Improving Reliability and Security with Aging and Pre-RTL Modeling

---

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Engineering)

by

Alec Roelke

May 2018

APPROVAL SHEET

This Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author Signature: Alec Roelke

This Dissertation has been read and approved by the examining committee:

Advisor: Mircea R. Stan

Committee Member: John Lach

Committee Member: Kevin Skadron

Committee Member: Samira Khan

Committee Member: Schuyler Eldridge

Committee Member: _____

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2018

Abstract

With the increasing importance of cloud computing, where low-power devices offload power-hungry computations to remote servers, the reliability of these servers becomes more important. At the same time, the emergence of the Internet of Things (IoT) has introduced a need for long-lasting electronics in devices with long lifetimes. Both types of systems are susceptible to aging: the slow degradation of circuit parameters that eventually leads to failure. As a result, architects need tools to evaluate the effectiveness of techniques for improving reliability, but post-RTL simulation is slow. In this work, I present a pre-RTL tool called OldSpot which enables optimization of aging resilience using high-level models that improve simulation speed by reducing unnecessary detail while decreasing accuracy loss. Existing aging models make assumptions about aging rates that do not hold in a system whose operational parameters change over time. OldSpot uses directed graphs to indicate how the failures of units within the system contribute to the failure of the whole to create a lifetime distribution, removing these assumptions. This enables analysis of architectural techniques like structural duplication to improve lifetime.

OldSpot can be included in a pre-RTL tool flow that includes power, performance, and temperature simulations to create a high-level characterization of all design metrics. To enable its use in this flow, I also present an implementation of the RISC-V ISA in the gem5 simulator, a high-level microarchitecture and memory modeling tool widely used for pre-RTL performance simulation. The flow is demonstrated by simulating a heterogeneous system containing a RISC-V CPU and an accelerator to show the importance of co-designing the two units rather than designing them separately.

Another limitation on the lifetime of IoT devices is their security, which can be ensured using a compact, low-power device called a Physical Unclonable Function (PUF). PUFs use natural silicon variations to create fingerprints. Despite PUFs' power and area advantages, they are also susceptible to aging, which affects variations and modifies their fingerprints. In this work, I show how directed aging can degrade the reliability of a PUF or even duplicate its fingerprint. I also demonstrate a method of resisting this degradation using active recovery.

To my family, who supported me the entire way and never once asked me when I would finish.

Acknowledgments

First I would like to thank my advisor, Mircea Stan, for his guidance on the work I did to create this dissertation. When my first project was clearly not working out, he suggested I start learning about NBTI and how to model it. I was drawn in by the challenge of creating a compact model for a complex mechanism. This led to a project in developing a model for NBTI recovery and eventually an architecture-level simulator that became OldSpot. It also led to investigation into the effects of aging on security and how NBTI could be used to degrade or even clone the fingerprint of an SRAM PUF. In the meantime, work had begun on a low-voltage RISC-V chip and we needed a way to quickly explore design space before developing RTL, which complemented my project on developing a pre-RTL lifetime model. Gem5 was a perfect tool for the job due to its maturity and wide use in the architecture community, except that it didn't support RISC-V. Excited by the chance to create a major contribution to the community, I began learning about gem5, RISC-V, and how to get RISC-V programs running in gem5. After about half a year of work, I not only finished rudimentary support but I got it included with the main release, have inspired others to help with the cause, and will continue to support and develop it into the future. Mircea's advice (and patience) throughout these projects was invaluable and I am incredibly thankful for his support.

Next I would like to thank the members of the gem5 community for helping me implement RISC-V in gem5 and making me its official maintainer. They helped me understand almost every aspect of gem5's ISA specification, with debugging, and with cleaning up the code so it was readable and could be merged into the rest of gem5's code base. In particular they gave me pointers as to how to properly interface with gem5's existing code. This project gave me a lot of insight into writing code (especially C++) that I could apply to later projects like OldSpot. I would also like to specially thank Tuan Ta, who is as of this writing developing major improvements to gem5's implementation of RISC-V that have impacts across the entire tool.

I would also like to thank Xinfei Guo, another of Mircea's students, whose work on NBTI aging and recovery inspired some of my work on PUF aging and on our NBTI model. His familiarity with device-level aging models was also invaluable in helping to develop OldSpot and getting the data we needed to demonstrate its capabilities.

Finally, I would like to thank the rest of the HPLP research group and the members of the VELOUR development team for their support. They provided me valuable insights that inspired me and helped push along my research.

I would also like to thank the UVA Department of Electrical and Computer Engineering for providing funding for my studies during my first year through the Charles L. Brown Fellowship and the National Science Foundation for providing subsequent funding through the Graduate Research Fellowship Program.

Contents

Contents **vi**
List of Tables . . . viii
List of Figures . . . ix
List of Algorithms . . . xi
List of Abbreviations . . . xii
List of Tool Names . . . xiv
List of Symbols . . . xv

1 Introduction **1**
1.1 Aging . . . 2
1.1.1 Aging Mechanisms . . . 2
1.1.2 Managing Aging . . . 6
1.2 Importance of High-level Modeling . . . 8
1.3 Hypothesis and Contributions . . . 9

2 Reliability Modeling **11**
2.1 High-level Reliability Modeling . . . 11
2.2 The OldSpot Framework . . . 14
2.2.1 System Specification . . . 14
2.2.2 Reliability Modeling and Simulation . . . 16
2.2.3 Assumptions . . . 17
2.3 Finding Reliability "Hot Spots" with OldSpot . . . 17
2.3.1 Tool Flow . . . 18
2.3.2 Accuracy Validation . . . 19
2.3.3 Finding Reliability Hot Spots . . . 22
2.4 Related Work . . . 23
2.5 Benefits of OldSpot . . . 25

3 Pre-RTL Simulation with High-level Models **26**
3.1 Tool Flow . . . 27
3.1.1 The gem5 Simulator . . . 28
3.1.2 Estimating Power and Area with McPAT . . . 29
3.1.3 Simulating Temperature with HotSpot . . . 30
3.1.4 Computing Voltage Droop with VoltSpot . . . 32
3.2 RISC5: An Implementation of the RISC-V ISA in gem5 . . . 33
3.2.1 Implementation Details . . . 35
3.2.2 Validation of RISC5 . . . 40
3.3 Co-optimizing CPUs and Accelerators with Constraints . . . 41
3.3.1 High-level Accelerator Modeling Tools . . . 42
3.3.2 Modeling Infrastructure and Tool Flow . . . 43
3.3.3 Co-optimizing Accelerators with CPUs . . . 45
3.4 Summary . . . 48

4 Manipulating PUF Reliability with Directed NBTI **50**
4.1 The Physical Unclonable Function . . . 51
4.1.1 Types of PUFs . . . 51
4.1.2 Security with PUFs . . . 52
4.2 The SRAM PUF . . . 54
4.3 Aging in SRAM PUFs . . . 56
4.4 Manipulating SRAM PUF Reliability . . . 58
4.4.1 Affecting a Fingerprint with NBTI . . . 58
4.4.2 Attacking an SRAM PUF with NBTI . . . 61
4.4.3 Restoring Reliability via Active Recovery . . . 64
4.4.4 Cloning a Fingerprint with NBTI . . . 69
4.5 Related Work . . . 72
4.6 Applications of Directed NBTI to PUF Reliability . . . 75

5 Conclusion **77**
5.1 Summary of Contributions . . . 78
5.2 Future Work . . . 79
5.3 Final Remarks . . . 79

Appendix A List of Publications **81**

Appendix B Overall Aging Rate Computation **82**

Appendix C Gem5-Aladdin Results **83**

Bibliography **87**

List of Tables

2.1 Simulated System Parameters 18

3.1 Example Simulation Flow Design Parameters 29
3.2 McPAT Results Summary 30
3.3 Simulation Features and Compatibility 34
3.4 Invalid Division Operations 36
3.5 Simulated System Parameters 46
3.6 Simulation Sweep Parameters 46

4.1 AS6C6264 Specification 59
4.2 Experiments 1 and 2 Wearout Conditions 59
4.3 IDT71256 Specifications 65
4.4 Recovery Experiment Stress Patterns 65
4.5 Clone Prediction Equation Parameters 71

List of Figures

1.1 Diagrams of the two mechanisms behind NBTI. . . . 3
1.2 Illustration of the physical mechanism behind electromigration and SEM photograph of a wire that has been affected. . . . 4
1.3 Diagram of the mechanism behind HCI. . . . 5
1.4 Illustration of aging rate over time. . . . 6

2.1 Illustration of changes to reliability caused by a long-term change in workload. . . . 13
2.2 Example failure dependency graph for a system with a single core and an accelerator that include several functional units. . . . 14
2.3 Simulation tool flow diagram used to evaluate reliability hot spots. . . . 18
2.4 Temperature and aging maps. . . . 19
2.5 Weibull rate parameter of each unit for each aging mechanism modeled by OldSpot. . . . 20
2.6 Accuracy validation of OldSpot. . . . 21
2.7 Error when using CALIPER to estimate the lifetime of a four-core x86 system with shared L3 cache. . . . 21
2.8 Area overhead of extending the lifetime of a three-core system. . . . 22

3.1 Illustration of a tool flow for simulating an application running on a chip. . . . 27
3.2 Illustration of HotSpot's compact thermal model. . . . 31
3.3 Heat maps of the temperatures of Rocket Chip and BOOM. . . . 32
3.4 VoltSpot modeling mechanism for voltage noise. . . . 33
3.5 Illustration of the tradeoff between simulation accuracy and speed. . . . 35
3.6 Illustration of the execution of an atomic memory operation in gem5. . . . 37
3.7 New predecoder state machine used by gem5 with the compressed instruction set. . . . 40
3.8 Validation of gem5 performance statistics against the same values from Chisel simulation and FPGA . . . 41
3.9 Illustration of the tool flow used to simulate the power, performance, and area of a single-core RISC-V system with one accelerator. . . . 44

4.1 Diagram of a fuzzy extractor. . . . 52
4.2 Circuit layout and noise margin diagrams for a 6T SRAM cell. . . . 53
4.3 Illustration of the skew of a 6T SRAM cell. . . . 54
4.4 PUF static noise margin diagram. . . . 55
4.5 Example fingerprint for an SRAM array. . . . 56
4.6 Change in skew over time for a 65nm process due to NBTI for different mismatches in size. . 57
4.7 Fraction of strong ones and strong zeroes. . . . 59
4.8 Accumulation of bit errors and rate of successful reconstruction for each chip's fingerprint as it ages. . . . 63
4.9 Average number of errors from the fresh fingerprint and success rate at reconstructing it for each chip as it ages and recovers. . . . 66
4.10 Effect of NBTI degradation and active recovery on each chip's uniqueness. . . . 67
4.11 Prediction of success rate for one SRAM's impersonation of another SRAM's fingerprint. . . . 70
4.12 Device vulnerability to NBTI for process sizes ranging from 32 nm to 350 nm . . . 72

C.1 Pareto-optimal frontiers for *rbm* area and performance. 83
C.2 Pareto-optimal frontiers for *pca* area and performance. 84
C.3 Pareto-optimal frontiers for *lda* area and performance. 84
C.4 Pareto-optimal frontiers for *rbm* power and performance. 85
C.5 Pareto-optimal frontiers for *pca* power and performance. 85
C.6 Pareto-optimal frontiers for *lda* area and performance. 86

List of Algorithms

2.1 OldSpot Monte Carlo iteration 15
4.1 SRAM PUF fingerprint recovery reliability 61
4.2 Reed-Muller code order 61

List of Abbreviations

ALU	**A**rithmetic and **L**ogic **U**nit
CPU	**C**entral **P**rocessing **U**nit
DAG	**D**irected **A**cyclic **G**raph
DDDG	**D**ynamic **D**ata **D**ependence **G**raph
DVFS	**D**ynamic **V**oltage and **F**requency **S**caling
EM	**E**lectro**m**igration
FIT	**F**ailures **i**n **T**ime
FPGA	**F**ield-**p**rogrammable **G**ate **A**rray
FPU	**F**loating **P**oint **U**nit
FS	**F**ull **S**ystem
HCI	**H**ot-**c**arrier **I**njection
IC	**I**ntegrated **C**ircuit
IoT	**I**nternet **o**f **T**hings
ISA	**I**nstruction **S**et **A**rchitecture
LR	**L**oad-**r**eserved
MTTF	**M**ean **T**ime **t**o **F**ailure
NBTI	**N**egative **B**ias **T**emperature **I**nstability
PBTI	**P**ositive **B**ias **T**emperature **I**nstability
PDN	**P**ower **D**elivery **N**etwork
RMW	**R**ead-**m**odify-**w**rite
PSNM	**P**UF **S**tatic **N**oise **M**argin
PUF	**P**hysical **U**nclonable **F**unction
RTL	**R**egister **T**ransfer **L**evel
SC	**S**tore-**c**onditional

SE	**S**ystem-call **E**mulation
SoC	**S**ystem **o**n **C**hip
SOFR	**S**um **o**f **F**ailure **R**ates
SRAM	**S**tatic **R**andom **A**ccess **M**emory
TDDB	**T**ime-**d**ependent **D**ielectric **B**reakdown

List of Tool Names

Aladdin	[52]	Pre-RTL power, performance, and area simulator for accelerators
ArchFP	[70]	Pre-RTL floorlplan prototyping tool
CACTI-P	[79]	Cache and memory access time, cycle time, area, and power model.
CALIPER	[40]	Lifetime simulator for multicore, failure-tolerant systems
Chisel	[82]	Hardware construction language based on Scala
gem5	[53]	High-level, detailed performance simulator with support for many ISAs
gem5-Aladdin	[93]	Heterogeneous system simulator combining gem5 and Aladdin
HotSpot	[11]	Pre-RTL, compact thermal modeling tool
McPAT	[54]	Multicore power, area, and timing simulator
OldSpot	[58]	High-level lifetime and reliability simulator
RAMP	[30]	Reliability-aware microprocessor high-level simulation tool
RISC5	[72]	Implementation of the RISC-V ISA in gem5
SimPoint	[73]	Workload phase analysis tool for finding representative regions
Sniper	[67]	Accurate, high-speed multicore x86 simulator
spike	[84]	RISC-V ISA simulator
VoltSpot	[10]	Pre-RTL PDN and voltage noise modeling tool

List of Symbols

t	Time
T	Temperature
V_{DD}	Supply voltage
I	Current
P	Power
V_T	Transistor threshold voltage
V_{T0}	Initial transistor threshold voltage (before aging)
$R(t)$	Probability that a system will function at time t
η	Weibull rate parameter
β	Weibull shape parameter
p	SRAM bit cell skew

Chapter 1

Introduction

The recent growth in usage of mobile devices has caused a paradigm shift in the way computations are performed. Rather than performing large computations with high-end hardware, users use low-power personal devices that send data to remote servers such as Microsoft's Windows Azure [1, 2] that compute the results and send them back to the client. For example, a virtual personal assistant in a cell phone or laptop would make use of such services to convert a user's speech into commands that are sent back to the device for execution. Additionally, this also opens a path for embedding low-power microprocessors into devices such as household appliances. This way, they can communicate with each other and their owners to exchange data about usage and preferences and receive commands, creating an "Internet of Things" (IoT).

This, along with the impending end of Moore's Law [3], has caused significant changes in both hardware and software design methodologies. Whereas historically it has been possible to rely on operational parameters, such as supply voltage, to shrink along with transistor size according to the principal of Dennard scaling [4] to enable power reduction and performance improvement, it is no longer possible to do so [5]. The resulting increase in power density has repercussions for devices all along the power spectrum. At the high end, in devices such as servers, it increases operating temperatures and introduces a concept known as "dark silicon," [5] where sections of a chip must be powered off to keep the system within power and thermal budgets. It also directly increases the cost of power delivery and cooling. At the low end, in IoT or mobile devices, increased power consumption can mean lower battery lifetime. This has caused an increased emphasis on design techniques such as low-voltage operation [6].

High power densities and temperatures also have adverse effects on reliability. Reliability problems can be classified into two groups: soft errors and hard errors [6]. Soft errors are caused by radiation, such as neutrons from cosmic sources or alpha particles from trace elements in packaging. If these particles strike a

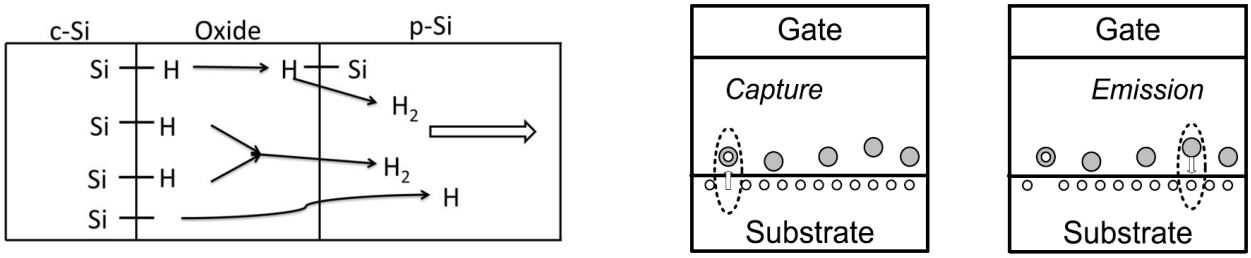
circuit, they can cause bits to flip, resulting in incorrect behavior. Even though soft errors are not affected by temperature, they have been shown to become worse as process sizes continue to shrink [7]. Techniques for handling soft errors exist from the circuit level [8] up to the software level [9]. Other kinds of transient reliability problems include voltage noise [10] and temperature variation [11], which cause timing errors through voltage droop and reduced transistor speed and are affected by variation in workload characteristics. These problems are generally aggravated by high power densities and, as a result, also become worse as process sizes shrink. In order to manage them, tools such as HotSpot [11] and VoltSpot [10] can be used to minimize them at design time while techniques such as dynamic voltage and frequency scaling (DVFS) can be used to manage them at runtime. In addition to transient errors, hard errors caused by manufacturing defects and aging also occur. Unlike transient errors, hard errors are permanent in nature and are often tied to processes that evolve over time. Aging, the focus of this work, is one of the major causes of hard errors.

1.1 Aging

Aging is the *slow degradation of device parameters over time.* This degradation causes slowdown and other failures, reducing performance and causing system failures through circuit timing violations and physical damage. In high-end devices such as large-scale servers, this can mean increased latency for a client that can cause failure to meet their requirements [12]. When that happens, the system must be replaced; this has been estimated to cost up to 45% of the total cost of ownership of these systems [13]. It also has implications for low-end devices such as embedded systems. Because such devices can be permanent installations like household appliances and are often required to last many years, the electronics inside them must last at least as long [14]. IoT and cloud computing have also caused a change in the utilization of electronic devices from sporadic to near-constant, accelerating aging [15] and reducing lifetime. The following sections will discuss aging mechanisms, their impacts on circuits and devices, and current techniques for mitigating them.

1.1.1 Aging Mechanisms

There are many mechanisms by which aging occurs that impact different parts of a device or circuit. The most prominent aging mechanisms are *negative bias temperature instability (NBTI)* and *electromigration (EM)* [16, 17], which, respectively, increase the threshold voltage of a transistor [18] and warp metal interconnects [19, 20]. Also prominent are *time-dependent dielectric breakdown (TDDB)*, which breaks down the gate dielectric [21], and *hot-carrier injection (HCI)*, which also increases threshold voltage [22].



(a) (b)

Figure 1.1: Diagrams of the two mechanisms behind NBTI. Threshold voltage is raised when hydrogen diffuses through the gate (a) [23] or charge carriers are trapped in the channel (b) [24], reducing the current that can be conducted.

Negative Bias Temperature Instability (NBTI)

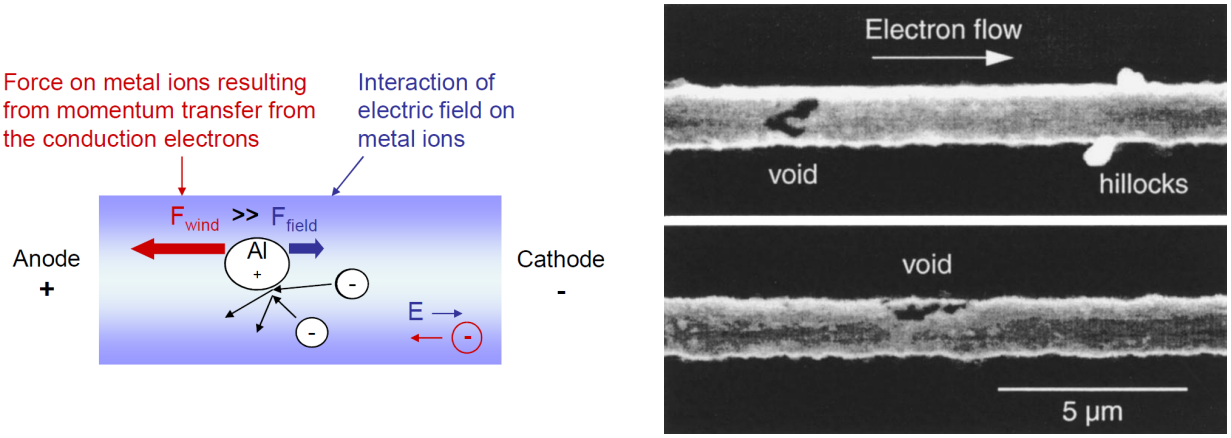
Negative bias temperature instability, or NBTI, creates a gradual increase in the magnitude of the threshold voltage, V_T, of a PMOS transistor. It is caused by the capture of holes in three types of defects: ones that are generated at the gate oxide interface (denoted as ΔN_{IT}), preexisting in the bulk (ΔN_{HT}), and are generated in the bulk (ΔN_{OT}) [18, 23] (Figure 1.1). When a negative bias is applied to the gate of a PMOS transistor, energy wells in the device deepen, forming hole traps and interface traps by exchanging hydrogen bonds at the gate interface. Traps fill according to:

$$\Delta N_{HT} \propto (V_G - V_{T0} - \Delta V_T)^{\Gamma_{HT}} e^{-\frac{E_{AHT}}{kT}} \left(1 - e^{-\left(\frac{t}{\tau}\right)^{\beta_{HT}}}\right) \tag{1.1}$$

where V_G is the gate voltage, V_{T0} is the original threshold voltage, ΔV_T is the threshold voltage shift due to NBTI, E_{AHT} is the activation energy of hole traps, k is Boltzmann's constant, T is temperature, and Γ_{HT}, τ, and β_{HT} are technology-dependent constants [18]. The value of τ is typically very small, indicating that the hole traps capture and release charges quickly and effectively causing ΔN_{HT} to appear time-invariant. This has affected the accuracy of NBTI measurements in the past due to relatively slow measurement times, resulting in underestimation of ΔV_T [25] and accrual of degradation during measurement [26]. Improvements on NBTI measurement techniques have enabled fast measurement that allows capture of ΔN_{IT} effects and more accurate modeling [27].

At the same time, hydrogen ions diffuse through the gate oxide from the interface to the channel (ΔN_{IT}) and eventually reach the interface with the gate terminal on the other side, reacting with hydrogen ions there to create hydrogen molecules. Interface traps are generated according to the following relationship:

$$\Delta N_{IT} \propto (V_G - V_{T0} - \Delta V_T)^{\Gamma_{IT}} e^{-\frac{E_{AIT}}{kT}} t^{1/6} \tag{1.2}$$



(a) (b)

Figure 1.2: Illustration of the physical mechanism behind electromigration (a) [28] and SEM photograph of a wire that has been affected (b) [29].

where E_{AIT} is the activation energy of interface trap generation and $\Gamma_{IT} = \Gamma_{HT}$ [18]. The effects of bulk generated traps are considered negligible, so ΔN_{OT} can be ignored. As these charges accumulate in traps, the threshold voltage of the transistor increases according to $\Delta V_{T,NBTI} \propto \Delta N_{IT} + \Delta N_{HT} + \Delta N_{OT}$, reducing the current the device can conduct and slowing it down. Similar processes cause electrons to become trapped in defects in NMOS transistors. This is called positive bias temperature instability, or PBTI, but it has a smaller effect than NBTI.

When the negative bias is removed, some holes are released from the bulk traps and some of the hydrogen diffuses back to the gate interface, partially recovering the degradation:

$$\Delta N_{HT} + \Delta N_{OT} = B' e^{-\left(\frac{t}{\tau_r}\right)^{\beta_r}} \tag{1.3}$$

where B' depends on the amount of accumulated degradation and τ_r and β_r are technology-dependent constants [18]. Due to the high rate at which interface traps capture and release charges, the NBTI degradation they cause is dominated by the rate at which interface traps are generated.

Electromigration (EM)

Electromigration, or EM, is another important factor in electronics aging. As electrons travel through metal (typically copper and aluminum) interconnect while current flows, they cause metal atoms to slowly migrate in the same direction through transfer of momentum [20], as shown in Figure 1.2(a). This can cause changes to the thickness of affected wires, creating mechanical stress, increasing resistance, and breaking circuits. An example photograph of metal wires affected by EM is shown in Figure 1.2(b) [29].

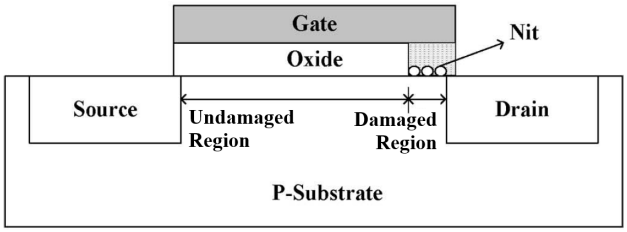


Figure 1.3: Diagram of the mechanism behind HCI. N_{it} represents the trapping of hot electrons in the oxide, causing an increase in threshold voltage [32].

A widely-used method of computing the mean-time-to-failure of an electronic system due to EM is Black's Equation [19]:

$$MTTF_{EM} = Aj^{-n}e^{\frac{E_a}{kT}} \tag{1.4}$$

where j is the current density, E_a is the activation energy of EM, and A is a technology constant that depends on physical characteristics. The value of j can be derived from technology parameters and the operating conditions of the wire [30]:

$$j = \frac{CV_{DD}}{wh} f\alpha \tag{1.5}$$

where C, h, and w are the parasitic capacitance and cross-sectional dimensions, respectively, of the wire and f and α are the frequency and switching probability, respectively, of the signal passing through it. Black determined experimentally that $n = 2$, but more recently n has been determined to be affected residual stress and current density [20].

Like NBTI, EM experiences some recovery when current is reduced or removed that is caused by a reverse flow of metal atoms [31]. It is also magnified by application of reverse currents, causing extended lifetimes in wires with bidirectional flow. Because of this, signal wires, which typically pass bidirectional currents, are less susceptible to EM than power wires that usually only pass current in one direction.

Hot-carrier Injection (HCI)

Hot-carrier injection, or HCI, is caused by the injection of energetic "hot" electrons, accelerated by the lateral electric field in the channel, into the oxide near the drain (illustrated by Figure 1.3), creating interface traps and raising threshold voltage [22]. Since this occurs mainly when high current is passing through the channel, HCI is more dominant during switching. It is characterized using a widely-used substrate current-based model [33]:

$$\Delta V_{T,HCI} = A_{HCI} e^{\frac{E_{ox}}{E_0}} e^{-\frac{E_a}{kT}} (\alpha f t)^{n'} \tag{1.6}$$

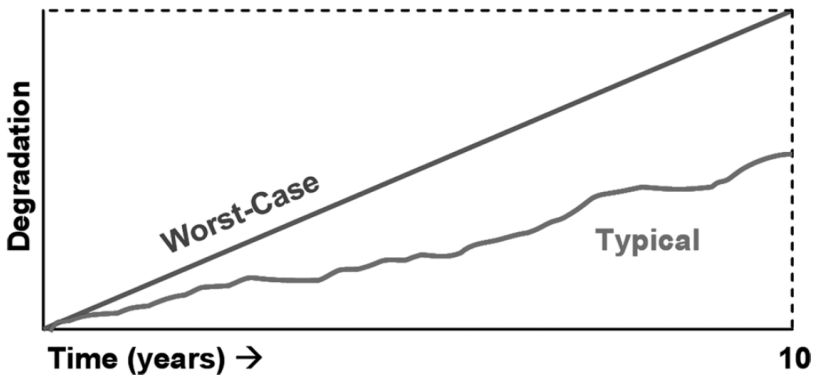


Figure 1.4: Illustration of aging rate over time. The aging rate of a system typically varies over time and is less than the worst-case that is often designed for [34].

where E_{ox}, E_0, and E_a are technology-dependent coefficients, α is the average switching activity of the gate, f is the clock frequency, t is the time since the beginning of operation, and $n' \approx 0.5$. Unlike NBTI, HCI is more prominent in NMOS transistors.

Time-dependent Dielectric Breakdown (TDDB)

Time dependent dielectric breakdown, or TDDB, is the process of wearing down a transistor's gate oxide over time. Eventually a conductive path forms, causing failure [21]. This is particularly problematic with high power densities that raise temperature and in devices with thin gate oxides. The authors of [21] show that TDDB is highly dependent on voltage and temperature. It's effect is expressed using meant-time-to-failure:

$$MTTF_{TDDB} \propto V_{DD}^{a-bT} e^{\frac{X+YT^{-1}+ZT}{kT}} \tag{1.7}$$

where a, b, X, Y, and Z are fitting parameters.

1.1.2 Managing Aging

The typical method for accounting for aging is to *statically calculate the worst-case degradation* that could occur and *add timing margins* at design time to account for it and ensure a target lifetime [35, 36]. This can amount to increasing margins by over 20% to ensure a 10-year lifetime. This is unsatisfactory because, as shown by Figure 1.4 [34], a system is not always aging at its worst-case rate. The slack between the worst-case degradation that could occur and the real amount of degradation that does occur means that power and performance is being wasted in order to ensure operation for the entire target lifetime. Unfortunately, due to varying and unpredictable workloads, it can be difficult to predict the real amount of aging that occurs. Rather than predict it, a system can *track aging as it occurs and dynamically modify operating parameters*

to adapt to it [37, 38]. Techniques for dynamic adaptation exist at all levels of abstraction from the circuit level up to the software level and usually involve redundancy [35, 39], failure tolerance [40], or intelligent management of resources [41, 42].

The most diverse methods for aging management are at the circuit level. To extend lifetime or reduce the margin necessary to account for aging, proactive techniques such as inclusion of redundant circuit copies or error correction can be applied [35]. These techniques incur area overheads and do not take into account the dynamic nature of aging; it is possible, over the course of a system's lifetime, that a non-critical circuit path will degrade faster than the critical path and change into the critical path [43]. If this happens to a path that doesn't have redundancy, then the system's lifetime will no longer be extended by the original critical path that does. More reactive circuit-level solutions for detecting aging often involve sensors that degrade at a predictable rate. This degradation will reduce circuit parameters of the sensor (e.g. with critical path copies [44]) or change its output characteristics (e.g. with ring oscillators [45]). When the changes in parameters or output of an aging sensor reach a threshold, an aging event is generated and the system can adjust its performance requirements to continue functioning at a reduced performance level, thus extending lifetime or reducing the required margins at design time. Such sensors incur area overhead like static solutions and have additional problems in that they may not be representative of the circuits they are intended to sense. While they can be placed adjacent to those circuits to create similar temperature and voltage environments, the activities of the sensors may not be the same as those of the circuits they sense, especially in the cases of simple circuits like ring oscillators. A dynamic, proactive solution that has been proposed [46] to mitigate NBTI aging is to periodically apply a reverse bias to the gates in a circuit. Doing so reduces the trap energies of the defects in the gate, increasing the rate at which charges are freed. The authors of [46] show that periodically applying reverse bias can reduce the permanent component of NBTI, ΔN_{IT} (see (1.2)) and greatly extend lifetime. Applied frequently enough, it can completely remove the permanent component.

