Runtime Monitoring for Dependable Hardware Design

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Arunkumar Vijayan

aus Pandalam, India

Tag der mündlichen Prüfung: 06.06.2019

Erster Gutachter: Prof. Dr. Mehdi B. Tahoori, KIT
Zweiter Gutachter: Prof. Krishnendu Chakrabarty, Duke University

Acknowledgement

First of all, I would like to express my immense gratitude to my advisor Prof. Mehdi Tahoori for his relentless support throughout my PhD. His guidance helped me in overcoming several obstacles in my research. In addition, I am extremely grateful to my co-advisor Prof. Krishnendu Chakrabarty in supporting me throughout my research with guidance and motivation.

I would like to thank all of my colleagues in the Chair of Dependable Nano Computing at Karlsruhe Institute of Technology for their technical as well as personal support. I especially thank Dr. Saman Kiamehr for his invaluable support as a friend and a mentor. I would also like to thank Dr. Fabian Oboril, Dr. Mojtaba Ebrahimi and Dr. Farshad Firouzi for their contributions to my research and for the numerous technical discussions we had.

Words cannot express my thankfulness to my wonderful colleagues Sarath Mohanachandran, Dr. Mohammad Saber Golanbari, Dr. Anteneh Gebregiorgis, Dr. Rajendra Bishnoi and Samir Ben Dodo who were with me almost throughout my journey with their continuous support in personal and professional life.

I would also like to thank Ms. Iris Schroeder-Piepka for her friendly support and also for handling all the required documents during the entire PhD. Along with that, I thank Ms. Audrey Bohlinger who helped a lot with the paperwork in the final months of my PhD.

Nothing would have been possible without the love and support of three special people in my life; my mother Geetha, my father Vijayan and my wife Amitha. I am indebted to them for the sacrifices they have made to see me achieve this goal.

Arunkumar Vijayan
Kleinbachstr. 19a
76227 Karlsruhe

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

Karlsruhe, April 2019
Arunkumar Vijayan

Abstract

With technology scaling advancement and globalization of integrated circuit (IC) manufacturing, a host of vulnerabilities affect dependability of computing hardware. Each integrated circuit chip is born with a unique personality due to process variations and grows uniquely due to operating conditions, workload and environment. Hence, design-time solutions for dependability, based on deterministic models, are no longer sufficient for ICs fabricated at nanoscale technology nodes. There is a need for runtime analysis of the state of a system and adoption of appropriate mitigation actions to ensure dependability.

Transistors are prone to workload-dependent aging phenomena that increase delay in circuit paths leading to false computations. In addition, specific workloads can induce accelerated aging leading to reduction in reliable lifetime of a chip. Apart from aging, transient computational errors (soft errors) can be caused by chip exposure to radiations that can result in abnormal behavior in critical systems. The propagation or masking of such errors is dependent on the workload executed on the system. Fabricated chips can also include malicious circuits called hardware Trojans, deliberately inserted during chip design or manufacture, that can compromise security. Due to the stealthy nature of inserted malicious circuits before their activation, it is extremely difficult to verify the chip as Trojan-free.

The complexity of these dependability issues makes simple dependability modeling and mitigation inefficient. This complexity arises from various sources including design (technology, device, circuit and architecture) parameters, fabrication parameters, runtime workload and environment. This is the motivation to explore machine learning and runtime methods that can potentially deal with such complexities.

In this thesis, we propose solutions to ensure dependable operation of computing hardware under different workload and environmental conditions. We devise machine learning techniques to model, monitor and mitigate various dependability effects. Different learning methods are used to identify low cost workload observables and to build prediction models that correlate the workload observables with dependability metrics corresponding to reliability and security attributes. We also developed low cost hardware monitoring circuits that can capture the workload observables during runtime with lower area and power overheads. In contrast to the state-of-the-art techniques exploiting micro-architectural observables for monitoring, we explore the potential of workload characterization at logic level of hardware abstraction. We identify better logic level features to enable fine-grained runtime monitoring. This logic-level analysis also comes with several knobs to tune for higher prediction accuracy and lower overheads.

We experimented this philosophy of identifying logic-level observables based on learning methods and implementing low cost monitors to enable adaptive mitigation of static aging, dynamic aging, radiation-induced soft errors and also to identify the activation of hardware Trojans. In this regard, we developed a prediction model to track the workload impact on aging degradation of a chip that can be used on-the-fly to decide upon mitigation techniques such as task migration, dynamic voltage and frequency scaling. This prediction model is implemented in software that potentially ranks workloads based on their aging stress severity. To ensure resilience against accelerated aging effect, we propose a monitoring hardware that monitors a subset of critical flip-flops for an accelerated aging phase of the workload and raise a flag when a timing-critical path experiences severe aging stress. We implemented a technique to

relax the stress by the execution of a specific subroutine that exercises the critically stressed timing paths. We propose a technique to estimate the online soft-error vulnerability of memory arrays and logic cores based on the monitoring of a small set of flip-flops in the design. We also developed a method based on anomaly detection to identify workload signatures of hardware Trojan payload during runtime activation of a Trojan as a last line of defense. Based on these experiments, this thesis demonstrates the potential of advanced feature extraction at logic-level of abstraction and learning-based prediction based on runtime data to achieve better dependability for hardware designs.

Zusammenfassung

Mit dem Voranschreiten der Technologieskalierung und der Globalisierung der Produktion von integrierten Schaltkreisen eröffnen sich eine Fülle von Schwachstellen bezüglich der Verlässlichkeit von Computerhardware. Jeder Mikrochip wird aufgrund von Produktionsschwankungen mit einem einzigartigen Charakter geboren, welcher sich durch seine Arbeitsbedingungen, Belastung und Umgebung in individueller Weise entwickelt. Daher sind deterministische Modelle, welche zur Entwurfszeit die Verlässlichkeit prognostizieren, nicht mehr ausreichend um Integrierte Schaltkreise mit Nanometertechnologie sinnvoll abbilden zu können. Der Bedarf einer Laufzeitanalyse des Zustandes steigt und mit ihm die notwendigen Maßnahmen zum Erhalt der Zuverlässigkeit. Transistoren sind anfällig für auslastungsbedingte Alterung, die die Laufzeit der Schaltung erhöht und mit ihr die Möglichkeit einer Fehlberechnung. Hinzu kommen spezielle Abläufe die das schnelle Altern des Chips befördern und somit seine zuverlässige Lebenszeit reduzieren. Zusätzlich können strahlungsbedingte Laufzeitfehler (Soft-Errors) des Chips abnormales Verhalten kritischer Systeme verursachen. Sowohl das Ausbreiten als auch das Maskieren dieser Fehler wiederum sind abhängig von der Arbeitslast des Systems. Fabrizierten Chips können ebenfalls vorsätzlich während der Produktion boshafte Schaltungen, sogenannte Hardwaretrojaner, hinzugefügt werden. Dies kompromittiert die Sicherheit des Chips. Da diese Art der Manipulation vor ihrer Aktivierung kaum zu erfassen ist, ist der Nachweis von Trojanern auf einem Chip direkt nach der Produktion extrem schwierig.

Die Komplexität dieser Verlässlichkeitsprobleme machen ein einfaches Modellieren der Zuverlässigkeit und Gegenmaßnahmen ineffizient. Sie entsteht aufgrund verschiedener Quellen, eingeschlossen der Entwicklungsparameter (Technologie, Gerät, Schaltung und Architektur), der Herstellungsparameter, der Laufzeitauslastung und der Arbeitsumgebung. Dies motiviert das Erforschen von maschinellem Lernen und Laufzeitmethoden, welche potentiell mit dieser Komplexität arbeiten können.

In dieser Arbeit stellen wir Lösungen vor, die in der Lage sind, eine verlässliche Ausführung von Computerhardware mit unterschiedlichem Laufzeitverhalten und Arbeitsbedingungen zu gewährleisten. Wir entwickelten Techniken des maschinellen Lernens um verschiedene Zuverlässigkeitseffekte zu modellieren, zu überwachen und auszugleichen. Verschiedene Lernmethoden werden genutzt, um günstige Überwachungspunkte zur Kontrolle der Arbeitsbelastung zu finden. Diese werden zusammen mit Zuverlässigkeitsmetriken, aufbauend auf Ausfallsicherheit und generellen Sicherheitsattributen, zum Erstellen von Vorhersagemodellen genutzt. Des Weiteren präsentieren wir eine kosten-optimierte Hardwaremonitorschaltung, welche die Überwachungspunkte zur Laufzeit auswertet. Im Gegensatz zum aktuellen Stand der Technik, welcher mikroarchitektonische Überwachungspunkte ausnutzt, evaluieren wir das Potential von Arbeitsbelastungscharakteristiken auf der Logikebene der zugrundeliegenden Hardware. Wir identifizieren verbesserte Features auf Logikebene um feingranulare Laufzeitüberwachung zu ermöglichen. Diese Logikanalyse wiederum hat verschiedene Stellschrauben um auf höhere Genauigkeit und niedrigeren Overhead zu optimieren.

Wir untersuchten die Philosophie, Überwachungspunkte auf Logikebene mit Hilfe von Lernmethoden zu identifizieren und günstigen Monitore zu implementieren um eine adaptive Vorbeugung gegen statisches Altern, dynamisches Altern und strahlungsinduzierte Soft-Errors zu schaffen und zusätzlich die Aktivierung von Hardwaretrojanern zu erkennen.

Diesbezüglich haben wir ein Vorhersagemodell entworfen, welches den Arbeitslasteinfluss auf alterungsbedingte Verschlechterungen des Chips mitverfolgt und dazu genutzt werden kann, dynamisch zur Laufzeit vorbeugende Techniken, wie Task-Mitigation, Spannungs- und Frequenzskalierung zu benutzen.

Dieses Vorhersagemodell wurde in Software implementiert, welche verschiedene Arbeitslasten aufgrund ihrer Alterungswirkung einordnet. Um die Widerstandsfähigkeit gegenüber beschleunigter Alterung sicherzustellen, stellen wir eine Überwachungshardware vor, welche einen Teil der kritischen Flip-Flops beaufsichtigt, nach beschleunigter Alterung Ausschau hält und davor warnt, wenn ein zeitkritischer Pfad unter starker Alterungsbelastung steht. Wir geben die Implementierung einer Technik zum Reduzieren der durch das Ausführen spezifischer Subroutinen auftretenden Belastung von zeitkritischen Pfaden. Zusätzlich schlagen wir eine Technik zur Abschätzung von online Soft-Error-Schwachstellen von Speicherarrays und Logikkernen vor, welche auf der Überwachung einer kleinen Gruppe Flip-Flops des Entwurfs basiert.

Des Weiteren haben wir eine Methode basierend auf Anomalieerkennung entwickelt, um Arbeitslastsignaturen von Hardwaretrojanern während deren Aktivierung zur Laufzeit zu erkennen und somit eine letzte Verteidigungslinie zu bilden. Basierend auf diesen Experimenten demonstriert diese Arbeit das Potential von fortgeschrittener Feature-Extraktion auf Logikebene und lernbasierter Vorhersage basierend auf Laufzeitdaten zur Verbesserung der Zuverlässigkeit von Harwareentwürfen.

Contents

Glossary

3PIP	Thid-party Intellectual Property
ACE	Architecturally Correct Execution
ATPG	Automatic Test Pattern Generation
BTI	Bias Temperature Instability
CMOS	Complementary Metal Oxide Semiconductor
DVFS	Dynamic Voltage and Frequency Scaling
FF	flip flop
FIT	Failure in Time
FPGA	Field Programmable Gate Array
GBM	Gradient Boosting Machine
HCI	Hot Carrier Injection
ISA	Instruction Set Architecture
LUT	Look-Up-Tables
LVF	Logic Vulnerability Factor
MTTF	Mean Time To Failure
MVF	Memory Vulnerability Factor
overfitting	situation in which predictor model fits to noise in training data.
RTL	Register Transfer Level
SER	Soft Error Rate
SoC	System-on-Chip
SP	Signal Probability
STA	Static Timing Analysis
SVM	Support Vector Machines
TDDB	Time Dependent Dielectric Breakdown

VCD	Value Change Dump
VF	Vulnerability Factor
VLSI	Very-Large-Scale Integration
workload	Set of input patterns and states applied to a hardware design.
workload observable	Representative of the signal activity of a workload.
workload phase	A workload segment with uniform characteristics.

List of Figures

List of Tables

List of own publications included in this thesis

Transactions

[24] Vijayan, Arunkumar and Koneru, Abhishek and Kiamehr, Saman and Chakrabarty, Krishnendu and Tahoori, Mehdi B, "Fine-grained aging-induced delay prediction based on the monitoring of run-time stress," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[27] Vijayan, Arunkumar and Kiamehr, Saman and Ebrahimi, Mojtaba and Chakrabarty, Krishnendu and Tahoori, Mehdi B, "Online Soft-Error Vulnerability Estimation for Memory Arrays and Logic Cores," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[26] Vijayan, Arunkumar and Kiamehr, Saman and Oboril, Fabian and Chakrabarty, Krishnendu and Tahoori, Mehdi B, "Workload-Aware Static Aging Monitoring and Mitigation of Timing-Critical Flip-Flops," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[29] Vijayan, Arunkumar and Tahoori, Mehdi B, and Chakrabarty, Krishnendu "Runtime Identification of Hardware Trojans by Feature Analysis on Gate-level Unstructured Data and Anomaly Detection ," **under Review** in *IEEE Transactions on Information, Forensics and Security*, 2019.

Conferences

[28] Vijayan, Arunkumar and Koneru, Abhishek and Ebrahimi, Mojtaba and Chakrabarty, Krishnendu and Tahoori, Mehdi B, "Online soft-error vulnerability estimation for memory arrays," in *VLSI Test Symposium (VTS)*, 2016.

[25] Vijayan, Arunkumar and Kiamehr, Saman and Oboril, Fabian and Chakrabarty, Krishnendu and Tahoori, Mehdi B, "Workload-aware static aging monitoring of timing-critical flip-flops.," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

List of co-author publications not included in this thesis

- Koneru, Abhishek, Arunkumar Vijayan, Krishnendu Chakrabarty, and Mehdi B. Tahoori. "Fine-grained aging prediction based on the monitoring of run-time stress using DfT infrastructure." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 51-58.*, 2015.
- Tahoori, Mehdi B., Abhijit Chatterjee, Krishnendu Chakrabarty, Abhishek Koneru, Arunkumar Vijayan, and Debashis Banerjee. "Self-awareness and self-learning for resiliency in real-time systems." In *IEEE 21st International On-Line Testing Symposium (IOLTS), pp. 128-131.*, 2015.
- Firouzi, Farshad, Fangming Ye, Arunkumar Vijayan, Abhishek Koneru, Krishnendu Chakrabarty, and Mehdi B. Tahoori. "Re-using BIST for circuit aging monitoring." In *20th IEEE European Test Symposium (ETS), pp. 1-2.*, 2015.

1 Introduction

The past fifty years of semiconductor industry was an era of aggressive technology scaling reaching to sub-10 nm device sizes. The tremendous improvement in complexity (in terms of number of transistors) and performance (in terms of operating frequency) achieved for various microprocessor generations is illustrated in Fig. 1.1 [1]. According to Moore's law, the number of transistors in a semiconductor chip doubles every 18 months due to technology scaling [2]. This enables chips with increasing complex functionality, higher performance and lower per-function cost for every new generation. For example, the central processor chip of IBM z14 processor contains 6.1 billion transistors [3] in contrast to less than a billion transistors in IBM z10 released a few years before z14 [4]. The average transistor cost reduced by 10× due to technology scaling from 130 nm node to 14 nm node [5]. Even the latest smartphones like iPhone XS can do 5 trillion operations per second in comparison to Cray 2, the fastest supercomputer of early 1990s, capable of only 1.9 billion operations per second [6]. As a result of the increase in performance and cost reduction in implementing complex functionalities, semiconductor chips have revolutionized automotive, space, healthcare, multimedia and communication domains [7].

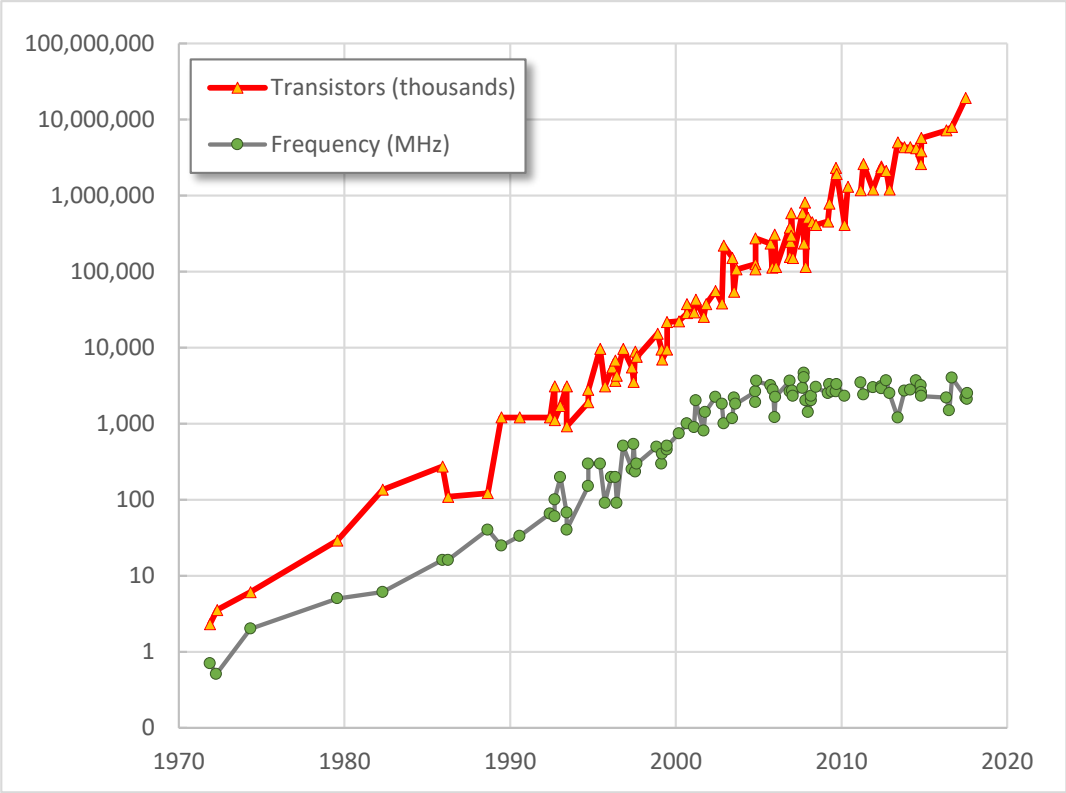


Figure 1.1: Illustration of complexity and performance increase with technology scaling advancement in microprocessors for the last four decades [1].

In the modern society, the ubiquity of semiconductor chips demands dependable operation for an expected lifetime. Dependability is defined as the ability to deliver trusted service [8].

With technology scaling advancement, many dependability challenges aggravate and hence, failure rate increases [9]. Smaller device sizes bring increasing variations between transistors during fabrication and also vulnerability to different environmental and runtime variations. The failure-free operation of a chip can be significantly affected by different reliability mechanisms such as device aging or radiation-induced soft errors [10]. Device aging refers to a set of degradation mechanisms that affect transistor properties, eventually leading to unexpected failures in computation. Soft errors are caused by alpha particles from packaging materials or neutrons from cosmic rays that may strike a semiconductor chip, and temporarily induce a wrong value in a memory cell or a circuit node leading to system failures. In addition to these sources of unreliabilities, several security threats have emerged due to the increasing globalization of semiconductor design and fabrication stages [11]. Each step in the design of a complex system-on-chip (SoC) is vulnerable to security issues such as inclusion of stealthy malicious logic called hardware Trojans, by third-party vendors [12]. Because of the ever increasing complexity, the verification and test efforts fall short to ensure secure operation of chips. Hence, techniques to ensure dependability of hardware designs with reliability and security as the two most relevant attributes are attaining primary focus in today's computing world.

The design-time solutions against reliability and security challenges in the context of dependable hardware design can be less efficient in the new era where chips are exposed to complex runtime variations. The rate of degradation of integrated circuits due to aging of transistors can significantly vary with the type of workload executed on them, the operating temperature, and the variations in supply voltage [13]. The vulnerability of circuits to radiation-induced soft errors also varies with the workload nature, and other runtime parameters [14]. The role of design decisions to ensure security in semiconductor chips is limited due to the threat of additional hardware inclusion at a later stage in the fabrication process [15]. Hence, in addition to design-time optimization of circuits for better reliable and secure operation, runtime monitoring is required to guide appropriate adaptation actions.

Cost-effective runtime monitoring can assist in tuning several knobs available to alleviate the impact of dependability issues. In modern-day systems, a large number of sensors and monitors are incorporated to track several runtime parameters such as power consumption and supply voltage [16]. With rising dependability requirements in various domains [17], there is a need for dependability monitors and models to make runtime decisions that can ensure dependable operation of a chip.

1.1 Problem statement and objective

Semiconductor chips, after fabrication, are utilized in different ways and are exposed to different set of runtime parameters. Hence, solutions for dependability problems cannot be static and needs to be adaptable against variations during lifetime operation of each chip. It is essential to use runtime data to make predictions on the status of various degradation and vulnerability parameters. Accurate prediction models are required in this regard which can guide adaptation actions in a proactive manner.

One of the transistor aging mechanisms known as Bias Temperature Instability (BTI) degrades the threshold voltage of a transistor that eventually results in slower switching of logic gates made of these transistors [39]. Aging of transistors leads to degradation of circuit delays, eventually increasing the number of timing violations, and a decrease in the expected lifetime [18]. If the degradation rate can be tracked in a fine-grained manner, appropriate adaptation actions such as adaptive voltage scaling [19], body biasing [20] or reliability-aware task mapping [21] can be taken. In the case of radiation-induced soft errors, each phase of a workload executed on a design, such as a processor, can have a specific nature causing higher

or lower vulnerability towards soft error events measured in terms of vulnerability factor. Prediction of accurate vulnerability factor during runtime can enable cost-effective protection mechanisms. In the case of security threats, malicious hardware (hardware Trojans) can get activated during runtime, and the prediction of the Trojan activation serves as a last line of defense. With increasing complexity of systems, the runtime monitoring and prediction of these parameters meet several challenges. The generation of enormous amount of runtime data during workload execution on a hardware block necessitates careful spatial and temporal selection of probing points in the hardware block. Reuse of existing monitors cannot be always an effective solution due to the granularity of access and hence, new cost-efficient runtime monitors with less overheads are required. Predictions from the patterns in runtime data needs to be fast and accurate to enable proactive mitigation of dependability issues.

Modern systems incorporate a range of sensors and monitors (e.g., razor flip-flops [22], critical path monitors (CPM) [23]) to track the impact of several reliability mechanisms on the functionality and performance of a circuit. These reliability mechanisms include aging due to Bias Temperature Instability (BTI) [10] and supply voltage fluctuations. In addition, additional hardware is added in the form of redundant units or error correction units in order to tackle the impact of radiation- induced soft errors [10]. With these sensor data available online, suitable adaptation policies can be triggered on-the-fly that can help in resilient operation of a system. However, the fundamental problem of the above method lies in the fact that these sensors monitor the effect (e.g., path delay increase due to BTI) of a reliability phenomenon rather than its cause (nature of workload). Hence, the adaptation policies can only be triggered after a measurable degradation occurs on the circuit. In addition, to secure against activation of hardware Trojans, performance-monitor based detection techniques are deployed. However, these monitors can be tampered by an adversary. In short, dependability-aware selection of appropriate features to monitor runtime data and development of prediction models that can predict accurately from the selected features are the needs of the hour.

The objective of this thesis is to enable runtime predictions on different dependability mechanisms by logic-level data analysis, and by exploiting machine learning techniques for workload compaction and representation. Different learning methods are used to identify low cost workload observables, and to build prediction models that correlate the workload observables with reliability and security metrics.

1.2 Contribution of this Thesis

In this thesis, we target the improvement of overall dependability of integrated circuits by analyzing the impact of runtime variations on various reliability and security issues as illustrated in Fig. 1.2. We devise machine learning techniques to model, monitor and mitigate various dependability effects originating from imperfections in device fabrication, design issues, and impact of runtime workload and environment. We target early runtime prediction of the impact of a workload phase on resilient operation of a circuit. This information about the impact of a workload phase can guide proper mitigation actions proactively such as relaxation of aging stress or tackling vulnerability of a circuit to soft errors. In this regard, these learning techniques can be used to correlate workload patterns to corresponding impact on system dependability under aging, soft errors and malicious Trojans. We propose a methodology to monitor hardware designs online, and predict dependability metric values on-the-fly based on prediction models constructed offline. Our technique involves workload analysis to extract hidden information that describes the relationship between the workload executed on the design, and corresponding values of dependability metrics (Eg: circuit-path-delay increase).

The workload data (set of system states over time) is analyzed in the logic level to maximize

the information content to use for reliability and security analysis. However, this increases the complexity of analysis and hence, we deploy domain-specific feature selection and feature engineering techniques to capture important features of a workload segment. In this regard, suitable workload observables are identified offline using correlation analysis and feature elimination techniques. A prediction model is built offline to correlate the workload observable with the dependability metric under consideration. Low cost hardware monitors are proposed to track the workload observables online and the monitoring information is fed to software predictors to make early predictions on the reliability and security metrics.

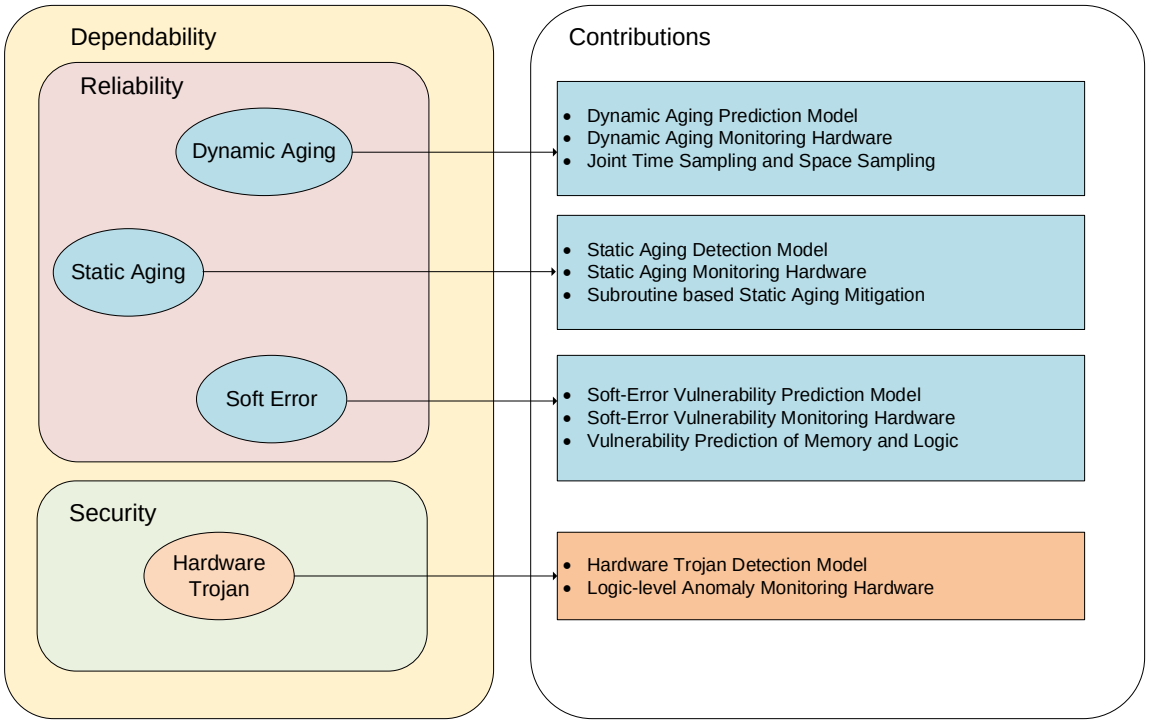


Figure 1.2: Overall contributions of this thesis in different attributes of dependability.

In particular, the novel contribution of this thesis are listed below.

- **Dynamic Aging Monitoring and Delay Prediction** [24]: Dynamic aging of transistors due to Bias Temperature Instability (BTI) involves aging stress and recovery phases resulting in long term degradation of circuit paths. This mechanism degrades path delay over time leading to timing failures. Direct monitoring of path delays based on actual measurements can only track a coarse-grained aging trend. We propose a fine-grained aging monitoring scheme based on estimating runtime aging stress of workloads. A prediction model generated offline based on workload stress analysis is implemented online with a low overhead monitoring hardware to make predictions on aging rates. As a result, we can achieve timely workload-stress estimations and aging rates during runtime that can be used in turn to take appropriate proactive and fine-grain mitigating actions and prevent the circuit from aging at higher rates.
- **Static Aging Monitoring and Mitigation** [25, 26]: Circuits are exposed to accelerated aging rates (static BTI) while the transistors are stressed for extended period of time without any recovery phase. This scenario can occur based on the nature of workload phase and it can lead to early timing failures as one year of dynamic BTI stress is similar to a few hours of static BTI stress. To address this problem, we identify correlated static aging phases of timing critical flipflops in an offline correlation

analysis and deploy a runtime monitoring scheme to raise a flag when the circuit enters an accelerated aging phase. A synthetic subroutine based mitigation technique is also proposed to relax the aging stress by exercising aged gates and flip-flops. This leads to an improvement in the overall lifetime of the circuit compared to the case where only dynamic BTI is considered.

- **Soft-Error Vulnerability Prediction** [27, 28]: The vulnerability of a system to soft errors is dynamic in nature due to various masking effects in different levels of abstraction. Hence, it is desirable to have a vulnerability prediction scheme that can turn on and off online protection mechanisms in a cost-effective manner. In this regard, we proposed a low overhead soft-error vulnerability prediction scheme by monitoring only a small number of flip-flops during runtime. Our prediction model can predict the online vulnerability of a whole system comprising of memory arrays and logic cores significantly more accurate than the state-of-the-art prediction techniques based on performance counters.
- **Runtime Hardware Trojan Detection** [29]: During the age of globalization of chip design and fabrication, untrusted designs in the form of third-party intellectual property (3PIP) poses a threat of malicious hardware inclusion in the form of hardware Trojans. The stealthy nature of these Trojans makes them hard to be identified before in-field operation of the chip. Hence, runtime techniques serve as a last line of defense. The state-of-the-art runtime detection techniques monitors functionally defined (semantically significant) signals to capture Trojan activation and these signals are vulnerable to be masked by a rogue designer. We propose an anomaly detection technique based on logic-level signals (semantically insignificant) that are hard to be masked by an adversary at a 3PIP level. We could achieve high classification accuracy on the identification of Trojan activation by monitoring the workload profile of a small number of flipflops in the processor core.

All aforementioned techniques involve workload characterization of open-source processor designs at logic level of abstraction. To accelerate the workload characterization, a Field-Programmable Gate Array (FPGA) implementation of open source processor core was used with cross-compiled workloads executed on the FPGA platform. We have used Leon3 [30] and OpenRISC 1200 [31] processor designs along with SPEC [32] and MiBench [33] workloads. For prediction model generation, python-based scikit-learn library [34] was used. In short, the workload execution environment was developed on a real FPGA platform in addition to a post-synthesis simulation environment that can carry out cycle-accurate workload simulations.

1.3 Outline

This chapter describes the motivation and contribution of this thesis. The rest of this thesis is organized as follows:

Chapter 2 provides a brief description of each reliability and security challenge considered in this thesis. The discussion spans from the basic physical effects governing the application-level impact of the dependability issues to the state-of-the-art runtime techniques available in the context of this thesis.

In Chapter 3, our methodology to cost-efficiently address the need for dynamic aging prediction is presented. Our fine-grained aging monitoring scheme to analyze and compare the aging stress of workloads during runtime is discussed. Offline prediction model construction based on machine learning is explained with examples and the overhead of online monitoring hardware is calculated. Further, space and time sampling techniques are described to reduce power and area overheads of monitoring.

Chapter 4 presents the worst-case scenario of circuit delay degradation due to static aging and analyzes the impact on different processor designs. A low overhead runtime monitoring scheme is detailed along with a mitigation scheme based on software subroutines. The lifetime improvement due to the proposed monitoring and mitigation scheme is also reported.

In Chapter 5, a low overhead runtime soft-error vulnerability prediction schemes for memory arrays and logic cores of a system is presented. The offline prediction model construction based on two machine learning algorithms is discussed in detail. The overall design of the online monitoring scheme to observe a small set of flipflops is explained with corresponding overheads.

Chapter 6 discusses a scheme for the runtime prediction of hardware Trojan activation. An anomaly detection technique based on monitoring the workload profile of selected circuit nodes is presented. The prediction accuracy achieved on a Trojan-inserted open source processor core executing several workloads is reported.

Finally, Chapter 7 concludes the thesis and a discussion on future implication of this work is given.

2 Background

In this chapter, the basic challenges to dependable hardware design in the age of aggressive technology scaling and globalized chip fabrication are discussed. In the context of this thesis, a brief introduction to dependability, various reliability and security challenges, and their run-time impact are discussed. Furthermore, a brief introduction to machine learning techniques is also provided.

2.1 Dependability Challenges: Overview of Hardware Reliability and Security

A computing system is said to be dependable if it can be trusted with its intended functionality. Dependability is the system property that integrates reliability, availability, safety, security and maintainability [8]. The emphasis on these different attributes can be different based on the underlying application. In the context of this thesis, security and reliability attributes are given a higher emphasis, and the term *dependability* is used as a hypernym as shown in Fig. 2.1.

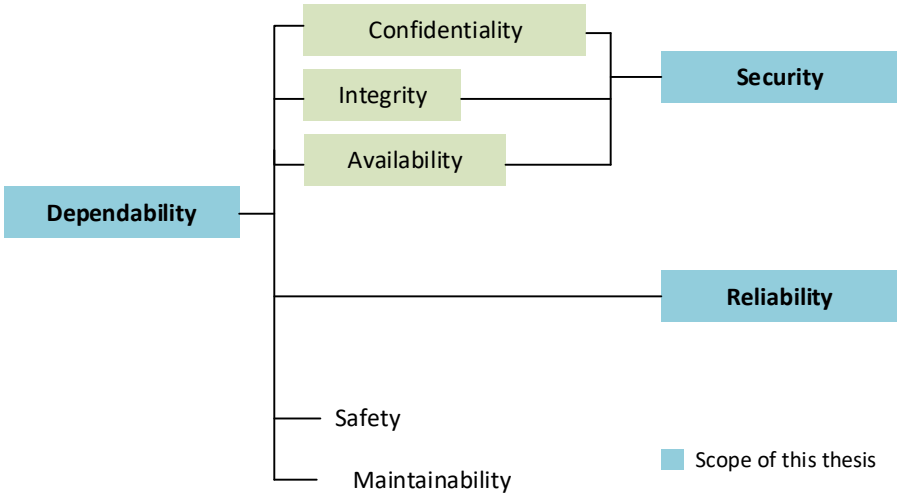


Figure 2.1: Illustration of the attributes of dependability in the scope of this thesis [8].

Figure 2.2: Dependability threats and their causal relationship [8].

According to [8], security is defined as the concurrent existence of confidentiality, integrity and availability. Confidentiality is satisfied only if there is no unauthorized disclosure of information. Integrity demands no improper alterations to system state. Availability can be described as the readiness of correct service. Reliability is termed as a measure of the continuous delivery of correct service.

In addition to the different attributes of dependability as discussed previously, dependability threats are manifested in the form of faults, errors, and failures as illustrated in Fig. 2.2. Fault can be a physical defect on a device. An error in the functionality of a system can be caused by a fault. If an external service is affected by the propagation of this error, a failure is said to be occurred. In short, a fault produces an error, that in turn causes a failure. It can also happen that a fault remains dormant or an error may not propagate based on other characteristics of a system.

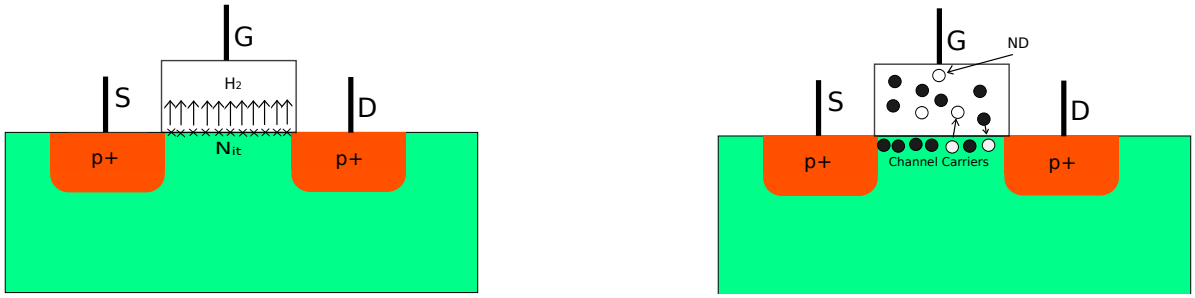
With a general overview of the different attributes and threats of dependability, we examine the specific reliability and security aspects tackled in this thesis. With technology scaling advancement, a host of vulnerabilities affect reliability of VLSI designs. In a modern SoC design, billions of transistors are packed in a single chip and this increases the complexity of verification and test efforts. In addition, several transistor degradation mechanisms such as aging cause slowing down of computations over time eventually leading to failures. Transient soft errors, caused by radiations, are another reliability issue that can cause bit flips in circuit nodes that can eventually propagate to cause system failures. The runtime dependence of these reliability issues adds an additional layer of complexity that makes design-time solutions inadequate. On the other hand, the globalization of chip design and fabrication stages introduces serious security concerns. The possibility of malicious hardware inclusion in a chip called hardware Trojan at any stage of design flow emerges as a major security threat for government bodies and other customers of the chip. In the following sections, we present an in-depth discussion on specific reliability and security challenges.

2.2 Reliability Challenges

Reliability $R(t)$ is defined as the probability that a computing system performs correct functionality for a given period of time t under specified conditions without any failure [35]. In the realm of hardware design, sources of unreliabilities include aging of transistors, radiation-induced soft errors, process variation, susceptibility to noise, and increasing complexities in different hardware abstraction levels. The severity of these reliability issues on a particular chip is largely dependent on the usage scenarios. To quantify the reliability of a system, different metrics such as *Mean Time To Failure (MTTF)* are used. The $MTTF$ of a system is the time for which the system is expected to operate without any failure.

In the context of this thesis, two reliability issues, (a) circuit aging phenomenon, and (b) radiation-induced soft error are discussed in detail.

2.2.1 Circuit Aging



(a) Reaction-Diffusion Mechanism

(b) Trapping-Detrapping Mechanism

Figure 2.3: NBTI Aging Models [36]

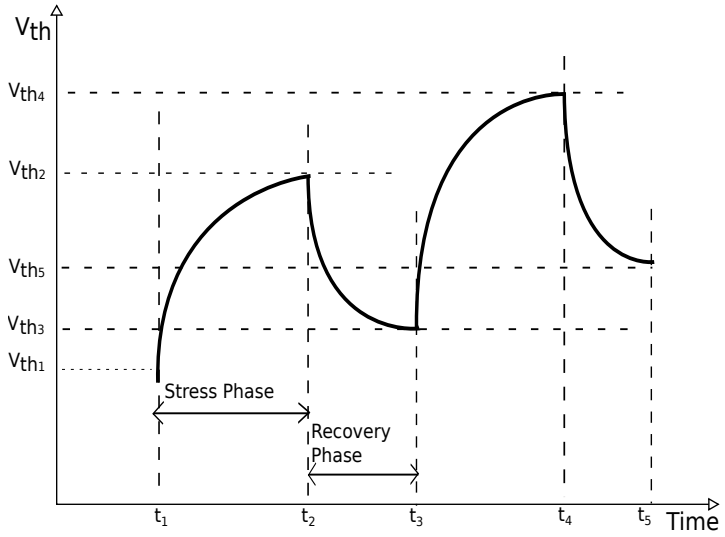


Figure 2.4: Illustration of stress and recovery phases due to NBTI

Circuit aging refers to the various degradation or wear-out mechanisms that make circuits less reliable over time leading to the occurrence of failures before the end of expected device lifetime. The degradation mechanisms include Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), Time Dependent Dielectric Breakdown (TDDB) and electromigration (EM). BTI, HCI and TDDB degrade the transistor characteristics. BTI causes an increase in the threshold voltage of a transistor that is turned on. Both PMOS (by Negative Bias Temperature Instability (NBTI)) and NMOS (by Positive Bias Temperature Instability (PBTI)) transistors are prone to this effect in advanced technology nodes. This effect can necessitate additional timing margin compromising performance of circuits to prevent early timing failures. HCI is caused by the high lateral electric field due to *drain-to-source* voltage (V_{ds}) of a transistor leading to the penetration of gate oxide by high energy carriers. This leads to a degradation in the threshold voltage of the transistor which eventually can cause timing failures. TDDB causes degradation of gate oxide characteristics leading to a leakage path through the oxide. In the worst case, it can cause a hard breakdown or permanent damage of the transistor. Electromigration causes wear out of interconnects due to high current density causing timing delays, open circuit or short circuit. In the context of this thesis, we describe BTI models and the effect of BTI on circuit delay in detail.

