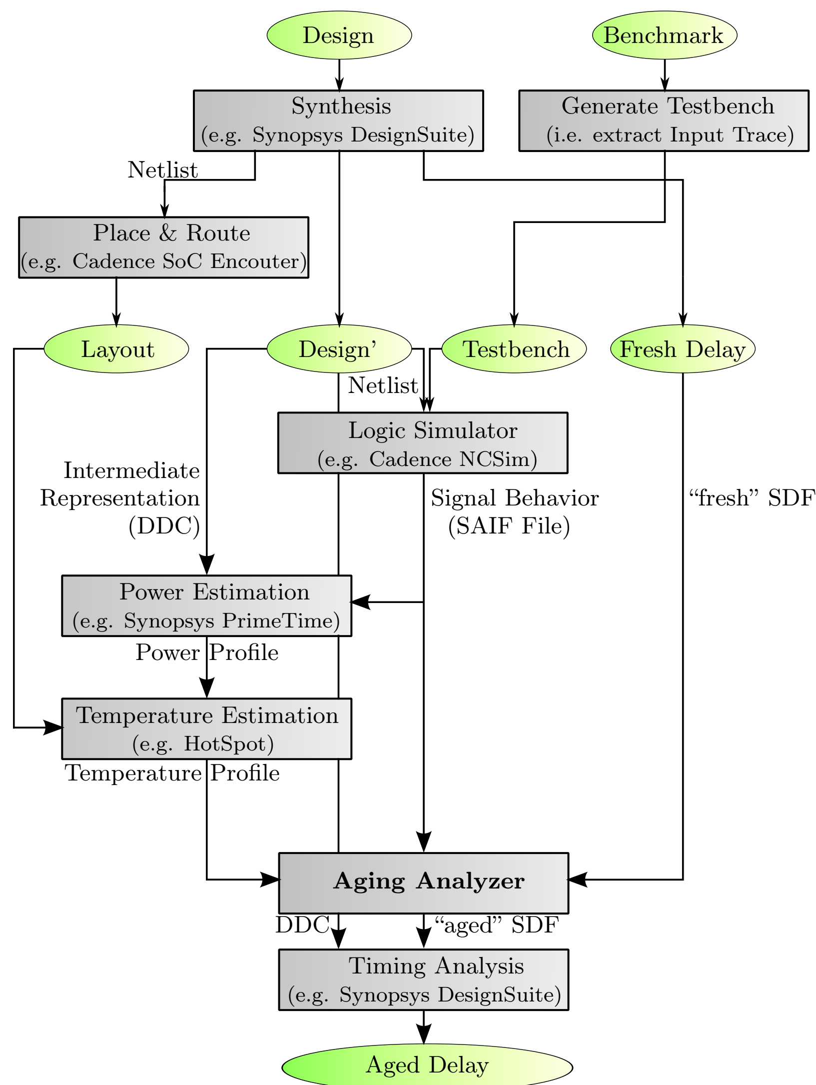


Figure 3.10.: Basic components of the RTL aging evaluation platform

of the transistors. Hence, it is very important for an accurate aging estimation, to precisely calculate these values. Therefore, a gate-level description of the design under investigation (e.g. a pipeline stage or an ALU) is necessary, to monitor all signals, while the circuit (e.g. processor) executes representative workloads (applications).

Hence, the first step within the RTL-framework, depicted in Figure 3.10, is to generate a gate-level description (netlist) of the design under investigation (DUI) using a standard synthesis software (e.g. Synopsys Design Compiler). During this step, also the fresh, not-yet aged circuit and gate delays are extracted and stored in form of a *Standard Delay Format* (SDF) file[7] (see Figure 3.11). In addition, the circuit description is stored using an intermediate representation format (*DDC* file), that contains all information about the circuit required for the power estimation or timing analysis steps.

Afterwards, the netlist of the DUI is used by a place-and-route software (e.g. Cadence SoC Encounter) to obtain the design layout which is required for an accurate thermal profile estimation. In parallel, the chosen workload is transformed to a testbench for the DUI, i.e. the workload is simulated and the primary input behavior of the DUI is extracted and then used to generate a testbench.

As a next step, the gate-level description of the DUI is used together with the testbench to simulate the circuit behavior, when the design executes a representative workload. During this logic simulation (e.g. performed with Cadence NCSim), the behavior of all internal signals is obtained and stored in a *Switching Activity Interchange Format* (SAIF) file, which contains the switching activity as well as the *signal probability*[8] for each signal in the design. This SAIF file is then used by a power analysis tool (e.g. Synopsys PrimeTime) together with the DDC description of the DUI to annotate the design with the extracted signal behavior[9] in order to obtain the power consumption for each (sub)-block of the DUI[10]. Afterwards, the results are given to a temperature model (e.g. HotSpot [3]), which calculates the temperature for each block. Since, the temperature of a block strongly depends on the temperature of its surrounding blocks, the temperature model has to consider the layout of the entire design.

The following step is to extract the delay degradation of the DUI. Therefore, the temperature information and the SAIF file are given to an aging estimation tool, which was developed in the scope of this thesis and is described in more detail in the next subsection. This tool accurately calculates the delay degradation for each gate in the DUI after a given amount of time, and generates an “aged” SDF file containing the degraded gate delays. Finally, this “aged” SDF is read by the synthesis tool, the design is annotated accordingly[11] and a new timing analysis is performed to extract the aged circuit delay. Together with the information

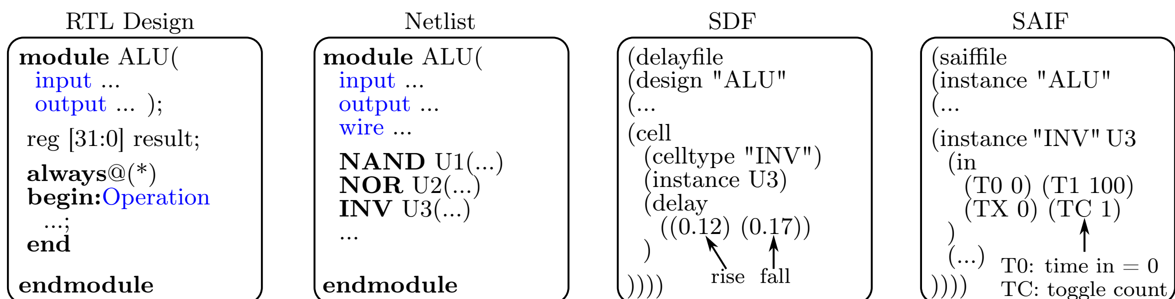


Figure 3.11.: Illustration of inputs and intermediate outputs of the RTL-platform

[7]Using Synopsys Design Compiler the corresponding commands are: `report_timing` and `write_sdf`
[8]Signal probability is the probability of a signal having the logic value ’1’
[9]Using Synopsys tools the corresponding command is `read_saif`
[10]Using Synopsys tools the corresponding command is `report_power`
[11]Using Synopsys tools the corresponding command is `read_sdf`

Synthesis + Timing Estimation	Synopsys Design Compiler D-2010.03-SP4
Synthesis Library	TSMC 65 nm General Purpose Library
Simulation + SAIF-Generation	Cadence NCSim 12.10-s005
Power Extraction	Synopsys PrimeTime D-2010.03-SP4
Temperature Extraction	HotSpot 5.02 [3]
Aging Analysis	Self-Developed C++-Tool

Table 3.5.: Tools employed in the RTL aging estimation framework

about the most critical path at design time and the given clock period the delay degradation as well as the MTTF for the DUI can be estimated. In this regard it is important to note that the timing analysis of the synthesis tool considers all possible paths, and thus the extracted delay information is very accurate. In contrast, many other aging estimation approaches (see Section 3.4.4) evaluate only a subset of potentially critical paths. However, since real designs are very balanced, several million paths can have almost the same delay [63], which makes these approaches almost intractable and less accurate.

Please note that the described cross-layer platform (gate- to application-level influences are visible) can be used with any synthesis, power estimation or temperature model. For this thesis, we used the tools described in Table 3.5. Moreover, as this framework requires an RTL design of a microprocessor that can execute real workloads, we have chosen the FabScalar microprocessor [1] (academic superscalar processor with configurable out-of-order instruction pipeline) and OpenSPARC T1 [6] (industrial processor with an in-order, 6-stage pipeline that features 4-way simultaneous multithreading). Both processors are synthesizable and include frameworks to execute real applications, i.e. the testbench generation is automatically handled by the opensource frameworks that come along with the microprocessor RTL description. Moreover, it is important to note that the simulation speed of the logic simulator is rather low, which means that only a few million clock cycles can be simulated within a few hours. Consequently, only short periods of applications and not their complete execution time can be simulated. Therefore, the influence of the operating system is not considered in this framework, whereas it is part of ExtraTime.

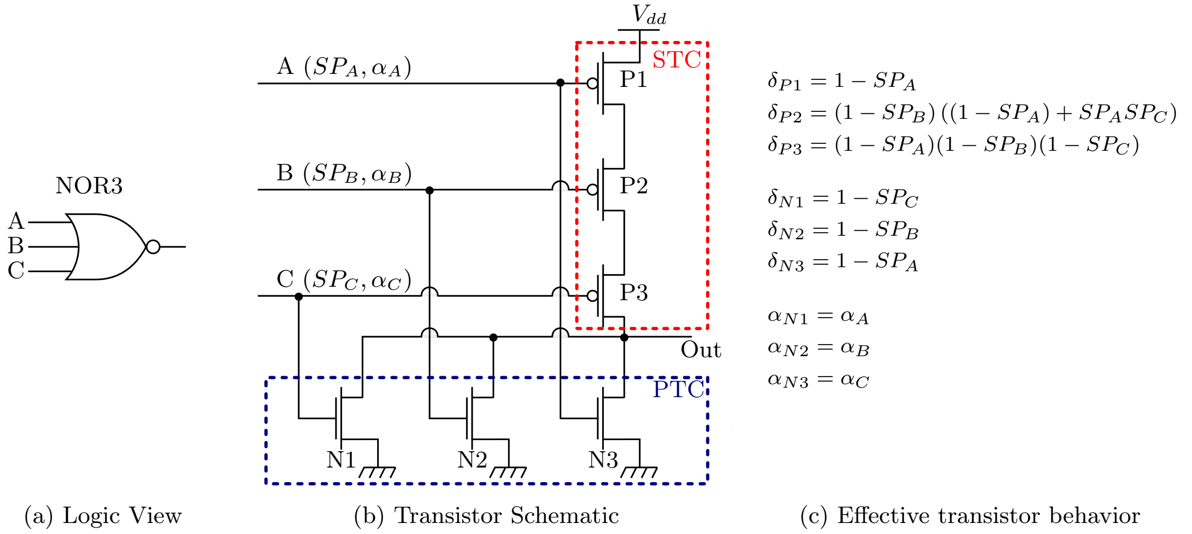
3.4.2. Aging Analysis Tool

Our developed aging analysis tool estimates the delay degradation for each gate in the circuit under investigation. As the first step, the SAIF file is parsed and for each gate the input signal probabilities and the input switching activities are gathered. Afterwards, the effective duty cycle and switching activity for each transistor in each gate is calculated considering the stacking effect[12]. For this purpose, the accurate modeling approach proposed in [143] is used, according to which all transistor stacks are divided into serial transistor connections (STCs) and parallel transistor connections (PTCs). As a result, the following rules can be derived:

- In an STC, a transistor is ON, if either all upper transistors in the STC are ON, or if at least one lower transistor in the STC is OFF.
- In an STC, a transistor can make a transition, if either all lower transistors in the STC are ON, or if at least one upper transistor in the STC is OFF.
- In a PTC, the ON- or OFF-state of a transistor as well as the possibility of a transition is independent from the other transistors in the PTC.

Based on these rules, which were validated and shown to be accurate in [143], the effective duty cycle and switching activity for each transistor can be derived, as the example in Figure 3.12 illustrates. By that means, the threshold voltage shift due to BTI and HCI for each transistor

[12]The stacking effect means that the duty cycle (switching activity) of a transistor depends not only on its own state, but also on the position inside the transistor stack and the states of the other transistors



(a) Logic View (b) Transistor Schematic (c) Effective transistor behavior

Figure 3.12.: Illustration of the employed aging modeling approach

after a given amount of time t_{aged} (e.g. 3 years) can be obtained considering the circuit temperature according to the transistor-level models detailed in Table 2.1 and Table 2.2. Therefore, similar to the approach in ExtraTime (see Section 3.2.2), the average transistor behavior observed during the simulation time is assumed to be preserved for t_{aged}. In other words, the simulation time is extrapolated to t_{aged}. Finally, with this information, the delay degradation (rise and fall delay) of the entire gate can be extracted. In this regard it is important to note that NBTI affects the rise delay as explained in Section 2.1 and HCI as well as PBTI contribute to the degradation of the fall delay.

Having the delay degradation for all gates, an "aged" SDF is generated by the aging analysis tool. Therefore, the default SDF with "fresh" delays is parsed, all delays are adjusted according to the obtained relative degradation data ($d_{aged} = \Delta^{rel} d \cdot d_{fresh}$), and finally an updated SDF is written back. Afterwards, the "aged" SDF is read by a synthesis tool and a timing analysis is performed to obtain the overall delay degradation of the entire circuit after t_{aged} as well as its TTF (by sweeping t_{aged}). Since the TTF is extracted based on the average circuit behavior, as just explained, the MTTF (i.e. lifetime) can be also represented by this value. *Consequently, in this thesis, the MTTF stands for the expected time until the delay of a circuit exceeds its guardband (timing violations start to appear) for a given application.*

3.4.3. Enhancements

Since the RTL-framework is based on commercial, highly optimized design tools, the runtime for the different steps except the logic simulation is very small. For example, the entire flow, starting with a behavioral RTL description of a load-store-unit with more than 50,000 gates requires less than 10 minutes (for synthesis, power estimation, thermal profiling, aging estimation and timing analysis) without the logic simulation step. However, the runtime of the logic simulation can be easily more than an hour for designs of such a size for a simulation time of 10^6 clock cycles. Thus, the overall runtime for the aging estimation flow is dominated by the time required to perform post-synthesis simulations for a sufficiently large number of clock cycles. In this subsection we will discuss two possibilities to eliminate these costly simulations and how the accuracy of the aging estimation is impacted using these techniques.

Since the post-synthesis simulations are performed to extract the (average) signal properties over a long period of time, the first approach is to use a default annotation (e.g. $\delta = 0.5$, $\alpha = 0.01$) for all primary inputs of the DUI and to propagate these information through the

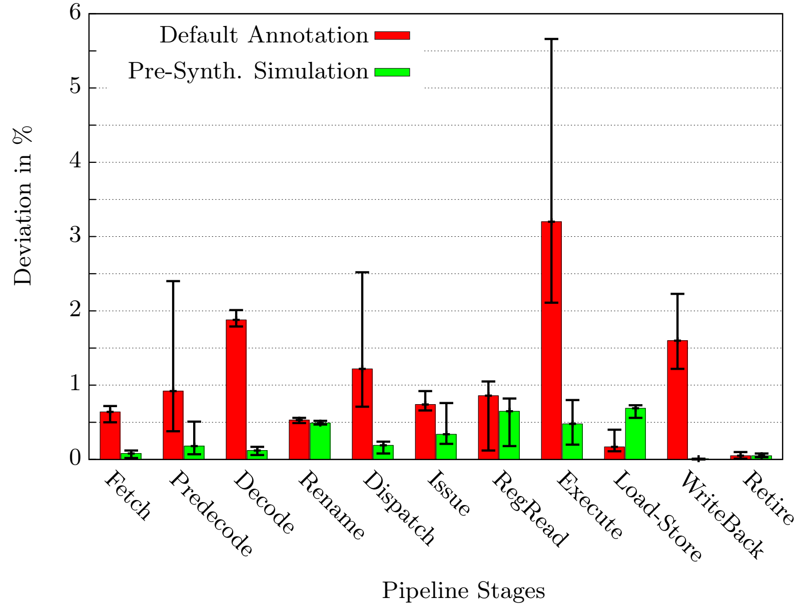


Figure 3.13.: Inaccuracy (min, avg, max for six SPEC2000 benchmarks) in terms of aging rate of the default annotation (duty cycle = 0.5, switching activity = 0.01) and a pre-synthesis simulation compared to comprehensive post-synthesis simulations for the FabScalar microprocessor (inaccuracy is calculated according to $|\Delta^{rel}_{accurate}d - \Delta^{rel}_{approximated}d|$)

remaining design to get the signal properties for all internal signals. In contrast, the second technique uses the real signal properties for all primary inputs, outputs and flipflops (extracted during higher-level, pre-synthesis simulation steps) and propagates these information through the remaining design. In both ways, the costly post-synthesis simulations can be avoided and the synthesis tool can be used to perform the signal property propagation to extract a SAIF file for the entire design[13]. However, the cost for the speedup are inaccurate signal properties compared to post-synthesis simulations as the signal property propagation is never 100 % accurate [190]. As a result, the aging estimation using these two techniques will be inaccurate. In fact, as shown in Figure 3.13, the inaccuracy for the first approach can reach almost 6 % in case of the FabScalar microprocessor which corresponds to an inaccuracy of more than 10x in terms of MTTF. This also shows how important it is to accurately capture the workload influence (i.e. to have a cross-layer solution), since not representative data can lead to huge over- or underestimations of the microprocessor lifetime. In contrast, the second approach is much more accurate (less than 1 % deviation), which is due to the fact, that real data is used to annotate the design. Therefore, this approach can be chosen whenever some small inaccuracy is acceptable or post-synthesis simulations are not feasible.

Please note that the high-level simulations, that are necessary for the second technique, are always performed during the typical design-flow and hence do not need to be conducted additionally. For example, during the design of a new microarchitecture or the verification of a behavioral RTL model such simulations are performed. Hence, no additional runtime is required, only the necessary data needs to be stored for the future.

3.4.4. Comparison with other Low-Level Platforms

Beside our cross-layer RTL-platform, there are also various other low-level aging evaluation platforms available in literature. However, many of them are incapable of handling real work-

[13] Using Synopsys tools the corresponding command is `write_saif -propagated`

loads and are intended to be used in combination with artificial (very often random) input combinations [63, 191, 192]. Hence, investigations of application-level effects is impossible with these frameworks.

Another challenge of many approaches is that only a set of potentially critical paths is considered during the analysis (*path-based*[14] platforms) [146, 191, 192]. However, as shown in Chapter 4 and in [49, 63] the critical path changes over time due to aging. As a result, the path-based approaches are always likely to be inaccurate or a huge number of paths (in complex designs more than 10^8 paths according to [63]) has to be considered, which leads to long evaluation times. Thus, for realistic designs, path-based approaches are intractable. Instead, *block-based*[15] approaches such as the one proposed in [63] or those employed by commercial synthesis tools (that are used by our framework) are favorable, since these consider all possible paths. In addition, since the number of nodes is considerably smaller than the number of paths, the block-based techniques are much faster than path-based solutions.

Moreover, various platforms do not use SAIF files to store and evaluate the signal probabilities and switching activities. Instead, *Value Change Dump* (VCD) files are generated [191, 193]. In a VCD file the exact signal behavior over time is captured, e.g. when a signal makes a transition from '1' to '0' or vice versa an entry is generated in the VCD file. As a result, these files consume a lot of memory (several GByte), if more than a few thousand clock cycles are simulated. In addition, the analysis of a VCD is very time consuming, which considerably increases the runtime of the power and aging estimation steps. As a consequence, the evaluation of the mentioned load-store-unit requires several hours instead of a few minutes required by our platform due to the usage of SAIF files.

In summary, our RTL platform is capable of executing real workloads and accurately analyzing transistor aging at gate-level in a block-based manner. Nevertheless, the runtime of a complete aging evaluation phase is still acceptable. Both aspects together make our platform favorable over all previously mentioned frameworks.

3.5. Summary

In this chapter, we presented two cross-layer multi-objective aging analysis frameworks and an experimental platform to support simulation data with real experimental results. The first cross-layer framework is the architectural platform ExtraTime which is built on top of a performance simulation. Furthermore, it employs power, thermal and aging models at architecture-level. Hence, this platform can be used, when detailed circuit-level information is not required or not available. The second aging analysis platform is an RTL-framework, that is designed for investigations that require detailed circuit- and gate-level information. Therefore, it makes use of standard industrial electronic design tools and accurate gate-level aging models. As a result, this framework allows an accurate and detailed aging analysis. The experimental platform is built around a real hardware system that incorporates high-resolution power and thermal models to accurately monitor the power consumption and temperature of various microarchitectural microprocessor components. By that means, it enables us to perform detailed power and thermal evaluations of real microprocessors.

[14] A path-based timing analysis sums all gate delays along specified paths for a given set of paths. At the end, the maximum of all path delays corresponds to the worst-case circuit delay, if the right paths were selected.

[15] A block-based timing analysis calculates the signal arrival times for each node working forward through the circuit. At the end, the maximum of all primary output and flipflop arrival times corresponds to the worst-case circuit delay.

Part III.

Aging Mitigation Techniques

OVERVIEW

As described previously in this thesis, accelerated transistor aging due to BTI and HCI significantly impairs the microprocessors lifetime. Therefore, efficient aging mitigation techniques are required to tackle this challenge, and by that means to co-optimize lifetime (i.e. MTTF), performance, power (energy) and cost. Since transistor aging leads to the challenge of constantly changing system conditions, both *design time* and *runtime* schemes based on sense-and-adapt as well as prediction-based strategies are necessary to efficiently improve the system lifetime [83]. In this regard, the design time solutions have to consider lifetime as one optimization objective and tune the design accordingly based on a given set of representative workload scenarios, while the runtime techniques have to deal with the constantly changing conditions and adapt the system to avoid critical states. Thus, the combination of both schemes enables effective and holistic aging mitigation solutions. In this part of the thesis, four complementary cross-layer aging mitigation approaches are proposed and analyzed using the previously introduced cross-layer multi-objective frameworks. Each of them addresses different layers in the abstraction stack and different microprocessor components as depicted in Figure 3.14.

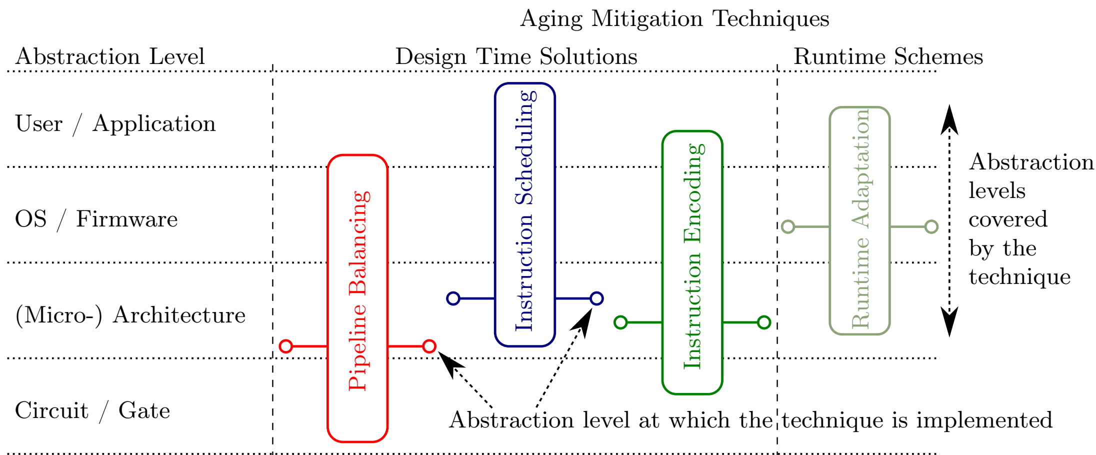


Figure 3.14.: Overview of the proposed aging mitigation techniques

At first, an aging-aware design methodology for instruction pipelines is proposed to improve the overall lifetime of a microprocessor (Chapter 4). The idea behind this approach is to balance the pipeline stage delays at the end of the desired lifetime rather than at design time (traditional scheme), since the traditional approach leads to the problem that a single pipeline stage limits the overall microprocessor lifetime and the operational clock frequency. Hence, the remaining stages are overdesigned using the traditional design approach, which is very inefficient. To perform the proposed aging-aware instruction pipeline optimization (at microarchitecture-level), circuit-level modifications are mandatory and accurate application-

level information is also required.

The second approach employs a special instruction scheduling technique to address the degradation of execution units, which are one of the most aging-critical components in modern microprocessors (Chapter 5). The novelty of this scheduling policy is that the timing-criticality (i.e. how much slack each instruction has at circuit-level) is considered during the scheduling process to reduce the workload of functional units that execute critical instructions. Hence, their idle time increases which can be exploited to considerably improve the lifetime of the functional units compared to non-cross-layer balanced scheduling approaches.

Beside the functional units also the decoding stages of the instruction pipeline can become critical. To tackle this challenge, we propose an aging-aware instruction set encoding technique in Chapter 6. The idea of this approach is based on the observation that the instruction opcodes significantly affect the aging rates of the instruction decoders. Consequently, the instruction opcodes are optimized to reduce the delay degradation of the decoding stages considering the influence of different instructions as well as circuit-level implications.

Finally, a dynamic runtime adaptation technique is employed to proactively mitigate aging of the entire microprocessor (Chapter 7). Therefore, an expert system is designed to continuously monitor and analyze the system states in terms of performance, temperature, power and in particular aging. Whenever the evaluation indicates a critical system trend, e.g. rapidly increasing aging rates, the supply voltage and frequency are proactively adjusted to avoid critical system states before these actually occur. In other words, this proactive DVFS policy is a combination of sense-and-adapt and prediction-based approaches. For this purpose, various parameters at (micro)-architecture-level up to application-level are monitored and also user specific parameters are taken into consideration during the system adaptation process.

CHAPTER

FOUR

AGING-AWARE DESIGN OF MICROPROCESSOR INSTRUCTION PIPELINES

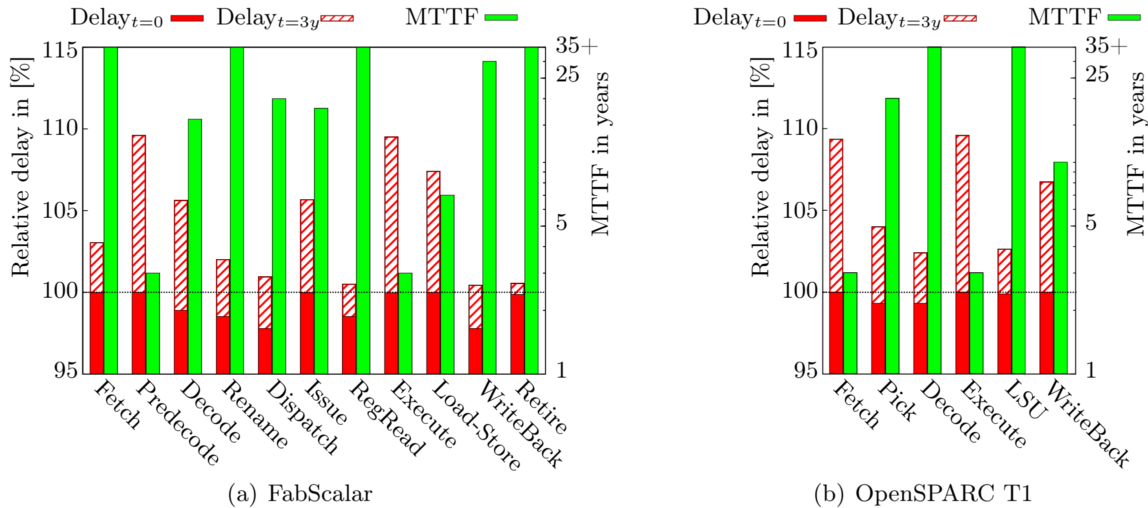
In this chapter, we present our novel approach for the design of an instruction pipeline. First, we explain the problem of the traditional instruction pipeline design paradigm and motivate why it should be replaced with our proposed aging-aware design methodology. Afterwards, we present the key idea as well as the design solution itself. Next, we compare the envisioned design solution with the traditional technique for two complementary microprocessors. These comprehensive tests show that the novel design paradigm can considerably enhance the processor lifetime compared to the standard design scheme, while the performance (i.e. clock frequency) can be maintained. At the same time, area and power consumption are improved as well. Finally, a summary of related work is provided followed by concluding remarks.

4.1. Problem Introduction and Motivation

Nowadays almost all microprocessors ranging from low-power embedded parts to high-performance processors use a pipelined architecture to increase the instruction throughput and by that means the performance [24]. To maximize the performance, designers follow the same paradigm since the dawn of the first pipelined microprocessors: They try to balance all pipeline stage delays at design time (called: *delay-balanced pipeline*). The advantage of this approach was the combination of high throughput together with efficient energy and area usage. This was due to the fact, that as long as a pipeline stage is faster than the slowest one (which determines the clock frequency), it can be often made slower using gate sizing or higher threshold voltage to save energy and die area [194, 195].

However, in nanoscale CMOS technologies, accelerated transistor aging due to BTI and HCI leads to increasing path delays and so degrades pipeline stage delays during runtime. Hence, nowadays the clock frequency of the shipped parts can no longer be set according to the worst-case delay at design time (t_{design}). Instead, manufacturers have to add safety margins (guardbands) to their delay-balanced designs, to ensure that the chips will be functional for a certain lifetime (t_{target}). A major problem associated with this guardbanding technique is that the aging rates vary widely among pipeline stages, due to different temperature and usage rates. Hence, although the original pipeline was delay-balanced, after some operational runtime the stage delays become highly imbalanced. This also affects the MTTF of different pipeline stages, which vary tremendously. Thus, a single pipeline stage can limit the overall microprocessor lifetime and performance, as the guardband and hence the clock frequency is mandated by the slowest stage at t_{target}.

To illustrate this circumstance, two microprocessors, namely FabScalar and OpenSPARC T1,



(a) FabScalar

(b) OpenSPARC T1

Figure 4.1.: Delay at design time and after 3 years (normalized to worst case design time delay) and MTTF for different pipeline stages for two processors designed according to the delay-balanced paradigm

were analyzed. For this purpose, both processors were synthesized with the Synopsys Design Compiler using the TSCM 65 nm library and evaluated with the framework detailed in Section 3.4. To achieve a lifetime of at least 3 years (= t_{target}) a timing guardband of 10 % was employed. The results of this analysis, depicted in Figure 4.1, clearly show that different pipeline stages have different wearout rates[1]. While the delay of FabScalar's execution stage increases by almost 10 % within 3 years, the delay of the retire stage increases by less than 1 %, although their delays at design time were similar (≈ 1.35 ns). Moreover, at design time, the load-store stage is the most critical pipeline stage, while after some operational time it is the execution stage. Hence, the most critical stage (i.e. most critical path) can change over time. In addition, also the imbalance in terms of MTTF can be huge. For instance, between the execution stage of FabScalar, which starts to fail first, and the retire stage, there is a factor of more than 20x difference. In fact, this is not a FabScalar specific issue, as similar results were observed for OpenSPARC T1 (see Figure 4.1(b)). *This means that one pipeline stage already produces timing failures, while other stages are still operating correctly. Hence, the latter are overdesigned (i.e. too fast). Consequently, slow-aging stages can have less slack[2] to save area and energy, while fast-aging stages should have more slack to improve the overall MTTF.*

In summary, in nanoscale CMOS technology nodes a delay-balanced design is no longer appropriate, as a single pipeline stage will limit the overall microprocessor lifetime and performance. Instead, a new design methodology is required to balance the lifetime of all pipeline stages and to co-optimize reliability, performance, power and area. For this purpose, we propose a radically new *MTTF-balanced pipeline design scheme* to replace the traditional delay-balanced paradigm. Using this paradigm, slow-aging stages have less slack compared to a delay-balanced design to save area and energy, while fast-aging stages have more slack, to improve the overall MTTF. As a result, the MTTF values of all pipeline stages are balanced, and as a direct consequence, the stage delays are also balanced at t_{target} rather than at t_{design}. By that means, the full optimization potential for MTTF, area, power and performance can be exploited.

[1] For other processor designs or other technology libraries Figure 4.1 might look different, i.e. other stages age faster, have different MTTF values, etc. However, the overall observation of delay and MTTF imbalance after a certain operational time remains valid (e.g. in [146] similar results were reported for the IVM microprocessor), as also shown by the two complementary processors chosen for this study.

[2] Slack is the time difference between the signal arrival time and the clock edge (see Figure 2.3)

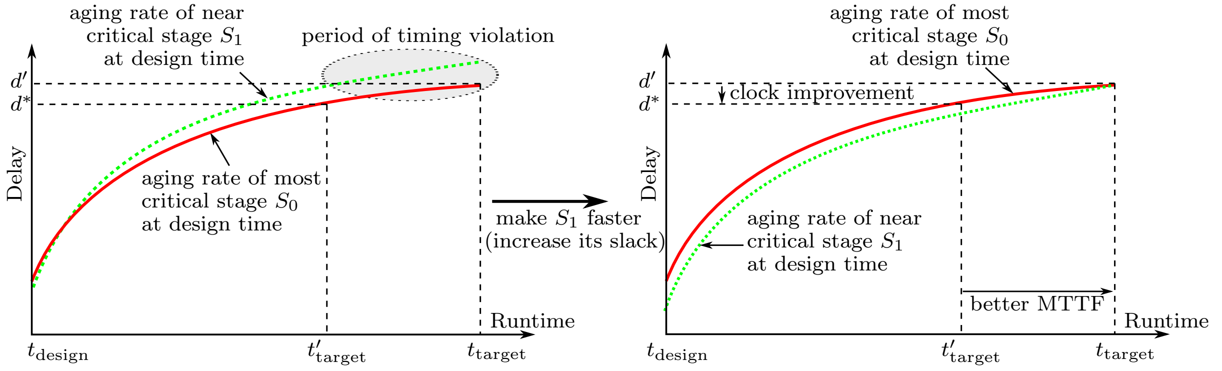


Figure 4.2.: Abstracted graphic to illustrate the effect of using an MTTF-balanced design on MTTF and performance
Left (Delay-Balanced): At t_{design} stage S_1 has less delay than stage S_0, but ages faster $\Rightarrow$ For clock period $= d'$ the lifetime $= t'_{\text{target}}$
Right (MTTF-Balanced): S_1 is accelerated $\Rightarrow$ After t_{target} stages S_0 and S_1 have same delay (d') $\Rightarrow$ longer MTTF possible (t_{target} instead of t'_{target})
Alternatively: For same MTTF (t'_{target}) a smaller clock period is possible (d^* instead of d')

4.2. Main Idea

As motivated before, the key idea of the MTTF-balanced design paradigm is to balance the delay and MTTF of all pipeline stages at t_{target}. In this regard, depending on the criticality of different pipeline stages at t_{target}, there are two possible optimization strategies:

- Stages that are faster (i.e. have more slack) than the critical stage after t_{target} can be designed slower (i.e. with less slack), by applying appropriate gate-sizing techniques, using a higher threshold voltage, etc., leading to extra energy and area savings [194, 195].
- If a stage S_1 degrades faster than the design-time-critical stage S_0, it can be designed faster (i.e. with more slack). This can be used in two different ways, both shown in Figure 4.2, where the dotted line represents the slow-aging, design-time-critical stage and the solid line, the fast-aging stage. First, the clock frequency (i.e. clock period) can be kept constant, so that a better MTTF can be achieved (that means that t_{target} can be increased). In Figure 4.2 that means that the clock period remains at d' and that the target lifetime increases to t_{target}. In the second case, the MTTF is kept constant (i.e. equal to t'_{target}), so that the clock period can be reduced (i.e. less guardband and hence higher frequency, meaning higher performance). Using the annotations from Figure 4.2, it means that the new clock period is $d^* < d'$.