Second, at the architecture level, redundancy is again a common method for mitigating aging. In a multicore system with heterogeneous workloads, each core will degrade at a different rate based on the workloads it is given. As cores fail, they can be deactivated to allow the system to continue to function at a reduced performance level [40]. Similarly, duplicating structures inside a core can increase its lifetime by allowing duplicates to take over when originals fail [39]. Memories can additionally be protected from aging by augmenting them with aging monitors and additional logic to place data in such a way as to balance the aging of memory cells and thus maximize lifetime, as [41] proposes for scratchpad memories.

Finally, at the software level, workloads can be optimized to balance aging across system components as evenly as possible. This often takes the form of aging-aware scheduling [42], which assigns tasks to cores in such a way as to balance their aging effects. This adds new dimensions to task schedulers beyond performance,

power consumption, and temperature, as optimal solutions for those metrics are not necessarily optimal for aging and vice versa [30].

Adding aging mitigation or management techniques to a design typically incurs overheads in the forms of increased power and area, reduced performance, or both. Increasing static timing margins has a disadvantage in adding unnecessary timing slack to ensure worst-case lifetime when the average case will not need as much. It also sacrifices early-life performance to ensure end-of-life reliability. Dynamic adaptation to aging allows a circuit to increase early-life performance at the expense of end-of-life performance while still maintaining a target lifetime, but has its own disadvantages. Aging detection incurs power and area overheads from extra hardware for sensing degradation or extra circuitry like redundant units to ensure continued functionality after a unit fails [47]. Additionally, the calculation necessary to adapt to aging and modify architectural and circuit parameters or schedule workloads creates an additional performance overhead.

1.2 Importance of High-level Modeling

The overheads caused by adding timing margins or mitigation techniques create tradeoffs for lifetime, power, performance, and area. As designs get more complex, the amount of available design space increases rapidly. Simulating them also becomes more time-consuming, reducing a designer's ability to explore design space using iterative techniques such as the Monte Carlo method that are often used for lifetime calculation [30, 40]. The most accurate estimations of these quantities come from circuit-level simulation, where libraries are available that characterize the behaviors and performance metrics of devices and gates and include parameters that describe how those metrics degrade as they age. This type of simulation, however, requires an RTL implementation of a design, which has high design and simulation overhead. On top of this, some phenomena have effects across multiple layers of abstraction. Two examples of this are temperature [11] and voltage noise [10], which affect circuit operation but are affected by microarchitecture, floorplan, power delivery, and pad placement. Design decisions for managing them must be made as early as the architecture design stage [48], while their effects are primarily circuit-level. In order to enable effective design decisions at the architecture level, it is important to be able to simulate them without needing RTL. High-level models not only improve simulation overhead over RTL simulation, but enable high-level decisions that depend on these low-level effects.

Aging is an important low-level effect that has implications across all layers of design abstraction. Because it often depends on circuit topology and activity, accurately predicting the amount of degradation that will occur is difficult without a complete simulation of a workload. Such simulations at the circuit level are time-consuming and infeasible for large numbers of instructions, reducing the usefulness of device-level aging

libraries. High level models remove much of the overhead of circuit simulation by making assumptions about a circuit's power, performance, and aging based on architecture-level behavior to reduce unnecessary detail and increase throughput. Further discussion of assumptions that are made to reduce simulation overhead and enable realistic lifetime prediction will be included in Chapter 2. With lower simulation overhead, not only does iteration become feasible for design exploration, but software effects on power can be studied. This is important because the rapid increase in low-power requirements, for example due to the rise of mobile computing and IoT, coupled with the end of Moore's Law means that software designers can no longer afford to ignore the power consumed by their workloads or rely on technology scaling to improve power consumption. The same is true for aging: it has been shown that lifetime is significantly affected by workload characteristics [49]. With small transistors that have higher power densities than their predecessors due to the lack of comparable voltage scaling [5], circuits are more sensitive to aging than ever.

Unfortunately, the increased speed over RTL simulation afforded by high-level modeling comes at the price of reduced accuracy. Several high-level tools [50–52] show error in their reports of metrics like power, performance, and area caused by assumptions they make about the behavior of circuitry that enables abstraction and reduced detail. These assumptions allow aggregation of the behavior of the devices within each architectural unit and memory that simplifies computation of these metrics. By abstracting away device activity and layout from a simulation, however, some error is introduced because the assumed aggregate behavior may not match the real behavior caused by a workload. This error can be improved by characterizing real hardware or RTL simulation and calibrating high-level tools using measured power, performance, and area [50], but this reduces the general usability of the tool. As a result, high-level simulation cannot be used as a total replacement for RTL simulation. Fortunately, accuracy is high enough that comparisons between different sets of choices can still be made [51, 52], enabling the use of these tools in a flow for exploring large design spaces caused by complex modern designs and allowing architects to incorporate them in their decisions, reducing the amount of space that needs to be explored by RTL simulation.

1.3 Hypothesis and Contributions

The hypothesis of this work is that the *reliability of an electronic system can be improved with pre-RTL lifetime and reliability simulation using high-level models* and the *security of these systems can be manipulated by applying directed aging.* To enable pre-RTL design space exploration to include lifetime as well as power, performance, and area, Chapter 2 introduces a tool called "OldSpot" that improves upon existing reliability models by removing restrictions on how failures can propagate through a system. This tool can also be used

to reveal areas in a design that are highly vulnerable to aging and bottleneck lifetime or to evaluate reliability management mechanisms such as hardware redundancy or reliability-aware task scheduling.

With a fast aging and lifetime simulator, designers can easily explore techniques to improve lifetime or to improve power and performance while maintaining a target lifetime. To get the best results for those models, accurate performance simulation is necessary. A widely-used tool for this purpose is gem5 [53], which traces performance at the architectural unit level, models a complete memory hierarchy, and supports execution of binaries compiled for several architectures like ARM, x86 and SPARC. In addition to providing behavioral models of architectural units for improved simulation time, it has several features that reduce simulation overhead such as system call emulation and checkpointing. It also can be used in conjunction with power [54–56], temperature [11], and voltage noise [10] models that are useful in computing aging acceleration factors to model the results of time-varying aging rates caused by changes in those quantities. High-level modeling tools and a flow for simulating power, performance, area, and lifetime of a design are discussed in Chapter 3.

Aging has implications not only for lifetime and reliability, but also security. This work explores the effects of aging on an architectural unit used for authentication, the SRAM PUF [57]. A PUF is a device that uses silicon process variations to create uniqueness, similar to a fingerprint of a human being, and is useful for security applications like device authentication. In an SRAM PUF, this fingerprint is defined by the data it contains immediately after powering on, which is affected by the relative strengths of the transistors in its cells. As a result of this, its output is subject to change by aging degradation, which changes the strength of a transistor. PUFs, and how they are affected by aging, are further discussed in Chapter 4, which shows how the fingerprint of an SRAM PUF can be affected by aging by controlling its reliability in successful authentication using directed NBTI.

The contributions of this work are summarized as follows:

- An open-source lifetime simulation tool named “OldSpot,” which will be discussed in Chapter 2.
- An implementation of the RISC-V ISA in the gem5 simulator to use with a simulation flow, which will be discussed in Chapter 3.
- A method by which an SRAM PUF can be attacked and even cloned using directed aging, which will be discussed in Chapter 4.

Chapter 2

Reliability Modeling

The goal of architecture-level reliability research is to characterize and improve the lifetime of a system or reduce costs associated with designing for a lifetime target. For that purpose, a description of the degradation of a device, i.e. ΔV_T for NBTI, is not useful because circuit-level effects such as process variation, noise, and activity can cause different amounts of degradation for different devices in a unit. Simulating aging for all transistors across a system to determine failures takes too long and provides more detail than a designer needs. Additionally, failures tend to be infrequent, occurring on the order of years, rendering simulation of a system until failure infeasible. To address this, reliability is often described for a unit or system using one of two metrics: mean time to failure (MTTF) or failures in time (FITs). These metrics are derived from device-level models of failure mechanisms such as those in Chapter 1.1 by making assumptions about how block-level behaviors like activity and temperature affect the devices inside and by extrapolating long-term behavior from short simulation times. MTTF and FITs are inverses of each other; MTTF describes the average lifetime of the system while FITs describes the number of expected failures that will occur in a given amount of time (usually 10^9 hours [30]). This chapter presents existing methods of computing unit- and system-level MTTF and a tool called "OldSpot" [58] that improves upon these methods to enable fine-grained modeling and nonuniform failure tolerance.

2.1 High-level Reliability Modeling

A common way of computing the reliability of a system over time is by using a probability distribution, $R(t)$, to express the probability that a system still functions at time t or, equivalently, the expected fraction of systems in a given population that still function at time t [59]. In the past, exponential [30] and lognormal [39] distributions have been used for this purpose, but more recently the Weibull distribution has been used

because it most accurately fits the reliability distribution of a system whose failure rate increases over time, as it does with aging [60]. The reliability function and MTTF for the Weibull distribution are:

$$\begin{aligned} R(t) &= e^{-\left(\frac{t}{\eta}\right)^{\beta}} \\ MTTF &= \eta\Gamma\left(1+\frac{1}{\beta}\right) \end{aligned} \tag{2.1}$$

where η is called the *rate parameter* and β the *shape parameter*. Larger values of η indicate longer lifetimes while β indicates how failure rate changes over time.[1] For aging, $\beta = 2$ [61], which indicates an increasing failure rate over time.

This is useful for a static system that ages at a constant rate, but a typical electronic system experiences changes in temperature, power consumption, voltage, and other operational parameters over time that affect the rate. Together, the values of these parameters represent the *execution state* of the system. During a particular execution state, these parameters are considered constant except temperature. In order to account for changes in execution state, it is possible to derive an average aging rate that is constant in time from the instantaneous aging rates that occur due to varying conditions [62]:

$$R\left(t\right) = R\left(\eta \cdot \sum_{s\in\mathcal{S}} \int_0^{\infty} \frac{\psi\left(T,s\right)}{\theta\left(T,s\right)} dT \cdot t\right) = R\left(\eta \cdot \Omega \cdot t\right) \tag{2.2}$$

where $\mathcal{S}$ is the set of execution states the system undergoes with discrete voltages and activity factors, $\theta\left(T,s\right)$ is the rate at which it ages in state s and temperature T, and $\psi\left(T,s\right)$ is the probability it is in state s at temperature T. This results in a constant, Ω, that represents the average aging rate of the system across the simulation time. By assuming that the workload of the system is periodic and that the duration of the simulation is small compared to lifetime (i.e. seconds or minutes compared to years), Ω can be used to extrapolate lifetime. By further assuming that the system's temperature settles quickly and so is effectively piecewise constant, (2.2) can be simplified [60]:

$$R\left(t\right) = e^{-\left(\frac{\sum t_i/\eta_i}{\sum t_i} t\right)^{\beta}} \tag{2.3}$$

where t_i is the duration of execution state i, during which the aging rate, η_i, can be considered constant. Equation (2.3) has an advantage over (2.2) in that it is easier to compute, but it is only true for Weibull distributions whereas (2.2) is shown in [62] to be true for any reliability distribution.

[1] A constant *aging rate* should not be confused with a constant *failure rate*. Aging rate indicates the speed at which degradation accumulates over time and failure rate indicates the probability that a failure will occur at any given time. When aging rate is constant (and $\beta > 1$), failure rate increases with time.

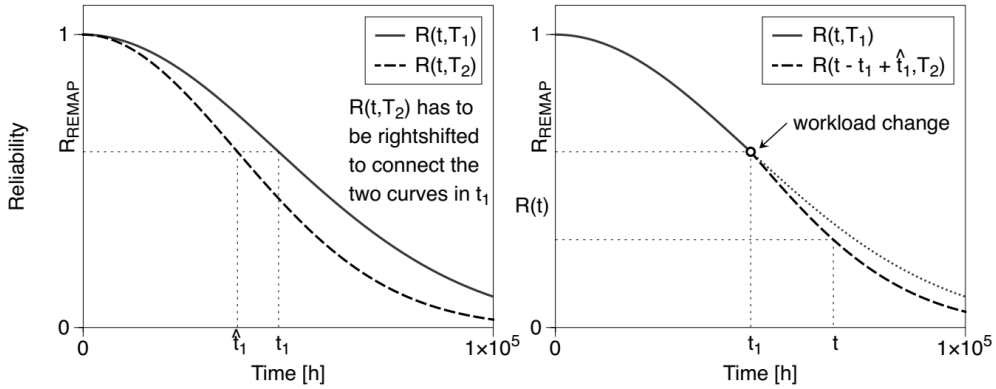


Figure 2.1: Illustration of changes to reliability caused by a long-term change in workload. When this occurs, the new reliability curve must be time-shifted to ensure a continuous $R(t)$ [40].

Both (2.2) and (2.3) depend on static expressions of reliability, where operating conditions are constant, to describe aging during each execution state. They can be determined using device-level models such as those in [20–23] to create aging rates that can be combined. A simple method for doing translating device-level models to the architectural unit level is presented in [63], which simply models each unit as a single transistor. By substituting the unit's operating conditions for the transistor's in each device model, the degradation of the transistor can be used as a proxy for the degradation of the unit, allowing the computation of its aging rate using (2.1). Another method is to represent the unit using a circuit called a FIT of reference circuit (FORC) [64], which is an easy-to-evaluate circuit that represents a particular aging mechanism. By computing the aging of the FORC and then relating its topology to that of the unit of interest, the authors of [64] claim that it is possible to completely separate the lifetime calculation of the unit from process-dependent effects.

Combining low-level models with (2.2) or (2.3) using [63, 64] enables the creation of a constant aging rate that can be used to extrapolate lifetime, but it doesn't capture the effects of long-term changes in workload characteristics that occur outside the simulation window [40]. When a resource fails, the surviving components of the system need to cover for the failed unit, increasing their workloads, raising their activities, and elevating their temperatures as a result. As Figure 2.1 shows, simply changing the aging rate to accommodate the new operating conditions causes a discontinuity in $R(t)$. The system's reliability at time t_1, when the shift occurs, and temperature T_1 is equivalent to the reliability at time $\hat{t}_1 = R^{-1}\left(R\left(t_1, T_1\right), T_2\right)$ at temperature T_2. By shifting $R(t, T_2)$ by $t - \hat{t}_1$ to produce $R\left(t - t_1 + \hat{t}_1, T_2\right)$, the discontinuity disappears and the model can account for the degradation that has occurred prior to t_1. While [40] only considers temperature changes in this model, it can be generalized to any change in workload characteristics that affects aging rate.

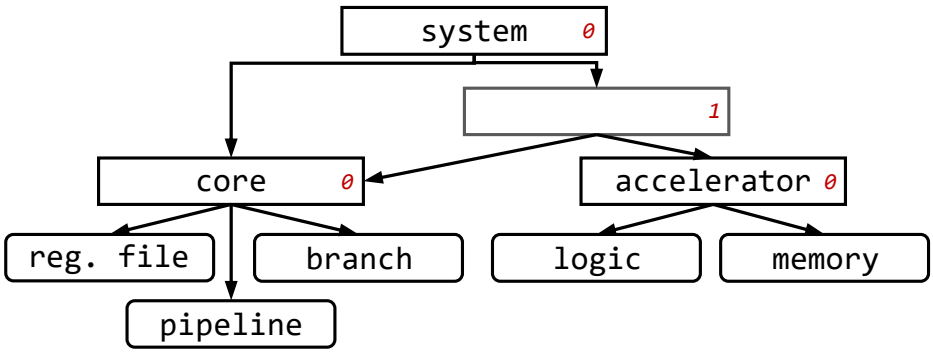


Figure 2.2: Example failure dependency graph for a system with a single core and an accelerator that include several functional units [58]. Red numbers represent the number of failures each group can tolerate. By adding a node between `system`, `core`, and `accelerator` that can tolerate one failure, it is possible to represent a system that can tolerate the failure of `accelerator` but not `core`.

2.2 The OldSpot Framework

The work in Chapter 2.1 enables lifetime and reliability estimation for homogeneous multicore systems at the core level. It allows an architect to evaluate a tradeoff between late-life performance and lifetime by predicting the benefit of allowing a certain number of cores to fail. But because these tools work at such a coarse granularity, the only recourse they provide for extending lifetime is to add more cores and tolerate more failures, adding significant area and power overheads. OldSpot improves upon existing tools by allowing specification by the user of *failure dependency*, or the ways in which the reliability of each unit in the design contribute to the overall reliability of the system. The tool does this by using a directed acyclic graph (DAG) that places components into a tree-like structure to specify how errors in units propagate across the system. This relaxes the assumption that all units contribute equally to the system's reliability and allows tolerance for failure in some places but not necessarily others. OldSpot is open-source, developed in C++, and available for download at `https://github.com/hplp/oldspot` or `http://lava.cs.virginia.edu/OldSpot/`.

2.2.1 System Specification

OldSpot's system specification is inspired by the idea of "structural duplication" proposed by [39], where the lifetime of an entire system may be improved by duplicating individual functional units within it. The authors of [39] propose to link units together in series and in parallel to indicate which units are duplicated. The system only fails when all of the units in a parallel group fail or when one of the units connected in series fails. OldSpot expands on this idea by receiving as input a *failure dependency graph* to combine parallel and series connections that specify how errors in individual units propagate across the system. A failure dependency graph is a DAG where leaf nodes represent instances of units in the design and internal nodes

Algorithm 2.1 OldSpot Monte Carlo iteration

let $\mathcal{U} \equiv$ {all units} ▷ Leaves, i.e. `reg.` `file`, `branch`, `pipeline` in Figure 2.2
let $\mathcal{G} \equiv$ {all groups} ▷ Internal nodes, i.e. `system`, `core`, `accelerator` in Figure 2.2
let $R \equiv$ root group ▷ i.e. `system` in Figure 2.2

$\mathcal{F} \leftarrow \emptyset$
$\mathcal{H} \leftarrow \mathcal{U}$
$t_f \leftarrow 0$
while $R \notin \mathcal{F}$ **do**
  for $h \in \mathcal{H}$ **do**
    SetConfiguration $(h, \mathcal{H})$ ▷ Set Weibull η parameter based on current activity
  $\Delta t \leftarrow \infty$
  for $h \in \mathcal{H}$ **do**
    $r_f \leftarrow \text{rand}\,[0, r_h)$
    $\delta t \leftarrow R_h^{-1}(r_f) - R_h^{-1}(r_h)$
    if $\delta t < \Delta t$ **then**
      $f \leftarrow h$
      $\Delta t \leftarrow \delta t$
  for $h \in \mathcal{H}$ **do**
    $t_h \leftarrow t_h + \Delta t - \left[R_{h,prev}^{-1}(r_h) - R_h^{-1}(r_h)\right]$
    $r_h \leftarrow R_h(t_h)$
  Add f to $\mathcal{F}$
  Remove f from $\mathcal{H}$
  for $g \in \mathcal{G}$ **do**
    if $|children(g) \cap \mathcal{F}| > tolerance(g)$ **then**
      Add g and its children to $\mathcal{F}$
      Remove g and its children from $\mathcal{H}$
  $t_f \leftarrow t_f + \Delta t$

represent groups of units that are connected to each other. For example, a core might be represented by an internal node connected to several leaf nodes that represent its pipeline, caches, and other functional units. Internal nodes can also be connected together to indicate how groups depend on each other. The entire system is represented by a root node at the top of the graph. A node can have multiple parent nodes to specify that its failures propagate in multiple ways. Each group also contains a number that specifies how many failures of its children that it can tolerate before it fails. An example failure dependency graph that depicts a system with one core and one accelerator that can tolerate the failure of the accelerator but not the core is shown in Figure 2.2. The `core` node is connected to both the `system` node and unnamed internal node to indicate that its failure causes overall system failure, but that it operates in parallel with the `accelerator` node in case `accelerator` fails.

A comparable method for describing failure propagation is the creation of a *fault tree* [65]. In general, a fault tree is a representation of the events that can lead to a specific failure. When applied to system lifetime due to aging, each event is represented as the failure of a unit or group that eventually leads to the failure of the whole system. A leaf node in a fault tree represents a fault event, similar to how leaf nodes in OldSpot's

failure dependency graphs represent failures of individual units. Internal nodes indicate how faults propagate through a system to eventually cause failure, which is the same for a failure dependency graph. The major difference between a fault tree and failure dependency graph is that a fault tree's internal nodes consist of boolean logic gates such as AND, OR, and XOR rather than accumulation of failures until a threshold is reached. This allows analytical derivation of the reliability of a system, but increases the complexity of representing tolerance for failures of certain resources like cores.

2.2.2 Reliability Modeling and Simulation

In order to compute the reliability distribution for each unit, OldSpot also makes use of traces of power, performance, temperature, and voltage, which are discussed in Chapter 1.1. For each time step in each unit's trace, the instantaneous aging rate of each aging mechanism is computed to produce a Weibull distribution. These distributions are combined according to the sum-of-failure-rates (SOFR) model [30], where the overall reliability of a system is the product of the reliability of its components, to create an overall reliability distribution for the unit (See Appendix B for details). Then, using (2.3), these reliability distributions are combined to create an average aging rate and corresponding average reliability distribution that describes the lifetime of the unit. Finally, in a similar manner to [39, 40], a Monte Carlo simulation determines the lifetime distribution of the entire system, using the failure dependency graph to determine when it fails. The procedure for each iteration of the Monte Carlo simulation is specified by Algorithm 2.1, which is adapted from Algorithm 1 in [40] to accommodate OldSpot's failure dependency graph.

During each iteration of the simulation, each healthy unit, h, is assigned a Weibull η parameter based on the set of remaining healthy units, $\mathcal{H}$, and the unit's activity for that configuration. This accounts for the change in activity that occurs when a unit fails and the system is executing its workload with a restricted set of resources. Then h is assigned a reliability value r_f at which it fails and the time to that reliability, δt, is computed. When δt has been computed for each unit, the minimum value is used to determine the next failure. The failed unit, f, is removed from $\mathcal{H}$ and added to the set of failed units, $\mathcal{F}$. The "age," t_h, of each healthy unit is updated based on its reliability for the previous configuration, $R_{h,prev}(t)$ and its reliability for the current configuration, $R_h(t)$ (see line 15).

As each unit fails, the groups they belong to keep track of how many of their children have failed and report failure they exceed their failure tolerance. When the root node reports failure, the system is considered failed and the next iteration begins with a fresh system. All of the fail times are collected to produce a reliability distribution at the end of the simulation and the mean time to failure can be computed as the average of all of the fail times.

2.2.3 Assumptions

In order to reduce simulation time and overhead in gathering data, OldSpot makes several assumptions about the nature of aging in its components. The foremost assumption it makes is that the reliability of an electronic system due to aging follows the Weibull distribution. Aging has been shown to cause an increased failure rate over time [39] rather than a constant one [30], which can be addressed using Weibull [40, 60] or lognormal [39] distributions. Of these, the Weibull distribution has been shown to be more accurate [60].

Next, as is common in the state of the art, OldSpot assumes that each aging mechanism is isolated and unaffected by changes to circuit parameters caused by other aging mechanisms. This has been partially shown by [66] to be untrue, particularly for NBTI and HCI because they affect the same parameter (ΔV_T), but [66] focuses on improving accuracy over the assumption that the only aging mechanism that acts on a device is the dominant one, which was the state of the art at the time of its writing; the authors show that that assumption can underestimate aging by up to 75%. OldSpot addresses this by combining the reliability models for all of its aging mechanisms rather than simply using the one with the lowest rate parameter, but does not address any possible "interference" between aging mechanisms that may occur.

The third assumption it makes is borrowed from [63], which assumes for NBTI that the activity factors for transistors in a logical unit are evenly distributed across the unit. This assumption can be improved by characterizing RTL implementations of different kinds of units, such as ALUs, FPUs, decoders, and so on, for the distributions of activity factors across the transistors within them. Similarly, OldSpot assumes that high-order bits of memories tend to not change often and thus their aging is dominated by NBTI and EM, which occur due to constant applied voltage and current. Like with logic units, this assumption can be improved by characterizing memories for their activity factors. Unlike with logic, the activities of transistors in memories cannot be estimated by usage but must be computed from data content. This requires periodic examination of the contents of a memory or register file, which can drastically increase simulation time for performance characterization.

2.3 Finding Reliability "Hot Spots" with OldSpot

Due to differences in activity across a chip that affect power consumption and temperature, the rate at which different functional units age is not uniform. In fact, some areas may age significantly faster than others due to much higher activity. These can be referred to as reliability "hot spots," [58] which can be targeted for aging mitigation to reduce power, performance, and area overheads that such techniques typically cause. For example, rather than adding an entire extra core for redundancy to extend lifetime, simply identifying

Table 2.1: Simulated System Parameters [58]

Parameter	Value
Instruction set	x86
Microarchitecture	Nehalem
Technology size	65 nm
Supply voltage	1.1 V
Core count	4
CPU clock frequency	2.66 GHz
Instruction cache	32 kB
Data cache	32 kB
Private L2 cache	256 kB
Shared L3 cache	8 MB

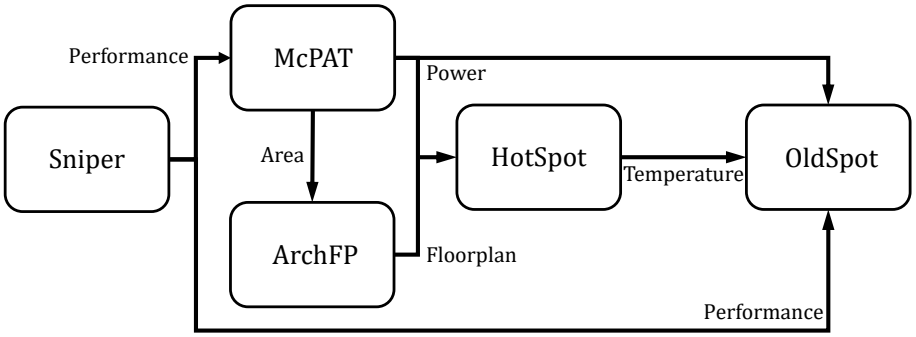
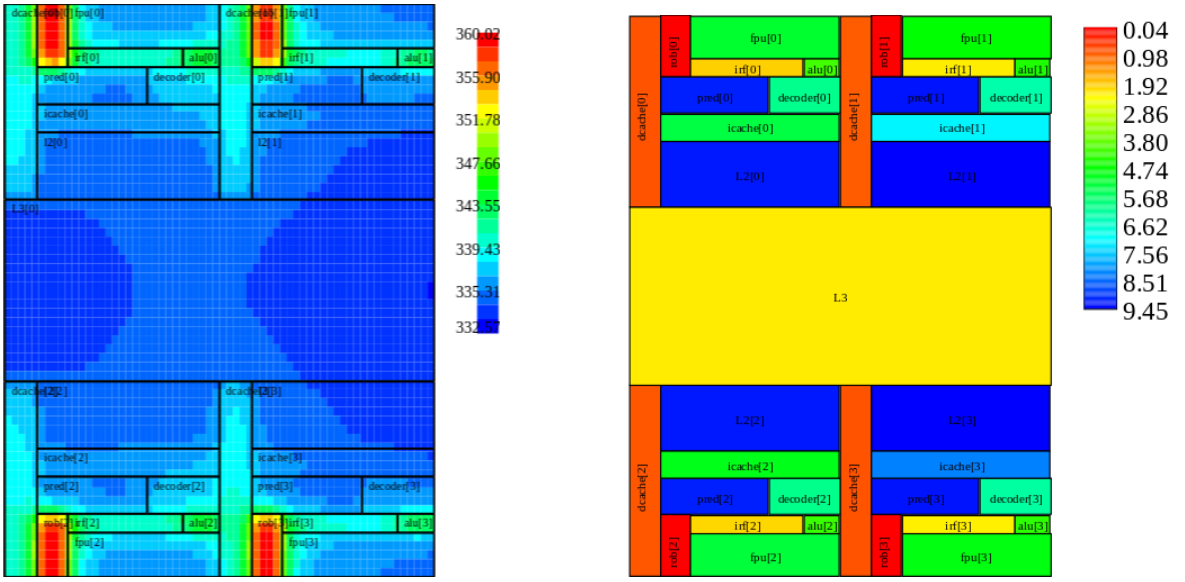


Figure 2.3: Simulation tool flow diagram used to evaluate reliability hot spots [58]. Performance data from the Sniper [67] x86 simulator is input into McPAT [54] to produce power and area data for HotSpot [11]. Power, performance, and temperature data from these tools are all input into OldSpot [58] to compute lifetime.

hot spots and duplicating them can improve lifetime almost as much at significantly lower cost. Alternately, early-life performance can be maintained longer by reducing the number of failures that must be tolerated while still meeting a target lifetime. This section shows how OldSpot can be used to analyze a system for reliability hot spots and then evaluate the effectiveness of duplicating them to extend lifetime.

2.3.1 Tool Flow

A four-core x86 processor with shared L3 cache based on Intel's Nehalem architecture described in Table 2.1 is simulated with the Sniper multicore simulator [67] to execute several benchmarks from the PARSEC [68] and SPLASH2 [69] suites. Performance data from Sniper is input into an integrated McPAT [54] to compute the area of the chip and power consumption of each benchmark. The power and area data are then input into HotSpot [11] to estimate temperature using ArchFP [70] to create a floorplan. Traces from the above tools are input into OldSpot along with a user-specified failure dependency graph for the processor, which creates a lifetime distribution for it using the Monte Carlo simulation described in Chapter 2.2. At the same time, OldSpot produces lifetime distributions for each unit in the system, enabling analysis of aging at



(a) (b)

Figure 2.4: Temperature (a) and aging (b) maps [58] representing the average temperature and relative aging rate, respectively, when running *cholesky* [68]. In (a), red indicates hotter temperatures and blue indicates cooler ones. In (b), red indicates faster aging while blue indicates slower aging.

the architectural unit level. This tool flow is illustrated by Figure 2.3. By inspecting each unit's reliability distribution, identifying the one with the earliest failures, and duplicating it, it is possible to extend lifetime with minimal area overhead.

2.3.2 Accuracy Validation

To confirm that OldSpot behaves as expected, the temperature of each unit in the system running *cholesky* can be compared to the rate at which OldSpot indicates it to be aging. Figure 2.4 contains heat maps from HotSpot and OldSpot that, respectively, show the temperature and aging rate for each unit computed by the tool. Because aging mechanisms accelerate exponentially with temperature, it is to be expected that units with higher temperatures will also age faster. Comparing Figures 2.4(a) and (b) shows this to generally be true.