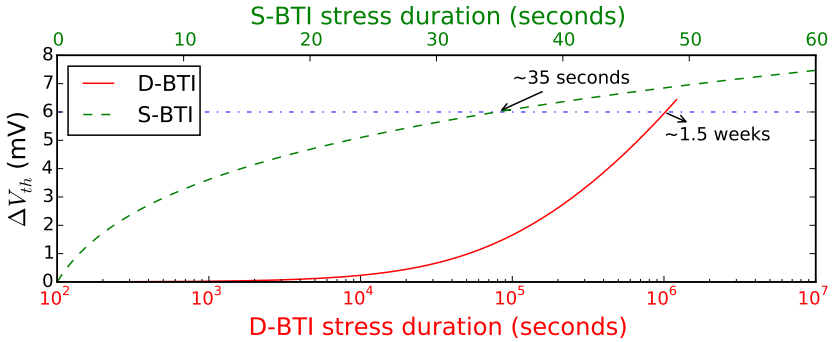


Figure 2.5: Comparison of ΔV_{th} due to D-BTI (duty cycle (α)=0.5) and S-BTI.

Bias Temperature Instability (BTI) is one of the major reliability threats in nanoscale tech-

nology nodes. Although both PMOS and NMOS transistors are affected by Negative Bias Temperature Instability (NBTI) and Positive Bias Temperature Instability (PBTI) respectively, we limit our analysis to NBTI. NBTI can cause an increase in the threshold voltage (V_{th}) of transistors along with a decrease in transconductance (g_m) and saturation current (I_{Dsat}) at negative gate voltages and elevated temperatures. The modeling of this phenomenon is complicated due to a recovery phase that involves a partial recovery of transistor degradation when a positive gate voltage is applied. The long-term effect of NBTI on a design is heavily dependent on the type and history of stress due to the workload under execution.

NBTI can be explained using two different mechanisms, (1) Reaction-Diffusion (RD) model [37, 38], or (2) Trapping/Detrapping (TD) Model [39, 40]. According to RD model, increase in threshold voltage occurs due to the breaking of covalent (Si-H) bonds at interface resulting in the generation of interface traps as illustrated in Fig. 2.3a. The increase in V_{th} is assumed to follow a power law relation with the time under stress and an exponential relation with the applied stress voltage.

The TD model proposes a fundamentally different phenomenon to explain the threshold-voltage increase, i.e., the capture and emission of charge carriers by interface traps as illustrated in Fig. 2.3b. The capture time constant determines the probability of trapping and emission time constant determines the probability of detrapping. The threshold voltage increases gradually with the change in number of occupied traps and hence, follows a logarithmic relation in V_{th} shift. TD model assumes a logarithmic relation with stress time and an exponential relation with temperature and stress voltage.

In short, both aging mechanisms assume a stress phase and a recovery phase for a transistor under NBTI as shown in Fig. 2.4 based on the type of bias applied. During a normal workload execution, the bias of a transistor alternates causing alternate stress and recovery phases. In long term, the cumulative effect of NBTI heavily depends on the type of workload executed on the design.

The V_{th} increase under dynamic stress estimated using a long-term NBTI model for time duration t can be expressed as in equation 2.1 [41],

$$\Delta V_{thd} = \left(\frac{n^2 {K_v}^2 \alpha C t}{\zeta^2 t_{ox}^2 (1-\alpha)} \right)^n , \tag{2.1}$$

where t_{ox} is the oxide thickness, ζ is the diffusion coefficient, α is the stress duty cycle, n is the time exponent, typically 0.16, and K_v is a function of V_{gs}, V_{th} and temperature.

In short, since long-term NBTI effect (termed as dynamic BTI (D-BTI)) involves recovery phases after most of the stress phases (stress duty cycle < 1), the overall effect on V_{th} and thereby, on the circuit delay is minimal for a short duration of a few hours.

The worst-case increase in V_{th}, and therefore the circuit delay, occurs when there are continuous stress phases and no recovery phases (stress duty cycle $= 1$). This scenario, termed as static BTI (S-BTI), induces accelerated aging. The V_{th} shift under static stress (ΔV_{ths}) is given by equation 2.2 [41],

$$\Delta V_{ths} = \left({K_v}^2 t \right)^n . \tag{2.2}$$

The Fig. 2.5 compares the threshold-voltage increase due to S-BTI and D-BTI. It can be observed that the increase in V_{th} due to one week of D-BTI stress is equivalent to a few seconds of S-BTI stress. In other words, circuits under S-BTI experience accelerated aging stress and this scenario needs to be considered for worst-case timing analysis.

Aging-induced Timing Failures

The aging mechanism described in the previous section increases the threshold voltage of transistors in a CMOS circuit. In the circuit level, the current driving capability of transistors

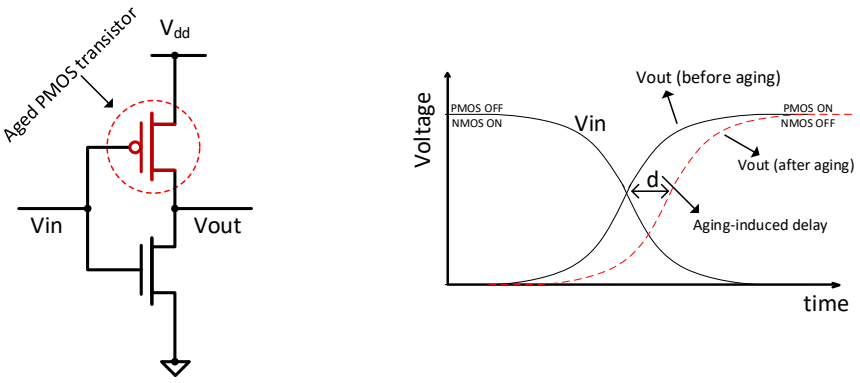


Figure 2.6: Illustration of an aged CMOS inverter (INV1) and the corresponding deviation in input-ouput characteristics in the form of an additional delay in the switching functionality.

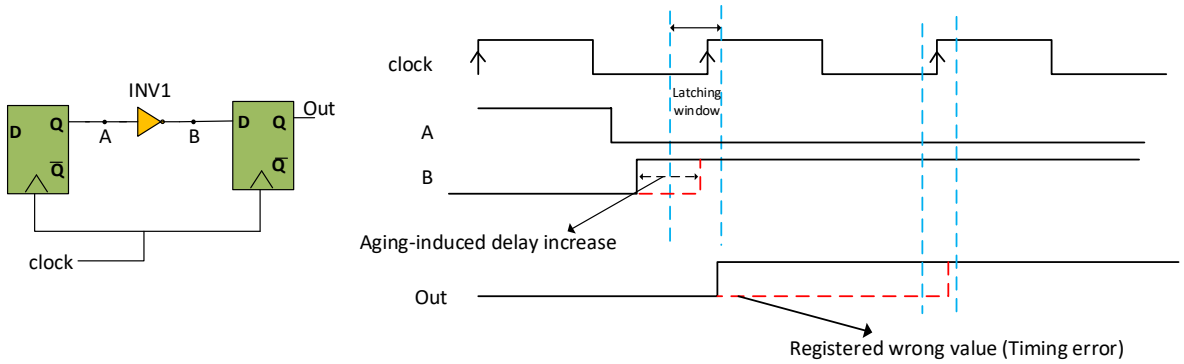


Figure 2.7: Illustration of a timing violation and failure caused by an aging-induced delay increase in a timing path.

are affected and it takes more time to charge the output capacitance of logic gates, eventually leading to an increase in propagation delay of logic gates in a circuit. This is demonstrated in Fig. 2.6 with the transfer characteristics of a CMOS inverter having an aged PMOS transistor. Due to aging, the output signal (V_{out}) of the inverter achieves switching with an additional delay (d). This aging-induced delay is a function of degraded threshold voltage of the transistor and this relationship can be approximated as shown in equation 2.3 [42],

$$d \approx \frac{C}{(V_{dd} - V_{th})^{\sigma}}, \tag{2.3}$$

where C and σ are technology parameters, V_{dd} is the supply voltage and V_{th} denotes the threshold voltage of the transistor.

In a standard-cell based digital design, a hardware block is designed to be executed at a specific clock frequency, which is a function of the worst-case delay of the circuit. A static timing analysis (STA) [43] of the circuit is carried out to determine the worst-case delay of a circuit and the path corresponding to the worst-case delay is termed as critical path. Due to variations in fabrication process, and also due to the runtime variations, the critical paths may change over time from pre-silicon to post-silicon and also during operational lifetime. Hence, a large number of critical paths and near-critical paths need to be considered for aging analysis during design-time. In this scenario, circuits are designed with an additional delay margin termed as guardband to compensate for the variations in circuit delay expected to be introduced by process variation and other runtime variations. This guardbanding technique ensures reliable operation of circuits by compromising the attainable performance.

As the aging mechanisms introduce additional delay in timing paths, some of the path delays could get degraded over time leading to a timing violation causing timing failures. This is illustrated in Fig. 2.7 when the aged CMOS inverter becomes part of a timing path. The switching event at signal B is registered in the flipflop at the end of the timing path only if the signal arrives before the latching window of the flipflop. When the delay increase in the inverter causes a late switching at B with respect to the clock signal, flipflop registers a wrong value causing a timing error or failure.

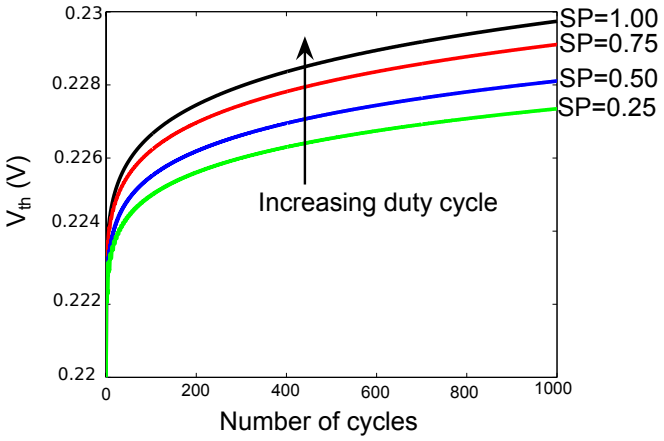


Figure 2.8: The dependence of V_{th} on signal probability (SP) [44]

These delay degradations of timing paths are heavily workload dependent. The aging-induced delay of a logic gate is a function of the fraction of time the input of the logic gate remains at logic 1. The probability that a circuit node having a value of 1 is termed as the signal probability (SP) of that circuit node. The dependence of threshold voltage degradation on SP is illustrated in Fig. 2.8 [44]. For a processor design, the workload under execution determines the SPs of internal circuit nodes and hence, the aging-induced circuit delay degradation.

Runtime Aging Monitors

Based on workload and runtime parameter variations such as temperature and supply voltage, aging-induced critical path delay of circuits varies. Hence, it is important to monitor the delay of a circuit over time to ensure reliable operation of the chip. Different aging monitoring mechanisms include in-situ sensors, silicon odometers, tunable replica circuits and representative path-based monitoring.

- In-situ sensors: This method tracks the circuit delay in the presence of aging based on modification of flip-flops which are located on the potential critical paths. Most of these flip-flops double-sample the data, first on the normal clock, and then on a delayed clock or by using a delay element. The sampled values are compared with a comparator to analyze whether the delay degradation encroached on the guard band. An example of an in-situ sensor called razor flipflop is shown in Fig. 2.9. Razor is employed for error detection and correction of delay path failures [22]. A shadow latch controlled by a delayed clock is employed along with a normal flipflop. A comparator compares the values in the latch and flipflop for any error. The original value is restored from the latch in case of a timing error.
- Silicon odometers: The beat frequency between a stressed Ring Oscillator Surrogate circuit (ROSC) and a reference ROSC is used to capture the effect of aging in silicon odometers [45]. The whole circuit is composed of two free-running ring oscillators

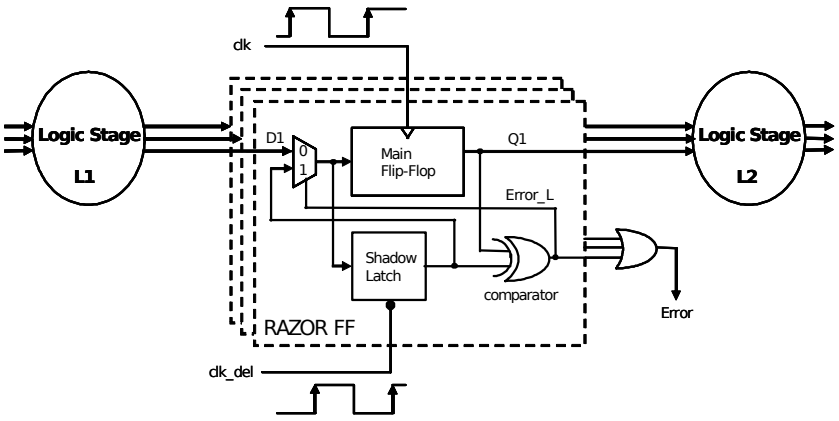


Figure 2.9: Razor flipflop deployed in a timing path to monitor the delay and restore the correct value in case of a timing error [22].

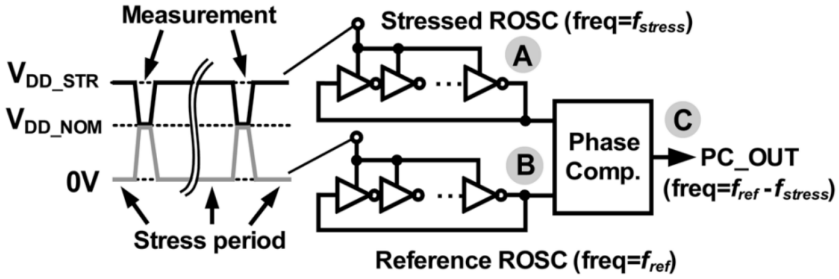


Figure 2.10: Illustration of BTI monitoring by measuring the beat frequency between two ring oscillators with one of them under stress and the other used as a reference [45].

and a phase comparator as shown in Fig. 2.10. During normal operation of the chip, one of the ROSCs is stressed with a voltage V_{DD_STR} and no stress is applied to the reference ROSC. During measurement periods, both ROSCs are brought to V_{DD_NOM}. The frequency degradation of stressed ROSC compared to the reference ROSC can be translated to aging degradation.

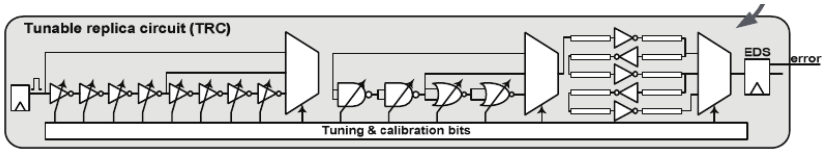


Figure 2.11: Illustration of tunable replica circuits that can be tuned to match the delay of critical paths for monitoring [46].

- Tunable replica circuits (TRC): A TRC has a digital delay sensor and it can be tuned to match the delay of a critical path at test time. As shown in Fig. 2.11, TRCs are composed of different logic stages such as inverters, NAND, NOR, pass gates and repeated interconnects to make the delay sensitivity similar to the critical path. TRCs are inserted adjacent to each pipeline stage in a design to track the local critical-path delays [46]. These are completely separated from the pipeline stages and hence, do not cause performance reduction.
- Representative critical path based monitors: Monitoring based on representative critical reliability paths (RCRPs) keeps a stand-alone circuit that is synthesized by identifying shared delay segments between different critical paths. Workload of these selected

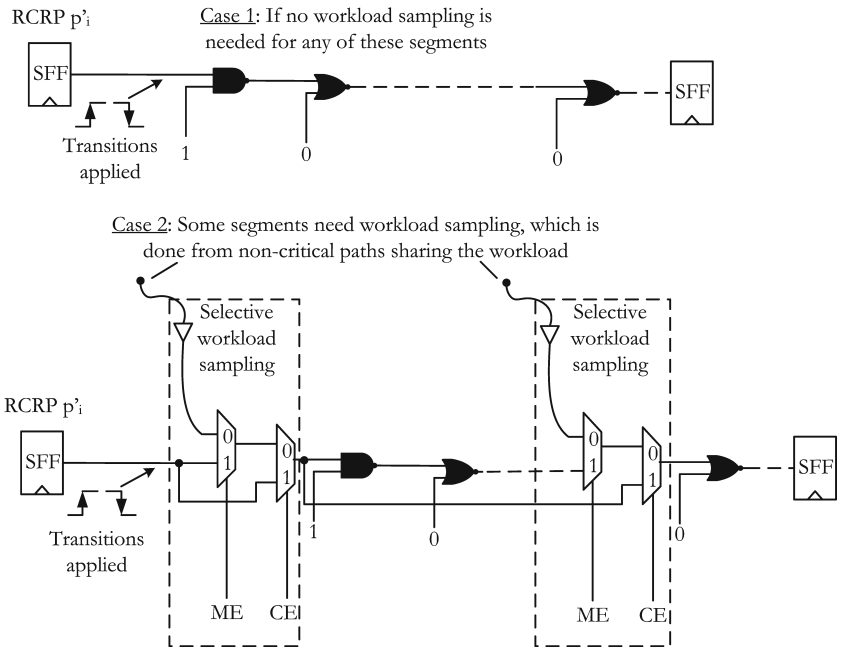


Figure 2.12: Representative critical reliability paths showing representation of critical paths with or without workload sampling [47].

delay segments can also be sampled to more accurately represent the critical reliability paths [47]. As illustrated in Fig. 2.12, without workload sampling, the delay segments can be connected together and the transitions can get propagated to induce aging in RCRP. In the case of workload sampling, multiplexers and buffers are inserted to sample workload impact such that the stand-alone RCRPs experience similar aging stress as functional circuit.

Runtime Aging Mitigation Techniques

Since aging phenomenon is heavily workload dependent, several runtime techniques are used to achieve aging mitigation or balanced aging of different hardware components. Advanced task mapping can be employed on multi-core systems in order to balance aging of different cores by mapping tasks based on their aging stress [48].

Along with task mapping, dynamic voltage and frequency scaling (DVFS) schemes can be tuned for joint optimization of reliability and performance [49]. Aging-induced degradation can also be recovered or frozen by allocation of idle intervals among cores. Hence, aging-aware adaptive runtime task allocation can be used to target a lifetime requirement by setting a proper idle/activity ratio. Dynamic cooling mechanisms can be used to reduce temperature such that aging gets slowed down [50].

Adaptive voltage scaling and adaptive body biasing techniques can also be used to control aging of circuits [51]. The current driving capabilities of two NBTI-stressed PMOS transistors with and without adaptive body biasing are compared in Fig. 2.13 [20]. This illustrates that the decrease in drive strength caused by NBTI can be compensated by tuning the body voltage of transistors. Supply voltage is another parameter that can be tuned to reduce the NBTI-induced aging rate. As shown in Fig. 2.14, the V_{th} degradation is higher for a higher supply voltage (V_{DD}).

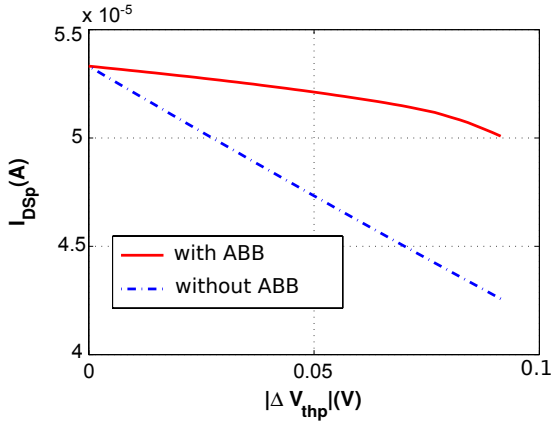


Figure 2.13: Illustration of the driving strength ((I_{DSp})) of two aging-stressed PMOS transistors, one with and the other without adaptive body biasing [20].

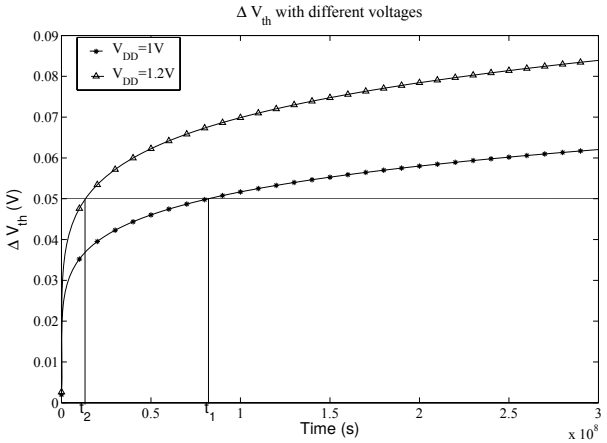


Figure 2.14: Illustration of the dependence of NBTI-induced threshold voltage degradation on supply voltage [19]

2.2.2 Soft Error

Soft error is defined as a transient error occurring in a computing system due to faults caused by particle strikes. This can be caused typically by alpha particles from the packaging material or neutrons from cosmic radiation. As shown in Fig. 2.15, a particle strike can generate large number of electron-hole pairs in the substrate of a transistor. Based on the amount of charge generated and a defined-parameter called critical charge for a circuit, an error might or might not occur in a circuit. The critical charge (Q_{crit}) is defined as the minimum charge necessary to cause a malfunction in a circuit [52]. If the charge induced by the particle strike reaches above Q_{crit}, a bit flip can occur in the circuit and this wrong value can propagate to the output of a system causing a malfunction or failure. Soft error rate is usually expressed in terms of Failure-in-Time (FIT), i.e., number of failures in one billion hours (10^9).

With technology scaling, transistors become smaller, hold less charge, and hence, more vulnerable to particle strike. On the other hand, shrinking sizes make it harder to get struck by particles. The soft error rate of a particular device depends on how these opposite effects interplay.

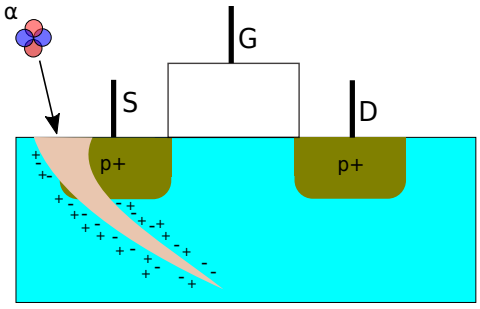


Figure 2.15: Effect of alpha-particle strike on a transistor eventually leading to soft error.

Soft-Error-Vulnerability Estimation

A particle strike can cause a malfunction in a transistor's operation that can result in a change in the output of a logic gate or a bit flip in a latch or memory cell. During the execution of a program, these bit flips can get masked and may not result in a user-visible error. To calculate the probability that an internal fault results in an externally visible error, the intrinsic FIT rate of a transistor needs to be derated by a number of vulnerability factors.

Architectural Vulnerability Factor (AVF) is defined as the probability that a bit-flip will propagate to a user-visible output [53]. A higher value of AVF of a bit indicates its higher vulnerability to cause a soft error. The circuit structures that need protection can be identified by their AVF values. A bit flip in a branch predictor does not cause a user-visible error in a program execution unlike a bit flip in a program counter. Hence, AVF of a branch predictor is 0% and AVF of a program counter is 100%. For other blocks such as instruction queue, AVF can vary between 0% and 100% based on the workload under execution. The overall soft-error rate (SER) of a chip can be defined as equation 2.4.

$$\text{Overall SER} = \text{circuit-level SER} \times \text{AVF} \tag{2.4}$$

A soft error can cause either a silent data corruption (SDC) in a circuit or a detectable unrecoverable error (DUE). The former affects the data integrity of the system while the later affects availability of the system. A silent data corruption (SDC) can be understood as an error in the data output that the user cares about. In contrast, detectable unrecoverable error (DUE) is defined as a detectable error that can cause a potential crash, however without any data corruption that user cares about.

There are several methods to calculate AVF of a hardware block. Statistical fault injection (SFI) is one of these methods that involves insertion of a large number of random faults in a system and subsequent checking of the percentage of these faults propagated to the output as an error [52]. To reach a statistical significance, the number of simulations required for SFI is quite large.

In the context of this thesis, we describe two soft-error vulnerability estimation techniques,

- **Architecturally Correct Execution (ACE) analysis** for addressable structures such as memory blocks
- **Error Probability Propagation (EPP) Analysis** for logic blocks such as pipeline stages.

Architecturally Correct Execution (ACE) analysis estimates vulnerability of a bit based on the fraction of clock cycles that bit needs to be correct for error-free program execution [54]. For a program execution of 1 million cycles, let us assume that a bit needs to be correct for only 0.3 million of cycles for the error-free execution of a program. During the remaining 0.7 million cycles, if an erroneous value of the bit does not affect the program output, then the

AVF of the bit is 0.3. The bit is termed as ACE bit for the 0.3 million cycles where a correct value is required and as an unACE bit for the remaining 0.7 million cycles. In short, a bit is ACE for a specific clock cycle if the bit can potentially propagate to the output and result in an error. The AVF of any hardware structure can be calculated based on ACE analysis. The AVF of a memory cell is the fraction of time the cell contains an ACE bit. The AVF of a memory block is the average AVF of all the bits in the memory block. For a hardware structure with N bits, AVF is defined as in equation 2.5 [52].

$$AVF_{structure} = \frac{\sum_{j=0}^{N} \text{ACE clock cycles}_j}{N \times \text{Total clock cycles}} \tag{2.5}$$

To estimate the soft error vulnerability of sequential logic blocks, a method based on error propagation probability is used [55]. A faulty value needs to propagate through combinational logic gates and needs to get latched by a flip-flop in order to be considered as an error. Hence, two levels of masking factors need to be considered in this scenario, (1) logic derating and (2) timing derating. Logic derating accounts for the probability that a glitch passes through the combinational logic gates and reaches at the input of a flipflop. Timing derating represents the overlap of the erroneous transient with the latching window of the flipflop such that a wrong value gets latched. The soft error rate of a node where the glitch originates can be determined as in equation 2.6

$$SER_{node} = \text{Nominal FIT} \times \text{Logic Derating} \times \text{Timing Derating} \tag{2.6}$$

, where Nominal FIT represents the technology-dependent raw soft-error rate.

The vulnerability factor or derating factor for a sequential logic block can be estimated as the product of probability of propagation from the node of origin to a flip-flop and the latching probability.

Impact of Runtime Masking Effects and Workload Execution

The particle strikes cause a bit flip or a wrong logic gate output and this erroneous value might not propagate to a user-visible output due to several masking effects. These masking effects can act at different levels of abstraction such as logic, architecture or application.

The masking mechanisms at different levels of abstraction is discussed below.

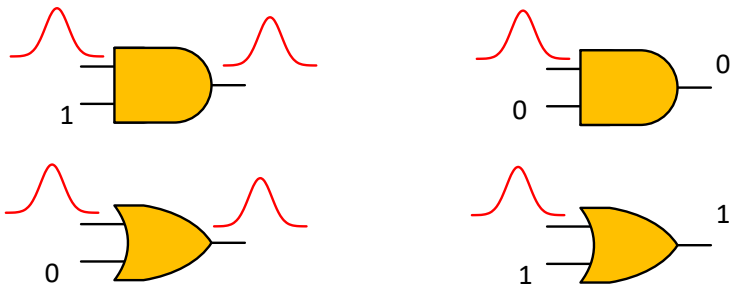


Figure 2.16: Illustration of logical masking or propagation of transient pulse from the input of a logic gate for different input combinations.

- Gate-level Masking: The gate-level masking mechanisms include logical, electrical and latching-window masking. Logical masking occurs if the erroneous value reaches to the input of a logic gate that do not have a role in determining the output (e.g. 0 as input to

an OR gate). Logical masking depends on the circuit topology and input since different logic paths are activated for different set of inputs [56]. The masking of transient pulse by different logic gates is illustrated in Fig. 2.16. For an AND gate, a transient pulse at one of the input gets masked if the other input is logic 0. For an OR gate, a transient pulse at one of the input gets masked if the other input is logic 1.

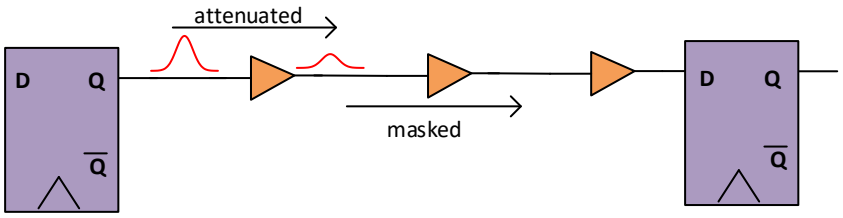


Figure 2.17: Illustration of electrical masking in the form of several stages of transient pulse attenuation.

Electrical masking occurs if the transient pulse gets attenuated before reaching the input of the flip-flop. This masking depends on the gates through which the transient propagates. As shown in Fig. 2.17, the transient glitch gets attenuated after propagating through a logic gate. This attenuation can eventually result in a masking while passing through several logic gates in the path before reaching the input of a flip-flop.

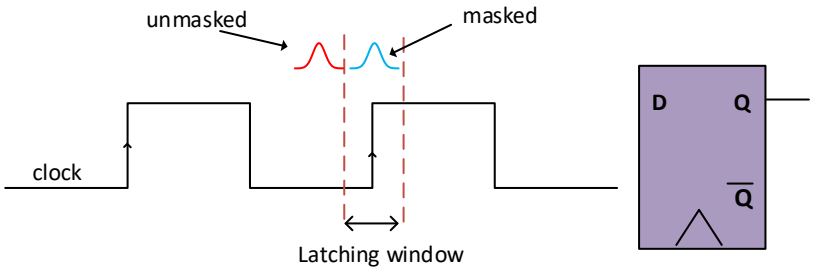


Figure 2.18: Illustration of latching-window masking where the transient pulse reaches outside the latching window of a flipflop.

Latching-window masking occurs when the propagated transient pulse reaches the input of a flipflop outside its latching window and hence, gets masked. To get stored in a flipflop on a specific clock cycle, the input signal should arrive at that clock edge within a time window shown as latching window in Fig. 2.18.

- Architectural and Application-level masking: In an architectural scope, an erroneous bit can get masked if it is written before being read. In application-level masking, an error bit in an instruction operand gets masked if a multiplication by zero is performed.

The masking of an error heavily depends on the workload under execution. The vulnerability of a processor to soft errors can vary across different workloads or even across different workload phases [57]. Hence, it is desirable to activate expensive protection mechanisms during high vulnerability phases and turn-off or switch to low protection mode during low vulnerability phases.

Soft Error Reduction Mechanisms

There are several device-level and circuit-level modifications that can help in reducing soft-error vulnerability of hardware blocks. However, these techniques cannot be adapted according to variations in vulnerability during runtime. Hence we look into dynamic runtime techniques

that can be tuned to achieve low or high protection according to the instantaneous vulnerability of the chip.

- Redundant Multi-Threading (RMT) [52]: This technique involves redundant execution of two identical threads for some time frame and the outputs are compared for any differences in the micro-architectural states. This technique provides excellent fault detection coverage at the cost of high performance overhead.
- Instuction Queue (IQ) squashing [58]: The scheme involves squashing of instructions that sit in vulnerable storage structures for longer durations. In the case of an instruction queue, a cache miss can cause instructions to reside for longer periods of time and hence, need to be squashed to lower the vulnerability.
- Adaptive Checkpointing and Restart: For efficient failure management, a checkpointing scheme at an adaptive rate can be used on the basis of failure history [59]. This scheme uses application replication and adapt the checkpoint period based on current failure rate.

2.3 Security Challenges

The hardware-part of a system was traditionally considered as a root of trust unlike its software counterpart. However, hardware-based vulnerabilities emerged due to the distributed, globalized and multi-step characteristics of IC design cycle [11]. In the supply chain, there are various entry points for an adversary to break the trust. It can occur during purchase of intellectual property (IP) cores from third-party design houses, fabrication in untrusted foundries or usage of third-party test facilities. The adversary can cause security issues in several ways by intruding the semiconductor supply chain. Stealthy malicious logic in the form of hardware Trojans can be inserted in a chip that can only get activated when a very rare combination of events occur which makes it extremely hard to detect. IP piracy is another attack where an adversary pirates the IP and illegally claims ownership of it. An untrusted foundry can manufacture excess copies of an IC and sell them illegally. Reverse engineering is another problem where an untrusted foundry extracts the design to the desired abstraction level. In addition, there can be side-channel attacks that extract secret information from a chip by listening to several physical parameters such as power consumption, electromagnetic emanations and photonic emissions. IC counterfeiting involves selling of illegally forged components by IC vendors. In the context of this thesis, we describe hardware Trojans in detail.

2.3.1 Hardware Trojans

With increasing complexity of semiconductor design, higher time-to-market pressure and increasing expenses, hardware firms have moved away from the vertical manufacturing model, where design specification to delivery was carried out in-house [60]. Many of these intermediate steps are outsourced in the current horizontal manufacturing model, and there are security issues arising from untrusted third-party design houses and foundries. An untrusted foundry, a computer-aided-design (CAD) tool, or a rogue designer at an IP design house can insert an additional hardware block, which can disrupt the functionality of a system after long in-field operation. This malicious circuitry is termed as a hardware Trojan that can be stealthy, and hard to get activated during chip testing or verification phases.

Trojan Architecture with Trigger and Payload

A functional hardware Trojan will have a trigger part and a payload part as shown in Fig. 2.19, where the payload modifies an internal signal A to A' when the trigger logic is activated. The

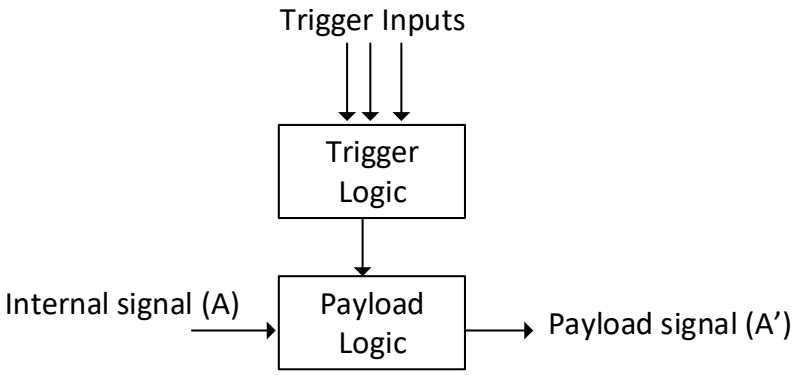


Figure 2.19: Illustration of a hardware Trojan architecture with a trigger logic and payload.

trigger logic activates the payload when a specific input vector is applied or the system enters a specific state. This activation condition can have very low probability to occur and this keeps the Trojan dormant under normal workload execution. Once the trigger is activated, the Trojan payload manifests as a change in the internal values of the hardware that can cause an information leakage, a denial of service (DoS) or a degradation in performance.

Trojan Taxonomy

Hardware Trojan designs are classified based on several features such as their insertion phase, abstraction level, activation mechanism, effects and location. The detailed taxonomy as shown in [61] is illustrated in Fig. 2.20. A short description of each characteristic is given below.

- Insertion Phase: Trojans can be inserted at various stages of the IC design cycle. In the specification phase, design constraints or functional specification can be altered. In the design phase, Trojan can be included as a part of third-party IP blocks or standard-cell designs. During fabrication phase, masks can be tampered or fabrication parameters can be altered to accelerate failure mechanisms. Untrusted IC testing can weaken Trojan detection probability by altering test vectors or by ignoring strategic faults. During assembly and packaging phase, the printed circuit board (PCB) on which the tested chip is deployed can have wires that are intentionally manipulated to leak side-channel information.
- Abstraction level: Trojans can also be classified based on the abstraction level at which they can be inserted. In the system level, the communication protocols and interconnections can be affected. In the development environment, a compromised CAD tool or an altered script can inject Trojans in designs. Each functional block in the form of registers and signals can be accessed at the register-transfer level and hence, a rare functional deviation of a module can be crafted in this level. In the gate level, the whole design is an interconnection of millions of gates and hence, Trojans with a small footprint can be inserted by adding a few extra logic gates. The power and timing characteristics of a circuit can be altered by modifying device sizes at transistor level. In the physical level, width of metal layers can be manipulated to achieve malicious effects.
- Activation Mechanism: Trojan classification can also be carried out based on how they get activated. There are always-on Trojans, and Trojans getting activated only when a specific trigger condition becomes true. This trigger condition can be activated internally or externally.

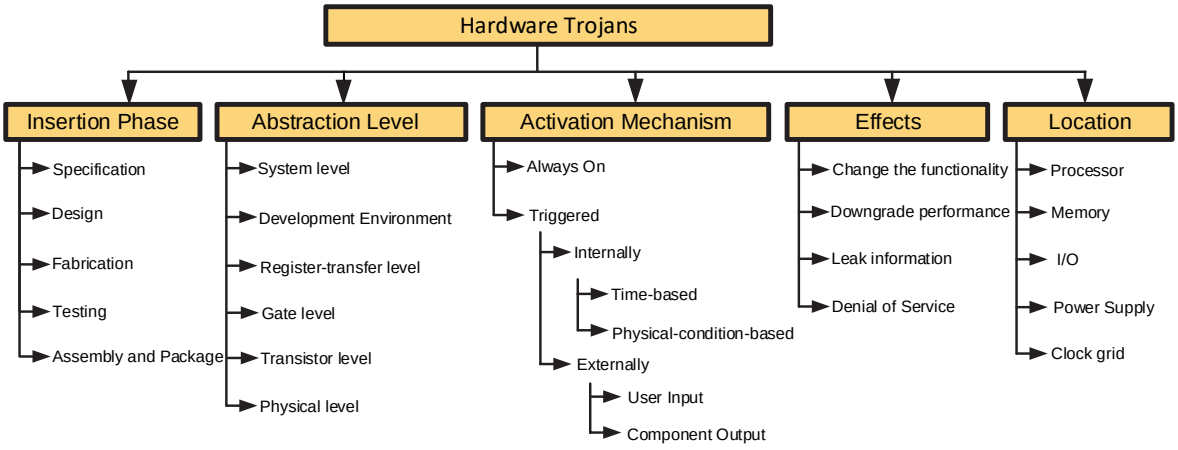


Figure 2.20: Hardware Trojan Taxonomy based on different characteristics of Trojans [61]

- Effects: A Trojan can belong to a specific class based on its effect or payload. The Trojan activation can change the functionality of the chip, downgrade the chip's performance, leak sensitive information or cause the chip to be unavailable for a user.
- Location: The location at which a Trojan is inserted can also vary, and this can be another criterion for classification. Trojan can be embedded into a processor core as part of one of its pipeline stages. It can lead to a change in the manner in which some of the instructions get executed. Trojans in memory blocks or their interface units can prevent access to certain memory addresses or cause data corruption. The chip's peripheral components can be manipulated to alter the data communication between a processor and external modules.
- Power Supply: The manipulation of the required supply voltage to the chip can cause more failures.
- Clock Grid: The clock grid can be manipulated to introduce clock skew, jitter or change the frequency to targeted components.

Trojan Detection Methods

Several measures can be taken to maintain a chain of trust to prevent Trojans at various levels [62]. However, it is not possible to conclude that an integrated circuit is Trojan-free. Traditional test and verification methods target the original functionality of the design corresponding to the specification. The design is not tested for functionalities outside the specification and hence, Trojans can exist without getting exposed in these stages. In this scenario, specific hardware Trojan detection methods are needed in addition to traditional test and verification methods.