  Depending on whether the first or second case is chosen as the optimization target, the design should be balanced either at t_{target} or t'_{target}, respectively. Since both cases require the same optimization process (once with t_{target} and once with t'_{target} as input), only the first case is presented and analyzed in this thesis.

In summary, slow-aging stages should be designed with less slack (i.e. slower) to save area and energy, while the timing slack for fast-aging stages should be increased (i.e. speed-up) to improve their MTTF and in turn the MTTF of the entire microprocessor or the overall performance (i.e. clock frequency). Thereby, the key design aspect is that the pipeline stage delays should be balanced after the target lifetime and not at design time. Since this is not achievable using the traditional delay-balanced design approach, we propose a new MTTF-balanced pipeline design which will be explained in detail in the following section.

Please note that it can happen that the design-time-critical stage is also the aging-critical stage. In this case, which could not be observed for FabScalar and OpenSPARC T1, only area and energy/power savings can be achieved using the MTTF-balanced design approach, while the overall MTTF and performance remain the same as with the traditional delay-balanced design paradigm. Nevertheless, since nowadays a majority of all systems is power constrained [196], the MTTF-balanced design scheme is also favorable in this case.

4.3. Aging-Aware Pipeline Design Methodology

In order to apply this idea to real microprocessors the design flow needs to be adjusted. Therefore, the flow to generate an MTTF-balanced pipeline is explained next. In this regard, for the matter of simplicity, we will only refer to the targeted lifetime t_{target}.

4.3.1. Generation of an MTTF-balanced Pipeline Design

The transformation flow, detailed in Algorithm 4.1, is a multi-purpose flow that can be used for various optimization objectives such as getting the best MTTF, while maintaining a given clock frequency or extracting a design that is as fast as possible for a given target lifetime t_{target}. The last case will be explained in detail now.

The starting point of the transformation process is a delay-balanced design, as it is used nowadays (Step 1). Next, the delay, d^*, after the given target lifetime t_{target} of the design-time-critical stage is extracted using the flow presented in Chapter 3.4 (Step 2). Since this stage cannot be designed any faster (otherwise it would not be critical at design time), the clock period of the final MTTF-balanced pipeline cannot be smaller than this delay. Since the final design should be as fast as possible, d^* will work as a reference for the clock period.

The next step (Step 3) is to extract the delay d_i of each pipeline stage after t_{target} and to compare it with d^*. If the delay is smaller than d^* (i.e. the stage is faster than necessary), a new, slower version of this pipeline stage will be generated. Therefore, we adjust the timing constraints for this pipeline stage and re-synthesize it. As the synthesis tool supports gate-sizing, path reorganization and time borrowing, all these techniques will be applied in parallel to optimize for delay, power and area efficiency. In addition, a higher threshold voltage can be used as well [194, 195]. However, as a result it is possible that different pipeline stage designs result in the same MTTF. In Section 4.3.2, we will explain how such scenarios are handled and how finally one design is chosen. In case re-synthesis is not feasible, it is also possible to modify only small sub-circuits or gates [197].

If the delay d_i is greater than d^* (i.e. stage is slower than necessary), a new, faster version (using gate sizing, etc.) will be generated. If this is not possible, the final design has to use a clock frequency of at least d_i. Hence, d^* will be increased and set to d_i. In that case Step 3 has to be restarted.

After all pipeline stages are analyzed and eventually modified, their new delay information is extracted (Step 4). Here it is extremely important to investigate all stages together and not only those that have been changed in Step 3. This is due to the fact that as long as one stage is modified, the power consumption and thus the temperature distribution will change, which can affect also the wearout and hence the delay of other stages. If it is detected in Step 4 that a stage, which was previously faster than necessary, is now slower than necessary, the changes leading to this situation will be reverted and the previous implementation will be used. Since these situations are undesired, the delay differences between the new and the old implementation should be very small (see Section 4.3.3 for more details).

If there is at least one modified stage remaining after Step 4, again Step 3 followed by Step 4 will be executed until no pipeline stage is modified anymore, i.e. until no stage can be tuned

```
1. Generate a delay-balanced design
2. d* = delay at t_target of stage, that is critical at t_design
   d' = delay at t_target of stage, that is critical at t_target
   d~ = given clock delay in case there is a clock target and no lifetime target
/* => clock period of delay-balanced design d_clk = d' > d* */

/* Start transformation procedure */
3. Forall stages i = 0,...,n do
     Extract d_i = delay at t_target of stage i
     stage^i_old = current version of stage i
     /* if stage is faster than necessary */
     If (d_i < d*) then
       stage^i_new = new version of stage^i_old with more delay at t_design
     /* if stage is slower than necessary */
     Elseif (d_i > d*) then
       stage^i_new = new version of stage^i_old with less delay at t_design
       /* if no (further) speedup is possible adjust d* */
       If (stage^i_new == stage^i_old) then
         d* = d_i
         Goto 3. /*restart with new d* */
       Endif
     Else
       stage^i_new = stage^i_old
     Endif
   End

/* Extract new delay information at t_target using flow presented in Section 3.4*/
4. Forall stages i = 0,...,n do
     Extract d_i,new = delay at t_target of stage^i_new
     /* if old version was faster and new version is slower take old one */
     If (d_i,new > d*) and (d_i < d*) then
       stage^i_new = stage^i_old
     Endif
   End

/* If no more modifications are possible, transformation is done */
5. Forall stages i = 0,...,n do
     If (stage^i_old ≠ stage^i_new) then
       Goto 3. /* restart */
     Endif
   End
6. Done. /* generated MTTF-balanced design with d_clk = d* */
```

Algorithm 4.1: Transformation of a delay-balanced design to an MTTF-balanced design

further. When this saturation state is reached, the transformation process is finished.

Please note that in some application areas it might be more important to minimize the die area or energy consumption instead of performance (clock frequency). In that case, the transformation procedure is very similar to the one explained before. The only difference is, that d' is used as reference delay in place of d^*. Hence, no stage will be accelerated. Instead, all stages, beside the one that is critical at t_{target}, will be designed slower, hence with less energy and area consumption.

The flow can also accept a given clock target instead of a lifetime target to find the MTTF-balanced design with the best MTTF. In this case d^* is replaced with $\tilde{d}$ and t_{target} is set according to the lifetime of the design-time-critical stage given the delay target $\tilde{d}$. If during the optimization phase a pipeline stage cannot satisfy the given clock target (slower than necessary), t_{target} will be reduced to the lifetime of this stage (instead of adjusting d^* as shown in Step 3) and the transformation process is restarted (see Step 3).

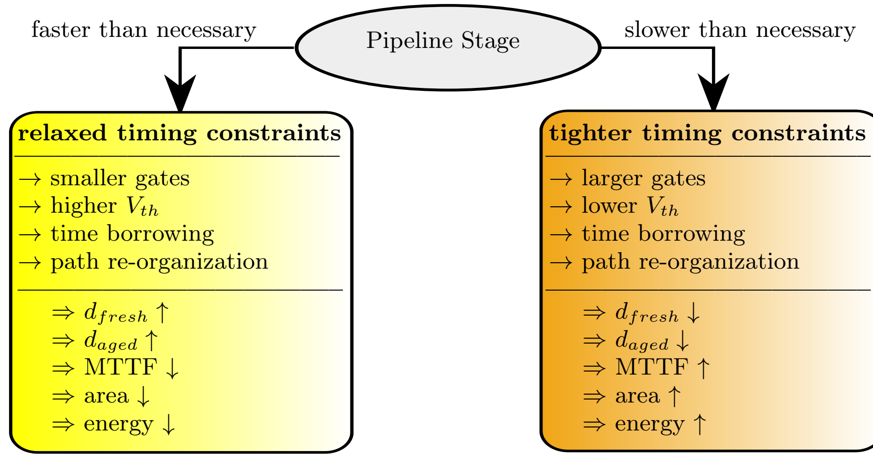


Figure 4.3.: Pipeline stage modification (faster/slower) required to generate a MTTF-balanced design

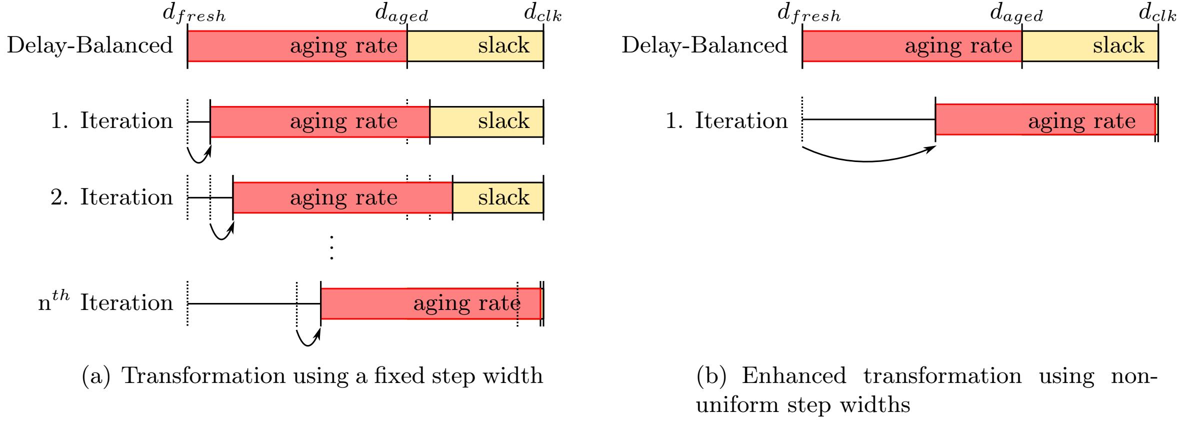
4.3.2. Modification (faster/slower) of a Pipeline Stage

A crucial part of the previously presented transformation flow is the modification of a pipeline stage, i.e. the step to generate a faster or slower version of a pipeline stage. As already mentioned, the synthesis tool is used for this purpose. Therefore, the timing constraints are tightened or relaxed (e.g. by 1 %) and then the pipeline stage is re-synthesized using the new timing constraints, while all other constraints are kept the same. To match the new timing constraints the synthesis tool applies gate-sizing (smaller gates for relaxed constraints, larger gates for tighter constraints), path re-organization as well as time borrowing techniques and also the transistor threshold voltage can be tuned (lower V_{th} for tighter constraints, higher V_{th} for weaker constraints) as illustrated in Figure 4.3. Hence, there are many different ways to obtain an optimized design, e.g. with and without V_{th}-tuning. As a consequence, it is possible that there are several different designs for the same pipeline stage that have different delays at t_{design} but the same MTTF. For example, the Fetch stage of FabScalar can achieve a lifetime of 7 years in two different ways: First, using the nominal V_{th} and a fresh delay of 1.38 ns, and second with a higher V_{th} and a fresh delay of 1.40 ns (see Section 4.4 for more details). In such scenarios it is of course the question, which design should be chosen for the final microprocessor. Therefore, to select one "final" design, the other design parameters such as area and energy consumption are consulted to co-optimize lifetime, performance area and energy/power. As a result, the design with the best energy consumption or smallest area among all available designs can be chosen. However, generating more versions of the same pipeline stage increases the transformation time. Hence, the number of generated versions strongly depends on the time budget of the manufacturer or designer.

4.3.3. Runtime Analysis and Improvements

The runtime for the transformation process from a delay-balanced to an MTTF-balanced pipeline is proportional to the number of necessary iterations, i.e. the number of synthesis steps and delay/MTTF estimation steps. Hence, to reduce runtime, the number of iterations has to be reduced.

The simplest implementation of the transformation flow uses a fixed resolution to tighten or relax the delay constraints in each iteration, as shown in Figure 4.4(a). However, such a uniform step width can lead to a huge number of iterations until the final MTTF-balanced implementation is found. For example, the delay constraint for FabScalar's retire stage could be relaxed by a total of 0.1 ns, which corresponds to at least 10 iterations considering all pipeline stages, if a step width of 0.01 ns is used.



(a) Transformation using a fixed step width

(b) Enhanced transformation using non-uniform step widths

Figure 4.4.: Simple and enhanced transformation flow for a single pipeline stage (that can have weaker timing constraints) from a delay-balanced to an MTTF-balanced design

To improve the runtime of the transformation process, we propose to use a non-uniform resolution as shown in Figure 4.4(b). We observed that tighter or weaker timing constraints have only a limited effect on the delay degradation itself. For example, if the delay degradation of a pipeline stage is roughly 9 % after 3 years, the gate-level modifications to have a faster/slower version of this stage do not affect this value very much. This is reasonable, since gate-sizing or re-organization of only a few paths do not affect the majority of the internal signals, and thus the aging rate is not significantly affected [63]. Hence, the aging rate of the original, delay-balanced version of a pipeline stage can be used to estimate its design time delay (i.e. timing constraints) for the MTTF-balanced version, according to the following equation:

$$d_{fresh}^{new} = d_{clk} \cdot \left(\frac{d_{aged}^{orig}}{d_{fresh}^{orig}}\right)^{-1} = \frac{d_{clk}}{\text{wearout rate}}, \tag{4.1}$$

where d_{clk} is the targeted clock period, d_{fresh}^{new} and d_{fresh}^{orig} are the design time delays of the modified and delay-balanced pipeline stage, respectively, and d_{aged}^{orig} is the aged delay at t_{target} of the delay-balanced version. Using this estimation, the timing constraints for re-synthesis are set and the design is optimized accordingly. Afterwards, it is evaluated whether the new design matches the MTTF-balanced criteria or not. In the latter case, the design is tuned further using a fixed step size. By that means, it is possible to significantly reduce the number of iterations. For example, in case of FabScalar the number of iterations is reduced from 10 to 3, which corresponds to a runtime improvement of more than 3x.

Besides the number of iterations, also the runtime of a single iteration step is crucial for the overall runtime. To keep this as low as possible, the pipeline stages are not fully (re-)synthesized every time an optimization step is performed. Instead, an intermediate representation of the last version is stored in form of a `ddc`-file (Synopsys database format), which is then used for further optimizations. As the ddc-file contains already the gate-level design with all optimizations to match the previously used timing constraints, the initial synthesis from a behavioral to a gate-level description as well as basic optimization are avoided, which significantly reduces the runtime. Overall these optimizations can reduce the runtime of a single iteration step to less than 1 minute for a single pipeline stage, if the runtime for the aging estimation step is not considered. Hence, the runtime for one transformation step is less than 10 minutes for the entire FabScalar processor considering all 11 pipeline stages and even less than 5 minutes for OpenSPARC T1, as OpenSPARC T1 has only 6 pipeline stages.

	FabScalar [1]	OpenSPARC T1 [6]
Frequency	740 Mhz	1140 Mhz
Architecture	out-of-order	in-order
Pipeline Stages	11	6
Simultaneaous Multithreading (SMT)	no	4-way
Frontend-Width (per Thread)	4 insts/cycle	1 inst/cycle
Exec. Units (ALU/MUL/AGEN)	1/1/1	1/1/1

Table 4.1.: Architecture comparison of FabScalar and OpenSPARC

The runtime required by the aging estimation step depends on the flow to obtain the aging rates, as discussed in Section 3.4.1. If time consuming, but very accurate post-synthesis simulations are employed, the runtime for this step can easily exceed several hours (for simulating 10^6 cycles). However, if pre-synthesis simulations are used in combination with signal property propagation techniques, the runtime is in the order of seconds. As a result, the overall runtime of the entire transformation flow can be reduced to less than 15 minutes [3] for an complete out-of-order processor.

4.4. Experimental Results

To evaluate the concept of a MTTF-balanced pipeline design, the proposed design paradigm and the classical delay-balanced one are compared in this section using the FabScalar microprocessor and OpenSPARC T1. Employing these two microprocessors we can confirm that the proposed approach is applicable to a wide range of microprocessor designs, as these are representatives of "complementary" microprocessor families as shown in Table 4.1. The MTTF-balanced designs were generated using the flow presented in Section 4.3. To have a fair comparison in terms of energy and area, we used the optimization objective to get the best MTTF for a given clock frequency, which is used by both the MTTF-balanced and the delay-balanced design. Thereby, the clock period is given by the longest of all pipeline stage delays plus an additional safety margin of 10 % to avoid timing failures due to aging. The ambient processor temperature was set to 40 °C resulting in processor temperatures between 50 °C and 75 °C, which is reasonable for modern processors (see for example our experimental data in Figure 3.9).

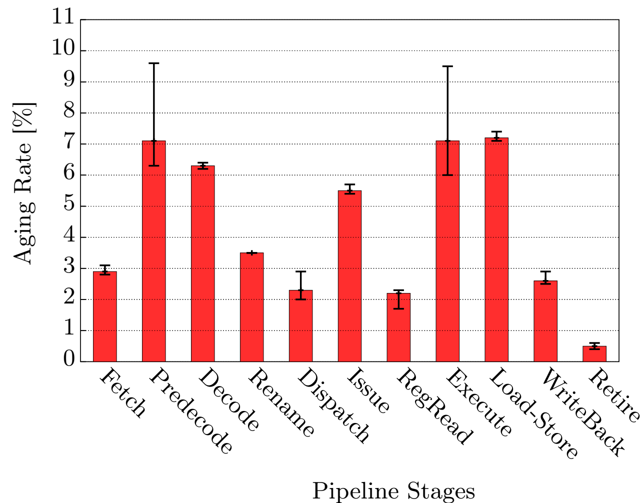


Figure 4.5.: Aging rates (after 3 years) of different pipeline stages of the FabScalar microprocessor for six SPEC2000 benchmarks (min, max, avg.)

[3]All runtime measurements in this chapter were performed on a system with AMD Opteron 6174 processors running at 2.2 GHz, 256 GByte RAM and Redhat Enterprise Linux 6.5 64 bit

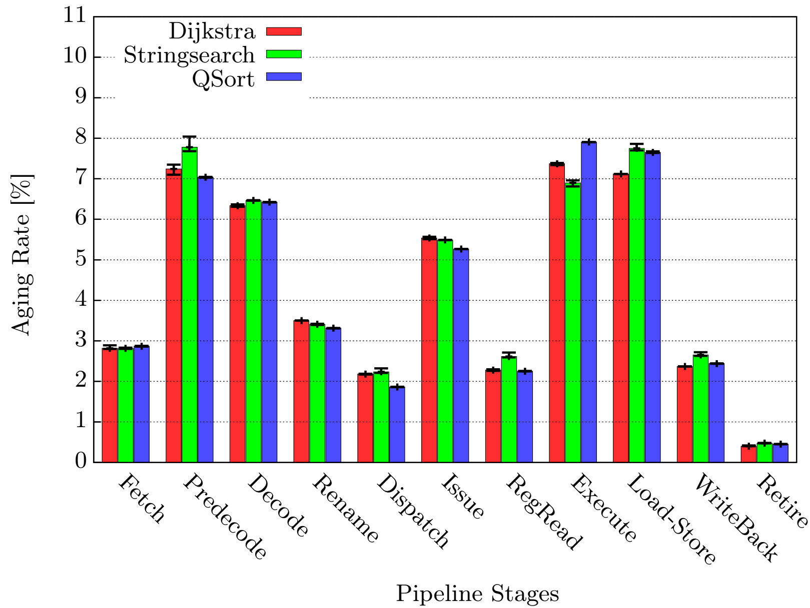


Figure 4.6.: Aging rates (after 3 years) of different pipeline stages of the FabScalar microprocessor for different datasets using MiBench applications (min, max, avg.)

As shown in Figure 4.5, the application choice for the aging estimation is crucial. Hence, the usage of not realistic, artificial workloads (e.g. applying random primary input combinations) can lead to an inaccurate aging estimation and thus an imbalanced design. Therefore, we evaluated real-world applications and tuned the designs according to these, i.e. we performed a cross-layer optimization. For this purpose, we employed in all of our experiments for FabScalar six different SPEC2000 benchmarks (bzip, gap, gzip, mcf, parser, vortex) provided with FabScalar, and simulated the processor behavior for 10^6 cycles after a warmup. For OpenSPARC T1 we used the regression test suite that comes with the simulation environment.

Table 4.2 summarizes the main results for FabScalar and Table 4.3 the ones for OpenSPARC. In these tables as well as in the rest of this section, for the sake of simplicity, we present only the worst-case delays and MTTFs for the different application tests as well as the average energy consumption over the used benchmarks.

Remark 4.4.1 (Dataset dependency of the aging rate)
Beside the application itself also the chosen input dataset can have an influence on the aging rates of different pipeline stages, as it determines the instruction operands. To analyze how strong the dataset impact is, we evaluated various MiBench[4] applications [198]. For all of them, the aging rates were almost independent of the chosen dataset as illustrated in Figure 4.6. An explanation for this independency is that the dataset mainly affects the instruction operands, while most of the instruction stream is just slightly altered. In addition, in each dataset there are "good" and "bad" operands in terms of aging, which leads to canceling effects. As a result, the dataset influence on the aging rates can be neglected, in particular compared to the impact of different workloads. Because of that, we evaluated only one dataset for each application.

[4]For this purpose we have chosen MiBench workloads as these are much smaller than SPEC applications, and thus can be simulated entirely. Hence, the impact of the dataset on operands and the instruction stream can be investigated

Stage	Delay-Balanced					MTTF-Balanced: Nominal V_{th}					MTTF-Balanced: V_{th}-Tuning					Changes
	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (1.48 ns) [years]	Energy [μJ]	Area [μm^2]	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (1.48 ns) [years]	Energy [μJ]	Area [μm^2]	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (1.48 ns) [years]	Energy [μJ]	Area [μm^2]	
Fetch	1.35	1.39	45	12.2	23262	1.38	1.46	7	12.0	25338	1.40	1.46	7	7.3	22459	GS, HVT
Predecode	1.35	1.48	3	16.6	35030	1.33	1.44	7	16.8	36482	1.33	1.44	7	16.8	36482	GS
Decode	1.34	1.43	16	6.1	24568	1.36	1.45	7	6.1	16697	1.42	1.46	7	2.8	17000	GS, HVT
Rename	1.33	1.38	50+	1.3	4049	1.33	1.38	50+	1.3	4049	1.43	1.46	7	0.7	3475	GS, HVT
Dispatch	1.32	1.37	20	0.1	1867	1.32	1.37	20	0.1	1867	1.41	1.46	7	0.1	1861	GS, HVT
Issue	1.35	1.43	18	14.3	30719	1.38	1.44	7	13.4	29157	1.38	1.44	7	13.4	29157	GS
RegRead	1.33	1.36	50+	2.1	12061	1.33	1.36	50+	2.1	12061	1.43	1.46	7	1.5	12074	GS, HVT
Execute	1.35	1.48	3	6.4	27529	1.33	1.45	7	6.5	28959	1.33	1.45	7	6.5	28959	GS
Load-Store	1.35	1.45	7	50.6	107664	1.35	1.45	7	50.6	107664	1.35	1.45	7	50.6	107664	none
WriteBack	1.32	1.36	30	4.0	3193	1.32	1.36	30	4.0	3193	1.43	1.46	7	2.5	3174	GS, HVT
Retire	1.35	1.36	50+	1.6	3201	1.36	1.37	50+	1.5	2579	1.45	1.46	7	1.0	2562	GS, HVT
Overall	1.35	1.48	3	115.4	273133	1.38 +2 %	1.46 -2 %	7 +233 %	114.5 -1 %	268406 -2 %	1.45 +7 %	1.46 -2 %	7 +233 %	103.4 -10 %	268746 -2 %	

Table 4.2.: Comparison of a delay-balanced and MTTF-balanced design for FabScalar in terms of design time delay, worst case MTTF, avg. energy consumption (without SRAM) and area (without SRAM) considering all benchmarks (GS = gate sizing, HVT = higher threshold voltage in the critical path)

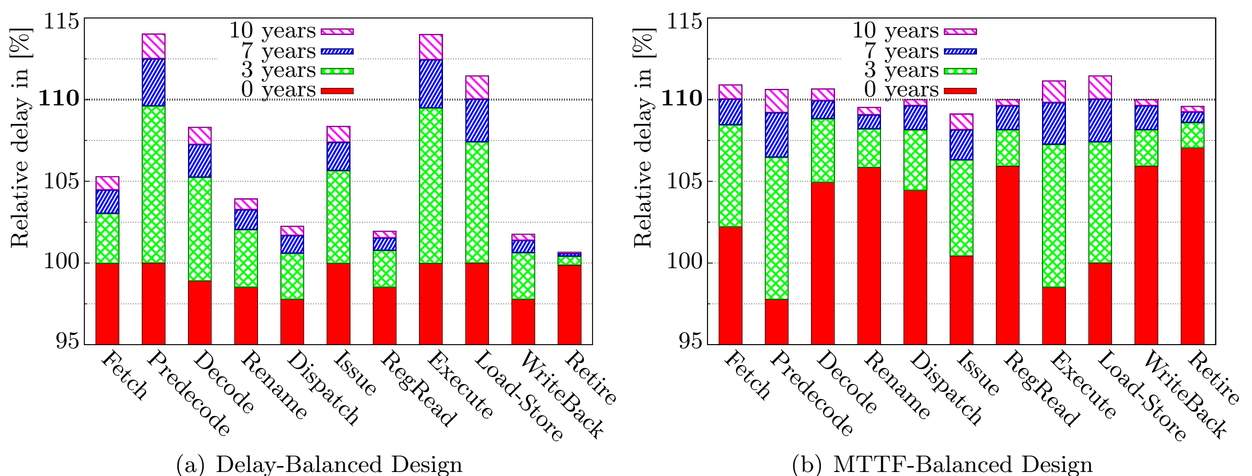
4.4.1. Optimization for FabScalar

The minimum clock period for FabScalar was 1.35 ns (= 740 MHz), limited by the Load-Store-Unit (LSU), which is the most complex unit of this microprocessor. Hence, given a margin of 10 %, the clock target was 1.48 ns. For these settings the standard delay-balanced design will fail after 3 years, as depicted in Figure 4.7(a). After 7 years the overall degradation reaches already 12.5 % and after 10 years the delay increase is around 14 %. In contrast, our proposed MTTF-balanced approach (see Figure 4.7(b)) is able to achieve a MTTF of 7 years (2.3x improvement). Therefore, the Predecode and Execute stage were designed faster (with less design time delay) using the synthesis optimization detailed in Section 4.3.2. All other stages were designed with less slack (i.e. slower). Therefore, a higher threshold voltage (20 % increase in V_{th}) in addition to the synthesis optimization techniques could be applied to all stages apart from the Issue stage, which could not match the timing constraints, if high-V_{th} transistors were used in the (near)-critical paths. By this means, the average energy consumption over all benchmarks (extracted with Synopsys PrimeTime) of the MTTF-balanced design is 10 % lower than the one of the traditional delay-balanced pipeline for a clock period of 1.48 ns. Furthermore, a higher V_{th} also reduces the aging rates, which can be used to achieve even higher energy savings. In addition, the area is reduced by 2 % if the MTTF-balanced version is employed (see Table 4.2).

As shown in Table 4.2, the area and energy savings are much smaller (1 % and 2 % for energy and area, respectively), if the threshold voltage is not increased. This is mainly due to the fact, that some of the pipeline stages cannot be designed any slower without using a higher V_{th}. The reason for this behavior is the academic nature of FabScalar due to which the original design is not efficiently balanced. However, no matter if V_{th}-tuning is used or not, the energy and area savings are not only positive side-effects. Instead, these are a result of the optimization process. As most pipeline stages are not aging-critical, energy consumption and area usage can be reduced for the majority of the pipeline stages resulting in overall energy and area savings.

Another information that can be inferred from Table 4.2 is the possibility to reduce the clock period of the MTTF-balanced design from 1.48 ns to 1.46 ns, while maintaining a lifetime of 3 years. As a result, the performance compared to the delay-balanced design will increase by more 2 %, and in addition, energy and area consumption will be lower compared to the delay-balanced version. This shows that the MTTF-balanced design paradigm can also be used to improve the performance, if an increased MTTF is of secondary interest.

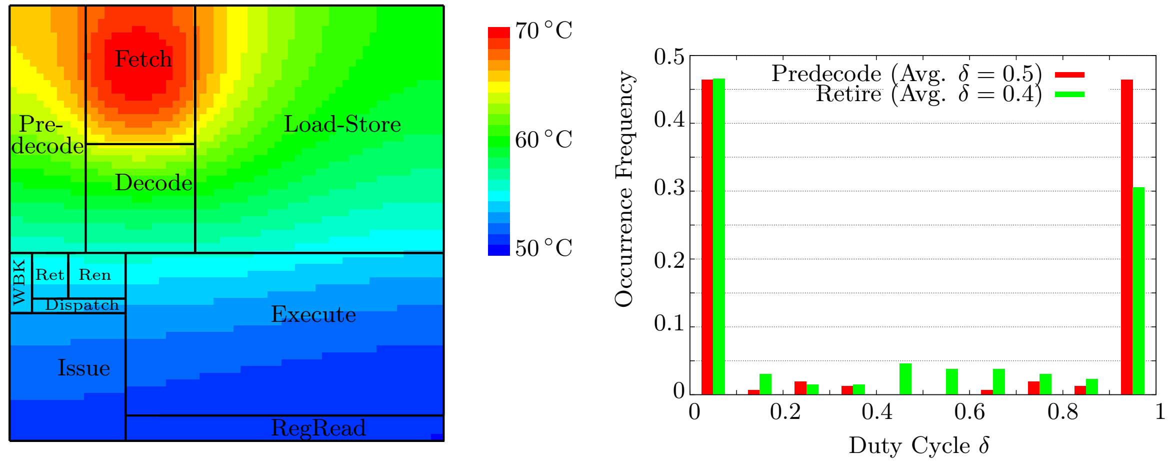
Moreover, Table 4.2 also shows that the critical path of a microprocessor (or in general a



(a) Delay-Balanced Design

(b) MTTF-Balanced Design

Figure 4.7.: Delay degradation of the delay-balanced design and MTTF-balanced design for the FabScalar microprocessor



(a) Simplified temperature distribution for the FabScalar microprocessor running the 181.mcf benchmark

(b) Duty cycle distribution for all transistors within the 100 most critical paths after 3 years for FabScalar's Predecode and Retire stage (higher duty cycle means faster wearout) for the 181.mcf benchmark

Figure 4.8.: Illustration of wearout affecting parameters (temperature, duty cycle)

circuit) changes over time. At the beginning, the LSU contains the most critical path. However, after less than 3 years the functional units become critical. This is a very important observation and underlines the necessity to consider all paths of a circuit for an accurate aging evaluation (as it is done in our RTL platform) and not only a small subset as discussed in Section 3.4.4.

Observations & Remarks

An interesting observation of our results is that the Predecode and the Execute stage have a very similar aging behavior, although their (microarchitecture-level) functionalities are totally different. The reason is that the delay degradation of both stages is very sensitive to the number of stall cycles that appear during the application execution (the higher the stall ratio, the faster these stages wear out), while other stages are less affected by these cycles. This can be explained as follows: During stall cycles the pipeline stage inputs remain constant, which means that also all internal signals are constant for a longer period of time. If many transistors inside the critical paths are under stress during these cycles, the delay degradation is accelerated. The worst results (degradation of almost 10 % in 3 years) were observed for the mcf benchmark, which had a stall ratio of 70 % for the pipeline front- and backend, while others, such as the parser benchmark, had a stall ration of only 10 % and caused a much slower delay degradation (less than 7.5 %). In contrast, a correlation between wearout rates and IPC (instructions per cycle) could not be observed.

Another interesting observation is the tremendous differences in terms of MTTF and delay degradation of different pipeline stages. As already mentioned earlier, these variations are due to the fact that the parameters influencing aging, i.e. temperature and usage (duty cycle and switching activity) are different for different pipeline stages. This circumstance is illustrated in Figure 4.8 for the FabScalar microprocessor and is also reported by various papers such as [146, 199]. As shown in Figure 4.8(b), the reason for the faster degradation of the Predecode stage is not only its higher temperature compared to many other stages (see Figure 4.8(a)), e.g. the Retire stage, but also the high duty cycle for many transistors in the most critical paths. Considering the 100 most critical paths after 3 years, the average duty cycle in these paths is roughly 0.5, while it is around 0.4 for the Retire stage. The difference in duty cycle and temperature is caused by three major factors: First, the gate-level implementation; second the microarchitecture design; and third the workload (i.e. input patterns) that is currently

executed by each stage [200]. Moreover, the degradation rate of a pipeline stage strongly depends on the amount of stress on the timing critical paths, while the behavior of all other paths is almost negligible. Hence, since the aging rate is affected by so many inter-related factors, it is a necessity to run detailed circuit-level simulations using real-world workloads to obtain accurate and reasonable results.

Another important point to note is the relation between delay and MTTF. The considerable MTTF improvements depicted in Table 4.2 come from the fact that the relation between runtime and delay degradation follows a root-like function. For example, in case of BTI the following relation can be used to estimate the delay degradation [129]:

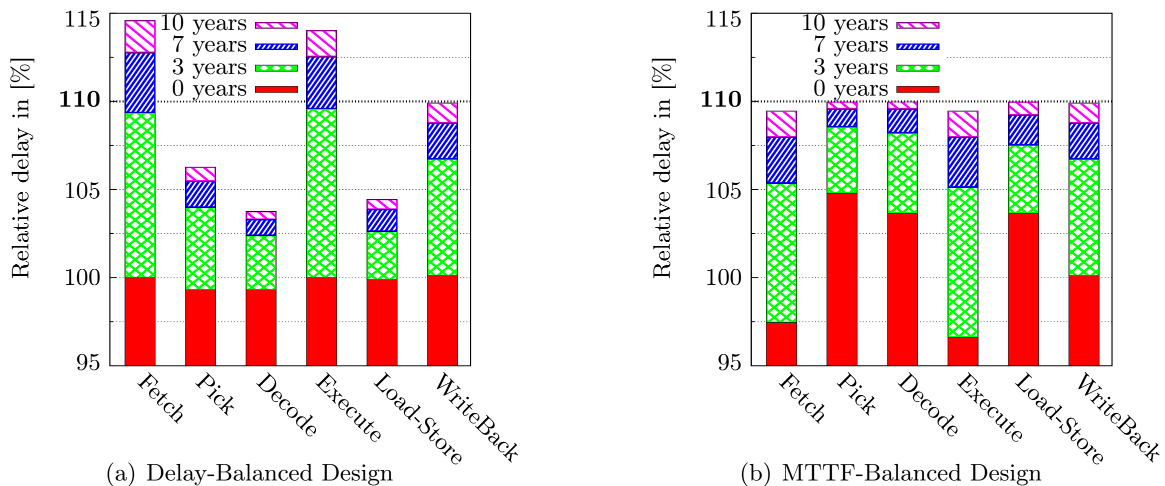
$$\frac{d(t) - d(0)}{d(0)} \sim \delta^n \cdot t^n, \tag{4.2}$$

where, d is the delay, δ the transistor's duty cycle, t the runtime and n is a technology constant equal to 0.25. Hence, reducing the design time delay ($d(0)$) from 1.35 ns to 1.33 ns, while the duty cycle is kept constant, improves the lifetime by approximately two times. In other words, small delay improvements lead to tremendous lifetime benefits. Please note that since the behavior for HCI is very similar, however with $n = 0.5$, the real results shown in Table 4.2 slightly differ from this estimation.