An exception to this is the L3 cache, which indicates relatively low temperature. Even though its temperature is close to those of the L2 caches, they age slowly while it ages quickly. This can be explained by the differences in area and total power consumption of the two caches. Because they have similar temperature, their power densities are likely comparable. But because the L3 cache is much bigger, it consumes more power overall, which means it also consumes more current. Even though models for NBTI, HCI, and TDDB do not indicate dependence on current draw, the one for EM shows a power-law dependence on current (see

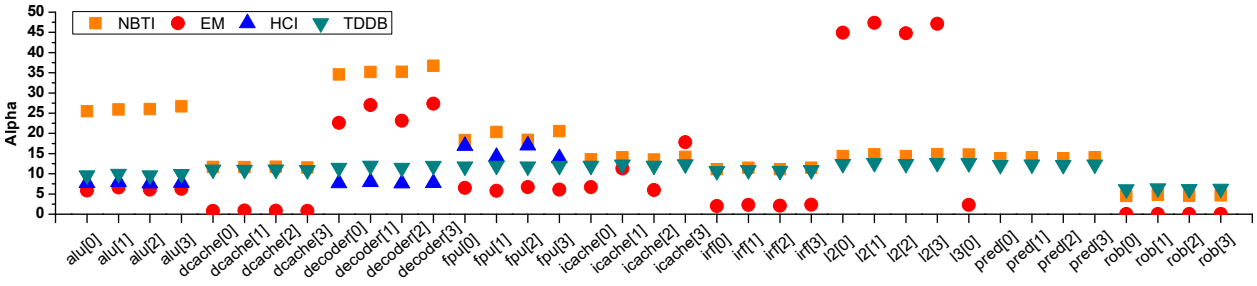


Figure 2.5: Weibull rate parameter of each unit for each aging mechanism modeled by OldSpot [58] (see Chapter 1.1) running *cholesky* [68]. Higher values mean slower aging rates. Units with no marker for HCI experience very low switching activity, which lead to a very slow HCI aging rate; as a result, the markers were omitted.

Chapter 1.1). Since the current drawn by the L3 cache is much higher than that of any of the L2 caches, its aging rate for EM will be much higher and cause a higher overall aging rate. This is confirmed to be the case in Figure 2.5, which shows the Weibull rate parameter for each aging mechanism in each functional unit. A large rate parameter value in Figure 2.5 indicates a longer lifetime, or slower aging rate. As it shows, the aging rates in the L2 and L3 caches due to NBTI, HCI, and TDDB are close to the same, but the L3 cache is shown to age much faster due to EM. As a result, its overall aging rate is also much faster.

There is also some variation in NBTI aging rates between different types of units. Since NBTI is related to the amount of time a transistor is conducting, high activities in devices will reduce their aging rates. Since the ALUs and decoders have high utilization when executing *cholesky*, they tend to be affected less by NBTI. As expected, Figure 2.5 shows slow aging due to NBTI for these units. On the other hand, the FPUs have low utilization, creating high rates of aging due to NBTI. The opposite is true for HCI, where high activity causes fast aging and low activity causes slow aging, which Figure 2.5 also shows. Finally, there is little variation in aging due to TDDB except for the ROBs, which appear to be most affected. Equation (1.7) shows that TDDB is primarily affected by temperature, which means that the units most affected by it will be the hottest as Figure 2.4(a) shows the ROBs to be.

Figure 2.6 shows the error of the MTTFs computed by OldSpot compared with those computed by the existing multicore lifetime simulation tool called CALIPER [40] for the system running each benchmark that can tolerate the failures of zero through three of its four cores. Because CALIPER [40] can only simulate homogeneous multicore systems without shared resources, the L3 cache is omitted in these simulations. Both tools were executed using 1000000 Monte Carlo iterations. In order to compute core-level aging rates for CALIPER, the unit-level aging rates were computed using [63] and (2.3) and combined using SOFR per core (see Appendix B for details). OldSpot was configured to tolerate the same number of failures per core as CALIPER, with each core failing when any of its units failed. As Figure 2.6 shows, OldSpot's results

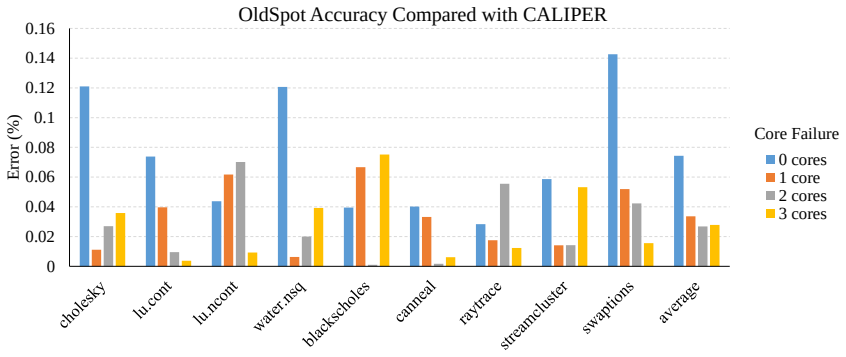


Figure 2.6: Accuracy validation of OldSpot [58] against CALIPER [40] using a four-core x86 system with no shared resources running several PARSEC [68] and SPLASH2 [69] benchmarks that can tolerate the failures of zero through three cores.

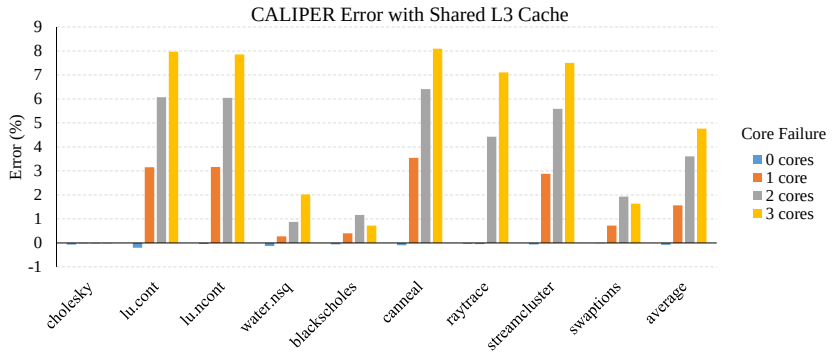


Figure 2.7: Error when using CALIPER [40] to estimate the lifetime of a four-core x86 system with shared L3 cache that can tolerate the failures of zero through three cores but not the L3 cache when running several PARSEC [68] and SPLASH2 [69] benchmarks. When the system can tolerate core failures but not failure of the L3 cache, CALIPER overestimates the system's lifetime.

are, on average, less than 0.08% in error compared to CALIPER and about 0.14% in the worst case. This confirms that accuracy is maintained when monitoring failures at the architectural unit level rather than at the core level and when using a failure dependency graph to propagate failures through the system. It does not confirm the accuracy of computing the unit-level aging rates from device-level models, but verifying those computations requires either RTL simulation or hardware measurement to verify device-level model parameters.

Figure 2.7 shows the benefit of estimating lifetime using OldSpot's failure dependency graph rather than using a tool such as CALIPER. It contains lifetime estimations from OldSpot for the same four-core x86 systems presented in Figure 2.6 except that the shared L3 cache, whose failure cannot be tolerated, is no longer omitted. Because CALIPER does not normally model shared resources, the MTTF it reports was combined with that of the L3 cache using SOFR as described in Appendix B, assuming the cache and set of cores both follow Weibull distributions with $\beta = 2$ [61]. The L3 cache's MTTF was computed using

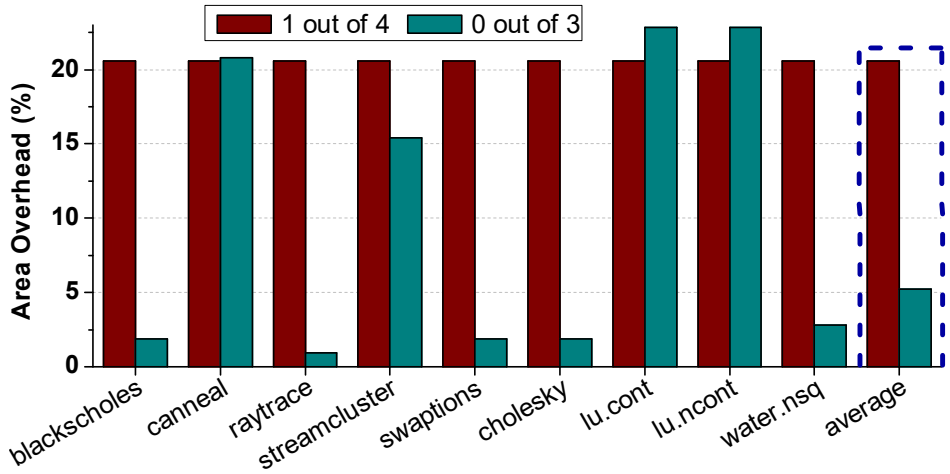


Figure 2.8: Area overhead of extending the lifetime of a three-core system by either adding a fourth core and tolerating one failure (red) or by identifying aging hot spots and duplicating them (green) [58]. The final bar on the right shows that, on average, overhead can be reduced by about 75%.

OldSpot's reported aging rate and (2.1). As Figure 2.7 shows, increasing the number of core failures that can be tolerated tends to cause CALIPER to overestimate the lifetime of the system by an increasing amount. This is due to inaccuracy of the assumption that the system follows a Weibull distribution with $\beta = 2$ when failure tolerance is introduced, which is not evident from CALIPER's computations. OldSpot can account for this by including the dependence on the L3 cache in its simulation, improving accuracy. On average, CALIPER's overestimation is about 1.5% with one tolerable core failure, 3.6% with two tolerable failures, and 4.8% with three. There is negligible error for a failure-intolerant system because the assumption about the shape of its reliability distribution holds. Similarly, the L3 cache in *cholesky* ages very slowly, making its contribution to the system's lifetime negligible and causing the results from OldSpot and CALIPER to be comparable.

2.3.3 Finding Reliability Hot Spots

The tool flow in Chapter 2.3.1 is executed with two processor configurations for each benchmark. First, the configuration defined by Table 2.1 is used with an additional failure tolerance of one core, named "one-out-of-four" in this work. In this case, when one of the four cores fails, the work of the failed core is picked up by the remaining three, increasing their workloads and aging rates but allowing the system to continue functioning until a second core fails. Next, the configuration is modified to include only three cores but to be intolerant of the failure of any, named "zero-out-of-three." By inspecting the lifetime distributions of each unit in the zero-out-of-three case and iteratively duplicating the unit with the highest number of failures, the lifetime of the zero-out-of-four case can be duplicated by the zero-out-of-three case with less area.

Figure 2.8 shows, for each benchmark, the area overhead incurred by duplicating enough structures in the zero-out-of-three system to allow it to last at least as long the one-out-of-four one (shown in green) with the overhead of adding an extra core for comparison (shown in red). In the best case it is possible to match the lifetime of the one-out-of-four case with less than a 1% area overhead while several others only incur a little more. Even so, several benchmarks incur significant overhead using structural duplication; in two cases (*lu.cont* and *lu.ncont*), the overhead of structural duplication exceeds that of adding a fourth core. These benchmarks' lifetimes are dominated by the L3 cache, which is shared by all cores and not considered for structural duplication due to its large size. Duplicating the L3 cache would increase the total area of the system by about 38%, which is almost twice the overhead of adding a fourth core. The result of this is that the benefits of duplicating structures inside cores for these benchmarks is lower than doing so for the other benchmarks whose lifetimes are primarily determined by the cores. It is likely that the lifetime of an L3-dominated system could be significantly improved by dividing the L3 cache into blocks and allowing graceful reduction of L3 cache size until all blocks fail, but this simulation was not performed to maintain focus on structural duplication. On average, the area overhead of extending lifetime using structural duplication was only 5%, about 25% of the overhead caused by adding a fourth core.

2.4 Related Work

One of the first methods of architecture-level reliability modeling was proposed by the authors of [30, 39], who created a tool called Reliability-Aware MicroArchitecture (RAMP) . In its first version, RAMP assumes a constant failure rate over time and uses an exponential distribution to express reliability. As the authors of [30] admit, this is inaccurate because failure rate due to aging typically increases with time. It is also limited by the use of the SOFR model to express the reliability of a system with multiple components, assuming that the failure of any component means the failure of the whole system. Despite these inaccuracies, these assumptions were used due to their acceptance as standard by industry at the time [30]. The authors of [30] use RAMP to propose dynamic reliability management to go along with dynamic thermal management and show that reliability-aware design and dynamic adaptation can improve a system's lifetime. Later, in [39], RAMP is improved to relax the two previous assumptions: the exponential distribution was replaced with a lognormal distribution and the series system was replaced with a series-parallel system as discussed in Chapter 2.2.1. Even so, it is still limited by the fact that it assumes constant aging rates over time and cannot model shifting workloads.

This is improved by [60] and [62], which propose similar methods for computing the reliability function for a system whose aging rate changes over the course of a workload. As discussed in Chapter 2.1, they introduce

methods of computing an average aging rate from the transient aging rates that occur during execution. By assuming that a workload is periodic and that the period is negligible compared to the lifetime of the device, this average aging rate can be used to derive its reliability function and predict its lifetime. Both methods are still limited, though, by the complex computations necessary to compute the reliability of a series-parallel system and by the fact that they can't account for changes in aging rate that happen outside the simulation window due to component failure or workload rebalancing.

CALIPER [40] further improves the capabilities of architecture-level reliability modeling by introducing a time-shifting factor to account for long-term changes in workload as discussed in Chapter 2.1. The authors of [40] include this in a Monte Carlo simulation to create a lifetime distribution of a homogeneous multicore system with some tolerance for failure after showing that computing the reliability function of such a system to create a closed-form equation is intractable. Like previous models, though, CALIPER is still limited by its assumption of homogeneity and inability to simulate systems at a finer granularity than the core level. Unlike OldSpot, which draws inspiration from CALIPER, it cannot model systems that can tolerate different amounts of failures in different groups of units such as systems that can tolerate core failures but not failures of shared resources.

In order to compute the reliability of a unit during an execution state, it is necessary to convert models for aging mechanisms from device-level models such as those presented in Chapter 1.1 to the unit level or higher. RAMP accomplishes this by simply computing the MTTF due to each failure mechanism and directly translating it from the device level to the unit level. ExtraTime [63] improves upon this technique by introducing several transformations on the device reliability function. First, it applies the architectural unit's execution state to a device-level degradation model. Then, rather than attempt to derive the MTTF directly from the model, which requires knowledge about the design and the process used to fabricate it, [63] converts the degradation model into *relative delay*, which expresses the increase in delay caused by aging as a fraction of the healthy device's delay. Finally, the relative delay model is integrated over the activities of the transistors in the unit. Since that information often depends on the physical layout of the unit, assumptions must be made about the distribution of activities. The authors of [63] assume that activities are uniformly distributed between 0 and 1 within a unit. This is useful for creating accurate estimations of the aging rate of a unit during a particular execution state, which can then be combined with higher-level models to compute the reliability of the entire system. OldSpot incorporates this along with principals from [39], [40], and [60] to compute the MTTF and aging rate of each unit from device level models. These can be used to create a reliability distribution for each unit which are combined as outlined in Chapter 2.2 to create a lifetime distribution for the system.

2.5 Benefits of OldSpot

Because OldSpot does not make assumptions about the organization of the components of a system, it can model more complex systems than previous tools were capable of. Prior tools assume that each major component, such as a core, is treated equally in terms of the system's dependence on its reliability: either any unit's failure means the failure of the entire system or the system can tolerate any combination of a given number of failed units without regard to what those units are. OldSpot is capable of modeling the following types of systems that previous tools could not:

- Systems that can tolerate the failures of some units but not others, such as modern systems that consist of general-purpose cores paired with application-specific accelerators. In some instances, it may be possible for the workloads of those accelerators to be moved onto the general-purpose cores when they fail to extend lifetime, albeit at reduced performance, but the converse is not true.
- Shared resources among multiple cores such as last-level caches. If the shared resource fails, the entire system also fails, but the cores may be able to tolerate failures amongst themselves.
- Individual cores at the architectural unit level or lower. A core's health does not depend on the continued functionality of the units in another core or accelerator.

The analysis of such a system is difficult with previous frameworks due to computational complexity (i.e. [62]) or restrictive assumptions (i.e. [40]). Monte Carlo simulation paired with OldSpot's failure dependency graph alleviates this complexity and allows fast simulation of lifetime.

Chapter 3

Pre-RTL Simulation with High-level Models

In order to get an accurate estimation of aging in an electronic system, an estimation of power and temperature are necessary. To that end, a designer can create a stats-based model tailored to a particular design [55] or use a general, ISA-agnostic model such as McPAT [54]. Stats-based models characterize power consumption for each type of operation a system might perform and compute the power consumption of a workload given traces of those operations. They are accurate for the systems they represent, but require RTL simulations or hardware measurements. Generic power models have their own descriptions of functional units that they use to compute power using usage data from performance simulation. They do not require hardware characterization, but sacrifice accuracy as a result. In either case, power information for a particular workload requires the use of activity information for the system and its components that can be generated by a performance simulator. The gem5 simulator [53] is a widely-used tool that supports many ISAs and provides detailed performance information about the execution of a program. Power and performance simulation tools enable the use of other tools to compute temperature [11], voltage noise [10], and lifetime [40, 58], forming a flow discussed in this chapter.

It is also important that these tools be open-source and freely available. This facilitates research and collaboration between academia and industry by avoiding complex and expensive licensing issues that can interfere with the acquisition of proprietary software or hardware and hinder communication between individuals who wish to use it. Being open-source additionally enables improvement through community effort and allows users to verify functionality and security through their own simulations and design rather

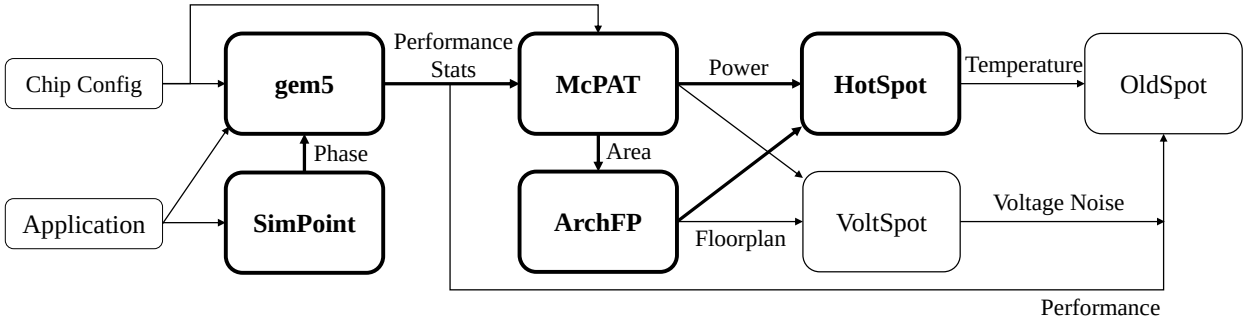


Figure 3.1: Illustration of a tool flow for simulating an application running on a chip [48]. Main simulation tools include gem5 [53], McPAT [54], HotSpot [11], VoltSpot [10], and OldSpot [58]. Additional tools include SimPoint [73] and ArchFP [70]. Together, these applications characterize the power, performance, lifetime, etc. due to running the application and the area due to the physical configuration. Bolded tools are included in the example presented in Chapter 3.1.

than having to trust vendors' assurances of those metrics. When code bases are large and complex, it also enables researchers to focus on specific portions without having to understand all of the code.

In addition to the tool flow, I present an extension of gem5 that enables high-level performance simulation of the RISC-V ISA [71] called "RISC5" [72]. Prior to its development, RISC-V simulation methods were limited to high-detail but slow RTL simulation or high-speed but low-detail binary translation. Using its high-level models of memories and micorarchitecture, gem5 runs faster than RTL simulation while providing enough detail to enable RISC-V designers to simulate power, temperature and reliability. This chapter shows how RISC-V designs can be simulated with high-level models using an example execution of simulation tools to estimate the power consumption and temperature of two RISC-V designs. It then presents an extension of the flow to show how high-level modeling can enable design space exploration for complex systems and improve the power, performance, and area of a heterogeneous RISC-V system with an application-specific accelerator by co-designing its components.

3.1 Tool Flow

This chapter details an open-source tool flow, illustrated in Figure 3.1, that can be used to perform a complete high-level simulation of a design. The flow includes the gem5 simulator [53], which contains microarchitecture and memory performance models; McPAT [54], which contains models to estimate power and area for microarchitecture and memory at the functional unit level; HotSpot [11, 74], which models temperature including package and die; VoltSpot [10], which models voltage noise; and finally OldSpot [58], which models lifetime and was discussed in greater detail in Chapter 2. I also present example results from this flow running through HotSpot using two RISC-V designs: one configured from a popular design called Rocket Chip [75]

and one based on the Berkeley Out-of-Order Machine (BOOM) [76]. RISC-V is a free-to-use, open-source instruction set developed at Berkeley that will be discussed in greater detail in Chapter 3.2.

3.1.1 The gem5 Simulator

Gem5 [53][1] is a performance simulator that models several different kinds of CPUs at the micorarchitecture level, the entire memory hierarchy from private and shared caches to main memory, and I/O. It is free and open-source with a BSD-like license to facilitate collaboration between researchers in academia and industry and its modular design enables focus on specific sections of its code without requiring understanding of the entire code base. With its support of a wide variety of instruction sets including popular ones like x86 and ARM, it can be used to model realistic workloads that might appear on real systems. Finally, it supports many features that reduce simulation overhead not present in other simulators, including several CPU models of varying levels of detail that can be freely switched during simulation, system-call emulation (SE) and full-system (FS) execution modes, and saving and loading system state.

Simulations can be performed using four CPU models: `AtomicSimpleCPU`, which models a single-cycle CPU and ignores memory timing; `TimingSimpleCPU`, which also models a single-cycle CPU but stalls to wait for memory requests; `MinorCPU`, which models a four-stage, in-order pipeline; and `DerivO3CPU`, which models an out-of-order CPU.[2] If greater functionality, such as simulation of application-specific accelerators, is needed, gem5 also supports the creation of new models using a combination of C and Python to describe the internal behavior and external interface and parameters, respectively. Gem5 supports switching between its CPU models on-the-fly during simulation either when the simulation reaches a provided number of instructions or by using a special instruction in the simulated program. By doing this, the simulation can be "fast-forwarded" through uninteresting regions using a fast, low-detail model such as `AtomicSimpleCPU` and then switching to a higher level of detail using `MinorCPU` or `DerivO3CPU` during regions of interest. Similarly, gem5 can save a checkpoint containing architecture, execution context, and memory state at any time during simulation, also using either a specified instruction interval or special instructions in the executed binary, and resume from any of them later with a different CPU model. This enables techniques such as phase analysis [73, 77] to statistically sample a program and create performance and power profiles without simulating the entire workload.

Gem5's accuracy has been evaluated against a dual-core ARM Cortex A9 in [51] using several scientific, media, and engineering benchmarks to represent hardware and workloads that might be found on mobile devices. Across all benchmarks measured, gem5 was shown to have between 1.39% and 17.95% error in

[1] Gem5 is available at `http://www.gem5.org`.

[2] These models used to be aliased as, respectively, "simple," "timing," "minor," and "detailed," but these aliases have been removed.

Table 3.1: Example Simulation Flow Design Parameters [72]

	Rocket Chip [75][3]	**BOOM** [76][4]
Process Size (nm)	45	45
Frequency (MHz)	1500	1500
Main Memory Size (MB)	4096	4096
Inst. Cache Size (kB)	16	32
Inst. Cache Associativity	4	8
Data Cache Size (kB)	16	32
Data Cache Associativity	4	8
L2 Cache Size (kB)	2048	512
L2 Cache Associativity	8	8

reporting execution time. The authors determined that benchmarks with high error in gem5's execution time report tended to have high L2 miss rates, indicating that gem5's DDR memory model is simplistic and inaccurate. This conclusion was supported by increasing the input sizes of benchmarks that had low L2 miss rates until those rates increased. With low input sizes and low miss rates, gem5's error remained relatively low and constant. When the input size increased high enough to introduce L2 cache misses, the simulation time error greatly increased. Even with this source of error, however, gem5's accuracy remains within acceptable limits to enable design space exploration [51].

In the example flow shown in Figure 3.1, gem5 simulated two RISC-V systems configured from Rocket Chip [75] and BOOM [76] using a 45nm process whose parameters are summarized in Table 3.1. RISC-V, and its implementation in gem5, will be further discussed in Chapter 3.2. The two designs were simulated using `MinorCPU` and `DerivO3CPU`, respectively, in SE mode running a one-million-instruction region of the *libquantum* SPEC CPU2006 benchmark [78] that was chosen using SimPoint [73].

3.1.2 Estimating Power and Area with McPAT

McPAT [54][5] is an integrated Multicore Power, Area, and Timing simulation tool that includes models for microarchitectural units such as functional units, networks on chip, and peripherals. It also includes CACTI-P [79] to create models for SRAM memories. These models are intended to be universal and able to describe any system in a similar manner to gem5, enabling use in a wide variety of settings and removing the requirement of power characterization using real hardware as in [55], albeit at the cost of accuracy. By including technology parameters from 180nm to 22nm and not relying on technology scaling to estimate them and by modeling both leakage and dynamic components of power, McPAT improves accuracy over

[3]Rocket Chip parameters were taken from default values found in the repository at `https://github.com/freechipsproject/rocket-chip` using commit hash `73e9508`.
[4]BOOM can be found at `https://github.com/ucb-bar/riscv-boom`, but its parameters were taken from [76].
[5]McPAT is available at `http://www.hpl.hp.com/research/mcpat/`.

Table 3.2: McPAT Results Summary [72]

	Rocket Chip [75]	**BOOM** [76]
Core Area from McPAT (mm^2)	4.18	1.37
Core Area without L2 cache (mm^2)	0.29	0.77
Power (W)	1.02	5.11

earlier models such as Wattch [80]. Its area estimates can be used to estimate a floorplan using a tool like ArchFP [70], enabling further simulation for quantities that depend on physical layout like temperature.

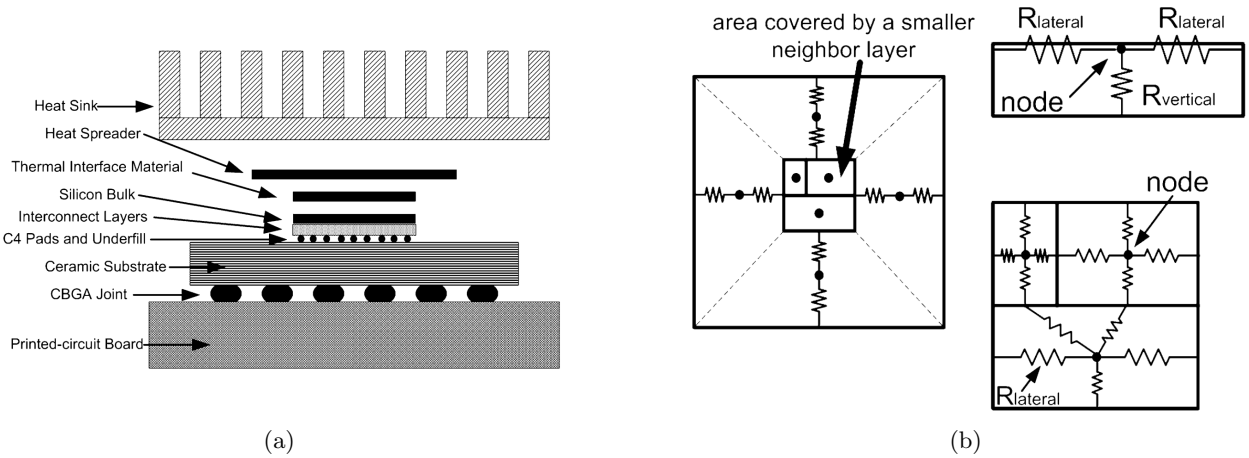
Unfortunately, McPAT is known to have some level of inaccuracy beyond what is caused by using generic, ISA-agnostic microarchitecture models to allow general use for power modeling. The authors of [50] identify several sources of error that can cause inaccurate calculations: modeling abstraction errors, modeling assumption errors, input or user errors, and coding errors. Many of these errors are fixed by the work in [50],[6] but even with these fixes the authors note that McPAT's estimations will still be inaccurate. This can be further improved by calibrating its internal models using data from a real design, but this must be done at a fine granularity or accuracy will not be significantly improved.

McPAT, with the fixes from [50] applied, is calibrated for a 45 nm process in the example flow using the area of BOOM estimated from a floorplan photograph in [76], excluding "uncore" areas not modeled by gem5, which was designed using 45 nm. These calibrations were applied to McPAT's results for Rocket Chip to create area estimations for it. The calibrated results, along with McPAT's power estimation, are summarized in Table 3.2. These results were then used with ArchFP to create a floorplan for each design, which can be seen in Figure 3.3. An interesting observation about the two designs is that BOOM is less than half the size of Rocket Chip even though it contains significantly more logic to handle out-of-order operation. This difference can be attributed to the larger L2 cache that Rocket Chip has. As shown by Table 3.2, when the L2 caches of both designs are removed from the area estimation, BOOM becomes over twice the size of Rocket Chip, as one might expect. BOOM also consumes greater power than Rocket Chip due to the overall higher activity caused by this logic and higher utilization of the execution units enabled by out-of-order execution.

3.1.3 Simulating Temperature with HotSpot

HotSpot [11][7] is a compact thermal modeling tool for computing the steady-state or transient temperature of a chip given a power trace acquired from a tool like McPAT. High temperatures and thermal hot spots not only exacerbate aging and limit lifetime, they also directly harm performance by reducing carrier mobility and increasing interconnect resistivity, increase leakage power, and negatively effect package reliability. By

[6]These fixes are available at `http://vlsiarch.eecs.harvard.edu/mcpat`.
[7]HotSpot is available at `http://lava.cs.virginia.edu/HotSpot/`.



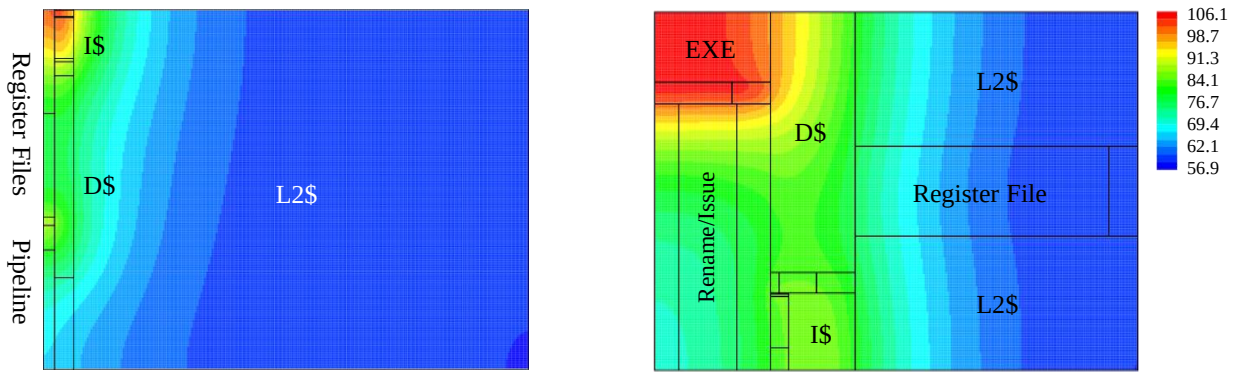
(a) (b)

Figure 3.2: Illustration of HotSpot's compact thermal model [11]. HotSpot contains models for every layer of a chip, shown in (a), and divides the die into sections that are connected using thermal resistances (b) and thermal capacitances to ground (not shown). The resulting circuit is solved using the input power trace to provide the temperature of each component.

creating a compact thermal model using the well-known duality between thermal phenomena and electrical components, HotSpot can quickly evaluate the temperature of a chip and avoid complex and slow numerical analyses that hinder architectural research and design optimization.