In general, there are destructive and non-destructive techniques to detect Trojans [63]. In destructive techniques, a chip after fabrication is reverse-engineered by delayering, imaging and analysis. The gate-level netlist can be recovered through this process, and a fingerprint based on power, temperature and electromagnetic profiles is created. The profiles of other chips are compared with the fingerprint of the reverse-engineered chip to detect Trojan signatures. Non-destructive Trojan detection techniques can be classified into invasive and non-invasive. Non-invasive techniques do not make any design alterations. Invasive techniques add additional circuitry in the existing design to aid Trojan detection.

Non-invasive techniques involve comparison of the characteristics of a chip with a golden

model. This can be carried out during test or after deployment. Some of the specific techniques used in this category are discussed below.

- Runtime Detection: Runtime detection approaches are considered as the last line of defense against Trojans, which are hard to get activated during pre-silicon and testing phases. There are techniques to monitor the real functionality of chips during runtime [64]. When a deviation in normal functionality is detected, appropriate countermeasures can be triggered. In addition, verifiable hardware guard modules are used to monitor the functionality of a processor with periodic checks [65].
- Logic Testing: In logic testing techniques, statistical generation of advanced test vectors is carried out to maximize the triggering probability of Trojans. Trojans are hard to be activated if inserted in circuit nodes with low controllability and observability. Hence, test vectors are generated to increase the toggling rate of nodes, such that it improves the probability of Trojans to get activated compared to random patterns [63].
- Side-Channel Analysis: The extra logic inserted as part of Trojan circuitry can affect path delays, current transient or power consumption of the chip. These effects caused by Trojan circuit can be captured by measuring these physical parameters through side-channel analysis. The main challenge is to differentiate the effect of a Trojan from that of process variation.

2.4 Machine Learning Basics

The increasing complexity of hardware designs, and the complex dependence of several resilience mechanisms on the usage pattern of the ICs in the field of operation make deterministic modeling of these mechanisms inaccurate. Hence, data-driven modeling based on machine learning algorithms emerges as a potential alternative to generate prediction models in the field of reliability and security. In this context, a description of relevant machine learning algorithms and methods are discussed.

Based on the form of data available for learning, learning methods can be classified into supervised and unsupervised. In supervised learning, a function, mapping the input and output variables, is inferred from data samples that are labeled with the class they belong to. Unsupervised learning is used for extracting patterns in a set of data samples that are not labeled.

2.4.1 Supervised Learning Techniques

In supervised learning, data samples are annotated with the corresponding class labels that are meant to be predicted for unseen samples. Supervised learning algorithms can be attempted to solve classification or regression problems. In classification, each sample belongs to a class and the prediction model is generated to predict the class of a sample from the feature values of the sample. In a regression problem, the prediction output is a real value instead of a label. In the context of this thesis, two supervised learning algorithms are discussed in detail, (1) Support Vector Machines, and (2) Decision Trees.

Support Vector Machines

Support Vector Machines (SVMs) belong to the class of supervised learning techniques that can be used for both classification or regression problems. SVMs construct a hyperplane or a set of hyperplanes to separate data samples belonging to different classes. If the data is not linearly separable, a kernel trick is used with specific functions to map the data samples to a higher dimensional space, where the samples can be linearly separated. The different kernels used

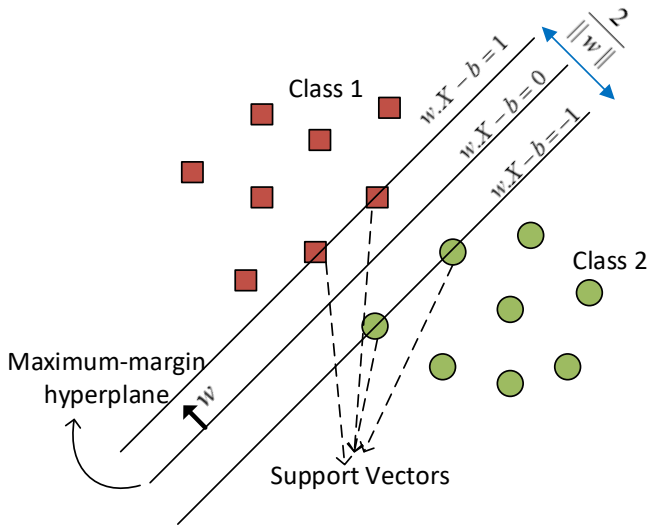


Figure 2.21: Illustration of linear two-class classification with maximum-margin hyperplane in support vector machines.

in SVM include polynomial function, Gaussian radial basis function, and hyperbolic tangent function. SVMs are efficient in high dimensional space, and also are memory efficient.

For a classification problem involving two classes, the training data set of n samples with m feature dimensions can be expressed as pairs in the form $(X_i, y_i)_{i=1}^{n}$, where X_i denotes feature vector with features $x_1, x_2, ..x_m$, and y_i denotes the target class label for each corresponding sample X_i. The objective is to find the optimal hyperplane that separates the data points in these two classes with maximum margin. The data samples that are used in defining the hyperplane are called support vectors.

With these set of data points, a hyperplane $wX + b = 0$ as shown in Fig. 2.21 needs to be found that maximizes the distances between the hyperplane and the nearest data points to both classes. Two other lines are shown with equations $w * X - b = 1$ and $w * X - b = -1$ such that the region marked by $w * X - b \geq 1$ belongs to class 1 and the region marked by $w * X - b \leq -1$ belongs to class 2. The distance between these two lines is $\frac{2}{||w||}$, and the maximum-margin hyperplane lies in the middle of these two lines. The problem of classification can be formulated as an optimization problem, i.e., minimizing $||w||$ subject to $y_i(w * X_i - b) \geq 1$ for $i = 1, 2, ..n$.

For non-linear classifications, different kernel functions are used to map the data points to a higher dimensional space. SVM-based estimators can be generated by selecting appropriate kernels, kernel parameters and a soft margin parameter (C). There is a trade off between maximizing the margin and minimizing the training error and this can be controlled by tuning value of C. Increasing the value of C can provide better accuracy over training data, but this may cause overfitting or poor generalization performance on unseen data. In radial basis kernel, the influence of a single training sample can be controlled by gamma parameter.

Decision Tree Learning

A learned model can be represented by a decision tree, and it approximates a discrete-valued function to make a prediction on the target value. Classification of a data sample is achieved by traversing the learned tree from the root node to one of the leaf nodes based on the attribute tests given at each node split. Decision trees can also be used to solve regression problems, where the prediction target can take real values.

Learning is achieved by deciding the best attribute test to split at each node starting from the root node. The quality of split is estimated based on different metrics such as Gini impurity or information gain [34]. Information gain is based on the concept of entropy. The decision is made by calculating the entropy of training data, and estimating information gain achieved by each split. Information gain can be considered as the reduction in entropy due to a specific attribute split, and the attribute that causes maximum reduction in entropy (maximum information gain) will be selected for splitting the root node. This is repeated until the leaf nodes are reached.

A decision tree based on the example given in [66] is shown in Fig. 2.22. The four attributes in this 14-point data set are *outlook* (*sunny*, *overcast*, *rain*), *temperature* (*hot*, *mild*, *cool*), *humidity* (*high*, *normal*), and *windy* (*weak*, *strong*), and the target variable is *playTennis* with binary (*yes* or *no*) values. The entropy (H) of a collection of samples S with positive and negative labels can be represented as equation 2.7,

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\log_2 p_+ - \log_2 p_- \tag{2.7}$$

where p_+ represents the fraction of positive samples in S, and p_- represents fraction of negative samples in S. The entropy of the 14-sample data set with nine positive and five negative samples can be represented as equation 2.8.

$$H(S(9+,5-)) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \tag{2.8}$$

The core of learning using decision trees lies in deciding which attribute to be selected to split each node in the tree. To decide between the attributes $Humidity$ and $Wind$, the information gain (IG) of both attributes are calculated as the reduction in entropy caused by the split as in equation 2.9 [66],

$$IG(S,A) = H(S) - \sum_{\nu \in \text{Values}(A)} \frac{|S_\nu|}{|S|} H(S_\nu) \tag{2.9}$$

where $Values$(A) denotes the set of values that attribute A can take, and S_ν is the subset of S for which attribute A has the value ν.

With the attribute $Humidity$, the data set $S[9+,5-]$ is split into a subset $S1[3+,4-]$ for $Humidity = High$, and $S2[6+,1-]$ for $Humidity = Low$. With the attribute Wind, the data set $S[9+,5-]$ is split into a subset $S3[6+,2-]$ for $Wind = Weak$ and $S2[3+,3-]$ for $Wind = Strong$.

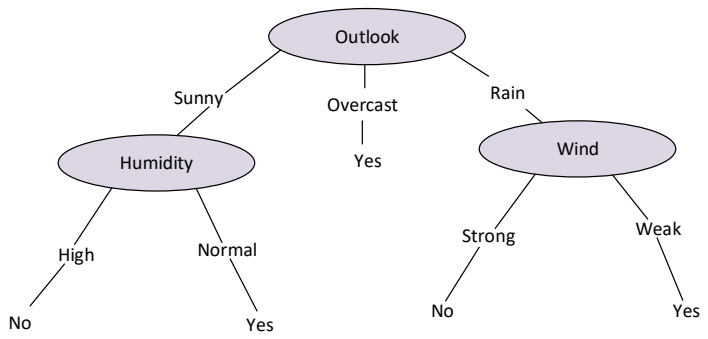


Figure 2.22: Illustration of a decision tree demonstrating a two-class classification [66].

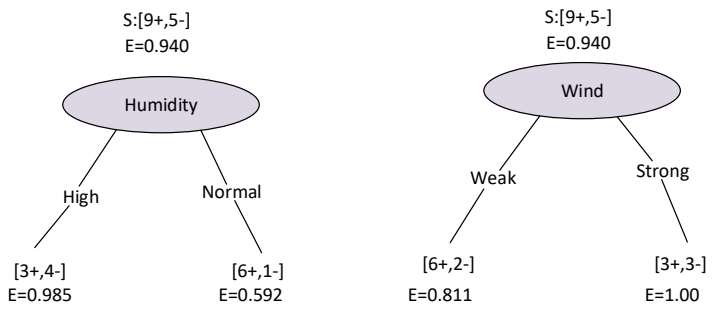


Figure 2.23: Illustration of information gain comparison between two attributes [66].

The information gain in the case of these two splits illustrated in Fig. 2.23 can be calculated as in equation 2.10 and equation 2.11 [66].

$$Gain(S, Humidity) = 0.940 - \frac{7}{14}0.985 - \frac{7}{14}0.592 = 0.151 \tag{2.10}$$

$$Gain(S, Wind) = 0.940 - \frac{8}{14}0.811 - \frac{6}{14}1.0 = 0.048 \tag{2.11}$$

Since $Gain(S, Humidity)$ is higher, $Humidity$ is chosen over $Wind$ to decide splitting of the root node.

The decision-tree based models are interpretable white box models. However, they are prone to overfitting and hence, techniques such as pruning are required to reduce overfitting with smaller tree size.

2.4.2 Unsupervised Learning Techniques

Unsupervised learning techniques are used when the training data has a set of feature vectors with no target values. In this case, the objective is to find similar groups in the data (clustering), discover distribution of data in the input space (density estimation) or reduce dimensions (dimensionality reduction) of the data [67]. Clustering involves grouping of data samples into clusters such that a sample is more similar to the cluster it is assigned than the other clusters. In the context of this thesis, a popular clustering technique called k-means clustering is described in detail.

K-means Clustering

The objective of k-means clustering is to find clusters and cluster centers from an unlabeled data set. Starting with a fixed number of clusters, k-means minimizes within-cluster variance by iteratively moving the cluster centers [68].

The steps followed in k-means after fixing an initial set of cluster centers are as follows [68].

- For each cluster center, identify the set of data points lying nearer to that center compared to other cluster centers.
- the mean of data points in each cluster is calculated, and this mean becomes the new cluster center for each corresponding cluster.

These two steps are repeated until convergence. The initial cluster centers are randomly chosen. Since it falls into local minima, the whole process is repeated several times to develop better models.

2.4.3 Ensemble Techniques

Ensemble learning involves employing multiple estimators for a prediction model. In this way, the combined prediction performance will be better than the prediction performance of any individual estimator. According to the methods in which the base learners are combined, ensemble learning can be divided into bagging, boosting and stacking. The two steps in ensemble learning are (1) generating multiple base learners, and (2) combining these base learners to form the composite predictor [68].

In bagging, several base learners are generated from bootstrapped samples of training data independently, and a combination of their predictions is used as the final output as shown in Fig. 2.24. In most cases, the average of the estimator outputs is taken as the final output.

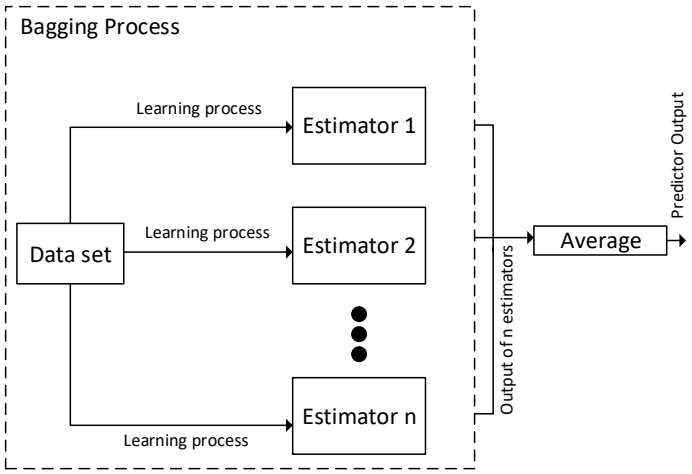


Figure 2.24: Illustration of bagging technique.

In boosting, a sequential generation of base learners is followed. The initial learner can be simple, and the subsequent learner is trained upon the training samples that are inaccurately predicted by the initial learner. This sequential process of generating estimators is illustrated in Fig. 2.25 with each estimator giving higher weight for samples inaccurately predicted by the previous estimator. A weighted average of the estimator outputs is used as the final prediction.

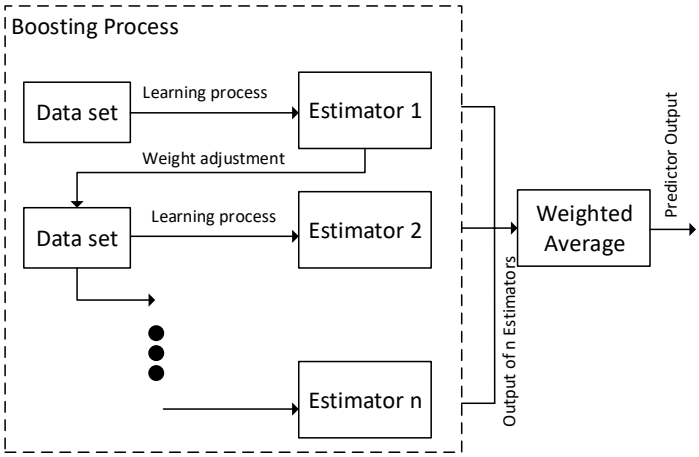


Figure 2.25: Illustration of boosting technique.

2.4.4 Feature Selection

In learning problems, the target variable may be determined by only a subset of the input features. In this case, if irrelevant features are used in function approximation, the model complexity increases, and this leads to overfitting. In addition, the computational cost of prediction will increase with more number of features. Hence, it is desirable to select only those features that are involved in a functional relationship between inputs and output. Feature selection techniques are classified into filter and wrapper methods. Wrapper methods generate a prediction model on a subset of training data, and estimates the error rate to decide the performance of the feature subset. This is a computationally intensive process. Filter methods employ metrics such as correlation coefficient or mutual information to eliminate irrelevant features at low computational cost. Some of the feature selection techniques are discussed below.

- variance threshold [34]: The variance of each feature is calculated, and the feature is eliminated if the variance is below a threshold value. A feature of zero variance has no information.
- univariate feature selection [34]: This technique selects the relevant features based on univariate statistical tests. The input features that possess strong statistical correlation with the output variable are selected.
- recursive feature elimination [34]: An initial model is generated with all features and the importance value of each feature is estimated based on feature coefficients. The least important feature is eliminated, and then the model is again constructed with the remaining features. This process is repeated until the required number of features remains.

2.5 Summary

In this chapter, a background of dependability along with the major attributes and threats are discussed. The reliability challenges, the impact on circuit timing and soft error vulnerability, and various reliability monitoring and mitigation techniques are explained. Afterwards, security challenges are discussed with an emphasis on hardware Trojans. Finally, a brief introduction on machine learning terminologies and algorithms is provided in the context of this thesis.

3 Dynamic Aging Monitoring and Delay Prediction

3.1 Overview

Run-time solutions based on online monitoring and adaptation are required for resilience in nanoscale integrated circuits, as design-time solutions and guard bands are no longer sufficient. Bias Temperature Instability (BTI)-induced transistor aging, one of the major reliability threats in nanoscale VLSI, degrades path delay over time and may lead to timing failures. Chip health monitoring is, therefore, necessary to track delay changes on a per-chip basis over the chip lifetime operation. However, direct monitoring based on actual measurement of path delays can only track a coarse-grained aging trend in a reactive manner, not suitable for proactive fine-grain adaptations.

In this chapter, we propose a low cost and fine-grained workload-induced stress monitoring approach, based on machine learning techniques, to accurately predict aging-induced delay. We integrate space and time sampling of selective flip-flops into the runtime monitoring infrastructure in order to reduce the cost of monitoring the workload. The prediction model is trained offline using support-vector regression and implemented in software. This approach can leverage proactive adaptation techniques to mitigate further aging of the circuit by monitoring aging trends. Simulation results for realistic open-source benchmark circuits highlight the accuracy of the proposed approach.

The rest of the chapter is organized as follows. Section 3.2 introduces the work in detail, presents a motivation and also lists the contributions. Section 3.3 overviews related prior work. Section 3.4 describes the overall methodology underlying aging monitoring and prediction. Section 3.5 presents the method used for designing and training the prediction model. Section 3.6 explains the online stress-monitoring technique used for aging prediction. Experimental results are presented in Section 3.7. Finally, Section 3.8 summarizes the chapter.

3.2 Introduction, Motivation and Contributions

Design-time solutions and guard bands for resilience are no longer sufficient for integrated circuits (ICs) fabricated at nanoscale technology nodes. This is due to large variations in the fabrication process, workload, and working conditions, which makes design-time solutions inefficient and impractical [69, 70]. Therefore, there is a need for run-time solutions based on real-time monitoring and adaptation. Chip manufacturers incorporate dynamic adaptation strategies such as voltage/frequency scaling, body biasing, and thermal management in response to slow-down due to aging, high temperature, current surge, process variations etc., but the *adaptation policies* are *static*. The decisions taken in response to system behavior are "hard-coded", e.g., in look-up tables, boot ROM, firmware, etc.; hence, today's adaptation methods are more reactive than predictive, and there is no solution available to train the adaptation policy dynamically in response to changes in chip behavior.

Most existing delay monitoring schemes can only support reactive aging mitigation techniques [22, 71, 72]. A reactive mitigation approach can be too late to prevent the occurrence

of a system failure. Hence, there is a need for proactive aging mitigation where suitable adaptation actions are adopted in advance. An ideal proactive delay monitoring technique should be capable of capturing fine-grained aging trends, i.e., monitoring aging trends in the scale of typical workload phases (in the order of milli/micro seconds). Some of the existing proactive techniques require the system to be physically aged in order to capture the aging trends [71, 73]. The time-constant for aging is in the order of weeks or months. Hence, these methods cannot be used for fine-grained aging monitoring to support proactive aging mitigation.

The design of an effective aging-aware delay monitoring scheme for proactive aging mitigation is challenging for two reasons: (i) it is impractical to accurately measure the delay degradation of a circuit over a short time period. This is because traditional delay monitoring sensors track path delays, and the path-delay degradation for a short time period is too small for these sensors to capture, i.e., they can only track coarse-grained aging trends; (ii) degradation rate depends on the currently running workload and working conditions. In other words, different workloads can degrade the timing of various circuit paths differently and the sensors placed during design time cannot capture this workload effect.

In this chapter, we present a method to perform low-cost and fine-grained workload-induced stress monitoring for accurate aging-induced delay prediction. The key idea here is to track recent circuit activity and extrapolate these trends to activate different mitigation measures, which include task migration [74], dynamic voltage and frequency scaling (DVFS) [75], adaptive body biasing (ABB) [76], and core power gating [75], based on the severity of the captured stress. This approach involves the ranking of workloads or workload phases based on their aging-stress severity. This ranking can then be used to activate aging mitigation techniques proactively. Note that the aging computation and prediction in this work are made on the basis of aging projection. The circuit is simulated with a representative workload for a short time period and the extracted behavior of the circuit is then extrapolated to estimate the aging trends. The workloads are then ranked based on these trends.

It has been shown that the impact of workload on aging can be computed on the basis of the gate-level *signal probabilities* (SPs) [77]. However, it is computationally impractical to simulate large designs at the gate-level on a cycle-by-cycle basis for realistic workloads. In addition, online monitoring of circuit node SPs of a large design with millions of gates is impractical due to area and power overhead constraints. We show in this chapter that, to rank workloads in terms of severity of aging, we can also use the SPs of a small set of flip-flops as a surrogate measure of signal probability. These flip-flops are chosen at design time by carefully studying the correlation between aging and flip-flop SPs for representative workloads. We utilize Support Vector Machines (SVM) to develop a predictive model for estimating aging based on SPs. This model is subsequently utilized at run-time as a software thread to evaluate the online workload-induced aging-stress severity. Moreover, the monitoring of a small set of flip-flops reduces the hardware cost of monitoring and is computationally less burdensome since not all flip-flops in the design need to be monitored at runtime. Furthermore, we propose a technique to sample these selected flip-flops infrequently in order to reduce power overhead.

Our experiments on two embedded processors running realistic workloads show that the accuracy of the proposed aging prediction method is extremely high (the prediction is nearly perfect), even in the case when the prediction is based only on the SPs of 0.64% of the total number of flip-flops. The key benefits of the proposed aging prediction method are listed below: (i) **Proactive**: the aging trend is predicted before a measurable delay degradation happens, hence countermeasures can be considered in a timely manner (ii) **Accurate**: the simulation results show that the correlation coefficient of actual aging trends and predicted aging trends is extremely high (higher than 0.9, where 1.0 indicates perfect correlation), (iii) **Low overhead**: since the proposed technique is based on the SP values of a small set of flip-flops, it imposes minimal area and power overhead.

A space-sampling approach is proposed that selects a small set of flip-flops offline based on their aging information; only these flip-flops are monitored during runtime. We introduce two different techniques to achieve space sampling: (1) correlation-based and (2) fan-in cone-based flip-flop selection. We advocate the fan-in cone-based method as the preferred approach because of its lower offline-characterization requirements.

3.3 Related Work

Worst-case guard-bands have been used in industry for many years. Designers use this conservative approach to ensure that circuits will operate under worst-case temperature, voltage, and workload conditions [78][79]. However, these worst-case assumptions are too pessimistic and hence, performance of the circuits are significantly compromised [71][80]. In addition, due to statistical behaviour of aging mechanisms, similar devices may age differently even for the same environmental and workload conditions, which makes aging in the field even less predictable [81]. Finally, guard bands cannot keep up with aging challenges in newer technologies [82][83].

An alternative approach for achieving resilience is referred to as on-line circuit failure prediction. This approach predicts the occurrence of a failure before errors actually occur [71]. Prediction requires information collection in real time on temperature, signal activity, signal delay, etc, and analysis of the data. Such information is usually collected through ring oscillators, temperature sensors, delay sensors, and special circuit structures. Circuit failure prediction can be used to take actions to prevent the chip from failing, and these actions are collectively referred to as on-line self-healing [84][85].

Dynamic reliability management (DRM) techniques were proposed in [86][87].

In [88], the processor uses runtime adaptation to respond to changing application behavior to maintain its lifetime reliability target. In [89], architectural-level models were developed for lifetime-reliability-aware analysis of applications and architectures. In [90], selective redundancy is applied at the micro-architectural level. The method proposed in [91] slows aging through application scheduling and voltage changes at key moments. In [92], the processor relies on compute cycles in which computations get finished early under higher supply voltages, followed by idle cycles that provide relaxation to recover from aging.

Dynamic adaptation is often implemented through the use of embedded lookup tables (LUTs). The adaptation policies are encoded and stored in the LUTs, but they are predetermined at design time. A representative LUT-based adaptation policy was proposed in [93]. When a hardware unit changes its state from standby to active mode, the power-management unit fetches a codeword from the LUT, which is provided as input to the adaptive body-bias controller. Another LUT-based technique adjusts the power-supply voltage and body bias to compensate for aging [94]. In [95], a LUT stores, on a case-by-case basis, the optimum values of body bias, supply voltage, and clock frequency to compensate for droop and temperature variations. In [73], a built-in proactive tuning (BIPT) system was proposed, based on a canary circuit that generates predictive warning signals. In [96], the authors proposed control policies to achieve better energy efficiency and lower cost than worst-case guard-banding. Dynamic cooling was introduced as an additional tuning parameter.

A number of the above methods have been adopted for industrial circuits. In [97], the design of a Texas Instruments 3.5G baseband and multimedia applications processor is presented. This SoC consists of multiple independently controlled power domains that use dynamic voltage/frequency scaling and adaptive voltage scaling. In addition, it implements adaptive body biasing [95]. In [98], runtime adaptation techniques are described for Intel's Itanium architecture microprocessor. The core supply voltage and clock frequency are dynamically modulated

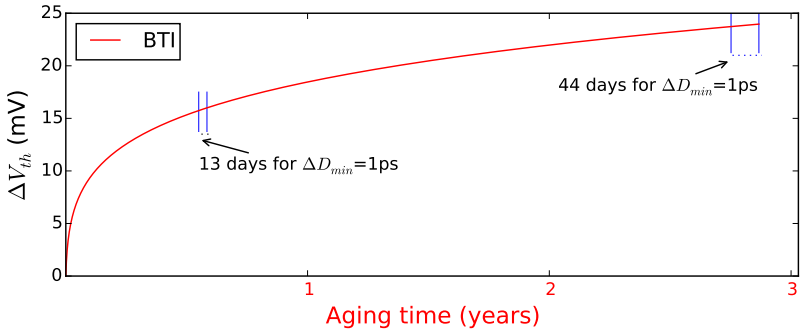


Figure 3.1: The difference in ΔD_{min} at different time points in the lifetime of a circuit.

in order to maximize performance within the power envelope [99]. In [100], the SmartReflex power-management techniques implemented on the OMAP3430 Mobile Multimedia Processor are presented. Active power reduction is achieved through aggressive voltage/frequency scaling and process compensation.

To enable adaptive aging mitigation, online monitoring of circuit degradation is required. State-of-the-art monitoring methods include in-situ sensors [22][71], tunable replica circuits [72], and representative critical reliability path based monitoring [101]. However, these monitoring methods suffers from three problems: (i) additional hardware or design modifications, which are undesirable due to their associated area and power overhead, (ii) intrinsic conflict between the accuracy of the method and hardware overhead, and (iii) existing techniques cannot track short-term aging trends, since they are designed to capture an aggregated measure of the degradation when a significant delay increase takes place. Therefore, the mitigation techniques based on these monitoring systems are *reactive*. *Proactive* aging mitigation has been advocated as a promising approach that can be more effective than reactive techniques [71, 73].

The proposed method is conceptually different from any critical-path based aging prediction method [101, 102]. During the online phase, critical-path monitoring methods rely on aging sensors. These sensors measure the actual path-delay increase, which typically occurs in longer time-scales. Even if high-resolution aging sensors [103] are used to capture the aging rate, the circuit delay needs to increase at least a few pico seconds in order to be detectable. The duration required for this minimum aging-induced delay increase (ΔD_{min}) is typically in the scale of days or weeks. In addition, circuit aging has a logarithmic relationship with time, i.e., the rate of circuit delay degradation decreases over time. Hence, the sensors cannot provide aging-rate information within an acceptable time for mitigation actions. Fig. 3.1 shows ΔD_{min} at different points of time in a three-year operation of the circuit. It implies that an aging sensor of resolution 1 ps requires around 44 days to capture a delay increase in a processor operating at 1 GHz after three years of operation. Since the circuit is closer to a timing failure after three years in comparison to its initial state, this significant increase in ΔD_{min} is unacceptable for any aging mitigation technique.

In contrast, our method evaluates the aging stress imposed by the workload-under-execution which does not require the physical aging of any of the critical paths. The flip-flops are tracked online to capture the characteristics of the workload, and we obviate the need to track the physical delay increase of the corresponding circuit path. As a result, we can achieve timely workload-stress estimations and aging rates that can be used in turn to take appropriate proactive and fine-grain mitigating actions and prevent the circuit from aging at higher rates.

3.4 Proposed Methodology

The proposed aging prediction method is based on tracking the severity of the workload-induced run-time stress. This information can be used to guide fine-grained proactive aging mitigation policies. The flip-flops in the design that can represent the workload information are identified based on their behaviour during offline workload characterization. A prediction model is constructed based on the correlation between the behaviour of these selected flip-flops and the aging behaviour of the circuit under different workload executions. These selected flip-flops are monitored during runtime using additional monitoring hardware in order to extract their online behaviour. This extracted information is translated by the prediction model to corresponding aging trends. The output from the aging prediction model can be used to proactively actuate aging mitigation measures. Details about aging mitigation measures are not presented in this chapter since the focus here is on aging prediction.

The overall flow of the method can be divided into two parts: (1) Offline correlation analysis, which consists of representative flip-flop selection, and prediction model generation, (2) Run-time stress monitoring. During offline correlation analysis, a small set of representative flip-flops are selected whose signal probabilities correlate with the aging trend of the entire circuit. Then, based on the SPs of these representative flip-flops, an aging-severity prediction model is generated by extracting workload information. This prediction model is deployed as a software in the system. During runtime, on-chip hardware is used to capture the features relevant for predicting aging-induced delay, which are then fed to the prediction model in real time. This hardware, samples the states of these representative flip-flops at runtime to extract their signal probabilities, using a so-called *time sampling* scheme, which is then sent to the software-based prediction model, trained and developed at design time, to predict aging severity of the running workload.

The subsequent sections describe the details of the offline correlation analysis and runtime monitoring schemes, respectively.

3.5 Offline Correlation Analysis and Prediction Model Generation

During design time, the effect of workload on aging-induced delay is analyzed. A set of representative workloads is applied to the circuit and the SPs of all nodes are calculated for each workload. The amount of circuit delay is projected by assuming the same amount and type of workload over a fixed period of time. The delay of each gate is updated based on the SP values and then aging-aware static timing analysis is performed [102]. In this way, the aging-induced delay corresponding to each representative workload is obtained. We select a small set of representative flip-flops that can represent the aging behaviour of the whole circuit as explained Section 3.5.3. The SPs of these representative flip-flops together with the circuit delay values are used to train an SVM-based model. The goal here is to capture the impact of workload on circuit delay by constructing an analytical aging-prediction model.

The flow-chart in Fig. 3.2 illustrates the procedure of construction of the aging prediction model. In summary, this phase involves: (i) estimation of BTI-induced delay degradation, flip-flop SP extraction and selection of representative flip-flops, and (ii) construction of an aging prediction model using SVM.

3.5.1 Aging-Induced Delay Degradation and SP Extraction

Bias Temperature Instability (BTI)-induced threshold voltage degradation of the transistors in logic gates depends on their input SP values. For each representative workload, SPs of the primary inputs are propagated in order to determine the SPs of the internal nodes. This is

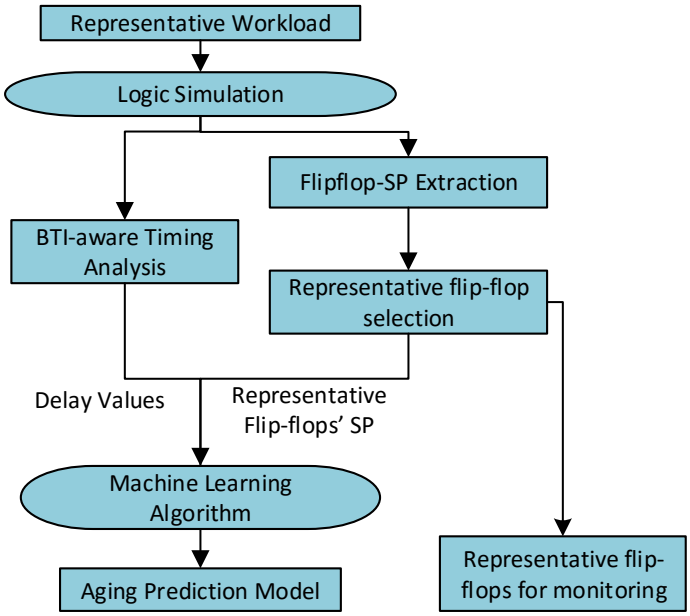


Figure 3.2: Flow-chart showing the steps involved in the offline characterization phase.

achieved by annotating the SP values of the primary inputs and carrying out zero delay simulation using Synopsys Power Compiler. The resulting SPs of the internal nodes are extracted from the switching-activity-interchange-format (SAIF) file generated by Synopsys Power Compiler. SPs and switching activities are required for power-profile analysis, and commercial logic simulators have a built-in feature to dump such information for desired signals and time intervals in a SAIF file. The SP values at the inputs of each logic gate are then translated to the threshold-voltage degradation of the transistors within these gates.

During logic synthesis, the Standard Delay Format (SDF) file of the circuit is generated, which contains the delay information about all the flip-flops and logic gates in the netlist. For each workload under consideration, these delay values of logic gates are updated to aging-induced delay values based on the SP values at the gate inputs. Static timing analysis with the updated gate delays yields an estimate of the aging-induced critical delay of the circuit. Note that STA is block-based and it implicitly considers the delay increase in all possible circuit paths.

Aging-aware timing analysis is carried out using the following steps:

1. Define the circuit netlist as a graph with logic gates as nodes and the wires as the edges.
2. Traverse through the graph, covering the forward cone of each flip-flop in the circuit until the next flip-flop in the path is reached.
3. For each node in the circuit, the propagation delay until that node is calculated by performing sum and max operations in a block-based analysis.
4. The maximum delay for the entire circuit is obtained in this way and is defined as the aging-induced circuit delay.

The only difference of this approach from conventional STA carried out by a commercial tool is that, we use aging-induced delay values of logic gates to compute circuit delay.

3.5.2 Predictor Training Using Support-Vector Machines

The next step in the offline phase is to train a predictor based on a training set consisting of the flip-flop SPs and aging-induced delay for several representative workloads. In this work, we use SVM, a supervised learning algorithm, to train the predictor. SVMs are very popular because of their resilience to over-fitting, robustness to outliers, and high prediction accuracy for a wide range of applications [104].

The delay shift due to BTI is a deterministic function only when the SP values of all nodes in the critical/near-critical paths are available in the online phase. Since it is expensive to track SPs of all of these nodes, we track a few selected flip-flops and deploy SVM to get accurate aging estimation at low cost. Note that we do not record and use the SPs of all the flip-flops in the design. Therefore, it is not feasible to use deterministic methods that rely on a large volume of data. SVM is efficient in this scenario since it utilizes a small subset of features after feature elimination.

An SVM-based model is trained based on the set of $(SP, Delay)$ pairs from the final training set. Before the training process, each set of SPs in the training set is converted to a vector form. We refer to this vector as the SP vector. For training an SVM-based predictor, the SP vectors are mapped into a higher-dimensional feature space and an optimal hyperplane (regression line) is constructed in this space. Let $(x_i, y_i)_{i=1}^{S}$ denote the training set, where $x_i \in \mathbb{R}^d$, and $y_i \in \mathbb{R}$. The training set consists of S SP vectors, $x_1, x_2, ..., x_S$, and each SP vector has d features and a corresponding target value (aging-induced delay), y_i, which is a real number. In this work, the ϵ-support-vector regression technique has been used. The objective of ϵ-support-vector regression is to find a regression line that fits most points with in ϵ-margin ($\epsilon > 0$), as shown in Fig. 3.3. The equation has the following form: $f(x) = \sum_{i=1}^{S} \beta_i k(x, x_i) + b$, where $b = \frac{1}{S} \sum_{i=1}^{S} (y_i - \frac{\beta_i}{|\beta_i|}\epsilon - (\sum_{j=1}^{S} \beta_i k(x_i, x_j)))$, $k(x_i, x_j)$ is the kernel function used to map the SP vectors to a high-dimensional feature space, and $(\alpha_i)_{i=1}^{S}$ are the Lagrange multipliers introduced during the process of derivation of this equation.

To illustrate the SVM-based regression methodology, consider a hypothetical scenario with four training workloads. In addition, assume that this system has five flip-flops. We form the training set as shown in Table 5.1, where the second column in each row corresponds to the set of SPs and the third column corresponds to the normalized aging-induced delay value. Using this training set and a linear kernel, we obtain the following model to predict aging-induced

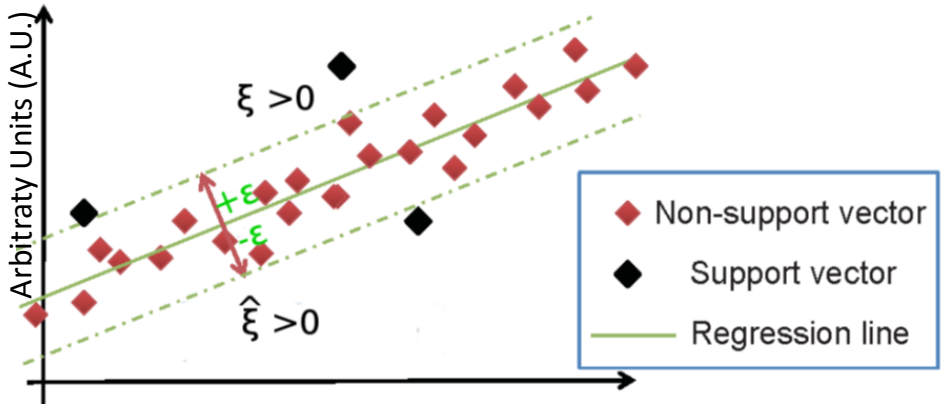


Figure 3.3: Illustration of the support-vector regression model.

Table 3.1: Hypothetical training set with four data samples

Workload	Set of SPs {SP1,SP2,SP3,SP4,SP5}	Normalized Aging-Induced Delay
W1	0.1, 0.1, 0.4, 0.8, 0.2	0.4
W2	0.2, 0.6, 0.2, 0.1, 0.5	0.8
W3	0.1, 0.4, 0.1, 0.6, 0.5	0.1
W4	0.2, 0.1, 0.7, 0.4, 0.3	0.6

delay based on a given set of SPs ($s = \{s_1, s_2, s_3, s_4, s_5\}$):

$$\begin{aligned} f(\boldsymbol{x}) &= 4.45 \cdot s_1 + 1.67 \cdot s_2 + 0.22 \cdot s_3 \\ &\quad +0.20 \cdot s_4 - 2.31 \cdot s_5 + 0.01 \end{aligned} \tag{3.1}$$

Suppose we have a set of SPs {0.2 0.1 0.7 0.4 0.3}, which is the fourth row in Table 5.1, as the input to our model. The aging-induced delay is evaluated to be $y = 0.6$ using (5.5). Let us consider a second set of SPs {0.1 0.2 0.6 0.1 0.1}. In this case, the model evaluates to $y = 0.7$.

The runtime of the prediction model depends on the number of its variables, i.e., the number of monitored flip-flop SPs. Since a typical design has millions of flip-flops, the runtime can be considerably large. Moreover, it is extremely expensive to monitor all the flip-flops in every clock cycle. In the following sub-sections, we describe space and time sampling-based approaches to reduce these overheads.

3.5.3 Representative Flip-Flop Selection (Space Sampling)

We propose two methods to select a small subset of flip-flops whose SPs are highly correlated to aging-induced delay: (i) Correlation-Based Flip-flop Selection and (ii) Fan-in Cone-Based Flip-flop Selection.

Correlation-Based Flip-Flop Selection

In correlation-based flip-flop selection, flip-flops are selected based on their behaviour for different workloads under consideration. The overall flow of this method is illustrated in Fig. 3.4. During workload characterization, several workloads with varying characteristics are executed on the synthesized gate-level netlist of the chip. The SPs of flip-flop values under various workloads are extracted. For the same workloads, aging-induced delay values due to the workload-stress are also extracted using the aging-analysis framework. A feature-selection method is then employed to select significant flip-flops whose SPs are highly correlated with aging-induced circuit delay. The benefits of employing feature selection are: (i) it reduces overfitting by eliminating redundant data, (ii) it improves accuracy by eliminating irrelevant data, and (iii) it reduces training time by reducing the size of the training set.