4.4.2. Optimization for OpenSPARC T1

OpenSPARC T1 is the open source clone of the industrial UltraSPARC T1 (*Niagara*) processor developed by Sun and released in 2005 [201]. Hence, the maximum clock frequency is much higher than for the academic FabScalar, i.e. we could operate OpenSPARC at 1140 MHz (clock period of 0.88 ns) using the TSMC 65 nm library limited by the WriteBack stage. Furthermore, the delays of the original design are much more balanced than those of FabScalar.

Using the standard delay-balanced design, OpenSPARC T1 can achieve a lifetime of 3 years for a timing margin of 10 %, while for a lifetime of 10 years already a guardband of 15 % is necessary as illustrated in Figure 4.9(a). In contrast, using our proposed MTTF-balanced design paradigm the MTTF can be extended from 3 years to 10 years (see Figure 4.9(b)), i.e. MTTF is improved by more than 3x (for 10 % timing margin). Therefore, the Fetch and Execute stages had to be designed with more design time slack (see Table 4.3), while all other stages were designed slower to save area and energy. However, without tuning the threshold voltage the savings for the remaining stages only compensate the energy and area costs due to



(a) Delay-Balanced Design

(b) MTTF-Balanced Design

Figure 4.9.: Delay degradation of the delay-balanced design and MTTF-balanced design for the OpenSPARC T1 microprocessor

Stage	Delay-Balanced					MTTF-Balanced										Changes
						Nominal V_{th}					V_{th}-Tuning					
	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (0.96 ns) [years]	Energy [μJ]	Area [μm^2]	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (0.96 ns) [years]	Energy [μJ]	Area [μm^2]	delay @t_{design} [ns]	delay @$t_{3\text{y}}$ [ns]	MTTF (0.96 ns) [years]	Energy [μJ]	Area [μm^2]	
Fetch	0.88	0.96	3	398.8	68547	0.85	0.92	10	399.0	68541	0.85	0.92	10	399.0	68541	GS
Pick	0.87	0.92	20	22.2	4139	0.89	0.94	10	22.0	4138	0.92	0.94	10	9.9	4204	GS, HVT
Decode	0.87	0.90	50+	7.4	1611	0.91	0.94	10	7.3	1610	0.91	0.94	10	3.3	1655	GS, HVT
Execute	0.88	0.96	3	289.0	56142	0.85	0.92	10	291.1	56353	0.85	0.92	10	291.1	56353	GS
Load-Store	0.88	0.90	50+	116.4	18307	0.91	0.93	10	116.3	18300	0.91	0.94	10	34.8	16381	GS, HVT
WriteBack	0.88	0.94	10	246.3	58987	0.88	0.94	10	246.3	58987	0.88	0.94	10	246.3	58987	none
Overall	0.88	0.96	3	1080.1	207733	0.91 +5 %	0.94 -2 %	10 +333 %	1081.9 +0 %	207929 +0 %	0.92 +5 %	0.94 -2 %	10 +333 %	984.4 -9 %	206121 -1 %	

Table 4.3.: Comparison of a delay-balanced and MTTF-balanced design for OpenSPARC T1 in terms of design time delay, worst case MTTF, avg. energy consumption (without SRAM) and area (without SRAM) considering all benchmarks (GS = gate sizing, HVT = higher threshold voltage in the critical path)

the faster version of the Fetch and Execute stage. With a higher threshold voltage the energy consumption can be reduced by 10 % and area by 1 %. This is especially obvious for the LSU, where energy can be reduced by almost 4x using a higher threshold voltage.

Similar to FabScalar, the clock frequency can be increased by 2 %, if MTTF is kept the same as for the delay-balanced design. Hence, also here the gained headroom can be used to boost the performance.

4.4.3. Comparison of FabScalar and OpenSPARC T1

As explained in the previous subsections, the lifetime for both FabScalar and OpenSPARC can be significantly extended using our proposed MTTF-balanced design paradigm, with better results for OpenSPARC. However, in general, it cannot be concluded that our technique is more efficient for simple, lightweight cores, since the two investigated processors as well as the used workloads are very different. This is also the reason why a direct comparison of the aging rates between FabScalar and OpenSPARC cannot be performed. Nevertheless a few conclusions can be drawn by the numbers presented in this dissertation. For both architectures, it is the execution stage which is aging-critical. Moreover, for both designs the delay degradation for the execution stage is very similar. This is due to the fact that similar ALUs were used and that the temperature as well as signal probabilities in the critical paths were in a similar range. Furthermore, we observed for both designs that the aging rate of a pipeline stage is not very sensitive to its design time delay. In other words, the aggressiveness with which a pipeline stage is designed seems to have only a small effect on its aging rate. In fact, the functionality, the workload and the temperature are far more important in terms of aging.

4.5. Comparison with Related Work

The proposed MTTF-balanced design paradigm is the first aging mitigation approach that addresses the entire instruction pipeline in a holistic way. In contrast, in previous techniques only small parts of the pipeline were addressed. Most of these solutions focused on the execution units of a microprocessor, since these are typically the lifetime-limiting factors [19, 146]. Various instruction scheduling techniques were evaluated in [76, 77] that aim at increasing the lifetime of the functional units, by balancing the incoming instructions among all available functional units. In addition, a novel aging-aware scheduling approach exploiting the timing criticality of different instructions to issue the most critical instructions to separate units is presented in Section 5 of this dissertation and summarized in [14]. In [145] the authors used an aging-aware NOP (no operation) instruction to alleviate the impact of NBTI on the ALU of an MIPS processor, which can be used for other execution units as well. Besides these techniques, in [11] it was proposed to periodically invert the instruction opcode to alleviate aging in the pipeline frontend. Also various techniques addressed wearout in memory elements, such as [153, 202] that used cell-flipping in order to make the duty-cycle close to 0.5. Another approach presented in [203] is intended to mitigate BTI-induced degradation in a register-file by flipping the leading bits of narrow-width values periodically. All of the aforementioned techniques are orthogonal to the proposed technique, and hence can be used in combination. This is also true for all other device-, circuit- and system-level techniques discussed in Section 2.4. For that purpose, it is only necessary to take the applied techniques during the aging estimation step into account to avoid an overestimation of the wearout rates (i.e. underestimation of MTTF) and thus imbalanced pipeline design.

While pipeline delay re-balancing is a novel idea for aging mitigation, similar techniques were already introduced to combat process variation and to reduce the power density. However, due to the different optimization targets, these techniques are typically not feasible for aging

mitigation. For example in [204] a cycle-time borrowing/stealing approach was proposed to re-balance the pipeline delay due to process variation, which was extended in [205] by added also variable supply voltage domains. The main idea of these schemes is that cycle time is "stolen" from fast stages and given to slow stages, so that the pipeline can operate at a clock period closer to the average stage delay. Potentially, this idea can be used similarly to our MTTF-balanced design paradigm. Stages that have high aging rates take some cycle time from stages with lower aging rates, to increase their MTTF. However, using these techniques, cycle time has to be redistributed, which is a complex task and not always possible. Instead, our design paradigm does not require a redistribution of cycle time, which makes our technique suitable for almost every design.

Another re-balancing technique using cycle-time borrowing was presented in [206], which is intended to balanced the power consumption of different pipeline stages. By that means, the problem that some pipeline stages consume much more energy than others is reduced. Potentially this can also help to avoid hotspots, which can slow down transistor aging. However, since the purpose is to minimize the overall power consumption, the timing slack for each pipeline stage is minimized after applying cycle-time borrowing to the pipeline. This slack reduction can negatively affect MTTF. In contrast, our technique tries to increase the timing slack of some stages to improve their MTTF and so MTTF of the entire processor.

Moreover, there are some techniques that applied the same principle of balancing the delay of various "components" at the end of the desired lifetime rather than at design time (in this thesis pipeline stages are balanced). In [63] this idea was extended to paths inside a circuit and in [207] the transistors inside gates were balanced according to the aging rates. Hence, since these two approaches target lower abstraction levels (device to circuit), they are orthogonal to our proposed scheme. Consequently, all three techniques can be combined to further optimize the overall microprocessor design as proposed in [23].

4.6. Summary and Conclusion

In this chapter, we presented a novel holistic cross-layer design paradigm for microprocessor instruction pipelines that takes the non-uniform delay degradation among all pipeline stages into account. For this purpose, circuit- to application-level information is exploited. As a result, the microprocessor lifetime can be significantly extended compared to state-of-the-art design approaches without reducing the performance (i.e. clock frequency), and at the same time power and area improve as well. Alternatively, the clock frequency can be increased considerably by reducing the necessary guardbands. Which of those two objectives is chosen in the end, depends on the application field. For instance, for a high-performance server a higher clock frequency might be more preferable than an extended lifetime, while for embedded systems with long mission times the first objective is favorable. In addition, it is also possible to select a combination of both objectives, i.e. slightly improved performance and lifetime.

CHAPTER
FIVE

AGING-AWARE INSTRUCTION SCHEDULING

As shown in the previous chapter, the execution stage is one of the most aging-critical stages in microprocessors and often limits the overall microprocessor lifetime. To alleviate this problem, a cross-layer instruction scheduling approach that combines knowledge from circuit-, microarchitecture- and application-level is presented in this chapter. To understand the necessity for this aging-aware instruction scheduling scheme, we first explain the shortcomings of state-of-the-art scheduling techniques. Afterwards, we introduce our novel scheduling methodology followed by various simulations that show the benefit of this aging-aware approach. Finally, we compare our technique with related work and provide some concluding remarks.

5.1. Problem Introduction and Motivation

A very important observation of the previous chapter is that the execution stage of the instruction pipeline can limit the overall microprocessor lifetime. This circumstance was also reported for the IVM processor [4] in [146]. Hence, to improve the microprocessor lifetime, it is of particular importance to address the execution units. Since wearout strongly depends on usage (gate bias, etc.) and temperature of the affected transistors, various works proposed enhanced instruction and application scheduling techniques to mitigate transistor aging [15, 77, 78] of execution units, cores or complete microprocessors. The goal of these techniques is to balance the necessary calculations on the available units (cores) to achieve equal wearout states on all units (cores), which should guarantee a longer lifetime of these parts. However, as this chapter shows, balancing the number of executed instructions (workload) over several functional units is not always the best aging mitigation strategy. This is mainly due to the fact that within the same clock cycle boundaries (timing boundaries), different instructions (workloads) executed by the same functional unit (core) have different timing criticality and consequently different time slacks[1]. For example, although all instructions executed in an ALU have an execution time of one cycle, there are some instructions whose delay is close to one cycle (e.g. arithmetic operations with long operands), while others need much less time (e.g. simple logic operations). Hence, the first group of instruction is called *timing-critical (TC)*, while the other one is named *non-timing-critical (NTC)*. In fact, this classification makes no difference from the performance point of view (always one cycle). However, in terms of aging, when the path delay of the circuit increases, the TC instructions start to fail already (i.e. have timing delays), while NTC instructions are still executed correctly. This key observation is illustrated in Figure 5.1 and is exploit by our proposed aging-aware scheduling.

To motivate this novel scheduling scheme, let us consider a microprocessor with two func-

[1]Slack is the time difference between the signal arrival time and the clock edge (see Figure 2.3)

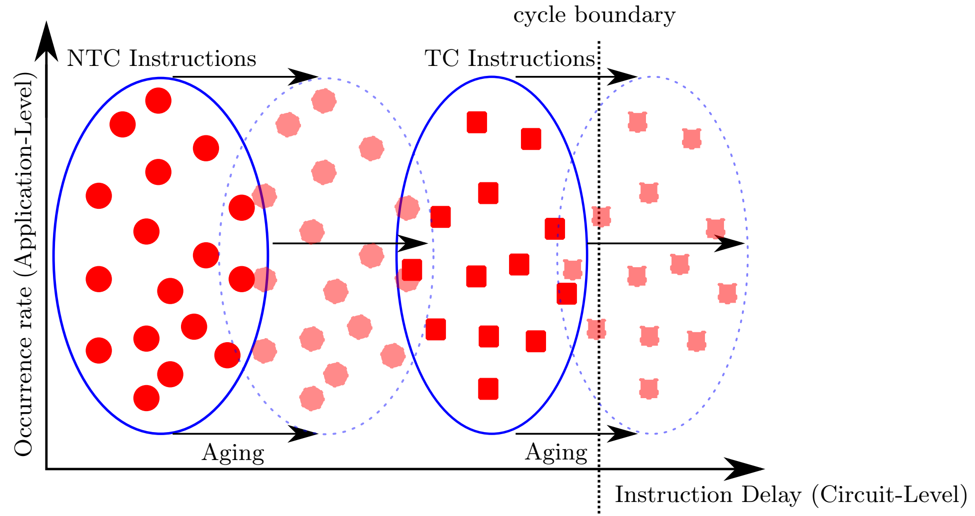


Figure 5.1.: Illustration of timing-critical (TC) and non-timing-critical (NTC) instructions

tional units of the same type (e.g. ALU) that uses a balanced instruction scheduling technique. Hence, both units will fail almost after the same time. Now imagine, that one unit is used only for the TC instructions and the other one only for the NTC instructions. Since the latter have a delay which is far less than the TC instructions, the first unit will always fail first, and thus determines the lifetime. However, as long as the fraction of TC instructions is less than 50 %, the critical unit will execute less instructions than in the balanced scenario. As a result, the idle ratio of this unit is increased. This can be used in combination with various architectural techniques such as power gating [64], clock gating [15], or input vector control (IVC) for NOP-instructions[2] [145] to alleviate the aging rates, and thus to increase the lifetime of this unit. As a result, the overall lifetime of both execution units increases, as the unit that executes the TC instructions is the limiting factor.

Based on this idea, *a novel cross-layer approach combining aging-aware instruction scheduling with specialized aging mitigation techniques* (e.g. IVC and power gating) to alleviate the impact of wearout on the execution units is presented in this chapter. The aging-aware instruction scheduling technique uses circuit-, architecture-, and software-level information to maximize the lifetime of the systems according to given power constraints. In the proposed microarchitecture, the instructions are categorized depending on their worst-case delays (circuit-level information) and occurrence rates (application-level information) into classes of critical and non-critical instructions. In other words, the aging-aware scheduling policy takes the "per-instruction" slack into account instead of the overall slack of a functional unit over all possible instructions, which is the traditional approach. The resulting classification is exploited by the instruction scheduler in such a way that each of these classes uses its own (specialized) functional unit(s). By that means, it is possible to increase the idle ratio of the units executing the critical instruction, which can be used to considerably extend the lifetime of the functional units by applying special mitigation techniques.

5.2. Aging-aware Instruction Scheduling Methodology

In order to efficiently slow down wearout, it is mandatory to reduce temperature and usage, which are two of the key aspects that influence transistor aging. In a functional unit like an ALU, these two parameters strongly depend on the sequence of executed instructions. Es-

[2] A NOP-instruction is a no-operation instruction, i.e. an instruction that effectively does nothing at all

pecially the idle periods between two instructions can be used to mitigate the aging impact, for example by applying power gating, clock gating or input vector control (IVC) for NOP-operations [15, 64, 145]. However, these techniques can only be used efficiently, if the idle periods between two instructions are very long, which is often not the case. The envisioned technique aims at increasing the idle periods between two instructions using an enhanced aging-aware instruction scheduler. By that means, the benefit that can be taken from power gating, clock gating or IVC can be increased. In the following this idea is introduced using an exemplary ALU. However, the idea is also applicable to other functional units.

5.2.1. Instruction Classification

The first step to make the envisioned aging-aware instruction scheduling technique possible, is to characterize all instructions based on their timing criticality. In this regard, by investigating the delay of various paths in a functional unit such as an ALU in detail, it is observed that different instructions have different execution times (path delays). Some instructions need just a small fraction of a clock cycle to complete, while others need almost the entire clock cycle. The real execution time (i.e. delay) thereby strongly depends on the instruction and the applied operands. This is illustrated in Figure 5.2, which shows the worst-case delay and occurrence rate (see Table 5.1) for each instruction supported by the IVM [4] ALU, which was synthesized using the Synopsys Design Compiler and the SAED 90 nm library.

From the timing perspective, the delay of a functional unit is determined by its longest path delay (slowest instruction) which is obtained from a static timing analysis. Once the clock period is set accordingly, from the performance perspective at microarchitecture-level all instructions are considered the same. However, due to aging effects, the delays of various paths in the circuit increase, which eventually leads to the problem that for some instructions the (intermediate) result will not be computed within the given timing boundaries, i.e. one clock cycle. In this regard, the instructions with a very small timing slack are the first ones that start to fail. This means that the *timing-critical (TC) instructions* (the ones with a real delay close to one cycle) determine the lifetime of the ALU. Moreover, the critical instructions (here it is ADDQ, i.e. add for 64 bit operands) occur much less frequent than the non-critical instructions. Using the ExtraTime platform (see Section 3.2), we observed that only 18 % of all instructions of various SPEC2000 benchmarks (compiled with GCC 4.3 and *-O3*-optimization) that are executed in the ALU(s) belong to the critical category, while 82 % are *non-timing-critical (NTC) instructions*, as detailed in Table 5.1.

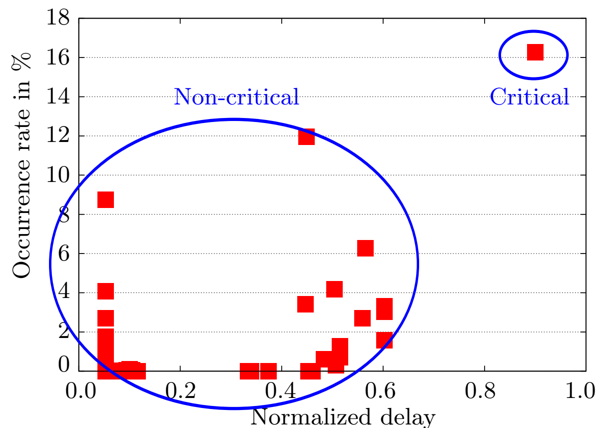


Figure 5.2.: Worst-case delay and occurrence rate of all instructions for the IVM [4] ALU

Workload	Instructions	NTC [%]	TC [%]
applu	291,012,195	73	27
bzip2	443,681,261	74	26
equake	219,397,379	85	15
gcc	440,405,407	86	14
gzip	505,045,344	82	18
lucas	345,330,553	80	20
mcf	385,659,378	88	12
mesa	356,736,706	88	12
mgrid	157,231,925	93	7
parser	363,009,269	89	11
swim	435,235,376	58	42
twolf	384,701,811	84	16
wupwise	364,318,715	90	10
average	**360,905,025**	**82.3**	**17.7**

Table 5.1.: Workloads and their instruction ratios (execution time = 0.5 seconds) using a single ALU

Based on this observation, which basically tells different instructions have different *aging-criticality*, the idea for the aging-aware scheduling is to use dedicated unit(s) for each of the two instruction classes, instead of executing both classes in the same unit(s).

Please note that for another ALU or functional unit with a different set of instructions and/or different implementation, the timing distribution of various instructions and their occurrence rate will change from what is shown in Figure 5.2, but the overall classification will still be valid. For instance, logic operations will have less delay than arithmetic operations (with wider word sizes and more sophisticated bit operations).

5.2.2. Aging-Aware Scheduling

Normally, the aforementioned information about the instruction delay is not used by the instruction scheduler (or the unit which decides which functional unit will be used). Instead, if more than one ALU is available, the scheduler will balance the workload, so that all ALUs will execute roughly the same amount of instructions, which is mainly due to performance reasons. Also some previous work on aging-aware instruction scheduling [15, 77], that consider the delay of all instructions to be the same (i.e. one clock cycle), show that a balanced technique is better than unbalanced ones in terms of MTTF. However, as discussed before, if the accurate timing of instructions is taken into account, a balanced scheduling is not the best choice from aging perspective. Hence, in the proposed microarchitecture each instruction class has its own dedicated ALU(s), i.e. ALU_{TC}(s) only for the TC instructions and ALU_{NTC}(s) only for the NTC ones, although this can lead to an unbalanced load among all available ALUs. Based on the data presented in Table 5.1 this means that all available ALU_{TC}s execute only 18 % of all instructions, while the other ALU(s) handle the remaining 82 %. Consequently, the TC-ALUs

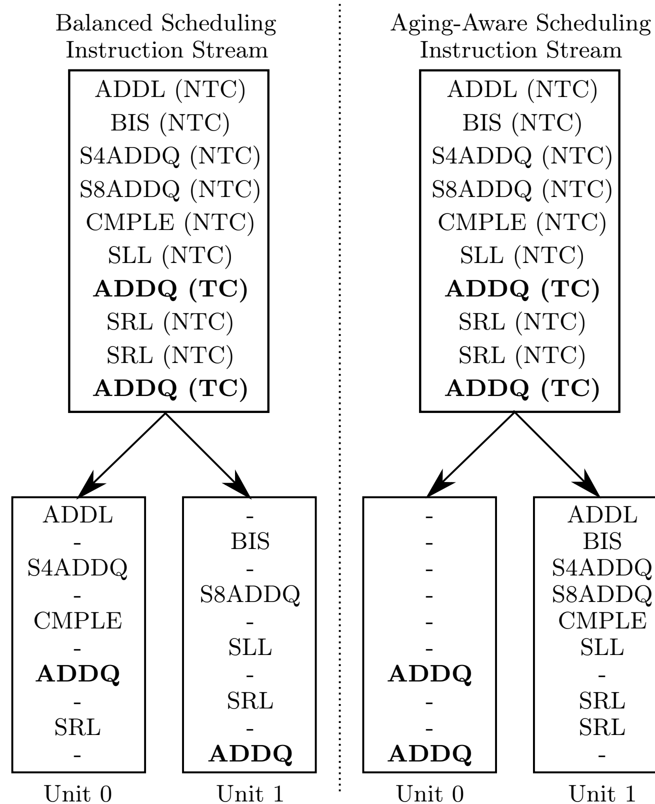


Figure 5.3.: Assignment of instructions to functional units for balanced and aging-aware scheduling policies

are idle most of the time, which is also shown in Figure 5.3 for an example with one ALU_{TC} and one ALU_{NTC}. Hence, these units can be power gated very efficiently, and also IVC can be applied most of the time. As a result, the lifetime of the TC-ALUs is significantly better compared a balanced scheduling approach (see Section 5.3 for more details). Moreover, the "critical ALU(s)" are the ones which determine the overall lifetime, since the non-critical instructions have a delay which is much smaller than the delay of the critical instructions and even with aging included never surpasses the delay of the critical ones, before these will start to fail. Thus, this "unbalanced" scheduling approach can considerable improve the overall lifetime.

Of course, the unbalanced load can lead to a performance loss compared to an architecture with only "full" functional units of the same type (see example in Figure 5.3). However, the specialized TC units can be much smaller than standard units (only 8 % of the standard area as explained in Section 5.3). Hence, instead of splitting a certain number of "full" units into a group of TC-ALUs and another group of NTC-units, it is possible to convert all "full" functional units to NTC-units and add additional TC-ALUs with small overall costs. As a result, the performance even increases and lifetime will benefit as well, as shown in Section 5.3. In addition, smaller units will consume less power and by that means stay cooler which will help to further slow down wearout.

Beside the potential performance penalty, also the implementation costs for the enhanced instruction scheduler need to be considered for the proposed technique. However, as the scheduler (or the unit which decides which functional unit will be used) selects the appropriate unit for the actual instruction based on the instruction opcode, no extra bits to encode the classification are necessary. For example, in our evaluated scenario the critical ADDQ instruction has the opcode 1020h. Whenever this opcode is detected, the corresponding instruction (ADDQ) is sent to an ALU_{TC}, while all other instruction will be executed by an ALU_{NTC}. Hence, there are only additional costs for the instruction scheduler, if the total number of execution units increases, since this raises the scheduler complexity.

5.2.3. Optimal Cut-off Line between TC and NTC Instructions

In order to improve the overall lifetime of the execution units, while maintaining a high performance, it is crucial to find the optimal cut-off line between TC and NTC instructions, i.e. to determine which instruction belongs to which group. This is a very complex optimization problem as illustrated in Figure 5.4 for the simple case in which there is one functional unit for the TC instructions (ALU_{TC}) and another one for the NTC instructions (ALU_{NTC}). If the cut-off line is shifted to the right, more instructions are considered as NTC. Hence, the ALU_{NTC} MTTF will decrease (more instructions lead to a lower idle ratio which lowers the MTTF; in

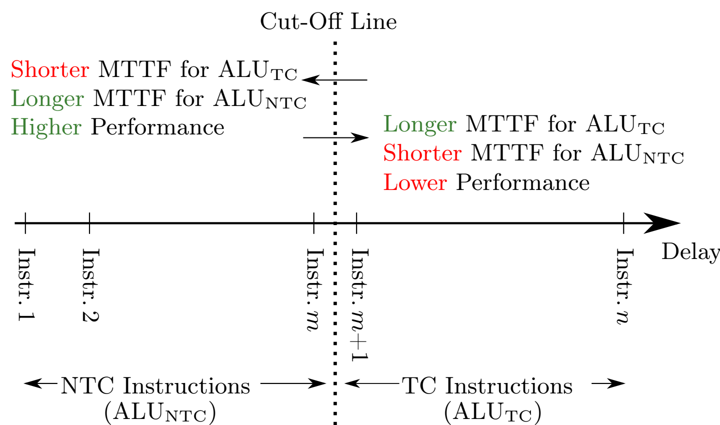


Figure 5.4.: Classification of instructions into TC and NTC and its impact on MTTF and performance

0. List all instructions according to their worst-case delay
   / $delay_{Instruction\ 1} \leq \cdots \leq delay_{Instruction\ n}$ */*

   $i = 0$
1. TC = {Instruction n}
   / Perform simulations with this setting and estimate MTTF */*
2. $\text{MTTF}_0 = \min(\text{MTTF}_{\text{ALU}_{\text{TC}}}, \text{MTTF}_{\text{ALU}_{\text{NTC}}})$

Do

   $i = i + 1$
   3. TC = TC $\cup$ {Instruction $n - i$}
      / Perform simulations with this setting and estimate MTTF */*
   4. $\text{MTTF}_i = \min(\text{MTTF}_{\text{ALU}_{\text{TC}}}, \text{MTTF}_{\text{ALU}_{\text{NTC}}})$

While $(\text{MTTF}_i > \text{MTTF}_{i-1})$

   / Revert last TC adjustment, as it reduced MTTF */*
5. TC = TC $\setminus$ {Instruction $n - i$}
6. **Done.**

Algorithm 5.1: Flow to obtain instruction groups such that the overall MTTF is maximized

addition the worst-case delay increases and hence the MTTF decreases). However, at the same time the ALU_{TC} MTTF will improve (less instructions lead to a higher idle ratio and hence higher MTTF). If the cut-off line is moved to the left, the ALU_{NTC} MTTF will increase (less instructions, lower worst-case delay), but the ALU_{TC} MTTF will decrease (more instructions). In addition, shifting the cut-off line also affects the performance. Fewer TC instructions typically lead to a decreased performance, as the load between all available units becomes more unbalanced. Instead, moving the cut-off line to the left increases the performance until 50 % of all instructions are classified as TC and NTC, respectively. Also the overall area consumed by the ALUs depends on the cut-off line. The fewer instructions are supported by the ALU_{TC}, the smaller this unit can be. Hence, placing the cut-off line is a complex optimization problem and the solution will be a trade-off between performance, lifetime, area and power.

Our proposed algorithm, to find the best solution in terms of MTTF is described in Algorithm 5.1. First, all instructions are sorted based on their worst-case delays. Next, only the instruction with the highest worst-case delay (instruction n) is considered as TC instruction (Step 1). Then, simulations are processed (e.g. with the ExtraTime platform) and the MTTF of all ALU_{NTC}(s) and all ALU_{TC}(s) (if there are several) is evaluated (Step 2). Afterwards, the set of TC instructions is expanded by instruction $n - 1$ (Step 3) and again the MTTF of both units is evaluated (Step 4). This process is repeated until the best MTTF considering all units is found. In this regard, it is important to note that as soon as the ALU_{TC} determines the overall lifetime, the search process is stopped. This is due to the fact that by adding more instructions to the TC class, the MTTF of this unit (or units) will become even shorter.

In our case the algorithm yielded the result depicted in Figure 5.2. However, as the cut-off line is a function of the number of TC and NTC units, different configurations can have different cut-off lines.

5.2.4. Further Extensions

Beside for a simple ALU, the proposed scheduling technique is also applicable to much more sophisticated functional units such as floating point units or vector execution units. Also these units will feature some instructions (typically multi-cycle instructions) that have a slack close to zero in one of their execution cycles.

Beside functional units, the concept of this scheduling technique can be also employed to schedule applications to difference cores of a multicore processor. In this regard, the timing criticality of applications, i.e. the deadline until the execution has to be finished, replaces the

timing criticality of instructions. Hence, applications with very tight deadlines will be executed on a particular core, while all the others use the remaining cores. To alleviate aging of the "critical" core, voltage and frequency scaling can be applied.

Moreover, the concept can be also extended to handle process variation, as well. If two functional units of the same type are available, one can be faster than the other one due to process variation, i.e. the timing slack of an instruction executed in the faster unit is larger than in the slower unit. In that case, the TC instructions should be scheduled to the faster unit, while the NTC instructions should use the slower unit. As a result, lifetime (due to larger slack for TC instructions) or performance (smaller overall delay, as clock period can be tightened) can be improved.

5.3. Experimental Results

In this section the implications of the envisioned aging-aware scheduling technique on the system performance, power consumption, area and in particular lifetime are presented. Therefore, in the first part a circuit-level analysis is conducted to evaluate power and area. In the second part, performance and reliability are studied using the architectural ExtraTime framework presented in Chapter 3.2. As case studies for the aging mitigation techniques that are employed for the functional units during idle periods input vector control (IVC) and power gating (PG) are used.

For all studies the IVM ALU was employed and NBTI was considered as primary aging phenomenon. However, also PBTI can be alleviated with the proposed approach, as it is very similar to NBTI. Because of that, also the results would be very similar, if PBTI is included in the analysis. Also HCI is addressed by the proposed scheduling technique, since it reduces the number of executed instruction in the critical unit which means also a lower switching activity. Given the results in Table 5.1, the number of executed instructions compared to a balanced scheduling approach is reduced by 2.8x, on average (if in both cases two functional units are used). As a result, based on the aging models from ExtraTime, the lifetime will improve by 2.8x, on average, if only HCI is considered. Instead, for BTI-induced wearout, the improvement is only 1.6x. Hence, BTI is more critical, and thus HCI is not considered in this section.

Further details on the entire processor configuration, technology parameters and the simulated workloads (SPEC2000 benchmarks) can be found in Table 5.2. Overall, eight different ALU configurations have been investigated to evaluate the scalability of the proposed technique, ranging from a low power solution (e.g. MIPS [24]) with only a single "full" ALU ("full" means that TC and NTC instructions are executed), to a performance version with up to four "full" ALUs (e.g. as in Intel's recent Core-i-processors [208]).

Processor	Single-core @ 3 GHz, out-of-order, 4-issue
L1-Cache	64 KByte, 3 cyc latency
L2-Cache	2 MByte, 15 cyc latency
Execution Units	1x-4x ALU, 2x MUL, 2x FPU
Expected wearout	10 % in 3 years, i.e MTTF = 3 years (90 °C)
Conditions	T_{start} =57 °C, V_{dd} = 1.0 V, V_{th} = 0.21 V
SPEC2000 benchmarks	applu, bzip2, equake, gcc, gzip, lucas, mcf, mesa, mgrid, parser, swim, twolf, wupwise
Runtime	0.5 seconds excluding initialization

Table 5.2.: Configuration details for the experiments

Configuration	1/0/0	2/0/0	0/1/1	3/0/0	0/2/1	4/0/0	0/3/1	0/2/2
Avg. Power [μW]	148 100 %	296 200 %	160 108 %	444 300 %	307 207 %	592 400 %	456 308 %	320 216 %
Area [μm^2]	45895 100 %	91790 200 %	49546 108 %	137685 300 %	95411 208 %	183580 400 %	141336 308 %	99092 216 %

Table 5.3.: Different ALU configurations ("full"/NTC/TC) and their impact on area and power

5.3.1. Power and Area Evaluation

As mentioned in Section 5.2.1, the timing critical instructions of the IVM ALU encompass only the ADDQ-instruction (ADD instruction for three 64 bit operands a+b = c). Hence, the specialized critical-ALUs need only to support the ADDQ-instruction. Therefore, the feature set can be heavily reduced for these specialized ALUs, resulting in a very small size. Using Synopsys Design Compiler and the SAED 90 nm library the size for such a specialized ALU is obtained to be only 8 % of a normal, general purpose ALU (3651 μm^2 vs. 45895 μm^2). Furthermore, the power consumption of these two configurations using Synopys PrimeTime was investigated. For this purpose, the instruction sequences for each ALU were extracted using ExtraTime and various SPEC2000 workloads. With ModelSim[3] the switching activity behavior for each benchmark and each ALU was extracted and afterwards PrimeTime was used to calculate the power consumption of each ALU for each application. On average, the power consumption of a specialized ALU_{TC} is only 8 % of the power consumption of a full ALU (12 μW vs. 148 μW), as leakage power is dominant. As a result, the overhead in terms of power and area for placing additional TC-ALUs is very small, as shown in Table 5.3. If the area and power consumption of the entire processor are taken into account, the overheads become even smaller and are (almost) negligible.