Figure 3.2 shows an illustration of how HotSpot's compact thermal models are formed. HotSpot includes models for every layer of a chip, including the package, silicon, interconnect, etc. (Figure 3.2(a)). The die is divided either by functional unit, into a grid, or into sub-grids within each functional unit. Each cell contains a node of an RC circuit that is connected to the nodes of adjacent cells, both within layers and between them, using resistors (Figure 3.2(b)) based on the the sizes of the cells and thermal properties of the material. Each node also contains a capacitance similarly determined that is connected to ground. To simulate the cooling solution, HotSpot includes a simple lumped thermal resistance model that represents natural convection and also supports the use of a more complex model that contains detailed package information. These components form an RC circuit that can be solved using standard techniques to rapidly evaluate the temperature of the chip.

The example flow uses the floorplan generated by ArchFP along with the power consumption computed by McPAT to create a thermal map for each design using HotSpot, shown in Figure 3.3. BOOM's generally higher temperature corresponds with its higher power consumption and lower area; in particular, the thermal hot spot over its execution units shows the higher utilization enabled by its out-of-order execution. Combined with the next-highest temperature belonging to the instruction cache, this suggests that the computation is performing many low-latency instructions such as arithmetic instructions and not waiting much for memory accesses. Figure 3.3 shows that the relatively high temperature of the data caches in both designs is mostly



(a) (b)

Figure 3.3: Heat maps of the temperatures of Rocket Chip [75] (a) and BOOM [76] (b). Blue areas indicate relatively cooler temperatures and red areas indicate relatively warmer temperatures.

due to temperature bleeding from adjacent high-temperature units, contributing to the conclusion that there are few data accesses in the simulated region.

3.1.4 Computing Voltage Droop with VoltSpot

VoltSpot [10][8] is a power delivery network (PDN) model for voltage noise simulation. It utilizes a fine-grained grid model capable of capturing the relationship between PDN design details (such as pad count and placement) and supply-voltage noise. As illustrated by Figure 3.4, the die is divided into two grids that represent V_{DD} and ground and connects to an RLC circuit representing the PDN at grid locations containing pads. In between these two grids are ideal current sources representing the load created by device activity and leakage, whose current values are $I = {}^{P}/_{V_{DD}}$, where P is the total power consumed by the region of the chip represented by the current source. Combined with other architecture-level tools, VoltSpot provides a platform for investigating the effect of application- and time-dependent noise, evaluating design and run-time noise mitigation techniques, estimating vulnerability to aging, and performing multi-dimensional design space exploration that includes I/O-pad allocation. It has been used to show the impact of power and I/O pad count on noise amplitude and event frequency, to compare noise mitigation techniques, and to study the effects of pad failures on noise. Because VoltSpot requires a cycle-by-cycle power trace to create accurate results, which drastically increases simulation time for earlier tools in the chain, it is not included in the example RISC-V tool flow.

[8]VoltSpot is available at `http://lava.cs.virginia.edu/VoltSpot/`.

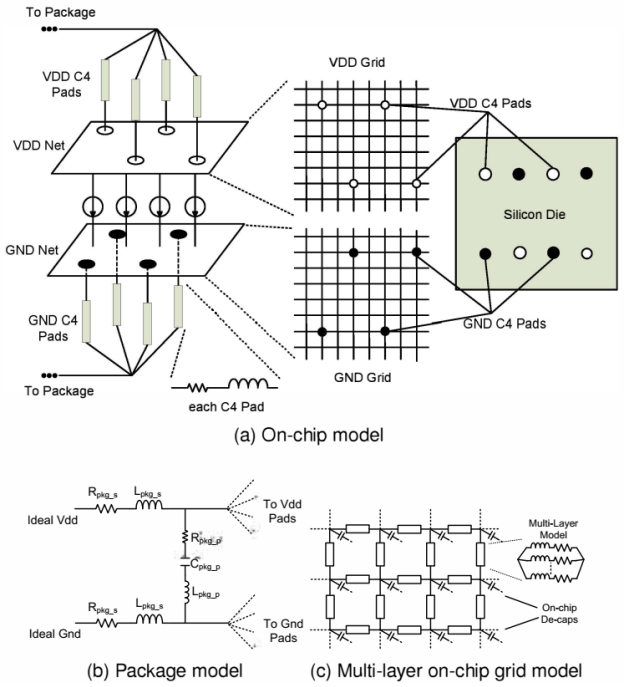


Figure 3.4: VoltSpot modeling mechanism for voltage noise [10]. VoltSpot divides the die into V_{DD} and ground grids that are connected by the RLC power delivery network model and ideal current sources representing circuit loads.

3.2 RISC5: An Implementation of the RISC-V ISA in gem5

In addition to open-source simulation tools, it is also important that there be available open-source instruction sets and hardware to allow the same kinds of collaboration between academic and industry researchers that tools like gem5 enable. To that end, the RISC-V ISA [71] was introduced to create such an instruction set while still being competitive with industry standards like ARM and x86. Other open ISAs exist, but mistakes in their design, such as branch delays in SPARC and MIPS, have caused them to be unable to compete and lose popularity [81]. RISC-V is designed in a flexible, modular fashion that allows researchers to easily develop improvements and extensions while also allowing hardware designers to focus only on functionality they need, improving its usefulness over earlier open-source ISAs. Furthermore, it contains no specification of the microarchitectural implementation it should run on, allowing development of both in-order [75] and out-of-order [76] designs.

Several RISC-V designs [75, 76] are implemented using Chisel [82], a hardware construction language developed as library for the Scala programming language [83] that improves the flexibility of hardware design through generators created using concepts used in object-oriented and functional programming models. It

Table 3.3: Simulation Features and Compatibility [72]

Feature	**gem5** [53]	**Chisel** [82]	**spike** [84]	**QEMU** [85]	**rv8** [86]
Binary translation	✔		✔	✔	✔
Checkpoints	✔			✔	
Multicore simulation	✔[10]	✔	✔	✔	
Performance statistics	✔	✔[11]	✔[12]	✔[12]	✔[12]
RTL simulation		✔			
System call emulation	✔	✔[13]	✔[13]	✔	✔
ASIC synthesis		✔			
FPGA tools		✔			
Phase analysis	✔				
Stats-based tools	✔	✔[11]			

can generate Verilog code that can be compiled into an RTL simulation[9] of the design, mapped to an FPGA, and used in ASIC design flow.

Existing RISC-V simulation tends to fit into two categories: detailed RTL simulation as discussed above or binary translation using emulators like spike [84], QEMU [85], and rv8 [86] (Figure 3.5). Using high-level models of architectural units and memory hierarchy, gem5 is capable of bridging the gap between these two categories by providing accurate results at faster speeds than RTL simulation does, facilitating exploration of large design spaces created by high complexity in modern designs [48]. Table 3.3 shows a comparison of simulation features between several RISC-V simulators, including gem5, a Chisel-generated RTL simulator, spike, QEMU, and rv8. Notably, gem5 is the only tool that is capable of utilizing phase analysis [73, 77] by taking advantage of its ability to switch level of detail during simulation to save simulation and restore simulation state. For more information on gem5's features, see Chapter 3.1.1. On the other hand, tools such as spike, QEMU, and rv8 are capable of much greater performance with binary translation than gem5 because they do not include microarchitectural models, making them ideal for testing software functionality and estimating its performance on RISC-V. This drawback renders it impossible to perform hardware design with them, whereas gem5 is capable of informing RISC-V hardware design with its microarchitecture and cache models through high-detail performance statistics that can be used with other high-level models. This can be used to further improve designs created using Chisel, which is capable of producing RTL-level simulations that provide performance information specific to a design that is necessary for hardware optimization.

[9]Earlier versions of Chisel were capable of directly creating a C++ RTL model of a design, which is used for validation in Section 3.2.2. This functionality has since been removed in chisel3 in favor of compiling Verilog code for simulation.
[10]Gem5 supports multicore simulation in both SE and FS modes, but RISC-V does not support it yet as of this writing.
[11]Chisel can only output performance statistics if a design has means of counting and outputting them.
[12]The simulated program is responsible for tracking performance counters available in RISC-V.
[13]System call emulation is supported via the RISC-V proxy kernel.

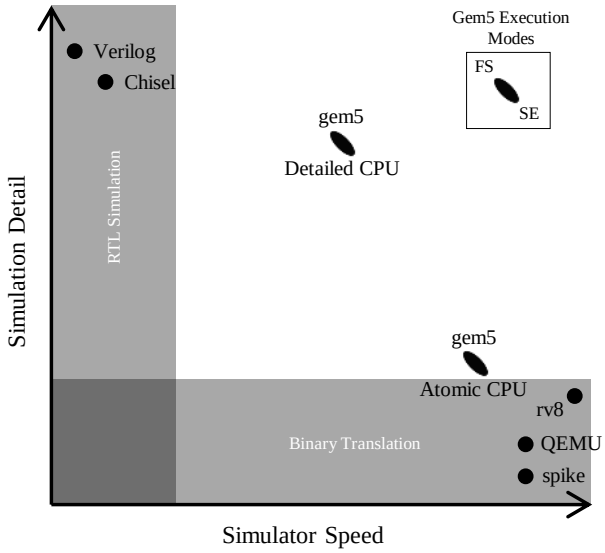


Figure 3.5: Illustration of the tradeoff between simulation accuracy and speed [72]. Other than gem5, most simulators most simulators achieve either high speed or high accuracy. With its high-level models of execution units and customization, gem5 can achieve some degree of both.

3.2.1 Implementation Details

An advantage RISC-V has over earlier open architectures is its modularity, which it achieves by dividing its instructions into a base instruction set and several extensions. It further allows customization of address and data width, referred to as `XLEN`, supporting 32-, 64-, and eventually 128-bit versions of each module [71]. A RISC-V implementation can identify which modules it supports with an ISA string that comprises of the letters "`RV`" followed by `XLEN` and then a list of letters representing the extensions that are included. Software is able to read this information from a dedicated register to determine compatibility with hardware [87]. The only mandatory module is the base instruction set, `I`, which defines basic integer arithmetic, memory, and control operations, although it can optionally be substituted by the `E` instruction set that defines a reduced architecture intended for low-power embedded systems. Standard extensions include the integer multiply and divide extension, `M`; the atomic memory extension, `A`; and the single- and double-precision floating point extensions, `F` and `D`. These form the "general-purpose" RISC-V ISA and, together with the base instruction set, are referred to using the letter `G`. Additional extensions include quad-precision floating point arithmetic, `Q`; compressed instructions, `C`; and support for dynamically-translated languages (e.g. Scala), `J`. Gem5 currently supports `RV64GC` based on their definitions in [71], which includes 64-bit instructions in the base instruction set, standard extensions, and compressed extension. At the time of this writing, gem5 also currently only supports executing RISC-V in SE mode with a single thread, so only nominal support for the privileged

Table 3.4: Invalid Division Operations [71]

Operation	Division by 0	Overflow
Signed Divide	-1	-2^{XLEN-1}
Signed Remainder	Dividend	0
Unsigned Divide	$2^{XLEN}-1$	N/A
Unsigned Remainder	Dividend	N/A

ISA [87] exists. This section includes details about the implementation of each module followed by validation against a Chisel simulation of Rocket Chip.

Base 64-bit Instruction Set (`RV64I`)

Because RISC-V instructions share many similar behaviors to those of other ISAs such as MIPS [88], much of the code gem5 uses to implement those ISAs was adapted for RISC-V. When instruction definitions in [71] were not sufficient to create their implementations in gem5, typically due to required knowledge of gem5's internal behavior, their definitions were created by referring to their analogues in [88] and inspecting gem5's code (usually for MIPS or Alpha). For example, the RISC-V `fence` instruction is similar to the MIPS `sync` instruction, so the implementation of `fence` in gem5 is based off its implementation of `sync`. This implementation does not yet include `fence`'s flags for specifying how memory and device accesses should be ordered because gem5 does not contain that behavior, so instead `fence` synchronizes all operations. The only instructions not implemented from `RV64I` are `uret`, `sret`, and `mret`, which return from traps and exceptions that typically execute with elevated privilege [87]. Privilege levels, traps, and exceptions do not exist in gem5's SE mode, so these instructions are unnecessary until full-system simulation is implemented. If one is encountered, the simulation halts.

Integer Multiply Extension (`RV64M`)

The multiply extension does not add additional instruction formats or new instruction behavior, since invalid operations in integer arithmetic do not cause exceptions. The only invalid operations are division by zero and division overflow ($-2^{XLEN-1} \div -1$), which produce special results that are enumerated in Chapter 6.2 of [71] and reproduced in Table 3.4. Rather than trusting the host system's implementation of these operations to produce the required results, gem5 explicitly checks for these cases and stores the correct value in the destination register.

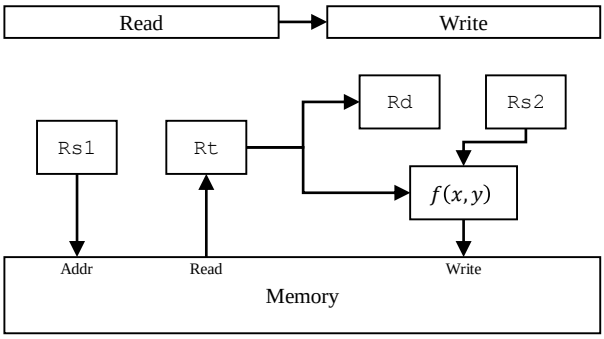


Figure 3.6: Illustration of the execution of an atomic memory operation in gem5 [90]. During the read micro-op, the memory address is read from `Rs1` and the data from that address is stored in the special AMO register, `Rt`. Then, during the write micro-op, the data stored in `Rt` and the other source register, `Rs2`, are operated on using $f(x, y)$ and the result is stored into memory.

Atomic Memory Extension (`RV64A`)

RISC-V includes two different ways of executing atomic operations [71]: load-reserved/store-conditional (LR/SC) instruction pairs and atomic read-modify-write (RMW) instructions. LR/SC sequences, which are similar to load-link/store-conditional sequences in MIPS [88], are a load-store instruction pair in which the load instruction reserves a memory location and the store instruction releases it if the location hasn't been written to by another thread since the reservation was made. Indication of a successful LR/SC sequence is stored in a destination register. This can be used for compare-and-swap, a common method of testing for atomicity in concurrent applications [71]. The atomic read-modify-write instructions atomically load a value from a memory location, store it in a register, perform an operation on it, and store the result back to memory. All of these instructions include flags to acquire and/or release a memory location according to *release consistency* [89]. Release consistency is a weak memory model that ensures that the results of a writer's operations on a memory location are seen by a reader if the reader acquires that location after the writer releases it.

Only two instruction sets have analogues to RISC-V's atomic instructions [88], neither of which are present in gem5. While the definition of these instructions in [71] is sufficient for implementing the modify step of RMW, each instruction requires two memory accesses (read and write). Since gem5 does not support multiple memory accesses per instruction when simulating memory with timing, splitting each atomic memory instruction into two micro-ops, one to read from memory and one to write the result back to memory, enables support for these instructions. The operation that performs the modification of the value read from memory

is arbitrarily the second micro-op. The process of executing an atomic RMW instruction is illustrated in Figure 3.6.

In order to enable the write micro-op of each atomic memory instruction to keep track of the data loaded from memory by the read micro-op, a new integer register is added to the implementation, denoted by `Rt` in Figure 3.6. This register is separate from the main register file and only used for storing a value loaded by the first micro-op of an atomic memory instruction. It is not accessible for use as an operand for any instruction. Each micro-op is marked with an `ACQUIRE` or `RELEASE` flag to indicate if the RMW instruction is acquiring and/or releasing a memory location. Load-reserved and store-conditional instructions are similarly flagged, and all instructions in `RV64A` can have either flag, both flags, or neither flag. Because gem5 only supports single-threaded simulation for RISC-V, the actual atomicity and memory consistency of the implementations of these instructions cannot be empirically verified.

Floating Point Extensions (`RV64FD`)

RISC-V includes two standard extensions for single- and double-precision floating point arithmetic (`F` and `D`, respectively). Except for the sizes of instruction operands and presence of instructions to convert between single- and double-precision values, these two extensions and the details about their implementations are the same. Generally gem5 depends on the host system's implementation of floating point arithmetic to match that of RISC-V. But since RISC-V conforms to the IEEE-754 2008 floating-point standard [91] while the x86 machine used for development did not, there are some cases where explicit checks to source register values and computation results need to be made to ensure that exceptions were being thrown correctly. This normally only applies to operands with value 0, `NaN`, or ∞.

Another important difference between the two standards is the rounding modes that are present. The IEEE 2008 floating-point standard defines five modes for rounding inexact results (*roundTowardPositive*, *roundTowardNegative*, *roundTowardZero*, *roundTiesToEven*, and *roundTiesToAway* [91]), while the development machine only has four (*roundTiesToAway* is missing). Additionally, it trusts that the host's implementations of the existing rounding modes are the same as RISC-V's definitions of them and makes use of the C floating-point environment library to control them. Since *roundTiesToAway* is not a standard part of that library, gem5 will halt simulation if an attempt is made to use it.

Other minor differences include floating point classification, where the development machine ignores sign while RISC-V does not, and exceptions thrown for producing inexact results. In the former case, the C `fpclassify` function is combined with explicit checks for information it does not provide, such as the sign of non-`NaN` values and whether a `NaN` is signaling or quiet, to produce results. In the latter case, gem5 uses the host's inexact result exception behavior to simplify the implementation.

Gem5's floating point behavior was verified against results from spike and Chisel simulations. While spike emulates RISC-V instructions on a host system like gem5 does, the Chisel simulator performs simulations at RTL level and so implements its own floating point arithmetic. The results of the floating-point computations between the two simulators were the same, but they disagreed on how exceptions should be generated, sometimes not conforming to the specifications in [71] and [91]. For example, spike may throw a divide-by-zero exception when performing a single-precision division by zero, correctly storing ∞ in the result register, but the Chisel simulation may not. In these cases, gem5's implementation conforms to the specifications without regard to the other simulators. In the above example, gem5 will throw a divide-by-zero exception.

Compressed Extension (`RV64C`)

Unlike the standard extensions, the compressed extension introduces different instructions depending on the value of `XLEN`. Gem5 implements the 64-bit version, `RV64C`. Additionally, rather than introducing new functionality, this extension creates additional encodings for frequently-used existing instructions that are half as long as standard encodings. All compressed instructions introduced can be rewritten using standard 32-bit instructions except for `C.JALR`, which has only a minor difference from its standard cousin `JALR` in that it can jump to addresses on two-byte boundaries rather than four-byte ones. Compressed instructions are only compatible with systems that implement the standard extensions [71].

This extension provides similar functionality to RISC-V as the Thumb instruction set does for ARM [92]. Thumb is comprised of half-length versions of popular ARM instructions and provides a restricted view of the register file like RISC-V compressed instructions do. The main difference between RISC-V's compressed extension and Thumb is that Thumb instructions can only be used when in a Thumb state, during which regular instructions cannot be used. This prevents interleaving compressed instructions with regular ones, which RISC-V allows. Additionally, the restricted set of registers enabled in Thumb is global to all Thumb instructions, whereas RISC-V's compressed extension has different restrictions for different types of instructions.

The addition of compressed instructions required two changes. First, special source and destination register operands were added to account for the smaller subset used by compressed instructions that reference `Rs1'`, `Rs2'`, and `Rd'`. Since these operands refer to registers `x8-x15` using indices 0-7, the "real" register indices can be derived by adding 8 to the compressed register indices. Second, the decoder state machine needed to be rewritten to account for multiple instruction widths. Without this extension, all instructions are 32 bits; the decoder only required a simple state machine that advances the PC by 4 bytes every time an instruction is fetched. The addition of 16-bit instructions requires some additional predecoding to determine instruction length and storage of instruction bytes in case a 32-bit instruction crosses a 32-bit boundary.

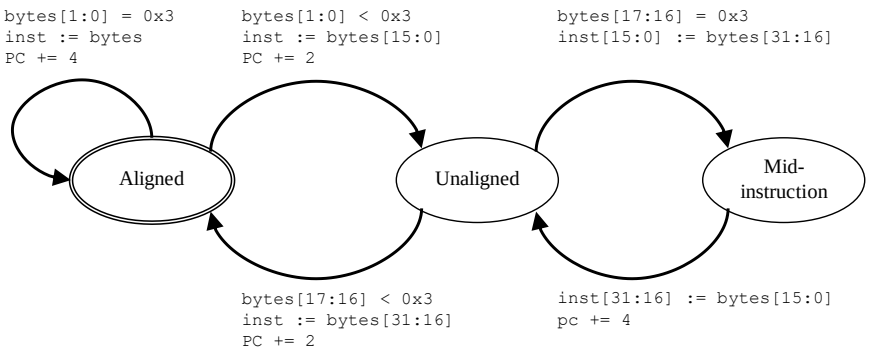


Figure 3.7: New predecoder state machine used by gem5 with the compressed instruction set [90]. `bytes` represents the bytes that have been fetched and are being predecoded and `inst` represents the instruction to be decoded. When aligned, the two least-significant bits of `bytes` indicate the length of the instruction. When unaligned, bits 17 and 16 are used instead. New bytes are fetched whenever entering the Aligned or Mid-instruction states.

The new state machine is illustrated in Figure 3.7. It begins by assuming the first instruction is aligned on a 32-bit boundary. If that instruction's two least significant bits are `0x3`, then it is a standard-size instruction, it is decoded, and the PC is incremented by 4 bytes. Otherwise, only the least significant half is decoded and the PC is only incremented by two bytes. When the PC is not aligned on a 32-bit boundary, then the bits 17 and 16 of the most recently-fetched 32-bit instruction word are checked to see if they form a compressed instruction. If they do, then the PC is incremented by two bytes and must be aligned; the predecoder returns to its initial state. Otherwise, it combines the upper 16 bits of the instruction word with the lower 16 bits of the next word and decodes the result, incrementing the PC by 4 bytes and returning to the previous state.

3.2.2 Validation of RISC5

Selected benchmarks from the SPEC CPU2006 suite [78] are simulated on gem5, a Chisel-generated C++ simulator,[14] and on a RISC-V soft core on an FPGA.[15] In order to keep track of performance statistics shown in Figure 3.8, the Chisel design was modified to add additional performance counters and each benchmark's source was modified to read and print their values. To improve parity with gem5, which would not accumulate these statistics while emulating system calls, the Chisel design was also modified to pause these counters during execution of privileged code. Gem5 and the simulator were both run on a four-core machine running at 3.7 GHz with 32 kB, 256 kB, and 10 MB of L1, L2, and L3 cache, respectively, and 32 GB of main memory. Since Rocket Chip also includes a flow for configuring a Xilinx Zynq FPGA on a Zedboard development kit

[14] As noted previously, chisel3 no longer generates a C++ RTL description of a design, preferring to generate only Verilog code and allowing external tools to compile it for simulation. This validation was performed using chisel2, which does have this functionality.

[15] Rocket Chip and chisel2 versions used to generate the simulator are, respectively, `73e9508` and `b18e98b`.

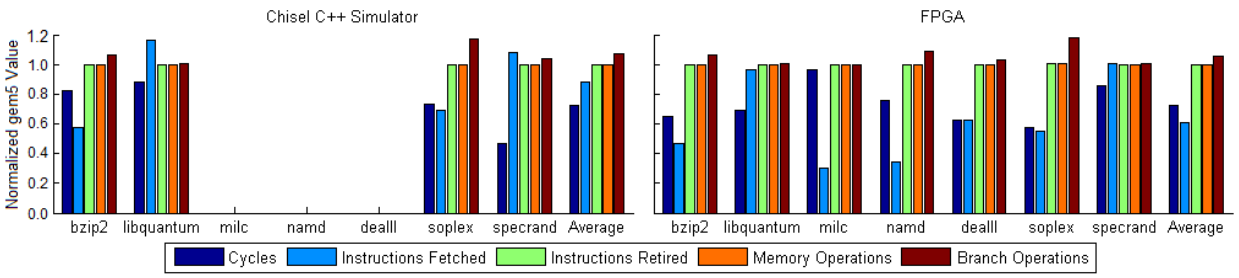


Figure 3.8: Validation of gem5 performance statistics against the same values from Chisel simulation (left) and FPGA (right) [72]. Some benchmarks are omitted due to excessively long simulation times. Gem5 is generally more accurate for statistics that accumulate later in Rocket's pipeline than for earlier.

as a RISC-V core, the Chisel design was also mapped to FPGA for further comparison. The design on the FPGA has a clock rate of 25 MHz and up to half the development board's 512 MB of DRAM.

Figure 3.8 shows each performance statistic obtained from gem5's output normalized to the value extracted from the C++ simulator and FPGA. It shows that gem5 is accurate in the number of instructions retired, number of memory operations performed, and number of branch instructions executed, but less so in the number of cycles it took to complete each benchmark and number of instructions fetched. This is due to differences between the microarchitectures and memories modeled by gem5 and implemented by Chisel. In particular, the branch prediction may be different, causing a significant difference in the number of instructions fetched and cycles executed while not affecting the number of retired instructions or memory operations. This also accounts for the slight difference in branch operations.

The FPGA, on average, took about 26.5 times less time to execute benchmarks than gem5 did while the Chisel simulator took, on average, about 32 times longer than gem5 to perform each benchmark. As a result, several benchmarks took too long to execute and produce data, so they are excluded from Figure 3.8 for C++. This slowdown is due to the very high level of detail of the Chisel simulator, which enables the capability for very accurate simulation but adds significant overhead. Gem5's abstraction of low-level signals using CPU models reduces overhead and allows it to run faster.

3.3 Co-optimizing CPUs and Accelerators with Constraints

This section demonstrates the use of pre-RTL simulation along with RISC5 to show that it is insufficient to design general-purpose and application-specific units of an SoC separately and explore the benefits of co-designing them. Because electronic systems often make use of several accelerators to improve power or performance when executing certain tasks, it can be easier to design the system in a modular fashion; separating the designs of each accelerator and the CPU from each other can reduce design complexity. The

costs of integrating these units with each other, such as interconnect or bus latency, is left up to software like device drivers to manage [93]. Unfortunately, this can make these costs difficult to estimate at accelerator design time [93]. The authors of [93] show that latencies caused by data movement and cache coherency management can account for up to 40% of the total runtime of an accelerator and have a significant impact on design optimization. By ignoring these in designing an accelerator, a hardware designer can be misled into making deceptively suboptimal design choices.

When designing devices with power, performance, or area constraints, it may no longer be possible to design each component in a modular fashion. Doing so may cause the overall system to become too large or consume too much power, failing to meet its design constraints. But removing resources from one unit in order to make way for another unit may create an overall suboptimal design. In order to determine the best distribution of resources given constraints on some design metrics, the entire system must be considered simultaneously. By co-optimizing the accelerator and CPU, the overall performance of an area- or power-constrained system can be significantly improved. The work presented in this section is also published in [94].

3.3.1 High-level Accelerator Modeling Tools

One of the main tools used in this work, Aladdin [52], is a pre-RTL accelerator power, performance, and area estimation tool that transforms a kernel specification written in C into an estimation of the hardware necessary to implement it given design constraints and input. It includes power, performance, and area characterizations for several types of logical units and makes use of CACTI-P [79] to estimate these metrics for memories. The authors compare Aladdin's results against those computed from RTL synthesis and simulation and show that Aladdin can achieve 0.9%, 4.9%, and 6.6% error in performance, power, and area, respectively while eliminating the need for slow RTL design and simulation. A weakness of Aladdin, as pointed out by [93], is that it does not account for external effects such as memory latency that will affect its design.

After pointing this out, the authors of [93] improve Aladdin by combining it with gem5 to create a tool called gem5-Aladdin, using gem5's built-in cache and memory models to compute the overheads of transferring data between the accelerator and shared memory and of maintaining cache coherency with the accelerator. Using gem5-Aladdin, the authors show that these costs reduce the amount of parallelism that an accelerator can support due to increased data transfer overheads caused by increased memory sizes on the accelerator. By reducing the amount of parallelism in the accelerator, its overall energy-delay product (EDP) decreases. The authors also explore the usage of caches rather than scratchpad memories on the accelerator to improve performance for workloads with irregular memory access patterns that DMA engines typically perform poorly

with. They show that considering all of these effects in designing an accelerator can improve its EDP by up to 7.4×. Even so, the authors of [93] assume that the only significant portion of the workload is performed by the accelerator and do not account for constraints that might affect the design of the entire system.

Another tool for simulating accelerated workloads is the Platform for Accelerator-Rich Architectural Design and Exploration (PARADE) [95]. Like gem5-Aladdin, PARADE integrates with gem5 to enable simulation of a full workload that includes accelerated segments. To model an accelerator, existing tools like Aladdin or RTL simulation can be used or the accompanying toolchain can be used to convert a high-level description of the accelerator kernel into RTL to be simulated. Unlike gem5-Aladdin, PARADE has a three-stage programming model where all of the accelerator input is initially loaded into scratchpad memories, the accelerator operates on that data, and finally all of the output data is loaded back into last-level cache. This limits the designs of accelerators it can model and can inflate the cost of data transfer by forcing the accelerator to wait until all of its data is available instead of allowing it to operate on the data as it arrives. PARADE is also capable of modeling composable accelerators that directly communicate with each other, whereas gem5-Aladdin only enables multiple discrete accelerators that can exchange data with shared memory. The authors of [95] use PARADE to evaluate the power and performance improvements that can be gained by using accelerators for several workloads and analyze the benefits of using composable rather than discrete accelerators, but, like gem5-Aladdin, consider only the work that is being accelerated and do not include constrained optimization.

3.3.2 Modeling Infrastructure and Tool Flow

To show the importance of co-optimizing the general-purpose cores alongside the application-specific accelerators in an SoC, power, performance, and area are simulated using several systems that each include a single RISC-V CPU core and a unit that is tailored to accelerate a portion of the workload in a benchmark from CortexSuite [96]. CortexSuite includes workloads meant to model functions commonly performed by the human brain in order to create a baseline for machine learning applications. Each benchmark is modified to replace a portion of its workload with an invocation of an accelerator that performs that function. This accelerator is designed using a C description of the workload portion to be analyzed by Aladdin. Each benchmark is compiled for RV64GC and executed on a modified gem5-Aladdin that has enabled compatibility for RISC5. Gem5 is configured to simulate a four-stage, in-order RISC-V CPU core that communicates with Aladdin through the interface provided by gem5-Aladdin and executes the general-purpose portion of the workload. When an accelerator invocation is encountered, gem5-Aladdin intercepts it and invokes Aladdin to analyze the accelerator and produce its power, performance, and area computations for the input.