Let us assume that N *workload phases* are available at design time for training. The aging-correlation analysis flow, explained in Section 3.5, can be used to generate a training set $\{(SP_i, Delay_i)\}_{i=1}^{N}$ of size N, where the set SP contains the SPs of all M flip-flops $SP_i = (SP_{ij})_{j=1}^{M}$ and $Delay_i$ represents the aging-induced delay under each workload phase in our design. Our goal is to find a set of m flip-flops, $m \ll M$, whose SPs are highly correlated with aging-induced delay. This can be carried out using a univariate feature-selection method that takes the set of N $\{(SP_i, Delay_i)\}_{i=1}^{N}$ pairs and the parameter m as input and returns m features (flip-flops). These m features are selected using an m-best feature selection algorithm in which the correlation between each individual feature and the circuit delay is evaluated, and the m best features are retained as the output [105].

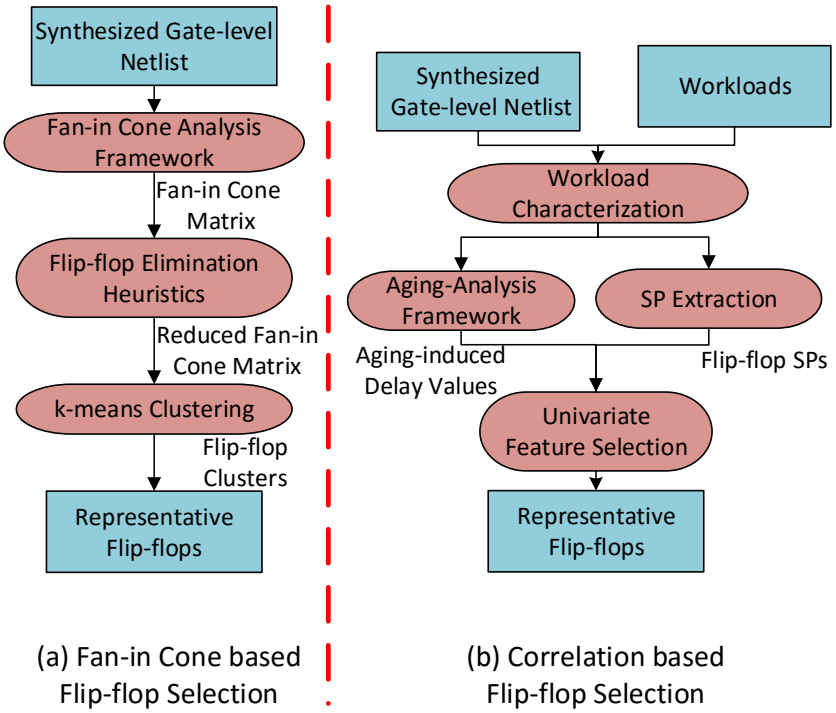


Figure 3.4: Overall flow of space-sampling techniques to identify representative flip-flops.

To illustrate the correlation-based flip-flop selection method, consider the hypothetical scenario explained in Section 3.5.2. The Pearson correlation coefficients [106], which take values in $[-1, 1]$, are used to represent the correlation between the SP of flip-flop i, $1 \leq i \leq 4$, and the delay value. The Pearson correlation coefficient between two sets of values, $(x_i)_{i=1}^{n}$ and $(y_i)_{i=1}^{n}$, is evaluated as follows:

$$r = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \Bigg/ \left(\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})} \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})} \right) \tag{3.2}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. We obtain the correlation coefficients for the five $(SP, Delay)$ pairs to be 0.2, 0.5, 0.8 and 0.4, respectively. Therefore, Flip-flop 2 (FF2) and Flip-flop 3 (FF3) are selected as being the most effective for prediction. Similarly, if our objective is to select the best three flip-flops, then Flip-flop 2, Flip-flop 3, and Flip-flop 4 are selected.

Fan-In Cone-Based Flip-Flop Selection

Although the correlation-based method provides high accuracy, it demands very high characterization effort, especially for larger designs with large number of flip-flops. The accuracy of the correlation-based method depends on the number of data samples, i.e., the number of workloads characterized. Moreover, the simulation of a sufficient number of workloads for correlation analysis is extremely time-consuming. The details of runtime will be discussed in Section 3.7.5. Hence, we propose a more practical method that selects flip-flops independent of workload characteristics.

In fan-in cone-based sampling, flip-flops are characterized by the gates in their fan-in cones. We assume that flip-flops with a larger number of logic paths through them hold more information in terms of workload since the flow of logic values during workload execution occurs

through flip-flops. Intuitively, the overlap in fan-in cones of flip-flops is assumed to represent the amount of redundancy in the information. If one flip-flop has a significant overlap in its fan-in cone with another flip-flop, only one of them needs to be selected for monitoring. For example, consider the scenario shown in Fig. 3.5. The gates present in the fan-in cones of flip-flops FF1, FF2 and FF3 are shown along with the overlap in their fan-in cones. Hence, from FF1, FF2 and FF3, only FF1 and FF2 are selected for monitoring.

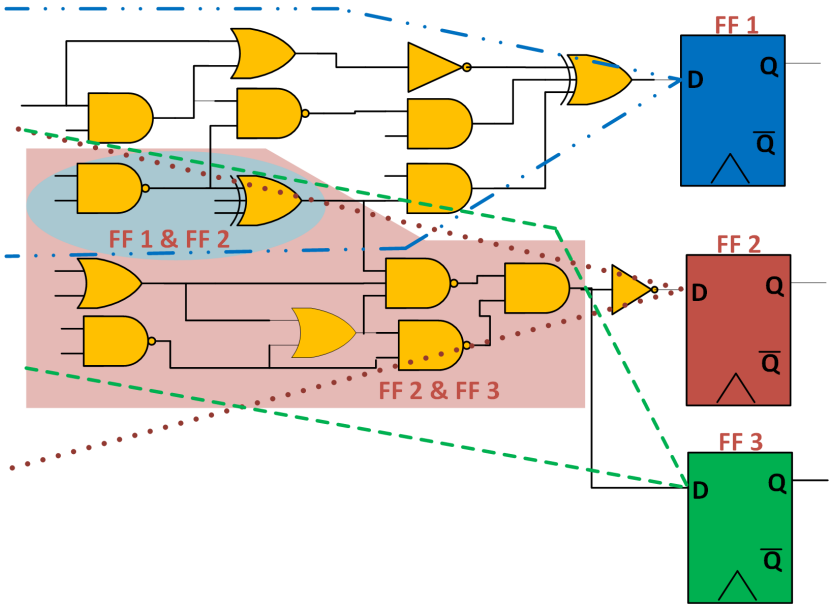


Figure 3.5: Fan-in cone characteristics of flip-flops.

The overall flow of fan-in cone-based flip-flop selection is shown in Fig. 3.4. The synthesized gate-level netlist of the design is required for fan-in cone analysis. A fan-in cone analysis framework captures the logic gates in the fan-in cone of each flip-flop. This involves a backward traversal of timing paths starting from the flip-flop input until we reach a primary input or another flip-flop's output. In this way, a fan-in cone matrix M is generated of size $n \times m$, where n and m refer to the number of flip-flops and gates in the design, respectively. Each element M_{ij} in the fan-in cone matrix can be either '0' or '1', representing the absence or presence of the gate G_j in the fan-in cone of flip-flop FF_i, respectively. For a realistic design, the size of the fan-in cone matrix would be huge for further computations. Hence, heuristics are used to eliminate a few rows or columns. This includes (i) elimination of flip-flops with only a few gates in their fan-in cone, and (ii) elimination of flip-flops with exactly the same gates in their fan-in cone. Thus we obtain a reduced fan-in cone matrix for further processing.

The set of flip-flops with unique logic gates and minimum overlap in their fan-in cones is identified using k-means clustering with the reduced fan-in cone matrix as the input. The k-means clustering method, an unsupervised machine learning technique for feature selection [107], can generate k clusters with each cluster containing flip-flops having similar features, i.e., significant overlap in their fan-in cones. The representative flip-flops for monitoring can be obtained by selecting one flip-flop from each such cluster.

Since we observed (from our results) that the effectiveness of the fan-in cone-based method is comparable to that of correlation-based method, we propose to use the former for flip-flop selection. Even though we are not advocating the use of correlation-based flip-flop selection, we utilize it to evaluate the effectiveness of fan-in cone-based selection.

In contrast to [108], the fan-in cone based algorithm in the proposed method does not consider the critically aged paths explicitly. The flip-flops are characterized by the presence

of unique gates in their fan-in cone, thereby unique information flow is expected the specific fan-in cones. We define the presence or absence of gates in the fan-in cone of flip-flops as the features. Subsequently, k-means clustering is carried out to reduce the large number of flip-flops into a few flip-flop clusters, based on their features. One flip-flop from a cluster can represent the workload information contained in that cluster. In other words, the clustering of flip-flops is carried out to eliminate redundant workload information, unlike in [108], where clustering is based on correlation between path delays or adjacency of nodes. Our objective is to find the minimum number of flip-flops having unique information to cover a maximum of gate states (signal probabilities).

3.5.4 Time Complexity of Flip-Flop Selection Methods

The execution time required for flip-flop selection methods is an important concern. The runtime for correlation-based flip-flop selection is much higher compared to fan-in cone-based selection due to the extensive workload characterization required to generate a sufficient number of training and testing samples.

We derive an expression for the runtime of fan-in cone-based flip-flop selection method in the worst-case. It is not possible to derive a simple closed-form expression for the runtime of correlation-based flip-flop selection method since it involves extensive workload characterization. Let the number of gates in the circuit be G, the number of flip-flops be F, and the number of signals (nets) be N. In order to generate the fan-in cone characteristics, a depth-first search is carried out starting from the flip-flop's input node until the traversal reaches the output of another flip-flop or a primary input. Therefore, for each flip-flop, the time required for finding the fan-in cone is $O(G + N)$. Since there are F flip-flops in our circuit, the runtime required for generating the fan-in cone matrix is $O(F(G + N))$.

The runtime of the heuristic used to eliminate flip-flops with the same gates in their fan-in cone is $O(F^2G)$ since the comparison of two vectors of length G takes $O(G)$ time. The runtime of the heuristic used to eliminate flip-flops whose fan-in cone has gates less than 5% of the number of gates in the largest fan-in cone is $O(F)$, if we assume that we can obtain the size of a vector in unit time. The runtime of k-means clustering algorithm is $O(nkdi)$ [109], where n is the number of input vectors (F in our problem), k is the number of clusters, d is the size of each vector (G in our problem), and i the number of iterations needed until convergence. The number of iterations until convergence is often small, and results only improve slightly after the first ten iterations. Therefore, the number of iterations until convergence is limited to ten in most implementations of k-means clustering algorithm. As a result, we can consider i as a constant in our analysis. We then obtain the runtime of the k-means clustering algorithm to select a fixed number of representative flip-flops as $O(FG)$. Since $G \gg F$ for large designs, we conclude that the runtime of the fan-in cone-based flip-flop selection method is $O(F(FG+N))$.

3.5.5 Time Sampling

The proposed aging-induced delay prediction method is based on the SPs of a small number of representative flip-flops. However, monitoring the representative flip-flops in every clock cycle to extract their SPs can be expensive in terms of total power consumption. This overhead in power consumption can be reduced by time-sampling the flip-flops instead of monitoring them in every clock cycle. In time-sampling, the flip-flops are sampled at regular intervals of time. For example, x% time-sampling implies that the flip-flops are sampled in x% clock cycles out of the total number of clock cycles in a workload segment. It is intuitive to see that the SP value of a flip-flop evaluated with a low sampling rate approximately represents the original SP value since SP is the average logic value of the flip-flop over the whole period

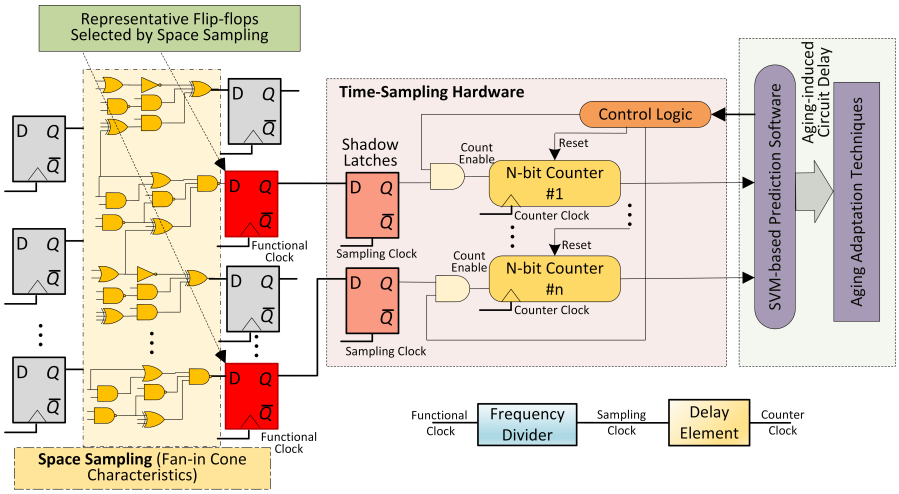


Figure 3.6: Illustration of flip-flop SP monitoring methodology.

under consideration.

Fig. 3.6 shows the additional hardware required to perform time sampling. A shadow latch and a counter are added to monitor the SP of each representative flip-flop. We also require a frequency divider to generate the clock for the shadow latches from the functional clock. The frequency divider is designed based on the time-sampling rate. For example, if we choose 1% time-sampling, then a divide-by-100 frequency divider is needed to generate the clock for the shadow latches. The clock generated for the shadow latches cannot be used for the counters as we have to take into account the setup time of the flip-flops used to design the counter. We therefore use a delayed clock for the counters.

Our results in Section 3.7 show that, even with very low sampling rates, significantly high prediction accuracy can be achieved.

3.6 Run-time Stress Monitoring

The SPs of the flip-flops relevant to the aging prediction are monitored during run-time using a synchronous-up counter, as shown in Fig. 3.6. The counter is enabled whenever the output of the flip-flop being monitored takes logic value '1'; therefore, the counter tracks the number of clock cycles for which the flip-flop output takes logic value '1'. The outputs of all the counters in the design are sampled at uniform time intervals, and these values are stored in a buffer. It should be clarified that the sampling frequency mentioned here is the frequency at which the counter values are sampled while time-sampling rate described in Section 3.5.5 specifies the frequency at which monitoring flip-flops are sampled. The sampling frequency to sample the counter values is decided at design time based on the characteristics of the representative workloads. The sampling frequency also determines the width of the counter that needs to be implemented. For example, if the output of a counter is sampled every 1000 clock cycles, then we need a counter that is at least 10 bits wide. Some additional control logic, as shown in Fig. 3.6, is also required to reset the counters after every sampling operation. The buffered outputs of all the counters are then transferred to the SVM-based prediction model, and then, this model translates these workload phase characteristics to aging-induced delay. The system can then take an appropriate protection measure based on the predicted aging trend.

There are two methods that can potentially be used to implement the prediction software. In the first method, the predictor is executed as a thread on any idle core on-chip. The role of this software thread is to collect flip-flop SP data from every core on-chip. These data,

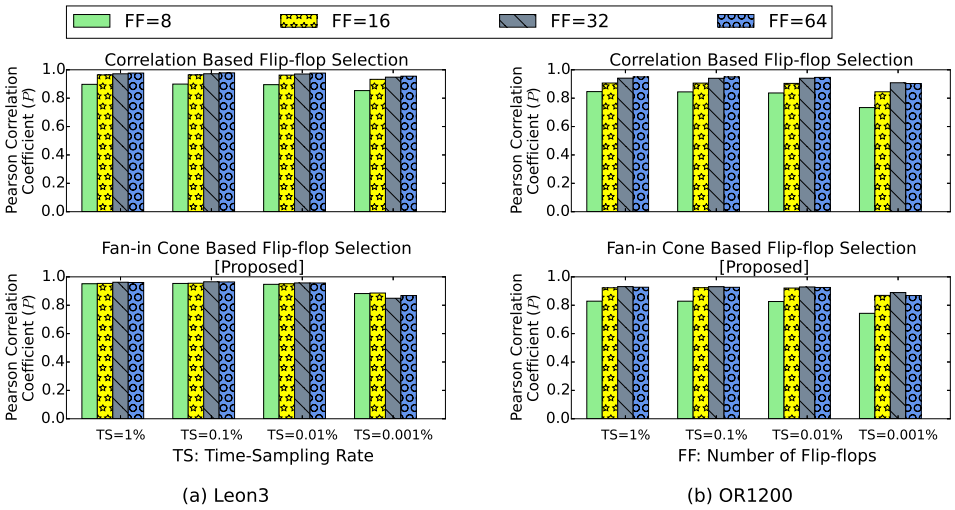


Figure 3.7: Results obtained using joint time-space sampling of flip-flops.

which are stored in a buffer on every core (Fig. 3.6), are transferred from one core to another based on a *handshake* mechanism. The core on which the thread is executing broadcasts a read signal to all the other cores. The core that is ready for this read operation sends back an acknowledgement (permission to read). Once the core executing the thread receives an acknowledgement from another core, it starts reading data from the buffer on that core and stores them in its buffer. In this way, data from all the cores are collected before the aging prediction process is started. This method of executing the prediction software does not require any additional hardware. However, an idle core may not always be available, in which case the system operation has to be interrupted to execute the prediction software. Moreover, migrating the predictor between different idle cores also involves overhead.

The other method is to execute the prediction software on a dedicated programmable microcontroller. The microcontroller communicates with every core on-chip to obtain flip-flop SP data. Therefore, it is necessary to define the interface between the microcontroller and the on-chip hardware. In [110], the communication between the energy management microcontroller and the processor core occurs through through the industry-standard Inter-Integrated Circuit (I^2C) interface. The microcontroller can read and write to the registers on every on-chip core through this interface. Therefore, the flip-flop SP data can be accessed by the microcontroller through the I^2C interface using buffer read operations. This method of implementation of the prediction software does not interrupt the processes running on any core, and therefore, has minimal performance overhead. However, additional hardware cost is incurred to implement the dedicated microcontroller on-chip.

3.7 Experimental Results

The effectiveness of the proposed approach was evaluated using two open-source embedded processors implemented at RTL, namely Leon3 and OpenRISC 1200 (OR1200), with running realistic application workloads. We implemented an SVM-based predictor, and the dependence of the prediction accuracy on the number of monitored flip-flops and on the time-sampling rate were studied for both processors. We also compared the accuracy of the two proposed flip-flop selection methods. Experiments were run on a 64 bit Linux machine with 16 GB of RAM and quad-core Intel Xeon processors running at 2.53 GHz.

3.7.1 Experimental Setup

OR1200 is five-stage pipeline embedded processor based on the 32-bit ORBIS32 instruction set architecture (ISA). Leon3 is a 32-bit processor based on the SPARC-V8 RISC ISA. The processor was synthesized using Synopsys Design Compiler with Nangate 45 nm library [111]. Our method is not limited to any particular technology node. The aging model used for evaluation can be easily updated to incorporate any newer technology node.

Six programs from MiBench, namely crc32, bitcount, qsort, susan, sha, stringsearch, and basicmath, were executed on the synthesized netlists of Leon3. Each workload was divided into several smaller workload segments of 10^3 clock cycles to collect the required number of workload samples for training and validation. Workload samples constituting the training set were used to construct a predictive model, and those constituting the test set were used to evaluate the accuracy of the predictive model. The aging-induced circuit delay values of Leon3 was computed using an NBTI based aging analysis framework. The SPs of the flip-flops in Leon3 were obtained for each workload phase from a SAIF file generated by performing a post-synthesis simulation in ModelSim. The SVM algorithms used to train and validate the aging predictor were implemented using Scikit-learn [34] with the built-in LibSVM software package.

For aging analysis, the entire logic core of the processor (including all combinational and sequential elements), except memory blocks (such as caches and register files), is considered to be a single circuit. Aging-aware STA is carried out on this core logic to extract aging-induced circuit delay. In our analysis, we have not included any analog blocks in the circuit. However, analog blocks can be included in the timing paths as black boxes if their propagation delays are defined.

3.7.2 SVM training and validation

The training data set for each processor benchmark consisted of 2000 SP vectors and their corresponding aging-induced delay values. A non-overlapping set of 2000 SP vectors and their corresponding aging-induced delay values were used to test the prediction model. The best values for the Radial Basis Function (RBF) kernel parameters used in the SVM model were determined using a five-fold cross-validation approach. In this approach, the training set was divided into five equal subsets and each subset was validated using a model trained on the remaining four subsets. A grid search was carried out on the parameters and the parameter values corresponding to the highest cross-validation accuracy were chosen. The prediction model was then trained using these best parameter values and the complete training set.

3.7.3 Evaluation of Prediction-Accuracy

We have selected representative flip-flops based on the two different space-sampling techniques as mentioned in Section 3.5.3. The metric used to evaluate the accuracy of prediction is the Pearson correlation coefficient.

Furthermore, we applied time-sampling techniques to reduce the frequency at which the selected representative flip-flops are sampled.

Joint Space-Time Sampling

Fig. 3.7 illustrates the results for joint space-time sampling on Leon3 and OR1200 processors. Joint time-space sampling involves selection of a set of flip-flops with space-sampling and further sampling of the logic values in these flip-flops at a pre-defined time-sampling rate. We

Table 3.2: Step by step correlation of SPs to aging-induced circuit delay.

		Pearson Correlation Coefficient (SP –>Delay)			
		LEON3		OR1200	
		Circuit Nodes	Flip-flops	Circuit Nodes	Flip-flops
Number of Features Selected	64	0.9893	0.9865	0.9889	0.9835
	32	0.9889	0.9846	0.9862	0.9813
	16	0.9846	0.9786	0.9719	0.9713
	8	0.9767	0.9565	0.9589	0.9323

obtained the results for four different time-sampling rates (1%, 0.1%, 0.01% and 0.001%) and four sets of flip-flops with different sizes (8, 16, 32, 64).

The prediction accuracy, evaluated as the Pearson Correlation Coefficient, under fan-in cone based flip-flop selection and correlation-based flip-flop selection is above 0.85 for sampling rates greater than 0.001% and flip-flop selection size greater than 8. The overall trend shows a significant reduction in accuracy when time-sampling rate reduces to 0.001% from 0.01%. Moreover, the prediction accuracy increases with an increase in number of selected flip-flops for correlation-based approach. However, it is not necessarily the case for the fan-in cone-based approach. The reason is that fan-in cone-based selection is a heuristic approach; therefore, it does not necessarily find optimal solutions. As a result, there are some cases in which the accuracy decreases even when more number of flip-flops are selected.

Since the correlation-based flip-flop selection method takes into account the nature of workload running on the chip, this method can provide better results for sufficiently large number of flip-flops. Hence, for a selection of 64 flip-flops, correlation-based flip-flop selection method provides a higher accuracy of 0.9784 compared to 0.9603 of fan-in cone based flip-flop selection method in Leon3 for a 1% time-sampling rate. However, as the number of flip-flops selected is reduced to 8, fan-in cone based sampling predicts with a better accuracy of 0.9514 compared to 0.8993 of the correlation-based method for Leon3. Hence, it is important to choose between these methods based on the feasibility and cost of flip-flop monitoring.

In summary, the key take-away messages from these results can be outlined as follows:

- A lower time-sampling rate can be adopted to reduce the power overhead of monitoring without compromising the accuracy of aging prediction.
- Hardware monitoring costs can be reduced by selectively monitoring a small number of flip-flops while maintaining high prediction accuracy.
- The proposed fan-in cone based flip-flop selection heuristic, with much lower runtime and complexity, is as effective as correlation-based flip-flop selection method having large runtime requirements.

Step-by-step correlation

The increase in threshold voltage of a transistor due to BTI depends on the SP at the gate terminal of that transistor. For accurate aging calculation, we require SP values for all internal nodes of the circuit netlist. Instead of monitoring the SP values of the internal nodes directly, we use flip-flop SPs to predict the aging trend. Table 3.2 compares prediction accuracies with two different observables (feature vectors): (1) SPs of circuit nodes, (2) SPs of flip-flops in the circuit. Note that we perform feature selection to eliminate insignificant features.

The results show insignificant variation in the Pearson Correlation Coefficient when the number of features exceeds 16. In other words, the aging-information loss while compacting circuit-node SPs to flip-flop SPs is insignificantly small.

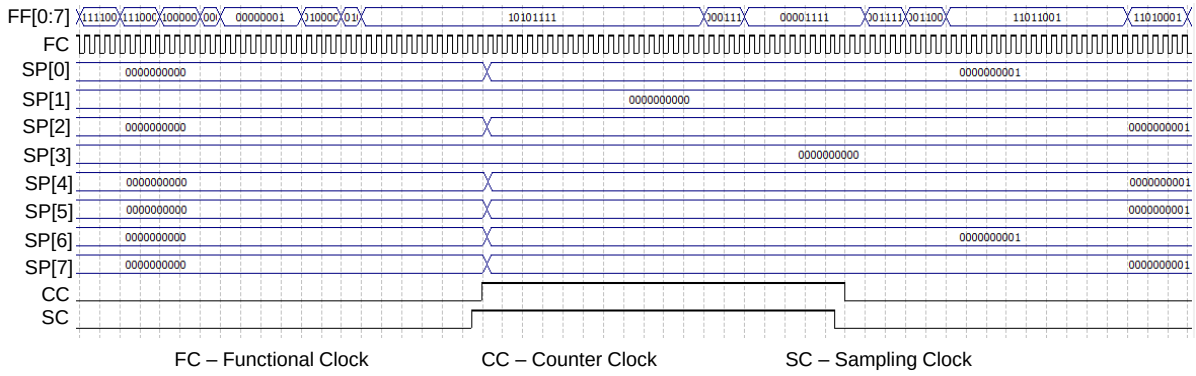


Figure 3.8: Timing simulation results for the time-sampling hardware.

3.7.4 Validation of Time-Sampling Hardware Design

The proposed method employs a time-sampling hardware as described in Section 3.5.5 that mainly consists of a shadow latch and a synchronous up-counter to monitor flip-flop SPs. We validated our design by carrying out timing simulations using its VHDL model. The model was implemented to sample eight flip-flops in 1% clock cycles out of the total number of clock cycles in a workload segment (assumed to be 10^5). Fig. 3.8 shows timing simulation results for this design. The clock signals for the shadow latches (sampling clock) and counters (counter clock) in our design are generated from the functional clock. The flip-flop values are sampled by the shadow latches when the sampling clock is high, and the counters are updated on the rising edge of the counter clock.

3.7.5 Overheads

For area and power estimations, the processor logic cores along with their corresponding memory blocks were considered. The overheads associated with the time-sampling hardware are extracted by adding it to the original processor netlist and the updated netlist is re-synthesized. The size of the counter is estimated based on the size of a workload segment, i.e., the number of clock cycles constituting one workload segment and the time-sampling rate chosen. For instance, if the workload segment is of size 10^6 with a time-sampling rate of 1%, a 14-bit counter is required.

Performance Overhead

Static Timing Analysis (STA) is carried out on the circuits using Synopsys Primetime to estimate the performance overhead. The results show that the additional hardware does not affect the maximum circuit delay, hence no performance overhead. This can be attributed to the re-optimization carried out by the synthesis tool to compensate for the additional load added to the output of the monitoring flip-flops. However, this re-optimization can increase the area overhead. In addition, the software thread executing the predictive model to compute aging can cause performance overhead. This overhead is dependent on the sampling frequency, i.e., how often the counter values are sampled to compute aging, and also on the runtime required for each computation. For instance, if the aging computation is performed every 10^6 cycles and if each computation takes 40 cycles, the performance overhead incurred by the software-thread execution is 0.004%.

Area and Power Overhead

The area overhead incurred due to the additional hardware for eight flip-flops under monitoring is 0.42% for Leon3 and 0.38% for OR1200 with 0.001% time-sampling. The size of the counters required to track SPs decreases with a reduction in the time-sampling rate. Moreover, a part of this area increase is due to the optimization carried out by the synthesis tool in the form of resizing of gates. Therefore, the actual area overhead will be lower than the values that we obtained.

For Leon3 and OR1200, the power overheads for a time-sampling rate of 0.001% were found to be as low as 0.07% and 0.12%, respectively. The dynamic power overhead of the monitoring hardware is significantly reduced with lower time-sampling rates, and hence, the overall power overhead becomes lower. In other words, the leakage power of the monitoring hardware has the major contribution to overall power overhead.

Overhead at Design Time

The correlation-based flip-flop selection required an estimated CPU time of 9.5 hours while fan-in cone based flip-flop selection required only 21 minutes. Therefore, the runtime for correlation-based flip-flop selection is much higher compared to fan-in cone-based selection. Since the effectiveness of the fan-in cone-based method is comparable to that of correlation-based method, we advocate the use of the former for flip-flop selection. In addition, the runtime in our setup to train the SVM model offline with the selected 64 features (flip-flops obtained through fan-in cone based flip-flop selection) was less than one minute for both processors.

3.8 Summary

We have proposed a method to predict the aging-induced delay based on flip-flop SPs. Unlike existing delay-monitoring schemes based on hardware sensors, our method imposes minimal area and power overhead since we rely on the SPs of a small number of flip-flops, which can be obtained by attaching simple counters to the flip-flop outputs. This method also makes it possible to capture fine-grained aging trends that can support proactive aging mitigation techniques. Simulation results for two embedded processors demonstrates that the proposed method can accurately predict workload-induced aging trends.

4 Static Aging Monitoring and Mitigation

4.1 Overview

The previous chapter discussed the reliability effect of Bias Temperature Instability (BTI) under normal workloads. The worst-case effects of BTI occur during specific workload phases in which flip-flops on a critical path do not switch their logic values for a long duration. These inactive flip-flops in the circuit experience accelerated workload-dependent static-BTI stress. The aging effect of static BTI for a few hours has been shown to be equivalent to one year of aging due to dynamic BTI, which can eventually cause circuit failure. The techniques available to mitigate static-BTI stress during standby mode of circuits are pessimistic, thereby limiting the performance of the circuit. To address this problem, we propose a runtime monitoring method to raise a flag when a timing-critical flip-flop experiences severe static-BTI stress. To reduce the monitoring costs, we select a small representative set of flip-flops offline based on workload-aware correlation analysis and these selected flip-flops are monitored online for static aging phases. Our experiments conducted on two processors show that less than 0.5% of the total number of flip-flops is required to be selected as representative flip-flops for S-BTI stress monitoring. We also propose a low-overhead mitigation scheme to relax critical flip-flops by executing a software subroutine that is designed to exercise critical flip-flops.

The rest of the chapter is organized as follows. Section 4.2 introduces the work in detail, presents a motivation and also lists the contributions. Section 4.3 overviews the aging models and related prior work. Section 4.4 describes the overall methodology underlying S-BTI estimation of flip-flops, selection of representative flip-flops and mitigation techniques. Section 4.5 explains the offline stages of S-BTI characterization, correlation and mitigation analysis. Section 4.6 covers online monitoring and mitigation of static aging. Experimental results are presented in Section 4.7. Finally, Section 4.8 summarizes the chapter.

4.2 Introduction, motivation and contributions

In nanoscale technology nodes, Bias Temperature Instability (BTI) is one of the predominant runtime reliability concerns in digital design [112, 113]. BTI, in effect, increases the magnitude of the threshold voltage of transistors in the circuit over time, which in turn increases the maximum circuit delay and causes timing failures in the system [114, 115]. The degradation depends on the workload profile of the flip-flops (FF) and logic gates in the circuit, which can only be determined during runtime. Hence, the effect of BTI under worst-case workload conditions should be evaluated and accounted for in the design in order to ensure resilient system operation.

BTI involves two phases of *stress* (threshold voltage (V_{th}) increases) and *recovery* (V_{th} decreases), based on the logic value at the gate terminal of the transistor [40, 116]. During lifetime operation of the circuit, the stress and recovery phases alternate, causing dynamic BTI (D-BTI), and the corresponding delay degradation in D-BTI-stress is a function of the duty cycle, i.e., the ratio of the stress-phase duration to the total runtime.

In contrast, long periods of inactivity in parts of the circuit can result in static BTI (S-BTI) stress [41]. This phenomenon can accelerate the aging effect on circuit delay degradation. The

delay degradation due to S-BTI stress of a few hours on logic circuits can be equivalent to D-BTI stress for one year [41]. The critical point of operation of a circuit is when the circuit enters a dynamic phase after a long duration of static-stress phase. A worst-case timing analysis should be carried out considering this critical point of operation. Techniques have therefore been presented to consider this effect on clock-gated circuits [117, 118].

In processors, depending on the workload under execution, there exists long durations of inactivity in a large number of flip-flops. Some of these flip-flops correspond to the most significant bits of address registers and exception-handling registers. In most workloads, these flip-flops are rarely accessed, hence they are inactive for extended periods of time. The workload phases inducing S-BTI stress for the flip-flops in a processor vary in both space and time. Hence, an offline-only estimate of the S-BTI stress in these flip-flops results in a pessimistic guardband thereby compromising circuit performance.

Most related work on aging stress estimation is based on D-BTI analysis [119–122]. The fact that logic elements (gates, latches and flip-flops) in a processor experiences static BTI stress during normal operation is supported by results presented in the literature [41, 117, 118, 123] and the effect on flip-flops in particular is analyzed in [124, 125]. Accelerated static aging scenarios are considered in [117, 118], but these methods are focused only on the stand-by mode of operation. The flip-flops that are vulnerable to *Static Aging Phases* (SAPs) due to specific workload scenarios are not considered in the analysis. On the other hand, accelerated aging behaviour of flip-flops under S-BTI stress has been pointed out in [124]. In this work, the flip-flops under S-BTI stress are characterized by their extremely low or high signal probability values and a selective replacement of vulnerable flip-flops at design time is proposed as a solution. However, for the same signal probability value, the longest SAP duration can vary significantly. Hence, this approach underestimates the runtime variation of signal probabilities. In general, all these methods ignore the workload dependency of S-BTI on timing-critical flip-flops. In other words, runtime monitoring of SAPs in critical flip-flops is essential to ensure reliable operation of the circuit.

Our aim in this work is to develop a low-overhead solution to identify workload phases as well as the specific timing-critical flip-flops experiencing S-BTI stress during those workload phases when the circuit is in functional mode. Our proposed method comprises of two phases: (1) an offline characterization of static-stress behaviour of flip-flops under workload execution; (2) online monitoring of selected representative flip-flops to enable BTI-aware runtime adaptation. During the offline phase, we identify critical flip-flops based on workload characterization and aging-aware timing analysis. To reduce the overhead of online monitoring, we perform offline correlation analysis to select a small subset of critical flip-flops as the representative set based on the correlation between their static-stress behavior under different workloads. These representative flip-flops will be monitored during runtime to generate an alarm signal when they experience long durations of S-BTI stress. Our experimental results on two processor designs show that only less than 0.5% of the total number of flip-flops in a processor is required to represent the flip-flops that are vulnerable to S-BTI.

For online monitoring, we propose a low-overhead monitoring hardware that is connected to the output of the representative flip-flops. The area and power overheads imposed by the additional hardware are less than 0.25% for both processor designs. The monitoring hardware checks for SAPs in representative flip-flops and marks them as being critical if no recovery in the form of a switching event occurs in a pre-defined interval.

To minimize the impact of static aging on critical flip-flops, we propose a technique similar to Software-based Self Test (SBST) [126, 127], which is a non-intrusive test method for processors. We design a subroutine, consisting of a sequence of instructions, to activate the switching of critical flip-flops. This subroutine is executed periodically, yet infrequently, when flip-flops enter the critical phase as defined by the monitoring methodology. The instruction

sequence is generated by a combination of two different techniques: (1) by exploiting a sequential Automatic Test-Pattern Generation (ATPG) technique, typically used for generating test patterns targeting different kinds of faults in a circuit-under-test, and (2) by selecting instructions based on behavioral analysis of critical flip-flops, similar to software-based functional testing. In the ATPG-based technique, we define stuck-at-faults at the targeted flip-flop outputs, which are defined as pseudo-primary outputs for the ATPG tool, and treat the generated test patterns as a sequence of instructions for the processor by restricting the patterns to be legal instruction formats. The flip-flops inaccessible by this technique are handled by a behavioral analysis method that finds trigger scenarios and corresponding instructions from the behavioral netlist of the design. Finally, the sequences of instructions selected by these two techniques are padded with another selected sequence of instructions to generate the subroutine. The instruction padding is carried out to nullify the effect of the execution of this subroutine on the system state. This subroutine execution relaxes static aging in flip-flops without leaving any footprint on the system state.

The corner-case scenarios of accelerated static aging are dependent on the nature of workload phases and our aim is to identify such problematic phases by monitoring a few selected flip-flops. These selected flip-flops are identified offline and found to be highly likely to have workload profiles that induce accelerated static aging. The selected flip-flops are surrogates for phases of inactivity and these phases are correlated across many such flip-flops. Hence, long phases of inactivity in a few flip-flops can represent similar long inactivity phases in other critical flip-flops. In terms of relaxation, we induce activity (transitions) in all critical flip-flops and not just the representative flip-flops.

4.3 Related Work

Most of the prior work in the literature considers only D-BTI to estimate the aging stress in flip-flops [119–122]. This approach is not suitable for processors since the aging stress of the flip-flops depends entirely on the workload profile. Hence, aging due to S-BTI needs to be considered as the worst-case scenario.

A different approach takes into account the static-aging scenarios by considering the system operation in both sleep mode and standby mode for worst-case analysis [41]. In this method, the identification of the timing-critical points in different modes of operation of the circuit has been carried out. These critical points occur when the circuit enters a dynamic stress phase after a long inactive phase, which has maximum BTI effect, thereby causing more timing violations. The end of the standby mode of a circuit has been identified as the critical moment for timing analysis, as the gate and path delays are maximum at that instant. However, apart from proposing a model to evaluate worst-case timing analysis in statically stressed circuit-paths, this work does not consider the case of individual flip-flops under static stress during runtime.

Several mitigation techniques for S-BTI have been introduced based on input-vector control and internal-node control [117, 118]. In [117], an internal-node control technique is proposed to minimize the effect of S-BTI on idle functional units in standby mode. However, a node control circuitry needs to be added to the output of each controlled gate, which introduces considerable overhead. In this work, only standby mode scenarios were considered. In addition, this method ignores the static aging effect in flip-flops. In [128, 129], the static aging effect under clock gating scenarios is considered. These approaches are orthogonal to our work and can be added alongside our technique.

A gate-replacement algorithm [123] together with optimal input-vector selection has been proposed in [118] to mitigate the S-BTI effect. In this work, a co-optimization has been carried

out to reduce BTI and leakage effect. However, this method considers D-BTI only during active mode of the circuit and S-BTI only during standby mode of the circuit. In other words, the S-BTI effect in flip-flops during active mode is ignored in the analysis.

In [124], a mitigation technique based on selective optimization of flip-flop standard cells operating under near-S-BTI stress is proposed. However, the longer durations of SAPs of flip-flops have not been considered. For example, a signal-probability (SP) based analysis would mark a flip-flop of SP of 0.5 under its workload profile as non-critical. However, this flip-flop might have a workload phase of several million clock cycles where it stores a logic value of 0 which does not affect its overall SP. Hence, an SP-based analysis of aging will miss such flip-flops or such workload phases that trigger worst-case scenarios. Moreover, this approach is pessimistic as it does not consider the impact of realistic workload on S-BTI and how it changes across different workloads. In addition, this method concentrates on the S-BTI effect on flip-flops and completely ignores the corresponding static stress on logic gates.

In existing techniques, standby or sleep mode is the only scenario considered for S-BTI stress analysis. However, we have found that a large number of flip-flops are under S-BTI stress during normal operation, i.e., not necessarily in the standby mode and their phases of static aging can significantly vary across different workloads. For instance, in our experiments, around 20% of the flip-flops in Leon3 were found to be critical due to S-BTI during active mode of operation. For example, an overflow-flag flip-flop in register-access stage of the integer pipeline has a static aging duration of 4.29 million clock cycles in Leon3 under MiBench workloads sha and susan_smooth. This is equivalent to around 42 minutes of D-BTI based aging degradation. Hence, an offline estimation of S-BTI stress and a solution based on an offline estimate will be pessimistic. Therefore, we propose a runtime monitoring technique to track SAPs of flip-flops and to adopt corresponding mitigation measures under critical conditions.