5.3.2. Performance and Lifetime Evaluation

Aging-Aware Scheduling & Input Vector Control

Usually input vector control (IVC) is used at gate-level to mitigate the influence of (N)BTI and/or leakage. However, IVC can be also used at microarchitecture-level. At that level, input vectors are applied at the primary inputs of the entire (micro)-architectural blocks, such as functional units. For an ALU, the input vector consists of an instruction opcode and two instruction operands. Since, the aging-aware input vector should not affect the program execution running on the processor, it has to be a No-Operation (NOP) [24]. In the chosen superscalar out-of-order processor the aging-aware input vector can be applied at the ALU inputs every cycle the ALU is idle, i.e. not executing an instruction. Since it takes some time, until all internal gates of the functional unit are in the state resulting in the minimal NBTI-induced wearout, the first cycle in every idle period is considered as a normal operational cycle with higher degradation rate.

To identify the NBTI-induced aging rates of the ALU, when the aging-aware input vector is applied, the following steps were performed: First, the gate-level description of the IVM ALU was extracted using Synopsys Design Compiler and the SAED 90 nm library. Then, a linear programming solver was used to extract the input vector resulting in the minimum NBTI-induced degradation similar to [145]. Finally, the wearout rate for this NBTI-aware input vector was calculated and used in the aging model of ExtraTime for the representative transistor. The extracted input vector resulting in the minimum NBTI-induced degradation leads to a relative delay increase of 6.4 % in 3 years.

The proposed aging-aware scheduling can significantly improve the efficiency of input vector

[3]Logic simulator from Mentor Graphics

Application	1 ALU	2 ALUs		3 ALUs		4 ALUs		
		Aging-Aware	Balanced	Aging-Aware	Balanced	Aging-Aware		Balanced
	1/0/0	0/1/1	2/0/0	0/2/1	3/0/0	0/3/1	0/2/2	4/0/0
applu	4.34	8.46	5.83	8.61	6.61	8.65	9.98	7.30
bzip2	3.63	7.71	5.73	8.27	6.64	8.36	10.68	7.38
equake	4.86	8.39	5.90	8.13	6.51	8.17	8.57	6.99
gcc	3.79	10.10	5.41	10.10	6.30	10.46	12.26	7.09
gzip	3.30	8.67	4.97	9.67	5.84	9.71	11.72	6.59
lucas	4.79	10.18	6.26	10.48	7.36	10.59	11.95	8.21
mcf	6.28	11.23	7.66	10.56	8.40	10.82	11.23	9.00
mesa	3.85	11.24	5.92	11.90	6.88	12.17	13.97	7.70
mgrid	4.54	9.09	5.67	9.14	6.42	9.16	9.36	6.95
parser	4.81	12.05	6.53	11.59	7.61	11.90	12.86	8.44
swim	7.23	9.96	8.44	9.82	9.39	9.88	10.58	9.78
twolf	4.73	10.65	6.21	10.26	7.20	10.49	11.89	8.08
wupwise	3.43	10.60	5.14	11.54	5.94	11.84	13.29	6.61
Average	**4.58**	**9.87**	**6.13**	**10.01**	**7.01**	**10.17**	**11.41**	**7.70**

Table 5.4.: MTTF in years for several SPEC2000 benchmarks for the standard (balanced) scheduling and the proposed aging-aware scheduling using an aging-aware IVC for different ALU configurations (“full”/NTC/TC)

Configuration		Performance Avg. IPC	MTTF [years]	Area-Cost μm^2	
1/0/0	(balanced)	1.21 (8)	4.58 (8)	45895	(1)
0/1/1	(aging-aware)	1.25 (7)	9.87 (4)	49546	(2)
2/0/0	(balanced)	1.39 (6)	6.13 (7)	91790	(3)
0/2/1	(aging-aware)	1.40 (5)	10.01 (3)	95411	(4)
3/0/0	(balanced)	1.42 (3)	7.01 (6)	137685	(6)
0/3/1	(aging-aware)	1.42 (2)	10.17 (2)	141336	(7)
0/2/2	(aging-aware)	1.40 (4)	11.41 (1)	99092	(5)
4/0/0	(balanced)	1.42 (1)	7.70 (5)	183580	(8)

Table 5.5.: Summary for the standard (balanced) scheduling and the proposed aging-aware scheduling using an aging-aware IVC for different ALU configurations (“full”/NTC/TC) in terms of performance, MTTF and costs; Values in () represent the ranking

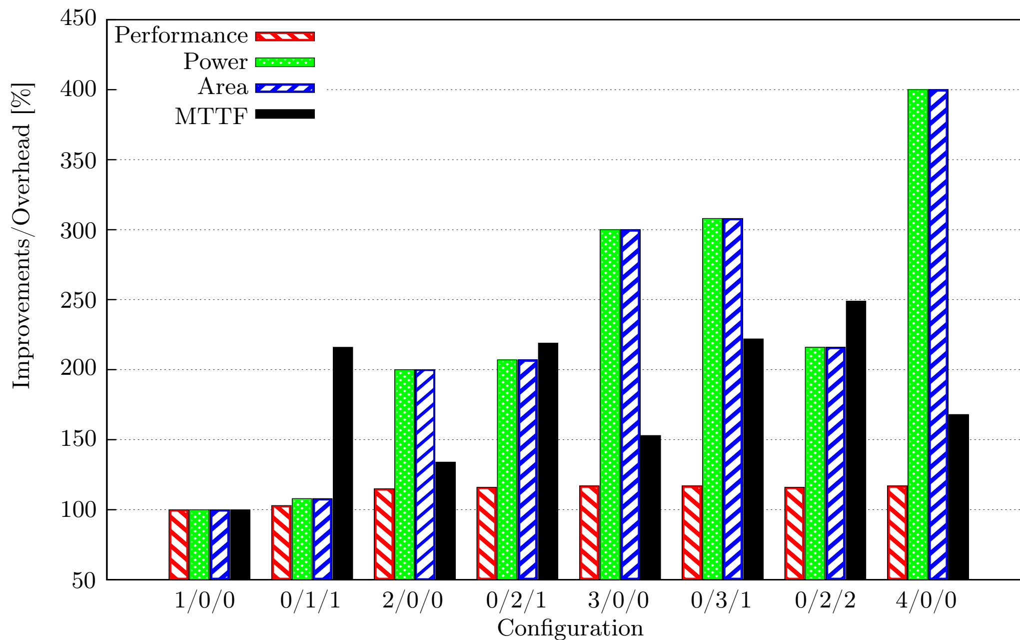


Figure 5.5.: Comparison of different ALU configurations (“full”/NTC/TC) using aging-aware IVC in terms of performance, MTTF, area and power consumption

control, as it increases the idle time of the timing critical units and hence MTTF, as illustrated in Table 5.4. Obviously, the technique is best, for low-power, embedded processors with a small number of units, as in these cases the idle time gains are best. If the total number of ALUs is kept constant, the proposed technique can increase the lifetime by at least 1.3x, in best case (i.e. for a small number of units) even by more than 2.1x. Furthermore, as shown in Table 5.5 and Figure 5.5, finding the best ALU configuration is a trade-off between performance, lifetime and cost. However, for all cases, the proposed aging-aware scheduling in combination with specialized functional units is superior to the traditional balanced scheduling in terms of lifetime, area and power costs. However, the unbalanced workload distribution leads to a performance penalty. This can be tackled by converting all "full" ALUs to NTC units and adding a few specialized TC ALUs. By that means, the performance can be even improved with only small area and power costs (see Table 5.5).

Please note that the scaling behavior of the computing performance, measured in IPC (instruction per cycle) depends not only on the number and type of the functional units, but also on the memory latencies as well as the width of the instruction pipeline frontend, i.e. how many instruction can be fetched and decoded in parallel. As a 4-wide frontend is used, the performance saturates soon by increasing the number of units.

Dynamic Input Vector Control for Different Operation Modes

A drawback of the input vector control approach is that it also affects power consumption, and the effect on power is not in the same direction as on lifetime. Hence, using the best input vector in terms of MTTF can increase the power consumption. In the studied case, the leakage power increased by 2 % using the aging-aware input vector, compared to a leakage-optimized input vector. Hence, depending on the criticality of various runtime constraints, such as power and lifetime, the user, the OS or a firmware can select different operation modes to either optimize for lifetime, or for power saving.

The impact of different input vectors on aging is depicted in Figure 5.6. As one can see, the aging-aware input vector (grey background) yields superior lifetime results for all ALU configuration. However, with increasing average duty cycle (ratio of stress to total time), the aging rates increase, and thus the lifetime improvements reduce. Moreover, the figure shows

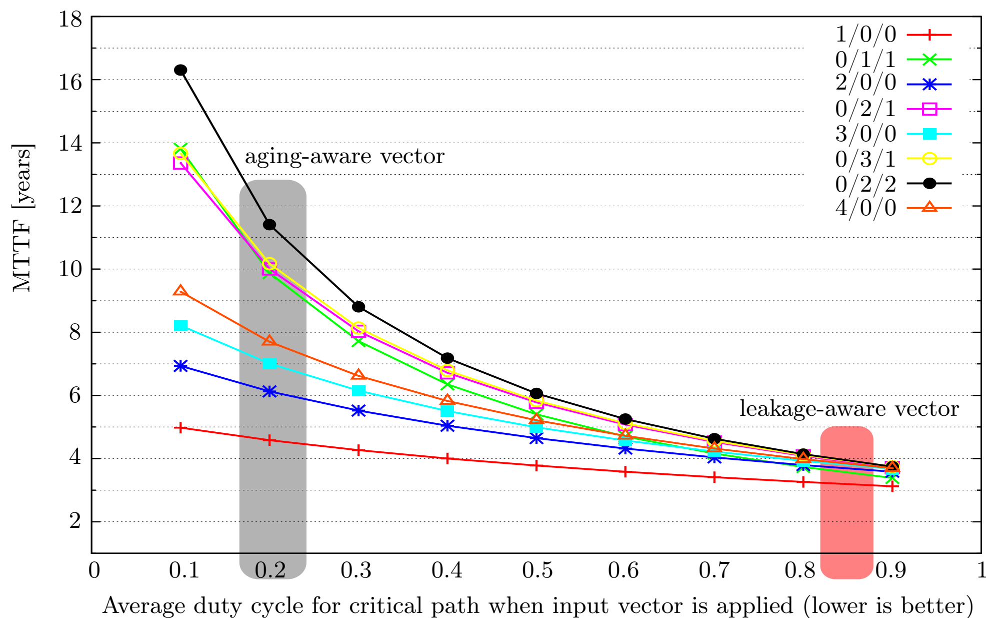


Figure 5.6.: MTTF for different input vectors and different ALU configurations ("full"/NTC/TC)

also that MTTF scaling, and hence the efficiency of input vector control strongly depend on the ALU configuration. If the idle ratio is small (i.e. if few units are available), the MTTF improvements for better input vectors are significantly smaller compared to cases where the idle ratio is higher (less than 2x vs. more than 4x).

As nowadays systems often support multiple operation modes such as a low power mode using a low supply voltage to reduce power consumption, or a high performance mode using a high frequency and supply voltage to achieve the best performance, two different input vectors were extracted. First, an aging-aware input vector that is applied whenever the high performance mode is used, and a leakage-aware vector which is applied during low power operation periods. The aging-aware input vector is the same as the one discussed in the previous subsection, i.e. it leads to an average duty cycle of 0.18 considering the most critical path. In contrast, the leakage-aware input vector results in an average duty cycle of 0.85 (red zone in Figure 5.6), and hence much higher stress time. Therefore, the leakage power is reduced by 2 %. Although the leakage-aware vector results in higher stress than the aging-aware vector, is does not considerably affect lifetime. This is due to the fact, that in the low power mode, frequency and supply voltage are significantly reduced, and hence transistor aging during these phases is not a limiting factor. However, if this vector is used during the high performance mode, when a high supply voltage is applied, MTTF would be 1.4x to 2.7x worse than with the aging-aware input vector.

The results also underline, that input vector control is much more efficient for aging mitigation than for power reduction. This is due to the fact that the power consumption depends on the status of all gates inside the circuit (i.e. additive), which is hard to control by applying a single input vector. In contrast, the worst-case delay degradation depends only on a few paths and the status of the gates inside these paths (i.e. selective).

Aging-Aware Scheduling & Power Gating

As mentioned before, beside IVC, also power gating can be used to mitigate NBTI-induced transistor aging. The power gating technique chosen for this evaluation is relatively simple. After an idle period of at least 200 cycles a functional unit can be power gated. The following power down time until the ALU is completely power gated is set to $3\,ns$ [209, 210]. Until no new instruction is incoming, the unit remains power gated (see Figure 5.7). When a new incoming instruction is detected, the unit is woken up. The wake up time is set to $7\,ns$ [209, 210] and afterwards it can execute the new instruction(s).

As it is shown in Table 5.6, the proposed aging-aware scheduling can significantly improve the efficiency of power gating and hence can greatly improve MTTF. Depending on the ALU-configuration, i.e. how many timing- and non-timing-critical units are available, the MTTF can be increased by several orders. If the number of ALUs remains constant, at least an improvement of around 1.8x is possible. If additional TC units are added to the design, even more than 2x are possible. However, in some cases, both approaches, the balanced and the aging-aware scheduling, deliver comparable results. This is due to the fact that a unit cannot be power gated in every idle period. Compared to the situation without power gating, in which MTTF is just 3 years, the improvements are even better and underline the efficiency of our proposed scheduling technique.

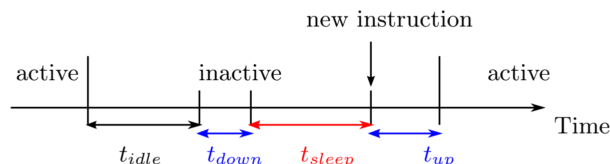


Figure 5.7.: Time flow of power gating periods of an execution unit

Application	1 ALU	2 ALUs		3 ALUs		4 ALUs		
		Aging-Aware	Balanced	Aging-Aware	Balanced	Aging-Aware		Balanced
	1/0/0	0/1/1	2/0/0	0/2/1	3/0/0	0/3/1	0/2/2	4/0/0
applu	3.62	6.77	3.62	6.83	3.68	6.85	9.67	3.74
bzip	3.61	3.61	3.61	3.61	3.61	3.61	3.95	3.61
equake	3.61	3.73	3.73	3.76	4.61	3.77	9.75	5.78
gcc	3.61	4.79	3.61	3.89	3.61	4.52	5.60	3.62
gzip	3.61	3.97	3.61	3.97	3.61	3.97	4.82	3.61
lucas	3.61	4.92	3.61	4.96	3.86	4.62	12.69	3.97
mcf	3.61	7.78	3.61	6.55	4.57	6.66	12.84	5.92
mesa	3.61	4.84	3.61	4.75	3.61	4.79	4.80	3.61
mgrid	3.65	32.82	3.65	32.57	3.72	32.68	33.77	5.87
parser	3.61	6.93	3.61	4.00	3.61	4.90	10.82	3.61
swim	3.61	4.83	3.61	4.84	3.67	4.84	9.98	3.99
twolf	3.61	5.95	3.61	4.77	3.63	5.62	9.81	3.72
wupwise	3.61	4.99	3.61	3.62	3.61	3.65	3.89	3.61
Average	**3.61**	**7.38**	**3.62**	**6.78**	**3.80**	**6.96**	**10.18**	**4.20**

Table 5.6.: MTTF in years for several SPEC2000 benchmarks for the standard (balanced) scheduling and the proposed aging-aware scheduling using power gating for different ALU configurations (“full”/NTC/TC)

Configuration		Performance Avg. IPC	MTTF [years]	Area-Cost μm^2	
1/0/0	(balanced)	1.21 (8)	3.61 (8)	45895	(1)
0/1/1	(aging-aware)	1.24 (7)	7.38 (2)	49546	(2)
2/0/0	(balanced)	1.39 (1)	3.62 (7)	91790	(3)
0/2/1	(aging-aware)	1.38 (6)	6.78 (4)	95411	(4)
3/0/0	(balanced)	1.39 (2)	3.80 (6)	137685	(6)
0/3/1	(aging-aware)	1.38 (4)	6.96 (3)	141336	(7)
0/2/2	(aging-aware)	1.37 (5)	10.18 (1)	99092	(5)
4/0/0	(balanced)	1.39 (3)	4.20 (5)	183580	(8)

Table 5.7.: Summary for the standard (balanced) scheduling and the proposed aging-aware scheduling using power gating for different ALU configurations (“full”/NTC/TC) in terms of performance, MTTF and costs; Values in () represent the ranking

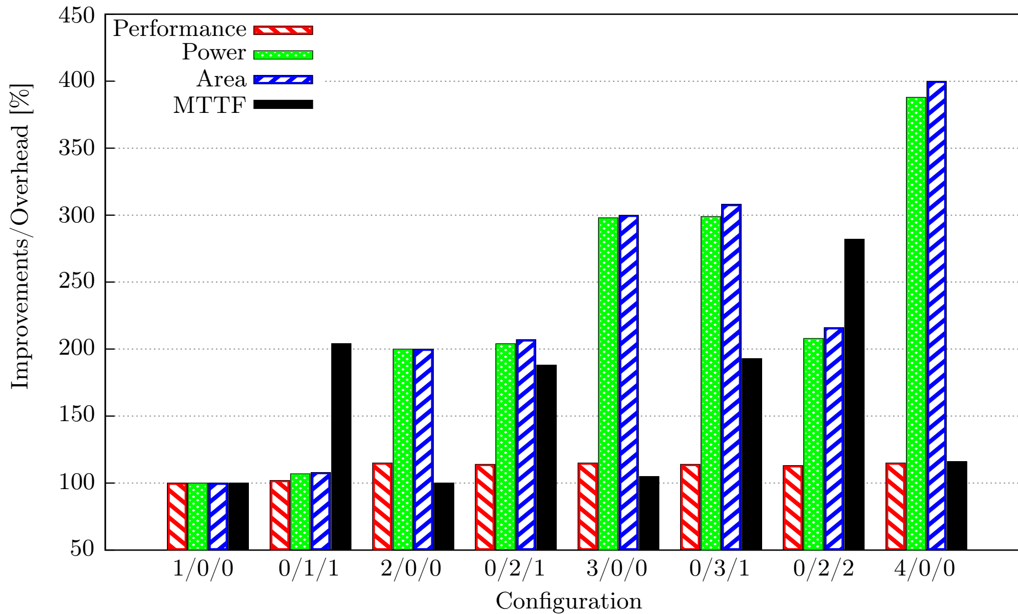


Figure 5.8.: Comparison of different ALU configurations (“full”/NTC/TC) using power gating in terms of performance, MTTF, area and power consumption

In addition, the performance penalty of the proposed aging-aware scheduling technique is usually very small as shown in Table 5.7. If 3 or 4 ALUs are available the performance penalty is at most 1.5 %, while at the same time MTTF can be significantly improved, and also the costs in terms of area and power are significantly lower, if the aging-aware scheduling is used. Only in case a design with 2 ALUs is transformed to a design with a single TC and a single NTC unit the performance will be significantly impaired. However, as the costs for TC units are very small (see Table 5.3) we propose to add specialized TC units to the design, such that the overall number of ALUs increases. By this means, with minimal additional cost, the performance penalty will be almost negligible and the lifetime will improve significantly.

Please note, that in some cases the performance with a higher number of ALUs is slightly lower than the performance with fewer ALUs. This is due to the fact, that with more ALUs, power gating can be applied more often. However, this also means, that the penalty due to the wake up latencies has more impact and hence the overall performance can decrease, although more ALUs are available. Moreover, the performance is in many cases lower, if power gating is used, compared to the IVC usage. This is because IVC can be used in every idle cycle, without any performance penalty, while power gating always infers a performance reduction due to long wakeup latencies.

Moreover, Table 5.7 and Figure 5.8 illustrate that the configuration choice is always a trade-off between performance, lifetime and cost. For example, the best performing configuration has only an average lifetime of 3.62 years which is the second to last place in our evaluation. In general our proposed aging-aware scheduling using specialized functional units is always the best choice, no matter how many units are available.

5.4. Comparison with Related Work

In various microprocessors such as FabScalar, OpenSPARC T1 (see Section 4.4) and IVM the execution units limit the overall microprocessor lifetime, due to their fast wearout [12, 14, 146]. Hence, in order to improve the overall microprocessor MTTF, wearout in the functional units needs to be addressed. Beside the circuit-level techniques discussed in Section 2.4, this can be achieved with microarchitectural solutions.

One solution is to periodically invert all operands to avoid that certain transistors are almost permanently stressed. Such an approach is proposed in [11] and is orthogonal to the proposed scheduling technique. Hence, both approaches can be employed together.

Another possibility is to use clock- and power gating to mitigate aging as proposed in [15, 64]. However, clock gating does not alleviate BTI, since the transistor states of the combinational logic are not affected. In addition, power gating is not very effective, as long as the default balanced scheduling approach is used, in which the functional unit is changed with every incoming instruction [15], since the idle periods are too short. As a result, pure clock and power gating approaches do not significantly improve the lifetime. To achieve a longer MTTF, clock gating can be combined with all kinds of input vector control [65, 211] or aging-aware NOP-instructions [145] to alleviate BTI-induced aging in functional units. However, also this approach can only be applied during idle cycles and because of that its effectiveness depends on the idle ratio. In this regard, it is possible to adjust the instruction scheduler to improve the length of the idle periods to maximize the recovery time. In combination with clock and power gating this can considerably improve the MTTF of the execution units [15]. However, by increasing the idle periods, the number of units that can be used in parallel reduces, which in turn significantly impairs the performance. For example, the scheduling approach from [15] that uses the same functional unit for 10^9 cycles before changing it, reduces the performance by 14 % on average, which is much more than using our proposed methodology.

Beside the performance problem, all balanced scheduling approaches proposed in literature for instructions and applications [15, 76–78] cannot achieve the same lifetime results as the aging-aware scheduling technique presented in this chapter, as shown in Section 5.3. This is due to the fact that the previous techniques ignore the different timing criticality of different instructions (applications). Consequently, these approaches treat all instructions (applications) in the same way which leads to a lower overall MTTF. This underlines that cross-layer approaches such as the aging-aware instruction scheduling can achieve better results, i.e. longer lifetime in combination with better performance or reduced cost.

5.5. Summary and Conclusion

In this chapter, we presented a unique cross-layer instruction scheduling policy to slow down the delay degradation of the functional units in microprocessors, as these are typically the most critical components. The novelty of this scheduling scheme compared to state-of-the-art solutions is that the timing criticality of different instructions executed by the same functional unit (at circuit-level) is taken into account during the scheduling process. In other words, the scheduler considers the "per-instruction" slack instead of the overall slack of a functional unit over all possible instructions, which is the conventional approach. As a result, critical and non-critical instructions are scheduled to different functional units to improve the idle ratio of those units executing the critical instructions. This higher idle ratio can be exploited by various aging relaxation techniques such as input vector control or power gating to alleviate wearout and by that means to improve the lifetime of the functional units. As demonstrated by our extensive simulation results, the lifetime can be improved considerably, with only a small performance penalty which is due to an unbalanced load distribution among all functional units. At the same time, power and area can be slightly improved. Alternatively, the proposed scheme also allows to improve both lifetime and performance, with only a small power and area overhead.

CHAPTER

SIX

AGING-AWARE INSTRUCTION SET ENCODING FOR LIFETIME IMPROVEMENT

Beside the functional units also the decoding stages of the instruction pipeline can become critical and limit the overall microprocessor lifetime, as these components also suffer from high wearout rates (see Figure 4.1). Therefore, to tackle wearout in these parts of the microprocessor, an *aging-aware instruction set encoding methodology (ArISE)* is proposed in this chapter. At first, we demonstrate why the decoding stages have to be addressed and motivate the envisioned aging mitigation technique. Next, the methodology to alleviate accelerated transistor aging inside the decoding stages is presented, followed by a comprehensive evaluation, which shows the benefits of this technique. Afterwards, a summary of related work is provided, and finally the chapter ends with a concluding section.

6.1. Problem Introduction and Motivation

As explained in Chapter 4 and Chapter 5, the microprocessor lifetime is often determined by the execution units [12, 146]. Therefore, various aging-aware solutions have been proposed to extend the lifetime of these units. However, many of these techniques such as aging-aware instruction scheduling presented in the previous chapter or aging-aware NOP instructions [145] are hardly applicable to other stages of the instruction pipeline. Hence, by using such techniques and extending the lifetime of the execution stages, the impact of other near-aging-critical stages on the overall microprocessor lifetime becomes more pronounced. As a result, MTTF improvements for the entire microprocessor become much smaller than the individual benefits for the execution units. Therefore, additional approaches are necessary to also improve MTTF of other pipeline stages. In particular the *decoding stages* have to be targeted, since these suffer from high wearout rates and are almost as critical as or even more critical than the execution stages, as illustrated in Figure 6.1. However, such approaches to improve lifetime of pipeline stages in the frontend are still very rare.

To improve the MTTF of the decoding stages, the instruction opcodes can be modified. Since these are part of the input stream of the decoding stages, the opcodes strongly affect the internal signal properties, i.e. signal activity and signal probability. Moreover, by changing the instruction opcode, the instruction set encoding[1] (ISE) is modified, which results in circuit-level changes which in turn also affect the aging rates. Both aspects together lead to a considerable influence of the ISE on the wearout of the decoding stages as depicted in Figure 6.1 for the FabScalar microprocessor (results were obtained with the RTL platform presented in

[1]The mapping of instructions (e.g. add) to their binary representation (e.g. 1101), i.e. opcode, is called Instruction Set Encoding.

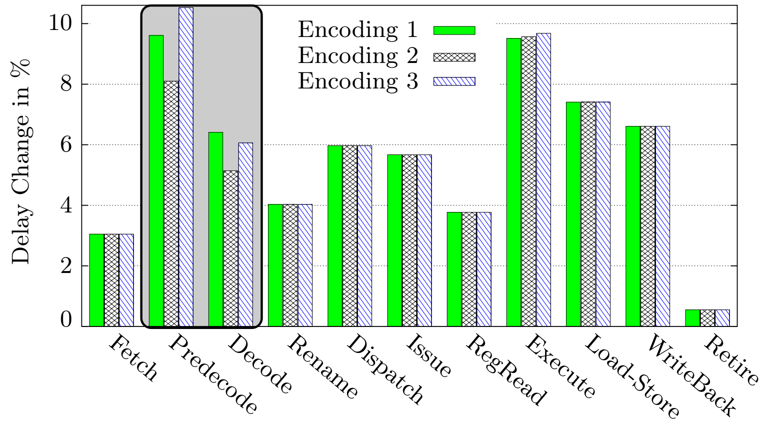


Figure 6.1.: Worst-case delay change after 3 years for different pipeline stages for 3 different ISEs

Chapter 3.4). In terms of MTTF the depicted difference in delay degradation rates translates to more than 2x difference, if a timing guardband of 10 % is used. Moreover, the results also show that the decoding stages can even become lifetime-limiting, if the ISE is not designed aging-aware (see Encoding 3). On the other hand, the effect of ISE changes on other stages is negligible, since the instruction opcode forms only a small part of the inputs of the other stages (or is no input at all).

In summary, it is important to address aging in the decoding stages of the instruction pipeline, which can be done by creating an aging-aware ISE. Therefore, *a novel aging-aware instruction set encoding technique* called *ArISE* is proposed in this chapter. This technique exploits the fact that the instruction set encoding (ISE) has a considerable impact on wearout of the decoding stages, since it affects the input patterns (via the applied opcodes) and the gate-level implementation of these stages, which both influence wearout. To find a good ISE in terms of MTTF, the approach uses hierarchical optimization algorithms to obtain an aging-aware opcode for each instruction in such a way, that the overall lifetime of the decoding stages is improved. Moreover, since the ISE also affects the energy consumption of memory components (via memory inputs switching activities) [212–214], the optimization algorithms consider the energy consumption of memory elements as a second optimization objective to co-optimize lifetime and energy consumption.

6.2. ArISE: Aging-aware Instruction Set Encoding

Designing an aging-aware ISE is a challenging task, as many instruction encodings have to be modified to become aging-aware. For example, in case of the FabScalar microprocessor the instruction set architecture contains 131 instructions that are encoded using 8 bits. This means that there are $(2^8)!/(2^8 - 131)! \approx 10^{297}$ encoding possibilities. Similar results are also obtained in case of real processors, for example for the x86 instruction set (also 8 bit opcode). In general, if the instruction set supports n instructions with $2^{k-1} \leq n \leq 2^k$ (i.e. k bit opcode), the complexity of the solution space is:

$$\frac{2^k!}{(2^k - n)!} >> e^n. \tag{6.1}$$

In addition, since most encodings infer modifications in the gate-level implementation of the stages and affect signal duty cycles as well as switching activities, each encoding requires a new synthesis and simulation flow to determine its aging impact. Moreover, instructions cannot be considered independently, since duty cycles and switching activities depend not only on

the actual instruction, but also on the preceding and subsequent instructions in the executed instruction streams. Therefore, to see the effect of a particular encoding, or even optimizing the encoding for only one instruction, one has to simulate complete instruction streams. Due to this complexity an exhaustive method is infeasible. Hence, a hierarchical heuristic methodology is used to find an aging-aware ISE. The formal description of the corresponding optimization objective is:

$$\text{Find} \max_{ISE \in \{\text{Set of ISEs}\}} \left\{ \min_{i \in \{\text{Decoding Stages}\}} \{MTTF_i\} \right\} \tag{6.2}$$

As heuristics simulated annealing and a genetic algorithm are employed, due to the following reasons. First, neighboring ISEs where at most two instructions are different should result in similar MTTF results, as only small changes in the hardware are required, and also the input signal properties will remain similar. Thus, using simulated annealing and exchanging two neighboring ISEs, the lifetime of the decoding stages can be improved in a stepwise manner. Second, the main challenge of simulated annealing is that the optimization is a sequential process. To overcome this issue, and by that means to improve the optimization runtime, we have chosen a genetic algorithm, where the entire set of ISEs is optimized with a high degree of parallelism. As a consequence, both choices provide very good results, as presented in Section 6.3. Therefore, in Section 6.2.1 the chosen simulated annealing approach is presented, followed by the methodology based on a genetic algorithm (Section 6.2.2). The hierarchical approach to improve the efficiency is discussed in Section 6.2.3. Afterwards, in Section 6.2.4, the methodology to co-optimize memory power consumption and lifetime of the decoding stages is proposed. Furthermore, a runtime analysis and further improvements are presented in Section 6.2.5. The application of ArISE to a real system is described in Section 6.2.6.

6.2.1. Optimization using Simulated Annealing

The starting point of the approach based on simulated annealing [215] is a random ISE, as detailed in Algorithm 6.1. For this ISE all pipeline stages have to be synthesized and their MTTF values need to be extracted according to the flow presented in Chapter 3.4 (Steps 1+2). Afterwards, a simulated annealing (SA) algorithm is invoked (Steps 3-9). As a first step, it generates a “neighbor” ISE for the current one (Step 5). For this purpose, the neighbor definition 6.2.1 is employed to guarantee that every possible ISE can be generated. In other words, the ISE search space used by SA is not reduced by this definition. In addition, this definition makes sure that the difference between the new and the old ISE is not that huge that the optimization process is too random.

Definition 6.2.1 (ISE Neighbor)
Two ISEs are neighbors if and only if a) both ISEs differ only in one instruction opcode (only possible if there are more binary opcodes than instructions), or b) the second ISE can be derived from the first one by exchanging the opcodes of two instructions.

The next step of the optimization algorithm is to estimate the overall microprocessor MTTF using the newly generated ISE (Step 6). Afterwards, it is evaluated if the new ISE will replace the old one (Step 7). Therefore, the exponential function detailed in Equation (6.3) is applied as cost function:

$$P(\text{accept ISE}) = \exp\left(-(d_{new} - d_{old})/T\right) \overset{?}{>} P_{reject}. \tag{6.3}$$

The inputs for the cost function are the worst-case delay (which can be used to represent MTTF) after 3 years for the new and old ISE. P_{reject} is the reject probability and T the annealing temperature, which can be iteratively reduced (Step 4) according to the SA principal [215]. Since SA tries to find the global optimum, it does not only accept solutions with lower costs

1. Select a starting ISE → ISE
2. Evaluate $MTTF$ and delay D after X years for ISE

/ run simulated annealing */*

3. **While** $MTTF < MTTF_{stop}$ or number of steps < limit **do**
   4. Adjust temperature T

   / Mutation */*

   5. Generate "neighbor" ISE of $ISE \rightarrow ISE'$

   / Evaluation */*

   6. Evaluate $MTTF'$ and delay D' after X years for ISE'
   7. **If** ISE' is acceptable according to Equation (6.3)
      8. $ISE = ISE', MTTF = MTTF'$
      9. $ISE_{best} = ISE$ */* memorize best ISE */*

      End

   End

Done.

Algorithm 6.1: Simulated annealing approach to generate an aging-aware ISE

(i.e. better MTTF) but also intermediate solutions with higher costs to avoid local optima. In both cases a neighbor for the new ISE is created (Step 8-9) and the evaluation continues with this new neighbor. Otherwise, if the costs for the new ISE are too high, a new neighbor for the old ISE is generated.

6.2.2. Optimization using a Genetic Algorithm

Another option to tackle the complex optimization problem of finding an aging-aware ISE is to employ a genetic algorithm (GA) [215, 216]. The advantage of an evolutionary approach instead of simulated annealing is that a pool of solutions is employed (as in GA) for the optimization process rather than a single solution that is iteratively improved (as in SA). Moreover, new candidate solutions are generated not only by "mutation" (as in SA), but also by "recombination" of two solutions from the pool, which promises better results than the semi-random search of SA.

The methodology using a GA approach is shown in Algorithm 6.2. The first step is to generate a set, s, of n random ISEs (Step 1). Afterwards, the overall microprocessor MTTF for each ISE in s is evaluated according to the flow presented in Chapter 3.4 (Step 2-3). If no ISE can satisfy the target lifetime, the genetic algorithm is started (Steps 4-15). Consequently, at first, the quality of each solution is judged based on its "fitness" (Step 5). The fitness, f_i, of ISE_i is defined as:

$$f_i = \frac{MTTF_i}{\sum_{j=1}^{n} MTTF_j} \in [0,1] \quad , \quad \sum_{i=1}^{n} f_i = 1 \tag{6.4}$$

According to this equation, the fitness f_i of ISE_i is determined by its MTTF ($MTTF_i$) and normalized to the sum of all MTTF values. As a result, all fitness values are between 0 and 1, and the sum equals 1. Moreover, the better the microprocessor MTTF with a particular ISE, the higher is the fitness value of this ISE. The calculated fitness values are then used to select $n = |s|$ ISEs from s (Steps 6-7). In this regard, the fitness of ISE_i determines the probability of selecting this particular ISE, i.e. the higher the fitness, the more likely this ISE will be selected. Note that since the selected ISEs are not removed from s, ISEs can be selected several times, which means that in the pool of selected ISEs, s', the same ISE can occur several times.

After n ISEs have been chosen, the next step is to recombine these ISEs (Steps 8-12). Therefore, two ISEs, ISE_i, and $ISE_j (i \neq j)$, in s' are randomly chosen and removed from s'.