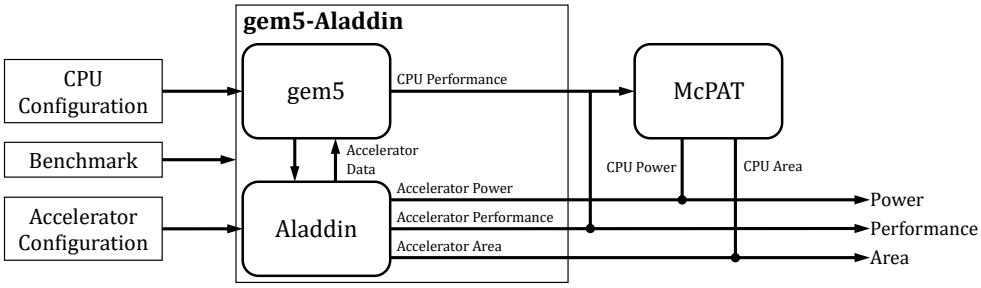


Figure 3.9: Illustration of the tool flow used to simulate the power, performance, and area of the single-core RISC-V system with one accelerator that is tailored to accelerate a portion of each benchmark's workload [94].

Gem5-Aladdin includes enhancements to Aladdin that trace the data flow through the accelerator and report the result of its computation back to gem5.

Aladdin's analysis consists of the construction of a dynamic data dependence graph (DDDG) from an instruction trace of the accelerator kernel using an ISA-independent intermediate representation [97]. Nodes of this graph represent computations in the accelerator kernel and edges represent data dependencies between them. After its initial construction, the DDDG is transformed to represent an idealized version of the kernel algorithm using optimizations such as removal of dependencies on loop index variables to expose parallelism and elimination of memory store operations using forwarding. After this, the DDDG is transformed two more times, adding nodes and edges to account for program dependencies such as branch decisions and user-specified hardware constraints. These constraints describe how much parallelism Aladdin is allowed to include in the accelerator, how many concurrent memory requests the accelerator's local memory can support, and the accelerator's clock frequency. In gem5-Aladdin, users can additionally include a cache with the accelerator and specify its configuration and which data structures it contains. After generating the DDDG, Aladdin can estimate the latency of the computation as well as the hardware required to perform it, producing power, performance, and area estimations for the accelerator.

After Aladdin performs its analysis, gem5-Aladdin traces the accelerator's execution so that the results can be reported back to gem5. Gem5 includes models for data transfer using both caches and DMA, which gem5-Aladdin leverages alongside its own latency computations for logic and memory in the accelerator in order to compute the overall computational delay. Once the benchmark is complete, gem5's reports on the performance statistics of the RISC-V CPU are input into McPAT. Gem5's, McPAT's, and Aladdin's results are collected to produce the overall power, performance, and area of the system as a whole. This flow, illustrated in Figure 3.9, is a subset of the flow shown in Figure 3.1 that does not include temperature, voltage noise, or lifetime simulation. It is also modified to include Aladdin's results for power and area to combine them with McPAT's for the CPU to compute overall system power and area. As Aladdin's power

and area models are characterized from a 40nm process, McPAT is configured to compute CPU power and area using that process and calibrations from [72].

3.3.3 Co-optimizing Accelerators with CPUs

This section details the simulation parameters used to characterize each benchmark with different distributions of resources between CPU and accelerator, presents the resulting design space, and then shows how optimizing the accelerator as a standalone unit and not considering the workload performed by the CPU can result in suboptimal designs. The included benchmarks from CortexSuite are described below. Other benchmarks are not included because changing the data block size affects results, they do not present a significant enough workload for the CPU, or they are too large to run with detailed simulation. Each benchmark has a portion of its workload that is executed on an accelerator dedicated to that function which is also described in this section.

rbm The *rbm* benchmark uses a restricted Boltzmann machine [98] to classify movie preferences for several users of a movie service. Because a restricted Boltzmann machine is similar to a two-layer fully-connected neural network, much of the workload is taken up by matrix multiplication. This is accelerated using a dedicated matrix-multiplication accelerator that can operate on a subset of the left matrix operand's rows simultaneously, repeating its operation on subsequent row blocks until the entire matrix multiplication is complete. The amount of area devoted to the accelerator is varied by controlling the size of the left matrix operand's row block.

pca The *pca* benchmark performs principal component analysis [99] on a set of data. The main workload of *pca* consists of the computation of a correlation matrix, which is performed using a dedicated accelerator. The amount of area devoted to it is varied by controlling the number of simultaneous columns of the correlation matrix that can be computed.

lda The *lda* benchmark performs latent Dirichlet allocation [100] to classify a given set of documents using a given set of topics. Latent Dirichlet allocation attributes words to the topics and classifies the document according to the topics its words are associated with. The accelerator for *lda* creates a maximum-likelihood estimation model of the input document for the set of topics used to classify it. The number of words in the document that can be processed in parallel can be varied to simulate controlling the devotion of resources to the accelerator. Because this benchmark makes use of pseudorandom numbers that affect its convergence rate, which in turn affect the number of times the accelerator is invoked, it is further modified slightly to use a constant seed for the random number generator and to restrict the number of iterations. This ensures the same execution for all system configurations.

Table 3.5: Simulated System Parameters [94]

Parameter	Value
Instruction set	RISC-V
Architecture	RV64GC
Process size	40 nm
Supply voltage	1 V
CPU Clock frequency	1 GHz
Instruction cache size	32 kB
Instruction cache associativity	2
Data cache size	See Table 3.6
Data cache associativity	See Table 3.6
Main memory size	4 GB

Table 3.6: Simulation Sweep Parameters [94]

Benchmark	Data Cache Range	Input Size	Accelerator Parameter	Accelerator Range
rbm	1–64 kB, {1,2,4} ways	51×31 network	Matrix row count	1–50 rows
pca	4–256 kB, {1,2,4} ways	128×64	Correlation matrix column count	1–16 columns
lda	4–128 kB, {1,2,4} ways	3 topics, 500 docs, 6907 terms	Term log-likelihood count	1–50 terms

The parameters for the RISC-V CPU used for simulation are summarized in Table 3.5. In addition to varying the parallelism of the accelerator for each of these benchmarks, the data cache size of the main CPU is varied to control the assignment of resources to it and affect the performance of off-accelerator portions of each workload. A summary of simulation parameters that are swept and the ranges for those parameters is included in Table 3.6. Because the system does not have a shared L2 cache, the accelerator shares main memory with the CPU.

Appendix C summarizes the results for all three benchmarks. It shows Pareto-optimal curves for area and performance in Figures C.1, C.2, and C.3 and for power and performance in Figures C.4, C.5, and C.6. Square- or diamond-shaped markers indicate Pareto-optimal design points for standalone accelerators, black circles indicate Pareto-optimal points for co-designed systems, and gray dots indicate suboptimal designs. The power-performance Pareto plots show accelerator designs for both small (squares) and large (diamonds) data caches, as the data cache size has an affect on both quantities in the accelerator. All three benchmarks show cases where designing its accelerator separately from the CPU could create a deceptively suboptimal design for both power and area.

As Figure C.4 shows for *rbm*, some of the Pareto-optimal points on the standalone accelerator design lie inside the Pareto-optimal curve for the overall design. The square markers show that designing the accelerator with a large data cache in mind can cause smaller designs to produce worse performance for a given power

constraint than could be achieved by exploring both design spaces simultaneously. For example, by increasing the size of the accelerator but decreasing the size of the CPU, it is possible to maintain a 40mW power constraint while improving performance by about 10%. Similarly, when designing an accelerator with a small data cache in mind, the performance of the CPU is impacted negatively enough that the overall performance of the system will suffer significantly, which is not apparent from Figure C.4(a). For example, it may appear from the diamond markers that it is impossible to achieve power consumption less than about 50 mW with a maximum 0.12-second performance constraint, but by increasing the data cache size and decreasing the accelerator size, it is possible to meet that constraint with a little over 30 mW consumed, about a 40% improvement.

The *rbm* and *pca* benchmarks behave similarly in that larger standalone accelerator designs that are Pareto-optimal tend to coincide with overall Pareto-optimal designs, but smaller designs may not. This makes sense because weak area constraints allow large accelerator sizes that enable large amounts of parallelism without significantly impacting the performance of the CPU. With tighter area constraints, however, the size of the accelerator begins to restrict that of the CPU and reduces performance of the CPU's workload enough to negatively impact overall performance. By decreasing the size of the accelerator and increasing the size of the CPU, the area constraint can be met while improving performance. For *pca*, with an area restriction of 0.4 mm^2, Figure C.2 shows that execution time can be reduced by about 25% this way. Figure C.1 indicates that *rbm* benefits less from this co-design, but can still improve execution time by about 6% with an area restriction of 0.16 mm^2 or area by about 12% with a performance restriction of 85 ms.

Co-designing the accelerator with the CPU has a greater effect on power than it does on area for *rbm* and *pca*. There is a weak relationship between data cache size and accelerator performance that translates to a small effect on accelerator power, as shown by Figure C.4(a). When the accelerator is designed with a large data cache in mind, designs with tighter power constraints suffer. On the other hand, when the accelerator is designed with a small data cache in mind, designs with tighter performance constraints suffer. In addition to the improvements to *rbm* listed previously, Figure C.5 shows that co-optimization can improve performance in *pca* with a power constraint of about 30 mW by about 40% or improve power consumption with a performance constraint of 0.08 seconds by about 66%.

Compared to *rbm* and *pca*, *lda* shows some different behaviors in Figures C.3 and C.6. Like *rbm*, whose performance begins to degrade when the system gets too big due to high data transfer latency, *lda* does not have any overall Pareto-optimal points once it gets bigger than 0.35 mm^2 or consumes more than 31.6 mW. Unlike either of the other two benchmarks, however, the performance gains for improving accelerator parallelism are much smaller, especially with physically-smaller overall designs, than the gains are for improving the general-purpose CPU performance. This indicates that the workload computed by the accelerator is a much

smaller portion of *lda*'s kernel than the other two benchmarks' accelerators portions are. As a result, when area or power are constrained, it is generally better to choose smaller accelerator designs. In this case, the design of the accelerator can be separated from that of the CPU. This is more prominent in Figure C.6 than Figure C.3, where it is clear that, except for the largest designs we simulated, the data cache size has a much larger effect on performance than parallelism in the accelerator. Even with this effect, small improvements can be made by co-designing the accelerator and CPU for certain constraints. With a performance constraint of about 9.5 seconds, reducing the parallelism of the accelerator and increasing the CPU data cache size can reduce overall area by up to 25%, although the effect on power is smaller at only about 5%.

Some of the results for *lda* are unexpected in that some of the standalone-accelerator Pareto-optimal points are overall worse in power, performance, and area than points with lower amounts of parallelism and that Pareto-optimal accelerator designs are either large or small; sizes in between are never optimal. This can be attributed to Aladdin's attempts to optimize for power and area by reusing functional units to perform computations while waiting for data transfers or for results from slower units. In the former case, due to the accelerator's interactions with the data cache in order to maintain coherency, the CPU experienced more data cache misses. Since overall performance is more strongly impacted by the CPU with *lda*, these data cache misses caused overall worse performance for these designs. In the latter case, with some designs, there were fewer opportunities to perform this kind of optimization, resulting in larger, more power-hungry accelerators that overall performed worse.

3.4 Summary

The implementation of RISC5 enables high-level simulation and design space exploration, such as the results presented in Chapters 3.1 and 3.3, for RISC-V. This allows architects who are interested in RISC-V to make improvements without needing to depend on the implementation and simulation of RTL, which present significant time and development overheads. It also allows cross-layer design space exploration using tools such as HotSpot, VoltSpot, and OldSpot by giving architects the ability to see the effects of their designs on metrics that previously were considered low-level and needed to be simulated using RTL such as temperature, voltage noise, and lifetime and needed to be simulated with RTL. Architecture design is further facilitated by the open-source nature of RISC-V and the tools in this flow, removing the burdens of obtaining licenses and allowing collaboration between researchers in industry and academia. The pre-RTL design capabilities were demonstrated by following the simulation flow in Figure 3.1 through HotSpot to create temperature maps of two RISC-V designs. A subset of this flow including gem5, McPAT, and HotSpot was modified to include gem5-Aladdin to simulate several RISC-V systems with application-specific accelerators to show how overall

power, performance, and area can be improved by co-designing the two units rather than prioritizing the accelerator. As Appendix C shows, this can improve power, performance, or area by up to 66%, 40%, or 25%, respectively.

Chapter 4

Manipulating PUF Reliability with Directed NBTI

In addition to reliability concerns, aging can have impacts on security as well. As IoT devices are given greater amounts of autonomy and interconnectivity in order to relieve their owners from having to manage them, they record more personal and sensitive information which must be protected. Unauthorized access not only allows an attacker to gain access to this data, but also control of the device itself, enabling attacks such as Trojan insertion, reverse engineering, and counterfeiting [101]. Such activity has been estimated to cost the electronics industry over $100 billion each year [102].

To combat this problem, a device can make use of secret-key authentication or encryption to ensure identity and security when communicating. Both of these cases require that a device have a secret key that it can use to verify the identity of another device or encrypt data. This is commonly accomplished using nonvolatile memories such as read-only memories, battery-backed volatile memories like SRAMs, or flash memory [103, 104]. Unfortunately, these devices come with power and area costs that not all use cases can afford. As a result, some of them forego security to meet these requirements [105]. These problems are addressed by devices known as physical unclonable functions (PUFs) [106, 107], which derive uniqueness from silicon process variations like how humans derive uniqueness from fingerprints. Because these variations affect properties that are also affected by aging, PUFs are susceptible to change via aging. This chapter discusses the potential for using aging in a directed manner to deliberately affect the security of a PUF based on SRAM.

4.1 The Physical Unclonable Function

PUFs are analogous to fingerprints in human beings. Just like how a fingerprint is defined by natural variations in human skin that are unique to each individual, the fingerprint of a PUF is defined by *natural variations in silicon that occur during fabrication* and are unique to each device. On top of this, their nature as being derived from silicon variations makes them difficult to tamper with and impossible to duplicate during fabrication. They have low overheads in power, performance, and area, making them useful for secure systems with tight power budgets. For less power- or area-constrained systems, they still have advantages over more conventional devices in that they typically only provide their secret information upon request and only for a short time, increasing resiliency against attack [105].

Unfortunately, the same source of randomness that gives each PUF its uniqueness is also a source of vulnerability to aging. The degradation caused by aging changes physical device parameters that are affected by variations and can be leveraged for uniqueness. These changes can become severe enough to significantly affect the output of the PUF, effectively changing its fingerprint and denying service to whatever resource it is attached to. As a result, the lifetime of an electronic device, especially a secure one, is not only affected by functional failures caused by aging but also by its reliability in evaluating its fingerprint.

4.1.1 Types of PUFs

PUFs can be implemented using a variety of technologies, such as optics [107], memristors [108], spintronics [109], and CMOS [57, 110]. CMOS PUFs can generally be divided into two categories: timing-based PUFs, which are typically circuit paths whose output depends upon the delays of their components; and memory-based PUFs, which are typically bistable circuits such as RAMs or sense amplifiers that store a value after powering on or being enabled. Delay-based PUFs can be implemented using ring oscillators (RO-PUFs) [110] and multiplexers (arbiter PUFs) [111]. A common type of memory-based PUF is the 6T SRAM cell [57] shown in Figure 4.2(a).

PUFs are often characterized by the number of *challenges* they can accept and *responses* they have to those challenges [103, 105]. A challenge to a PUF is an input to the PUF, such as the assertion of an enable signal or a number indicating how to set internal structures. A response is the PUF's output as a result of the challenge. PUFs can typically be divided into two categories, "weak" or "strong," based on how many challenge-response pairs they have [103, 105]. Weak PUFs are characterized by the relatively small number of challenges they can accept and the similarly small number of responses they can generate. Because of this, the output of the PUF must be kept secret using cryptographic techniques such as hash functions. As a

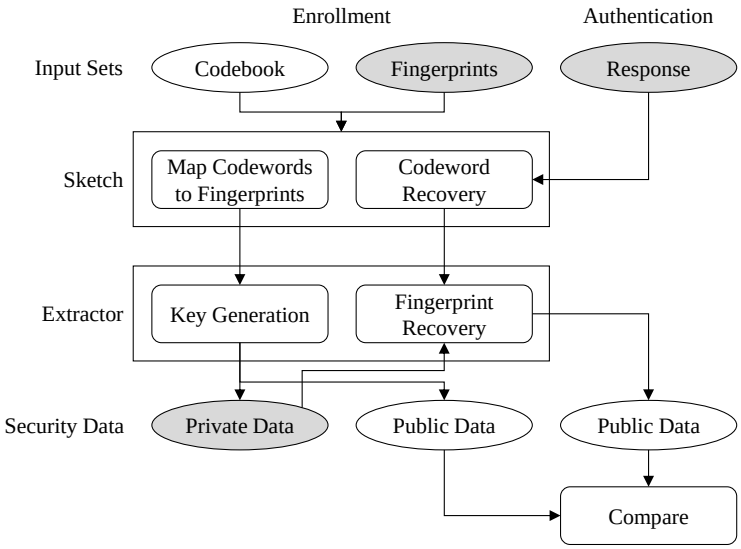


Figure 4.1: Diagram of a fuzzy extractor. Items in rectangles indicate functions and items in ovals indicate the data those functions operate on. Arrows indicate the direction of data flow. A gray background indicates data that must be kept secret. Note that the functions making up a fuzzy extractor can be made public as long as the data corresponding to identifiers is kept secret.

result, weak PUFs are useful for cryptographic key generation. Strong PUFs, on the other hand, have many challenge-response pairs. Even if an attacker were able acquire a sample of these pairs from a PUF, he or she would not be able to derive the rest and would not be able to gain any information about the PUF's internal behavior. While weak PUFs can be used for authentication when paired with other cryptographic functions, strong PUFs can be directly used without any processing on their outputs because their challenge-response pairs do not expose secret information.

4.1.2 Security with PUFs

A common problem with using a PUF as a security credential is variation in its responses caused by electrical noise. Temperature and voltage changes also affect responses in a more predictable manner. In order to get a reliable response from a PUF, error correction is necessary. Additionally, the fingerprint of a weak PUF must not be readable or inferrable by an attacker during the authentication process. Even though a strong PUF does not require additional processing to hide its fingerprint, it may still be susceptible to noise that must be corrected. This section outlines a general procedure for correcting errors and, when applicable, encrypting a fingerprint to hide it from external view, enabling the use of a PUF for authentication. The procedure is illustrated in Figure 4.1.

An identifier whose output is affected by noise, such as a PUF, is known as a "fuzzy identifier" [112, 113]. To use such a device for authentication, a "fuzzy extractor" must be created to correct noise in the identifier's

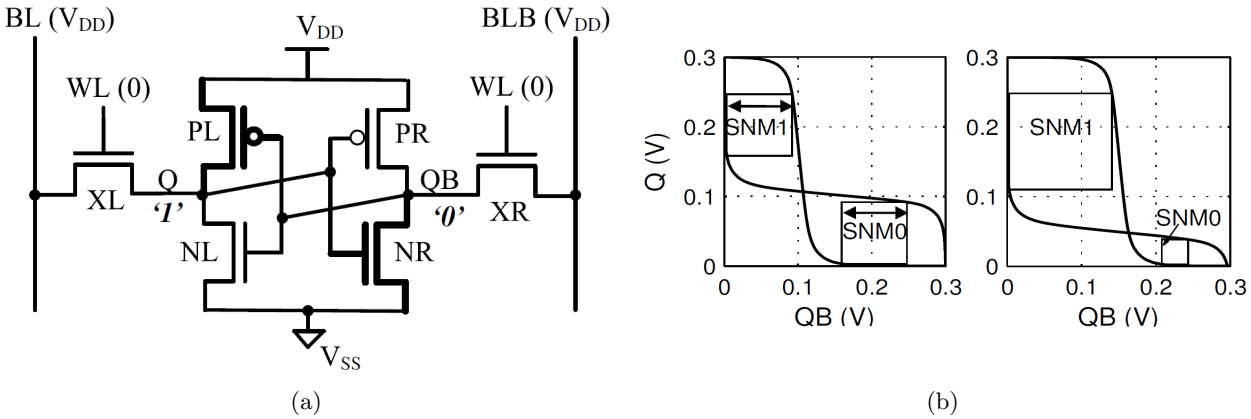


Figure 4.2: Circuit layout (a) and noise margin diagrams (b) for a 6T SRAM cell [114]. When V_{DD} is enabled, the two sides race to pull up to 1. The strengths of the opposing transistors affect the rate at which the voltage can rise. In the balanced case (b, left), Q and QB rise at the same time and the metastable state where the "wings" cross is reached. In an imbalanced case (b, right), the left side is stronger and the voltage will settle with Q = V_{DD} and QB = 0.

response and hide its fingerprint. In general, a fuzzy extractor is a set of functions that maps non-uniformly-distributed random numbers, like fuzzy identifiers' fingerprints, to uniformly-distributed ones, like hashes, and removes noise from the fuzzy identifiers' responses.

A set of fuzzy identifiers defines a space in which their fingerprints resides; for example, with a set of memory-based PUFs this space would contain all of the values the memories could store (e.g. 0 through 255 for an 8-bit SRAM). Also within this space lies a "codebook," $\mathcal{C}$, or a set of evenly-distributed elements called "codewords" that are no further apart from each other than the minimum distance between any two fingerprints. It is also important that the codebook and set of fingerprints are distinct. This way a mapping between fingerprints and codewords can be defined by selecting the codeword closest to each fingerprint. Ensuring that the codewords are closer to each other than the two closest fingerprints are ensures that each fingerprint maps to exactly one codeword and prevents ambiguity.

This mapping makes up the first of a pair of functions called a "fuzzy sketch." The other function performs error-correction on the noisy response of the fuzzy identifier and recovers the codeword it was originally mapped to. The fuzzy extractor itself is a pair of functions that make use of the fuzzy sketch to generate private and public data from an identifier's fingerprint and corresponding codeword, or "enrolls" the identifier, and to regenerate an identifier's fingerprint and public data from its response and private data. The regenerated public data can then be verified by an observer to authenticate the holder.

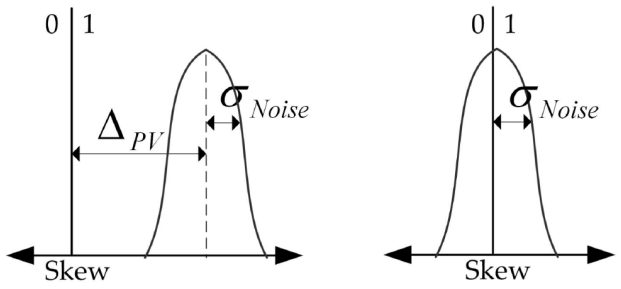


Figure 4.3: Illustration of the skew of a 6T SRAM cell [57]. The width of the curve, σ_{Noise}, illustrates the effect of noise while the offset from the vertical axis, Δ_{PV}, represents the effects of size mismatch caused by process variation. The left side corresponds to Figure 4.2(b, right), with higher chance of storing a 1 (1-skewed), while the right side shows an equal chance for both 1 and 0 (not skewed).

4.2 The SRAM PUF

The SRAM PUF is a weak PUF whose fingerprint is defined by the value each bit contains when it is powered on. As a result, it only has one challenge-response pair. The most common type of SRAM PUF uses 6T SRAM cells (Figure 4.2(a)) for each bit. When such a cell is powered on, the two sides, Q (left) and QB (right), race to pull up to V_{DD}. Eventually the cell must settle at a stable state with one side containing V_{DD} and the other containing 0 depending on the relative strengths of the transistors.

Before power is enabled, Q and QB are both 0. When power becomes enabled, the PMOS transistors, PL and PR, allow current to flow and begin raising the voltage of their corresponding sides. As this happens, the NMOS transistors, NL and NR, begin to conduct as well and start pulling their corresponding sides back down. If each pair of transistors were equally strong, the current pulling each side up would be the same, resulting in a metastable state as shown in Figure 4.2(b, left). Because of small variations between transistors caused by the manufacturing process, this is rarely the case. One side of the cell (the "stronger" side) will be stronger and have a higher current than the other and as a result is able to raise its voltage faster. When this happens, the other side's PMOS transistor begins to turn off while its NMOS transistor continues to pull the voltage down. As a result, the weaker side will drop to 0 while the stronger side will rise to V_{DD}. An additional factor in this process is noise, which can sometimes push the weaker side high enough to overcome the stronger side and rise to V_{DD}. This creates a random element in the state of the cell at power on that is heavily influenced by variation.

The power-on tendency of a 6T SRAM cell is indicated by a metric called "skew" [57] that indicates the imbalance in the two cross-coupled inverters. Skew is illustrated by Figure 4.3, which indicates power-on tendency with the area underneath the curve on the left or right side of the vertical axis; if more area is on the 1 side, the cell is more likely to contain a 1 when it powers on and is called "1-skewed"; the same is

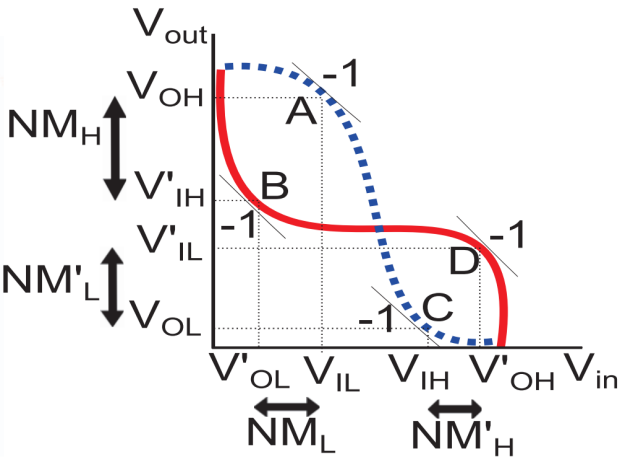


Figure 4.4: PUF static noise margin diagram [115]. Noise margins computed using the voltage transfer curves of the cross-coupled inverters in a 6T SRAM cell can be used to characterize startup tendency.

true for 0. The left graph indicates an SRAM cell where process variations, represented as Δ_{PV}, cause it to always power on containing a 1 and the right graph shows an ideal, balanced cell. The random effects of noise are represented using σ_{Noise}.

Skew can be measured using the probability, p, of the cell to contain a 1 after it powers on, also referred to as "one-probability" [116]. It can be modeled using special "PUF static noise margins" (PSNMs) [115]. This is different than read or write noise margins because it describes power-on tendency and not read or write behavior. They are computed using the transfer functions of the cross-coupled inverters arranged in a butterfly shape shown in Figure 4.4. The voltages at critical points A through D in the diagram can be computed by determining the mode of operation of each of the transistors in the SRAM cell and setting their drain currents to be equal. This process is described in more detail in [115]. From there, the noise margins NM and NM' can be computed [115]:

$$\begin{aligned} NM &= \min\left(V_{OH} - V'_{IH}, V_{IL} - V'_{OL}\right) \\ NM' &= \min\left(V'_{OH} - V_{IH}, V'_{IL} - V_{OL}\right) \end{aligned} \tag{4.1}$$

This enables the computation of a metric that indicates startup tendency and can be used to compute p:

$$PSNM_{ratio} = \frac{NM}{NM'} \tag{4.2}$$

A cell where $PSNM_{ratio} > 1$ will have p closer to 1 and be "1-skewed," while a cell where $0 \leq PSNM_{ratio} < 1$ will have p closer to 0 and be "0-skewed." The authors of [115] show that cells where $PSNM_{ratio}$ is less than 0.995 or greater than 1.005 are typically strongly skewed.

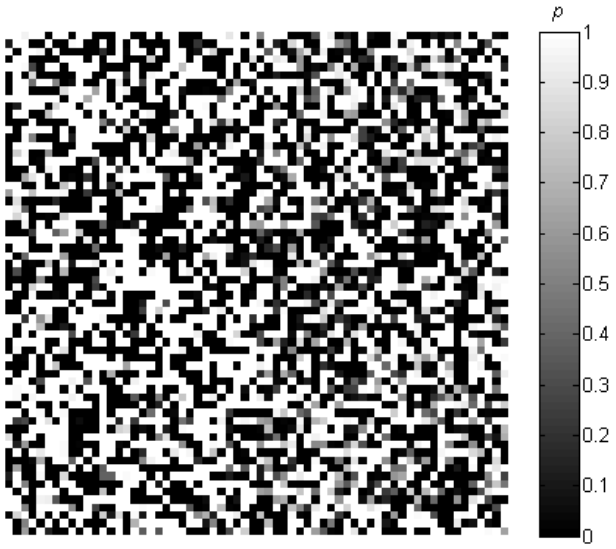


Figure 4.5: Example fingerprint for an SRAM array [117]. Across a chip, the skews of the SRAM cells produce a unique pattern where most cells are skewed toward 1 (black) or 0 (white), but some some are close to 0.5, or are balanced (gray).

With the noise margins computed from (4.2), the value of p can be found using a standard normal distribution [116]:

$$p = \Phi\left(\frac{PSNM_{ratio} - 1}{\sigma_{noise}}\right) \tag{4.3}$$

where $\Phi(x)$ is the standard normal cumulative distribution function and σ_{noise} represents the variance of V_{DD} due to noise. Process variations that cause variance in physical dimensions, threshold voltage, and other transistor characteristics change the drain current and critical voltages, and therefore the PSNMs, of the SRAM cell. If PL is weaker than PR due to shorter length or higher threshold voltage, the corresponding noise margin will be lower and p will be closer to 0, causing the cell to be 0-skewed. In the same way, a weaker PR causes p to be closer to 1.

The skews of the cells across an entire SRAM array can be used to create its fingerprint, shown in Figure 4.5. Variations in SRAM cells across a chip are not necessarily correlated to the variations of corresponding cells in another chip from the same die. As a result, these skews create a unique pattern for each chip and are useful as PUF fingerprints [57]. As shown in Figure 4.5, most cells in an array are heavily 0- or 1-skewed, implying that skew is very sensitive to process variations.