By considering the aging effects on flip-flops, the aging effects on combinational gates are implicitly considered. The flip-flop values can be considered as a representative of the bias on entire logic block at any given clock cycle in the normal operation of a circuit since the flip-flop states can be translated to the corresponding logic values of circuit nodes. The combinational logic gates in the forward cone of a flip-flop under static aging may also experience static aging effect. These logic gates release their stress while the associated flip-flop gets switched if the switching is not masked. Our proposed method can address S-BTI in both sequential elements (FFs) and combinational logic gates.

In addition, our analysis of static aging phases can be extended to consider Positive Bias Temperature Instability (PBTI) [130] as well. For example, SAPs can be redefined as any workload phase in which a transistor has an input of a constant logic 0 or logic 1. The mitigation technique remains the same as it requires only a single switching to relax static PBTI effect. Since the flip-flops are switched back to their original state, they undergo both $0 \rightarrow 1$ transition as well as $1 \rightarrow 0$ transition. The $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions at the input of an inverter relaxes the PMOS transistor and NMOS transistor, respectively.

4.4 Proposed Methodology

The SAP of a flip-flop in a circuit is defined as the time interval for which the logic value of the flip-flop remains constant. In processors, for instance, the higher significant bits of address registers, exception handling registers, etc. are rarely exercised by typical workloads. Hence, these registers/FFs store a constant value for an extended period of time. In a typical design, the flip-flops under long SAPs of millions of clock cycles can cause accelerated aging of the circuit paths. This increased critical path delay can result in a catastrophic failure of the

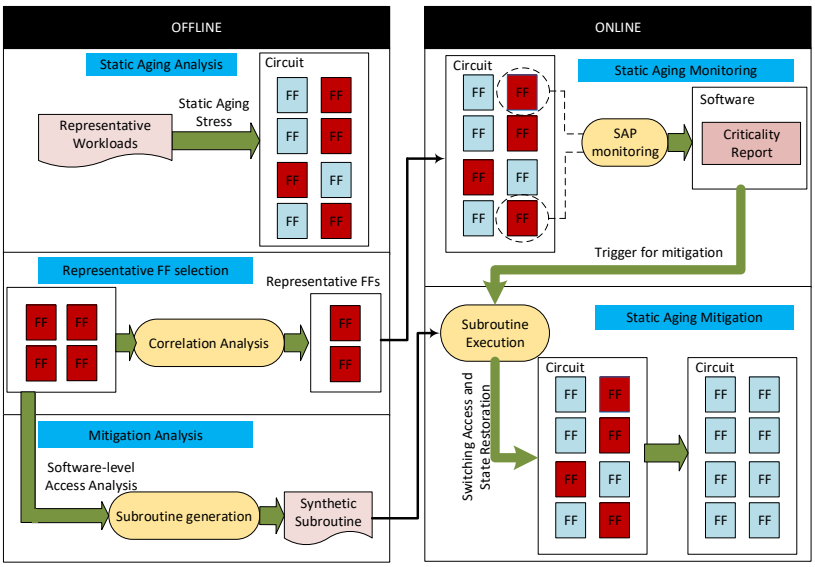


Figure 4.1: Overview of the proposed technique.

circuit when an infrequent switching event occurs subsequently. Hence, it is crucial to monitor these critical flip-flops during runtime to report their criticality as well as to adopt mitigation techniques to prevent a timing failure. In addition, the set of flip-flops experiencing static aging can be different for different workloads. For instance, we observed that the number of critical flip-flops in Fabscalar for six different workloads can vary from 5 to 376. Hence, providing countermeasures during the design of critical flip-flops will be both inadequate and expensive.

Overview of the proposed technique is shown in Fig. 4.1. The proposed technique identifies vulnerable flip-flops in a processor for different workloads based on the duration of static aging. We employ representative workloads that can capture the circuit behaviour in general, covering a wide field of applications. A correlation analysis based on the static-stress behaviour of the flip-flops is carried out to extract a small subset of representative flip-flops. These representative flip-flops are then monitored online for the time-point at which they enter an SAP. Once a flip-flop enters SAP, a criticality counter watches the length of this SAP for a threshold. If the flip-flop crosses the threshold, a flag is raised as an enabler for the mitigation methods. In other words, the criticality status of the representative flip-flops are reported to a software thread in regular intervals and the software thread initiates mitigation measures.

We propose mitigation in the form of execution of a software subroutine when the criticality of a flip-flop is reported. This software subroutine is designed in the offline stage by searching for suitable instructions in the ISA of a processor that can switch the flip-flops identified as critical. The main advantage of such a software-based approach is that the software can be updated to mitigate the aging effect of newer workloads not seen in the offline phase. For example, a workload, which is not considered in the offline characterization stage, could impose severe static aging on a flip-flop marked as non-critical in the offline stage. This static aging scenario can be better handled by the mitigation scheme through a software update.

The steps involved in the proposed framework can be divided into two phases: (1) offline phase (Section 4.5.1), which involves offline characterization, static aging correlation of flip-flops and mitigation analysis; (2) online phase that includes online monitoring of critical flip-flops for static aging and subroutine-based mitigation of static aging (Section 4.6).

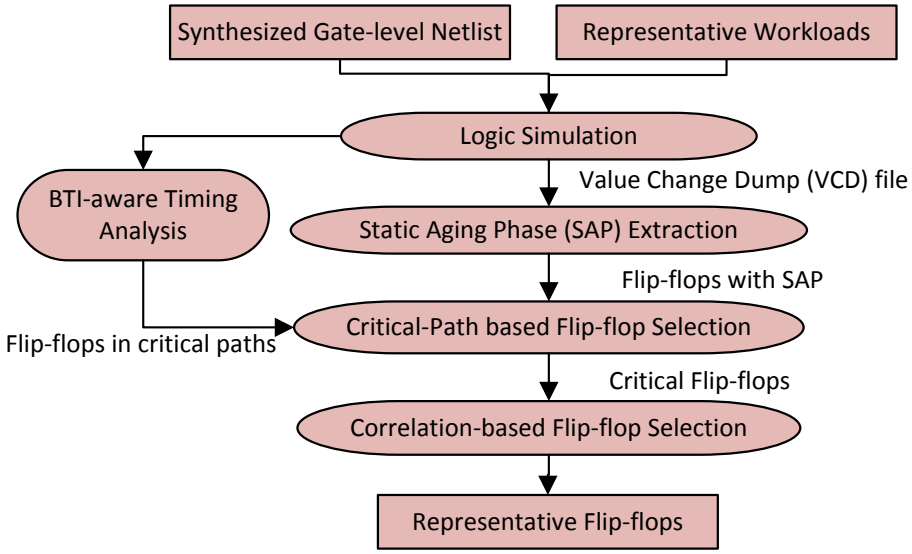


Figure 4.2: Steps involved in the offline characterization phase.

4.5 Offline Phase

The offline phase of the proposed technique involves (1) a characterization and correlation analysis of static aging in flip-flops under different representative workloads and, (2) the generation of a subroutine that on execution can mitigate static aging during functional operation of a circuit. The two stages are described in detail in Section 4.5.1 and Section 4.5.2.

4.5.1 Offline characterization and correlation analysis

In the offline characterization step, several workloads are simulated to identify the flip-flops that possess critical static aging behaviour. The different steps involved in offline characterization are illustrated in Fig. 4.2.

The first step is the execution of several representative workloads on the synthesized gate-level netlist of the design. The value change dump (VCD) files generated in this process are analyzed to extract SAPs of different flip-flops. For example, the clock cycle index corresponding to each switching event in a flip-flop is extracted. If the interval between two such switching events in the flip-flop is above the threshold (for instance, 1 million clock cycles of constant value), that flip-flop is considered to have an SAP in that interval. The SAPs are extracted under each workload for a flip-flop. The flip-flops with at least one SAP of duration above a threshold are identified as the first set of candidates (S_{sap}) for further analysis. A BTI-aware timing analysis is carried out in parallel to extract another set of flip-flops (S_{tc}) that belongs to critical or near-critical paths. An SAP in these flip-flops can potentially lead to a circuit failure, hence these flip-flops are significant for our analysis. The flip-flops on near-critical paths can become timing-critical due to aging effect, hence those flip-flops are also included in our analysis. In the next step, the intersection of these two sets of flip-flops, that belong to critical or near-critical paths and possess SAPs in workload characterization, are identified as critical flip-flops ($S_{crit} = S_{sap} \cap S_{tc}$). The overhead of monitoring all these critical flip-flops during runtime can be considerably large. Hence, we propose a method to select a group of representative flip-flops for monitoring. The proposed method analyzes the correlation in SAPs of critical flip-flops to extract the required representative flip-flops. We explain critical-path based flip-flop selection and correlation-based representative flip-flop selection in the subsequent sections.

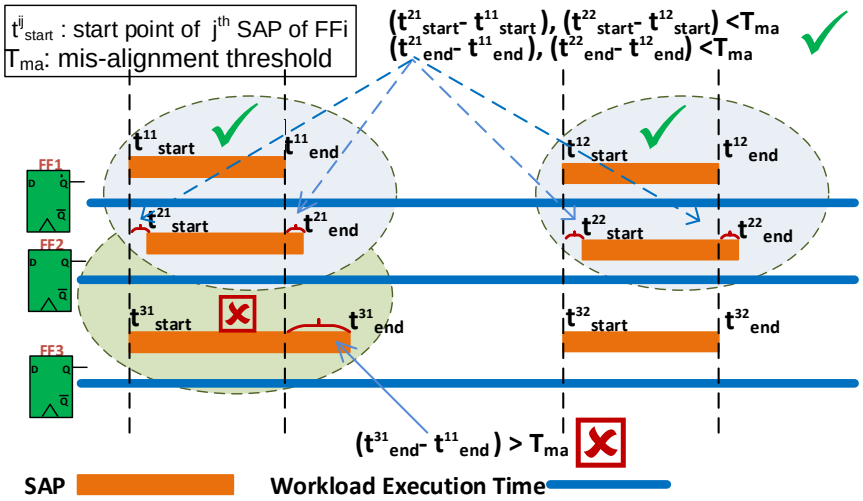


Figure 4.3: Correlation analysis of flip-flops based on the overlap of concurrent SAPs.

Critical-path based flip-flop selection

The critical-path based flip-flop selection involves a D-BTI-aware static timing analysis (STA) to extract critical and near-critical paths. D-BTI induced delay degradation of a logic gate depends on the signal probability at its input nodes. The signal probability (SP) values at input nodes for all gates in the design are extracted by executing representative workloads. These SPs are translated to aging-induced delay values using D-BTI models [116]. An STA is carried out based on the computed gate delays to extract post-aging critical paths of the circuit. Flip-flops from critical or near-critical paths only are considered for monitoring. For any such path, both the start-point flip-flop and the end-point flip-flop are considered. This is because, the static aging-induced delay can affect the clock to output delay ($T_{clk->Q}$) and set-up time (T_{su}) for the start-point flip-flop and end-point flip-flop, respectively [124]. The selected set of flip-flops are then analyzed for correlation in the next phase.

Correlation-based representative flip-flop selection

The SAPs of different flip-flops located on critical paths can be significantly correlated. For example, the most significant bits (MSBs) of address registers share the same SAPs since these bits are exercised only when a specific set of instructions is executed. We exploit the correlation between SAPs of flip-flops to reduce the overhead of monitoring. The key idea is to select a set of representative flip-flops that can represent the SAPs of all the critical flip-flops based on a correlation analysis. The workloads used for the offline correlation analysis should be carefully chosen to represent the online behavior of the chip.

We define the start and end points of an SAP as the clock cycles at which the flip-flop enters the SAP (t_{start}) and the clock cycle at which it exits the SAP (t_{end}) by a switching event, respectively. A correlation coefficient cannot be simply used between two flip-flops, since a single SAP of significant misalignment in start or end point for these two flip-flops can lead to a false-positive prediction of static aging during runtime.

Two flip-flops, FF_1 and FF_2, are defined to be correlated if their SAPs satisfy two conditions: (1) all SAPs should be above a minimum critical duration, T_{sad_min} defined based on the aging model ($t_{end}^{FF_1} - t_{start}^{FF_1}, t_{end}^{FF_2} - t_{start}^{FF_2} > T_{sad\_min}$), and (2) for each concurrent SAP pair in FF_1 and FF_2, the start points ($t_{start}^{FF_1}$, $t_{start}^{FF_2}$) and end points ($t_{end}^{FF_1}$, $t_{end}^{FF_2}$) should be the same or be different by no more than a misalignment threshold T_{ma}, i.e.,

$\left|t_{end}^{FF_1} - t_{end}^{FF_2}\right|, \left|t_{start}^{FF_1} - t_{start}^{FF_2}\right| < T_{ma}$. For instance, T_{ma} can be considered as 1% of T_{sad_min}.

The correlation between flip-flops is illustrated in Fig. 4.3. Out of the three pairs of flip-flops, the only correlated pair is FF1 and FF2 based on the above conditions. In summary, different sets of correlated flip-flops by the correlation analysis can be selected. Note that the correlation defined here is binary and can only admit two possibilities: (1) perfect correlation, (2) no correlation.

For each workload, the invalid flip-flop SAPs are eliminated based on SAP duration. Correlation analysis is carried out between all concurrent SAPs of each pair of flip-flops. If correlated, the pair of flip-flops is added to a correlated group. This process is repeated until all flip-flop pairs are examined during correlation analysis. Thus, the correlated flip-flop groups are obtained for each workload under consideration. Since two flip-flop groups from two different workloads can include a common flip-flop, an intelligent choice of representative for a group, as detailed in the following, can reduce the total number of representative flip-flops.

The selection of the minimum number of representative flip-flops in this case can be formulated as a minimal hitting-set problem [131]. This problem is known to be NP-complete, hence heuristic solutions are necessary. In our case, let the correlated flip-flop groups under all workloads be represented as sets S_1, S_2, ..., S_n. We construct a representative flip-flop set RS by adding one element selected from each correlated flip-flop set. The objective is to minimize the size of RS by a suitable choice of elements from each set.

We adopt a greedy approach to find the minimal hitting set. The flip-flop belonging to the largest number of flip-flop groups has been selected as the element of choice at each iteration. Before each subsequent iteration, all the flip-flop groups represented by already-selected flip-flops are removed from further consideration. This process is repeated until all the flip-flop groups are removed. The set of selected flip-flops in each iteration constitutes the representative flip-flop set.

We derived an expression for the time complexity of the pairwise correlation analysis. Let the number of timing-critical F in a circuit be p . The maximum number of SAPs in any flip-flop can be considered as a finite constant for a given duration of a workload. The runtime to delete all SAPs below T_{sadmin} is $O(p)$. The pairwise correlation across all flip-flops to determine correlated flip-flops has a complexity of $O(p^2)$. Hence, the overall time-complexity to generate the correlated flip-flop groups is $O(p^2 + p)$, i.e., $O(p^2)$.

To calculate the time complexity of the greedy approach for minimal hitting-set selection, let the number of flip-flops in an flip-flop group be n and the number of flip-flop groups for a workload be m. The runtime to select the flip-flop belonging to maximum number of flip-flop groups is $O(mn)$. The flip-flop groups that can be represented by the selected flip-flop can be eliminated in $O(m)$. This process is repeated m times to generate the minimal hitting set. Hence, the time complexity of this greedy approach to select representative flip-flops is $O(m(nm + m))$, i.e., $O(m^2 n)$.

4.5.2 Offline static aging mitigation analysis

The next step after the identification of critical flip-flops and representative critical flip-flops is to mitigate the effect of static aging. This is achieved by developing a software subroutine that can access the critical flip-flops to switch their value, and thereby relax the effect of static aging.

Flip-flop switching can also be achieved by adding extra combinational logic for each critical flip-flop. However, the number of critical flip-flops can be very large (as shown later in Table 4.3) for a design, making a hardware solution very expensive. Hence, a software based solution is desirable. Moreover, a software-based solution provides the flexibility to modify

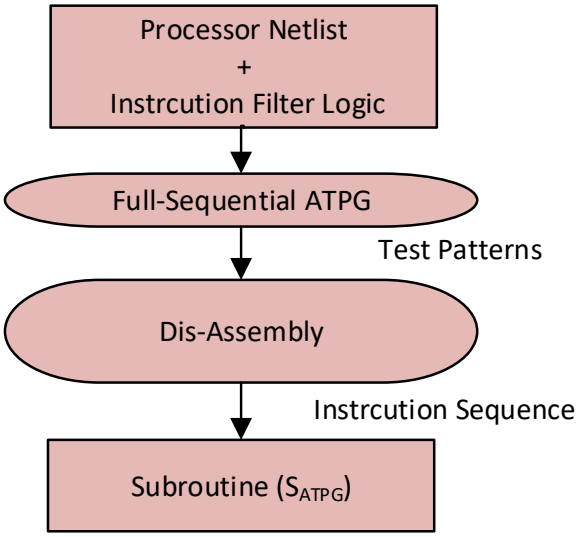


Figure 4.4: Flow chart showing the steps involved in ATPG-based subroutine generation

the mitigation approach online and thereby, to tackle the static aging effect of an anomalous workload that was not considered in the offline characterization phase

Our proposed software subroutine can be designed offline by exploring cross-layer access mechanisms that can switch the critical flip-flops from the software layer. The cross-layer access is enabled implicitly without any microarchitectural-level or gate-level modification. The subroutine should also be constrained in such a way that its execution should not leave any footprint on the architectural state of the processor. The designed subroutine is stored on-chip and executed on receiving a trigger from the static aging monitoring hardware.

One of the most important characteristics of our mitigation scheme is that it is not hard-wired. The subroutines are reconfigurable on-the-fly. Hence, our approach has the flexibility to revisit the list of critical flip-flops online and if any of the omitted flip-flops are found to be under accelerated static aging, the subroutines can be modified to include specific instructions that can exercise these flip-flops and relax their aging-induced degradation.

We propose a three-stage process to generate a software subroutine that can be executed to relax flip-flops under critical static aging. The first stage involves an ATPG-based detection of suitable instruction sequences that can exercise the critical flip-flops. While this technique can be fully automated and the number of patterns (instructions) to toggle those critical flip-flops can be optimized, it has limited coverage as it uses the sequential ATPG features. Therefore, those flip-flops that cannot be exercised by this technique, can be handled in a second stage that involves a semi-manual selection of instructions based on the functionality of the critical flip-flops under consideration. These generated subroutines can potentially alter the microarchitectural state of the processor and hence, it is important to add additional instructions that achieves state restoration. In this regard, the third stage of instruction padding is added in the process flow to select and add instructions that can switch back the flip-flop values back to their original state. The three stages are described in detail in the subsequent sections.

ATPG-based toggling instruction sequence

In post-silicon testing of digital integrated circuits, test patterns generated in pre-silicon stage, are used to test for various modeled faults, such as stuck-at-faults, transition faults, and path

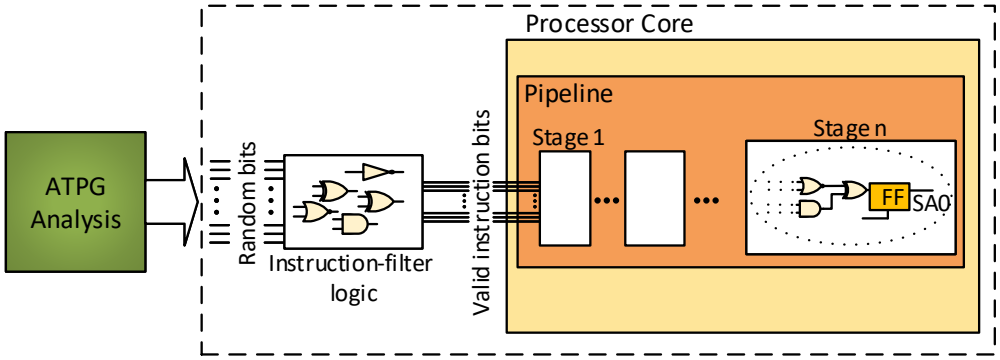


Figure 4.5: Illustration of ATPG-based subroutine generation settings for mitigating static aging.

delay faults. This automatic test pattern generation (ATPG) process can be exploited to generate instruction sequences that mitigate critical static aging of flip-flops. Our aim is to find the instruction sequence that on its execution, causes switching of the critical flip-flops. To find such an instruction sequence, ATPG should be carried out with the critical flip-flop output nodes defined as stuck-at-fault targets while they are defined as pseudo primary outputs (no further propagation is necessary) and the input stimuli limited to those corresponding to the instruction bits in a processor core (legal opcodes and instructions). In this setting, the input patterns that can switch the flip-flop can be extracted and a dis-assembler is used to convert the test patterns to corresponding instruction sequences.

The complete instruction-sequence (subroutine) generation settings for a typical processor design is illustrated in Fig. 4.5 and the corresponding process flow is shown in Fig. 4.4. The synthesized and flattened gate-level netlist of a processor is used for sequential ATPG analysis. This netlist is augmented with an instruction-filter logic to filter out the illegal instruction patterns that can be generated in the analysis. The instruction-filter logic converts any illegal instruction pattern to a pattern corresponding to *nop* instruction. The critical flip-flop outputs, identified from the offline characterization stage, are added as target stuck-at-faults to the fault list. The primary inputs excluding the instruction bits are marked as X, such that these are not used in pattern generation.

A typical ATPG process for a given target fault involves three phases:

- initialization of the circuit to a defined state.
- activation of the fault target.
- propagation of the fault effect to the primary output.

In our case, we are only interested in the initialization and activation phases. Hence, we define the target flip-flop outputs as observable nodes or primary output nodes to obsolete the fault propagation phase.

The ATPG for the processor design needs to be executed in full-sequential mode with no scan cells inserted, such that the controllability is limited to primary inputs corresponding to the instruction word. However, some of the instruction words corresponding to the test sequence can remain unimplemented or undefined in the instruction set architecture (ISA) of the processor. To filter such patterns, an instruction-filter logic is attached at the input of the processor netlist as shown in Fig. 4.5 and the combined netlist is used for test pattern generation. The patterns observed at the output of the instruction-filter logic is collected and converted to the corresponding instructions in the ISA of the processor using a dis-assembler. This sequence of instructions forms the subroutine to be used for static aging mitigation. Since we utilize standard ATPG tools to generate test patterns, several optimizations are possible in this process of subroutine generation. For example, the number of instructions in

the subroutine can be significantly reduced by employing ATPG optimizations to reduce the number of test vectors. In addition, a single test pattern that targets multiple critical flip-flops can be generated by executing ATPG for a set of stuck-at-fault targets.

Although ATPG-based subroutine generation and execution is a low-overhead solution to mitigate static aging, there are a few limitations for this approach. First, it might be extremely difficult to find patterns that can switch flip-flops that are dependent on a specific state of the system, for example, a flip-flop indicating overflow. In addition, since we employ sequential ATPG, the probability to find a pattern (coverage) is lower than that of a full-scan mode. Moreover, the instructions that can be included in the subroutine are also limited. For example, a *jump* or a conditional branch instruction to an arbitrary address cannot be included in the subroutine, as it disrupts the sequential execution of subroutine. Hence, the corresponding test vectors should be filtered by the instruction-filter logic. This filtering reduces the probability to find a successful fault-activation pattern using ATPG. However, an alternative approach to include the branching instructions would be to execute the subroutine in a test state similar to Microprocessor Hardware Self-Test (MIHST)-based approach in [132]. In this manner, the processor executes instructions provided by MIHST unit, but does not control the execution flow.

Functionality-based instruction sequence

The flip-flops that could not be switched by the sequential ATPG-based approach are analyzed to find their functionality in a processor logic and then we manually combine instructions that can switch the state by creating specific scenarios. For example, an overflow flip-flop could be switched by an *add* operation with suitable operands. A list of critical flip-flops and their switching scenarios are described in Table. 4.1. For example, the special register bit corresponding to a window invalid mask could be switched by executing a special instruction called *Write Window Invalid Mask (WRWIM)* in SPARC architecture [133].

The flow-chart for the functionality-based subroutine generation is explained in Fig. 4.6. The critical flip-flops in the flattened processor netlist that can be switched by S_{ATPG} are disregarded in the generation of functionality-based subroutine (S_{func}). The remaining critical flip-flop instances in the post-synthesis processor netlist are analyzed to extract the hierarchical path from the top module. For example, consider the flip-flop $c0mmu\_dcache0\_r\_reg\_SIZE\_\_0\_$ from the synthesized netlist of Leon3 processor. This flip-flop belongs to a signal in the integer pipeline unit storing the size of data word to be handled in a *store*/*load* instruction. For this information, the signals in the pre-synthesis behavioral description corresponding to the flip-flop instances are extracted. The next step is to find the trigger conditions of these signals from their corresponding behavioral code. For the flip-flop instance under consideration, the trigger condition corresponds to a match with a subopcode of *load* instruction. To enable these trigger conditions, proper instruction sequences need to be found from an ISA (instruction set architecture) analysis. In this example, the instruction to make the trigger condition *true* is a *load* instruction, i.e., *ldd* [%$r4$], %$r5$. The combined instruction sequence for all critical flip-flops forms a sub-routine (S_{func}) that is stored on-chip along with S_{ATPG}. Examples of such subroutines for both methods are shown in Table 4.2.

Instruction Padding

The execution of the subroutines (S_{ATPG} and S_{func}) during normal operation of the processor can alter the processor state. Hence, an instruction padding step is followed; i.e., a few instructions are added to the subroutines, such that the system state is restored. These additional instructions can only be selected semi-manually based on the foot-print analysis of

Table 4.1: Functionality based flip-flop switching

Flip-flop	Switching Scenario
Special register bit for *trap* type	Divison by zero
Register bit for *load* type	*load* of a double word
State bit in division unit	Special division operation
Window invalid mask (WIM) bit	Execute instruction WRWIM (write WIM)
Annul bit in exception stage	Instruction-induced exception trap
Interrupt request enable	Write process status register (WRPSR)
Current Window Pointer (CWP) bit	Write process status register (WRPSR)
Divison unit flag	Special division operation

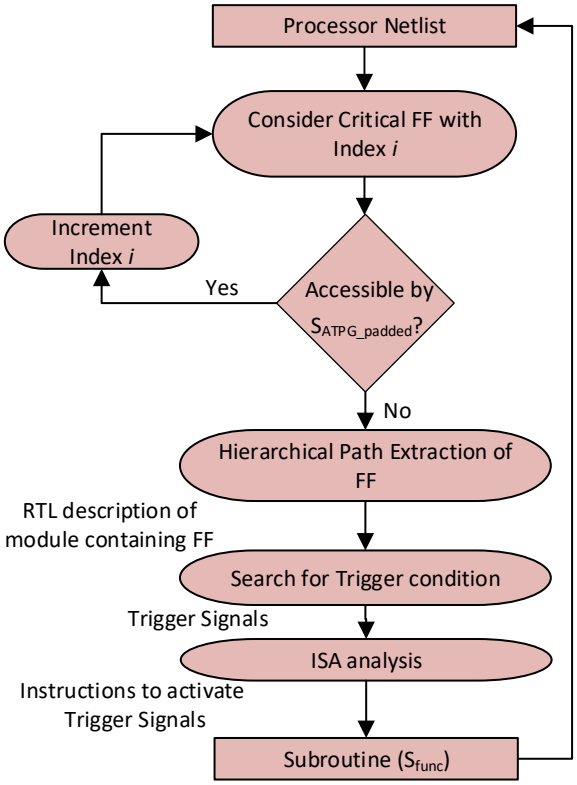


Figure 4.6: Flow chart showing the steps involved in functionality-based subroutine generation

Table 4.2: Examples for subroutine generation

ATPG based subroutine	*st* %*r*1, [*addr*_1]; *st* %*r*2, [*addr*_2]; *st* %*l*5, [*addr*_3]; *st* %*o*4, [*addr*_4]; *orcc* %*o*0, `-0x2ae`, %*l*5; *sdiv* %*o*4, `0xb55`, %*g*0; *mulscc* %*r*2, %*r*2, %*r*1; *ld* [*addr*_1], %*r*1; *ld* [*addr*_2], %*r*2; *ld* [*addr*_3], %*l*5; *ld* [*addr*_4], %*o*4;
Functionality based subroutine (Register bit for *load* type)	*st* %*r*5, [*addr*] ; *ldd* [%*r*4], *r*5; *ldd* [*addr*], %*r*5

the initial instruction sequence. For example, if the content of a register would get affected by the execution of ATPG-based subroutine (S_{ATPG}), then a few instructions are added at the beginning of this subroutine to backup the register content in memory (*store* instruction) and at the end of the subroutine to restore the register content to its earlier value (*load* instruction). In short, the generated subroutines are padded with selected instructions such that the combined sequence leaves no footprint on the architectural state of the processor.

4.6 Online Phase

The online phase of our proposed technique involves (1) monitoring of representative critical flip-flops for static aging phases, and (2) mitigation of static aging using synthetic subroutine execution. The two stages are explained in detail in Section 4.6.1 and Section 4.6.2.

4.6.1 Online Monitoring of Static Aging

The representative flip-flops selected in the offline phase are monitored online to identify the critical workload phases that can induce accelerated static aging of the circuit. Our proposed method deploys a monitoring hardware that generates a criticality report and sends the report to a software at regular intervals. The software keeps track of the flip-flops which continuously lie in critical state in each report. If the flip-flops are observed to be in a SAP above a threshold duration, an activation signal for mitigation measures is generated by the software. The mitigation measures are carried out to relax the accelerated aging occurring in these most critical flip-flops. This is achieved by enforcing a switching event on critical flip-flops.

Monitoring Hardware

The monitoring hardware used to track static aging in flip-flops is illustrated Fig. 4.7. The representative flip-flops selected in the offline phase are connected to the monitoring hardware. A switching event in a representative flip-flop can be translated as the recovery phase of the corresponding flip-flop group. Hence, we track switching events in these flip-flops to report the recovery event to the software. To achieve this, the representative flip-flops are observed by a switching-event detector. This detection circuit generates a pulse at its output when a logic transition occurs in the corresponding flip-flop. The output of the switching-event detector is encoded using a priority encoder. In any given clock cycle, a non-default value at the output of the priority encoder indicates the index of the flip-flop that switches its state in that particular clock cycle. If two representative flip-flops switch their states in the same clock cycle, priority encoder ensures a valid output by encoding either of these switching events. Since the representative flip-flops are least correlated among themselves, the probability of a switching event at the end of a concurrent SAP pair occurring simultaneously in any two of them is almost zero. Hence, the encoding of switching events can be carried out with high accuracy.

We maintain a criticality word in a critical-flag register to represent the recovery/aging state of the corresponding representative flip-flop. All representative flip-flops are marked as critical at the beginning of each monitoring interval. The start and end of each such interval are determined using a timer. For example, the bits of the critical-flag register are all set to '1' in the beginning of a monitoring interval. For each switching event that occurs in any representative flip-flop, corresponding flip-flop index is generated at the output of the priority encoder. The flag bit in the critical-flag register corresponding to this index is reset to '0' to mark the recovery state of that representative flip-flop. This process is repeated until either (1) all the flip-flops are marked to be recovered or (2) the monitoring interval ends. The

critical-flag register content is sent to the software at the end of each such monitoring interval. The software maintains a criticality Look-Up Table (CLUT) that marks the flip-flop index along with the criticality grade. If a flip-flop is marked as critical for two consecutive intervals, the criticality grade is incremented. Appropriate mitigation measures are adopted when the criticality grade of a flip-flop exceeds a pre-defined threshold.

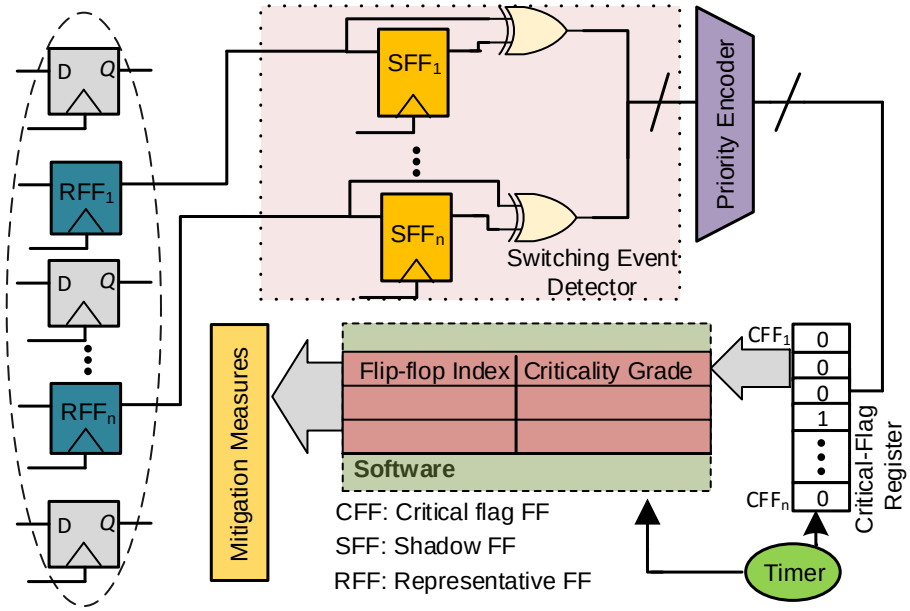


Figure 4.7: Illustration of the online static aging monitoring hardware.

The status of the flip-flops in the detection circuit are tracked in the criticality look-up table (CLUT) and hence, the state of these flip-flops can be switched after a critical static aging phase (SAP). Moreover, the flip-flops in the monitoring hardware do not belong to critical or near-critical path. Hence, these flip-flops can make a recovery before they reach a worst-case scenario.

The area overhead of the monitoring hardware can be reduced by replacing the priority encoder with AND gates at the output of the switching event detectors (SED_{out}). The critical-flag register (CFF) should switch to zero when a switching event is detected and should not respond to any further switching event in a monitoring interval, i.e, the AND gate is connected as $CFF_{in} = CFF_{out} \& SED_{out}$.

In modern microprocessors, software threads can access machine specific registers (MSRs) in less than a microsecond [134]. The monitoring interval of our proposed method can be in the order of milliseconds (corresponding to an SAP duration in the order of million clock cycles) for a 1 GHz processor, hence, the critical-flag register can be accessed by the software with almost no performance overhead.

The system-level support is required in the case of communication between monitoring hardware and the software which tracks static aging status of critical flip-flops. The status can be read in uniform intervals by reading a single critical-flag register as shown in Fig. 4.7. For example, in ARM-based designs, a custom register can be easily created and a memory map block or a co-process access interface can be implemented for read and write [135].

4.6.2 Online Mitigation of Static Aging

The software associated with static aging monitoring setup, as explained in Section 4.6.1, maintains a criticality grade for each critical flip-flop. If any of the monitoring flip-flops crosses a pre-defined threshold, a trigger is activated for mitigation and the subroutine, designed

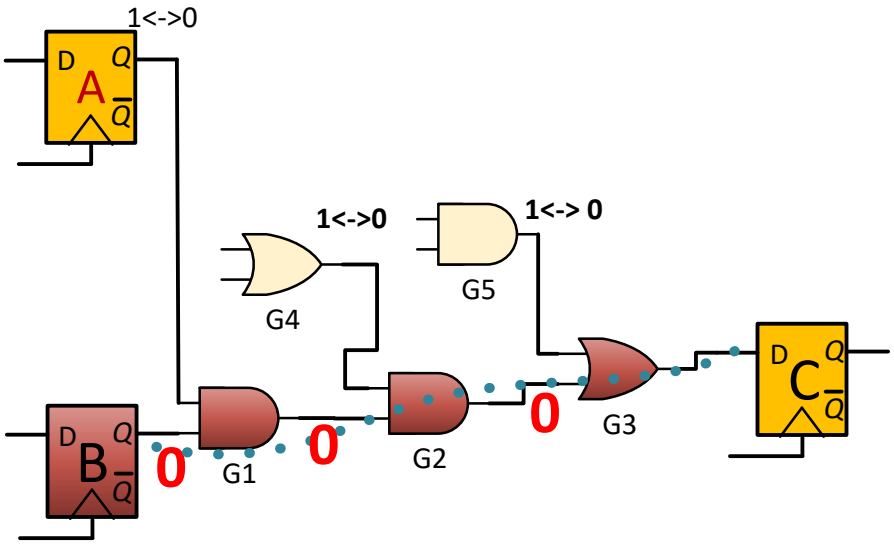


Figure 4.8: Illustration of switching event propagation from flip-flops under static aging to the logic gates under static aging in their forward fanout cone.

offline for mitigation, is executed. During the execution of this subroutine, the static aging critical flip-flops switch their values and relax the accelerated aging effect. The subroutine is constrained not to leave any residual footprint on the functional state of the processor. Hence, the switched flip-flop states are reversed with consequent switchings such that the system state is restored.

In addition to critical flip-flops, most logic gates in the forward fan-in cone of these flip-flops can also get recovered from S-BTI stress by the execution of the subroutine. In other words, the static aging effect on the logic gates can be intuitively correlated to the static aging of one or more flip-flops in their input cone. Hence, in most cases, a switching event in these flip-flops is sufficient to generate a switching in these gates and thereby, a relaxation in their static aging.

For example, consider the circuit shown in the Fig. 4.8. The flip-flop B is considered to be under static aging storing a logic value of 0. The value is propagated forward in its fan-out cone accelerating the aging based degradation of transistors in gates G1, G2 and G3. The inputs of these gates and of the corresponding transistors in these gates will undergo switching (from 0 to 1 and also from 1 to 0) when the flip-flop B switches from 0 to 1 for a short duration and then, 1 to 0 (to restore its original state). These switchings leads to a relaxation of the accelerated static aging in these gates.

Note that, here G4, G5 and flip-flop A are shown to have normal switching activity and hence, are not under static aging. Hence, they do not mask the propagation of switching events from flip-flop B in its forward cone. Otherwise, if any of G4, G5 or flip-flop A happens to be under static aging, then the flip-flop A and the corresponding flip-flops in the fan-in cone of G4 and G5 should also be switched along with B to ensure a relaxation in the logic gates under stress.

In addition, the sequence of switching events mitigates aging effects on both NMOS and PMOS transistors, i.e., of PBTI and NBTI, respectively [130]. In short, the execution of the proposed subroutine can induce both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions at PMOS and NMOS transistors, leading to the relaxation of both NBTI and PBTI stress. In addition, these transitions are propagated to the logic gates in the forward fanout cone of the flip-flop, thereby leading to their relaxation.

The switching of logic value of a flip-flop can also be translated as corresponding switching

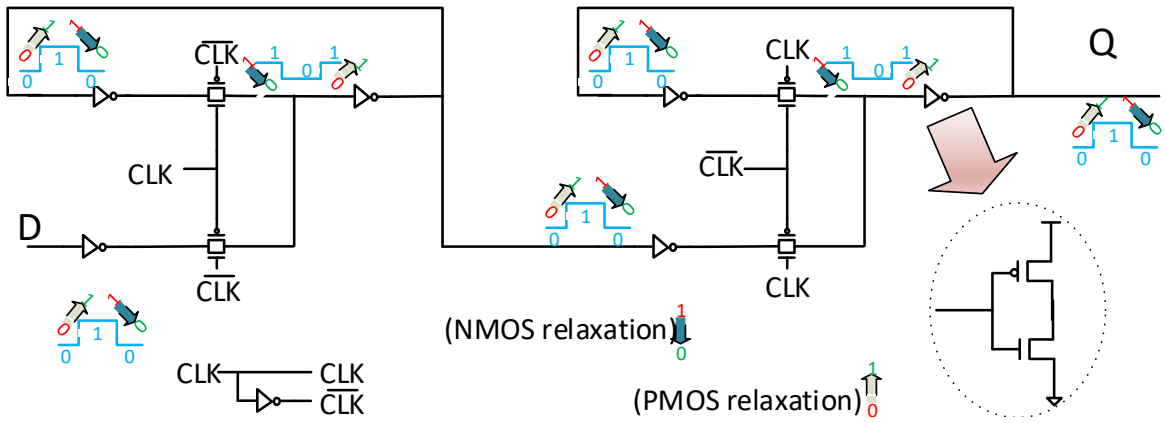


Figure 4.9: Illustration of static aging relaxation of the internal transistors of a master-slave flip-flop on subsequent switching events.

events at the input of transistors inside a flip-flop. For example, let us consider a standard master-slave flip-flop under accelerated static aging phase as shown in Fig. 4.9. When there is a forced switching at the D input of the flip-flop, the internal inverters undergo a corresponding switching activity at their inputs. Hence, a switching event from 0 to 1 and a subsequent switching from 1 to 0 relaxes all transistors in the invertors.