1. Generate a pool of n random ISEs $s = \{ISE_1, \dots, ISE_n\}$
2. Evaluate $MTTF_i$ for every $ISE_i \in s$
3. $MTTF = \max MTTF_i$

/ run genetic algorithm */*
4. **While** $MTTF < MTTF_{stop}$ or number of steps < limit **do**
   / Fitness */*
   5. Extract fitness f_i of every $ISE_i \in s$ according to Equation (6.4)

   / Selection of n ISEs */*
   6. Generate empty set s'
   7. Select $ISE_i \in s$ with probability f_i and put it in s' (n times)

   / Recombination */*
   8. Generate empty set s
   9. **While** $s' \neq \emptyset$ **do**
      10. Select $ISE_i, ISE_j \in s'(i \neq j)$
      11. Generate crossovers ISE'_i, ISE'_j and put these in s
      12. Remove ISE_i, ISE_j from s'
      End

   / Mutation */*
   13. For all $ISE_i \in s$: Modify opcode k with probability p (for all k)

   / Evaluation */*
   14. Evaluate $MTTF_i$ for every $ISE_i \in s$
   15. $MTTF = \min MTTF_i$
   End
Done.

Algorithm 6.2: Genetic algorithm to generate an aging-aware ISE

Then, the chosen ISEs are combined to generate two crossovers, which are put in a new pool s. This procedure is repeated until there are no more ISEs in s' (therefore n needs to be even). In this thesis, the recombination of two ISEs works as follows: For each opcode of the first ISE child, it is randomly decided from which parent the opcode is inherited as illustrated in Figure 6.2, while the second ISE child inherits the other, not-selected, opcode. For instance, in Figure 6.2 the opcode for `add` in $ISE_{1\times2}$ is randomly chosen to be `a3` which comes from ISE_2. Consequently, in $ISE'_{1\times2}$ the opcode has to `a6` from ISE_1. However, this recombination process cannot be purely random, as there is the constraint that not two instructions in one ISE can have the same opcode, which needs to be considered during the crossover phase. For example, in Figure 6.2, the opcode for `xor` in $ISE_{1\times2}$ cannot be `a7`, which comes from ISE_2, as `a7` is already used for `sub`. Hence, the opcode from ISE_1 has to be used, i.e. `a8`.

The next GA step is to allow a mutation of the new ISEs (Step 13). Therefore, every ISE opcode is changed with the probability p. Again, since not two instructions can have the same opcode, always two opcodes are rotated in case an opcode is modified. Finally, these newly generated ISEs are evaluated based on the overall microprocessor lifetime (Steps 14-15) and the GA procedure starts again.

Please note that the MTTF evaluation is the most time consuming part for the SA and GA approaches, as a re-synthesis of the affected pipeline stages is required. Hence, if the SA approach evaluates N solutions, the runtime is $O(N)$. Instead for GA it is only $O(N/n)$, as all solutions within one set can be evaluated in parallel. For example, SA requires about 6 hours to perform 100 iterations (see Section 6.2.5), i.e. to evaluate 100 solutions. In contrast, if $n = 20$, GA only requires 20 minutes to analyze the same amount of solutions, which is a huge advantage of the GA approach.

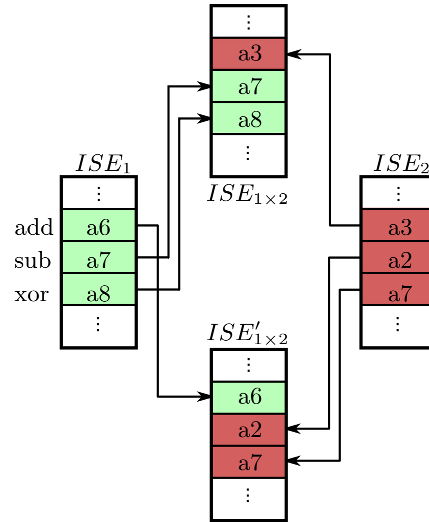


Figure 6.2.: Exemplary recombination of ISE_1 & ISE_2 to $ISE_{1\times2}$ and $ISE'_{1\times2}$

6.2.3. Hierarchical Optimization

To improve the efficiency of the optimization process, the number of steps until an acceptable solution is found needs to be reduced. Therefore, the *hierarchical optimization approach* shown in Algorithm 6.3 and illustrated in Figure 6.3 is applied. First the instructions are classified into different (sub)groups based on their characteristics, e.g. all branch/jump-instructions are put into the same group (Step 1). Afterwards, at each hierarchy-level, the (sub)groups are ranked according to their aging impact (Step 2). However, as the aging rate strongly depends on the encoding, the real aging rates cannot be used for this ranking. As a matter of fact, the instructions affecting the hardware-implementation (of the decoder), i.e. the instructions that are decoded in this stage, have the biggest aging impact. For example, in FabScalar's predecode stage only the branch/jump-instructions are handled, and hence these have the strongest influence. Therefore, the impact on hardware modifications due to a particular coding scheme is used to form the ranking. If there are several (sub)groups affecting the hardware-implementation, they are ranked depending on their occurrence frequency (Step 2.2). The next step is to find the best encoding for each (sub)group, for all hierarchy-levels starting at the coarsest hierarchy-level (Step 3). Thereby, at each level, the group ranking determines the

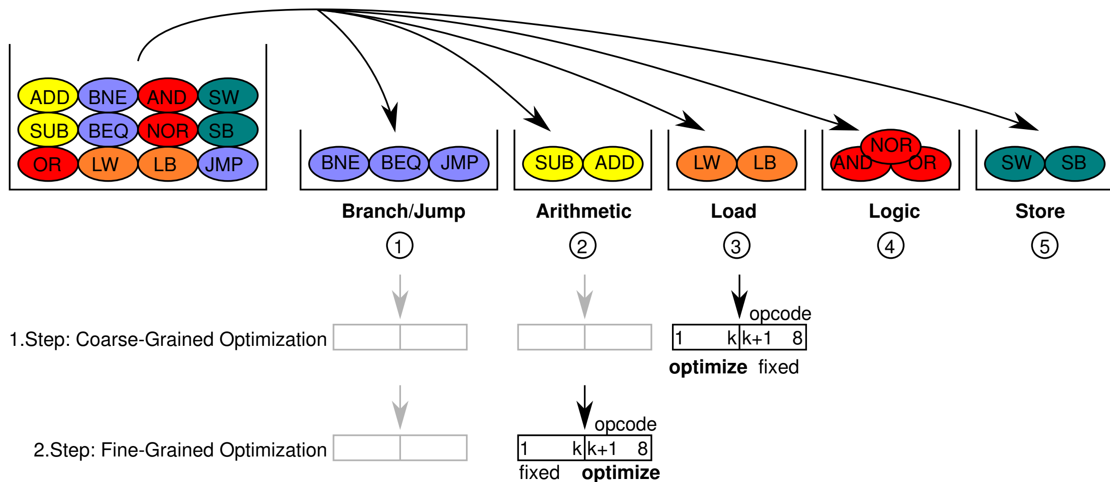


Figure 6.3.: Instruction classification, ranking and sequence of the hierarchical optimization process

1. Partition instructions into groups and subgroups
   /**Instruction groups, subgroups inside groups,* */
   /**instructions insides subgroups, etc.**/

2. Rank each group (and subgroups subsequently)
   2.1 Based on their hardware-impact
   2.2 **If** there are groups/subgroups with same ranking
       then use occurrence frequency to rank these

3. **For** the coarsest down to the finest hierarchy-level **do**
   For the highest ranked group down to the lowest **do**
   3.1 Find the best encoding for the elements within that group
       /**Either exhaustive or with simulated annealing**/
   3.2 Stop as soon as MTTF is satisfactory
   Endfor
   Endfor

Algorithm 6.3: Hierarchical approach to obtain an aging-aware ISE

optimization order. As a result, an exhaustive technique can be applied, if there are only a few (sub)groups at the same hierarchy level with considerable aging impact. Otherwise, a heuristic optimization process, either GA or SA, can be applied for the (sub)groups.

Using this approach, at each hierarchy-level, only the opcode bits corresponding to that level are modified. For example, if there are 16 groups at the coarsest level, only the four most significant opcode bits are modified (i.e. 16! configurations). As we used only 16 instruction groups, each of which contained at most 16 instructions the search space for the entire optimization process became much smaller than the original space ($16!^2 \approx 4 \cdot 10^{26}$ vs. 10^{297}), leading to fewer optimization steps. Although this search space reduction can theoretically reduce the solution quality, the ISEs obtained with this approach are sill very good and yield lifetime improvements of up to 2.4x, as shown in Section 6.3.

We compared this approach with a non-hierarchical one that uses the entire search space. To limit the runtime for the latter, we enforced that one of the modified instructions in each iteration had to be one of the 10 most frequent instructions (here: the 10 most frequent instructions occurring in the SPEC2000 benchmarks) to avoid modifying only the encoding of infrequent instructions, which have negligible effect on the overall aging rate. While this version usually finds a feasible ISE within the first 80 to 90 iterations, the hierarchical approach needs usually at most 30 iterations (i.e. at most 2 hours) to obtain an ISE which can achieve a lifetime of at least 6 years (3 years with the default ISE), i.e 3x improvement in runtime.

Please note that the hierarchy can be divided into fewer or more optimization levels, depending on the size of the instruction set and maximum runtime. Fewer levels allow to analyze more ISEs, and thus a better solution can be obtained compared to an approach with more levels, which however, has a much shorter runtime.

Furthermore, please note that also other grouping schemes can be used for this purpose. In this thesis, we considered a two-level categorization and grouped the instructions based on their type, e.g. all branch, arithmetic, load, store and logic instructions had their own group, which is also the highest level of the hierarchy. The next hierarchy-level is the lowest level, i.e. all instructions inside a group are optimized independently. Other schemes, for example based on the occurrence frequency, are also possible and can affect the runtime as well as MTTF. However, as within the 16 most frequent instructions 10 would belong to the branch/jump group, the difference between these two approaches for the processor under consideration is minimal. Therefore, we only present the results using the type-based classification.

As a final point, it is also important to note that instruction grouping is a standard in the design of ISEs, to avoid that similar instructions have divergent opcodes. Hence, without

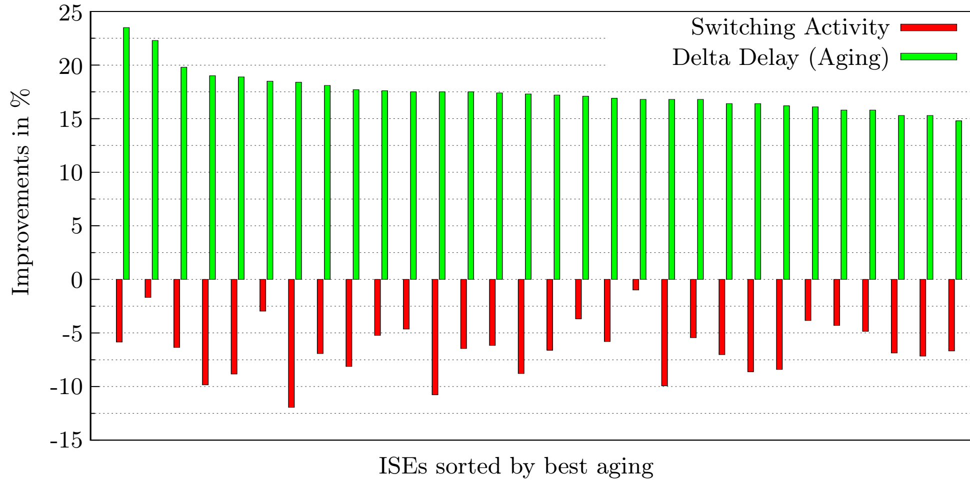


Figure 6.4.: Improvements of various ISEs in terms of aging (delta delay) and switching activity of the opcode bits in the instruction buffer compared to the default ISE (negative numbers mean that the ISE is worse than the default ISE)

this hierarchical approach, aging-aware ISEs obtained with the standard simulated annealing or genetic algorithm approach may contain various divergent opcodes. Thus, the hierarchical approach is not only beneficial in terms of simulation effort and runtime, but also in terms of ISE design.

6.2.4. Co-Optimization of Lifetime and Memory Power Consumption

ISEs are typically designed in such a way, that the switching activity in instruction-related memories such as the instruction cache, instruction buffer or pipeline registers is reduced to lower the energy consumption of these memories [212–214]. Thus, as our proposed approach changes the ISE, it also affects the switching activity of memory cells that store the bits of the instruction opcode (by up to 12 % as shown in Section 6.3.3). Consequently, the aging-aware ISE also has a significant influence on the power consumption of instruction-related memories. Moreover, as these memories are quite small, dynamic power is typically more important than leakage power due to a high access ratio. For example, using FabMem [217] we obtained the read and write power for the instruction buffer of FabScalar to be 40x to 60x higher than the leakage power. Hence, an increase in the dynamic power will significantly affect the overall power consumption of these memories.

To study this effect we use the instruction buffer of FabScalar as case study, which stores up to 32 decoded instructions including their opcode. For this memory, Figure 6.4 shows the change in switching activity of the opcode bits as well as the aging improvements for the decoding stages for various ISEs. As one can see, there is no correlation between the aging rates of the decoding stages and the switching activity inside the memories. Moreover, the figure shows that the switching activity of the affected bits increases by up to 12 %. Hence, it is important to co-optimize the switching activity inside the memories and the aging behavior of the decoding stages.

For this purpose, as the dynamic power consumption is a direct function of the switching activity, we added the average toggle rate of the affected bit-cells to the optimization process. This is achieved by modifying the fitness function of the GA optimization (Equation (6.4)) and the acceptance probability of the SA approach (Equation (6.3)). Beside lifetime or delay, these have to also consider the average toggle rate (TR) of the affected memory cells. Consequently,

the modified fitness function looks like:

$$f_i = \frac{MTTF_i \cdot \frac{1}{tr_i}}{\sum_{j=0}^{n} MTTF_j \cdot \frac{1}{tr_j}} \in [0,1] \tag{6.5}$$

where tr_i is the average toggle rate of the affected memory cells using ISE_i. For the SA optimization, a second acceptance probability is defined, and an ISE is only accepted, if both acceptance probabilities ($P1$ and $P2$) are satisfied.

$$\begin{aligned} P1 &= \exp\left(-(d_{new} - d_{old})/T\right) \overset{?}{>} P_{reject_d} \\ P2 &= \exp\left(-(tr_{new} - tr_{old})/T\right) \overset{?}{>} P_{reject_{TR}} \end{aligned} \tag{6.6}$$

Using these modifications, the average switching activity and the lifetime are co-optimized, and thus ISEs that only optimize one of those two aspects at the cost of the other parameter are avoided.

To extract the toggle rate of the affected memory cells, the input stream for each affected memory block is dumped to a file using the starting ISE. Afterwards, for each new ISE, only the affected opcode bits are modified in this stream and the new average toggle rate is obtained. As this process is very fast, the runtime of a single optimization step is not impaired by the co-optimization.

6.2.5. Runtime Analysis and Further Improvements

The runtime for a single SA or GA step depends mainly on the time required by the RTL platform described in Chapter 3.4 to obtain the MTTF for the affected stages. The major contributors to the runtime of this flow are the necessary re-synthesis steps of the affected stages every time the ISE is modified and the required simulations to extract the behavior of all signals. In contrast, as explained in Chapter 3.4, the remaining steps are negligible. For this reason, we propose to replace the (post-synthesis) simulations during the aging estimation phase as shown in Figure 6.5. Instead, prior to the start of the optimization process, a behavioral simulation is performed and the input-stream for all affected stages is stored in a file. Then, during the optimization process, this input-stream is modified according to the ISE changes, i.e. old opcodes are replaced with modified ones. The resulting input signal properties are then given to the synthesis tool (in form of a Switching Activity Interchange Format, SAIF, file). The synthesis tool propagates these properties through the entire design and calculates the signal properties for all (internal) signals. By that means, extracting the internal signal properties takes a negligible fraction of time compared to post-synthesis simulation (a few seconds vs. ≈ 30 minutes for 10^6 clock cycles). However, the aging estimation accuracy will be impacted, as signal correlations are not taken into account. Nevertheless, the inaccuracy is very small as shown in Section 3.4.3, and hence this scheme is accurate enough to be used during the optimization process. For the final results again the accurate but time-consuming flow is employed to extract the signal properties to ensured that the obtained results are accurate and that the optimization process was successful.

As a result, the runtime for a single SA or GA step depends now mainly on the time required for synthesizing the stages, which is required every time aging (i.e. MTTF and delay) has to be estimated. In addition, only aging-critical stages that are sensitive to ISE need to be considered (here: decoding stages). Doing so, a single step takes in our case less than 4 minutes, which means that 100 steps can be performed within 6 hours[2].

[2]All runtime measurements in this chapter were performed on a system with AMD Opteron 6174 processors running at 2.2 GHz, 256 GByte RAM and Redhat Enterprise Linux 6.5 64 bit

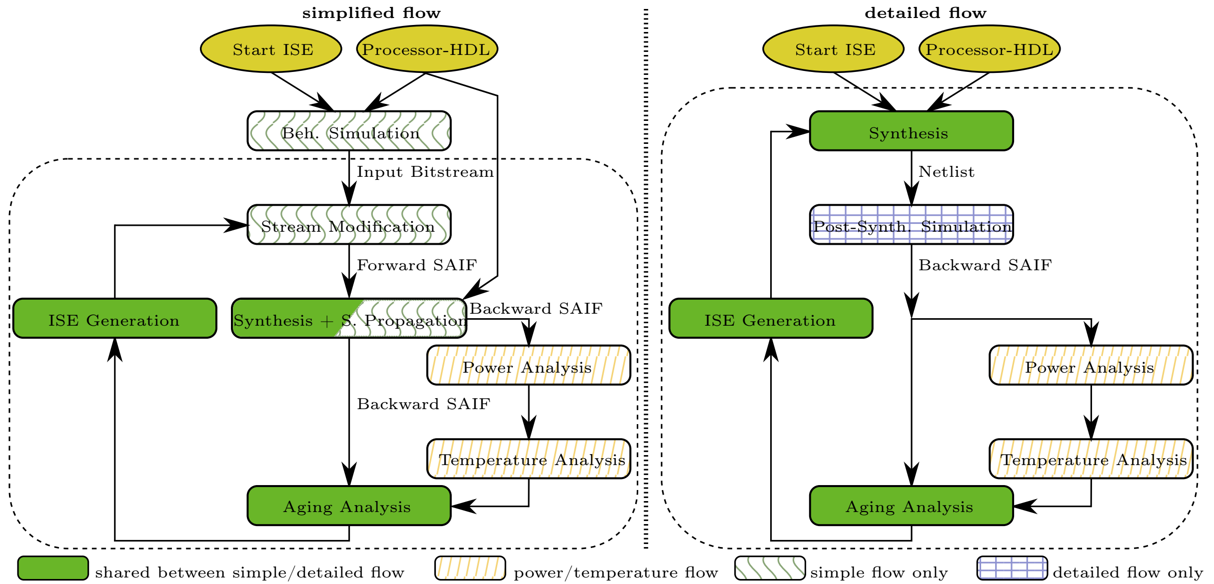


Figure 6.5.: ISE optimization flows including aging analysis: Left: fast version without post-synthesis simulation, Right: slow and accurate version

Moreover, we avoid re-evaluating the same ISE in multiple steps, to reduce the runtime further. Also, the best solution found during all performed iterations is memorized, such that we can always apply the best ISE in terms of MTTF that was found and not only the last accepted one, which may not necessarily be the best one.

Another improvement to reduce the runtime is to add optimization constraints. In particular this holds for the SA approach which we extended with the possibility to use lower and upper quality bounds for the solutions as shown in Figure 6.6. Using upper bounds means that whenever the lifetime is better than a given threshold ($MTTF_{up}$), the changes that led to this particular ISE will be kept for all following iterations. In contrast, if the lifetime is worse than a given threshold ($MTTF_{low}$), the changes that led to that particular ISE will be avoided for all following iterations (lower bound). By that means, the search space becomes significantly smaller, which improves the convergence speed. However, the solution quality can be affected (see Section 6.3 for more details). Therefore, the bounds need to be carefully selected. In this regard, the selection depends on various design constraints, such as target lifetime, lifetime without enhancements and optimization budget (e.g. time available to optimize ISE).

6.2.6. Applying Modified Instruction Encoding

An ISE modification affects the usability of existing software binaries since software compiled for the original ISE can no longer be executed. To address this issue a software- or a hardware-based technique can be used.

The software-based solution re-compiles applications using a compiler based on the modified ISE either on-the-fly (runtime compilation as it is used for instance by OpenCL [218]) or

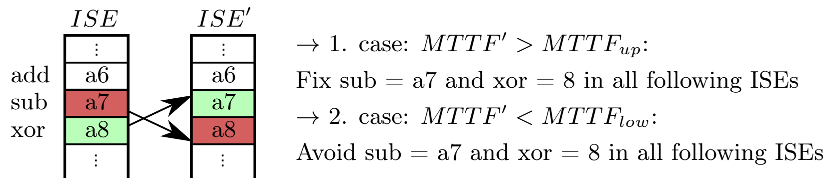


Figure 6.6.: Using lower and upper bounds as optimization constraints

Processor	FabScalar [1]
Frequency	740 Mhz + 10 % guardband
Technology	TSMC 65 nm
Architecture	out-of-order
Pipeline Stages	11
Simultaneous Multithreading	no
Frontend-Width (per Thread)	4 insts/cycle
Exec. Units (ALU/MUL/AGEN)	1/1/1
Ambient Temperature	40 °C

Table 6.1.: Architecture setup of the FabScalar microprocessor

when the application is started for the first time. In case of application-specific processors re-compilation is not necessary, since hardware and compiler are typically designed in close interaction and backwards compatibility is seldom required. In general, the advantage of this approach is that it infers no additional hardware costs and is easy to implement.

The hardware-based approach is intended for processors that have to be backward compatible to old software (e.g. x86-based microprocessors), or when re-compilation is not a feasible solution. Therefore, a mapper is used, which translates the standard ISE into the aging-aware ISE at runtime by using a lookup-table or logic-statements (if-else). This mapping can be done while the instructions are written to or read from the instruction cache. Our analysis shows that, the overhead of such a solution is negligible (less than 0.4 % overhead for FabScalar). Besides area overhead this mapper can potentially also impact performance, since it could increase the critical path length. However, in case of FabScalar using a mapper did not negatively affect the critical path, i.e there was no performance penalty. Therefore, for this dissertation the hardware-based approach was employed.

For processors that rely on microcodes, i.e. each instruction is translated by the hardware into several micro-instructions, the idea of an aging-aware ISE design can be also applied to micro-instructions. In this case, no additional hardware overhead would be necessary to ensure backward compatibility, as the micro-instructions are only used internally by the processor and all instruction set related encodings could remain the same.

6.3. Experimental Results

In this section, the impact of an aging-aware ISE on the FabScalar microprocessor is evaluated, the co-optimization of lifetime and memory power consumption is analyzed and our proposed technique is compared to the method from [11], which periodically inverts the opcode bits to achieve better wearout rates.

The processor, detailed in Table 6.1, was synthesized using the TSMC 65 nm library with performance-optimized timing constraints, i.e. to achieve the highest possible clock frequency (740 MHz $\equiv$ 1.35 ns clock period). The lifetime was estimated based on a guardband of 10 % with an ambient temperature of 40 °C resulting in processor temperatures between 50 °C and 75 °C, which is reasonable for modern processors (see for example our experimental data in Figure 3.9). For the evaluation we employed six SPEC2000 benchmarks provided with FabScalar (bzip, gap, gzip, mcf, parser, vortex) and simulated 10^6 cycles after a warmup (see Section 4.4 and Remark 4.4.1 for further explanations). Afterwards, the observed behavior during the simulated period was used to evaluate the power consumption and lifetime estimation based on the flow presented in Section 3.4. Accordingly, the overall lifetime of the decoding stages was 3 years with the default ISE, i.e. after 3 years the worst-case delay of the decoding stages is larger than 1.485 ns (i.e. 1.35 ns + guardband). The runtime of the different optimization approaches was measured on a server system with AMD Opteron 6174 processors running at 2.2 GHz, 256 GByte RAM and Redhat Enterprise Linux 6.5 64 bit.

SA	P_{reject} / limit	0.9 / 100
GA	limit / n / p	5 / 20 / 0.0625
General	$MTTF_{stop}$	6 years

Table 6.2.: Optimization parameters

6.3.1. Evaluation of an Aging-aware Instruction Set Encoding

In case of FabScalar, the Predecode stage is more critical than the Decode stage (MTTF of 3 years vs. 16 years), which means that the major optimization focus is on instructions with considerable aging impact on the Predecode stage to extend the overall MTTF. Therefore, all instructions were classified according to the hierarchical principle explained in Section 6.2.3. As a result, the branch/jump instruction group containing all branch and jump instructions had the highest influences on the wearout of the Predecode. Hence, first the best encoding for the corresponding instruction group was exhaustively determined using 16 iterations, with the result that the best encoding (using 8 bit) for this group is 0001XXXX[3].

Afterwards, the encoding for each instruction in the branch/jump group was optimized using the following approaches with the parameters detailed in Table 6.2.

SA Simulated Annealing: The encoding for the instructions inside the branch/jump is set according to Algorithm 6.1 using 100 iterations. Furthermore, the quality constraints explained in Section 6.2.5 were used. SA-L stands for the lower quality bound (here: $MTTF_{low} = 4.5$ years which corresponds to 1.5x improvement), SA-U for the upper quality bound (here: $MTTF_{up} = 5$ years which corresponds to 1.67x improvement) and SA-B means that both bounds are active. If no quality constraints are employed, the name SA is used.

GA Genetic Algorithm: The encoding for the instructions inside the branch/jump is set according to Algorithm 6.2 using 5 iterations with $n = 20$.

Please note that in all these optimization algorithms all decoding stages, i.e. the Predecode and Decode stages were considered. This is necessary, as some encodings improve the aging rates for one stage, but decrease the lifetime for the other stage. In contrast, the remaining pipeline stages were not (significantly) affected by modified ISEs (see Table 6.7), and thus these were excluded from the optimization process to improve the runtime. Furthermore, it is important to note that the quality bounds for SA were chosen based on the original lifetime of the decoding stages of 3 years. As we wanted to improve the lifetime by at least 1.5x, we set the lower quality bound to 4.5 years. The upper quality bound was set to 5 years, since higher values were not reached often enough to accelerate the convergence, and lower values did not help to achieve acceptable lifetime results. However, for other designs or lifetime targets the bounds may be chosen differently.

As all of the aforementioned approaches are based on a random selection process, we conducted 30 independent runs for each of the three approaches to evaluate which of them is best for the problem of finding an aging-aware ISE. Moreover, as soon as the worst-case delay is smaller than 1.46 ns (i.e. overall MTTF of decoding stages is better than 6 years) the optimization was stopped and the next run was started, to reduce the simulation time. The results of this analysis are given in Table 6.3.

It is obvious from the presented results that the simulated annealing (SA) approach without quality bounds is in summary the best optimization technique in this comparison. On average over all runs, it can achieve a lifetime of 5.6 years for the decoding stages, which corresponds to an improvement of 1.87x. In 27 % of all runs, it can improve the MTTF even by more

[3] XXXX represents the four bits used by the instructions in the branch/jump group, 0001 stands for the group encoding

		SA	SA-L	SA-U	SA-B	GA
All runs (30)	Avg. best† MTTF [years]	**5.6**	5.5	5.4	5.5	5.5
	Avg. best† delay after 3y [ns]	**1.461**	1.462	1.462	1.462	1.462
	Avg. No. of ISEs evaluated‡	37.6	40.4	**22.5**	42.5	56
	Avg. runtime‡ [min]	150.4	161.6	90.0	170.0	**11.2**
Runs with MTTF > 6 years	Occurrence	**8**	5	6	7	5
	Avg. best† MTTF [years]	**6.4**	6.0	6.1	6.0	6.2
	Avg. best† delay after 3y [ns]	**1.457**	1.459	1.458	1.459	1.458
	Avg. No. of ISEs evaluated‡	26.4	53.7	**24.4**	58.3	64
	Avg. runtime‡ [min]	105.6	214.8	97.6	233.2	**12.8**
Best run	Best MTTF [years]	**7.1**	6.2	6.7	6.3	6.8
	Best delay [ns]	**1.453**	1.458	1.455	1.457	1.455

Table 6.3.: Comparison of different optimization strategies to generate an aging-aware ISE for the decoding stages (best values are marked bold, MTTF & delay are worst-case over all decoding stages and all benchmarks) †: Avg. best MTTF (delay) is the average over all runs of the best MTTF (delay) obtained in each run ‡: Avg. No. of ISEs (runtime) is the average over all runs of the No. of ISEs (runtime) considering only the iterations until the best MTTF is obtained

than 2.1x and the best MTTF achieved is 7.1 years and hence an improvement of 2.37x. The quality bounds for SA do not help to improve the solution quality. In fact, the solution quality becomes worse, because the search space is much smaller. However, the advantage is that the number of performed iterations can be lower (see Section 6.2.5). In particular, SA-U benefits from this fact. As a result, SA-U performs only 40 iterations per run (on average), as then no more ISEs can be chosen (all opcodes are fixed). On the other hand, SA-L requires even more iterations than the standard SA approach, since also potentially “good” opcodes are excluded during the optimization process (see Section 6.2.5). Also the genetic algorithm (GA) is slightly less effective compared to the SA approach in terms of lifetime optimization. A possible explanation is that the combination of two “good” ISEs is not very likely to be “good” as well (see Table 6.4). Hence, the recombination step in GA usually does not improve the ISE quality significantly. However, as mentioned in Section 6.2.2, the degree of parallelism for GA is much higher. Hence, since $n = 20$, 20 times more runs can be performed in the same amount of time using GA instead of SA. This makes GA by far the fastest optimization method in our experiments, i.e. it requires just around 12 minutes to achieve an average lifetime of 5.5 years, while simulated annealing requires 150 minutes (10x slower) for 5.6 years (only 2 % better). This runtime advantage can be used to analyze more ISEs and thus improve the final solution.

Of course, the lifetime impact of a chosen ISE significantly depends on the application, since

	P1	P2	C1	C2	C3	C4	C5	C6
JUMP	1f	16	16	1f	1f	16	16	1f
JAL	18	10	10	10	10	18	18	18
JR	1c	18	18	18	18	1c	1c	1c
JALR	15	13	15	13	15	15	13	13
BEQ	1e	1f	1f	1e	1e	1f	1f	1e
BNE	1b	19	1b	19	1b	1b	19	19
BLEZ	13	1a	13	1a	13	13	1a	1a
BGTZ	1d	1b	1d	1b	1d	1d	1b	1b
BLTZ	12	11	12	11	12	12	11	11
BGEZ	16	17	17	16	16	17	17	16
BC1F	17	1e	1e	17	17	1e	1e	17
BC1T	19	12	19	12	19	19	12	12
MTTF	**5.7**	**6.8**	**5.3**	**3.6**	**4.2**	**4.4**	**4.6**	**4.0**

Table 6.4.: Recombination of two “good” ISEs (P1 & P2) resulting in “worse” ISEs (C1-C6) for GA (hexadecimal values represent the instruction opcodes)

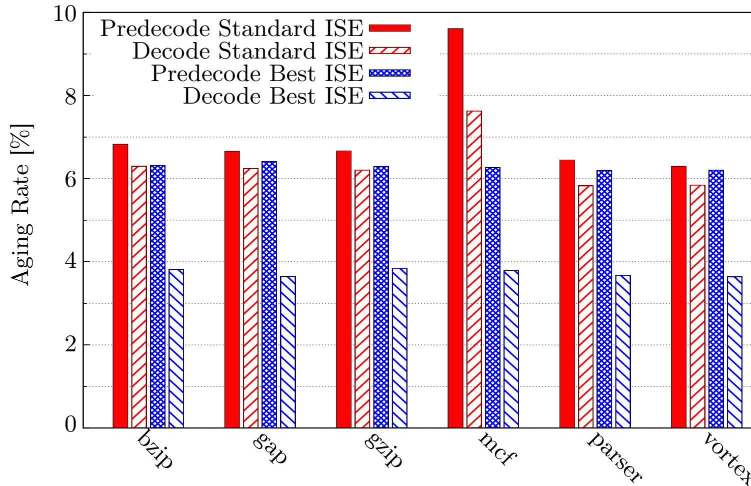


Figure 6.7.: Delay degradation of the decoding stages for different benchmarks using the standard ISE and the best ISE in terms of lifetime

Instruction	JUMP	JAL	JAR	JALR	BEQ	BNE
Start ISE	01	02	03	04	05	06
Best ISE	11	17	13	14	18	1c
Instruction	BLEZ	BGTZ	BLTZ	BGEZ	BC1F	BC1T
Start ISE	07	08	09	0a	0b	0c
Best ISE	12	16	1a	15	1e	1b

Table 6.5.: Differences between the default ISE and the best ISE in terms of MTTF (all opcodes in hexadecimal)

Stage	Standard Encoding			Best Encoding		
	Delay [ns] (0y)	Delay [ns] (3y)	MTTF [years]	Delay [ns] (0y)	Delay [ns] (3y)	MTTF [years]
Predecode	1.350	1.480	3.0	1.350	1.453	7.1
Decode	1.340	1.421	15.9	1.350	1.402	45.8
Overall	1.350	1.480	3.0	1.350	1.453	7.1 **+2.37x**

Table 6.6.: Improvements of the best ISE in terms of MTTF and delay (both worst-case over the used SPEC2000 benchmarks) for the evaluated Predecode and Decode stage

input patterns and temperature are application dependent. As depicted in Figure 6.7, the most critical application is the mcf benchmark, since the Predecode stage is very sensitive to the high stall cycle count of this benchmark (more than 70 %), as discussed in Section 4.4.1. The consequence of this observation is that representative workloads have to be considered for the ISE optimization, i.e. a cross-layer approach, like the one presented here, is mandatory.