4.3 Aging in SRAM PUFs

Just like any CMOS device, the parameters of SRAM are affected by NBTI. As discussed in the previous section, process variations affect the skew of each cell to create a tendency to power on using a particular

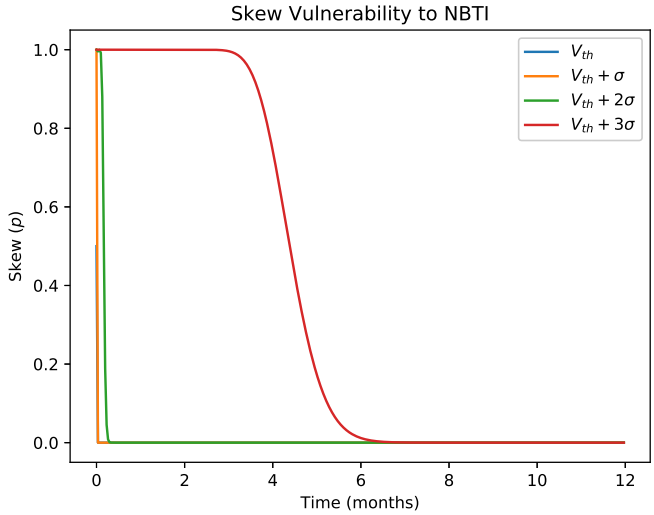


Figure 4.6: Change in skew over time for a 65nm process due to NBTI for different mismatches in size [119]. Mismatches in transistor sizes due to variations are modeled as differences in V_T of one, two, and three standard deviations in the measured distribution of V_T for 65nm designs [120].

pattern that can be used as a fingerprint for a PUF. NBTI causes a long-term degradation of the parameters of a transistor, reducing the current that passes through it; this weakens the affected side of the cell, changing its skew.

NBTI is a process where applying a negative bias to the gate of a PMOS transistor modulates the energy of charge traps inside it. If a trap gains enough energy, it can capture a charge carrier in the channel [24]. This process is illustrated in Figure 1.1(b). Each trapped charge causes a shift in the transistor's threshold voltage. At any given time, an empty trap has some probability of capturing a charge and a filled trap has some probability of freeing its captured charge based on temperature, gate bias, and number of full or empty traps. While a negative bias is applied and the device is degrading, the threshold voltage degradation can be described using the sum of an exponential function of the form $1 - e^{-t/\tau}$, where τ is a time constant for degradation, and a power-law function of the form $t^{1/6}$. When the bias is reduced or removed, some recovery from degradation is experienced which takes the form of exponential decay, $e^{-t/\tau}$ [66]. The stress process exhibits both recoverable and permanent components, which limit the effectiveness of recovery. Others have shown that reversing the bias of the gate can further improve recovery from degradation ("active recovery") and reduce this permanent component [118]. For more information on the mechanism behind NBTI and how it affects a transistor, see Chapter 1.1.1.

In an SRAM, an increase in a PMOS transistor's threshold voltage caused by NBTI reduces its drain current, reducing NM or NM' and causing a corresponding shift in p. The authors of [114] show that this effect is data-dependent; while an SRAM cell stores a value, the PMOS transistor of the corresponding side

is degrading due to NBTI. As a result, if the cell contains a 1, PL's threshold voltage increases, causing a decrease in NM, $PSNM_{ratio}$, and p. This process is shown in Figure 4.6, which presents the change in p caused by NBTI over one year due to storing a 1 for a balanced cell and for cells with mismatches modeled as differences in V_T. Figure 4.6 shows that NBTI can be effective in switching 1-skewed cells to 0-skewed within one year in a 65-nm process operating continuously at 1.1 V and 300 K for differences in V_T of up to three standard deviations of the distribution of V_T for that process as measured by [120]. Similarly, because 6T SRAM cells are symmetrical, aging while containing a 0 causes NM' to decrease, increasing $PSNM_{ratio}$ and p. In [103], it is suggested that this might be usable to attack an SRAM PUF, whose fingerprint depends on these values for its cells, by coercing individual bits' skews toward certain values.

4.4 Manipulating SRAM PUF Reliability

The following sections detail four experiments that are performed to explore the idea of coercing SRAM cells' skews using directed aging:

1. Test the effectiveness of modifying an SRAM's fingerprint using aging.
2. Since the effects of aging on SRAM cells are data-dependent [114], measure the effects of various storage patterns on reliability in reconstructing an SRAM PUF's fingerprint using its aged power-on state.
3. Test the effectiveness of active NBTI recovery in restoring the reliability of a degraded SRAM PUF.
4. Measure the effectiveness of using the fingerprint of one SRAM PUF to generate an aging pattern for another SRAM PUF and effectively clone the first PUF's fingerprint.

Each of these experiments uses increased voltage and temperature to activate and accelerate, respectively, the aging process.

4.4.1 Affecting a Fingerprint with NBTI

This experiment explores the ability of directed NBTI to cause significant, controllable changes to an SRAM PUF's fingerprint. It also shows that the permanent component of NBTI is significant enough that changes to a PUF's fingerprint will remain even after natural recovery while powered off for a long period of time. Two experiments are performed: first, NBTI is accelerated with only high temperature while the SRAM stores all ones; second, NBTI is accelerated with high temperature and activated with high voltage while the SRAM is filled with zeroes.

Both experiments make use of an 8-kB AS6C6264 commercial SRAM IC whose relevant parameters are shown in Table 4.1. Both experiments have the same procedure:

1. Read the initial state of the SRAM.

Table 4.1: AS6C6264 Specification [121]

Parameter	Value
Capacity	9182 × 8 bits
Nominal V_{DD}	3 V
Maximum V_{DD}	5 V
Operating Temperature	0-70 °C

Table 4.2: Experiments 1 and 2 Wearout Conditions

Experiment	Parameter	Value
1 (Accelerated Aging)	Temperature	120 °C
	V_{DD}	5 V
	Time	3 days
	Pattern	All ones
2 (Accelerated, Active Aging)	Temperature	120 °C
	V_{DD}	7 V
	Time	3 days
	Pattern	All zeroes

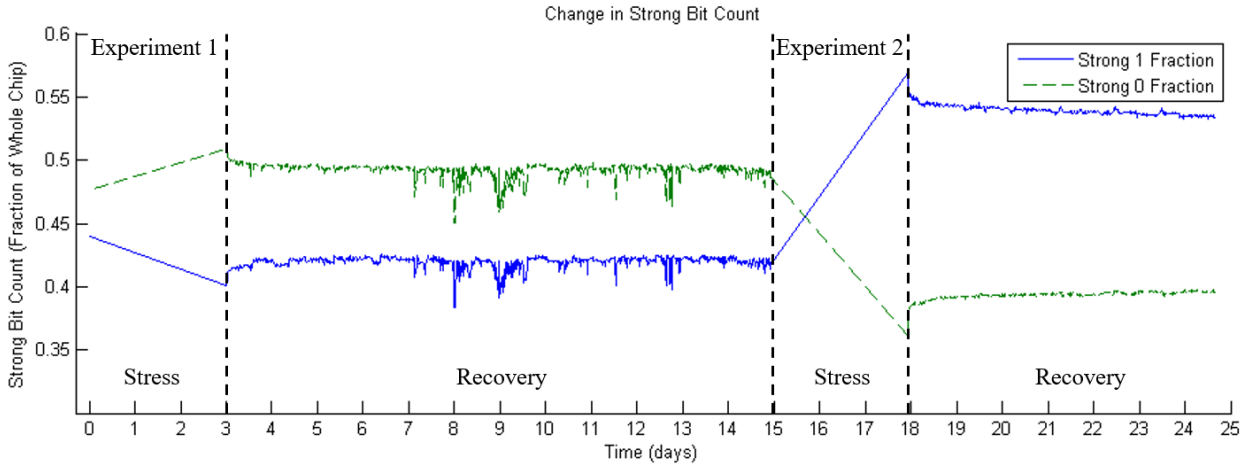


Figure 4.7: Fraction of strong ones (blue, solid) and strong zeroes (green, dashed) over the course of experiments 1 and 2 [117]. Dashed vertical lines indicate times at which stress switched to recovery or vice versa. The increased changes in strong bit count for experiment 2 support the effectiveness of raising V_{DD} to modify a fingerprint. In both experiments, most recovery occurs during the first few hours and then slows down nearly to a stop, suggesting high retention of NBTI effects.

2. Stress the SRAM for 72 hours using its pattern and conditions from Table 4.2.
3. Read the final state of the SRAM and allow it to naturally recover at room temperature while powered off. Read its state every 15 minutes until all recoverable stress has been relieved.

All measurements are performed at nominal V_{DD} (3 V) and after allowing the chip to cool to room temperature for 30 minutes. Each one consists of one hundred power-on state readings. Table 4.2 enumerates the stress conditions and pattern of each of the two experiments performed.

Figure 4.7 shows the changes to the number of bits that are heavily 0- or 1-skewed, which will be referred to as "strong bits," over time as the SRAM ages. A strong bit is one whose skew is either at least 0.9 ("strong one") or at most 0.1 ("strong zero"). During the first experiment, where the SRAM stored all ones, the number of strong ones decreased by 8.8% and the number of strong zeroes increased by 6.87%. The effects were much greater during the second experiment, where the SRAM stored all zeroes but the voltage was

elevated. During that experiment, the number of strong ones increased by 36.42% from the value at the start of the second experiment and strong zeroes decreased by 25.96%. This corresponds to an increase in strong ones of 29.31% from the value of the fresh chip and a 24.17% decrease in strong zeroes. Due to the symmetry of the 6T SRAM cell and because the numbers of strong zeroes and ones are close for the fresh chip, it is reasonable to assume that these numbers would be similar, but for opposite strong bits, if the bit patterns had been reversed; that is, if the first experiment had written all zeroes and the second experiment had written all ones. Since the change in strong bits after the second experiment is so much larger than that of the first experiment, even accounting for the fact that the second experiment had to overcome the changes caused by the first experiment, we can conclude that elevating the voltage had a significant effect on increasing NBTI aging on the SRAM. Raising V_{DD} from 5 V to 7 V, only a 40% increase, nearly tripled the changes in strong bits that occurred.

Describing a cell only in terms of p may not capture the true nature of the imbalance between its transistors, especially for strong bits. For example, even though two bits may have skews of 1, one of them may be more imbalanced than the other and thus will be less susceptible to NBTI aging. In [122], the authors propose a method for identifying those bits. They show that bits whose skews are exactly 0 or 1 that are physically surrounded by other bits with the same skew tend to be more reliable. When exposed to extreme temperature and voltage conditions, these bits are consistent in their power-on values even when others that are strong in normal conditions are not. In order to find these bits, the authors offer a heuristic method when the physical layout of the SRAM is unknown: arrange the bits of the SRAM so that each row appears in address order and assume that bits that are adjacent in this string are also adjacent physically. If a bit with a strong skew in this string is surrounded by enough bits with the same strong skew, then that bit can be used as part of a fingerprint. The authors label these bits as "stable." A bit in this experiment's SRAM can be considered stable if it has a strong skew (0 or 1) and is surrounded by at least twenty neighbors on each side, using the aforementioned heuristic, that have the same strong skew. By this definition, the SRAM chip that was used contains 280 stable bits. At the end of the second experiment, 74 of these bits were no longer stable.

During the recovery phase after each experiment, Figure 4.7 shows that most of the recovery a device experiences occurs quickly and then slows down significantly. During the first hour of the first recovery cycle, the number of strong ones drops by 2.56% and strong zeroes rises by 1.17%, but during the second hour they change by only 0.31% and 0.29%, respectively, and after the first day there are nearly no changes at all. After the second experiment, the results are similar. During the first hour of recovery, the number of strong ones increases by 3.16% and the number of strong zeroes decreases by 6.07%, but during the second hour they change only by 0.19% and 0.63%, respectively. This suggests a high retention of the skew changes caused by NBTI and that an SRAM will not recover naturally from an attack.

Algorithm 4.1 SRAM PUF fingerprint recovery reliability

function RELIABILITY($f, \mathcal{S}, \mathbf{e}, r, m$)
  $\mathcal{C} \leftarrow \mathrm{RM}(i, m)$
  for $j \leftarrow 1$ **to** 50 **do**
    $c \leftarrow$ random element of $\mathcal{C}$
    $w \leftarrow c \oplus f$
    for $s \in \mathcal{S}$ **do**
      $C^* \leftarrow$ RECOVER$(w, s, \mathbf{e}, r, m)$
      if $c \oplus C^* = 0$ **then**
        $successes \leftarrow successes + 1$
  $reliability \leftarrow successes / (50 \cdot |\mathcal{M}|) \cdot 100\%$
function RECOVER($w, s, \mathbf{e}, r, m$)
  $C' \leftarrow w \oplus s$
  for $i \leftarrow 1$ **to** $|s|$ **do**
    $L_i \leftarrow (\log(1 - \mathbf{e}_i) - \log \mathbf{e}_i)^{C'_i}$
  $L^* \leftarrow$ GMC-DECODE-RM$_{r,m}(L)$ ▷ Algorithm 2 from [123]
  for $i \leftarrow 1$ **to** $|L^*|$ **do**
    $C^*_i \leftarrow 0$ **if** $L^*_i < 0$ **else** 1
  return C^*

Algorithm 4.2 Reed-Muller code order

$i \leftarrow 1$
repeat
  $reliability \leftarrow$ RELIABILITY$(f, \mathcal{S}, \mathbf{e}, i, m)$ ▷ Algorithm 4.1
  $i \leftarrow i + 1$
until $reliability < 100\%$
$r = i - 1$

4.4.2 Attacking an SRAM PUF with NBTI

Having shown that even the strong bits of an SRAM's fingerprint can be affected by NBTI, NBTI is applied as an attack by increasing the failure rate of the SRAM to reconstruct its fingerprint from a reading of its power-on state. SRAM fingerprints are enrolled and reconstructed using the fuzzy extractor defined in [123].

Four AS6C6264 chips are stressed storing different data patterns. Their fingerprints are defined using one hundred reads of their power-on states at room temperature before any NBTI is applied. Chip S1 is stressed while storing its fingerprint to simulate the effect of a standalone PUF; chip S2 stores arbitrary bits to simulate the effect of using the PUF as data storage, as there have been proposals to use SRAM PUFs as part of the normal system memory after evaluation [104]; and chips S3 and S4 store all ones and all zeroes, respectively, to simulate the effect of a PUF that is cleared after evaluation to hide its fingerprint. Clearing chip S3 with all ones is expected to have a different effect on its reliability than clearing chip S4 with all zeroes, as both chips have more 0-skewed cells than 1-skewed cells. As a result, the reliability of the chip storing all zeroes, chip S4, to recover its fingerprint should reduce faster.

In order to make use of the fuzzy extractor defined in [123], a codebook and helper data are created using a Reed-Muller block code to map fingerprints to codewords and aid with fingerprint reconstruction. A Reed-Muller code RM (r, m) with parameters r and m has a block length of 2^m bits, has a minimum distance between codes of 2^{m-r} bits, and can encode a message of length

$$k = \sum_{i=0}^{r} \binom{m}{i} \tag{4.4}$$

bits. The effective strength of the SRAMs' secret keys can be determined by multiplying this message length with the min-entropy, H_∞, [57] of the SRAM chips:

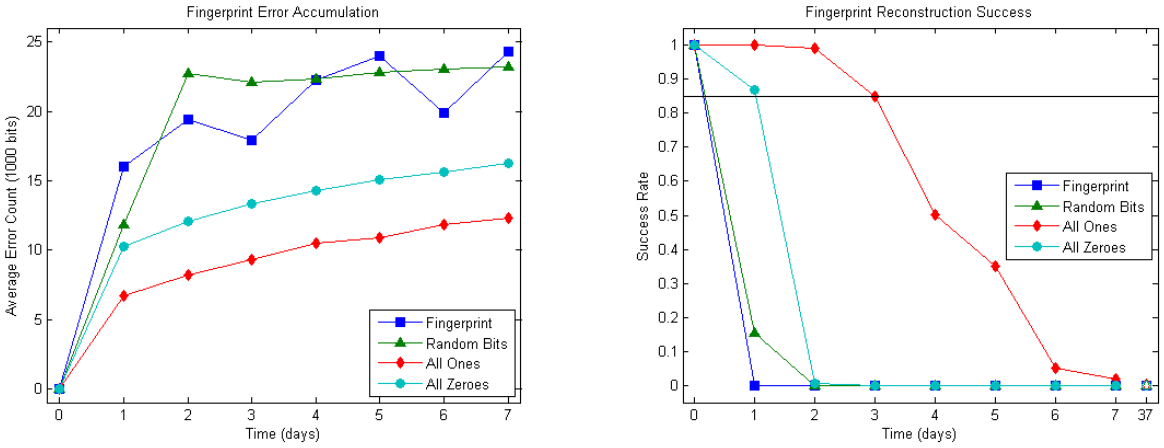
$$H_\infty = -\log_2 \prod_{i=1}^{n} \min\left(p_i, 1 - p_i\right) \tag{4.5}$$

where n is the number of bits in the SRAM and p_i is the skew of bit i. The order, r, of the Reed-Muller code is determined using Algorithms 4.1 and 4.2 with f representing the chip's fingerprint and $\mathcal{S}$ representing the set of readings that are used to define it. The helper data for the algorithm consists of w, the bitwise XOR of a code from the codebook and f, and a vector, $\mathbf{e}$, describing the probability of each bit to be in error, or $\min\left(p_i, 1 - p_i\right)$. The use of a hash family as described by [123] is not included with this procedure, as its only purpose is to hide the chip's fingerprint and does not improve reliability in recovering it. This procedure is repeated for each chip, and the lowest value of r found across the four chips is used to generate the final codebook for the experiment.

With the helper data generated, each chip is stressed using its corresponding pattern. NBTI aging is accelerated using temperature elevated to 120 °C and activated using voltage raised to 7 V. Every day for one week, the chip is returned to nominal V_{DD} and room temperature to perform a measurement of one hundred readings of the SRAM's power-on state. Each reading is used to attempt to recover the original fingerprint of the SRAM with each of the 50 codes generated during the previous step. If the success rate for recovering the fingerprint drops below 85% [123], the SRAM is unreliable. After one month, a final measurement is performed to measure the permanence of the damage done to each chip's fingerprint.

Algorithm 4.2 produces $r = 4$ for the set of chips used for this experiment. Since the entire SRAM is being used as a PUF and its fingerprint is to be combined with a codeword, codewords need to be 65536 bits long, or $m = 16$. This corresponds to a message length of 2517 bits, computed with (4.4). With the chips' min-entropy of 5.08%, this is equivalent to a 127-bit secret key.

Figure 4.8(a) shows the average number of bit errors in each chip's fingerprint as it ages with its stored pattern. Chip S1, which stored its fresh fingerprint while aging, has the fastest initial increase in errors and



(a) (b)

Figure 4.8: Accumulation of bit errors (a) and rate of successful reconstruction (b) for each chip's fingerprint as it ages [117]. The horizontal line at 85% in (b) indicates the minimum success rate necessary for an SRAM to be reliable as a PUF [123]. The empty markers in (b) indicate the success rate after one month of natural recovery and represent the degradation caused by permanent NBTI effects.

the highest final error count with some crossing over with chip S2, which stored random bits. After chip S1, chips S2, S4, and S3 have lower error counts, in that order. Chip S1's fast accumulation of errors is due to the inverse nature of the effect NBTI has on the skew of an SRAM cell. Because the side of the cell corresponding to the value stored gets weaker as it ages, the skew of the cell moves away from that value and the probability it will contain that value at power-on decreases. A cell that contains the value it is skewed toward while aging can eventually be flipped and become skewed toward the opposite value this way, while a cell containing the value opposite its skew will only become stronger. If an SRAM stores its fingerprint and ages, the maximum number of cells that are susceptible to flipping will flip, explaining the fast rate of error accumulation. Chip S2's close final error count to chip S1 suggests that the random bits used for aging were closer to its fingerprint than all ones or all zeroes were to their chips' respective fingerprints. These results correspond with those published in [124]. Chip S1 also experienced a greater amount of variation in its error count than the other chips, suggesting that, after the first day of stress, NBTI had pushed many of the chip's bits close to being balanced. As time passed, those bits would become more skewed away from their initial skew while other bits would become balanced. A greater number of balanced bits increases the effects of noise, thus increasing variation in the resulting fingerprint measurement. As expected, clearing the chips with all ones caused a different effect than clearing with all zeroes. For similar reasons why storing the fingerprint had the fastest effect, all zeroes was closer to chip S4's fingerprint than all ones was to chip S3's. Because of this, there were more bits whose skews were susceptible to being flipped in chip S4.

Figure 4.8(b) presents the success rate of each chip to recover its fresh fingerprint after each day of stress as computed by Algorithm 4.1. This rate is calculated using the number of times each chip's original fingerprint could be reconstructed using its aged data and each of the 50 codes generated during the enrollment process presented in the previous section. The horizontal line in Figure 4.8(b) at 85% indicates the success rate above which an SRAM is considered reliable enough to be useful as a PUF [123]. All four chips passed below this line after three days of stress while chips S1 and S2 took only one day. As predicted by [124] and corresponding with the rate of error count accumulation in Figure 4.8(a), storing chip S1's fingerprint during aging caused it to fail faster than the others. Closely following it is chip S2, which dropped to about 15% success rate after the first day and then to 0% after the second, suggesting that using an SRAM PUF for data storage after evaluation is nearly as damaging as leaving it on after evaluation. This is corroborated by [124], which shows that using an SRAM PUF for storage is nearly as damaging as storing its calculated fingerprint. Chips S3 and S4, which were cleared with 1 and 0, respectively, during aging, both took over a day to become unreliable. Chip S3 trailed far behind chip S4, taking three days to become unreliable. This corresponds with the rates at which they accumulated errors shown in Figure 4.8(a). However, chip S3 does not trail chip S4 in Figure 4.8(a) relative to the other chips as much as it does in Figure 4.8(b), which suggests that there is a region of error accumulation in which success rate is sensitive and outside of that region it is insensitive.

Figure 4.8(b) also shows using hollow symbols at 37 days the success rate at reconstructing the fresh fingerprint of each chip after it has been allowed to naturally recover at room temperature and while powered off for one month. With all four chips, the success rate shows no significant improvement and remains close to 0%. This implies that once a PUF has been affected by NBTI and becomes unreliable that it will not naturally recover enough to become usable again.

4.4.3 Restoring Reliability via Active Recovery

The previous section showed that natural recovery from NBTI while powered off will not restore an SRAM PUF's fingerprint enough for it to be reliable as an identifier. In [118], the authors show that NBTI recovery can be enhanced by applying a negative voltage to an aged device, which they call "active recovery." Just like with stress, recovery can be accelerated by increasing temperature. This principle is applied in this section to repair several SRAMs' original fingerprints by applying a slight negative V_{DD} to activate recovery and by increasing temperature to accelerate it, thereby creating a potential defense against the attack outlined in the previous section.

Table 4.3: IDT71256 Specifications [125]

Quantity	Value
Size	32k × 8 bits
Nominal V_{DD}	5 V
Maximum V_{DD}	5.5 V
Operating Temperature	0-70 °C

Table 4.4: Recovery Experiment Stress Patterns

Chip	Stress Pattern	Recovery Threshold
R1	Fingerprint	About 0%
R2	Random bits	About 0%
R3	All ones	85%
R4	All ones	About 0%
R5	All zeroes	85%
R6	All zeroes	About 0%

This experiment is performed with several commercial 256-kB IDT71256 SRAM chips[1] whose parameters are shown in Table 4.3. The experiment begins the same way as in in Chapter 4.4.2, where several SRAMs' fingerprints are characterized using one hundred reads at room temperature and nominal V_{DD} and each one is assigned a stress pattern that is shown in Table 4.4. As in Chapter 4.4.2, each chip is stressed while storing its assigned pattern at elevated temperature and voltage for 24 hours and then its fingerprint is read again at room temperature and nominal V_{DD}. Unlike in Chapter 4.4.2, where stress ends after one week, the stress phase ends when the SRAM has reached a desired success rate as enumerated in Table 4.4, after which the active recovery phase begins. This process is the same for each chip:

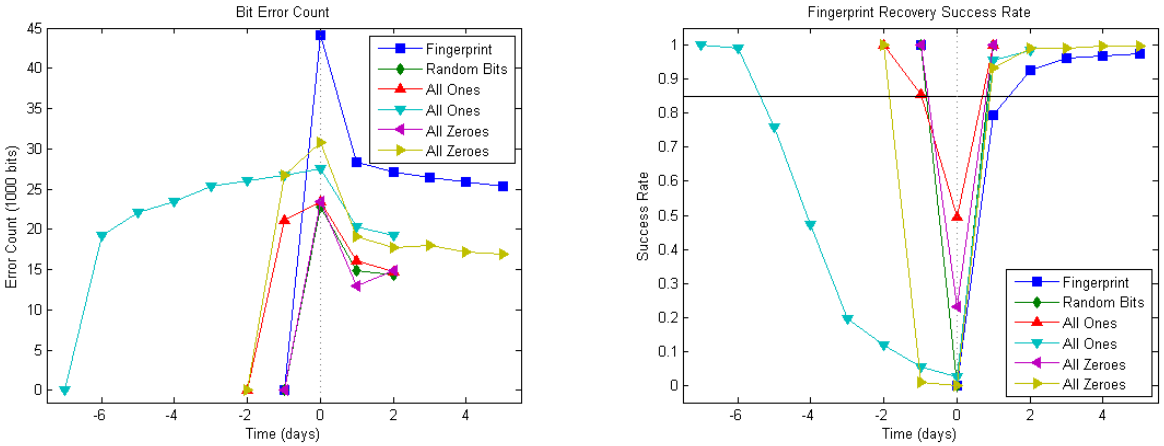
1. At room temperature and nominal voltage, read the fingerprint.
2. Apply slight negative voltage and high temperature and allow the chip to recover for 24 hours.
3. Measure the SRAM's success rate in reconstructing its original fingerprint using Algorithm 4.1. If it isn't 100%, repeat from step 1.

Stress and recovery occur at 120 °C to accelerate both processes. Each chip's V_{DD} is elevated to 7 V to activate stress and reduced to -0.5 V to activate recovery. The same method outlined in Chapter 4.4.2 is used to create helper data for each SRAM's fingerprint and compute each SRAM's success rate in fingerprint reconstruction.

For the patterns with slower effects that take several days to render an SRAM unreliable, this experiment is performed twice. First, active recovery begins once the SRAM has become unreliable, at an 85% reconstruction success rate. Second, active recovery begins once the SRAM's success rate has dropped to close to 0%. By doing this it is possible to measure the resilience of the damage caused by the stress pattern. Not every pattern supports this procedure because faster patterns cause success rate to reduce to 0% after only a single cycle. Each chip's target success rate before beginning recovery is noted in Table 4.4.

Algorithm 4.2 produces a Reed-Muller code order of $r = 5$ for the IDT71256 chips used in this experiment. Combined with the chips' size of 256 kB, this corresponds to a message length of 12,616 bits. Due to the chips'

[1]Note that even though the IDT71256 chips used in Chapter 4.4.3 are different than the AS6C6264 ones used in Chapter 4.4.2, the fundamental behavior of NBTI depends only on the 6T cell layout, which is the same for both chips. Any differences in technology size only affects vulnerability to NBTI and how fast PUF reliability is degraded.



(a) (b)

Figure 4.9: Average number of errors from the fresh fingerprint (a) and success rate at reconstructing it (b) for each chip as it ages and recovers [119]. Time 0, marked with a vertical dotted line, indicates the day at which stress switched to recovery. The horizontal line at 85% in (b) indicates the minimum rate which an SRAM is usable as a PUF.

min-entropy of 3.25%, this yields a 410-bit secret key. Figures 4.9(a) and (b), respectively, show the average number of errors that occur when reading each chip's fingerprint and the rate of success at reconstructing the original fingerprint from the stressed one. Time 0 in both figures represents the day at which the active recovery phase began, and the horizontal line at 85% marks the minimum success rate at which an SRAM is considered reliable for use as a PUF.

The stress phase behaved similarly to the previous experiment, with the fingerprint and random bits reducing success rate and increasing errors the fastest, followed closely by all zeroes and then finally all ones. One might expect that the active recovery phase would behave oppositely; the fingerprint would recover the slowest, followed by arbitrary bits, all zeroes, and finally all ones. Interestingly, in terms of success rate, the SRAM storing random bits while aging actually recovered the fastest rather than second slowest. All zeroes and all ones were close to being tied for second slowest recovery. The success rates of each of those three returned above 85% in only one day. As expected, the SRAM storing its fingerprint recovered slowest by far, taking several days to return to being reliable.

In terms of error count, the behavior is similar. After aging, the SRAM storing its fingerprint during aging shows the highest number of errors, followed by all zeroes, then arbitrary bits, and finally all ones. The fact that the ordering of all zeroes and arbitrary bits is reversed with error count compared to success rate suggests that there is a greater tendency toward storing zero in the tested SRAMs' fingerprints than in those from the previous experiment. This is supported by the findings of [126], who found a similar tendency in another chip produced by the same manufacturer. The result is that the effect of storing random bits or

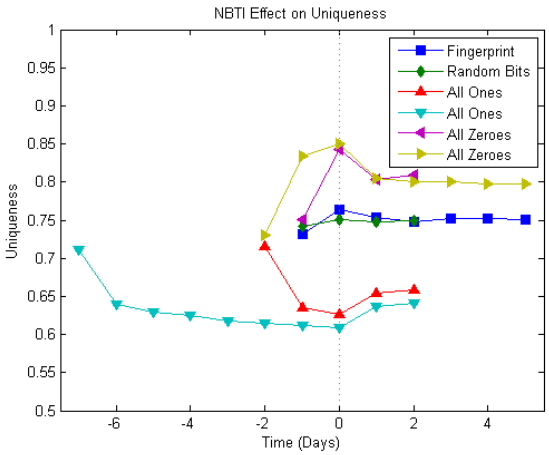


Figure 4.10: Effect of NBTI degradation and active recovery on each chip's uniqueness [119]. Time 0, marked with a dotted line, indicates the time at which stress switched to recovery.

all zeroes is similar enough that variance in the result caused by noise becomes a significant factor. There is a similar effect for all zeroes and all ones after recovery: even though all zeroes recovered more errors than all ones did after the first day of recovery, all ones had a higher success rate, suggesting that variance is significant there as well. The SRAM that stored its fingerprint recovered the greatest number of errors while also taking the longest to recover its success rate, indicating a narrow "window" where success rate varies with error count while remaining constant (at 100% or 0%) outside of it. This also explains why the chip storing arbitrary bits became unreliable nearly as fast as the one storing its fingerprint, but experienced fewer errors and recovered faster. The next section will discuss this further and use it to predict the amount of time it will take to apply directed NBTI to effectively clone an SRAM's fingerprint.