The relaxation effect due to a switching event is explained in [41]. A switching of logic value at the input of the statically aged transistor introduces a fast initial exponential transient [136] of the relaxation component. The value of this transient is dependent on the technology node under consideration. For latest nodes, there is a distribution of time-constants considered for relaxation and drop in ΔV_{th} is observed in discrete voltage steps [137]. A relaxation model defined for the 65 nm node assumes a relaxation time constant in the range of microseconds and in later nodes, this value tends to decrease [138]. In addition, our mitigation technique can be adjusted to increase the time under relaxation by inserting NOPs in the subroutine before restoring the system state.

4.7 Experimental Results

The runtime dependency of static aging characteristics is evaluated for two processors. The monitoring hardware is validated using a Register-Transfer Level (RTL) implementation. Thus, the area and power overheads for the additional hardware are estimated. For this set of experiments, the representative flip-flops are extracted for the Leon3 [139] and Fabscalar [140] processors for different workloads. Experiments were run on a 64-bit Linux machine with 16 GB of RAM and quad-core Intel Xeon processors running at 2.53 GHz. Leon3 is a 32-bit embedded processor based on the SPARC-V8 RISC Instruction Set Architecture (ISA) having seven pipeline stages. Fabscalar is a superscalar out-of-order processor based on PISA (Portable ISA) having eleven pipeline stages. For both processors, we used a single core configuration. The processors are synthesized using Synopsys Design Compiler with the Nangate 45 nm library [111]. The gate-level netlist of Leon3, synthesized for a maximum operating frequency of 485 MHz, contains 35,400 logic gates and 2356 flip-flops. The gate-level netlist of Fabscalar, synthesized for a maximum operating frequency of 740 MHz, contains 133,142 logic gates and 7563 flip-flops.

Six workloads from the MiBench workload suite [141] and six SPEC [32] workloads are executed on Leon3 and Fabscalar, respectively and VCD files were generated for the flip-flop

Table 4.3: Representative flip-flop (FF) selection for different workloads for Leon3 and Fabscalar with $T_{sad\_min} = 3$ million cycles.

	Leon3						Fabscalar					
Total FFs	2356						7563					
FFs on critical path (<10% of maximum slack)	451						536					
Workloads	stringsearch	qsort	susan_smooth	sha	basicmath	crc32	gzip	gap	parser	vortex	mcf	bzip
No. of Critical FFs	34	37	24	36	21	33	207	7	17	5	376	196
No. of Critical FFs (Union)	42						450					
No. of Correlated FF Groups	2	5	2	1	3	2	17	5	10	3	12	16
No. of Representative FFs	7						36					

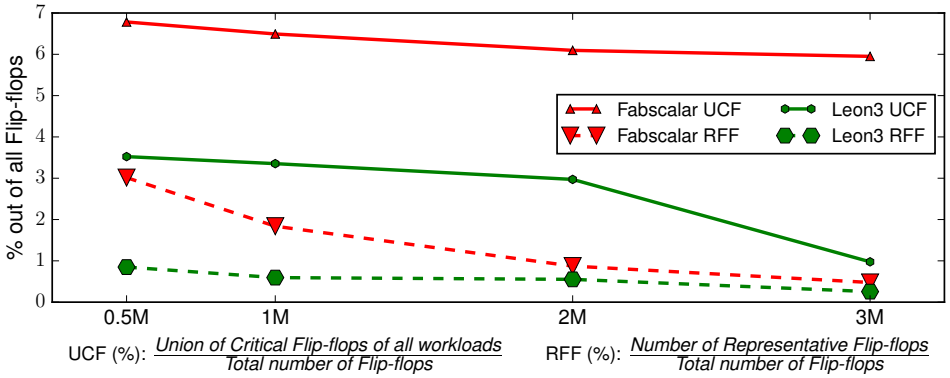


Figure 4.10: Results demonstrating the variation in percentage of (a) union of critical flip-flops of all workloads (UCF), and (b) representative flip-flops (RFF), with the minimum duration of SAPs for Fabscalar and Leon3 processors.

outputs. A VCD parser is used to extract the SAPs in each flip-flop. A BTI-aware STA has been carried out using an in-house framework to extract the critical path slacks. The flip-flops belonging to the critical paths, having a slack less than 10% of the maximum slack, are selected for correlation analysis. For Leon3 (Fabscalar), out of a total of 2356 (7563) flip-flops, we extracted 451 (536) flip-flops on critical paths for further analysis.

4.7.1 Representative flip-flop selection

For each workload, flip-flops were selected based on a minimum SAP duration. For instance, the number of selected flip-flops of Leon3 and Fabscalar for each workload with a minimum SAP duration of 3 million clock cycles are shown in Table. 4.3. The union of the sets of critical flip-flops of Leon3 and Fabscalar for different workloads were found to be 42 and 450, respectively. In addition, for each workload, flip-flops were grouped into several sets where each set comprises of flip-flops having significant SAP correlation as defined in Section 4.4. In other words, a single flip-flop from each such correlated set of flip-flops can be selected as that set's representative. However, since these sets across workloads are not disjoint, the selection of minimum number of representative flip-flops is not trivial.

We employed a greedy algorithm to select representative flip-flops by solving the minimal hitting-set problem as shown in Section 4.5.1. In our approach, only 7 flip-flops are required to represent the static BTI stress in 42 critical flip-flops in Leon3, which has 2356 flip-flops in total. The number of representative flip-flops to be selected can be further reduced by using more efficient algorithms to solve the minimal hitting-set problem.

The average runtime required for pairwise correlation analysis of flip-flops for different workloads was less than one minute. The runtime required for the greedy algorithm to solve the minimal hitting set problem was less than ten seconds.

The minimum SAP duration can be increased to focus on the most critical workload phases. For a typical design, minimum SAP can be evaluated based on the increase in circuit delay due to S-BTI, and the safety margin defined for timing analysis. Our post-synthesis simulation setup can only handle small workloads with a few hundred million processor clock cycles due to runtime constraints. Even with a few hundred million cycles we executed using post-synthesis simulations, in our academic setup, we could confirm the existence of static BTI and long durations of inactivities in some flip-flops. With larger workloads, the effect on delay degradation can be critical and may lead to timing failures.

In Fig.4.10, the minimum SAP duration is varied from 0.5 million to 3 million clock cycles

to extract the number of representative flip-flops for both processors. The results show that the number of flip-flops to be monitored for a minimum SAP duration (T_{sad_min}) of 3 million clock cycles are 7 (0.30% of the total number of flip-flops), and 36 (0.48% of the total number of flip-flops) for Leon3 and Fabscalar, respectively. The reduction in the number of representative flip-flops can be attributed to two factors; (1) the flip-flops having shorter SAPs get filtered out when T_{sad_min} increases, (2) the number of correlated flip-flops increases due to the reduction in number of SAP pairs to be considered for correlation analysis, i.e., the correlation criteria is relaxed.

For different workloads running on two processors, we have proved that a large number of flip-flops are undergoing static aging in a correlated manner, hence a few flip-flops can represent the whole behaviour. In specific cases, if the correlation is not valid, we may need to adopt new low overhead solutions.

4.7.2 Mitigation Measures

For mitigation, we execute a subroutine that can switch the critically aged flip-flops to relax their aging. The subroutine generation process is explained in Section 4.6.2. The ATPG-based instruction sequence for Leon3 netlist was generated using Synopsys Tetramax by masking the primary outputs and constraining the irrelevant primary inputs. The static aging critical flip-flop outputs are added in the fault list with stuck-at-faults assigned. We executed a full-sequential ATPG with no scan cells inserted to find the patterns that can switch the critical flip-flops. The fault propagation was filtered out by assigning the flip-flop outputs as observable primary outputs.

For generating the subroutine, we fixed the criticality threshold for SAP as 3 million clock cycles. The runtime of ATPG tool to develop the subroutine was less than one minute. The results showed that 21 flip-flops (50%) out of 42 flip-flops under critical static aging in Leon3 processor could be switched by the ATPG-based subroutine within 53 clock cycles. The rest of the flip-flops could be switched by creating specific scenarios as explained in Section 4.5.2 and illustrated in Table 4.1. The switching scenarios involving division operation consume 36 clock cycles in Leon3 divider unit and other switching scenarios require 3 clock cycles each. Hence, the total number of clock cycles required for the execution of the functionality-based subroutine is extracted as 123 clock cycles.

4.7.3 Overheads

The proposed method employs a monitoring hardware comprising of switching event detectors, a priority encoder and a criticality-flags register, to monitor S-BTI stress on critical flip-flops. This monitoring hardware is connected to the outputs of the representative flip-flops in the synthesized gate-level netlist of the processors.

The extra hardware added for S-BTI stress monitoring has resulted in an additional area overhead of 0.22% for Leon3 and 0.09% for Fabscalar. The power overhead of the monitoring hardware is estimated as 0.04% for Leon3 and 0.05% for Fabscalar using Synopsys Power Compiler tool. The leakage power of the monitoring hardware dominates in power overhead since the flip-flops under S-BTI switches infrequently during their operation. The additional load at the output of the flip-flops under monitoring did not cause an impact on the overall circuit delay. The area overhead is estimated by synthesizing the processor cores twice, once with the monitoring hardware added to it and then, without the monitoring hardware. The area overhead of the proposed technique is negligible (<0.22%) compared to that of internal node control [117] (1.6%) and input vector control [118] (3.5%). For the flip-flop re-designing technique [124], area optimization leads to 0.81% increase in power consumption, which is one

order of magnitude higher compared to our technique (<0.05%). These methods also report performance improvement in the form of static-aging guardband reduction [117, 124]. In our case, we can adjust the corresponding minimum SAP duration, to achieve the desirable guardband reduction. The mitigation technique employing software subroutines lead to negligible performance overhead. This subroutine is executed only once in three million clock cycles in the worst case and no additional circuit is added for this mitigation.

The subroutine to mitigate 42 critical flip-flops in Leon3 consumes 176 clock cycles (123 and 53 clock cycles for functionality-based and ATPG-based subroutines, respectively) for a critical SAP duration of 3 million clock cycles. The corresponding performance overhead in the worst case is only 0.0059%.

4.7.4 Lifetime Enhancement

Our method can effectively track SAPs and trigger mitigation actions enabling the circuit to recover completely from static aging effect. We compared the lifetime improvement of our approach with conventional methods that re-design flip-flops to tackle static aging. Another conventional method is to reduce the frequency of operation (reactive approach) to take care of aging once the circuit has degraded. We propose a proactive mitigation approach, by introducing short interruptions to static aging acceleration phases, preventing the early failure of the device. The lifetime of a circuit is defined as the duration for which the circuit reliably operates within its delay margin. The lifetime improvement was calculated by comparing the worst case scenarios occurring in two cases, i.e., (1) without SAP monitoring, and (2) with SAP monitoring. For the first case, the circuit, which was simulated for the effect of dynamic aging for 2.9 years, experiences the worst case when an SAP occurs in its critical or near-critical paths. If the total increase in circuit delay exceeds the timing margin, a failure occurs and the lifetime for error-free operation is significantly affected. For the second case, a significant lifetime improvement can be achieved by mitigating SAP effect with the proposed monitoring technique. The lifetime improvement is calculated for a case where (a) an SAP of 3 million clock cycles stresses the circuit on top of a dynamic aging stress of 2.9 years to the case where (b) SAP effect is mitigated and dynamic aging of 2.9 years is the only aging effect.

In conventional methods [117, 118, 124], the lifetime impact on the circuit and the required timing guardband is due to dynamic and static aging of the circuit. In contrast, our runtime monitoring method eliminates the chance of a timing failure due to static aging. Hence, in our analysis, we observed a 1.9X lifetime improvement (2.9 years to 5.5 years for a minimum SAP of 3 million clock cycles) by applying the proposed technique to these processor designs.

4.8 Summary

We have highlighted the importance of determining the workload phases that causes static-BTI stress in timing-critical flip-flops of processor designs. We have emphasized the need for runtime monitoring of critical flip-flops. We have described the design of runtime monitoring hardware that can raise a flag when the critical flip-flops experience severe aging stress. We select a small set of representative flip-flops to track the static aging phases of all critical flip-flops, thereby reducing the area and power overhead for monitoring. The area and power overheads imposed by the monitoring hardware are less than 0.25% for two processor designs. We have also presented a low-overhead static aging mitigation method based on the execution of a software subroutine designed offline.

5 Soft-Error Vulnerability Prediction

5.1 Overview

Radiation-induced soft errors are a major reliability concern in circuits fabricated at advanced technology nodes. Online soft-error vulnerability estimation offers the flexibility of exploiting dynamic fault-tolerant mechanisms for cost-effective reliability enhancement. We propose a generic run-time method with low area and power overhead to predict the soft-error vulnerability of on-chip memory arrays as well as logic cores. The vulnerability prediction is based on signal probabilities (SPs) of a small set of flip-flops, chosen at design time, by studying the correlation between the soft-error vulnerability and the flip-flop SPs for representative workloads. We exploit machine learning to develop a predictive model that can be deployed in the system in software form. Simulation results on two processor designs show that the proposed technique can accurately estimate the soft-error vulnerability of on-chip logic core, such as sequential pipeline logic and functional units as well as memory arrays that constitute the instruction cache, the data cache, and the register file.

The rest of the chapter is organized as follows. Section 5.2 introduces the work in detail, presents a motivation and also lists the contributions. Section 5.3 overviews the preliminaries and related prior work. Section 5.4 describes the proposed soft-error vulnerability prediction method. Experimental results and evaluation are presented in Section 5.7. Finally, Section 5.8 summarizes the chapter.

5.2 Introduction, motivation and contributions

Soft errors, caused by high-energy neutrons from cosmic rays and alpha particles from the packaging material, have become a major reliability concern for nanoscale integrated circuits. Higher transistor densities coupled with aggressive supply-voltage scaling and lower noise margins have led to an exponential increase in the radiation-induced soft-error rate (SER) [142]. Memory arrays such as caches and register files are major contributors to the overall SER [142, 143]. Additionally, the rates of flip-flop Single Event Upsets (SEUs) are comparable to SRAM cell SEU rates in recent technology nodes and are expected to dominate in future technologies [142]. Protection against soft error is costly in memory arrays (such as error correction codes [144]) and even costlier for sequential logic [145]. Rapid, but accurate, soft-error vulnerability evaluation for all these components is crucial for system-level dependability analysis and optimization.

Soft-error-resilience is typically achieved by implementing error correction codes (ECCs), hardening techniques or by performing system-level reliability optimization at run-time. In the ECC-based approach for memory arrays, the soft-error vulnerability of each memory array is evaluated at design time with respect to several representative workloads, and a coding technique that satisfies the reliability requirement of the intended application is implemented [146]. For sequential logic on the other hand, offline SER analysis is used to identify vulnerable sequential elements are identified, and these are selectively hardened by adding redundant transistors to ensure a feedback path that can restore the logic value in the event of an upset. The SER can be estimated using statistical fault injection [147], radiation testing [148], or architec-

tural vulnerability analysis [146, 149] for memory arrays and simulation-based fault injection analysis [150], signal-probability based error propagation [151] and RTL-level vulnerability factor propagation [152] for sequential logic. However, this approach imposes considerable area and power overheads, even for applications that are not associated with high reliability requirements. For example, the simple single error-correction double error-detection (SECDED) technique requires seven additional columns for protecting a 32-bit memory, which results in 21.9% area (and proportional power) overhead. For sequential logic, hardening of vulnerable sequential cells can result in an area overhead of 20% [144] and proportional power overhead.

In online system-level reliability optimization techniques, the soft-error vulnerability is estimated at run-time, and protection measures such as task replication or selective ECC activation are used only for critical tasks with relatively high soft-error vulnerability. In addition, there is a wide range of online protection mechanisms that can be turned on and off at runtime. For instance, it is possible to dynamically turn off one thread in a Redundantly Multithreaded (RMT) system which detects faults by comparison of the execution of two identical threads [153]. Similarly, turning off ECC activation before a 'read' can improve performance [154]. Vulnerability reduction mechanisms, such as activation of pipeline flushing can also be determined during runtime. Instruction throttling and selective redundancy are also runtime controllable, as explained in [155]. To deploy these online protection mechanisms efficiently, it is necessary to know the soft-error vulnerability of an upcoming workload phase during runtime. In this regard, different techniques to predict soft-error vulnerability have been proposed in the literature [57, 153, 154, 156]. However, online soft-error vulnerability estimation is challenging due to limited access to system-state during runtime. In short, a cost-efficient system should be able to handle worst-case scenarios with full-fledged protection mechanisms and less-vulnerable scenarios with basic protection mechanisms. To establish such flexibility in protection mechanisms, there is a need for soft-error vulnerability prediction.

The online soft-error vulnerability of a memory array can be estimated by modifying the RTL model of the processor to mimic the error-propagation behavior of the data stored in memory arrays by adding and tracking flag bits [57]. However, its implementation for all on-chip memory arrays requires extensive modifications to the RTL model of the processor. An alternative solution is to predict the soft-error vulnerability of memory arrays using existing performance counters [153, 154, 156, 157]. The accuracy of this approach depends on the number of performance counters as well as the sampling time intervals, and thus, it is not effective for embedded processors with only a few performance counters. In the case of sequential logic blocks, offline techniques ([152],[151],[150]) used to estimate soft-error vulnerability ignore the workload dependence of vulnerability. Online techniques include emulating the error injection online by setting an error bit [57] and evaluating quantized vulnerability factor using linear regression [154]. However, these online methods concentrate on logic blocks instead of flip-flops, which are one level lower in granularity compared to logic blocks, hence they can only achieve limited accuracy. Therefore, there is a need for a generic technique that can accurately estimate the online soft-error vulnerability without any major modifications to the design.

In this chapter, we propose a general technique to estimate the online soft-error vulnerability of the entire system (a combination of memory arrays and sequential logic cores) using support vector machines and gradient boosting techniques. The correlation is based on the signal probabilities (SPs), i.e., the probability that a signal/node takes logic value '1', of a small subset of representative flip-flops. These flip-flops are chosen at design time by carefully studying the correlation between soft-error vulnerability of processor components and flip-flop SPs for several representative workloads. We exploit two different machine-learning techniques to develop a predictive model for estimating soft-error vulnerability based on SPs. These models are subsequently utilized at run-time as software threads to evaluate the online soft-error vulnerabilities. In contrast to existing techniques, our technique can be easily implemented on

any system as it relies only on the SPs of a few flip-flops, which can be obtained by attaching simple counters to the flip-flop outputs.

Our approach aims at prediction of the vulnerability of the entire system including sequential logic core. Our experiments on two embedded processors show that the accuracy of the proposed soft-error vulnerability prediction method is very high for both memory and logic soft error rates with a very low number of of representative flip-flops. Therefore, the key benefits of the proposed soft-error vulnerability prediction method, when compared to existing methods, are its accuracy and applicability to a broader range of hardware designs.

5.3 Related Work and Preliminaries

Accurate SER estimation is a key requirement for performing cost-effective reliability optimization. The SER of a hardware component c is computed as [146]:

$$SER_c = FIT_c \times VF_c \tag{5.1}$$

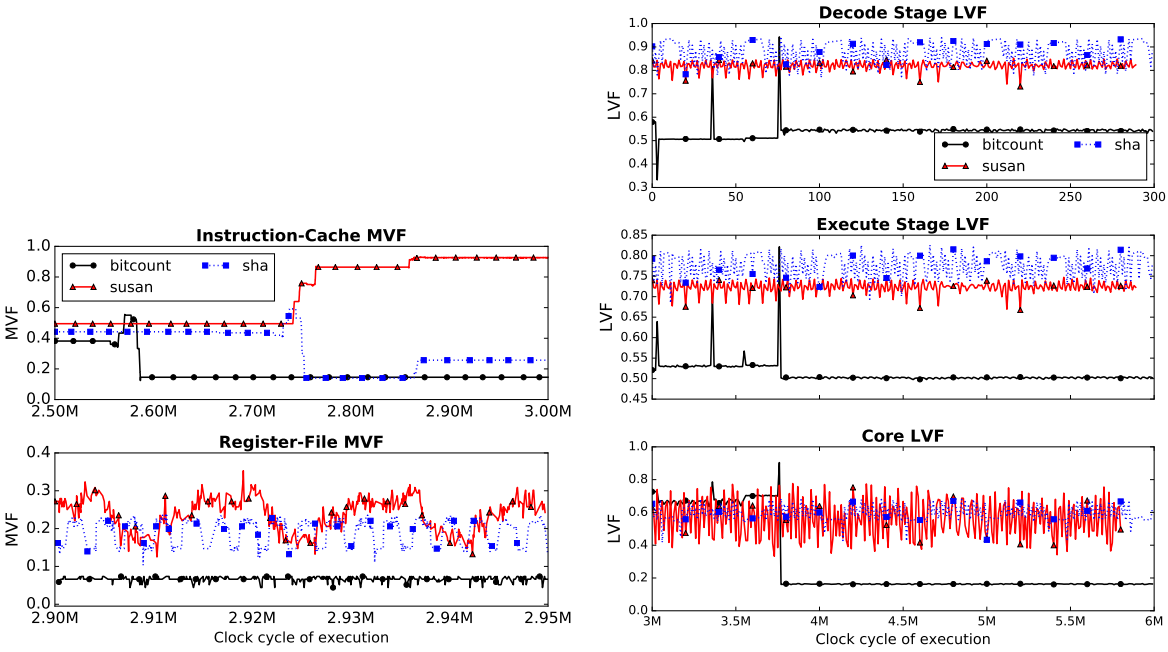
where failure in time (FIT) is the error-generation rate of the target hardware component, which depends on the fabrication technology, the type of component (e.g. SRAM cell, flip-flop, or combinational logic cell), and working environment. For instance, a typical 6T-SRAM cell in 45 nm technology has an error-generation rate of 1095 errors per million cells per billion hours of operation at sea-level [158], and this rate increases nearly 500 times at an altitude of 40,000 feet [159]. For a typical flip-flop implemented in 45 nm technology, the error generation rate is reported as 300-433 errors per million cells per billion hours of operation [160]. In Equation (5.1), the vulnerability factor (VF) is the conditional probability that an error generated in the target hardware component results in a system-visible failure. The VF depends on the hardware implementation and characteristics of the running workload.

The VF of a system can be categorized into two groups, namely,

1. Memory Vulnerability Factor (MVF) and
2. Logic Vulnerability Factor (LVF).

The VF corresponding to storage structures such as caches and register files is termed as MVF. For MVF, any memory cell in a storage structure is considered as a potential source of error. LVF refers to the VF corresponding to a sequential logic core of a processor that is made of a control unit, pipeline stages and arithmetic units, as well as other random logic, accelerators, etc. However, combinational logic elements are less susceptible to soft errors [161–163] and hence, we consider only soft error events in sequential elements (flip-flops and latches) for LVF computation. However, the proposed methodology is easily extensible to consider combinational logic cells as error sites as well.

In order to identify vulnerable components/workloads, SER estimation studies typically focus on VF assessment as FIT is mostly a technology concern. VF is very sensitive to the workload characteristics, and hence, it can significantly vary across different workloads and also across different execution phases of the same workload. Moreover, MVF significantly varies between different memory arrays. This sensitivity of MVF to workload characteristics can be seen in Fig. 5.1a. In this figure, the variation of the MVF is depicted for four MiBench workloads [141]. These MVF values were obtained for the instruction cache and the register file in a 32-bit, SPARC-V8 ISA-based Leon3 processor. In Fig. 5.1b, the variation in LVF of the logic core and pipeline stages of Leon3 for different benchmarks are depicted. The LVF values also tend to vary across workloads and within a single workload. Moreover, the LVF of different pipeline stages are significantly different.



(a) MVF (b) LVF

Figure 5.1: The VF of the memory arrays and sequential logic blocks in Leon3 for three MiBench workloads (experimental setup described in Section 5.7).

Offline vulnerability estimation

A common approach for offline VF estimation is Monte-Carlo-based fault injection [147, 164, 165], in which errors are injected in random bits and cycles, and the effect of these errors on the circuit outputs is observed. The accuracy of fault-injection techniques depends on the number of injected faults, and as shown in [166], achieving an acceptable level of accuracy requires large number of samples, which translates to very high runtimes. The study in [167] shows that high-level error injection techniques can be highly inaccurate when compared to flip-flop error injection techniques.

For MVF, architecturally correct execution (ACE) analysis [149], which has lower computation times compared to fault-injection-based techniques, is an alternative approach for the MVF estimation. In ACE analysis, the lifetime of a bit is divided into ACE and un-ACE intervals. A time interval is classified as un-ACE interval if it can be proven that a soft error in this interval does not result in an error at the circuit output. The fraction of time for which a bit is in ACE state during the execution of the workload is the MVF of that bit.

For LVF estimation of sequential circuits, signal probability based analysis of error propagation has been proposed in [151]. A system-level analysis in terms of mean time to manifest error for a flip-flop has been carried out in [168] for small sequential circuits. A combination of sequential circuit simulation and statistical fault injection has been proposed in [169] for system-level soft error analysis. In [152], a hybrid technique, that combines a higher level ACE analysis and propagation of the extracted VF values through RTL level node graph, has been used to compute the LVF of sequential bits. A mixed-mode simulation combining RTL-level and high-level simulations is adopted in [170] to estimate soft-error vulnerability of uncore components with high speed-up. In [171], a rapid vulnerability estimation technique (RAVEN), which is based on local fault simulations and probabilistic calculations, is proposed.

Offline VF estimation is typically used to identify the most vulnerable components and

protect them by hard-coding the measures for soft-error-resilience at design time. However, this solution results in unnecessary performance and power penalties due to the application of protection measures even during phases of low soft-error vulnerability. Our technique is complementary to all these offline VF estimation techniques, since estimation of VF online requires an offline characterization that can be achieved by any of the above techniques.

Online vulnerability estimation

Online VF prediction can be accomplished by adding virtual flag bits to the RTL model of the processor to emulate the error-propagation behavior from the target structure [57]. The flag bit of the target structure is set to '1' to emulate the injection of an error, and the next instruction that has this structure in its path propagates the flag bit. The processor is then monitored for a fixed number of clock cycles to determine the effect of error on the workload under execution. As with offline fault-injection techniques, this is repeated multiple times to compute the average error-propagation probability. This technique requires extensive modification to the processor RTL model as each register and memory entry has to be equipped with a flag bit to propagate virtual errors along with the normal data.

In [172], store-instruction traces leading to cache block evictions are tracked online to estimate live time of cache blocks. Although, this method is proposed in [172] primarily to shorten the dead-time vulnerability period of cache blocks, it can be extended to predict VF, assuming repetitive program behavior. However, this approach entails significant overhead since it involves tracking history of instruction traces for all cache lines to estimate VF.

Another effective approach for online MVF estimation is to predict the soft-error vulnerability using available performance counters. Although such performance counters are implemented for other purposes, they tend to be correlated with the VF of different components [154]. This approach has been used for estimating the VF of cache arrays [157], issue queue [153, 154, 173], re-order buffer [154, 173], and load/store queue [153, 154] in out-of-order processors. In all these studies, different learning models such as simple linear [157], boosted regression tree [173], Bayesian Additive Regression Trees (BART) [174], Support Vector Regression and Artificial Neural Network [175] are exploited to predict VF from architectural-level features.

For complex super-scalar and out-of-order processors, the availability of a large number of performance counters (e.g., 160 in [153]) can be exploited to obtain an accurate VF prediction. However, there are only a few micro-architectural performance counters in embedded processors (simple in-order processors), and as shown in Section 5.7, this approach is not effective in predicting VF for such processors. In addition, these performance counters are not designed for VF prediction, and hence, they may not be accessible at the required granularity level in many processor architectures.

Existing online techniques [57, 154] estimate the vulnerability of logic elements at a higher-level of granularity, i.e., for logic blocks, but ignore flip-flop or latch-level vulnerability analysis. Hence, the accuracy of LVF prediction is significantly compromised, i.e., around 10% reduction in accuracy according to our results.

5.4 Proposed Methodology

We predict the VF of memory arrays (MVF) and logic cores (LVF) based on the SPs of logic flip-flops. The reasoning behind MVF prediction is that any data transfer from/to memory arrays is not only propagated through logic flip-flops, but also controlled by them. Therefore, the values of flip-flops contain useful information such as access types (read/write), usage (invalid entries), and the stored data. To further motivate the use of flip-flop SPs for predicting

the MVF of memory arrays, some preliminary results are presented in Fig. 5.2. This figure shows the MVF values of the instruction cache and the SP of a particular flip-flop in Leon3 during the execution of a workload from MiBench. The MVF extraction method is explained in Section 5.7.1. From this figure, we can clearly observe that the MVF of the instruction cache is highly correlated to the flip-flop SP. This particular flip-flop was identified using the correlation-based flip-flop selection method, explained in Section 5.5.2. The variations in MVF values for different workloads occur in different timescales as illustrated in Fig. 5.1a. For a workload with MVF varying in a scale of thousands of cycles, prediction accuracy as claimed in this work is practically sufficient. If MVF varies in every cycle, a better observable than SPs of flip-flops will be required.

Similarly, the LVF of a logic core can also be predicted using SPs of a small set of flip-flops. Since LVF is dependent on the error propagation probability on circuit paths, which in turn depends on the SPs of off-path nodes [151], flip-flop SPs serve as a reasonable candidate to predict LVF. The variation in LVF across different workloads or workload segments is dependent on the variation in SPs of circuit nodes. Hence, flip-flop SPs as an observable to predict LVF is an intuitive choice.

The proposed technique can be divided into two phases: (i) offline VF-correlation analysis and prediction-model generation, (ii) run-time vulnerability prediction. The VF-correlation analysis is performed to find the most relevant flip-flops for predicting the VF of memory arrays and logic core. A machine learning based VF prediction model is also developed in this phase. During runtime, counters are connected to the selected flip-flops and the count values are fed as input to the machine-learning model to compute the online VF of the memory arrays and logic cores.

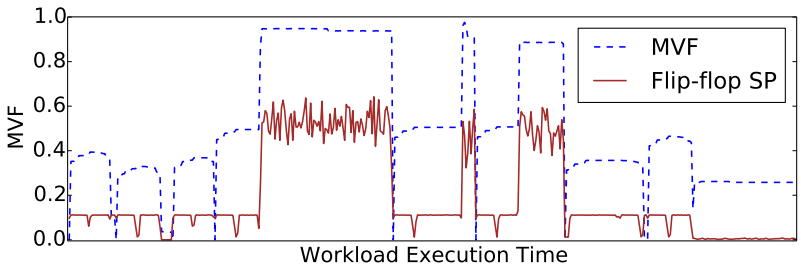


Figure 5.2: Variation of instruction-cache MVF of Leon3 and SP of a selected flip-flop during MiBench workload execution.

5.5 Offline Correlation Analysis

The representation of workload dependency of VF in terms of a parameter that can be measured online is the primary goal of an offline correlation analysis. The VF depends on the system state over multiple clock cycles. The MVF of a memory array at a particular clock cycle depends on the memory access trace, not only for the past but also for the future clock cycles, as data written earlier may be accessed later. This means that the MVF cannot be estimated based only on the current status of the system. Therefore, the system status over millions of clock cycles has to be monitored. Similarly, the LVF of the logic core depends on the logic values of the circuit nodes during the propagation of an error through the circuit paths in multiple clock cycles. Hence, the workload phase of the processor needs to be considered for both LVF and MVF estimation. However, it is impractical to monitor the values of all flip-flops (i.e., system state) over millions of clock cycles. Hence, the entire workload

execution time is divided into smaller segments of constant size and the SPs of flip-flops across each workload segment are used to abstract the system state across that workload phase.

The overall flow of the offline phase is shown in Fig. 5.3. The MVFs of all on-chip memory arrays, LVF of logic core, and SPs of all flip-flops are obtained for different segments of workloads. Each workload is partitioned into workload segments of equal duration and a fixed number of workload segments are randomly selected as the training set. Hence, we obtain a comprehensive training set with elements having diverse characteristics. Next, a subset of flip-flops whose SPs (SP_1) are highly correlated to the MVF of each memory array and another subset whose SPs (SP_2) are highly correlated to the LVF of logic core, are determined using support vector machines and gradient boosting techniques, respectively. Subsequently, two different predictive models are built based on SPs of these two sets of flip-flops. The necessity of these two different sets of representative flip-flops and predictive models can be attributed to the difference in error residency and propagation in memory and logic core leading to different $SP_1 \rightarrow LVF$ and $SP_2 \rightarrow MVF$ correlations.

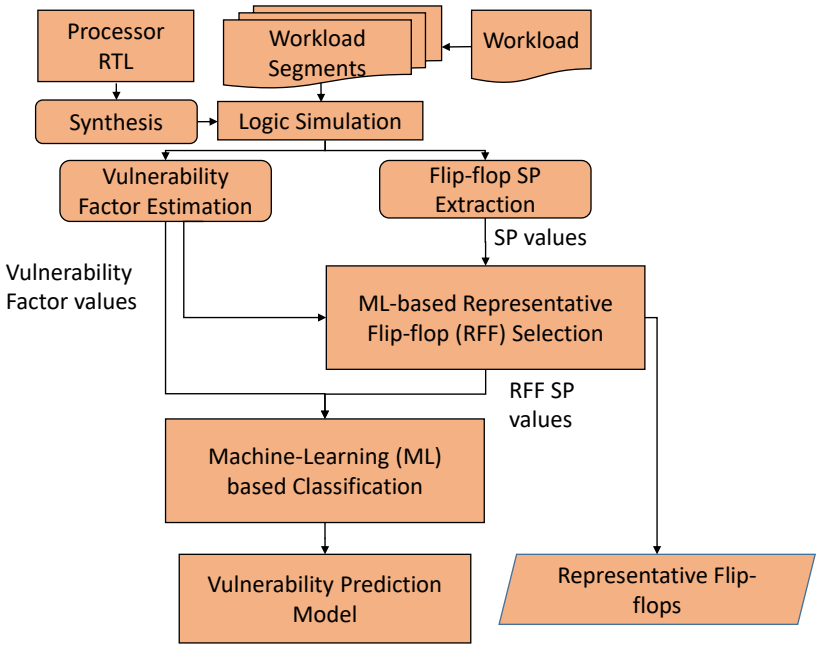


Figure 5.3: Offline correlation analysis and prediction model generation.

The SP of flip-flops over each workload segment can be obtained by performing post-synthesis logic simulation. Since SPs and switching activities are required for power-profile analysis, commercial logic simulators have a built-in feature to dump such information for desired signals and time intervals in a switching-activity-interchange-format (SAIF) file. We use this feature to compute SPs of flip-flops across each workload segment.

Simple counters with sizing according to the length of workload segments are attached to the representative flip-flops in the design phase in order to sample flip-flops online and store SPs. These counter values serve as the input to the vulnerability prediction model when deployed online.

5.5.1 Vulnerability Factor Estimation

Any offline VF estimation technique can be used to obtain the VF of each memory array or logic block. In general, computing the VF of a hardware structure at a particular clock cycle

involves the estimation of the fraction of bits in that structure that can result in an erroneous output.

MVF estimation

The estimation of MVF can be carried out using ACE analysis [149] or Monte-Carlo-based fault injection methods [147, 164, 165] as explained in Section 5.3. Since fault injection analysis is computationally expensive for generating sufficient data samples, ACE analysis is chosen for this work.

The MVF values of three on-chip memory arrays, namely (i) instruction cache, (ii) data cache, and (iii) register files, can be computed using ACE analysis [149]. For each of these memory arrays, the initial step is to track the clock cycles in which a read or a write operation occurs during the execution of a workload. The addresses to which the read or the write operations occurs and the access type for each of those operations are recorded. This information can then be used to determine the ACE and un-ACE intervals for each memory location. Consider the example in Fig. 5.4, which shows the access pattern for a particular address in a memory array. If a write operation occurs in clock cycle n_1, followed by a read operation in clock cycle n_2, then the interval $[n_1, n_2]$ is considered as an ACE interval. If a subsequent write happens in clock cycle n_3, then the interval $[n_2, n_3]$ is considered as an un-ACE interval. This analysis is repeated for all the addresses in the memory array. The AVF of the memory array can then calculated as the ratio of the average number of ACE bits in a cycle and the total number of bits in that array.

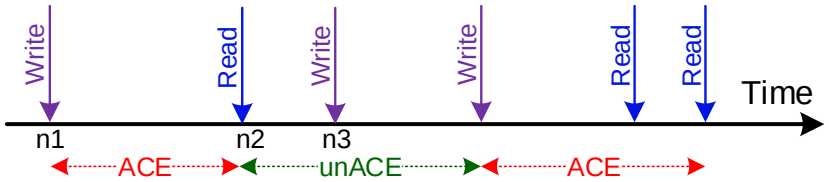


Figure 5.4: Illustration of ACE and un-ACE intervals based on the read and write access patterns.

LVF Estimation

For the LVF estimation of a logic core, (a) the error propagation probability of an SEU occurring at any flip-flop in the circuit to the input of the next reachable flip-flop in the timing path, and (b) the probability that this error getting sampled within the timing window of the reachable flip-flop needs to be considered. The error propagation and sampling probabilities are dependent on logical and temporal masking factors [151], respectively, in this case.

For example, consider a soft-error event occurring in a flip-flop F shown in Fig. 5.5. The flip-flops in its forward cone are considered as reachable flip-flops (flip-flop E in this case) for the error to get propagated. Logical masking depends on the type and input value of logic gates in the propagation path that determines whether the error gets masked. For example, if the SP value of node b (SP^b_{W1}) during the execution of workload segment $W1$ is 0, then the AND gate connected to node b would mask the error propagated from F. Hence, the logical masking effect depends on the SP values of off-path nodes, or in other words, the nature of workload under execution.

To quantify the logic masking effect, we can use any of the previously proposed error-propagation techniques [55, 151]. In these techniques, the relative probability that an error is propagated to a circuit node is calculated based on the type of logic gates in the circuit path. For each logic gate, a set of equations represent the probabilities that an error at its input

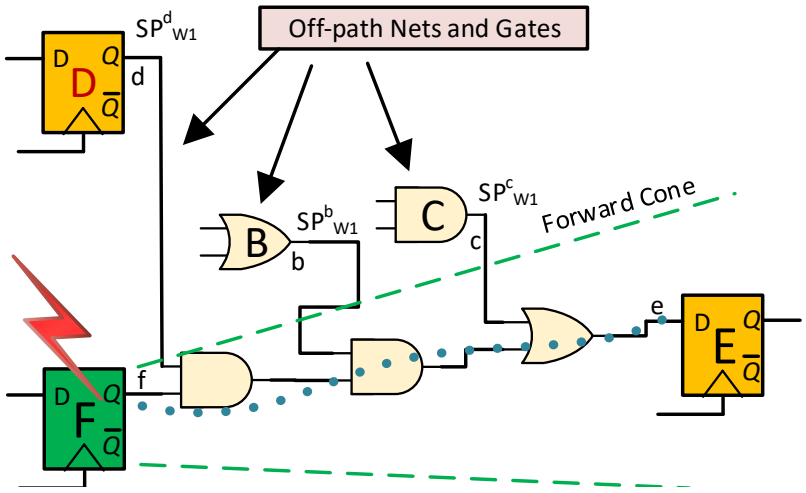


Figure 5.5: Flip-flop SEU due to particle strike and error propagation scenario in a timing path.

propagates to the output. In this way, the probabilities at the input node of each reachable flip-flop in the forward cone of a fault site is computed and is termed as logical masking factor. Similarly, the temporal masking factor represents the masking of an error if it propagates relatively late to the latching flip-flop which has a fixed timing window for the error to get latched [55]. Then, the latching probability (LP) at the input of each reachable flip-flop is calculated as the product of logical and temporal masking factors.