6.3.2. Impact of an Aging-aware ISE on the entire Processor

To evaluate the effect of an aging-aware ISE on the entire processor, we picked the best encoding that was obtained during the previous investigation. With this encoding, shown in Table 6.5, the lifetime of the decoding stages improved by 2.37x as shown in Table 6.6. This improvement is mainly due to the Predecode stage benefits, but also the Decode stage shows a significant lifetime improvement with this encoding (almost 46 years instead of 16 years). Beside the decoding stages also the Execute stage as well as the Load-Store-Unit (LSU) are affected by ISE changes. Moreover, as the instruction mapper is part of the Fetch stage, also this stage is modified. However, as shown in Table 6.7, the effect of the chosen encoding on

Stage	Standard Encoding			Best Encoding			
	Delay [ns]	Energy [μJ]	Area [μm^2]	Delay [ns]	Energy [μJ]	Area [μm^2]	Changes
Fetch	1.35	12.2	23262	1.35 (±0 %)	12.7 (+4 %)	24434 (+5 %)	yes
Predecode	1.35	16.6	35030	1.35 (±0 %)	16.6 (±0 %)	34984 (−1 %)	yes
Decode	1.34	6.1	23616	1.35 (±0 %)	5.1 (−16 %)	23707 (+1 %)	yes
Rename	1.33	1.3	4050	1.33 (±0 %)	1.3 (±0 %)	4050 (±0 %)	no
Dispatch	1.33	0.1	1867	1.33 (±0 %)	0.1 (±0 %)	1867 (±0 %)	no
Issue	1.35	14.3	30719	1.35 (±0 %)	14.3 (±0 %)	30719 (±0 %)	no
RegRead	1.33	2.1	12061	1.33 (±0 %)	2.1 (±0 %)	12061 (±0 %)	no
Execute	1.35	6.4	27529	1.35 (±0 %)	6.3 (−2 %)	27468 (−1 %)	yes
LSU	1.35	50.6	107664	1.35 (±0 %)	50.6 (±0 %)	107664 (±0 %)	yes
WriteBack	1.34	4.0	3183	1.34 (±0 %)	4.0 (±0 %)	3183 (±0 %)	no
Retire	1.35	1.6	3201	1.35 (±0 %)	1.6 (±0 %)	3201 (±0 %)	no
Overall	1.35	115.4	273133	1.35 (±0 %)	114.8 (-0.5 %)	274289 (+0.4 %)	

Table 6.7.: Comparison of the FabScalar's standard ISE and the best obtained one in terms of design-time delay, avg. energy (w/o memory) and area (w/o memory) considering all benchmarks

these stages is very small. Overall, the power consumption (and energy consumption) of the entire microprocessor (w/o memory elements) is slightly better using the aging-aware encoding compared to the standard encoding. However, this is only some positive side-effect and is negligible considering the entire processor. Furthermore, with a different processor using a different implementation, these effects may change. The overall area increases slightly (less than 0.4 %), as an additional mapper translating from the old to the new aging-aware ISE is required. In this regard, it is important to note that, if ISE changes lead to significant changes in the energy consumption of the microprocessor (w/o the memories) due to an increased switching activity, a co-optimization similar to the one proposed in Section 6.2.4, can be employed to co-optimize lifetime and dynamic energy consumption of the affected pipeline stages.

Please note that the considerable MTTF improvements come from the fact that the relation between runtime and delay degradation follows a root-like function. For example, in case of BTI the following relation can be used to estimate the delay degradation [129]:

$$\Delta d(t) \sim \delta^n \cdot t^n, \tag{6.7}$$

where, d is the delay, δ the transistor's duty cycle, t the runtime and n is a technology constant equal to 0.25. Hence, a delta delay reduction from 9.5 %, to 7.6 % corresponds to a duty cycle, which is roughly 2.4x smaller than the original one. As a result, the lifetime improves by this factor. Since the behavior for HCI is very similar, however with $n = 0.5$, the real results shown in Table 6.6 slightly differ from this estimation.

6.3.3. Impact of an Aging-aware ISE on the Memory Power Consumption

Beside the logic, also the processor memories are a major contributor to the overall power/energy consumption. Because of that, as pointed out in Section 6.2.4, we employ a co-optimization to improve the lifetime of the decoding stages and the memory power consumption. In this part, we compare the results of this co-optimization with an optimization that does not consider the switching activity inside the memory blocks. For this purpose, we picked the SA approach.

Due to the way ArISE is implemented in hardware (see Section 6.2.6), the largest instruction-related memory, i.e. the instruction cache, is not affected by modifications of the ISE. Hence, only the energy consumption of the instruction buffer and the pipeline registers changes by employing a modified ISE. Moreover, the default ISE of FabScalar has the lowest average switching activity of all investigated ISEs, and consequently all modified ISEs lead to an increase in the toggle rate and hence also energy consumption compared to the default ISE. This behavior is independent from the chosen optimization strategy.

		Optimization of MTTF only	Co-Optimization of MTTF and TR
All runs (30)	Avg. best† MTTF [years]	**5.6**	5.3
	Avg. best† delay after 3y [ns]	**1.461**	1.463
	Avg. TR increase‡ [%]	5.8	**3.0**
	Avg. No. of ISEs evaluated‡	**37.6**	54.1
	Avg. runtime‡ [min]	**150.4**	216.4
Runs with MTTF > 6 years	Occurrence	**8**	4
	Avg. best† MTTF [years]	**6.4**	6.3
	Avg. best† delay after 3y [ns]	**1.457**	1.458
	Avg. TR increase‡ [%]	6.0	**2.5**
	Avg. No. of ISEs evaluated‡	**26.4**	58
	Avg. runtime‡ [min]	**105.6**	232
Best run	Best MTTF [years]	**7.1**	6.8
	Best delay [ns]	**1.453**	1.454
	TR increase [%]	5.4	**1.5**

Table 6.8.: Comparison of optimization strategies with and without considering the toggle rate (TR) in memories (TR increase w.r.t. default ISE)

Using the proposed co-optimization technique the switching activity of the aging-aware ISE is significantly smaller, without affecting the lifetime improvements too much. As shown in Table 6.8, on average over 30 independent optimization runs, the achieved lifetime is 5.3 years, which is still an improvement of 1.77x compared to the default encoding (3 years lifetime). However, the average increase of the switching activity in the affected memories is only 3 % instead of 5.8 %, if no co-optimization is employed. In case of the ISE with the best lifetime, the improvement is even more significant (1.5 % vs. 5.4 %). This means that for an instruction buffer storing 32 decoded instructions, the dynamic energy consumption of the affected memories increases by only 2 % (1 % for best ISE) and not by 4 %, since the write energy is almost 2x larger than the read energy (33 pJ vs. 17 pJ according to FabMem [217]) and in almost every clock cycle instructions are written to and read from such a memory. This shows the importance of a co-optimization of memory switching activity and lifetime of the decoding stages. However, due to the additional optimization effort, the runtime increases by 44 % on average, since more ISEs have to be evaluated. Nevertheless, the overall runtime is still in an acceptable range (less than 6 hours).

6.3.4. Aging-aware ISE vs. Periodical Inversion

Another technique that tries to alleviate wearout in the pipeline frontend was proposed in [11]. The idea of this approach is to periodically invert the opcode bits, to balance the transistor duty cycles (around 0.5) and by that means to slow down BTI-induced delay degradation. To compare this approach with our aging-aware ISE, we implemented the periodical inversion approach in the Predecode stage of FabScalar. Therefore, all necessary signals and gates have been added to the design.

However, the improvements of this duty-cycle balancing approach in terms of MTTF are much smaller than that of our proposed technique as summarized in Table 6.9. Inverting the ISE every 1000 cycles increases MTTF by just 1 year. Moreover, if the ISE is inverted permanently, the improvements become even slightly better. Hence, our proposed technique significantly outperforms the periodic inversion technique of [11], which is due to two reasons. First, HCI-induced wearout is not considered in [11]. Second, the periodical inversion is intended to balance wearout (duty cycles ≈ 0.5). However, this does not necessarily yield the best MTTF. The reason is that it is often better to reduce wearout of the most critical paths as much as possible (duty cycles $<< 0.5$) at the expense of faster wearout of non-critical paths (duty cycles

	Our Technique with TR constraint	Periodical Inversion [11]		
		never	always	every 10^3cyc
Δ-Delay @ 3y	7.7 %	9.1 %	9.0 %	9.1 %
MTTF	6.8 years	4.0 years	4.1 years	4 years

Table 6.9.: Comparison between our proposed technique and periodical ISE inversion [11] in terms of delay degradation and MTTF (never = ISE is never inverted, always = ISE is always inverted, every 10^3cyc = Inversion period of 10^3 cycles)

$>> 0.5$). Overall this way a much better MTTF can be achieved.

An advantage of the duty cycle balancing approach is that is does not increase the memory switching activity noticeably and hence has no influence on the memory power consumption. In contrast, our proposed co-optimization technique has a small effect on the memory power consumption, as shown in the previous section. However, the impact is almost negligible, in particular if the entire microprocessor is considered. As a result, our co-optimization approach is superior to the periodical inversion scheme.

Please note that the delay degradation with periodical inversion is not directly comparable with the degradation of the standard design, as some additional circuitry is necessary to re-invert the opcode everywhere it is used. Hence, also the overall circuit wearout is slightly different.

6.4. Comparison with Related Work

ISE optimization is a common approach to reduce the dynamic power consumption of instruction buffers and registers by minimizing the input switching activity [212–214]. Therefore, the goal is always to minimize $\sum_{\{(i,j)\}} w_{(i,j)} H_{(i,j)}$, where (i, j) represents a transition from instruction i to instruction j, $H_{(i,j)}$ is the Hamming distance of these two instructions, and $w_{(i,j)}$ stands for the number of transitions from instruction i to instruction j. To solve this NP-hard problem for the instruction opcode bits, in [214] various heuristic solutions including a simulated annealing approach were evaluated, and in [212] binary as well as algebraic design diagrams were employed. In [213] also other components of the instruction field such as the encoding of source and destination registers is optimized using simulated annealing. However, minimizing the overall switching activity of the opcode bits, does not necessarily alleviate transistor aging due to BTI and HCI in the frontend of the instruction pipeline. Instead, the (internal) duty cycles and switching activities of the transistors in the critical paths are much more important and these are not only influenced by the opcode bits, but also by various other input signals as well as the gate-level circuit design. As a result, the aforementioned low-power techniques are not optimal for aging mitigation. For this purpose, other optimization objectives than minimizing the overall switching activity of the opcode bits are required. These are employed by our ArISE philosophy, and thus ArISE is orthogonal to the low-power techniques. In fact, this also means that power can be impacted. For example, we observed a 12 % higher switching activity at the inputs of the instruction buffer (SRAM-based) corresponding to the opcode bits for the aging-aware ISE (see Section 6.2.4). This shows, that although power and wearout are coupled (via temperature), they require different optimization strategies. Hence, both approaches need to be combined to co-optimize power and lifetime. Therefore, we presented enhanced optimization processes that also include the memory switching activity as optimization constraint (see Section 6.3.3).

Mitigation of accelerated transistor aging in decoding stages was first targeted in [11]. The authors proposed to periodically invert the instruction opcode to make the transistor duty cycles close to 0.5, which should mitigate transistor aging due to BTI. However, HCI is not addressed

and the overall MTTF improvements are much lower than those of our proposed approach (see Section 6.3.4). Another technique that also addresses the lifetime of the pipeline frontend was presented in Chapter 4. This scheme improves the lifetime of the entire microprocessor by balancing the delays of all instruction pipeline stages at the desired lifetime. However, if the decoding stages are aging-critical, this technique requires an overdesign (i.e. faster design which means larger and more power hungry design) of these stages to improve their lifetime. In contrast, the approach proposed in this chapter infers only minor overheads in terms of area or power. Nevertheless, it is worth to note that both methodologies, the periodical inversion as well as the pipeline stage balancing, are orthogonal to the ArISE technique. Thus, these can be used in combination to achieve even better lifetime results.

Beside these techniques that directly target the pipeline frontend or the instruction opcode, also all device- to circuit-level techniques as well as system- and application-level schemes presented in Section 2.4 can be employed to alleviate wearout in the decoding stages. In fact, as these are complementary to our proposed approach, these can be also used in combination with ArISE to improve the lifetime further.

6.5. Summary and Conclusion

In this chapter, we proposed a novel cross-layer instruction set encoding methodology to address the delay degradation of the instruction decoding stages in microprocessors, which can become the lifetime limiting components. To derive an optimal mapping between instructions and opcodes with respect to the overall lifetime of the decoding stages, a set of heuristic approaches was presented. In addition, also the memory power consumption is taken into consideration by these heuristics, since pure aging-aware instruction set encodings typically increase the memory power consumption considerably, as shown by our simulation results. As a result, the lifetime of the decoding stages can be significantly improved compared to state-of-the-art solutions with only a small impact on the overall power consumption. At the same time, the performance (i.e. clock frequency) can be maintained, and the implications on the overall die area are negligible.

CHAPTER
SEVEN

AGING-AWARE PROACTIVE DYNAMIC RUNTIME ADAPTATION

The aging mitigation techniques presented in the previous chapters have to be employed during the design phase, and hence their efficiency strongly depends on how good the real workload behavior of the final product can be predicted a priori. Hence, for effective and holistic aging mitigation also runtime solutions based on senses-and-adapt as well as prediction-based methodologies are required to cope with runtime variations. In this chapter, we propose a *dynamic approach based on proactive system adaptation* to alleviate the impact of accelerated transistor aging at runtime. To this end, this approach complements the three techniques presented before. In the first part of the chapter, we explain the problem of static approaches and motivate why dynamic runtime adaptation is a great opportunity to mitigate aging. Afterwards, our proposed technique based on dynamic voltage and frequency scaling (DVFS) is explained, followed by various tests that show the benefits of this scheme. Next, a comparison with other runtime adaptation methods is provided. Finally, a summary and some conclusions are presented.

7.1. Problem Introduction and Motivation

Accelerated transistor aging due to BTI and HCI can be addressed at various abstraction-levels and at design time (i.e. during the design flow) or at runtime (i.e. while the system is operating in the field). While the design time solutions have the advantage that almost every parameter can be tuned to mitigate the effect of accelerated transistor aging, they come along with the shortcoming that these approaches are static and optimized for particular system conditions (e.g. workload, temperature). Hence, these techniques cannot react on sudden condition changes. As a consequence, it can happen that the efficiency of design time solutions is considerably reduced, if the predicted system conditions do not (permanently) hold in the field. Therefore, dynamic runtime adaptation approaches have a great potential to alleviate transistor aging, beside the "classical" design time solutions.

One of the most common runtime adaptation techniques in modern microprocessors and modern operating systems is *Dynamic Voltage and Frequency Scaling* (DVFS), which is employed to "combine" high performance with low power consumption. For this purpose, frequency and supply voltage are reduced to save energy, whenever a DVFS-capable microprocessor is executing only a light workload or no workload at all. Moreover, it is also possible to increase supply voltage and frequency beyond their default values to boost performance for certain phases of application execution, in case power and temperature are not critical [219]. In both cases, supply voltage and frequency can be typically adjusted every millisecond, resulting in a constantly fluctuating frequency in the field, as depicted in Figure 7.1. Famous examples

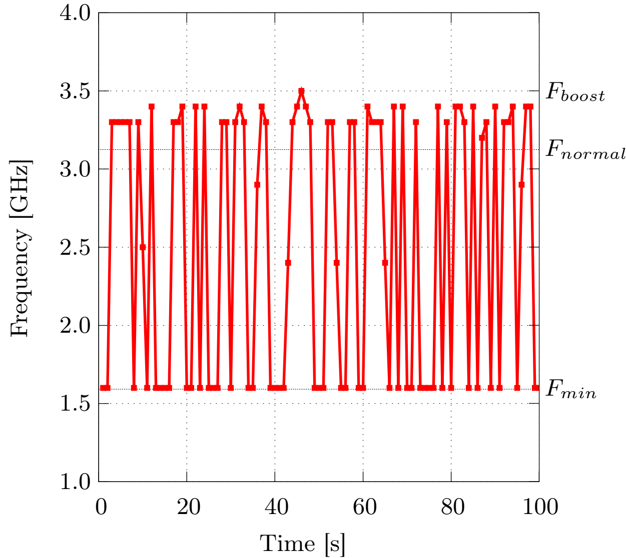


Figure 7.1.: Frequency of an Intel Core i5-3450 while a Youtube video is played back using the DVFS ondemand governor of Linux [9]

Frequency [MHz]	Voltage [V]
F_{min}= 3500	1.10
3400	1.09
3300	1.08
3200	1.08
F_{normal} = 3100	1.07
3000	1.06
2900	1.04
2800	1.02
2700	1.00
2600	0.98
2500	0.96
2400	0.94
2300	0.92
2200	0.91
2100	0.90
2000	0.89
1900	0.89
1800	0.88
1700	0.88
F_{min} = 1600	0.88

Table 7.1.: Intel Core i5-3450 P-States extracted with our experimental platform described in Section 3.3

of microprocessor supporting DVFS are Intel's Core-i processors [85], AMD's Opteron offerings [181], the IBM Power 7+ and Power 8 for high performance servers [188, 189] as well as Qualcomm's low power embedded S4 processor employed in various smartphones [220]. In this regard it is important to note that modern microprocessors support a variety of *P-States* (Frequency-Voltage settings), as shown in Table 7.1 for the case of an Intel Core i5-3450.

Furthermore, DVFS can be also applied to alleviate heat problems [157, 158, 221]. Therefore, the microprocessor temperature is monitored and whenever it becomes critical or near-critical, supply voltage and frequency are reduced to decrease the power consumption and consequently to lower the temperature.

Moreover, since frequency and in particular the supply voltage have a strong influence on the aging rates (see Chapter 2.3), DVFS can be also employed to mitigate the impact of accelerated transistor aging. Therefore, some previous techniques such as [70, 71, 78] applied coarse-grained stepwise voltage scaling policies to compensate the aging-induced delay degradation in a *reactive* manner. In other words, these techniques monitor the delay degradation of the circuits and whenever critical values are reached, these approaches increase the supply voltage step-by-step at runtime to extend the time a certain frequency can be used. By this means, the microprocessor lifetime is improved. However, as a matter of fact, these "damage-control" *sense-and-adapt* kind of approaches are not very efficient [76], since the lifetime is extended at the cost of a considerably increased supply voltage, i.e. increased power consumption. Instead, a *proactive* approach that tries to *prevent* critical system states before they actually occur using *prediction-based* techniques, promises to be more efficient as illustrated in Figure 7.2, i.e. it promises longer MTTF results and lower energy consumption.

Therefore, we propose a novel *proactive, fine-grained and highly dynamic DVFS approach* to extend lifetime and improve power as well as heat in this chapter. The technique is based on an *expert system* that after each time frame determines the voltage-frequency configuration for the next time period based on the current and former system states, the predicted system behavior and user specific constraints. Hence, it combines sense-and-adapt with prediction-based strategies. By this means, supply voltage and frequency are lowered whenever it is possible, to achieve lower aging rates. Moreover, if the prediction indicates that the system is

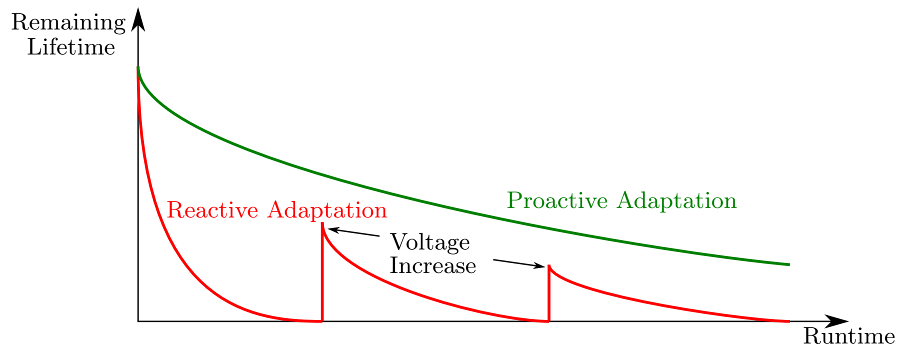


Figure 7.2.: Reactive vs. proactive adaptation philosophy

going to be in a critical state, supply voltage and frequency are reduced as well, to avoid that state before it actually occurs. For this purpose, the expert system continuously monitors the microprocessor states and can adjust the voltage and frequency every millisecond (i.e. very fine-grained and highly dynamic adaption), and thus is feasible for inherently dynamic systems where user/application needs can change in the order of seconds or less.

7.2. Dynamic Runtime Adaption Methodology

In this section, we present our dynamic runtime adaptation methodology. First, the monitoring part is introduced, followed by the presentation of the expert system that is used to determine the next (i.e. new) runtime configuration.

7.2.1. Runtime Monitoring

Many computing systems have tight constraints regarding power, operating temperature and performance. Hence, if dynamic runtime adaptation techniques are used, it is important to monitor these three aspects during runtime and make adaptation decisions dependent on the current state of the system, the history of system states, the predicted system behavior in future and user inputs. Furthermore, for our purpose the wearout status (lifetime) due to transistor aging needs to be monitored as well. Therefore, various sensors available in many modern microprocessors are accessed to gather all this data. The performance information such as IPC (instructions per cycle), activity of execution units, etc. is delivered by special performance counters (special purpose registers). The power consumption as well as the temperature can be obtained from on-die sensors [85, 188, 189, 220], and aging can be measured using special path delay sensors [222]. However, not all microprocessors feature all these sensors. In case some of these sensors are missing, it is also possible to employ software-based models similar to those used in the ExtraTime platform (see our experimental platform in Section 3.3) to extract the necessary information about performance, power, temperature and wearout.

A crucial part for the decision making processes is the granularity at which information is available, i.e. whether for instance only one single power or temperature value is available for an entire microprocessor, or if data for each microarchitectural component is accessible. In this regard it holds that the finer the granularity, the better can be the adaptation. However, at the same time the analysis and decision making effort increases considerably and also there is a rise in implementation costs. Thus, the system designers have to carefully decide which components should be monitored. For example, since our proposed DVFS policy changes the frequency and supply voltage of an entire processor core, the worst component inside the targeted core matters most for the decision making process. Consequently, as the functional units are typically the most aging-critical parts of a microprocessor (see the previous chapters), we limited the wearout analysis to these components in this thesis.

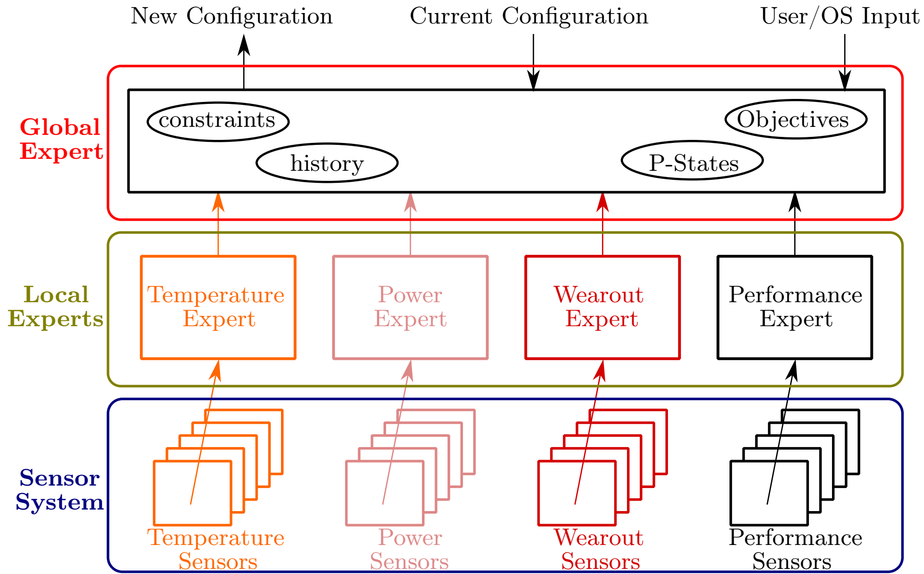


Figure 7.3.: Organization of the expert system for dynamic runtime adaptation

7.2.2. Expert System

All monitored data is sent to an expert system similar to the one depicted in Figure 7.3. It is responsible for making decisions regarding the next runtime configuration and is split into two parts. First, there is one *local* expert for each sensor group. These experts preselect the monitored data, to reduce the data amount that has to be analyzed later on. Nevertheless, the local experts can also initiate an immediate adaptation of the system, for example if critical values, e.g for temperature, are detected. Further details on the preselection process of the local experts and the critical values used in this thesis are given in Table 7.2.

Furthermore, there is one *global* expert used to find the best fitting runtime configuration in case no local expert detects a critical status. The inputs of this expert are the current runtime configuration, the preselected data from the local experts, history of the most recent system states (wearout, temperature, etc.) and various objectives for which the optimal runtime configuration for the next time frame has to be found. The process to find this configuration is explained in detail in the following subsection. Furthermore, the global expert should be able to take input from the user, for example to choose special objectives which fit the needs of the user most. By that means, both self-adaptation and user-controlled adaptation are possible.

7.2.3. Dynamic Voltage and Frequency Scaling Policy

The global expert contains multi-dimensional objectives for runtime adaptation, since simple one-dimensional objectives, such as max(lifetime), max(performance), min(power) or min(temperature) are not the best choice for typical computing systems. For example, neglecting the performance requirements, when maximizing the lifetime would lead to the choice of the lowest possible P-State, which in turn would result in a very low performance. Hence, the global

Local Expert	Preselection Function	Critical If
Temperature	$T_{max} = \max T_i$	$T_{max} \geq 100\,^{\circ}\mathrm{C} = T_{crit}$
Power	$P_{total} = \sum P_i$	$P_{total} \geq 25\,\mathrm{Watt} = P_{crit}$
Performance	no preselection	none
Wearout	$MTTF = \min MTTF_i$	$MTTF \leq 11\,\mathrm{Days} = MTTF_{crit}$

Table 7.2.: Functionality of the local experts

```
While system is running do
  Wait until time = i · 1 ms

  /*** Perform i^th analysis and adaptation step                    ***/
  /*** i.e. X^{i-1} = value of last interval; X^i value for future interval ***/

  /* Evaluation and data aggregation by local experts */
  Gather and preselect all sensor information according to Table 7.2
  If (T^{i-1}_max > T_crit) or (P^{i-1}_max > P_crit) or (MTTF^{i-1} < MTTF_crit)   /* critical state? */
    (F^i, V^i) = Reduce(F^{i-1}, V^{i-1})
  Else
    /** Analysis by global expert is performed **/
    /* Evaluation of system trends */
    Predict system trends according to Equation (7.1)
    If (T^i_trend critical) or (P^i_trend critical) or (MTTF^i_trend critical)   /* critical trend? */
      (F^i, V^i) = Reduce(F^{i-1}, V^{i-1})
    Else
      /* Suggest new (F,V) based on performance */
      Construct {(F^i_sug, V^i_sug)} according to Equation (7.2)

      /* Evaluate all (F,V) for next interval */
      Forall (F,V) do
        Estimate T^i_(F,V), P^i_(F,V) and MTTF^i_(F,V) according to Equation (7.3)
        If (T^i_(F,V) > T_crit) or (P^i_(F,V) > P_crit) or (MTTF^i_(F,V) < MTTF_target)   /* critical next state? */
          {(F^i_sug, V^i_sug)} = {(F^i_sug, V^i_sug)} \ {(F,V)}
        End
      Endfor

      /* Select new (F,V) based on temperature, power and MTTF */
      If ({(F^i_sug, V^i_sug)} ≠ ∅)
        (F^i, V^i) = min {(F^i_sug, V^i_sug)}
      Else
        (F^i, V^i) = max {(F,V) | (T^i_(F,V) < T_crit) ∧ (P^i_(F,V) < P_crit) ∧ (MTTF^i_(F,V) > MTTF_target)}
      End
    End
  End
Done.
```

Algorithm 7.1: Proactive DVFS methodology using trend analysis

expert has to contain complex objectives to optimize the system configuration for a certain goal, but with respect to several constraints. A very important objective for computing devices is to maximize the lifetime, while the required performance is still ensured (as well as power and temperature constraints). In the following we will explain, how this objective can be achieved using fine-grained and proactive DVFS. For this reason, we focus on the ith analysis step, i.e. what analysis and decisions are made between the *DVFS interval* $[t_{i-1}, t_i]$ and the next DVFS interval $[t_i, t_{i+1}]$. The length of such an interval is 1 ms, due to the explanations given in Section 7.3 about the optimal sampling interval (see also Table 7.4).

1. Analyze recent trend of Temperature, Power, Wearout

After the local experts aggregated the sensor data and no critical value was detected, the global expert initiates the decision making process for the P-State for the next DVFS interval (i.e. (F^i, V^i)) as shown in Algorithm 7.1 which describes our proposed DVFS policy. Next, the first step of the global expert is to analyze the history of the last n system states obtained from the last n DVFS intervals (i.e. $n \leq i$). Based on the history and the current state, *linear trend functions* (LTF) are built for wearout (i.e. MTTF), temperature and power. An LTF is basically a linear regression of n data points using the least square fitting method[1] as

[1]Least square fitting means finding a fitting parameter a such that $\sum |f(x,a) - y|$ is minimal for a given fitting function f and a set of values y.

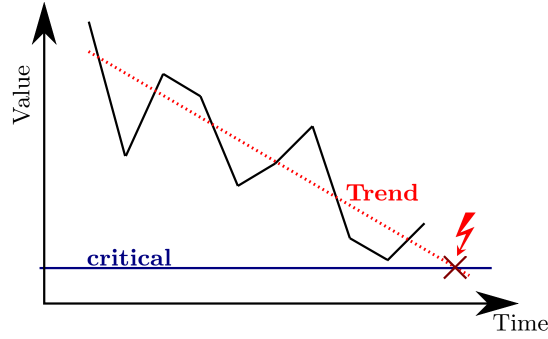


Figure 7.4.: Linear trend function used to predict future (critical) system states

illustrated in Figure 7.4. With the obtained trend functions, the directions (i.e. derivatives) of the trends for power, temperature and MTTF (i.e. towards better or worse system states) are extracted according to Equation (7.1).

$$\begin{aligned} T^i_{trend} &= LTF^i_T(T^{i-1}, \ldots, T^{i-n}), \\ P^i_{trend} &= LTF^i_P(P^{i-1}, \ldots, P^{i-n}), \\ MTTF^i_{trend} &= LTF^i_{MTTF}(MTTF^{i-1}, \ldots, MTTF^{i-n}), \end{aligned} \tag{7.1}$$

If one of these trends X^i_{trend} is considered to be critical, the P-State in the next DVFS interval $[t_i, t_{i+1}]$ will be the next smaller P-State than the one used in the last DVFS interval $[t_{i-1}, t_i]$. In other words, voltage and frequency are scaled one level down to reduce wearout, temperature, or power. For instance, given the P-States in Table 7.1, the frequency is reduced by 100 MHz and the supply voltage is lowered accordingly. In this regard it is important to note that T^i_{trend} (P^i_{trend}) is considered to be critical if T_{crit} (P_{crit}) will be reached within the next 10 DVFS intervals (i.e. 10 ms). Instead, $MTTF^i_{trend}$ is critical, if the trend indicates that the overall lifetime target cannot be achieved. By that means, it is possible in early life phases to detect critical aging trends that would lead to a violation of the lifetime target and adjust frequency and voltage accordingly.

The number of investigated system states n has a huge influence on the extrapolated values and by that means on the decision making process. If n is too small, individual events have too much influence on the trend. If, in contrast, n is too large, outdated system states still affect the trend function. In our case, it has been shown that $n = 10$ is a good compromise between proactive reliability enhancements and performance impact.

2.1. Suggest new P-State based on Performance

If the trend evaluation does not indicate problems, the next step is to find all possible P-States, which guarantee that the performance constraints are fulfilled. Therefore, the global expert accesses various load/performance indicators for the last DVFS interval $[t_{i-1}, t_i]$, such as IPC^{i-1}, number of executed instructions in different execution units (activity $A^{i-1}_{EU_j}$), current frequency F^{i-1}, etc. Based on these parameters all P-States are suggested for a usage in the next DVFS interval, that satisfy the performance constraints, i.e.:

$$\{(F^i_{sug}, V^i_{sug})\} = \left\{(F, V) | F \geq F_{base} = f(\text{IPC}^{i-1}, A^{i-1}_{EU_j}, ..., F^{i-1})\right\} \tag{7.2}$$

where F_{base} is the minimum frequency fulfilling the performance requirements. For each suggested frequency $F^i_{sug,k}$ the supply voltage is set according to the corresponding P-State.

Please note that the function f has not only a huge impact on performance, but also on wearout, temperature and power consumption. The function f reflects the “aggressiveness” with which the frequency/voltage is scaled up or down. From the wearout perspective, a very

aggressive downscaling is desirable, while from the performance point of view an aggressive upscaling is needed. Hence, the function f (and with it the "aggressiveness" of DVFS), which is used to estimate the frequency needed for the next time period, can be used to optimize the DVFS behavior in various ways, i.e. to make the DVFS policy more aging-aware or more performance-aware. Since the global expert is capable of taking inputs from the user or operating system, the function f can be changed during runtime, depending on the current needs.

In this thesis, the function f is always a polynomial. The simplest case is thereby a linear function with the structure $f(P) = a \cdot P + b$, where P is a vector containing the above mentioned load indicators such as IPC^{i-1} or $A^{i-1}_{EU_j}$. The parameters a and b are set in such a way, that $f(P)$ returns the maximum allowed frequency, in case the maximum performance is required and that the minimum allowed frequency is returned, if the minimum performance is sufficient. For example, if P only contains the activity $A \in [0, 1]$ of the first execution unit, the maximum frequency is 3 GHz and the minimum is 1 GHz, f has the following form:

$$f(P) = f(A) = (3GHz - 1GHz) \cdot A + 1GHz.$$

2.2. Select new P-Sate based on Temperature, Power, Wearout

However, the aforementioned way to select the P-State for the next time frame is only one part. While the first step is used to suggest a new P-State according to the performance needs, the second part takes care of the other parameters such as power P, temperature T and wearout $MTTF$. For each P-State of the microprocessor, power, temperature and wearout after the "next" DVFS interval i are estimated, based on the current values and the chosen P-State:

$$\begin{array}{rcll} P^i_{(F,V)} & = & g_P(P^{i-1}, T^{i-1}, F, V) & \forall (F, V), \\ T^i_{(F,V)} & = & g_T(T^{i-1}, P^i, F, V) & \forall (F, V), \\ MTTF^i_{(F,V)} & = & g_{MTTF}(MTTF^{i-1}, T^i, F, V) & \forall (F, V). \end{array} \quad (7.3)$$

The functions g_P, g_T, g_W are basically power, temperature, and wearout models (similar to those employed in our ExtraTime platform) based on various voltage and frequency combinations. Note that the temperature model requires power information, and the wearout model is also dependent on temperature, i.e. first P^i has to be obtained, followed by T^i and then $MTTF^i$ can be predicted. Afterwards, all P-States are removed from $\{(F^i_{sug}, V^i_{sug})\}$, that lead to critical values of P^i, T^i or $MTTF^i$. If the set is empty afterwards, the largest, non-critical pair (F, V) is chosen as next P-State, since its performance will be closest to the one requested by F_{base}. Otherwise, the smallest pair out of the set is chosen, since this P-State will cause the lowest wearout rates of all in the set, while still maintaining the performance target given by F_{base}.