An important metric for evaluating the usefulness of a PUF is its uniqueness, d_{inter}, which is a measure of the difference in responses between different PUFs of the same type [127]. In the case of SRAM, this is the average distance between fingerprints of all pairs of PUFs in a group:

$$d_{inter} = \frac{2}{k\,(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD\,(R_i, R_j)}{n} \tag{4.6}$$

where k is the number of PUFs in the group, n is the number of bits in the PUF's response, $HD\,(R_i, R_j)$ is the Hamming distance between fingerprints of PUF R_i and R_j, and $HD\,(R_i, R_i) = 0$ for any i. The uniqueness of a single PUF, $d_{inter,i}$, is then the average distance between that PUF's fingerprint and every other one's fingerprint:

$$d_{inter,i} = \frac{2}{k-1} \sum_{j=1}^{k} \frac{HD\,(R_i, R_j)}{n} \tag{4.7}$$

Figure 4.10 shows the uniqueness of each chip among the group as it ages and recovers. Since there is no correlation between the fingerprints of the chips in the group, the effect of aging a PUF while storing its fingerprint would not be expected to have much effect on its uniqueness. Random bits would be expected to behave similarly. Figure 4.10 confirms this, showing only small changes in uniqueness of chips R1 and R2. Because all of the chips' fresh fingerprints had significantly more zeroes than ones, however, aging while storing all zeroes and all ones does have an effect on uniqueness. As Figure 4.10 shows, storing all ones in both cases reduced uniqueness while storing all zeros increased it. When an SRAM PUF is aged while storing all ones, the number of zero-skewed cells increases. Since all the chips used in this experiment have more zeroes than ones in their fresh fingerprints, changing the skew of a cell in one of the chips from nonzero to zero has a high chance of causing that cell to have the same skew as the corresponding cells of the rest of the chips. As more cells across the chip become more zero-skewed, its fingerprint will become more similar to those of the rest of the chips with a limit at the average number of ones that the rest of the chips have. Oppositely, if the chip is aged while storing all zeroes, zero-skewed cells will start to become more one-skewed, which have a high chance of having a different skew than the corresponding cells of the other chips. Therefore, aging while storing all zeroes will increase the uniqueness of a chip in this group.

During recovery, the NBTI degradation that accrued during the aging phase decreases, strengthening the transistors that had been weakened and moving the SRAM cells' skews back toward their initial values and restoring each chip's uniqueness. Just as with error count, uniqueness does not recover fully due to the permanent component of NBTI stress. Even though active recovery with negative voltage reduces this component, it does not eliminate it entirely.

Other work improving the reliability of an SRAM PUF using aging has focused on taking advantage of the data-dependent nature of the skew shift caused by NBTI to strengthen an SRAM's fingerprint [128, 129]. These works attempt to reinforce the skew of each bit by aging it with a 1 if it is 0-skewed or 0 if it is 1-skewed. Additionally, the authors of [129] use this process to improve the balance of 0- and 1-skewed bits on the chip in order to improve uniqueness. This reduces the number of errors that occur when evaluating the PUF and improves its security, but is not compatible with the use of active recovery to relieve existing degradation. Applying a negative voltage to a cell that has been improved using directed aging will remove the reinforcement to its skew.

A more direct comparison is with [124], which measures the effectiveness of applying directed NBTI to resisting changes in fingerprint due to aging over a device's lifetime. Figure 4.9(a) shows that the number of bit errors after active recovery ranges from about 0.7% of bits on the chip to about 1.2%, depending on how many errors accumulated during stress. This is comparable with the effectiveness of writing continuously-changing random values to the SRAM, which, as shown in [124], results in about 1% of bits in error. As [46] shows,

however, more proactive application of active recovery will reduce the permanent component of stress, further reducing the number of bit errors accumulated.

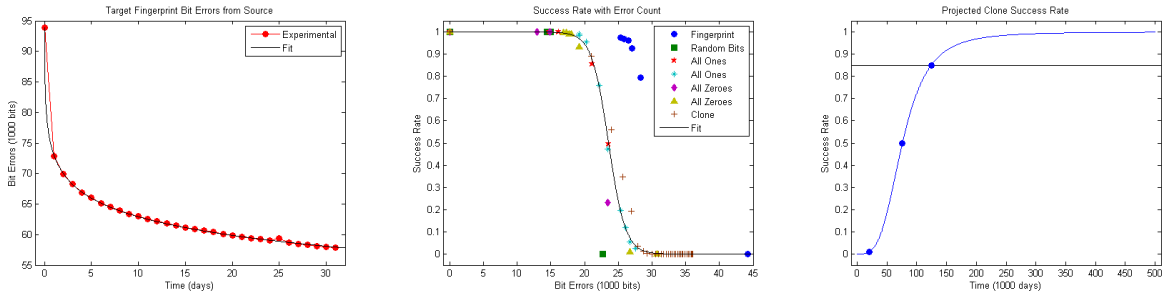
4.4.4 Cloning a Fingerprint with NBTI

Chapter 4.4.1 and [114] show that it is possible to exert control over the direction of a skew shift that is caused by NBTI. Chapters 4.4.1 and 4.4.2 showed that NBTI can significantly affect the fingerprint of an SRAM enough to effectively erase it. In this section, these principals are combined to use directed NBTI to clone the fingerprint of an SRAM PUF. After reading the fingerprint of the "source" SRAM using a process such as in [130] and taking advantage of the fact that storing a particular bit value shifts skew toward the other one, it should be possible to manipulate the fingerprint of a "target" SRAM such that it is close enough to that of the source that it can reliably impersonate it, effectively cloning the source's fingerprint.

This experiment makes use of two commercial 256-kB IDT71256 SRAM chips whose parameters are shown in Table 4.3: one which serves as the source and the other which serves as the target. Each SRAM's fingerprint is characterized at room temperature and nominal V_{DD} using one hundred reads of its power-on state as in the previous experiments. Due to the inverse nature of the effect of NBTI on SRAM, the source's fingerprint is inverted and then written to the target. The target is then exposed to high voltage and temperature to activate and accelerate the aging process while remaining powered on. Every 24 hours, it is cooled to room temperature and nominal V_{DD} to read its aged fingerprint and then returned to aging conditions. Just like with the previous experiment, NBTI is accelerated by raising the temperature to 120 °C and activated by raising V_{DD} to 7 V. This process is repeated each day for one month.

As in Chapter 4.4.3, fifth-order Reed-Muller codes are used as helper data, corresponding to a 410-bit secret key. The process for determining the success rate of the target SRAM impersonating the source is the same as the process used in Chapter 4.4.2 for determining the success rate of an SRAM in identifying itself, except that the source's fingerprint and helper data are substituted for the target's. A rising success rate indicates that the target's fingerprint is increasingly resembling the source's fingerprint.

Figure 4.11(a) shows the number of bit errors of the target SRAM's fingerprint from those of the source SRAM (the "source errors") over time as the target ages. As expected, the number of source errors begins at a high value, with about 36% of the bits on the chip in error, and decreases over time at a decreasing rate while the number of target errors begins at 0 and increases over time at a decreasing rate. The decreasing rate of change of the fingerprint errors appears to have a logarithmic behavior much like [24] uses to characterize the change in V_T over time caused by NBTI.



(a) (b) (c)

Figure 4.11: Prediction of success rate for one SRAM's impersonation of another SRAM's fingerprint [119]. The fitted black lines in (a) and (b) can be combined to predict when the error rate drops low enough for the target SRAM to be able to impersonate the source SRAM. The horizontal line at 85% in (c) indicates the minimum success rate at which such an impersonation could be considered successful.

The success rate for attempting to authenticate the target SRAM as the source SRAM is not shown because it never rises above 0% during the experiment. Even so, Figure 4.11(b) implies that enough degradation could increase the success rate. Figure 4.11(b) shows, for each SRAM in the recovery experiment and for the target SRAM, the success rate of each SRAM after each day of stress against the number of bit errors from that SRAM's fresh fingerprint. It implies that the success rate is closely tied to the bit error count with little dependence on the pattern it stored while aging. The outlying points in Figure 4.11(b), labeled as "clone," correspond to chip S1, which was stressed while storing its fingerprint and was shown in Chapter 4.4.2 to have been affected the most by aging. The algorithm in [123] for reconstructing a fingerprint weighs each bit by its probability to be in error; a bit with a lower error probability has a higher weight than one with a higher one. The stable bits as defined by [122] are given the highest weights and are also likely the least affected by aging. Because of this, they also likely recover their skews the fastest during recovery. The result of this is that the overall number of errors recovers more slowly during recovery than the number of errors in bits with high weights during fingerprint reconstruction, leading to a higher success rate for the same overall error count. Patterns that are not based on the aging SRAM's fingerprint, as the uniform and random patterns are, have a lower effect on stable bits and thus produce similar relationships between error count and success rate.

Using Figure 4.11(a) to estimate a relationship between source error count and time and Figure 4.11(b) to estimate one between success rate and error count, it is possible to create a relationship between success rate and time as shown in Figure 4.11(c) and use it to predict how long the target SRAM must be aged to

Table 4.5: Clone Prediction Equation Parameters [119]

Equation	Parameter	Value
(4.8)	A_ϵ	93882
	B_ϵ	4405.6
	C_ϵ	112.29
(4.9)	A_S	8.03×10^{-4}
	B_S	18.9
(4.10)	A	3.24×10^{24}
	B	112.29
	C	3.54

successfully impersonate the source. The relationship is derived as follows:

$$\epsilon(t) = A_\epsilon - B_\epsilon \ln(1 + C_\epsilon t) \tag{4.8}$$

$$S(\epsilon) = \frac{1}{1 + e^{A_S \epsilon - B_S}} \tag{4.9}$$

$$\therefore S(t) = S(\epsilon(t)) = \frac{1}{1 + A(1 + Bt)^{-C}} \tag{4.10}$$

where A_ϵ, B_ϵ, C_ϵ, A_S, and B_S are fitting parameters, t is time, $A = e^{(A_S A_\epsilon - B_S)}$, $B = C_\epsilon$, and $C = A_S B_\epsilon$. Because the relationship between source errors, ϵ, and time appears to be logarithmic, like the behavior of threshold voltage shift caused by NBTI in a single transistor [24], a function of similar form was chosen for (4.8). The fitted values of the parameters for this experiment are shown in Table 4.5.

Equation (4.10) is illustrated in Figure 4.11(c). The horizontal line at 85% indicates the success rate below which an SRAM PUF is considered to no longer be reliable. In the case of cloning a fingerprint, a success rate above 85% will indicate that the target SRAM can reliably impersonate the source. Solving (4.10) for t with $S = 0.85$ suggests that it will take about 124,568 days to clone the source SRAM's fingerprint, which is about 341.3 years. Relaxing the requirement to a 50% success rate reduces the wait time to about 209 years. Simply getting any success at all takes over 50 years. So, while it is certainly possible to clone the fingerprint of an SRAM PUF using NBTI, it is impractical due to the large amount of time it will take.

The time it takes to perform this experiment can be reduced in three ways: by increasing the temperature to further accelerate NBTI, by increasing V_{DD} to further activate NBTI, and by choosing an SRAM with higher min-entropy to increase sensitivity to skew shift. Increasing temperature or voltage may be impractical due to availability of resources or, more importantly, because raising either one too high may cause damage to the chip or the devices inside. With the final option, an SRAM fabricated using smaller devices will generally have higher min-entropy than one using larger devices due to being operated at lower voltages and being more sensitive to noise. Additionally, the degradation caused by NBTI has been shown to be affected by

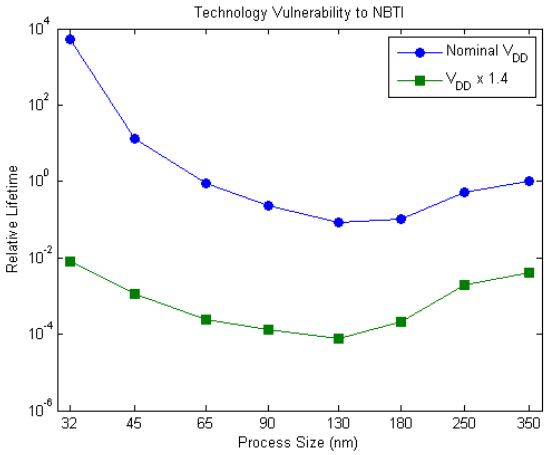


Figure 4.12: Device vulnerability to NBTI for process sizes ranging from 32 nm to 350 nm [119]. Vulnerability is expressed as lifetime under constant stress normalized to that value for a 350-nm device using process parameters taken from [131]. Circles express vulnerability using nominal V_{DD} while squares express vulnerability when V_{DD} is increased by 40% to activate NBTI.

silicon process [131]. Figure 4.12 shows the vulnerability of each process starting from 350 nm, which was used to fabricate the chips for the experiments in this work, through 32 nm using NBTI data and process parameter projections taken from [131]. As it shows, decreasing process size increases vulnerability to NBTI down to 130 nm. Below 130 nm, high-κ dielectrics enable thicker gates, reducing the oxide electric field and vulnerability to NBTI. Even so, devices are still more vulnerable than those at 350 nm until 32-nm technology. Using the data from Figure 4.12, it is possible to estimate how long it would take for SRAMs fabricated using smaller process sizes to be able to successfully impersonate an SRAM PUF. Repeating this experiment using a modern process size could take 6.75 years to create a reliable clone, 4.1 years to create a 50% reliable clone, and about one year for any successful impersonations.

4.5 Related Work

The driving concept behind this work, that the transistors in a 6T SRAM cell that are experiencing NBTI effects are determined by the data contained by the cell and can be controlled, is presented by [114]. The authors show that storing a value in a 6T SRAM cell causes NBTI stress that weakens one side of the cell. This can be used to improve the balance of imbalanced cells and even to repair cells that are so imbalanced that they cannot reliably read, write, or hold data. In a diagram like Figure 4.2(b), this corresponds to a shape where one of the wings is closed. Applying NBTI to weaken the side of the open wing will reduce that wing while opening the closed one, enabling the cell to be used to store data. The authors propose that this could be performed during the burn-in process to reduce minimum operational voltage for read, write, and

hold operations by up to 128, 75, and 91 mV, respectively, and increase overall yield. This work makes use of this process to control the skew of SRAM cells across a PUF and modify its fingerprint.

Another important principal in these experiments is shown by [118]. The authors of [118] show that natural recovery from NBTI, which occurs while no bias is applied to the gate of a device and while the device is typically at room temperature, is slow compared to stress. They also show that, even with repeated cycles of recovery, the permanent component of NBTI stress will accumulate and eventually render the device unusable. Using a ring oscillator implemented on an FPGA and measuring how its frequency changes with stress, the authors show that not only does increasing gate voltage increase NBTI stress, but applying an opposite bias to the gate improves the rate of recovery and reduces the permanent component of stress. This is known as active recovery. Both processes are accelerated by increasing temperature. In [46], the authors apply this principle by periodically applying active recovery to the ring oscillator. Doing so for only one hour every 31 hours can nearly entirely remove the permanent stress component and reduce the design margins for a given target lifetime by over 100×.

The fuzzy extractor algorithm defined in [123], which builds upon the concepts introduced in [112] and [113], is used extensively in this work. The authors implement their algorithm on an FPGA using SRAM as the fuzzy identifier. To hide the SRAM's fingerprint, a secret key is generated by applying a hash function from a family of hashes. The public key is generated using helper data from the algorithm; namely, an identifier indicating which hash function from the family was used, the output of the sketch function on the fingerprint, and a vector containing the probability of each bit of the SRAM to be in error from its fingerprint. Using the probability vector and sketch output, the original fingerprint of the SRAM PUF can be recreated from one reading and then the secret key can be recreated using the hash function selected with the identifier. The authors create a 128-bit secret key from 1536 bits of SRAM using their algorithm.

Then, in [124], they explore the data-dependent properties of SRAM aging and their effects on PUF reliability. With several different data patterns stored during stress, they discover that the best way to improve reliability is to stress a device while it stores the inverse of the pattern it contained at power-on while the best way to reduce reliability is to not invert that value. If it is not feasible to invert the entire SRAM's stored bit pattern at power-on, inverting part of it and storing that is almost as good if this can be periodically repeated and errors can be corrected. The metric the authors use for reliability comparison is bit error count rather than measuring actual success rate for identification. Additionally, they do not explore the effects of recovery, active or otherwise, on their methods for improving or reducing reliability.

The authors of [128] also explore techniques for improving the reliability of two types of memory-based PUFs: one based on SRAM and the other based on sense amplifiers. The first technique is similar to what [124] proposes, where directed aging is applied to reduce the number of errors in a PUF's response. By

applying accelerated and activated NBTI stress to an SRAM PUF storing the bit inverse of its fingerprint, they are able to reduce the error count by 40% after 120 hours. Next, they propose to reduce error count by evaluating the PUF several times after a challenge. By reading a PUF 1000 times and averaging the result to define its fingerprint and by reading it 40 times and averaging the result to define each response to a challenge, bit errors can be reduced to 0.5%. Finally, the authors propose to increase reliability by controlling the rate at which a PUF is activated. For SRAM, this means controlling the time it takes for V_{DD} to rise to nominal and for sense amplifiers this means controlling the time it takes for the enable signal to rise to 1. By increasing ramp time to 14 seconds, they are able to reduce bit errors to 0.67%. Chapter 4.4.3 explored an additional method of increasing reliability by applying active recovery to aged SRAM.

Reference [129] also explores the idea of taking advantage of data-dependent NBTI to increase the reliability of an SRAM PUF and add a step to balance the number of 1-skewed cells with those that are 0-skewed. The authors propose a method for measuring the number of cells that are 1-skewed and 0-skewed and balancing them by controlling the pattern stored in the device while it ages. Then, once the number of 1-skewed and 0-skewed cells are close to equal, the reliability of the PUF can be increased using a process similar to the one proposed in [128]. This work also does not account for the effects of recovery and only measures the effectiveness of their proposed method using simulations and not real hardware.

In [132], the authors evaluate the resilience of different kinds of PUFs against aging degradation. They find that ring oscillator PUFs, which define their fingerprints based on the differences between output frequencies of same-sized ring oscillators, and SRAM PUFs can degrade by 12.76% and 7%, respectively, in 4.5 years while arbiter PUFs and two-choose-one (TCO) PUFs only degrade 4.5% and 2.41% in 10 years. The authors also do not consider the effects of recovery in their work or verify their results using real hardware.

Finally, [130] presents an attack on an SRAM PUF in which the fingerprint is characterized and then modified using a focused ion beam (FIB). The authors characterize the fingerprint of an SRAM by measuring photon emissions from it during power-on. Using those emissions, the state of the SRAM after power-on, and then its fingerprint, can be reconstructed. After that, a FIB is used to expose the transistors of another SRAM and modify them so the skews of its cells match those of the original SRAM. This can be done by either destroying a transistor in a cell, causing it to always power on at the desired value but rendering it unable to write data, or by thinning a transistor and shifting the skew of the cell away from its corresponding value. The authors show that this can breach the security of an SRAM PUF, but it requires expensive machinery and may require the destruction of the target device. If that device is dual-used for security and data storage, the system making use of it may no longer function.

4.6 Applications of Directed NBTI to PUF Reliability

Several potential use cases for SRAM PUFs have been proposed in [104], including key protection for applications or critical assets, encryption of data, and device identification. All three of these have implications for IoT, such as protecting devices from unauthorized access, protecting data transmitted by devices from unauthorized access, and ensuring that a device on a network is who it claims to be. The process shown in Chapter 4.4.2 can be used as a denial-of-service attack, whereby an attacker with the ability to control the voltage and temperature conditions of a device using an SRAM as a PUF can effectively erase its fingerprint and remove it from the network. The result of such an attack would range from loss of owner access to a device to loss of telemetry data provided by a device to its manufacturer or loss of a critical component in a security system. An owner of a broken device can simply replace it, but loss of telemetry data or connection to a security device can cause monetary loss. Further, Figure 4.12 shows that using small process sizes for an SRAM can significantly increase its vulnerability to NBTI. This attack is least effective on the devices used for the experiment, but would be most effective on devices fabricated using 130 nm technology. Such an SRAM PUF could become unreliable in as little as 28.5 minutes if fabricated in 130 nm. Even though the high-κ dielectrics used for the gates of 90-nm through 45-nm transistors decrease the oxide electric field and reduce vulnerability to NBTI, they are still significantly more vulnerable than the devices used for experimentation. They would fail in 48.4 minutes and 7.1 hours, respectively. Devices manufactured in 32-nm technology have low enough oxide electric fields that they can resist NBTI more strongly.

Fortunately, work has been published to increase the resiliency of SRAM PUFs to aging degradation [124, 128, 129]. Chapter 4.4.3 showed a method by which active, accelerated recovery from NBTI can be used to quickly relieve stress on an SRAM PUF and restore its original fingerprint close enough to pass authentication. This can improve the reliability of a PUF beyond even what natural recovery is capable of. In all cases tested, simply powering down a device at room temperatures causes almost no improvement of fingerprint reconstruction success rate, while applying negative voltage recovered success rate above the reliability threshold within two days or less.

This method can be applied nearly transparently to the user of a device by taking advantage of inactive time, as suggested by [46]. Waiting until a device has been compromised by NBTI and then applying active, accelerated recovery could cause inconvenience, as the discovery of failure is likely to occur when an attempt is made to use the device which must be followed by a period of inoperability while it recovers. The authors of [46] propose that by proactively applying recovery techniques during times in which a device is not in use, its lifetime can be extended without this inconvenience. An SRAM PUF that is used as memory after

responding to a challenge can be proactively recovered along with the rest of the units on the device. A standalone PUF has a much larger opportunity for recovery; since it is only in use during authentication, its active recovery can start as soon as authentication is complete. With a proactive technique, the lifetime of an SRAM PUF can be extended significantly.

Chapter 5

Conclusion

In my work, I presented methods for estimating the lifetime, improving the reliability, and manipulating the security of electronic systems using high-level modeling and directed aging. With the rise in adoption of IoT and mobile devices that connect to large-scale remote servers to perform computations, the reliability of these devices has become increasingly more important both to maintain customer satisfaction [12] and reduce maintenance and replacement costs [13]. These problems are exacerbated by aging, which causes the slow degradation of transistor parameters over time. This occurs through several mechanisms, including NBTI, electromigration, HCI, and TDDB, that affect a device or circuit in different ways but generally cause reduced performance and eventually circuit failure. Worse, all of these mechanisms accelerate at high temperatures that are caused by the lack of comparable scaling between supply voltage and transistor size [5]. The typical method for accounting for aging in electronics design is to add static timing margins [35, 36], but this is unsatisfactory because it requires designing for worst-case degradation when the typical case is often much less, as shown by Figure 1.4 [34]. These factors have created great interest in modeling aging [20–23] and developing techniques to manage it or reduce its effects [35, 39–42, 118].

In order to evaluate these management techniques and measure their effects on a design, it is necessary to simulate them. The most accurate results come from circuit-level simulations using device-level models, but the high complexity of modern designs with hundreds of millions to billions of transistors renders that infeasible. Furthermore, reliability concerns can have repercussions at all stages of design, including stages such as architecture design where RTL may not yet have been created [6]. Therefore, high-level, pre-RTL simulation tools are necessary to reduce simulation overhead and enable early-stage reliability choices to be made. Such tools exist for performance [53], power, area [54], temperature [11], and voltage noise [10], with efforts being made to develop architecture-level models and tools for lifetime and reliability [30, 40, 49, 62].

5.1 Summary of Contributions

Chapter 2 introduces a new lifetime simulation tool called "OldSpot," which can be integrated into an existing flow containing power, performance, and area simulators [58]. Earlier tools make assumptions about the organization of architectural units across an electronic system and what types of failures can be tolerated. By breaking a system down into a "failure dependency graph," where nodes represent units and groups of units in the system and edges represent how failures propagate, it is possible to remove both of these assumptions. For example, OldSpot can simulate a CPU that operates alongside several accelerators where failure in any combination of the accelerators is tolerable, as failed accelerators' workloads can be completed at reduced performance by the CPU, but the CPU's failure cannot be tolerated. Such a system can be represented using a diagram like Figure 2.2. I showed how OldSpot can be used to model structural duplication at the architectural unit level and achieve the same lifetime as a one-core-out-of-four failure-tolerant system using a zero-out-of-three system with less area cost.

In order to accurately estimate aging, simulations of power, performance, and temperature are necessary. In Chapter 3, I presented a flow of pre-RTL simulation tools that enable high-level estimation of these quantities and can be used with OldSpot. An important aspect of each of these tools is their free availability with open-source licenses. This facilitates architecture research by removing the need for expensive licenses that can take time to acquire and allows modification for a researcher's own needs. The RISC-V ISA [71] addresses this by being open-source and modular to enable collaboration and customization. To facilitate designs using RISC-V, I created a limited implementation for it in the gem5 simulator [53] that supports single-threaded execution with system call emulation. This allows the simulation flow to include RISC-V designs, which will be useful for architects who wish to use RISC-V.

Degradation due to aging does not only have an effect on timing requirements and circuit failures, but can also affect security. Because of their low power requirements and high performance, PUFs have great potential for ensuring security and identity for low-power devices such as mobile and IoT devices. But since they rely on silicon variations to create uniqueness and randomness which affect many of the same parameters that are vulnerable to aging, their functionality can be significantly impacted. In Chapter 4, I explored the potential for using directed NBTI aging to reduce the success rate of a PUF based on an SRAM to reconstruct its fingerprint, effectively "erasing" it. Using active, accelerated aging, the success rate was reduced from 100% to 0% in less than three days and remained 0% after one month of relaxation while powered off. While this suggests that natural recovery will not normally restore a PUF's functionality if it has been damaged by aging, I then showed that applying a small negative voltage to actively recover NBTI as proposed by [46, 118]

can restore PUF functionality within two days. As indicated by the results in [46], applying this treatment at frequent enough intervals could extend the lifetime of the PUF nearly indefinitely. Finally, to explore the limits of potential use cases of directed NBTI aging in an SRAM PUF, I attempted to coerce the fingerprint of one SRAM to resemble that of another and developed a model to predict the amount of time required for successful "cloning." With the commercial SRAM ICs used in the experiment, it could take about 340 years to create a successful clone, but by using newer process nodes it could take as little as one year.

5.2 Future Work

While some aging mechanisms exhibit some form of recovery or self-healing, particularly NBTI [18] and electromigration [31], little work has been done on modeling or measuring the potential of active recovery using reversed voltages or currents. Prior work exists at the device and circuit level [46, 118, 119], but no architecture-level tools yet allow designers to explore the possibility of incorporating it into their designs. OldSpot's unique way of representing systems at the architectural unit level enables easy extension to include passive and active recovery. Since it has been shown that periodic application of active recovery, for example during hours of low to no usage while a device's owner is asleep, can extend lifetime nearly indefinitely, this presents an interesting mechanism designers can take advantage of to increase device lifetime.

Further improvements are also necessary to fully include RISC-V in the simulation flow presented in Chapter 3. At the time of this writing, gem5 only supports single-threaded RISC-V binaries executing in system call emulation mode. Support for multithreaded binaries requires either native gem5 support for existing threading libraries such as `pthreads` or extension of the special gem5 threading library called `m5threads` [133]. Full-system simulation will require full support for the RISC-V privileged architecture [87] as system call emulation only requires support for user-level code. It will also require implementation of gem5's special pseudo-instructions that allow control of the simulation and interaction with the host machine and its file system from within a full-system simulation. This will enable power and performance simulation of more realistic workloads using RISC-V and facilitate its use with OldSpot.

5.3 Final Remarks

Open-source, high-level modeling tools allow architecture researchers to rapidly iterate and explore design space while avoiding the overheads of implementing and simulating RTL and the impediments of proprietary licensing. They also allow architects to perform cross-layer design exploration, measuring the effects of their ideas on low-level metrics like temperature and voltage noise and, conversely, enabling them to evaluate how

those ideas are affected by these low-level effects. Tools for measuring power, performance, temperature, and voltage noise are already mature. The addition of OldSpot improves architects' abilities to measure lifetime as well. By adding RISC-V to gem5, it is now possible to perform these analyses while entirely avoiding licensing issues, facilitating collaboration between researchers in industry and academia. RISC-V is a rising star in the hardware design community, and increasing the options for simulating and making design choices with it can only help to improve adoption. With all of these new and continually-improving options for designing computing hardware, electronic systems can become more resilient to the worsening effects of aging.

Appendix A

List of Publications

Conference Publications

[C1] Alec Roelke, Xinfei Guo, and Mircea R Stan. Oldspot: A pre-RTL model for aging and lifetime optimization. IEEE, 2018. To be submitted.

[C2] Alec Roelke and Mircea Stan. Co-optimizing CPUs and accelerators in constrained systems. In *System-on-Chip Conference (SOCC), 2018 31st IEEE International*. IEEE, 2018. Under review.

[C3] Alec Roelke, Runjie Zhang, Kaushik Mazumdar, Ke Wang, Kevin Skadron, and Mircea R Stan. Pre-RTL voltage and power optimization for low-cost, thermally challenged multicore chips. In *2017 IEEE 35th International Conference on Computer Design (ICCD)*, pages 597-600. IEEE, 2017.

[C4] Alec Roelke and Mircea R Stan. Attacking an SRAM-based PUF through wearout. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 206-211. IEEE, 2016.

Journal Articles

[J1] Alec Roelke and Mircea R Stan. Controlling the reliability of SRAM PUFs with directed NBTI aging and recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018. Under review.

Workshop Papers and Presentations

[W1] Alec Roelke and Mircea R Stan. RISC5: Improving support for RISC-V in gem5. The RISC-V Foundation, 7th RISC-V Workshop, November 2017.

[W2] Alec Roelke and Mircea R. Stan. RISC5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV 2017)*, 2017.

Appendix B

Overall Aging Rate Computation

In order to compute the overall reliability, $R(t)$, of a system undergoing multiple simultaneous aging processes, the sum-of-failure-rates (SOFR) model [30] can be used. The SOFR model assumes that the failure of any component means the failure of the entire system, so the reliability of the system is the product of the reliability of each component, $R_i(i)$. The overall aging rate of the system, η, can be computed from the aging rate of each component, η_i, assuming all of them follow a Weibull distribution with the same shape parameter, β:

$$R(t) = \prod_{i=1}^{N} R_i(t) \tag{B.1}$$

$$e^{-\left(\frac{t}{\eta}\right)^{\beta}} = \prod_{i=1}^{N} e^{-\left(\frac{t}{\eta_i}\right)^{\beta}} \tag{B.2}$$

$$= \exp\left(-\sum_{i=1}^{N}\left(\frac{t}{\eta_i}\right)^{\beta}\right) \tag{B.3}$$

$$\exp\left(-t^{\beta}\frac{1}{\eta^{\beta}}\right) = \exp\left(-t^{\beta}\sum_{i=1}^{N}\frac{1}{\eta_i^{\beta}}\right) \tag{B.4}$$

$$\therefore \frac{1}{\eta^{\beta}} = \sum_{i=1}^{N}\frac{1}{\eta_i^{\beta}} \tag{B.5}$$

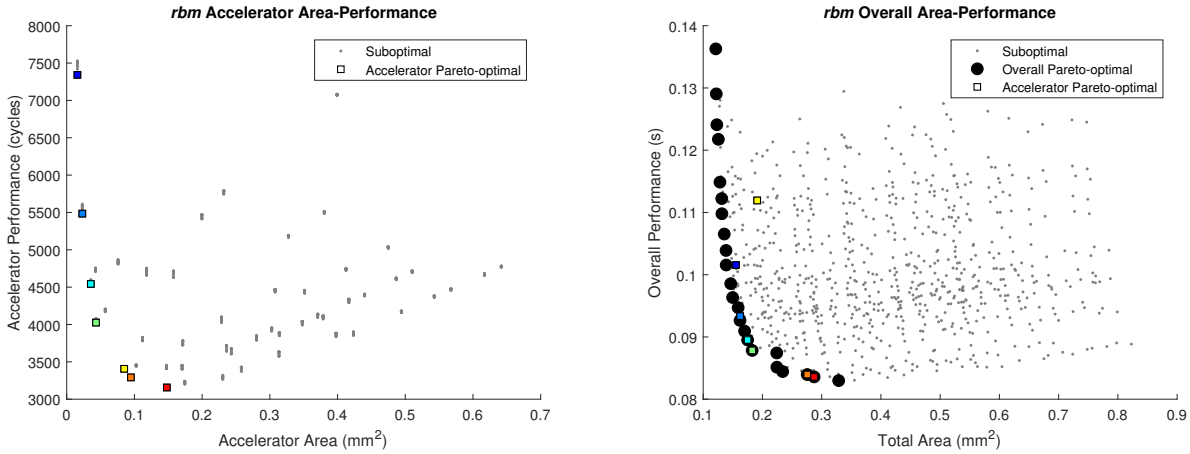
$$\eta = \left(\sum_{i=1}^{N}\frac{1}{\eta_i^{\beta}}\right)^{-\frac{1}{\beta}} \tag{B.6}$$

This can be used to compute the average aging rate of a single unit undergoing several aging mechanisms (see Chapter 1.1) or the aging rate of a system that fails when any of its component fails.