For a flip-flop FF_1, failure probability (FP) is calculated based on the latching probability of all of its reachable flip-flops in its fan-in cone [151] as shown in Eq. (5.2).

$$FP_{FF_1} = \prod_{i \in FF,PO} 1 - LP_{FF_i,PO_i}(FF_1) \tag{5.2}$$

Overall FP or LVF of a logic core is calculated as the average of the computed FPs of all n flip-flops in the circuit as expressed in Eq. (5.3).

$$LVF = \frac{\sum_{j=1}^{n} FP(FF_j)}{n} \tag{5.3}$$

For each workload segment, the LVF of the circuit is estimated by analysing the forward cone of all the flip-flops in the netlist. Initially, the circuit nodes are assigned SP values obtained from the post-synthesis logic simulation of the processor core. For each flip-flop in the core, block based analysis is carried out to calculate the four value propagation probabilities of each node. Additionally, STA is carried out to extract the circuit path delays. From the path delays and propagation probabilities, FP and in turn LVF of the circuit is estimated. This is repeated for all workload segments to generate pairs of SP vectors and LVF values.

5.5.2 Correlation-Based Flip-Flop Selection

The proposed online VF prediction technique relies on the run-time SPs of flip-flops. However, it is impractical to monitor SPs of all flip-flops at run time. Moreover, the complexity of the prediction model depends on the number of its variables, i.e., the number of monitored flip-flop SPs. Therefore, we use a correlation-based flip-flop selection method to select a small subset of flip-flops whose SPs are highly correlated to the VF of the target component.

Let us assume that N *workload segments* are available at design time. The VF-correlation analysis flow, explained in Section 5.5, can be used to generate a training set consisting of N

pairs of (SP, VF), where the sets SP and VF contain the SPs of all M flip-flops and the VF values of all memory arrays and the logic core in our design, respectively. For each memory component or logic block, our goal is to find a set of m flip-flops, $m \leq M$, whose SPs are highly correlated with the VF of that component. This is done using a univariate feature selection method that takes the set of N (SP, VF) pairs and the parameter m as input and returns m features (flip-flops). These m features are selected using an m-best feature selection algorithm in which the correlation between each individual feature and the VF is evaluated, and the m best features [105] are retained as the output. The N $(SP^{'}, VF)$ pairs constitute the final training set for the predictive model, where the sets $SP^{'}$, $SP' \subset SP$, and VF contain the SPs of the m best flip-flops and the VF of all components in our design, respectively.

To illustrate the correlation-based flip-flop selection method, consider a hypothetical scenario with five training workloads. In addition, assume that this system has four flip-flops and two memory arrays. We form the training set as shown in Table 5.1, where the second column in each row corresponds to the set of SPs and the third column corresponds to the VF values of the two memory arrays, Mem 1 and Mem 2. The Pearson correlation coefficients, which take values in $[-1, 1]$, are used to represent the correlation between the SP of flip-flop i, $1 \leq i \leq 4$, and the VF values of the two memory arrays. The Pearson correlation coefficient between two sets of values, $(x_i)_{i=1}^{p}$ and $(y_i)_{i=1}^{p}$, is evaluated as follows:

$$r = \sum_{i=1}^{p}(x_i - \bar{x})(y_i - \bar{y}) \Bigg/ \left(\sqrt{\sum_{i=1}^{p}(x_i - \bar{x})} \sqrt{\sum_{i=1}^{p}(y_i - \bar{y})} \right) \tag{5.4}$$

where $\bar{x} = \frac{1}{p}\sum_{i=1}^{p} x_i$, $\bar{y} = \frac{1}{p}\sum_{i=1}^{p} y_i$ and p is the number of samples. We obtain the correlation coefficients for Mem 1 to be 0.8, 0.4, 0.5 and 0.6. The correlation coefficients for Mem 2 are 0.7, 0.9, 0.6 and 0.7. Therefore, if we want to select the best two flip-flops for Mem 1, then Flip-flop 1 and Flip-flop 4 are selected. Similarly, if our objective is to select the best three flip-flops for Mem 2, then Flip-flop 1, Flip-flop 2, and Flip-flop 4 are selected. The number of flip-flops to be selected depends on the accuracy of the prediction model constructed from the flip-flop data. For unacceptably low accuracies of the predictor, the flip-flop selection process needs to be revisited to extract a larger number of features.

Table 5.1: Hypothetical training set with five data samples

Workload	Set of SPs	VF Value		VF Class	
	{SP1,SP2,SP3,SP4}	Mem 1	Mem 2	Mem 1	Mem 2
W1	0.1, 0.2, 0.1, 0.7	0.4	0.2	−1	−1
W2	0.3, 0.2, 0.5, 0.7	0.3	0.3	−1	−1
W3	0.4, 0.5, 0.2, 0.1	0.7	0.1	+1	−1
W4	0.3, 0.7, 0.1, 0.4	0.1	0.5	−1	+1
W5	0.7, 0.1, 0.4, 0.4	0.6	0.3	+1	−1

5.5.3 VF Predictor Training

The next step in the offline phase is to train a predictor based on the final training set obtained from the correlation-based flip-flop selection procedure. To build the prediction model offline based on correlation between SP vectors and VF values, suitable machine learning techniques can be employed. The technique used should be carefully chosen based on the type of data and the complexity of the correlation function. For MVF prediction, the ability of support vector machines (SVMs) to extract linear combination of features and the high predictive power makes

them an appropriate choice [176]. For LVF prediction, gradient boosting machine (GBM) has been used to attain better prediction accuracy since the correlation tends to be more complex.

MVF Predictor Training Using Support Vector Machines

In this work, we use SVM, a supervised learning algorithm, to train the MVF predictor. SVMs are very popular because of their resilience to over-fitting, robustness to outliers, and high prediction accuracy for a wide range of applications [104]. Since our training samples are limited in number, SVM is expected to perform better than other supervised learning techniques. An SVM-based classifier is trained for each memory array based on the set of (SP, MVF) pairs from the final training set. Instead of directly using the MVF values for training a prediction model, each MVF value is assigned a class by dividing the MVF range, $[0, 1]$, into n classes. The MVF values in the i^{th} class, $1 \leq i \leq n$, are in the range $[\frac{i-1}{n}, \frac{i}{n}]$. This conversion is justified because our goal is to predict phases of high vulnerability during workload execution for selecting the appropriate adaptation action (such as enabling or disabling ECC, task migration and so on) depending on the severity of MVF, and the actual value of MVF is not of much significance.

Before the training process, each set of SPs in the training set is converted to a vector form. We refer to this vector as the SP vector. For training an SVM-based predictor, the SP vectors are mapped into a high-dimensional feature space and an optimal hyperplane (classifier) is constructed in this space. Prediction for new input vectors are made using this function. Let us consider a two-class classification problem to begin the discussion about an SVM based classifier. Let $(x_i, y_i)_{i=1}^{S}$ denote the training set, where $x_i \in \mathbb{R}^d$, and $y_i \in \{-1, 1\}$. The training set consists of S SP vectors, $x_1, x_2, ..., x_S$, and each SP vector has d features and a corresponding target value (MVF), y_i, that is either $+1$ or -1. The goal in SVM-based classification is to find an optimal separating hyperplane, as shown in Fig. 5.6, to separate one class from another.

The classifier can be considered as a function $f(x)$ such that, those vectors x_i for which $f(x_i)$ is positive are placed in one class, while vectors x_i for which $f(x_i)$ is negative are placed in another class. If the number of classes $K > 2$, then it is a multi-class classification problem. According to [177], one-versus-one method is an efficient way to tackle multi-class classification problems. In this method, $\frac{k(k-1)}{2}$ (all possible pairs of classes) two-class classifiers are trained, and then, a voting mechanism is used to classify inputs from the test set.

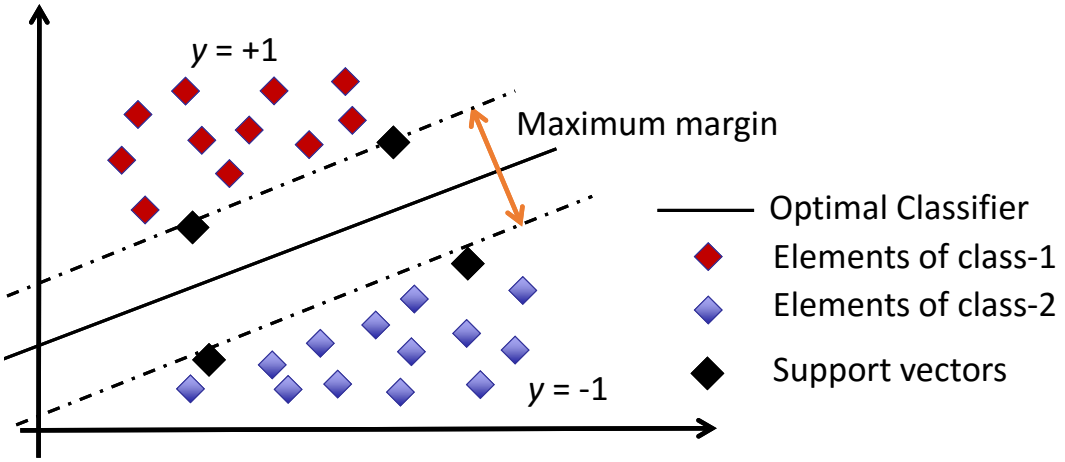


Figure 5.6: Illustration of two-class SVM classification.

To illustrate the SVM classification methodology, consider the hypothetical scenario ex-

plained in Section 5.5.2. We divided the MVF values in the training set into two classes: low MVF [0-0.5) and high MVF [0.5-1]. Moreover, the low MVF class is denoted by '-1' and the high MVF class is denoted by '$+1$'. In this scenario, we need to train an SVM-based classifier for each memory array. The second column in Table 5.1, where each row in this column corresponds to the set of SPs, and the MVF classes for Mem 1 and Mem 2 in the fourth column, constitute the training set for Mem 1 and Mem 2, respectively.

Using the training set for Mem 1 and a linear kernel, we obtain the following predictive model to classify a given set of SPs ($s = \{s_1, s_2, s_3, s_4\}$) into low MVF and high MVF classes:

$$\begin{aligned} f(\boldsymbol{s}) \quad = \quad & sgn(0.508 \cdot s_1 - 1.481 \cdot s_2 + 1.736 \cdot s_3 \\ & +0.950 \cdot s_4) \end{aligned} \tag{5.5}$$

Suppose we have a set of SPs $\{0.1\ 0.3\ 0.2\ 0.1\}$ as the input to our model. The MVF class is evaluated to be $y = -1$ using (5.5).

LVF Prediction Using Gradient Boosting

In comparison with the simple correlation shown in Fig. 5.2 for MVF prediction, which was established using SVM, LVF prediction requires a more robust technique. For determining the complex correlation between SP vectors and LVF values, we found that a stagewise approach combining several classifiers is more effective. We used a different supervised learning method based on decision trees to predict LVF; this method is referred to as gradient boosting machine (GBM) [176]. In comparison to SVM, gradient boosting provides better accuracy and scalability for LVF prediction.

Decision tree is a learning method that is relatively fast to construct and are immune to the effects of outliers. Hence, this method can be resistant to irrelevant features in our data. The major concern with decision trees is their inaccuracy in prediction, but it can be dramatically improved by a technique called boosting. The idea of boosting is to combine the outputs of many "weak" classifiers to produce a powerful "committee". The prediction from all of these classifiers are then combined through a weighted majority vote to produce the final prediction [176]. Gradient boosting is robust to over-fitting.

As in the case of SVM, the training set consists of S SP vectors, $x_1, x_2, ..., x_S$, and each SP vector has d features and a corresponding target value (LVF), y_i. The goal of training is to find an approximate function $\hat{F}(x)$ that minimizes the expected value of a loss function $L(y, f(x))$. This is achieved in the form of a weighted sum of functions $h_i(x)$, called weak learners, i.e., small decision trees of fixed size. This additive model introduced by gradient boosting can be expressed as in Eq. 5.6, where M is the number of classifiers used and γ_m is the weight of each classifier.

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x) \tag{5.6}$$

This additive model is built in a sequence of several stages as shown in Eq. 5.7.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{5.7}$$

The decision tree $h_m(x)$, at each stage, is selected in such a way that the loss function gets minimized. In gradient boosting, the steepest descent technique is employed to solve the minimization problem [178].

For LVF prediction, as in the case of SVM, we use SP vectors as the feature vectors and LVF values as the target values. For our classification problem, we make use of an ensemble of decision tree classifiers to boost the prediction performance. Initially, with a base classifier,

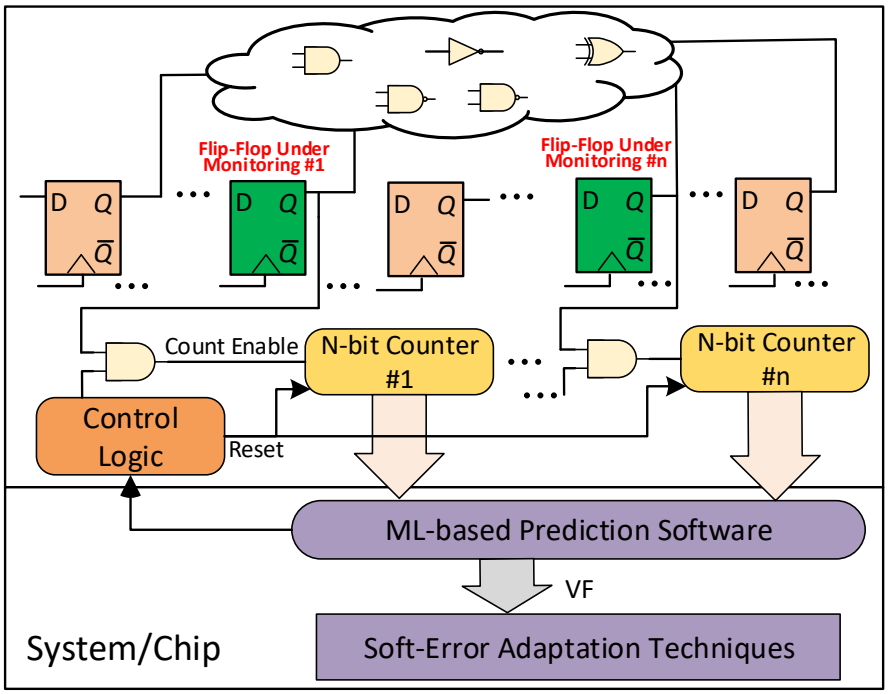


Figure 5.7: Illustration of flip-flop SP monitoring methodology.

we assign equal weights to all training samples. This base classifier can be inadequate and may not classify all samples with high accuracy. In the next step, a second classifier is used to predict the misclassified samples with high accuracy. This is achieved by assigning higher weights to those samples that are misclassified by the base classifier and lower weights for correctly classified samples. In this way, an overall high accuracy is achieved by utilizing all the classifiers used in such a series of predictions.

5.6 Run-time Vulnerability Prediction

The signal probabilities of the flip-flops relevant to the VF prediction are monitored during run-time using a synchronous-up counter, as shown in Fig. 5.7. The counter is enabled whenever the output of the flip-flop being monitored takes logic value '1'; therefore, the counter tracks the number of clock cycles for which the flip-flop output takes logic value '1'. The outputs of all the counters in the design are sampled at uniform time intervals. The sampling frequency is decided at design time based on the characteristics of the representative workloads. A higher sampling frequency is chosen if the VF value changes more frequently during the execution of a workload. On the other hand, a lower sampling frequency is chosen if the VF value does not change very frequently. The sampling frequency also determines the width of the counter that needs to be implemented. For example, if the output of a counter is sampled every 1024 clock cycles, then we need a counter that is at least 10 bits wide. Some additional control logic, as shown in Fig. 5.7, is also required to reset the counters after every sampling operation.

The outputs of all the counters are then transferred to the SVM (GBM) based prediction model, and then, this model translates these workload phase characteristics to MVF (LVF). The system can then take an appropriate protection measure based on the predicted MVF (LVF) value. We prefer a software implementation of the predictor for the following reasons: (1) hardware implementation results in area overhead, (2) soft error vulnerability estimation does not require cycle-by-cycle observation using on-chip hardware.

The prediction software can be implemented as a thread that runs concurrently with other threads on every core. The role of this software thread is to use the counter outputs to predict MVF of the memory arrays and LVF of the logic core. This method of executing the prediction software does not require any additional hardware. However, some performance overhead is incurred because of the regular execution of the software thread.

5.7 Experimental Results

The effectiveness of the proposed approach was evaluated using two open-source embedded processors, namely OpenRISC 1200 (OR1200) and Leon3. We implemented an SVM-based predictor for MVF prediction, and the dependence of the prediction accuracy on the number of monitored flip-flops was studied for different memory arrays. We also compared the accuracy of the proposed method with a performance-counter-based method [153]. For LVF prediction, we implemented a GBM-based predictor since it delivered better accuracy.

5.7.1 Experimental Setup

OR1200 is a five-stage pipeline embedded processor based on the 32-bit ORBIS32 instruction set architecture (ISA). Leon3 is a 32-bit processor based on the SPARC-V8 RISC ISA. The processors were synthesized using Synopsys Design Compiler with Nangate 45nm library [111].

Six programs from MiBench, namely crc32, bitcount, qsort, susan, sha, stringsearch, and basicmath, were executed on the synthesized netlists of Leon3 and OR1200. These workloads belong to categories such as automotive and industrial control, network, security, telecommunications, and office automation with different program characteristics. Each workload was divided into several smaller workload segments of 10^5 clock cycles to collect the required number of workload samples for training and validation. Since we randomly mixed workload segments from different types of workloads, the prediction accuracy is not specific for any particular workload. Workload samples constituting the training set were used to construct a predictive model, and those constituting the test set were used to evaluate the accuracy of the predictive model. The workload segment size is chosen as 10^5 clock cycles based on the trend of change in VF values during workload execution as shown in Fig. 5.1. Moreover, the huge runtime of post-synthesis simulation of the design makes it almost impossible to extract sufficient data samples for a prediction model if workload-segment size is in the order of millions of clock cycles.

For MVF prediction, the MVF values of the instruction cache, the data cache, and the register file were computed using the ACE analysis technique described in [149]. An LVF extraction framework based on error-probability propagation as explained in Section 5.5 was utilized to generate LVF values for the logic cores. The SVM and GBM algorithms used to train and validate the VF predictors were implemented using Scikit-learn [34] with built-in LibSVM software package.

The SPs of the flip-flops in Leon3 and OR1200 were obtained for each workload segment from a SAIF file generated by performing a post-synthesis simulation in ModelSim.

5.7.2 Validation Experiments

In our case, since it takes significantly large runtime to generate each training sample, we needed to obtain fairly accurate prediction models based on a feasible (and limited) number of training samples. For training, we used an equal number of samples in the training phase for both MVF and LVF prediction-model construction. In this way, the choice of the number of

samples represents almost equal offline characterization effort to train MVF and LVF prediction models.

For MVF prediction, the training data set for each processor benchmark consisted of 300 data samples and the remaining 300 data samples were used to test the prediction model. The best values for the Radial Basis Function (RBF) kernel parameters used in the SVM classifier were determined using a five-fold cross-validation approach. In this approach, the training set was divided into five equal subsets and each subset was validated using a model trained on the remaining four subsets. This technique is usually adopted to avoid overfitting when the sample size is limited. A grid search was carried out on the parameters and the parameter values corresponding to the highest cross-validation accuracy were chosen. The prediction model was then trained using these best parameter values and the complete training set. After this training process, the model was tested on a test set with 300 data samples. The "test set" here represents unseen data that can be used to estimate the generalization error or out-of-sample error when prediction model is applied on newer samples. Since, we had more data samples available from MVF characterization compared to LVF characterization, the extra samples were added in the test set.

For LVF prediction using GBM, we have used 300 data samples for training and the remaining 100 data samples to test the model. To find the appropriate values of the tuning parameters of GBM, we performed a grid search with five-fold cross validation similar to that for MVF prediction.

5.7.3 Classification Results

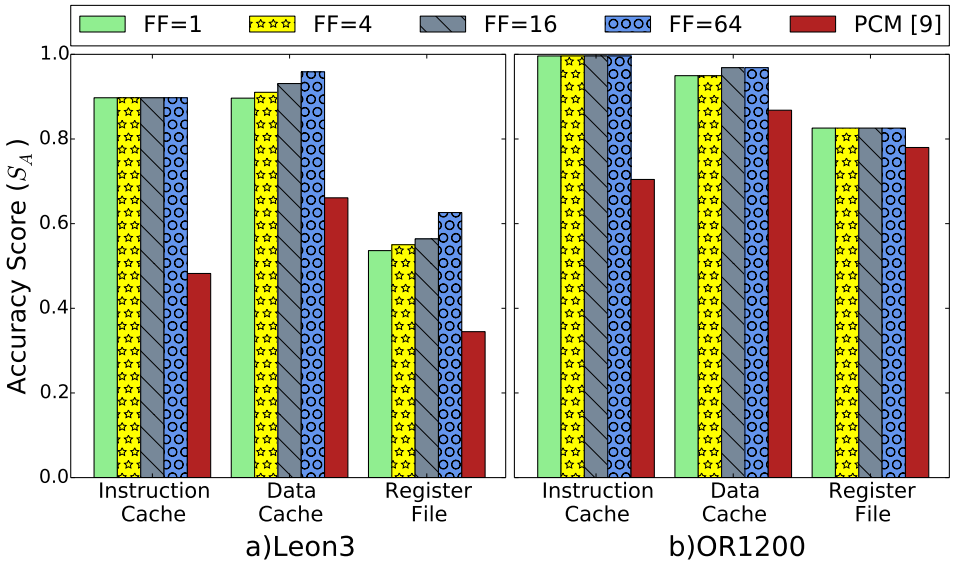


Figure 5.8: Comparison of accuracy scores (S_A) of the proposed method with PCM (prediction based on performance counters as explained in Section 5.7.4) for (a) Leon3, and (b) OR1200 memory arrays by varying number of monitored flip-flops (FF).

We defined three MVF classes: low MVF $[0 - 0.33)$, medium MVF $[0.33 - 0.66]$, and high MVF $(0.66 - 1.0]$. We can increase the number of MVF classes as required and it depends on the type of mitigation action to be performed. Our method is not restricted to any particular number of MVF classes. The prediction accuracy of the proposed method was evaluated using

accuracy score (S_A), defined as follows:

$$S_A(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i), \tag{5.8}$$

where $\hat{y}_i$ is the predicted MVF-class-label of the i-th sample, y_i is the corresponding true MVF-class-label, and $1(\hat{y}_i = y_i)$ is the indicator function, i.e., $1(\hat{y}_i = y_i) = 1$ if $\hat{y}_i = y_i$ and $1(\hat{y}_i = y_i) = 0$ if $\hat{y}_i \neq y_i$. In other words, S_A is the fraction of correct predictions over n_{samples}. A value of '1' for S_A shows a perfect match between the actual MVF-class-labels and the predicted MVF-class-labels. Therefore, the closer the value of S_A is to '1', the higher is the prediction accuracy.

The MVF prediction accuracy varies with the number of flip-flops used in the prediction model, as shown in Fig. 5.8. The results are presented for the instruction cache, the data cache, and the register file. The S_A values for the instruction cache and the data cache of both Leon3 and OR1200 exceed 0.9, even in the case when only a single flip-flop is used in the prediction model. The slightly lower values of S_A for register files can be attributed to its smaller size compared to caches. The register file size of Leon3 is 13% of the cache size. Therefore, the read and write accesses to each address location of register file become more frequent. This random nature of accesses can reduce prediction accuracy. Moreover, the read and write accesses in the case of register file is heavily dependent on the nature of instructions under execution. On the other hand, access to a cache location entirely depends on cache utilization at that instant and pipeline-stall events.

In most cases, there is an architectural explanation for the flip-flops with highest correlation to the MVF values. For instance, consider the instruction cache of Leon3. We discovered that the single flip-flop with the highest correlation to the MVF of the instruction cache has the pipeline-hold signal, i.e., the signal that stalls the pipeline in the case of a cache miss, in its fan-in cone. The pipeline-hold signal is representative of the cache misses occurring in the processor, which in turn represents the write accesses to the instruction cache. Since the MVF computation based on ACE analysis largely depends on the write and read accesses to the instruction cache, the selection of this particular flip-flop is justified.

If only three LVF classes are used for LVF prediction, the prediction accuracy can be low when the data points fall near the class boundaries. Hence, we increased the number of classes to five and represented the misclassification by attaching a level of severity.

We achieve this by assigning higher penalty to a misclassification if the predicted LVF class and actual LVF class are too far apart. The severity in misclassification can be represented using a confusion matrix as shown in Fig. 5.9 where the adjacent rows (columns) represent classes that have LVF values close to each other. The class labels 0, 1, 2, 3 and 4 represents the LVF classes; $[0-0.2)$, $(0.2-0.4)$, $(0.4-0.6)$, $(0.6-0.8)$, and $(0.8-1]$ respectively. The value (color) of an element in the confusion matrix C represents the number of prediction events corresponding to that particular element. For example, if a data sample belonging to class 3 is wrongly predicted as class 0 by the prediction model, the value of $C_{0,3}$ will get incremented. The samples classified correctly will appear in the main diagonal of the confusion matrix, i.e, ($C_{i,j}$ where i=j). From Fig. 5.9, it is evident that most of the wrongly classified samples belong to an adjacent class to the true class label. In other words, very few samples are misclassified with a significantly large error. To quantify the prediction accuracy, we define a metric called weighted accuracy score (S_W) as shown in Eq. (5.9):

$$S_W = \frac{1}{n_{samples}} \sum_{i,j} (1 - \frac{i-j}{n_{classes}-1}) C_{i,j}, \tag{5.9}$$

where $n_{samples}$ is the total number of samples and $n_{classes}$ is the total number of classes.

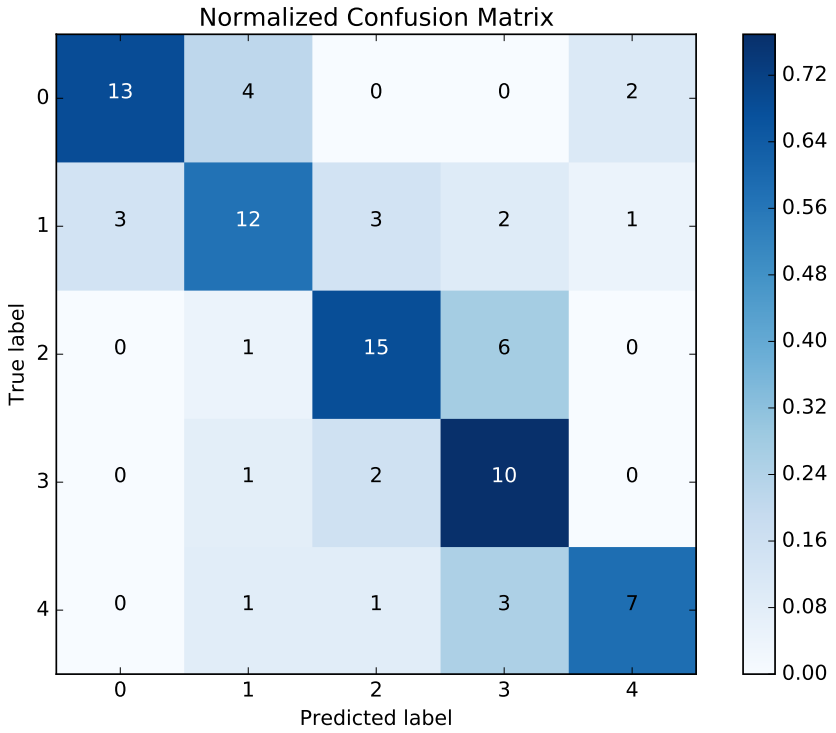


Figure 5.9: Confusion matrix showing predicted and true class labels for LVF prediction for the logic core of Leon3. The number of test samples corresponding to each VF class is overlayed on the matrix.

The results for LVF prediction are shown in Fig. 5.10. For Leon3 and OR1200, with eight flip-flops being monitored, we were able to reach a significantly high accuracy of 0.82 and 0.88, respectively. The accuracy is not significantly improved with an increase in the number of representative flip-flops, hence there is no benefit in monitoring more flip-flops.

5.7.4 Comparison with Related Work

We implemented an MVF estimation method based on performance-counter monitoring (PCM) for comparison. Performance counters were implemented on Leon3 and OR1200 to track the instruction cache hit rate, the data cache hit rate, and the instructions per cycle (IPC). Similar to [153], linear regression was employed to correlate the MVF of memory arrays with performance counter values. For a fair comparison, we used S_A as the metric for evaluating the accuracy of the PCM-based method. However, to employ S_A as the evaluation metric, we had to divide the actual and the predicted MVF values into three classes, as explained in Section 5.7.3.

The comparison results are also shown in Fig. 5.8. The S_A values of the proposed method, even in the case when the prediction is based the SP of a single flip-flop, is significantly higher than the PCM-based method for all three memory arrays of both the processors. On average, we obtained a 38% increase in the accuracy score with our method when compared to the PCM-based method. The higher accuracy of the proposed method can be attributed to the additional logic-level masking information obtained from flip-flop monitoring that cannot be extracted from performance counters. However, if performance counters are available as a part of the processor system, they can be added to our feature list to extract any additional architectural-level information for building the predictor.

Although there are some existing techniques that predicts the vulnerability factor of logic

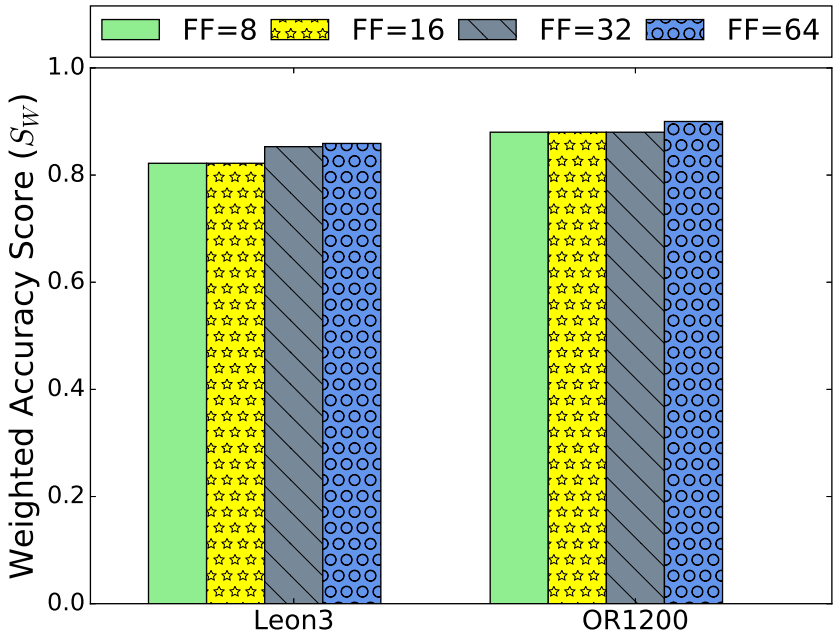


Figure 5.10: Variation in Weighted Accuracy Score (S_W) for LVF prediction with the number of representative flip-flops selected for Leon3 and OR1200 logic cores.

blocks [57, 179], the analysis is carried out only at the architectural level. For example, in [57], the relative AVF prediction error (as a percentage of real AVF) for four processor structues is estimated as around 30%. Since our method relies on the flip-flop level observables for better granularity and analyzes entire logic core, the misclassification rate of only around 20% (or one-class error in most cases) can be considered as a significant improvement.

5.7.5 Overheads

The proposed method employs synchronous up-counters to monitor flip-flop SPs. These counters are connected to their corresponding flip-flops in the synthesized gate-level netlist of Leon3 and OR1200. The updated netlist is re-synthesized and optimized using Synopsys Design Compiler to extract the associated overheads. The size of these counters is determined in the offline phase based on the size a workload segment, i.e., the number of clock cycles constituting one workload segment. For instance, a workload segment of 10^5 cycles requires a 17-bit counter.

Performance Overhead

Performance overhead is estimated by conducting a Static Timing Analysis (STA) on the circuits using Synopsys PrimeTime. The STA reports no increase in maximum circuit delay; therefore no performance overhead is caused by the additional hardware. Although one might expect some performance penalty due to higher load at the output of monitored flip-flops, the synthesis tool re-optimizes the circuit netlist to satisfy the timing constraint. However, this re-optimization may incur some area overhead. In addition, some performance overhead is expected due to the software thread executing the predictive model to compute the VF. However, this overhead depends on the sampling frequency, i.e., how often the counter values are read to compute the VF, and also on the runtime required for each VF computation. For example, if the VF computation is carried out every 10^5 cycles and if it requires 50 cycles for each VF computation, the performance overhead caused by the software-thread execution will be 0.05%.

Figure 5.11: Variation of prediction accuracy with workload-segment size (WS) for different hardware structures in Leon3.

Area and Power Overhead

The additional hardware added for the MVF prediction of the three memory arrays and LVF prediction of the logic core considered in this work results in an area overhead of 0.93% for Leon3 and 0.70% for OR1200. Note that a part of the increase in area can be attributed to the resizing of gates in synthesis-driven optimization, hence the actual area overhead of the proposed method is lower than what is reported above. For Leon3 and OR1200, the power overheads were found to be as low as 0.93% and 0.77%, respectively.

5.7.6 Optimization Prospects

The sampling of flip-flop values to estimate SP can be carried out infrequently to reduce the dynamic-power consumption due to N-bit counters. This approach can also reduce the area overhead. For example, a 17-bit counter for each memory array or logic core is required for a workload segment of 100K clock cycles without any sampling. However, if flip-flops are sampled once every thousand cycles, we require only a 7-bit counter for the hundred sampled cycles. For our case of a logic core with three on-chip memories, this approach reduces the number of flip-flops in counters by 40. However, an additional 10-bit counter is required to count until 1000 to indicate the sampling-clock-cycle. In summary, such a sampling approach could reduce area overhead by 44%.

The variation in accuracy obtained for experiments with different workload-segment sizes is shown in Fig. 5.11. There is no uniform trend for prediction accuracy across different hardware structures with increase in workload segment size. This can be attributed to the temporal variation in rate of change of VF for different workloads. In other words, the optimal workload-segment size can differ across workloads and workload phases. It can be observed that the accuracy of prediction for different hardware structures differs with the choice of workload-segment size. Hence, to achieve better accuracy, careful choice of workload-segment sizes for different hardware structures is recommended. For instance, register-file workload

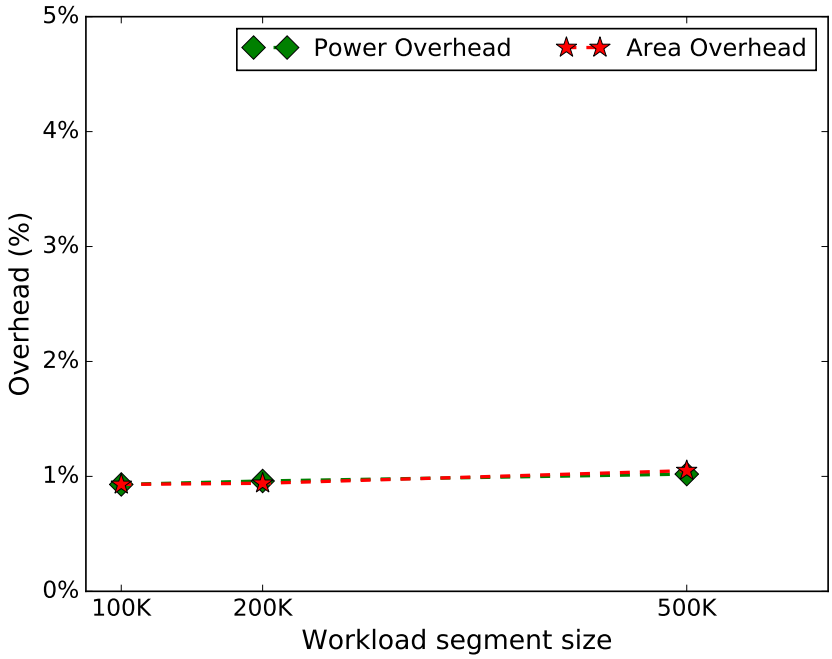


Figure 5.12: Variation of overheads with workload-segment size for Leon3 processor.

profiling can be carried out in workload-segment size of 500K cycles for better accuracy. On the other hand, 100K segment size can be chosen for instruction cache.

The variation of area and power overheads with workload-segment size is illustrated in Fig. 5.12. With a 100 K to 200 K change in workload-segment size, the only change in monitoring hardware is in the size of the counter deployed to track SPs that needs to be changed from 17-bit to 18-bit. Hence, the overall increase in overheads is negligibly small.

5.8 Summary

We have proposed a method to predict the VF of memory arrays and sequential logic blocks during run time. Unlike state-of-the-art VF prediction methods that require a large number of performance counters, the proposed method relies on the SPs of a small number of flip-flops, which can be obtained by attaching simple counters to the flip-flop outputs. Therefore, the proposed method can be applied to a broad range of hardware designs. Simulation results for embedded processor designs demonstrate that the proposed method can accurately predict the VF of on-chip memory arrays and logic core with minimal area and power overhead.

6 Hardware Trojan Detection

6.1 Overview

As the globalization of chip design and manufacturing process becomes popular, malicious hardware inclusions such as hardware Trojans pose a serious threat to the security of digital systems. Advanced Trojans can mask many architectural-level Trojan signatures and adapt against several detection mechanisms. Runtime Trojan detection techniques are considered as a last line of defense against Trojan inclusion and activation. In this chapter, we propose an offline analysis to select a subset of flip-flops as surrogates and build an anomaly detection model based on the activity profile of flip-flops. These flip-flops are monitored online and the anomaly detection model implemented online analyzes the flip-flop data to detect any anomalous Trojan activity. The effectiveness of our approach has been tested on several Trojan-inserted designs of Leon3 processor. Trojan activation is detected with an accuracy score of above 0.9 (ratio of the number of true predictions to total number of predictions) with no false positives by monitoring less than 0.5% of the total number of flip-flops.

The rest of the chapter is organized as follows. Section 6.2 introduces the work in detail, presents a motivation and also lists the contributions. Section 6.3 overviews preliminaries and related work. Section 6.4 describes the overall methodology underlying hardware Trojan detection. Experimental results are presented in Section 6.5. Finally, Section 6.6 summarizes the chapter.

6.2 Introduction, motivation and contributions

Hardware security has emerged as a major concern today as major semiconductor companies have moved towards outsourcing of various stages of hardware design and fabrication to untrusted third-party vendors [12, 61, 63]. Many solutions have been discussed to keep a check on untrusted foundries [180, 181]. A hardware Trojan is generally defined as a malicious modification of a hardware design by an adversary at any stage of design or manufacturing of a chip. The modification of third-party IPs [182] of processor cores can be leveraged to cause the entire functional sabotage of a System-on-Chip. Moreover, malicious modifications of hardware in the form of a hardware Trojan is possible in any stage of chip design and any design abstraction level, leading to Denial of Service (DoS), information leakage, chip destruction, or functional failure [183].

Several offline and runtime techniques have been proposed to detect hardware Trojans inserted at different stages of chip design and manufacturing [184–188]. A holistic solution is hard to achieve since it is extremely difficult to confirm the presence of a Trojan without a golden netlist. Logic and functional-level tests can fail to detect Trojans that are very hard to get activated [189]. In this scenario, runtime detection techniques serve as the last line of defense.

There are several runtime techniques proposed in the literature based on verifying legitimate operations of the processing units. Some of these techniques include system-level monitoring of slack time during bus operation [182], including a hardware property checker to detect permitted and prohibited behavior [190], assertion-based dynamic checkers [191], reconfigurable

logic to verify standard communication protocols [64], and processor protection unit [192] to verify legitimate operation of the pipeline. A different class of techniques that analyzes run-time parameters of the circuit includes thermal profile analysis [193], path-delay signature analysis [194], and performance counter monitoring and prediction [195]. In general, these techniques either verify predefined properties of hardware designs or depend on semantically significant features such as performance counters that are prone to manipulation. In both cases, an advanced Trojan designed by an adversary can potentially mask the specific anomalous signatures in these observables or mimic legitimate operations. In our context, a semantically significant signal can be defined as a signal that implies and contains its functionality in the design document available to a human designer.

In our approach, we consider any functionally-defined (semantically significant) signal in the description of an IP as prone-to-manipulation during IP design and hence, not reliable as an observable/feature to represent *normal* behavior of the IP. We rely on signals in the gate-level netlist that are generated by a synthesis tool and hence, inherently obscure or semantically insignificant as our representative of *normal* IP behavior. For example, the information content of performance counters is dispersed in numerous memory elements and logic gates at the netlist-level such that the information of these counters are still available in the activity profile of relevant circuit nodes even when the counter registers are manipulated.