Summing it up, the three steps to determine the P-State for the next time frame allow a proactive lifetime extension (due to trend analysis, the function f, and the prediction in 2.2). Of course, there is a chance that the prediction fails, i.e. a wrong behavior is predicted. This can lead to a P-State lowering (e.g. critical value is predicted based on the trend analysis) or a frequency increase (e.g. prediction indicates that more performance is necessary). However, in both cases, the very fine-grained time steps (i.e. 1 ms) allow almost immediate corrections (i.e. P-State adaptation), in case the real behavior differs from the predicted one. As a result, the performance penalty or lifetime impacts due to mispredictions will be very small, which is underlined by the good results presented in Section 7.3.

Please note that in this dissertation we used linear trend functions, polynomial scaling functions (f) and a fixed one-to-one-mapping between frequency and voltage (similar to the one provided in Table 7.1). However, all approaches can be extended to support nonlinear trend functions, more sophisticated scaling functions and various supply voltages per frequency grade

Processor L1-Cache / L2-Cache	Single-core@3 GHz, out-of-order, 4-issue 64 KB, 3 cyc latency / 2 MB, 15 cyc latency
Expected wearout	MTTF = 3 years
DVFS parameters	DVFS interval = 1 ms, stall latency = 1 μs
SPEC2000 benchmarks	applu, bzip2, equake, gcc, gzip, lucas, mesa, parser, twolf, wupwise
P-States (F-V-States)	1.0 GHz/0.6 V, 1.5 GHz/0.7 V, 2.0 GHz/0.8 V, 2.5 GHz/0.9 V, 3.0 GHz/1.0 V

Table 7.3.: Configuration details for the experiments

depending on the wearout status, respectively. For instance, the DVFS ondemand governor of modern operating systems could replace the scaling function [9]. As a result, the mathematical relations will become more complex, but the overall methodology will still work.

7.2.4. Possible Implementation of the Expert System

In modern systems the DVFS implementation is split into two parts, which can be reused by our proposed expert system. The performance monitors, temperature and power sensors are implemented in hardware as well as the functionality to handle critical temperature or power states. Our local experts can use these functionalities to detect and treat critical system states. In contrast, delay sensors are not as widespread. If they are implemented in hardware as well, they will increase the transistor count and die size. However, these sensors are very small. In case eight of these are build into an ARM Cortex v8 the die size would increase by less than 1% [222]. However, in case the transistor budget is limited the local experts can be also implemented in software as a part of the operating system.

The more sophisticated analysis and decision making parts of current DVFS solutions are embedded into the kernel of the operating system. Our proposed global expert can use these parts. In fact, the available routines need to be extended by the proactive, wearout-aware schemes. A negative performance impact is thereby not to be expected, since the software routines do not use computationally expensive operations.

7.3. Experimental Results

7.3.1. Experimental Setup

In order to evaluate the benefits of the proposed proactive DVFS methodology we employed our ExtraTime framework (see Chapter 3.2). The evaluated 32 nm processor runs at 3 GHz and has one super-scalar, out-of-order core. Further details about the processor configuration can be found in Table 7.3. The following evaluations were based on the execution of 10^9 instructions of various SPEC2000 benchmarks. Thereby the simulations did not include the initialization phase of each benchmark, which was executed but not included in the measurements. Moreover, neither the power consumption nor the temperature were violating the given constraints (see Table 7.2) and also their trends never reached a critical state.

Since gem5 does not support DVFS, we added this feature to our ExtraTime platform. In our implementation a frequency/voltage change leads to a pipeline stall of a few μs, which is typical for modern processors that support DVFS via digital PLLs [223]. Hence, the performance can be (negatively) affected, which makes it unreasonable to adjust voltage or frequency in the order of μs. In our case a 1 ms sampling interval yields the best compromise between dynamics of the system and performance, which is illustrated in Table 7.4. This underlines that only fine-grained techniques can combine long lifetime and high performance.

Please note that the aging models for BTI and HCI in ExtraTime (see Section 3.2.2) are only

Sampling Period [ms]	500	100	10	**1**	0.1	0.01	0.005
Performance [%]	100	99	98	**98**	97	94	91
worst MTTF [%]	100	100	135	**163**	164	164	164
average MTTF [%]	100	225	246	**290**	290	290	290

Table 7.4.: Effect of different sampling periods on performance and lifetime for the used SPEC2000 benchmarks

valid if the parameters such as supply voltage or frequency are constant over time. However, due to the dynamic scaling of voltage and frequency at runtime, these parameters will change. Hence, the V_{th} shift has to be calculated in a stepwise manner, where in each step frequency and voltage are constant. Therefore, every time voltage or frequency changes, the aging rates have to be adjusted according to the new parameters. Since $\Delta V_{th}(t)$ is continuous, the parameter change at time t_1 will not lead to a "jump" in ΔV_{th}, but to a continuous change, i.e:

$$\Delta V_{th}(t_1, T_1, V_{dd,1}, f_1) = A' \cdot \Delta V_{th}(t_1, T_2, V_{dd,2}, f_2)$$

Using this equation A' can be derived, and by that means the V_{th} shift for the following time frame in which the parameters are again constant using the Equations (3.11) and (3.15) can be determined. This process is also illustrated in Figure 7.5 for an example, in which the frequency is changed at time t_1.

7.3.2. Results

We investigated different f-functions (polynomials according to Equation (7.2)) to find out, how aggressive the up/down-scaling of frequency/voltage should be. The results summarized over all executed applications can be found in Figure 7.6. As it can be seen, the difference in terms of performance and aging mitigation of the presented techniques is huge. While the choice of a linear f-function leads to a performance loss of 26 % compared to the non-DVFS case on average (0.86 s average runtime vs. 0.68 s), it can extend the lifetime (MTTF) by more than 200 % (i.e. 3x) in the worst-case (9.1 years vs. 3 years). This is due to the fact that very often voltage and frequency are considerably reduced due to the very aggressive DVFS strategy. In contrast, in case a polynomial of degree 5 is used as f-function (i.e. hexa policy), the downscaling of frequency/voltage is much more conservative and hence the performance impact is minimized (only 2 % on average). However, the lifetime in the worst-case is only extended by 4 %. If, in addition, the proactive trend analysis is enabled, the same policy can achieve a lifetime improvement of 63 % in worst-case and almost 2.9x (8.7 years vs. 3 years) on average, while the performance impact remains almost negligible. Compared to the standard hexa policy the huge lifetime improvements are due to the proactive, small P-State adaptations that avoid critical situations in advance as depicted in Figure 7.7, and hence no strong "emergency" adaptations have to be applied (which reduce performance but do not extend lifetime a lot). In addition,

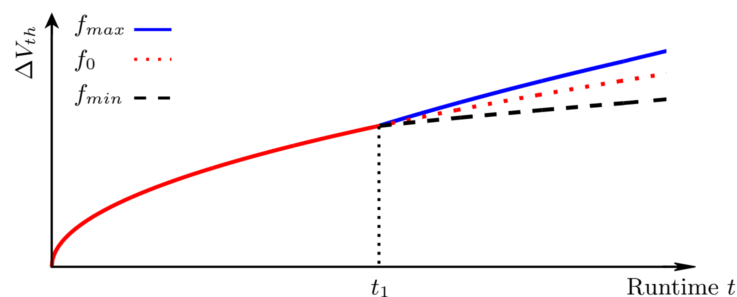


Figure 7.5.: Effect of parameter changes on wearout at time t_1

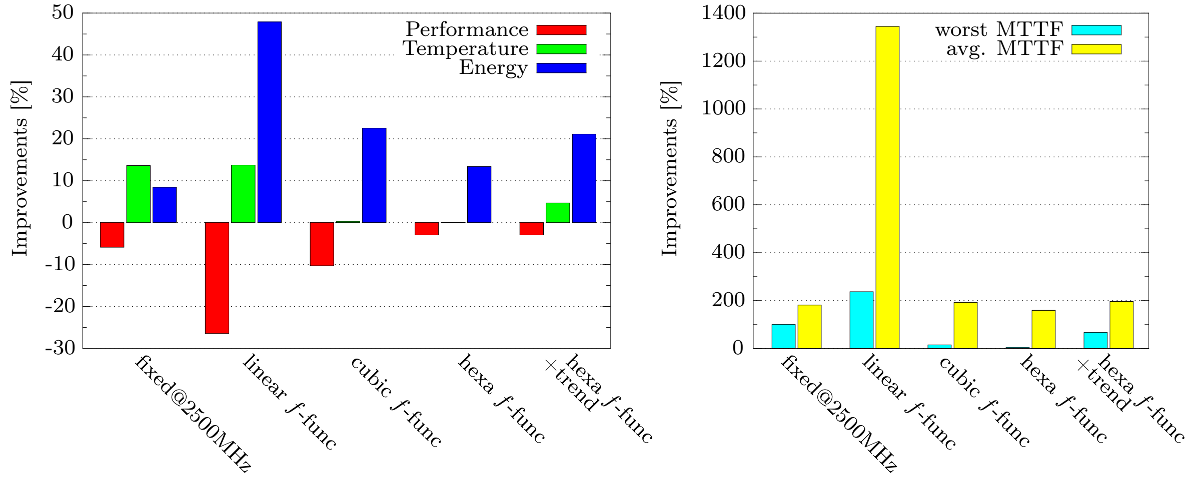


Figure 7.6.: Effect of different DVFS policies on MTTF, energy (avg. over all benchmarks), temperature (max. over all benchmarks) and performance (avg. over all benchmarks) w.r.t. a configuration with no DVFS running at 3 GHz

these small, proactive adaptations also reduce the maximum temperature (89 °C vs. 94 °C) and lower the average energy consumption by 21 % (11.2 Ws vs. 14.2 Ws). In this regard it is important to note that the differences between DVFS without and with the trend analysis clearly show that an aging-aware DVFS is required to efficiently improve the microprocessor lifetime. If instead a power-aware DVFS without aging analysis is employed, the lifetime will improve as well (since from frequency and voltage are also lowered from time to time), however the lifetime improvements will be considerably smaller compared to our proposed proactive aging-aware DVFS, since critical aging trends will not be detected.

Furthermore, we also evaluated a static approach with 2.5 GHz. Also this method can considerably improve the lifetime. However, the average performance loss is 6 % (worst-case: 11 %). This underlines one major disadvantage of static techniques: for some applications they are suitable, for others they are not. “Static DVFS” schemes or DVFS techniques with very long adaptation intervals such as [70, 155], where an application is either executed with 2.5 GHz or

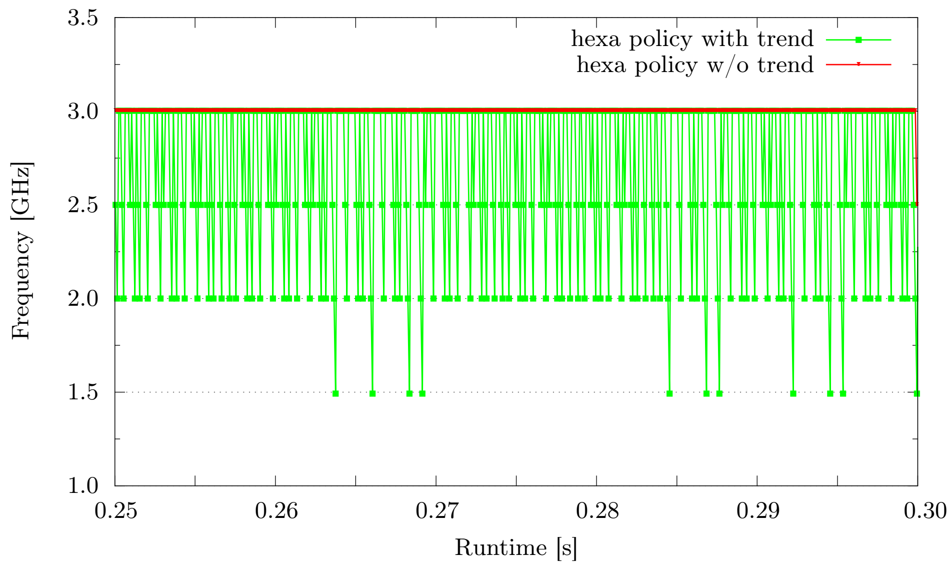


Figure 7.7.: Comparison of a DVFS technique with and without trend analysis for the applu benchmark

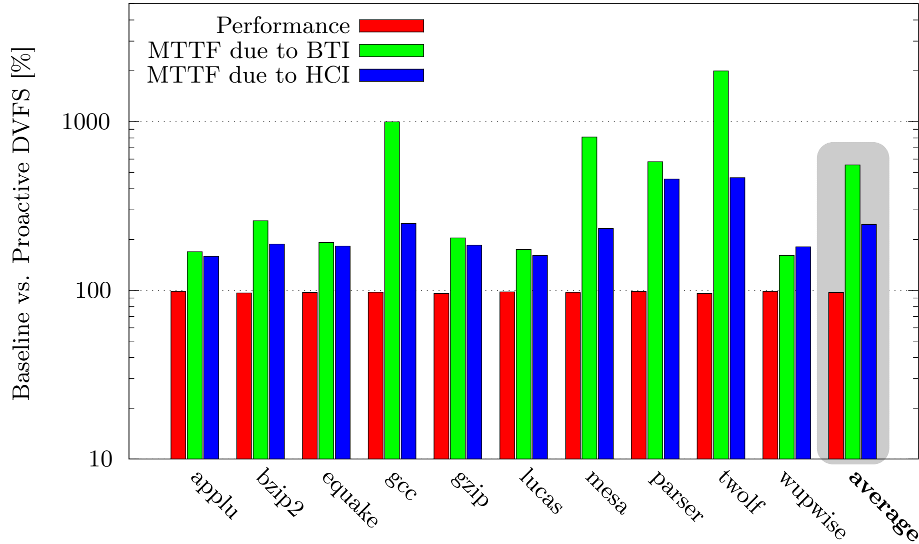


Figure 7.8.: Influence of application on the efficiency of DVFS (hexa f + trend) w.r.t. a configuration with no DVFS running at 3 GHz

3 GHz will minimize this performance penalty, however MTTF will also be considerably lower compared to our proposed fine-grained and proactive DVFS methodology.

Since scaling of voltage/frequency and its efficiency strongly depends on the executed workload, the improvements can vary a lot. In Figure 7.8 the results for several applications for the proactive "hexa policy" are given, in which the MTTF ranges between 4.7 and 14.8 years, i.e. the chosen DVFS policy can extend lifetime by up to 5x. Furthermore, it is important to note that for different applications and different policies the dominance of BTI and HCI on wearout are different. In other words, the efficiency of the proposed DVFS technique for BTI and HCI is different. Hence, neglecting one of these effects can lead to wrong predictions and hence adaptations.

7.4. Comparison with Related Work

As mentioned in the beginning of this chapter, DVFS is a very popular approach to modify a microprocessor configuration at runtime to improve its efficiency. In this regard, the objective can be energy reduction, avoidance of thermal issues, or lifetime improvement.

For the last aspect, some previous work increase the supply voltage only in a stepwise manner, so that frequency and hence performance can be kept on the original level [70, 78]. Also some of these techniques increase frequency in early life (possible due to guardbands) to improve the performance [71]. Still, after a certain operating time, frequency and performance are back on the original level (i.e. specified frequency). In other words, these *static* techniques try to address transistor aging after it has been accumulated beyond a certain level, i.e. in a *reactive* manner. Because of that, these approaches are often less efficient than our proposed proactive scheme [76]. For instance, to improve the lifetime from 3 years to 8.7 years, the supply voltage has to be increased by 4 % to 5 % after 3 years. Thus, the energy consumption increases considerably (between 5 % to 10 % depending on the ratio between leakage and dynamic power) during the second phase of the lifetime and also the temperature will be significantly higher. In comparison, our proactive technique can achieve a MTTF of 8.7 years, while at the same time the energy consumption and temperature throughout the entire lifetime are improved (see Section 7.3). However, an advantage of the reactive approaches is, that the performance

is not affected. Nevertheless, even if the frequency of a microprocessor featuring a reactive aging alleviation DVFS scheme is reduced by 2 % (i.e. this processor has a similar performance compared to one using our proactive DVFS), its energy consumption and temperature behavior will be worse than with our proactive DVFS approach. Still, both the reactive and proactive techniques can be employed together to combine the best of both sides, which means even longer lifetimes with good energy efficiency.

Furthermore, there are also some coarse-grained DVFS techniques that increase/decrease frequency and voltage depending on the system needs to improve the lifetime, such as [70, 155]. However, these use much longer adaptation periods (i.e. in the order of days or more), which can lead to huge performance impacts (up to a factor of f_{max}/f_{min}) or can not yield good aging mitigation results (see Table 7.4), since these techniques cannot immediately react on dynamic events due to changing environmental conditions, performance or power demands. This makes them infeasible for some embedded systems e.g. smartphones, where user/application needs can change in the order of seconds or less.

Please note that the proposed proactive DVFS technique can be also combined with all lower level aging mitigation techniques mentioned in this thesis. As a result, the overall microprocessor lifetime will improve further. Alternatively, some of the lifetime benefits can be used to lower the guardbands and thus to improve the clock frequency (i.e. performance). Also power-aware DVFS schemes can be employed to increase the microprocessor MTTF. However, as pointed out in Section 7.3, the lifetime improvements will be significantly smaller (see for example our DVFS scheme without trend analysis) underlining that for considerable lifetime improvements an aging-aware strategy is required.

7.5. Summary and Conclusion

In this chapter, we provided an innovative cross-layer dynamic runtime adaptation technique to improve the overall microprocessor lifetime. This approach relies on a fine-grained and proactive DVFS policy which is controlled by a multi-level monitoring and decision making system. Compared to state-of-the-art solutions the main novelty of this DVFS scheme is the proactive system state analysis that evaluates not only the current state (i.e. performance, power, temperature and remaining lifetime) but also the trend of recent system states. This enables the decision making system to detect trends that could lead to critical system states in future, and thus to avoid these states before they really occur or at least to alleviate their impact. In other words, the proposed methodology employs prediction-based (e.g. trend analysis) strategies beside the classical sense-and-adapt schemes. As a result of these proactive features, the presented DVFS policy can improve the microprocessor lifetime considerably compared to reactive state-of-the-art strategies, with only a small performance penalty. This, however, can be compensated by reducing the guardbands, which is possible due to the significant lifetime improvements. In addition, also power and temperature benefit from the proposed technique.

Part IV.

Summary, Conclusions and Outlook

CHAPTER

EIGHT

SUMMARY, CONCLUSIONS AND OUTLOOK

8.1. Summary and Conclusions

The continuous downscaling of VLSI technology leads to predictability and reliability challenges, meaning that modern computing systems face a variety of issues such as increasing process variation and design complexity, accelerated wearout as well as higher susceptibility to noise. The result are fragile and vulnerable systems. Moreover, wearout and the higher noise susceptibility lead to continuously changing system conditions in the field. The consequences are very expensive systems due to huge design margins or higher failure rates as well as a considerably reduced operational lifetime [10, 30, 35–39]. Hence, these aspects have to be considered throughout the entire design phase, i.e. reliability as design constraint has to be as important as the traditional parameters performance, power consumption and cost. This need for a reliability-aware design philosophy is stated by both industry and academia, and a lot of effort is spent to develop reliability-aware design solutions, for instance in the scope of the DFG[1] Priority Program SPP 1500 “Dependable Embedded Systems” [82, 224] in Germany or the NSF[2] Variablity Expedition [225] in the USA.

This dissertation tackles the challenge of accelerated transistor aging which can cause timing failures and eventually results in a shorter operational lifetime of microprocessors fabricated in nanoscale technology nodes. For this purpose, a set of unique integrated aging modeling and evaluation frameworks was presented and several novel cross-layer aging mitigation techniques were designed to efficiently extend chip lifetime with only small impacts on performance, power and cost. The contributions of this thesis improve the state-of-the-art by considering lifetime as a design constraint which is equally important as the traditional parameters, and by taking also higher abstraction layers from (micro)architecture to application-level into account, which have often been neglected before. To this end, the interdependencies among performance, power, temperature, cost and lifetime can be analyzed with higher accuracy at higher abstraction levels than with previous approaches, and lifetime can be more effectively co-optimized with the traditional design parameters, e.g. longer lifetime at lower cost.

The set of aging modeling and evaluation frameworks consists of three novel platforms covering the entire abstraction stack from circuit-level up to application-level. The first framework is the architectural platform ExtraTime that models aging, power and temperature at architecture-level, and hence can be employed already in early design phases for an aging-aware design space exploration. Based on the same models, we also developed an experimental framework using real processors that enables us to underline the simulation results with real data.

[1]Deutsche Forschungsgemeinschaft
[2]National Science Foundation

The third platform is an RTL-framework considering circuit- up to application-level information. Thus, it enables an accurate analysis of performance, power, area as well as aging, and consequently it provides very accurate lifetime data.

Using these platforms, four novel cross-layer aging-mitigation techniques addressing different microprocessor components were developed. Among these, there are three design time solutions that target the functional units, the decoding stages of the instruction pipeline and the pipeline design philosophy, respectively. Finally, an aging-aware proactive dynamic system adaptation approach was provided, that tackles wearout at runtime. All of these approaches result in considerable lifetime improvements with only small costs in terms of performance, power or die area. Moreover, since these schemes are complementary to each other, they can be employed together to achieve more efficient design solutions considering the entire microprocessor. However, it is important to note that the overall lifetime improvements will not be the superposition of all isolated improvements, as there are interdependencies among the different approaches. While, for example, the aging-aware instruction scheduling and the aging-aware instruction set encoding techniques are orthogonal to each other, the aging-aware instruction pipeline design scheme is dependent on both of them. Consequently, if all three approaches are applied together, the effectiveness of the pipeline design paradigm may be affected. Thus, as a next step, these interdependencies have to be studied.

Based on the contributions of this thesis several conclusions can be drawn. The first one is that although reliability is not for free, it has not to come with high cost. Nevertheless, reliability has to be carefully traded off with the traditional design parameters. This can be achieved by considering reliability (i.e. lifetime) as design constraint during all design phases instead of only adding (huge) safety margins to the final design. Because of that, the microprocessor characteristics lifetime, performance, power and cost can be more effectively co-optimized. Furthermore, the consideration of the abstraction layers ranging from circuit- to application-level enables the design of very effective and low-cost aging mitigation solutions. This is due to the considerable impact of the higher abstraction layers and the great freedom to optimize various associated knobs in an aging-aware manner. Finally, this thesis demonstrated that whenever the knowledge of various abstraction layers is combined in form of cross-layer approaches (e.g. the timing-aware instruction scheduling or considering application-level influences for the pipeline design), the design can be co-optimized more effectively, i.e. the lifetime can be improved at lower cost compared to traditional non-cross-layer design solutions.

8.2. Outlook

From today's perspective, there is no real end of Moore's law for conventional CMOS technologies in sight, as the microelectronic industry keeps on scaling, despite of all the challenges. For instance, the ITRS roadmap projects that within the next 10 years feature sizes of less than 3 nm can become reality [10], as depicted in Figure 8.1. However, to make this possible, the system design has to be revolutionized from ground up due to the extent of the reliability issues in the upcoming technology nodes (e.g. see Figure 1.3). From device to application-level reliability has to become a first order design constraint to keep failure rates low, and maintain sufficiently long operational lifetimes. In addition, software and hardware have to be designed

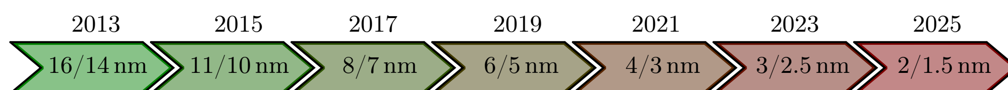


Figure 8.1.: Technology roadmap based on ITRS [10] from 2013 until 2025

to be able to detect and correct errors of different kinds as well to be adaptable to various environmental conditions. Therefore, reliability-aware cross-layer solutions such as those presented in this thesis have to be developed and employed. However, because of the future extent of various reliability issues, the presented approaches have to be extended, to cope also with other reliability challenges. In addition, a variety of different techniques has to be used in combination for effective failure rate reduction and lifetime improvements. In particular, both design time and runtime solutions have to be employed. For this purpose, it is important to study the interdependencies of different approaches, which did not happen, yet. Moreover, also novel toolsets at higher abstraction levels are required that allow the combined analysis of interacting reliability effects.

The aging alleviation techniques presented in this thesis are designed to address primarily BTI- and HCI-induced degradation. However, in terms of wearout also the oxide degradation due to TDDB has to be considered in future, as identified by the ITRS roadmap [10]. For this purpose, our proposed approaches can be extended to cope also with TDDB. In this regard it is important to note that TDDB-induced degradation happens while current is flowing through the transistor channel. Thus, mitigation techniques with focus on BTI also help to mitigate TDDB. Nevertheless, more specialized approaches may become necessary in future.

Another main challenge in future technology nodes is process variation due to increasing manufacturing issues. Hence, design time solutions such as those presented in this thesis have to deal with delay distributions instead of fixed delays for gates, paths and circuits. This, for example, can be achieved by employing Monte-Carlo simulations for the lifetime analysis, or by considering several specific design corners. Nevertheless, as design variability is inevitable, large safety margins have to be employed. To minimize these, dynamic runtime adaptation techniques such as the one provided in this dissertation have to be employed to achieve the best configuration for every microprocessor [226]. In addition, also the aging behavior itself will become more and more stochastic in nature [142, 227]. As a result, different devices suffer from different aging rates even if all conditions such as temperature, supply voltage or workload are the same. Therefore, stochastic device-level models have to be developed and abstracted to higher-levels to allow a thorough study. In this regard it is important to note that the aging mitigation techniques proposed in this work are independent of the underlying aging model, as explained in Section 2.3. Hence, even under stochastic aging behavior the approaches can improve the overall microprocessor lifetime.

BIBLIOGRAPHY

[1] N. Choudhary, S. Wadhavkar, T. Shah, H. Mayukh, J. Gandhi, B. Dwiel, S. Navada, H. Najaf-abadi, and E. Rotenberg, "FabScalar: Automating Superscalar Core Design," *IEEE Micro*, vol. 32, no. 3, pp. 48–59, May 2012.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[3] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[4] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 61–71, Jun. 2004.

[5] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transactions on Architecture and Code Optimization*, vol. 10, no. 1, pp. 5:1–5:29, Apr. 2013.

[6] "OpenSPARC Overview," http://www.oracle.com/technetwork/systems/opensparc/index.html, [Online; accessed Nov. 2014].

[7] H. Nguyen, "Resiliency Challenges in Future Communications Infrastructure," in *Proceedings of the Communications Quality and Reliability Workshop*, May 2014.

[8] S. Taylor, "POWER7+: IBM's Next Generation POWER Microprocessor," in *Hot Chips: A Symposium on High Performance Chips*, Aug. 2012.

[9] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor: Past, Present and Future," in *Proceedings of the Linux Symposium*, pp. 215–230, Aug. 2006.

[10] International Technology Roadmap for Semiconductors, in *ITRS 2013 Edition – Process Integration, Devices, and Structures*, 2014.

[11] E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti, "Combating Aging with the Colt Duty Cycle Equalizer," in *Proceedings of the International Symposium on Microarchitecture*, pp. 103–114, Dec. 2010.

[12] F. Oboril and M. Tahoori, "Aging-Aware Design of Microprocessor Instruction Pipelines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 5, pp. 704–716, May 2014.

[13] F. Oboril and M. Tahoori, "Exploiting Instruction Set Encoding for Aging-Aware Microprocessor Design," *ACM Transactions on Design Automation of Electronic Systems*, pp. 1–23, 2014, submitted for review.

[14] F. Oboril, F. Firouzi, S. Kiamehr, and M. Tahoori, "Negative Bias Temperature Instability-Aware Instruction Scheduling: A Cross-Layer Approach," *Journal of Low Power Electronics*, vol. 9, no. 4, pp. 389–402, Dec. 2013.

[15] F. Oboril and M. Tahoori, "ExtraTime: Modeling and Analysis of Wearout due to Transistor Aging at Microarchitecture-Level," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 1–12, Jun. 2012.

[16] F. Oboril and M. Tahoori, “ExtraTime: A Framework for Exploration of Clock and Power Gating for BTI and HCI Aging Mitigation,” in *Proceedings of Zuverlässigkeit und Entwurf*, pp. 62–69, Sep. 2011.

[17] F. Oboril and M. Tahoori, “Reducing Wearout in Embedded Processors using Proactive Fine-Grained Dynamic Runtime Adaptation,” in *Proceedings of the European Test Symposium*, pp. 68–73, May 2012.

[18] F. Oboril, F. Firouzi, S. Kiamehr, and M. Tahoori, “Reducing NBTI-induced Processor Wearout by Exploiting the Timing Slack of Instructions,” in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 443–452, Oct. 2012.

[19] F. Oboril and M. Tahoori, “MTTF-Balanced Pipeline Design,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 270–275, Mar. 2013.

[20] F. Oboril and M. Tahoori, “ArISE: Aging-Aware Instruction Set Encoding for Lifetime Improvement,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 1–6, Jan. 2014.

[21] F. Oboril, J. Ewert, and M. Tahoori, “High-Resolution Online Power Monitoring for Modern Microprocessors,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1–4, Mar. 2014, to appear.

[22] F. Oboril and M. Tahoori, “ExtraTime: Eine Mikroarchitektur-Simulationsumgebung zur Modellierung, Analyse und Linderung von Alterungseffekten,” *GMM MECHATRONIK 11/2012*, pp. 13–14, Nov. 2012.

[23] F. Oboril, M. Ebrahimi, S. Kiamehr, and M. Tahoori, “Cross-Layer Resilient System Design Flow,” in *Proceedings of the International Symposium on Circuits and Systems*, pp. 1–4, May 2015, to appear.

[24] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach.* Elsevier, 2012, ISBN: 978-0123838728.

[25] J. Shen and M. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors.* Waveland Press, 2013, ISBN: 978-1478607830.

[26] Intel, http://ark.intel.com/, [Online; accessed Nov. 2014].

[27] Intel, http://download.intel.com/pressroom/kits/IntelProcessorHistory.pdf, 2007, [Online; accessed Nov. 2014].

[28] D. Perlmutter, “Sustainability in Silicon and Systems Development,” in *Proceedings of the International Solid-State Circuits Conference*, pp. 31–35, Feb. 2012.

[29] G. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, Apr. 1965.

[30] S. Borkar and A. A. Chien, “The Future of Microprocessors,” *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, May 2011.

[31] T. Mak, “Is CMOS More Reliable with Scaling?” in *Proceedings of the Online Testing Workshop*, Jul. 2002.

[32] S. Guertin and M. White, “CMOS reliability challenges the future of commercial digital electronics and NASA,” in *NEPP Electronic Technology Workshop*, Jun. 2010.

[33] A. Kahng and V. Srinivasan, “Big Chips,” *IEEE Micro*, vol. 31, no. 4, pp. 3–5, Aug. 2011.

[34] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, “Exascale Computing Study: Technology Challenges in Achieving Exascale Systems,” *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 2008.

[35] S. Borkar, “Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation,” *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[36] A. Coskun, T. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici, “Analysis and Optimization of MPSoC Reliability,” *Journal of Low Power Electronics*, vol. 2, no. 1, pp. 56–69, Apr. 2006.

[37] A. Leon, B. Langley, and J. Shin, “The UltraSPARC T1 processor: CMT reliability,” in *Proceedings of the Custom Integrated Circuits Conference*, pp. 555–562, Sep. 2006.

[38] T. Austin, V. Bertacco, S. Mahlke, and Y. Cao, "Reliable Systems on Unreliable Fabrics," *IEEE Transactions on Design & Test of Computers*, vol. 25, no. 4, pp. 322–332, Jul. 2008.

[39] European Nanoelectronics Initiative Advisory Council, *ENIAC Strategic Research Agenda - European Technology Platform Nanoelectronics*, 2nd ed. ENIAC, 2007.

[40] V. Narayanan and Y. Xie, "Reliability Concerns in Embedded System Designs," *Computer*, vol. 39, no. 1, pp. 118–120, Jan. 2006.

[41] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM Journal of Research and Development - Advanced silicon technology*, vol. 50, pp. 433–449, Jul. 2006.

[42] S. Mitra, K. Brelsford, Y. Kim, H. Lee, and Y. Li, "Robust System Design to Overcome CMOS Reliability Challenges," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 1, pp. 30–41, Mar. 2011.

[43] A. W. Strong, E. Y. Wu, R. P. Vollertsen, J. Sune, G. La Rosa, T. D. Sullivan, and S. E. Rauch III, *Reliability Wearout Mechanisms in Advanced CMOS Technologies*. Wiley-IEEE Press, 2009, vol. 12, ISBN: 978-0471731726.

[44] C. Wu, D. Lin, A. Keshavarzi, C. Huang, C. Chan, C. Tseng, C. Chen, C. Hsieh, K. Wong, M. Cheng, T. Li, Y. Lin, L. Yang, C. Lin, C. S. Hou, H. Lin, J. Yang, K. Yu, M. Chen, T. Hsieh, Y. Peng, C. Chou, C. Lee, C. Huang, C. Lu, F. Yang, H. Chen, L. Weng, P. Yen, S. Wang, S. Chang, S. Chuang, T. Gan, T. Wu, T. Lee, W. Huang, Y. Huang, Y. Tseng, C. Wu, E. Ou-Yang, K. Hsu, L. Lin, S. Wang, T. Kwok, C. Su, C. Tsai, M. Huang, H. Lin, A. Chang, S. Liao, L. Chen, J. Chen, P. Lim, X. Yu, S. Ku, Y. Lee, P. Hsieh, P. Wang, Y. Chiu, S. Lin, H. Tao, M. Cao, and Y. Mii, "High Performance 22/20nm FinFET CMOS Devices with Advanced High-K/Metal Gate Scheme," in *Proceedings of the International Electron Devices Meeting*, pp. 27.1.1–27.1.4, Dec. 2010.

[45] C. Auth, C. Allen, A. Blattner, D. Bergstrom, M. Brazier, M. Bost, M. Buehler, V. Chikarmane, T. Ghani, T. Glassman, R. Grover, W. Han, D. Hanken, M. Hattendorf, P. Hentges, R. Heussner, J. Hicks, D. Ingerly, P. Jain, S. Jaloviar, R. James, D. Jones, J. Jopling, S. Joshi, C. Kenyon, H. Liu, R. McFadden, B. McIntyre, J. Neirynck, C. Parker, L. Pipes, I. Post, S. Pradhan, M. Prince, S. Ramey, T. Reynolds, J. Roesler, J. Sandford, J. Seiple, P. Smith, C. Thomas, D. Towner, T. Troeger, C. Weber, P. Yashar, K. Zawadzki, and K. Mistry, "A 22nm High Performance and Low-Power CMOS Technology Featuring Fully-Depleted Tri-Gate Transistors, Self-Aligned Contacts and High Density MIM Capacitors," in *Symposium on VLSI Technology*, pp. 131–132, Jun. 2012.