Appendix C

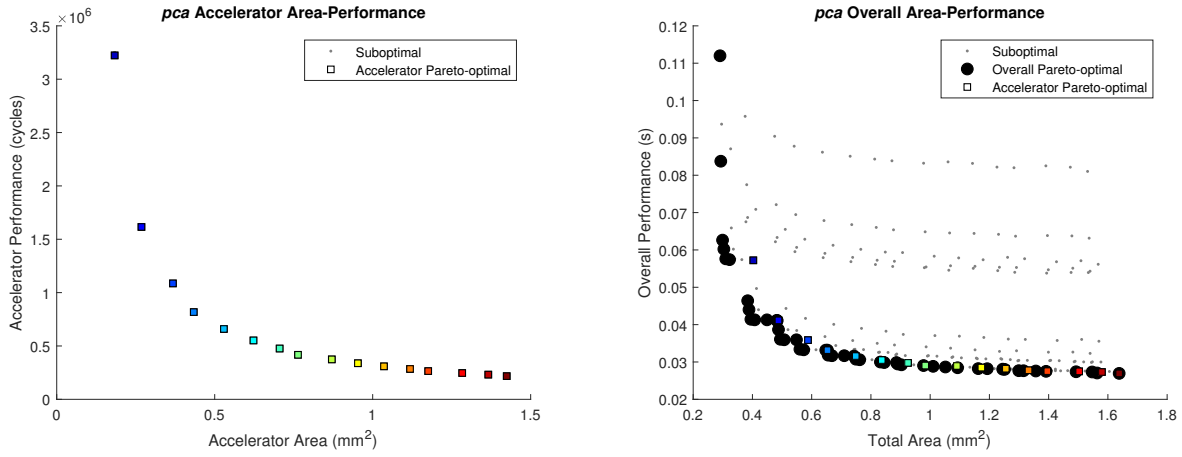
Gem5-Aladdin Results

Area-performance Pareto Frontiers



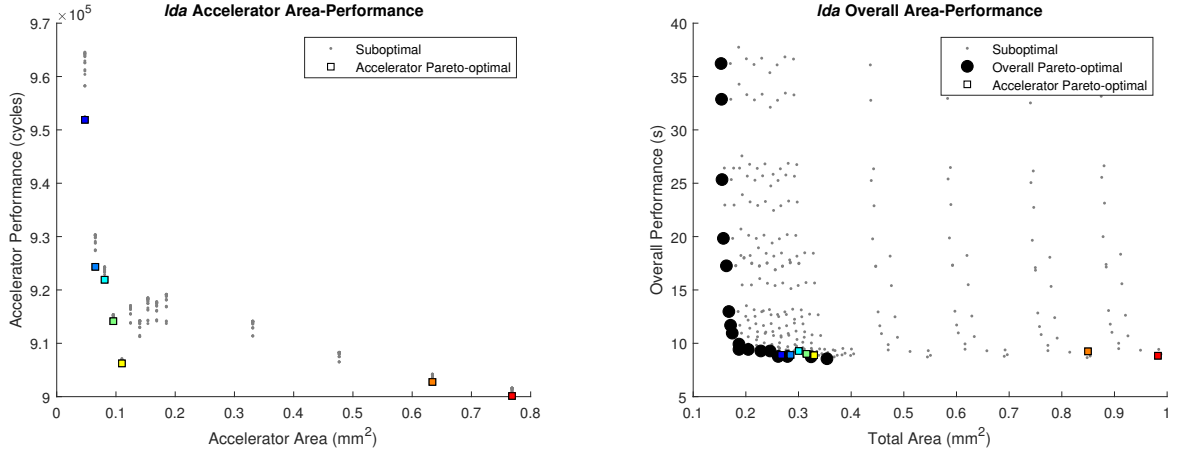
(a) (b)

Figure C.1: Pareto-optimal frontiers for *rbm* area and performance; (a) shows Pareto-optimal designs for a standalone matrix-multiplication accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.



(a) (b)

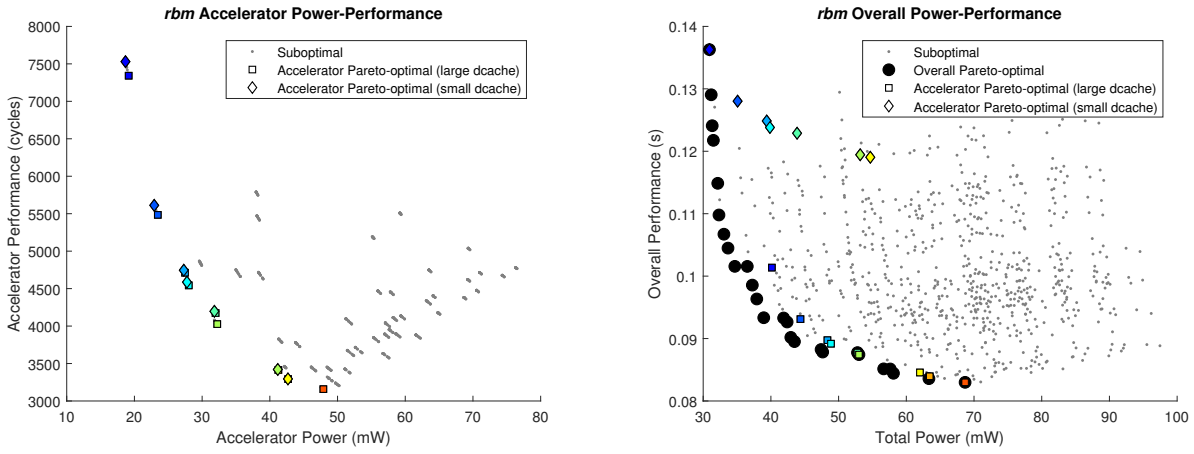
Figure C.2: Pareto-optimal frontiers for *pca* area and performance; (a) shows Pareto-optimal designs for a standalone covariance matrix accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.



(a) (b)

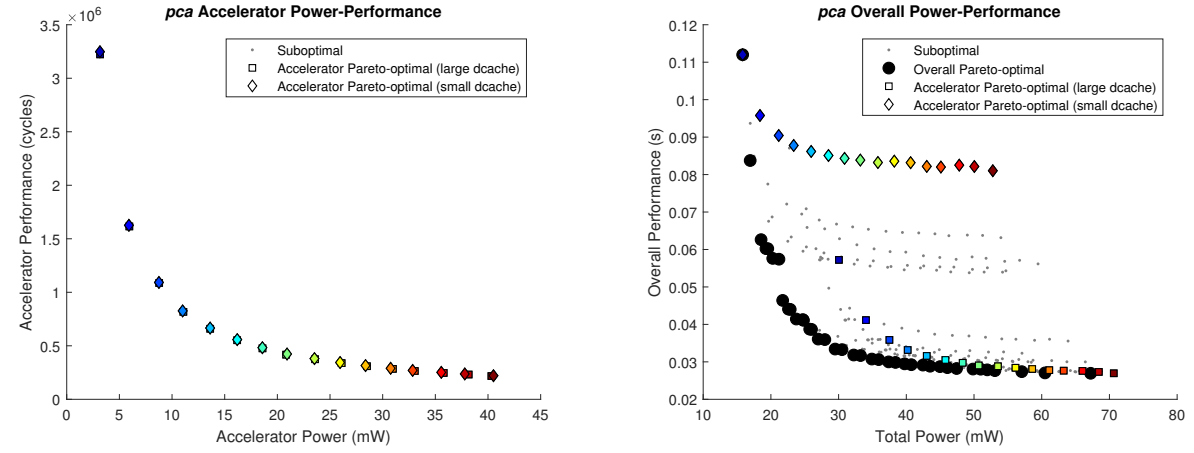
Figure C.3: Pareto-optimal frontiers for *lda* area and performance; (a) shows Pareto-optimal designs for a standalone maximum-likelihood estimation model accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.

Power-performance Pareto Frontiers



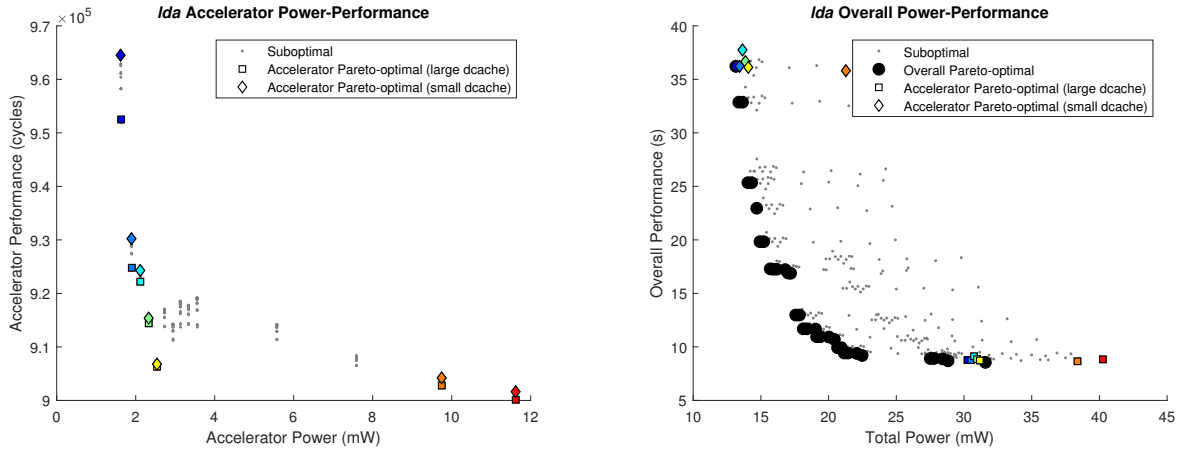
(a) (b)

Figure C.4: Pareto-optimal frontiers for *rbm* power and performance; (a) shows Pareto-optimal designs for a standalone matrix-multiplication accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.



(a) (b)

Figure C.5: Pareto-optimal frontiers for *pca* power and performance; (a) shows Pareto-optimal designs for a standalone covariance matrix accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.



(a) (b)

Figure C.6: Pareto-optimal frontiers for *lda* area and performance; (a) shows Pareto-optimal designs for a standalone maximum-likelihood estimation model accelerator designed while considering external effects and (b) shows Pareto-optimal designs for a co-designed RISC-V CPU and accelerator. Boxes of the same color in (a) and (b) correspond to the same design point.

Bibliography

[1] David Chappell. Introducing the windows azure platform. Technical report, Microsoft Corporation, 2010.

[2] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[3] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[4] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[5] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 365–376. ACM, 2011.

[6] Karthik Swaminathan, Nandhini Chandramoorthy, Chen-Yong Cher, Ramon Bertran, Alper Buyuktosunoglu, and Pradip Bose. BRAVO: Balanced reliability-aware voltage optimization. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 97–108. IEEE, 2017.

[7] Subhasish Mitra, Pradip Bose, Eric Cheng, Chen-Yong Cher, Hyungmin Cho, Rajiv Joshi, Young Moon Kim, Charles R Lefurgy, Yanjing Li, Kenneth P Rodbell, et al. The resilience wall: Cross-layer solution strategies. In *VLSI Technology, Systems and Application (VLSI-TSA), Proceedings of Technical Program-2014 International Symposium on*, pages 1–11. IEEE, 2014.

[8] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18. IEEE, 2003.

[9] George A Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I August, and Shubhendu S Mukherjee. Software-controlled fault tolerance. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(4):366–396, 2005.

[10] Runjie Zhang, Ke Wang, Brett H Meyer, Mircea R Stan, and Kevin Skadron. Architecture implications of pads as a scarce resource. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 373–384. IEEE Press, 2014.

[11] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006.

[12] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[13] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[14] D Blaauw, D Sylvester, P Dutta, Y Lee, I Lee, S Bang, Y Kim, G Kim, P Pannuto, Y-S Kuo, et al. Iot design space challenges: Circuits and systems. In *VLSI Technology (VLSI-Technology): Digest of Technical Papers, 2014 Symposium on*, pages 1–2. IEEE, 2014.

[15] Ed Sperling. Chip aging accelerates. *Semiconductor Engineering*.

[16] Sheldon X-D Tan, Hussam Amrouch, Taeyoung Kim, Zeyu Sun, Chase Cook, and Jörg Henkel. Recent advances in EM and BTI induced reliability modeling, analysis and optimization. *Integration, the VLSI Journal*, 2017.

[17] KN Tu, Yingxia Liu, and Menglu Li. Effect of joule heating and current crowding on electromigration in mobile technology. *Applied Physics Reviews*, 4(1):011101, 2017.

[18] Kaustubh Joshi, Subhadeep Mukhopadhyay, Nilesh Goel, and Souvik Mahapatra. A consistent physical framework for N and P BTI in HKMG MOSFETs. In *Reliability Physics Symposium (IRPS), 2012 IEEE International*, pages 5A–3. IEEE, 2012.

[19] James R Black. Electromigration—a brief survey and some recent results. *IEEE Transactions on Electron Devices*, 16(4):338–347, 1969.

[20] Xin Huang, Tan Yu, Valeriy Sukharev, and Sheldon X-D Tan. Physics-based electromigration assessment for power grid networks. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[21] E Wu, J Sune, W Lai, E Nowak, J McKenna, A Vayshenker, and D Harmon. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides. *Solid-State Electronics*, 46(11):1787–1798, 2002.

[22] Tibor Grasser. *Hot Carrier Degradation in Semiconductor Devices*. Springer, 2014.

[23] S Mahapatra, N Goel, S Desai, S Gupta, B Jose, S Mukhopadhyay, K Joshi, A Jain, AE Islam, and MA Alam. A comparative study of different physics-based NBTI models. *IEEE Transactions on Electron Devices*, 60(3):901–916, 2013.

[24] Jyothi Bhaskarr Velamala, Ketul B Sutaria, Hirofumi Shimizu, Hiromitsu Awano, Takashi Sato, Gilson Wirth, and Yu Cao. Compact modeling of statistical BTI under trapping/detrapping. *IEEE Transactions on Electron Devices*, 60(11):3645–3654, 2013.

[25] C Shen, M-F Li, CE Foo, T Yang, DM Huang, A Yap, GS Samudra, and Y-C Yeo. Characterization and physical origin of fast vth transient in nbti of pmosfets with sion dielectric. In *Electron Devices Meeting, 2006. IEDM'06. International*, pages 1–4. IEEE, 2006.

[26] JF Zhang, Z Ji, MH Chang, B Kaczer, and G Groeseneken. Real vth instability of pmosfets under practical operation conditions. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 817–820. IEEE, 2007.

[27] GA Du, DS Ang, ZQ Teo, and YZ Hu. Ultrafast measurement on nbti. *IEEE Electron Device Letters*, 30(3):275–277, 2009.

[28] Jens Lienig. Electromigration and its impact on physical design in future technologies. In *Proceedings of the 2013 ACM International symposium on Physical Design*, pages 33–40. ACM, 2013.

[29] Changsup Ryu, Kee-Won Kwon, Alvin LS Loke, Haebum Lee, Takeshi Nogami, Valery M Dubin, Rahim A Kavari, Gary W Ray, and S Simon Wong. Microstructure and reliability of copper interconnects. *IEEE transactions on electron devices*, 46(6):1113–1120, 1999.

[30] Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. The case for lifetime reliability-aware microprocessors. In *ACM SIGARCH Computer Architecture News*, volume 32, page 276. IEEE Computer Society, 2004.

[31] Xin Huang, Valeriy Sukharev, Taeyoung Kim, Haibao Chen, and Sheldon X-D Tan. Electromigration recovery modeling and analysis under time-dependent current and temperature stressing. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 244–249. IEEE, 2016.

[32] Xiaojun Li, Jin Qin, and Joseph B Bernstein. Compact modeling of MOSFET wearout mechanisms for circuit-reliability simulation. *IEEE Transactions on Device and Materials Reliability*, 8(1):98–121, 2008.

[33] Wenping Wang, Vijay Reddy, Anand T Krishnan, Rakesh Vattikonda, Srikanth Krishnan, and Yu Cao. Compact modeling and simulation of circuit reliability for 65-nm CMOS technology. *IEEE Transactions on Device and Materials Reliability*, 7(4):509–517, 2007.

[34] Eric Karl, David Blaauw, Dennis Sylvester, and Trevor Mudge. Multi-mechanism reliability modeling and management in dynamic systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(4):476–487, 2008.

[35] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 99. ACM, 2013.

[36] Kunhyuk Kang, Saakshi Gangwal, Sang Phill Park, and Roy Kaushik. NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution? In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 726–731. IEEE, 2008.

[37] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. Adaptive techniques for overcoming performance degradation due to aging in digital circuits. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pages 284–289. IEEE, 2009.

[38] Hassan Mostafa, Mohab Anis, and Mohamed Elmasry. NBTI and process variations compensation circuits using adaptive body bias. *IEEE transactions on semiconductor manufacturing*, 25(3):460–467, 2012.

[39] Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 520–531. IEEE, 2005.

[40] Cristiana Bolchini, Matteo Carminati, Marco Gribaudo, and Antonio Miele. A lightweight and open-source framework for the lifetime estimation of multicore systems. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 166–172. IEEE, 2014.

[41] Cesare Ferri, Dimitra Papagiannopoulou, R Iris Bahar, and Andrea Calimera. NBTI-aware data allocation strategies for scratchpad memory based embedded systems. In *Test Workshop (LATW), 2011 12th Latin American*, pages 1–6. IEEE, 2011.

[42] Lin Huang, Feng Yuan, and Qiang Xu. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 51–56. IEEE, 2009.

[43] John Sartori and Rakesh Kumar. Software canaries: software-based path delay fault testing for variation-aware energy-efficient design. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 159–164. ACM, 2014.

[44] Farshad Firouzi, Fangming Ye, Krishnendu Chakrabarty, and Mehdi B Tahoori. Representative critical-path selection for aging-induced delay monitoring. In *Test Conference (ITC), 2013 IEEE International*, pages 1–10. IEEE, 2013.

[45] Deepashree Sengupta and Sachin S Sapatnekar. Predicting circuit aging using ring oscillators. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 430–435. IEEE, 2014.

[46] Xinfei Guo and Mircea R Stan. Work hard, sleep well-avoid irreversible IC wearout with proactive rejuvenation. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 649–654. IEEE, 2016.

[47] Lin Huang and Qiang Xu. Characterizing the lifetime reliability of manycore processors with core-level redundancy. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 680–685. IEEE, 2010.

[48] Alec Roelke, Runjie Zhang, Kaushik Mazumdar, Ke Wang, Kevin Skadron, and Mircea R Stan. Pre-RTL voltage and power optimization for low-cost, thermally challenged multicore chips. In *2017 IEEE 35th International Conference on Computer Design (ICCD)*, pages 597–600. IEEE, 2017.

[49] Tushar Gupta, Clément Bertolini, Olivier Héron, Nicolas Ventroux, T Zimmer, and F Marc. Effects of various applications on relative lifetime of processor cores. In *Integrated Reliability Workshop Final Report, 2009. IRW'09. IEEE International*, pages 132–135. IEEE, 2009.

[50] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 577–589. IEEE, 2015.

[51] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy evaluation of gem5 simulator system. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7. IEEE, 2012.

[52] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 97–108. IEEE Press, 2014.

[53] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[54] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[55] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. Accurate and stable run-time power modeling for mobile and embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1):106–119, 2017.

[56] Donggyu Kim, Adam Izraelevitz, Christopher Celio, Hokeun Kim, Brian Zimmer, Yunsup Lee, Jonathan Bachrach, and Krste Asanović. Strober: fast and accurate sample-based energy simulation for arbitrary RTL. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 128–139. IEEE Press, 2016.

[57] Daniel E Holcomb, Wayne P Burleson, and Kevin Fu. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, 2009.

[58] Alec Roelke, Xinfei Guo, and Mircea R Stan. Oldspot: A pre-RTL model for aging and lifetime optimization. IEEE, 2018. To be submitted.

[59] Lawrence M Leemis. *Reliability: probabilistic models and statistical methods*. Prentice-Hall, Inc., 1995.

[60] Yun Xiang et al. System-level reliability modeling for MPSoCs. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 297–306. ACM, 2010.

[61] Failure mechanisms and models for semiconductor devices. Technical Report JEP122H, JEDEC Solid State Technology Institution, September 2016.

[62] Lin Huang and Qiang Xu. Agesim: A simulation framework for evaluating the lifetime reliability of processor-based socs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 51–56. European Design and Automation Association, 2010.

[63] Fabian Oboril et al. ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[64] Jeonghee Shin, Victor Zyuban, Zhigang Hu, Jude A Rivers, and Pradip Bose. A framework for architecture-level lifetime reliability modeling. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 534–543. IEEE, 2007.

[65] Ramachandra Ramakumar. *Engineering reliability: fundamentals and applications*. Prentice Hall, 1993.

[66] Hussam Amrouch, Victor M van Santen, Thomas Ebi, Volker Wenzel, and Jörg Henkel. Towards interdependencies of aging mechanisms. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pages 478–485. IEEE Press, 2014.

[67] Trevor E Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. An evaluation of high-level mechanistic core models. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3):28, 2014.

[68] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[69] Steven Cameron Woo et al. The SPLASH-2 programs: Characterization and methodological considerations. In *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, pages 24–36. IEEE, 1995.

[70] Gregory G Faust, Runjie Zhang, Kevin Skadron, Mircea R Stan, and Brett H Meyer. ArchFP: Rapid prototyping of pre-RTL floorplans. In *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*, pages 183–188. IEEE, 2012.

[71] Krste Asanović and Andrew Waterman. The RISC-V instruction set manual, volume I: User-level ISA, version 2.2. Technical report, EECS Department, University of California, Berkeley, May 2017.

[72] Alec Roelke and Mircea R. Stan. RISC5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV 2017)*, 2017.

[73] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News*, volume 30, pages 45–57. ACM, 2002.

[74] Runjie Zhang, Mircea R Stan, and Kevin Skadron. HotSpot 6.0: Validation, acceleration and extension. Technical Report CS-2015-04, University of Virginia, 2015.

[75] Krste Asanović et al. The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[76] Christopher Celio, David A. Patterson, and Krste Asanović. The berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor. Technical Report UCB/EECS-2015-167, EECS Department, University of California, Berkeley, Jun 2015.

[77] Roland E Wunderlich, Thomas F Wenisch, Babak Falsafi, and James C Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 84–95. IEEE, 2003.

[78] John L Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[79] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B Brockman, and Norman P Jouppi. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 694–701. IEEE, 2011.

[80] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[81] Krste Asanović and David A. Patterson. Instruction sets should be free: The case for RISC-V. Technical Report UCB/EECS-2014-146, EECS Department, University of California, Berkeley, Aug 2014.

[82] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1216–1225. ACM, 2012.

[83] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the scala programming language. Technical report, 2004.

[84] Andrew Waterman and Yunsup Lee. RISC-V ISA simulator. `https://github.com/riscv/riscv-isa-sim`, 2011–2018.

[85] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[86] Michael J Clark. rv8: RISC-V simulator for x86-64. `https://rv8.io/`, 2017–2018.

[87] Andrew Waterman and Krste Asanović. The RISC-V instruction set manual volume II: Privileged architecture version 1.10. Technical report, The RISC-V Foundation, May 2017. Draft.

[88] Tony Chen and David A. Patterson. RISC-V geneology. Technical Report UCB/EECS-2016-6, EECS Department, University of California, Berkeley, Jan 2016.

[89] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. *Memory consistency and event ordering in scalable shared-memory multiprocessors*, volume 18. ACM, 1990.

[90] Alec Roelke and Mircea R Stan. RISC5: Improving support for RISC-V in gem5. The RISC-V Foundation, 7th RISC-V Workshop, November 2017.

[91] Dan Zuras, Mike Cowlishaw, Alex Aiken, Matthew Applegate, David Bailey, Steve Bass, Dileep Bhandarkar, Mahesh Bhat, David Bindel, Sylvie Boldo, et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[92] Arm infocenter, 2004.

[93] Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[94] Alec Roelke and Mircea Stan. Co-optimizing CPUs and accelerators in constrained systems. In *System-on-Chip Conference (SOCC), 2018 31st IEEE International*. IEEE, 2018. Under review.

[95] Jason Cong, Zhenman Fang, Michael Gill, and Glenn Reinman. PARADE: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 380–387. IEEE, 2015.

[96] Shelby Thomas, Chetan Gohkale, Enrico Tanuwidjaja, Tony Chong, David Lau, Saturnino Garcia, and Michael Bedford Taylor. CortexSuite: A synthetic brain benchmark suite. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 76–79. IEEE, 2014.

[97] Yakun Sophia Shao and David Brooks. Isa-independent workload characterization and its implications for specialized architectures. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 245–255. IEEE, 2013.

[98] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM, 2008.

[99] Joseph F Hair, William C Black, Barry J Babin, Rolph E Anderson, Ronald L Tatham, et al. *Multivariate data analysis*, volume 5. Prentice hall Upper Saddle River, NJ, 1998.

[100] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[101] Rekha Govindaraj and Swaroop Ghosh. A strong arbiter PUF using resistive RAM within 1T-1R memory architecture. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pages 141–148. IEEE, 2016.

[102] Ujjwal Guin, Ke Huang, Daniel DiMase, John M Carulli, Mohammad Tehranipoor, and Yiorgos Makris. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proceedings of the IEEE*, 102(8):1207–1228, 2014.

[103] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[104] PUF–physical unclonable functions. Technical Report 9397 750 17366, NXP Semiconductor, February 2013.

[105] Ulrich Rührmair and Daniel E Holcomb. PUFs at a glance. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[106] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160. ACM, 2002.

[107] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[108] Anas Mazady, Md Tauhidur Rahman, Domenic Forte, and Mehdi Anwar. Memristor PUF—a security primitive: Theory and experiment. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):222–229, 2015.

[109] Soroush Khaleghi, Paolo Vinella, Soumya Banerjee, and Wenjing Rao. An STT-MRAM based strong PUF. In *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*, pages 129–134. IEEE, 2016.

[110] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual design automation conference*, pages 9–14. ACM, 2007.

[111] Srinivas Devadas, Edward Suh, Sid Paral, Richard Sowell, Tom Ziola, and Vivek Khandelwal. Design and implementation of PUF-based "unclonable" RFID ICs for anti-counterfeiting and security applications. In *RFID, 2008 IEEE International conference on*, pages 58–64. IEEE, 2008.

[112] Xavier Boyen. Reusable cryptographic fuzzy extractors. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 82–91. ACM, 2004.

[113] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *International conference on the theory and applications of cryptographic techniques*, pages 523–540. Springer, 2004.

[114] Jiajing Wang, Satyanand Nalam, Zhenyu Qi, Randy W Mann, Mircea Stan, and Benton H Calhoun. Improving SRAM Vmin and yield by using variation-aware BTI stress. In *Custom Integrated Circuits Conference (CICC), 2010 IEEE*, pages 1–4. IEEE, 2010.

[115] Mafalda Cortez, Apurva Dargar, Said Hamdioui, and Geert-Jan Schrijen. Modeling sram start-up behavior for physical unclonable functions. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*, pages 1–6. IEEE, 2012.

[116] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. A soft decision helper data algorithm for sram pufs. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 2101–2105. IEEE, 2009.

[117] Alec Roelke and Mircea R Stan. Attacking an SRAM-based PUF through wearout. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 206–211. IEEE, 2016.

[118] Xinfei Guo, Wayne Burleson, and Mircea Stan. Modeling and experimental demonstration of accelerated self-healing techniques. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[119] Alec Roelke and Mircea R Stan. Controlling the reliability of SRAM PUFs with directed NBTI aging and recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018. Under review.

[120] Wei Zhao, Frank Liu, Kanak Agarwal, Dhruva Acharyya, Sani R Nassif, Kevin J Nowka, and Yu Cao. Rigorous extraction of process variations for 65-nm cmos design. *IEEE Transactions on Semiconductor Manufacturing*, 22(1):196–203, 2009.

[121] Alliance Memory. *8K x 8 Bit Low Power CMOS SRAM*, February 2007. Rev. 1.0.

[122] Kan Xiao, Md Tauhidur Rahman, Domenic Forte, Yu Huang, Mei Su, and Mohammad Tehranipoor. Bit selection algorithm suitable for high-volume production of SRAM-PUF. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 101–106. IEEE, 2014.

[123] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In *CHES*, volume 9, pages 332–347. Springer, 2009.

[124] Roel Maes and Vincent van der Leest. Countering the effects of silicon aging on SRAM PUFs. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 148–153. IEEE, 2014.

[125] IDT. *CMOS Static RAM 256K (32K x 8-Bit)*, September 2013.

[126] Geert-Jan Schrijen and Vincent van der Leest. Comparative analysis of SRAM memories used as PUF primitives. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1319–1324. EDA Consortium, 2012.

[127] Abhranil Maiti, Vikash Gunreddy, and Patrick Schaumont. A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded systems design with FPGAs*, pages 245–267. Springer, 2013.

[128] Mudit Bhargava, Cagla Cakir, and Ken Mai. Reliability enhancement of bi-stable PUFs in 65nm bulk CMOS. In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pages 25–30. IEEE, 2012.

[129] Achiranshu Garg and Tony T Kim. Design of SRAM PUF with improved uniformity and reliability utilizing device aging effect. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 1941–1944. IEEE, 2014.

[130] Clemens Helfmeier, Christian Boit, Dmitry Nedospasov, and Jean-Pierre Seifert. Cloning physically unclonable functions. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 1–6. IEEE, 2013.

[131] Rakesh Vattikonda, Wenping Wang, and Yu Cao. Modeling and minimization of PMOS NBTI effect for robust nanometer design. In *Proceedings of the 43rd annual Design Automation Conference*, pages 1047–1052. ACM, 2006.

[132] Mohd Syafiq Mispan, Basel Halak, and Mark Zwolinski. NBTI aging evaluation of puf-based differential architectures. In *On-Line Testing and Robust System Design (IOLTS), 2016 IEEE 22nd International Symposium on*, pages 103–108. IEEE, 2016.

[133] Daniel Sanchez. m5threads: a pthread library for the M5 simulator. `https://github.com/gem5/m5threads`, 2009.