[46] S. Mittl, A. Swift, E. Wu, D. Ioannou, F. Chen, G. Massey, N. Rahim, M. Hauser, P. Hyde, J. Lukaitis, S. Rauch, S. Saroop, and Y. Wang, "Reliability Characterization of 32nm High-K Metal Gate SOI Technology with Embedded DRAM," in *Proceedings of the International Reliability Physics Symposium*, pp. 6A.5.1–6A.5.7, Apr. 2012.

[47] A. E. Islam, S. Mahapatra, S. Deora, V. D. Maheta, and M. A. Alam, "Essential aspects of negative bias temperature instability (nbti)," *ECS Transactions*, vol. 35, no. 4, pp. 145–174, May 2011.

[48] S. Ogawa and N. Shiono, "Generalized diffusion-reaction model for the low-field charge-buildup instability at the Si-SiO_2 interface," *Physical Review B*, vol. 51, no. 7, pp. 4218–4230, Feb. 1995.

[49] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao, "The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 173–183, Feb. 2010.

[50] S. Pae, M. Agostinelli, M. Brazier, R. Chau, G. Dewey, T. Ghani, M. Hattendorf, J. Hicks, J. Kavalieros, K. Kuhn, M. Kuhn, J. Maiz, M. Metz, K. Mistry, C. Prasad, S. Ramey, A. Roskowski, J. Sandford, C. Thomas, J. Thomas, C. Wiegand, and J. Wiedemer, "BTI Reliability of 45 nm High-K + Metal-Gate Process Technology," in *Proceedings of the International Reliability Physics Symposium*, pp. 352–357, May 2008.

[51] E. Takeda, Y. Nakagome, H. Kume, and S. Asai, "New hot-carrier injection and device degradation in submicron MOSFETs," *IEEE Proceedings I, Solid-State and Electron Devices*, vol. 130, no. 3, pp. 144–150, Jun. 1983.

[52] K.-L. Chen, S. Saller, and R. Shah, "The case of AC stress in the hot-carrier effect," *IEEE Transactions on Electron Devices*, vol. 33, no. 3, pp. 424–426, Mar. 1986.

[53] X. Li, J. Qin, and J. B. Bernstein, "Compact Modeling of MOSFET Wearout Mechanisms for Circuit-Reliability Simulation," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 1, pp. 98–121, Mar. 2008.

[54] A. Bravaix, C. Guerin, V. Huard, D. Roy, J. Roux, and E. Vincent, "Hot-Carrier Acceleration Factors for Low Power Management in DC-AC stressed 40nm NMOS node at High Temperature," in *Proceedings of the International Reliability Physics Symposium*, pp. 531–548, Apr. 2009.

[55] T. Nigam, B. Parameshwaran, and G. Krause, "Accurate product lifetime predictions based on device-level measurements," in *Proceedings of the International Reliability Physics Symposium*, pp. 634–639, Apr. 2009.

[56] K. Kang, S. P. Park, K. Roy, and M. A. Alam, "Estimation of Statistical Variation in Temporal NBTI Degradation and its Impact on Lifetime Circuit Performance," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 730–734, Nov. 2007.

[57] A. Amouri, F. Bruguier, S. Kiamehr, P. Benoit, L. Torres, and M. Tahoori, "Aging effects in FPGAs: an experimental analysis," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 1–4, Sep. 2014.

[58] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit Failure Prediction and Its Application to Transistor Aging," in *Proceedings of the VLSI Test Symposium*, pp. 277–286, May 2007.

[59] X. Yang and K. Saluja, "Combating NBTI Degradation via Gate Sizing," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 47–52, 2007.

[60] B. C. Paul, K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy, "Temporal Performance Degradation under NBTI: Estimation and Design for Improved Reliability of Nanoscale Circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 780–785, Mar. 2006.

[61] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and Minimization of PMOS NBTI effect for Robust Nanometer Design," in *Proceedings of the Design Automation Conference*, pp. 1047–1052, Jun. 2006.

[62] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-aware Synthesis of Digital Circuits," in *Proceedings of the Design Automation Conference*, pp. 370–375, Jun. 2007.

[63] M. Ebrahimi, F. Oboril, and M. Tahoori, "Aging-aware Logic Synthesis," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–8, Nov. 2013.

[64] A. Calimera, E. Macii, and M. Poncino, "NBTI-Aware Power Gating for Concurrent Leakage and Aging Optimization," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 127–132, Aug. 2009.

[65] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang, "On the efficiancy of Input Vector Control to mitigate NBTI effects and leakage power," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 19–26, Mar. 2009.

[66] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie, "Temperature-aware NBTI modeling and the impact of input vector control on performance degradation," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1–6, Apr. 2007.

[67] D. R. Bild, G. E. Bok, and R. P. Dick, "Minimization of NBTI Performance Degradation using Internal Node Control," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 148–153, Mar. 2009.

[68] S. Das, C. Tokunaga, S. Pant, W. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.

[69] J. Keane, T. Kim, X. Wang, and C. H. Kim, "On-chip Reliability Monitors for Measuring Circuit Degradation," *Microelectronics Reliability*, vol. 50, no. 8, pp. 1039–1053, Aug. 2010.

[70] M. Basoglu, M. Orshansky, and M. Erez, "NBTI-Aware DVFS: A New Approach to Saving Energy and Increasing Processor Lifetime," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 253–258, Aug. 2010.

[71] O. Khan and S. Kundu, "A Self-Adaptive System Architecture to Address Transistor Aging," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 81–86, Mar. 2009.

[72] L. Zhang and R. Dick, "Scheduled Voltage Scaling for Increasing Lifetime in the Presence of NBTI," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 492–497, Jan. 2009.

[73] K. Choi, R. Soma, and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 4–9, Mar. 2004.

[74] G. Dhiman and T. Rosing, "System-Level Power Management Using Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 676–689, May 2009.

[75] S. Mitra, "Circuit failure prediction for robust system design in scaled CMOS," in *Proceedings of the International Reliability Physics Symposium*, pp. 524–531, 2008.

[76] T. Chan, J. Sartori, P. Gupta, and R. Kumar, "On the Efficacy of NBTI Mitigation Techniques," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1–6, Mar. 2011.

[77] T. Siddiqua and S. Gurumurthi, "A Multi-Level Approach to Reduce the Impact of NBTI on Processor Functional Units," in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 67–72, May 2010.

[78] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proceedings of the International Symposium on Microarchitecture*, pp. 129–140, Nov. 2008.

[79] A. DeHon, H. Quinn, and N. Carter, "Vision for Cross-Layer Optimization to Address the Dual Challenges of Energy and Reliability," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1017–1022, Mar. 2010.

[80] B. Zandian and M. Annavaram, "Cross-layer Resilience Using Wearout Aware Design Flow," in *Proceedings of the International Conference on Dependable Systems Networks*, pp. 279–290, Jun. 2011.

[81] G. Georgakos, U. Schlichtmann, R. Schneider, and S. Chakraborty, "Reliability Challenges for Electric Vehicles: From Devices to Architecture and Systems Software," in *Proceedings of the Design Automation Conference*, pp. 98:1–98:9, Jun. 2013.

[82] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Hartig, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. Teich, N. When, and H. Wunderlich, "Design and architectures for dependable embedded systems," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis*, pp. 69–78, Oct. 2011.

[83] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. Rosing, M. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and Opportunistic Computing in Presence of Hardware Variability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, Jan. 2013.

[84] R. E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar. 1999.

[85] Intel, *Desktop 3rd Gen Intel Core Processor Family: Datasheet*, 2012.

[86] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Pearson Education India, 2005, ISBN: 978-0321547743.

[87] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl, "A physical alpha-power law MOSFET model," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 218–222, 1999.

[88] H. Im, M. Song, T. Hiramoto, and T. Sakurai, "Physical Insight into Fractional Power Dependence of Saturation Current on Gate Voltage in Advanced Short Channel MOSFETS (Alpha-Power Law Model)," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 13–18, Aug. 2002.

[89] J. Laprie, "Dependability: Basic Concepts and Terminology," in *Dependability: Basic Concepts and Terminology Dependable Computing and Fault-Tolerant Systems* , vol. 5, pp. 3–245, 1992, ISBN 978-3-7091-9172-9.

[90] S. Mitra, S. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Proceedings of the Design Automation Conference*, pp. 12–17, Jun. 2010.

[91] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," in *Proceedings of the Design Automation Conference*, pp. 7–12, Sep. 2006.

[92] S. Yerramilli, "On the Need for Convergence Between Design Validation and Test," in *Proceedings of the International Test Conference*, pp. 14–14, Oct. 2006.

[93] Intel, *Desktop 3rd Gen Intel Core Processor Family: Specification Update*, 2014.

[94] Intel, *60- and 66-MHz Pentium Processor Specification Update*, 1997.

[95] AMD, *Revision Guide for AMD Family 10h Processors*, 2012.

[96] Intel, *Intel Xeon Processor E3-1200 v3 Product Family*, 2014.

[97] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar, "Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 184–193, Jan. 2011.

[98] R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Transactions on Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005.

[99] M. Nicolaidis, Ed., *Soft Errors in Modern Electronic Systems*. Springer, 2011, ISBN: 978-1441969927.

[100] J. Autran, P. Roche, S. Sauze, G. Gasiot, D. Munteanu, P. Loaiza, M. Zampaolo, and J. Borel, "Altitude and underground real-time SER characterization of CMOS 65nm SRAM," in *European Conference on Radiation and Its Effects on Components and Systems*, pp. 519–524, Sep. 2008.

[101] A. Dixit and A. Wood, "The Impact of New Technology on Soft Error Rates," in *Proceedings of the International Reliability Physics Symposium*, pp. 5B.4.1–5B.4.7, Apr. 2011.

[102] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[103] S. Nassif and S. Hassoun, "Guest Editors' Introduction: On-Chip Power Distribution Networks," *IEEE Transactions on Design & Test of Computers*, vol. 20, no. 3, pp. 5–6, Jun. 2003.

[104] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974.

[105] J. Keane and C. Kim, "An odomoeter for CPUs," *IEEE Spectrum*, vol. 48, no. 5, pp. 28–33, May 2011.

[106] Y. Miura and Y. Matukura, "Investigation of Silicon-Silicon Dioxide Interface Using MOS Structure," *Japanese Journal of Applied Physics*, vol. 5, no. 2, pp. 180–180, 1966.

[107] C. Hu, S. Tam, F. Hsu, P. Ko, T. Chan, and K. Terrill, "Hot-electron-induced MOSFET degradation – Model, monitor, and improvement," *IEEE Transactions on Electron Devices*, vol. 32, no. 2, pp. 375–385, Feb. 1985.

[108] K. Jeppson and C. Svensson, "Negative Bias Stress of MOS devices at high electric field and degradation of MOS devices," *Journal of Applied Physics*, vol. 48, no. 5, pp. 2004–2014, May 1977.

[109] C. Nunes, P. F. Butzen, A. I. Reis, and R. P. Ribas, "BTI, HCI and TDDB aging impact in flip-flops," *Microelectronics Reliability*, vol. 53, no. 9-11, pp. 1355–1359, Nov. 2013.

[110] H. Luo, X. Chen, J. Velamala, Y. Wang, Y. Cao, V. Chandra, Y. Ma, and H. Yang, "Circuit-level delay modeling considering both TDDB and NBTI," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 1–8, Mar. 2011.

[111] *Bias Temperature Instability in HKMG MOSFETs: Characterization, Process Dependence, DC/AC Modeling and Stochastic Effects*, Mar. 2014, Tutorial.

[112] J. Zhang and W. Eccleston, "A comparative study of positive and negative bias temperature instabilities in MOSFETs," in *Proceedings of the International Electron Devices and Materials Symposium*, pp. 14–17, Jul. 1994.

[113] M. Ershov, S. Saxena, H. Karbasi, S. Winters, S. Minehane, J. Babcock, R. Lindley, P. Clifton, M. Redford, and A. Shibkov, "Dynamic recovery of negative bias temperature instability in p-type metal-oxide-semiconductor field-effect transistors," *Applied Physics Letters*, vol. 83, no. 8, pp. 1647–1649, Aug. 2003.

[114] H. Reisinger, O. Blank, W. Heinrigs, A. Muhlhoff, W. Gustin, and C. Schlunder, "Analysis of NBTI Degradation- and Recovery-Behavior Based on Ultra Fast VT-Measurements," in *Proceedings of the International Reliability Physics Symposium*, pp. 448–453, Mar. 2006.

[115] T. Grasser, B. Kaczer, W. Goes, H. Reisinger, T. Aichinger, P. Hehenberger, P. J. Wagner, F. Schanovsky, J. Franco, M. Luque, and M. Nelhiebel, "The Paradigm Shift in Understanding the Bias Temperature Instability: From Reaction-Diffusion to Switching Oxide Traps," *IEEE Transactions on Electron Devices*, vol. 58, no. 11, pp. 3652–3666, Nov. 2011.

[116] J. Stathis and S. Zafar, "The negative bias temperature instability in MOS devices: A review," *Microelectronics Reliability*, vol. 46, no. 2-4, pp. 270–286, Aug. 2006.

[117] A. Edwards, "Interaction of H and H_2 with the silicon dangling orbital at the <111> Si/SiO_2 interface," *Physical Revision B*, vol. 44, pp. 1832–1838, Jul. 1991.

[118] C. Helms and E. Poindexter, "The silicon-silicon dioxide system: Its microstructure and imperfections," *Reports on Progress in Physics*, vol. 57, no. 8, pp. 791–852, 1994.

[119] D. Schroder, "Negative Bias Temperature Instability (NBTI): Physics, Materials, Process, and Circuit Issues," in *Proceedings of the International Reliability Physics Symposium*, Mar. 2005, Tutorial.

[120] A. Singh, *Electronic Devices and Integrated Circuits.* PHI Learning Pvt. Ltd., 2011, ISBN: 978-8120344716.

[121] A. Rockett, *The Materials Science of Semiconductors.* Springer, 2007, 978-0387256535.

[122] G. Chen, K. Chuah, M. Li, D. Chan, C. Ang, J. Zheng, Y. Jin, and D. Kwong, "Dynamic NBTI of PMOS Transistors and Its Impact on Device Lifetime," in *Proceedings of the International Reliability Physics Symposium*, pp. 196–202, Mar. 2003.

[123] R. Fernandez, B. Kaczer, A. Nackaerts, S. Demuynck, R. Rodriguez, M. Nafria, and G. Groeseneken, "AC NBTI studied in the 1 Hz – 2 GHz range on dedicated on-chip CMOS circuits," in *Proceedings of the International Electron Devices Meeting*, pp. 1–4, Dec. 2006.

[124] S. Pae, A. Ashok, J. Choi, T. Ghani, J. He, S. Lee, K. Lemay, M. Liu, R. Lu, P. Packan, C. Parker, R. Purser, A. Amour, and B. Woolery, "Reliability Characterization of 32nm High-K and Metal-Gate Logic Transistor Technology," in *Proceedings of the International Reliability Physics Symposium*, pp. 287–292, May 2010.

[125] S. Mahapatra, "Electrical Characterization and Modeling of Negative Bias Temperature Instability in p-MOSFET Devices," in *Proceedings of the International Reliability Physics Symposium*, Mar. 2006.

[126] 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," *IEEE Transactions on Device and Materials Reliability*, vol. 7, no. 4, pp. 509–517, Dec. 2007.

[127] M. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71–81, Jan. 2005.

[128] A. Fick, "´Uber Diffusion," *Annalen der Physik*, vol. 170, no. 1, pp. 59–86, 1855.

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

[130] P. Li, G. Stamoulis, and I. Hajj, "A Probabilistic Timing Approach to Hot-Carrier Effect Estimation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 10, pp. 1223–1234, Oct. 1994.

[131] K. Lee, C. Kang, O. Yoo, R. Choi, B. Lee, J. Lee, H. Lee, and Y. Jeong, "PBTI-Associated High-Temperature Hot Carrier Degradation of nMOSFETs With Metal-Gate/High-k Dielectrics," *IEEE Electron Device Letters*, vol. 29, no. 4, pp. 389–391, Apr. 2008.

[132] W. McMahon, A. Haggag, and K. Hess, "Reliability Scaling Issues for Nanoscale Devices," *IEEE Transactions on Nanotechnology*, vol. 2, no. 1, pp. 33–38, Mar. 2003.

[133] E. Amat, T. Kauerauf, R. Degraeve, R. Rodriguez, M. Nafria, X. Aymerich, and G. Groeseneken, "Channel hot-carrier degradation in pMOS and nMOS short channel transistors with high-k dielectric stack," *Microelectronic Engineering*, vol. 87, no. 1, pp. 47–50, 2010.

[134] C. Schlünder, "Device reliability challenges for modern semiconductor circuit design–a review," *Advances in Radio Science*, vol. 7, no. 15, pp. 201–211, 2009.

[135] J. Hicks, D. Bergstrom, M. Hattendorf, J. Jopling, J. Maiz, S. Pae, C. Prasad, and J. Wiedemer, "45nm Transistor Reliability," *Intel Technology Journal*, vol. 12, no. 2, pp. 131–144, Jun. 2008.

[136] T. Yamamoto, K. Uwasawa, and T. Mogami, "Bias temperature instability in scaled p+ polysilicon gate p-MOSFET's," *IEEE Transactions on Electron Devices*, vol. 46, no. 5, pp. 921–926, May 1999.

[137] H. Luo, Y. Wang, K. He, R. Luo, H. Yang, and Y. Xie, "Modeling of PMOS NBTI Effect Considering Temperature Variation," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 139–144, Mar. 2007.

[138] S. Kumar, C. Kim, and S. Sapatnekar, "An analytical model for negative bias temperature instability," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 493–496, Nov. 2006.

[139] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging analysis of circuit timing considering NBTI and HCI," in *Proceedings of the International On-Line Testing Symposium*, pp. 3–8, Jun. 2009.

[140] V. Huard, C. Parthasarathy, N. Rallet, C. Guerin, M. Mammase, D. Barge, and C. Ouvrard, "New characterization and modeling approach for NBTI degradation from transistor to product level," in *Proceedings of the International Electron Devices Meeting*, pp. 797–800, Dec. 2007.

[141] H. Reisinger, T. Grasser, W. Gustin, and C. Schlunder, "The statistical analysis of individual defects constituting NBTI and its implications for modeling DC- and AC-stress," in *Proceedings of the International Reliability Physics Symposium*, pp. 7–15, May 2010.

[142] H. Kükner, P. Weckx, P. Raghavan, B. Kaczer, C. F., L. Van der Perre, R. Lauwereins, and G. Groeseneken, "Impact of duty factor, stress stimuli, gate and drive strength on gate delay degradation with an atomistic trap-based BTI model," *Microprocessors and Microsystems*, vol. 37, no. 8, Part A, pp. 792–800, Nov. 2013.

[143] S. Kiamehr, F. Firouzi, and M. B. Tahoori, "Input and Transistor Reordering for NBTI and HCI Reduction in Complex CMOS Gates," in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 201–206, May 2012.

[144] H. Amrouch, V. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards Interdependencies of Aging Mechanisms," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 478–485, Nov. 2014.

[145] F. Firouzi, S. Kiamehr, and M. B. Tahoori, "NBTI Mitigation by NOP Assignment and Insertion," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 218–223, Mar. 2012.

[146] M. DeBole, R. Krishnan, V. Balakrishnan, W. Wang, H. Luo, Y. Wang, Y. Xie, Y. Cao, and N. Vijaykrishnan, "New-Age: A Negative Bias Temperature Instability-Estimation Framework for Microarchitectural Components," *International Journal of Parallel Programming*, vol. 37, pp. 417–431, Aug. 2009.

[147] N. Hatami, R. Baranowski, P. Prinetto, and H. Wunderlich, "Multilevel Simulation of Nonfunctional Properties by Piecewise Evaluation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 4, pp. 37:1–37:21, Aug. 2014.

[148] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Lifetime Reliability: Toward an Architectural Solution," *IEEE Micro*, vol. 25, pp. 70–80, May 2005.

[149] S. Feng, S. Gupta, and S. Mahlke, "Olay: Combat the Signs of Aging with Introspective Reliability Management," *Ann Arbor*, vol. 1001, p. 48109, Jun. 2008.

[150] JEDEC Solid State Technology Association and others, "Failure Mechanisms and Models for Semiconductor Devices," *JEDEC Publication JEP122-B*, 2003.

[151] D. Schroder and J. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, p. 1, 2003.

[152] J. Keane, X. Wang, D. Persaud, and C. Kim, "An All-In-One Silicon Odometer for Separately Monitoring HCI, BTI, and TDDB," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 817–829, Apr. 2010.

[153] 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*, pp. 210–218, Mar. 2006.

[154] A. Gebregiorgis, M. Ebrahimi, S. Kiamehr, F. Oboril, S. Hamdioui, and M. Tahoori, "Aging Mitigation in Memory Arrays Using Self-controlled Bit-flipping Technique," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 1–6, Jan. 2015.

[155] E. Mintarno, J. Skaf, R. Zheng, J. Velamala, Y. Cao, S. Boyd, R. Dutton, and S. Mitra, "Self-Tuning for Maximized Lifetime Energy-Efficiency in the Presence of Circuit Aging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 760–773, May 2011.

[156] A. Marongiu, A. Acquaviva, and L. Benini, "OpenMP Support for NBTI-Induced Aging Tolerance in MPSoCs," in *Stabilization, Safety, and Security of Distributed Systems*, ser. Lecture Notes in Computer Science, R. Guerraoui and F. Petit, Eds., vol. 5873, pp. 547–562, Nov. 2009.

[157] J. Lee, K. Skadron, and S. Chung, "Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers*, vol. 59, pp. 127–133, Jan. 2010.

[158] M. Bao, A. Andrei, P. Eles, and Z. Peng, "On-line Thermal Aware Dynamic Voltage Scaling for Energy Optimization with Frequency/Temperature Dependency Consideration," in *Proceedings of the Design Automation Conference*, pp. 490–495, Jun. 2009.

[159] T. Ebi, H. Amrouch, and J. Henkel, "COOL: Control-based Optimization of Load-balancing for Thermal Behavior," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis*, pp. 255–264, Oct. 2012.

[160] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, "mDTM: Multi-Objective Dynamic Thermal Management for On-Chip Systems ," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1–6, Mar. 2014.

[161] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Reliability Modeling and Management in Dynamic Microprocessor-based Systems," in *Proceedings of the Design Automation Conference*, pp. 1057–1060, Jun. 2006.

[162] R. Mahajan, C. Chiu, and G. Chrysler, "Cooling a Microprocessor Chip," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1476–1486, Aug. 2006.

[163] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluation*, 3rd ed. A. K. Peters Ltd, 1998, ISBN: 978-1568810928.

[164] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *Proceedings of the International Symposium on Microarchitecture*, pp. 7–19, Dec. 2003.

[165] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. Kim, and K. Flautner, "Razor: Circuit-Level Correction of Timing Errors for Low-Power Operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov. 2004.

[166] T. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," in *Proceedings of the International Symposium on Microarchitecture*, pp. 196–207, Nov. 1999.

[167] A. Meixner, M. E. Bauer, and D. Sorin, "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," in *Proceedings of the International Symposium on Microarchitecture*, pp. 210–222, Dec. 2007.

[168] J. Tschanz, K. Bowman, S.-L. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De, "A 45nm Resilient and Adaptive Microprocessor Core for Dynamic Variation Tolerance," in *Proceedings of the International Solid-State Circuits Conference*, pp. 282–283, Feb. 2010.

[169] K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Transactions on Computers*, vol. 33, pp. 518–528, Jun. 1984.

[170] A. Roy-Chowdhury, N. Bellas, and P. Banerjee, "Algorithm-Based Error-Detection Schemes for Iterative Solution of Partial Differential Equations," *IEEE Transactions on Computers*, vol. 45, pp. 394–407, Apr. 1996.

[171] D. Boley, G. H. Golub, S. Makar, N. Saxena, and E. J. McCluskey, "Floating Point Fault Tolerance with Backward Error Assertions," *IEEE Transactions on Computers*, vol. 44, pp. 302–311, Feb. 1995.

[172] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault Tolerant High Performance Computing by a Coding Approach," in *Proceedings of the Symposium on Principles and Practice of Parallel Programming*, pp. 213–223, Feb. 2005.

[173] F. Oboril, M. Tahoori, V. Heuveline, D. Lukarski, and J.-P. Weiss, "Numerical Defect Correction as an Algorithm-Based Fault Tolerance Technique for Iterative Solvers," in *Proceedings of the Pacific Rim International Symposium on Dependable Computing*, pp. 144–153, Dec. 2011.

[174] F. Oboril, I. Sagar, and M. Tahoori, "A-SOFT-AES: Self-Adaptive Software-Implemented Fault-Tolerance for AES," in *Proceedings of the International On-Line Testing Symposium*, pp. 104–109, Jul. 2013.

[175] W. Liao, L. He, and K. Lepak, "Temperature and Supply Voltage Aware Performance and Power Modeling at Microarchitecture Level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1042–1053, Jul. 2005.

[176] M. Pedram and S. Nazarian, "Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1487–1501, Aug. 2006.

[177] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. Drake, L. Pesantez, T. Gloekler, J. Tierno, P. Bose, and A. Buyuktosunoglu, "Introducing the Adaptive Energy Management Features of the Power7 Chip," *IEEE Micro*, vol. 31, no. 2, pp. 60–75, Mar. 2011.

[178] Anandtech, http://www.anandtech.com/show/5771/the-intel-ivy-bridge-core-i7-3770k-review, Apr. 2012, [Online; accessed Nov. 2014].

[179] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1, 2A, 2B, 2C, 3A, 3B, and 3C*, May 2012.

[180] J. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. Snoeren, and R. Gupta, "Evaluating the effectiveness of model-based power characterization," in *USENIX Annual Technical Conference*, pp. 1–14, Jun. 2011.

[181] R. Jotwani, S. Sundaram, S. Kosonocky, A. Schaefer, V. Andrade, G. Constant, A. Novak, and S. Naffziger, "An x86-64 core implemented in 32nm SOI CMOS," in *Proceedings of the International Solid-State Circuits Conference*, pp. 106–107, Feb. 2010.

[182] P. Gschwandtner, M. Knobloch, B. Mohr, D. Pleiter, and T. Fahringer, "Modeling CPU Energy Consumption of HPC Applications on the IBM POWER7," in *Proceedings of the International Conference on Parallel, Distributed and Network-Based Processing*, pp. 536–543, Feb. 2014.

[183] C. Isci and M. Martonosi, “Runtime power monitoring in high-end processors: methodology and empirical data,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 93–104, Dec. 2003.

[184] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, “Online Power and Performance Estimation for Dynamic Power Management,” *IBM, RC-24007, Tech. Rep*, 2006.

[185] K. Singh, M. Bhadauria, and S. McKee, “Real Time Power Estimation and Thread Scheduling via Performance Counters,” *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, Jul. 2009.

[186] Y. Sun, L. Wanner, and M. Srivastava, “Low-cost Estimation of Sub-system Power,” in *Proceedings of the Green Computing Conference*, pp. 1–10, Jun. 2012.

[187] S. Wang, H. Chen, and W. Shi, “SPAN: A software power analyzer for multicore computer systems,” *Sustainable Computing: Informatics and Systems*, vol. 1, no. 1, pp. 23–34, Mar. 2011.

[188] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. Rubio, F. Rawson, and J. Carter, “Architecting for Power Management: The IBM POWER7 Approach,” in *Proceedings of the International Symposium on High Performance Computer Architecture*, pp. 1–11, Jan. 2010.

[189] J. Friedrich, H. Le, W. Starke, J. Stuechli, B. Sinharoy, E. Fluhr, D. Dreps, V. Zyuban, G. Still, C. Gonzalez, D. Hogenmiller, F. Malgioglio, R. Nett, R. Puri, P. Restle, D. Shan, Z. Deniz, D. Wendel, M. Ziegler, and D. Victor, “The POWER8 processor: Designed for big data, analytics, and cloud environments,” in *Proceedings of the International Conference on IC Design & Technology*, pp. 1–4, May 2014.

[190] B. Krishnamurthy and I. G. Tollis, “Improved Techniques for Estimating Signal Probabilities,” *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1041–1045, 1989.

[191] J. Chen, S. Wang, and M. Tehranipoor, “Critical-reliability path identification and delay analysis,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 2, pp. 12:1–12:21, Mar. 2014.

[192] W. Wang, Z. Wei, S. Yang, and Y. Cao, “An Efficient Method to Identify Critical Gates under Circuit Aging,” in *Proceedings of the International Conference on Computer-Aided Design*, pp. 735–740, Nov. 2007.

[193] N. Hatami, R. Baranowski, P. Prinetto, and H. Wunderlich, “Efficient System-Level Aging Prediction,” in *Proceedings of the European Test Symposium*, pp. 1–6, May 2012.

[194] O. Coudert, “Gate Sizing for Constrained Delay/Power/Area Optimization,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 465–472, Dec. 1997.

[195] T. Karnik, Y. Ye, J. Tschanz, L. Wei, S. Burns, V. Govindarajulu, V. De, and S. Borkar, “Total Power Optimization by Simultaneous Dual-Vt Allocation and Device Sizing in High Performance Microprocessors,” in *Proceedings of the Design Automation Conference*, pp. 486–491, Jun. 2002.

[196] International Technology Roadmap for Semiconductors, in *ITRS 2013 Edition – Executive Summary*, 2014.

[197] J. Chen, S. Wang, and M. Tehranipoor, “Efficient Selection and Analysis of Critical-Reliability Paths and Gates,” in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 45–50, May 2012.

[198] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, “MiBench: a free, commercially representative embedded benchmark suite,” in *Workshop on Workload Characterization*, pp. 3–14, Dec. 2001.

[199] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-Aware Microarchitecture,” *SIGARCH Computer Architecture News*, vol. 31, no. 2, pp. 2–13, May 2003.

[200] E. Mintarno, V. Chandra, D. Pietromonaco, R. Aitken, and R. Dutton, “Workload dependent NBTI and PBTI analysis for a sub-45nm commercial microprocessor,” in *Proceedings of the International Reliability Physics Symposium*, pp. 3A.1.1–3A.1.6, Apr. 2013.

[201] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: a 32-way multithreaded Sparc processor,” *IEEE Micro*, vol. 25, no. 2, pp. 21–29, Jun. 2005.

[202] Y. Kunitake, T. Sato, and H. Yasuura, "Signal Probability Control for Relieving NBTI in SRAM Cells," in *Proceedings of the International Symposium on Quality Electronic Design*, pp. 660–666, Mar. 2010.

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

[204] A. Tiwari, S. R. Sarangi, and J. Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation," *SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 323–334, Jun. 2007.

[205] X. Liang, G. Wei, and D. Brooks, "ReVIVal: A Variation-Tolerant Architecture Using Voltage Interpolation and Variable Latency," *IEEE Micro*, vol. 29, no. 1, pp. 127–138, Jan. 2009.

[206] J. Sartori, B. Ahrens, and R. Kumar, "Power Balanced Pipelines," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 1–12, Feb. 2012.

[207] S. Kiamehr, F. Firouzi, M. Ebrahimi, and M. Tahoori, "Aging-aware Standard Cell Library Design," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1–4, Mar. 2014.

[208] Realworldtech, "Intel's Haswell CPU Microarchitecture," Realworldtech: http://www.realworldtech.com/haswell-cpu/, Nov. 2012, [Online; accessed Nov. 2014].

[209] S. Kim, S. V. Kosonocky, and D. R. Knebel, "Understanding and Minimizing Ground Bounce During Mode Transition of Power Gating Structure," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 22–25, Aug. 2003.

[210] K. Usami, T. Shirai, T. Hashida, H. Masuda, S. Takeda, M. Nakata, N. Seki, H. Amano, M. Namiki, M. Imai, M. Kondo, and H. Nakamura, "Design and Implementation of Fine-Grain Power Gating with Ground Bounce Suppression," in *Proceedings of the International Conference on VLSI Design*, pp. 381–386, Jan. 2009.

[211] D. Bild, R. Dick, and G. Bok, "Static NBTI Reduction Using Internal Node Control," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 4, pp. 45:1–45:30, Oct. 2012.

[212] L. Benini, G. de Micheli, A. Macii, E. Macii, and M. Poncino, "Reducing Power Consumption of Dedicated Processors Through Instruction Set Encoding," in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 8–12, Feb. 1998.

[213] A. Chattopadhyay, D. Zhang, D. Kammler, and E. M. Witte, "Power-efficient Instruction Encoding Optimization for Embedded Processors," in *Proceedings of the International Conference on VLSI Design*, pp. 595–600, Jan. 2007.

[214] S. Kim and J. Kim, "Opcode Encoding for Low-Power Instruction Fetch," *Electronics Letters*, vol. 35, no. 13, pp. 1064–1065, Jun. 1999.

[215] X. Yang, *Nature-Inspired Metaheuristic Algorithms: Second Edition.* Luniver press, 2010, ISBN: 978-1905986286.

[216] D. Simon, *Evolutionary Optimization Algorithms.* John Wiley & Sons, 2013, ISBN: 978-0470937419.

[217] T. Shah, "Fabmem: A multiported ram and cam compiler for superscalar design space exploration." Ph.D. dissertation, NC State University, 2010.

[218] K. O. W. Group *et al.*, *The OpenCL Specification*, 2008.

[219] Intel, *Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors*, Nov. 2008.

[220] Qualcomm Inc., *Snapdragon S4 Processors: System on Chip Solutions for a New Mobile Age – White Paper*, Oct. 2011.

[221] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 721–725, Nov. 2002.

[222] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala, "A Distributed Critical-Path Timing Monitor for a 65nm High-Performance Microprocessor," in *Proceedings of the International Solid-State Circuits Conference*, pp. 398–399, Feb. 2007.

[223] A. Bashir, J. Li, K. Ivatury, N. Khan, N. Gala, N. Familia, and Z. Mohammed, "Fast Lock Scheme for Phase-Locked Loops," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 319–322, Sep. 2009.

[224] "DFG Priority Program SPP 1500 "Dependable Embedded Systems"," http://spp1500.itec.kit.edu/, [Online; accessed Nov. 2014].

[225] "NSF Variability Expedition," http://www.variability.org/, [Online; accessed Nov. 2014].

[226] S. Nassif, V. Kleeberger, and U. Schlichtmann, "Goldilocks failures: Not too soft, not too hard," in *Proceedings of the International Reliability Physics Symposium*, pp. 2F.1.1–2F.1.5, Apr. 2012.

[227] B. Kaczer, S. Mahato, V. de Almeida Camargo, M. Toledano-Luque, P. Roussel, T. Grasser, F. Catthoor, P. Dobrovolny, P. Zuber, G. Wirth, and G. Groeseneken, "Atomistic approach to variability of bias-temperature instability in circuit simulations," in *Proceedings of the International Reliability Physics Symposium*, pp. XT.3.1–XT.3.5, Apr. 2011.