The activity profile, or more specifically, the *workload profile* (L_p or average logic value of a circuit node over a period of time) of flip-flops in the netlist is identified as a suitable surrogate to represent the workload pattern. The L_p tends to vary with the execution of different workloads on the circuit. The patterns in the variation of this average value can be studied offline to distinguish between the effect of a Trojan payload and a real workload. A group of circuit nodes having maximum sensitivity to workload variation during the execution of real-world applications can be selected as representatives to keep track of a workload moving out of its normality.

A small subset of representative flip-flops is selected offline based on a feature selection technique to monitor the online workload pattern. An anomaly detection model is constructed offline to detect anomalous Trojan activation phases online by analyzing the activity profile of the selected flip-flops. This anomaly detection model can also be updated online based on incremental learning such that the model can get adapted to different online scenarios that are different from workloads used in offline training.

Our results on four different Trojan-inserted designs of the Leon3 processor show that the Trojan activation can be detected with an accuracy score (ratio of the number of true predictions to total number of predictions) of above 0.9 with no false positives by monitoring less than 0.5% of the total number of flip-flops in the design. The extra monitoring hardware causes only a negligible area and power overhead of less than 0.5%.

6.3 Preliminaries and Related Work

Hardware Trojans, in our context, are defined as malicious hardware units added to a chip during any of design and manufacturing stages. These Trojans can cause Denial-of-Service, change of functionality, or performance degradation. The Trojan architecture has two sections; (1) a Trojan trigger, and (2) a Trojan payload.

A majority of Trojans are designed to get activated only rarely on a trigger condition such that the hardware components corresponding to these Trojans remain stealthy and inactive in the test or verification phases. Trojan trigger can be a rarely-true condition in a behavioral description code. For example, a Trojan gets triggered when the processor executes a specific sequence of instructions or when a specific instruction gets executed n times where n is tracked

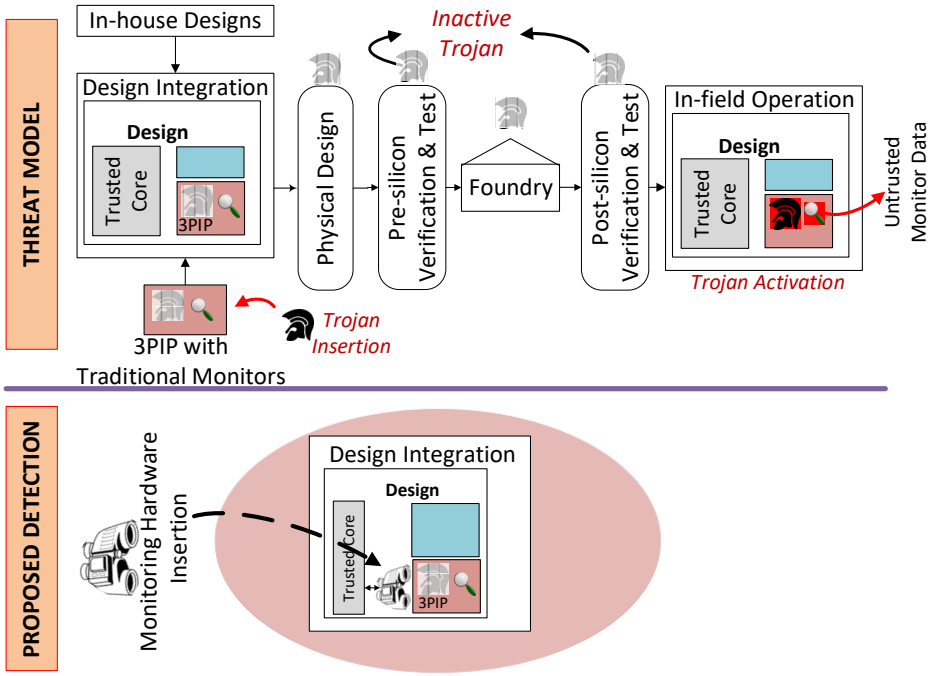


Figure 6.1: Threat model showing stages of Trojan insertion and activation in SoC design flow along with the proposed detection technique.

in a counter.

Trojan payload is the effect of Trojan activation once the Trojan gets triggered. This payload circuit can change the values of internal nodes or overwrite registers or status flags. For example, in a processor pipeline logic, this can be manifested as a corruption of address bus, data bus or instruction register.

The threat model we considered in this work is illustrated in Fig. 6.1 along with the proposed detection mechanism. Similar threat models based on an untrusted netlist of a processor IP core have been discussed in previous work [196]. The hardware Trojan can be inserted by an adversary in a 3PIP [12] design team as a modification of pipeline behavior of a processor core. We defined the threat model specifically for third-party processor IP cores [196]. For other IPs with a specific functionality, it would be easier to find Trojan payloads from workload profile of circuit nodes when the hardware deviates from its functionality. The processor IP can have varying legal usage scenarios while executing general-purpose workloads which makes the Trojan-payload detection harder. The processor Trojans can functionally affect an entire chip eco-system leading to a Denial of Service, change in functionality or degradation of performance. These Trojans are hard to get triggered because of their extremely low probability of true-activation conditions. The functional verification and test stages cannot detect these Trojans with traditional verification and test mechanisms since these Trojans remain inactive during these phases. In this scenario, runtime detection techniques are necessary to track the behavior of such untrusted systems and report anomalous operation when the Trojans get activated.

In addition to the traditional 3PIP threat model, we also consider an adversarial modification of the IP to mimic normal behavior under Trojan activation as shown in Fig. 6.2. For example, abnormal behavior of a 3PIP core due to Trojan activation can be masked by an adversary designing the untrusted core by including an alternative pseudo-monitoring unit. When the Trojan trigger is activated, functional unit of the core is disabled or deviated from the normal functionality. In addition, a pseudo-monitoring unit streams unsuspicious recorded

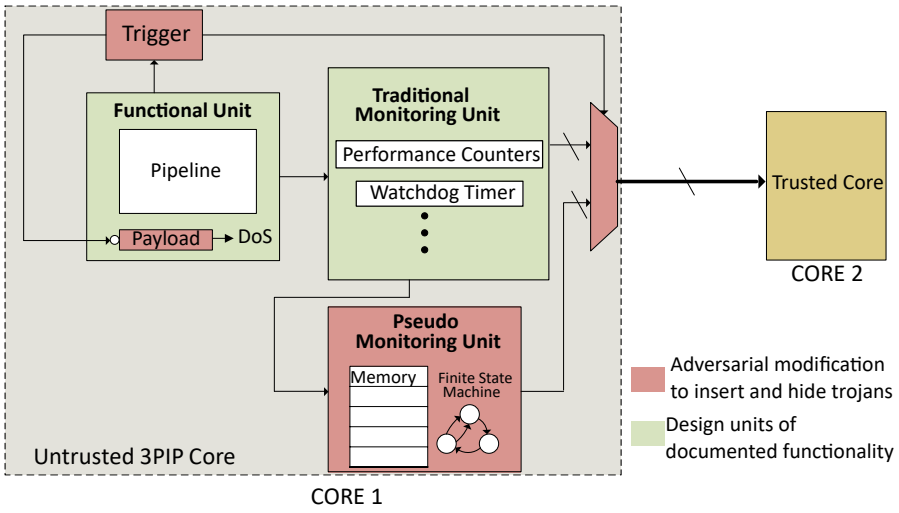


Figure 6.2: Illustration of a Trojan attack scenario including manipulation of architectural-level monitors to mask the attack.

data to mimic a normal behavior of functional unit. In this way, a Trojan detection algorithm in a trusted core monitoring the untrusted core can be unaware of the attack.

The use cases of our approach are limited to Trojan payloads that results in a change in normal workload pattern. Hence, the Trojans causing leakage of information would be outside of our threat model. Our approach is applicable to Trojans that remain active at least for a duration in the order of microseconds if the processor is clocked at 1 GHz. In standard benchmarks [61] used in previous work, a Trojan remains active permanently once it gets triggered and hence, our assumption is valid. The novelty of this work lies in the attack scenario we considered (Fig. 6.2) and the usage of unstructured data for Trojan detection as a more reliable runtime observable and as less prone to manipulation.

There are several Trojan detection techniques applicable during design, verification or test phases that exploits specific properties of Trojan nets such as low controllability and observability, unusage of circuit parts, side-channel signatures etc. In [197], transition probabilities of nets in a netlist are modeled using geometric distribution and lower transition probabilities at suspicious nets are increased by inserting dummy flip-flops. A functional analysis technique (FANCI) proposed in [198] exploits the notion of *control value* (influence of each input on the output) to find suspicious wires in a design based on truth-table analysis. Properties of Trojan trigger nets such as low controllability and observability [199] and rare/seldom usage (unused circuit identification (UCI) [200]) have also been used to identify suspicious signals. A Trojan detection algorithm called HaTCh is proposed in [201] that uses logic testing to blacklist unactivated transitions in internal wires and raises exception on untrusted transitions. A machine learning technique based on one-class support vector machines (SVM) is used in [202] to detect Trojans from IC images obtained by reverse engineering. An anomaly detection technique on side-channel leakage has been used to detect Trojans in [203]. In [204], a static Trojan detection method is proposed based on SVM that analyzes netlist to classify nets as Trojans or normals. To achieve higher Trojan detection coverage, a genetic-algorithm based ATPG technique has been proposed in [205]. However, these techniques are all offline and can be complementary to our runtime Trojan detection technique for advanced Trojans that are hard to be traced offline.

The Trojans that are stealthy and operating through legitimate set of actions can only be detected after their activation during runtime. There are several techniques discussed in the

literature for runtime Trojan detection and are complementary to side-channel and other test-time approaches [12]. In [182], hardware Trojans built into the slave IP cores of heterogeneous Multi-Processor System-on-Chips (MPSoCs) are targeted and the slack time during the bus operation in MPSoCs is monitored to detect the Trojans. A runtime Trojan detection using machine learning methods is proposed in [206] that targets many-core router communication attacks. However, these approaches are limited to system-level monitoring and can detect a suspicious behavior only if it is escalated to system-level communication.

Advanced Trojans combining hardware and software are targeted in [190] by verifying permitted and prohibited behavior of ICs using a hardware property checker. In a similar way, assertion-based dynamic checkers are proposed in [191] for detecting malicious inclusions in processor designs. However, these approaches cannot be comprehensive since it is impractical to take into account every possible Trojan behavior of a processing unit in the property checker.

The reconfigurable logic proposed in [64] performs checks on user-specified security violations and verification of correct implementation of standard communication protocols. A processor protection unit (PPU) proposed in [192] verifies legitimate operation of a pipeline by checking the opcode, the required cycles of execution for each instruction and the properties of specific internal signals. For complex designs, the verification of different properties of a processing unit is very expensive and such approaches are not sufficient to detect Trojans executing legitimate operations.

In [193], the effect of Trojan activation on power consumption of a chip is analyzed by checking the thermal profile. The power consumption waveforms with and without hardware Trojans are analyzed with machine learning techniques to detect Trojan activation in [207]. In [208], chaos theory is applied to interpret power consumption data to capture Trojan activation during runtime. A ring-oscillator based on-chip thermal sensor grid is employed to detect Trojan activation in [209]. However, these approaches are not beneficial for Trojan activation scenarios having negligible impact on the side channels. Path delay characterization is carried out and an authentication scheme based on path delay signatures is proposed in [194]. This approach is expensive for large designs such as processor cores as the number of paths increases exponentially with logic elements. In addition, these signatures can get distorted due to runtime variations and can lead to many false positives in Trojan detection. A Trojan detection scheme based on analyzing performance counter values by machine learning techniques is proposed in [195]. This technique can be less effective if the adversary can manipulate performance counter registers.

In general, many of the runtime detection techniques perform several checks to verify the legitimate operation of a processing unit. Advanced stealthy Trojans can manipulate system level signals to satisfy these known checks. Any semantically significant feature in a processor, such as a performance counter, can be altered by a Trojan activation and cannot be trusted as a representative of trustable runtime behavior. In this regard, we exploit the dispersed workload information available in the logic-level data flow of a processor. The signals in the logic level of abstraction are semantically insignificant for an adversary and hence, hard to manipulate or mask. For example, although the performance counter registers can be overwritten in the RTL code by an adversary, the logic-level flip-flops in the data path terminating at performance counter registers are still accessible. We implement an anomaly detection model by selecting representative flip-flops as features instead of micro-architectural registers that are prone to manipulation due to their tag. Our technique should be considered as a final runtime defense to detect Trojans that have escaped the offline test and verification based detection schemes. In other words, our approach is orthogonal to other runtime techniques [182, 192, 206] and does not compete with them. Our approach can be successful in a scenario in which other techniques depend on a non-trustable observable to predict Trojan activation.

6.4 Proposed Methodology

The main focus of this work is to select appropriate features from a processor netlist for the representation of normal behavior of the processor unit during runtime and thereby, enabling the detection of Trojans by exposing anomalous behavior. In other words, we select a subset of signals from an unstructured implementation (gate-level netlist) of an untrusted IP such that the correlation of selected signals with the functionality of the IP can be used as an indicator of abnormal behavior (activation of Trojans). The behavioral semantics of the nets and signals are inherently obscured in the unstructured representation (gate-level netlist) due to various synthesis steps, and there is no one to one correspondence. This property of unstructured observables (signals) prevents an adversary, who blocks semantic observables, from breaking through the proposed approach. Hence, we select representative signals from the gate-level netlist and implement a monitoring hardware on a trusted core [210, 211] to monitor these selected signals (unstructured observables). An anomaly detection algorithm executed on the trusted core analyzes the monitored data to detect Trojan activation. It should be noted that the gate-level signal is monitored for its dynamic workload footprint rather than any static controllability or observability measurements.

According to our threat model defined in Section 6.3, a system integration team receives an IP in the form of a synthesizable RTL or synthesized netlist from the untrusted IP provider such that they could run simulations to verify the functionality of the IP and run system-level simulations to validate system integration. In this scenario, we assume that we could execute workloads on the untrusted processor IP and extract relevant features from the netlist based on the workload data.

The representative features are selected by analyzing the logic-level data flow in a processor under workload execution during an offline characterization phase. We analyze the scope of workload profile (L_p) of a small subset of signals in a processor core as a representative feature for anomaly detection. Selection of average logic value (L_p) as a feature reduces the hardware cost of feature extractor significantly compared to other feature transforms like wavelets or checksums. L_p is chosen as the representative feature since the information content in L_p of signals is similar to that of a performance counter that tracks, for example, cache hit signal over a period of time. Since performance counters are not specifically designed for the purpose of Trojan detection, they are limited in information content, unreliable and also might not be accessible in the required granularity.

Out of all signals in a given gate-level netlist, we limited our search space only to state elements or flip-flops for two reasons; (1) the bits stored in flip-flops in a clock cycle define the state of a system at that clock cycle and hence, contain information of all nets in a netlist, (2) a flip-flop output can be connected to monitoring hardware with minimum modifications to critical timing paths. During later stages of physical design and scan flip-flop insertion, although more flip-flops may get added to the design, the selected flip-flops still contribute significantly to the core functionality.

The number of signals that should be masked in the RTL in order to hide the Trojan does not correspond to the same number of signals in a synthesized netlist. For example, if the signal probability of a flip-flop in the synthesized netlist is masked, this information is still partially available in the fan-in and fan-out cone of the flip-flop. Hence, a flip-flop or a gate in the fan-in or fan-out cone of a flip-flop can be selected and the same information can be extracted by a signal-probability propagation. In this scenario, the masking of the gate-level information is very difficult for an attacker.

During the offline phase, the representative flip-flops are identified by feature selection techniques that maximizes the overall workload information content. The workload profile of selected flip-flops under the execution of different workload segments are analyzed to build an

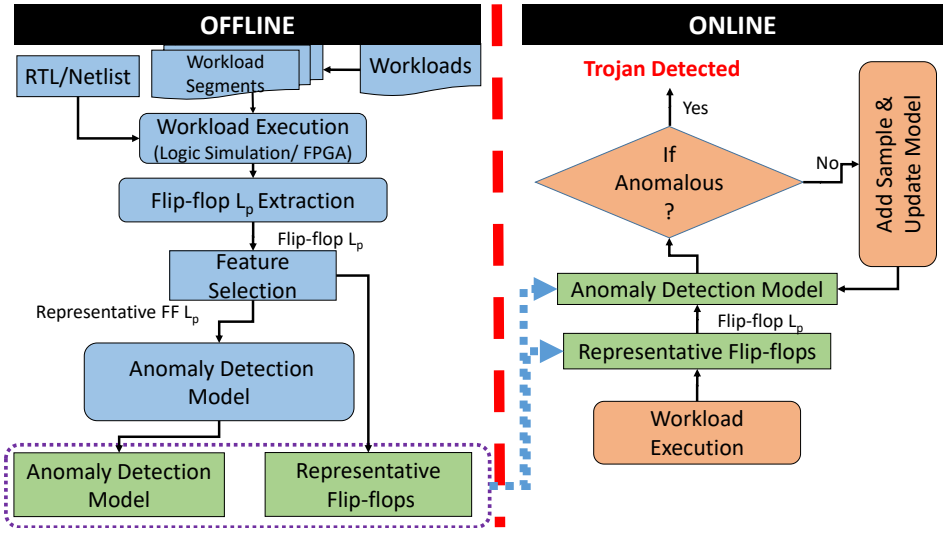


Figure 6.3: Proposed flow showing offline characterization, prediction model construction and its online deployment.

anomaly detection model. A monitoring hardware is deployed online to capture the flip-flop L_p values to a trusted core [210, 211] for analysis. The anomaly detection model implemented online as a software thread on the trusted core analyzes the flip-flop L_p values to detect Trojan activation. The incremental anomaly detection model gets updated online with each verified non-anomalous sample and thereby, increasing prediction accuracy over time.

It should be noted that we have not used synthetic workloads for our characterization. We used standard MiBench [212] workloads for training, and tested our trained model on workloads that have not been encountered during training phase. These workloads belong to categories such as automotive and industrial control, network, security, telecommunications, and office automation with different program characteristics. The workloads can also be chosen based on the intended application of the IP.

The proposed technique can be divided into two stages: (1) offline characterization and construction of anomaly-detection model, and (2) online monitoring and incremental prediction, as shown in Section 6.4.1 and Section 6.4.2, respectively.

6.4.1 Offline Characterization and Construction of Anomaly-Detection Model

The different steps in an offline characterization phase are illustrated in Fig. 6.3. It is assumed that the 3PIP processor core as a part of the design is untrusted according to the threat model defined in Section 6.3. Different workloads are executed on the gate-level netlist of the processor IP using a post-synthesis simulation and the activity of circuit nodes are captured in a value change dump (VCD) file. Each workload is considered to be made of several equal sized workload segments. For each workload segment, corresponding L_p of flip-flops is extracted from the VCD file to represent the behavior of the processor in that workload phase. To reduce the number of flip-flops as representatives, a feature selection technique is used. From the selected representative flip-flops, an anomaly detection model is constructed based on isolation forest (iForest) algorithm [213]. This model is implemented online to detect anomalous Trojan activation.

Representative Flip-flop Selection

Since a design team can only characterize normal behavior of a processor, we can only use non-anomalous L_p samples in our training. Hence, the feature (flip-flop) selection process is carried out with training data corresponding only to a single class (non-anomalous). A design team cannot be expected to have access to anomalous class samples in the training phase. In this scenario, filter or wrapper-based feature selection techniques cannot be used. Hence, we eliminate features based on their information content and mutual redundancy.

To evaluate information content, we employed variance threshold algorithm [214] that removes low variant features. In this method, features with zero-variance (no change in values across all workload samples) and also variance less than a specific threshold value are removed. By tuning the variance threshold, we can reduce the number of flip-flops to be selected for further analysis. The decision of a threshold depends on how many flip-flops are needed to be selected to achieve an acceptable trade-off between area overhead and Trojan-prediction accuracy. The flip-flops having low variance across several workload segments cannot be a good representative of the overall workload pattern. For example, several flip-flops possess an L_p of 0 or 1 irrespective of the workload phase. Since we target the detection of Trojan payload signature in workload phases rather than Trojan trigger, these low variant flip-flops are insignificant. In addition, flip-flops with L_p nearly 0 or 1 are candidates in the suspect list of test-time Trojan detection methods [12]. In these methods, such flip-flops are considered significant as they are suitable targets to insert Trojan trigger due to their input nodes with low controllability. Note that non-switching flip-flops ($L_p = 0$ or 1) are considered as observables of low importance by our feature selection algorithm since their information content would be zero. In this regard, flip-flop L_p with high variance are monitored in our runtime technique.

To remove redundant features, we evaluated the correlation of features by calculating Pearson correlation coefficient [106] between L_p of all pairs of flip-flops. In each iteration, a flip-flop is selected and all other flip-flops having high correlation with the selected flip-flop are eliminated from further processing. This procedure is continued with the remaining flip-flops to select the final set of representative flip-flops.

Construction of Anomaly Detection Model using iForest

A small set of representative flip-flops are selected in the previous step and their L_p under different workload segments are analyzed as training samples. A decision-tree-based machine learning algorithm called *i*Forest is employed to construct an anomaly detection model [213]. A comparison of the performance of *i*Forest with other anomaly detection techniques is presented in Section 6.5.3. *i*Forest is preferred in high dimensional problems and also with training set having no anomalies [215]. In our case, the training set cannot necessarily include Trojan activated samples. In addition, *i*Forest has a linear runtime complexity and low memory requirements making it an appropriate choice for this problem. Most existing model-based approaches for anomaly detection (based on classification or clustering) construct a profile of normal instances and then, identify instances that do not conform to the normal profile as anomalies. Hence, these methods are optimized to profile normal instances but not to detect anomalies, eventually leading to low prediction accuracy [213]. Isolation forest explicitly isolates anomalies instead of profiling normal instances.

The basic working principle of the *i*Forest algorithm is illustrated in Fig. 6.4. Let us consider an anomalous sample shown as A_i and a normal sample shown as N_i in a two-dimensional feature space XY with x and y as feature variables. We split the XY feature space into two partitions by selecting a random feature (x or y) and a random value for that feature. The random split value in x (y) is represented by a vertical (horizontal) line as illustrated in

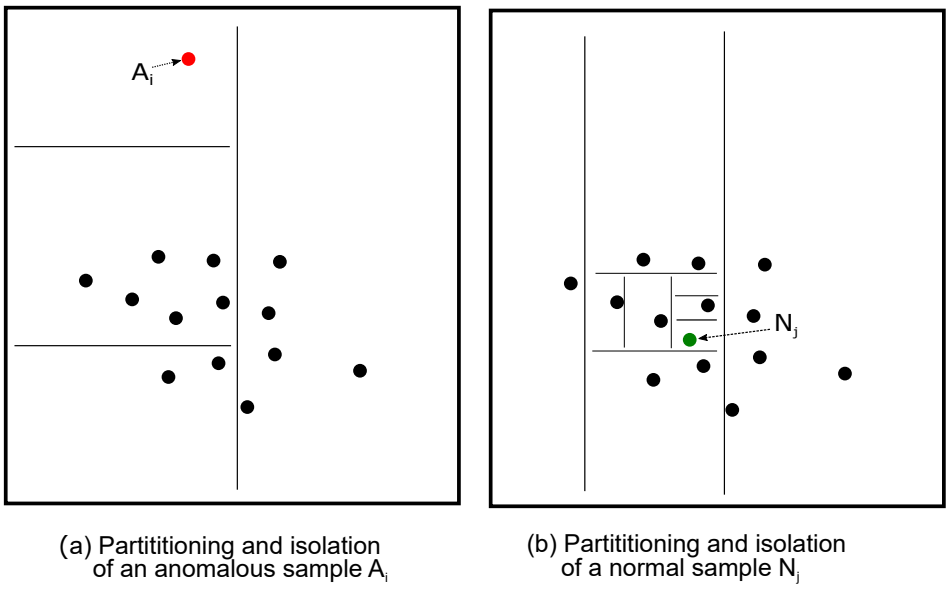


Figure 6.4: Isolation properties of samples by random partitioning.

Fig. 6.4. It can be observed that the anomalous sample A_i is isolated in a very few number of splits and the normal sample N_i is isolated by a significantly large number of splits. Hence, the easy-to-isolate property of anomalous samples is exploited in *i*Forest algorithm with the random partitioning of feature space expressed in the form of binary decision trees as shown in Fig. 6.5. In our case, a workload or a workload segment is considered as a sample and the average logic value (L_p) of a net during a workload segment is considered as a feature.

The *i*Forest algorithm constructs isolation trees by recursive partitioning based on a random selection of a flip-flop and a split value for its L_p as illustrated in Fig. 6.5. Several such isolation trees are constructed to form an isolation forest. The number of splittings required to isolate a training sample is equivalent to the path length from the root node to the terminating node. The average path length of a sample over an isolation forest is a measure of normality and hence, forms the decision function. The samples having short path lengths over large number of isolation trees are considered to be anomalies. In this way, the anomaly detection model is constructed offline based on the L_p of a few selected representative flip-flops.

The low runtime complexity of *i*Forest is due to its model construction based on partial usage of training samples. For a sub-sample size ψ, the time complexity of *i*Forest is estimated as $O(t\psi \log \psi)$ in the training stage and $O(nt \log \psi)$ in the evaluation stage, where n is the number of instances for evaluation at a time, and t is the number of iTrees in the model [213].

6.4.2 Online Monitoring and Incremental Prediction

The *i*Forest based anomaly detection model constructed offline is implemented online on a trusted core to ensure safe operation. The representative flip-flops selected offline are connected to a monitoring hardware as shown in Fig. 6.6. The required circuit connections can be made since the design integration team has access to the synthesized gate-level netlist. The flip-flop L_p samples corresponding to different workload phases are generated by connecting each representative flip-flop outputs to an N-bit counter. The counter values are sampled by the software thread on a trusted core in regular intervals which analyzes the anomaly score to detect any Trojan activation. When a Trojan is activated during workload execution, a sudden shift is observed in the anomaly score as shown in Fig. 6.7.

Table 6.1: Functionalities covered by selected Trojans and the corresponding Trojans implemented on different designs.

Selected Trojans	Functionalities	Similar Trojans [61]
NOP-insertion Trojan	Denial-of-Service	PIC16F84-T200,memctrl-T100
Cache-disabling Trojan	Performance Degradation	s35932-T300, EthernetMAC10GE-T100
Address-bus corrupting Trojan	Change in Functionality	b19-T300,PIC-16F84-T100,PIC16F84-T200,PIC16F84-T400
Jump disabling Trojan	Change in Functionality	b19-T500,MC8051-T600

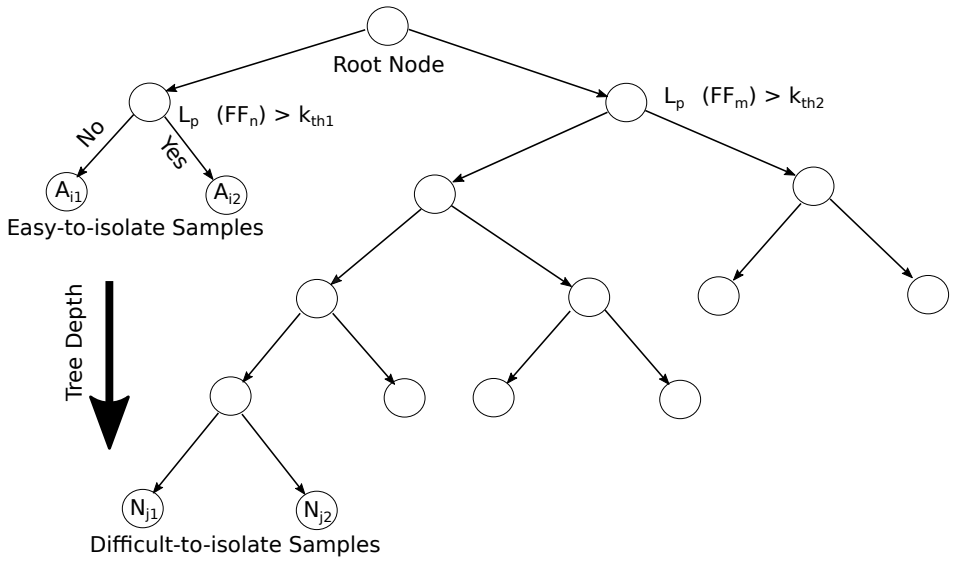


Figure 6.5: Illustration of a random decision tree for isolating samples.

The anomaly detection model can incrementally learn from newer non-anomalous samples obtained online by updating the existing isolation trees or by adding new isolation trees. The anomaly score of an online workload sample is tested by the offline-trained model and if found non-anomalous (true negative), this sample is added to the existing model. The addition of a true negative sample to the model updates the number of samples belonging to a leaf node of each iTree in the model. If an anomalous behavior is detected and is confirmed as a non-Trojan event (false positive), an extra iTree can be added to the iForest model which classifies similar events as negative and the newly added iTree can be given a higher weight in threshold score estimation. The addition of simple decision tree estimators (iTrees) to improve on wrong prediction events is inherent to ensemble classifiers such as random forest classifier or iForest classifier. In this way, the online model can better adapt to different workload patterns that are not encountered in the offline training phase.

The communication between the trusted core and an untrusted core can be established similar to that of an on-chip logic analyzer core communicating with a Leon3 processor core through an advanced high-performance bus (AHB) in ARM Advanced Microcontroller Bus Architecture (AMBA) protocol [30]. The flip-flop outputs are equivalent to trace signals that usually gets captured in a logic analyzer. Once a Trojan payload is detected, a debug trace can be written to accessible memory in a trusted core and a reset signal can be asserted.

6.5 Experimental Results

The effectiveness of the proposed approach was evaluated by building different Trojan benchmarks on the open-source Leon3 processor. Leon3 is selected over other processor implementations such as Oregano 8051 or PIC used in TrustHub [61] due to its multi-core support, extensive customization capabilities, toolchain support, regular updates and support for newer FPGA boards. To ensure reproducibility of this work, we have provided access to the Trojan-inserted Leon3 source code along with Xilinx project files [216]. We built the anomaly detection model based on iForest and analyzed the prediction accuracy while running different realistic workloads on the processor. We also carried out experiments to observe the change in accuracy with the number of flip-flops selected for monitoring.

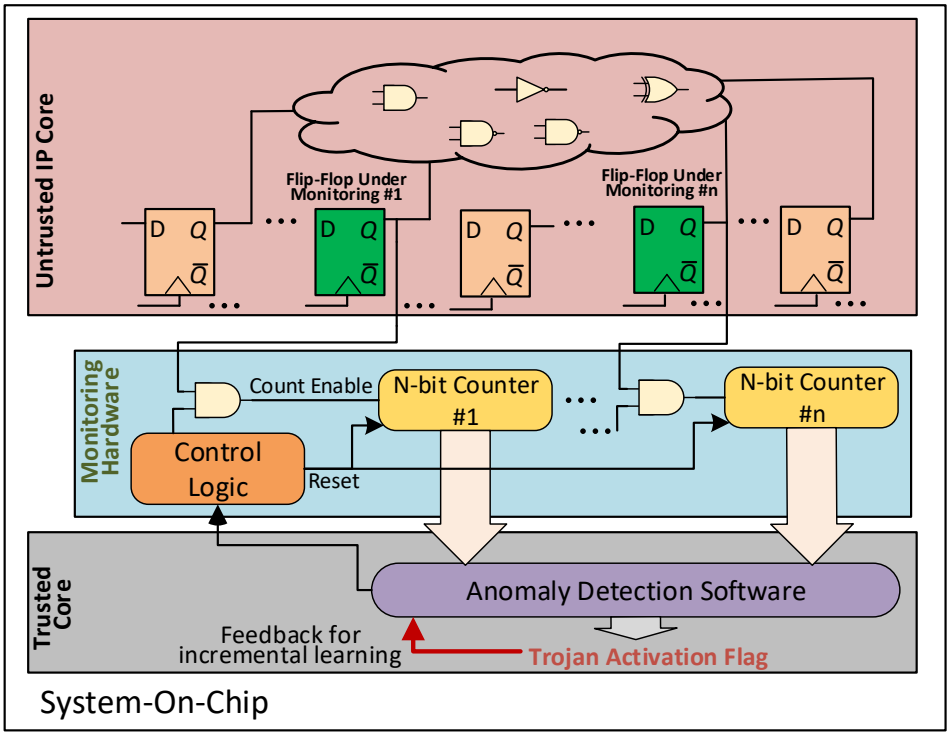


Figure 6.6: Illustration of runtime monitoring hardware and anomaly detection

6.5.1 Experimental Setup

The execution of workloads by post-synthesis simulation and the flip-flop L_p extraction consumes huge runtime and hence, we utilized a Leon3 implementation on FPGA, as an emulation platform. We carried out the workload characterization by executing six MiBench workloads on a Leon3 processor implemented on a Xilinx Kintex 7 FPGA (KC705). Leon3 is a seven-stage pipelined processor based on SPARC-V8 RISC instruction set architecture (ISA). The flip-flop outputs were sampled using an integrated logic analyzer to a host machine. The MiBench workloads were loaded and executed on Leon3 using grmon debug monitor [30]. The *i*Forest algorithm for anomaly detection was implemented using scikit-learn library [34].

6.5.2 Implementation of Different Trojans

Different Trojans were implemented by modifying RTL of Leon3 followed by synthesis, place & route and bitstream generation.

Trojan Trigger is implemented as the 26^{th} bit of a $32-$bit counter that increments its count on every match with a specific instruction (e.g. XOR) in the instruction register and stops counting once the Trojan is triggered. We can increase the complexity of the trigger by altering the trigger condition. However, it is not necessary for this work since the proposed method detects Trojan activation after it occurs by analyzing Trojan payload instead of trigger. The trigger signature can provide additional information for anomaly detection. However, it is intentionally not used in our technique to show that the prediction performance is independent of the type of Trojan trigger.

We implemented four different Trojans that activates different payloads.

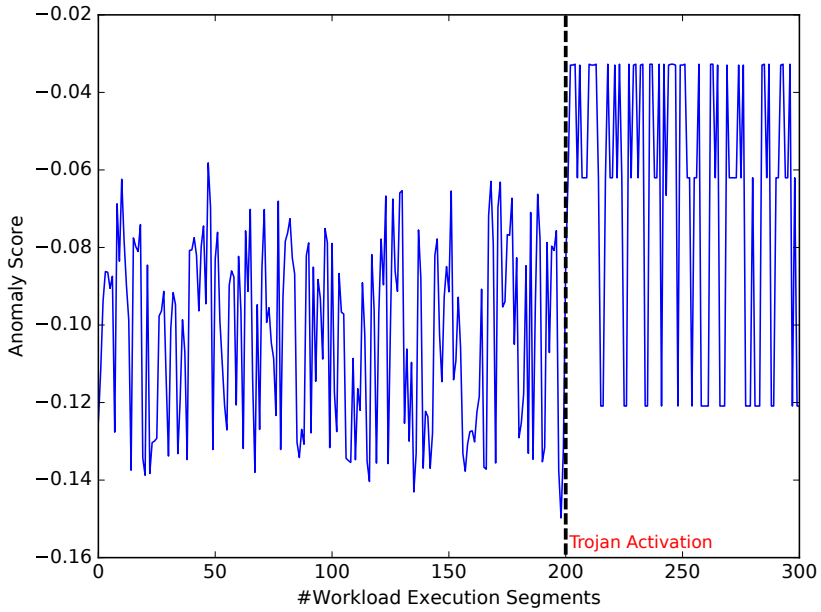


Figure 6.7: Illustration of the shift in anomaly score when the NOP-insertion Trojan is activated during the execution of a susan_smooth application in Leon3.

Trojan 1: NOP-insertion Trojan

This Trojan inserts No-Operation (NOP) instruction in the instruction register replacing any existing instruction such that the processor executes only NOPs when the Trojan is activated. This results in a Denial-of-Service and is based on the PIC16F84-T200 Trojan benchmark in Trust-Hub [217, 218].

Trojan 2: Cache Disabling Trojan

This Trojan tampers with the read enable of the instruction and data caches of the processor similar to Trojan 3 of [195] resulting in performance degradation.

Trojan 3: Address-bus Corrupting Trojan

This Trojan corrupts the address bus of the processor leading to change in its functionality or Denial-of-Service. This Trojan is similar to the PIC16F84-T100 in Trust-Hub.

Trojan 4: Jump Disabling Trojan

This Trojan disables the jump instruction in the pipeline leading to change in the functionality. This Trojan is similar to MC8051-T600 in Trust-Hub.

In Trojan taxonomy, based on the effect of a Trojan, they are classified into four categories.

- Denial of Service
- Degrade Performance
- Change Functionality
- Leak Information

We selected four Trojans from the first three categories for our experiments. These four Trojans corresponds to ten similar Trojans in Trust-Hub as shown in Table 6.1. The Trojans causing leakage-of-information were not considered in our threat model.

Table 6.2: Prediction performance metrics of the proposed anomaly detection method

Scenario	Trojan Size (%)	#Test samples	$\sigma(L_p)$	$\mu(L_p)$	#Selected flip-flops	Precision	Recall	F_1 score	(S_A)
Trojan 1	0.002	40	0.395	0.204	12	1.000	0.950	0.974	0.975
Trojan 2	0.001	40	0.447	0.293	12	1.000	0.850	0.919	0.925
Trojan 3	0.002	40	0.377	0.194	12	1.000	1.000	1.000	1.000
Trojan 4	0.001	40	0.370	0.229	12	1.000	0.950	0.974	0.975
Overall		160			12	1.000	0.941	0.970	0.969

6.5.3 Trojan Detection Accuracy

The Trojan activation is detected as positive when the average anomaly score of a series of online samples estimated by the *i*Forest algorithm exceeds a pre-defined threshold score. In a qualitative sense, the threshold score can be understood as how much deviation from a legal workload pattern would be flagged as a Trojan payload. The decision-time threshold cannot be known in advance as it can change during the incremental online learning of the classifier. The threshold score can be initialized offline and updated online based on online data collected from the monitoring hardware. A predictive action of the predictor software is labeled as a true (false) positive if the Trojan is (not) activated and is detected as a Trojan activation. A predictive action of the predictor software is labeled as a true (false) negative if a normal execution of the workload is detected as normal (anomalous).

To ensure the generalization capability of our prediction model, we test our model with workloads that are not encountered in the training process. It is also possible that the prediction model encounters false positives on the events of exceptions or error conditions. However, we can avoid these by considering the fact that exceptions or error conditions are limited to a short time duration before retaining the normal execution. The identification flag for Trojan can be raised if the Trojan continues execution beyond such timescales. To differentiate Trojan payload from short error conditions or exceptions of maximum duration (t_e) in a workload, a decision time (t_d) can be defined such that, $t_e < t_d$, for each Trojan detection event. An anomalous activity is only recognized as a Trojan payload if the anomalous flag is raised for a duration of more than t_d.

In a two-class classification problem, precision is defined as the proportion of positives (anomalous activity) that are correctly identified as such and recall is defined as the proportion of negatives (normal activity) that are correctly identified as such. The prediction performance of Trojan detection is estimated based on F_1 score which is calculated as the harmonic average of precision and recall. For the two-class classification problem (Trojan activation detected (positive) or not detected (negative)), if the number of true positives is expressed as T_P, true negatives as T_N, false positives as F_P and false negatives as F_N, then precision is expressed as (6.1):

$$precision = \frac{T_P}{T_P + F_P}, \tag{6.1}$$

recall as (6.2):

$$recall = \frac{T_P}{T_P + F_N}, \tag{6.2}$$

and F_1 score as (6.3):

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}. \tag{6.3}$$

For training the anomaly detection model, we used 2500 flip-flop L_p samples from five different MiBench workloads with no Trojan activated. Since we assume that the Trojans escape the chip test and verification phases, the Trojan trigger condition should be complex enough to not get triggered by normal workload execution. The anomaly detection model is then tested on samples from the execution of another workload (not included in the training set) running under four different Trojan-included designs of Leon3. The test set includes workload samples before and after Trojan activation to estimate false positives and false negatives.

The prediction results are as shown in the confusion matrix in Fig. 6.8. The confusion matrix shows the prediction results for 160 test samples to two classes, positive and negative. The eighty negative class samples are predicted accurately with no false positives. For eighty

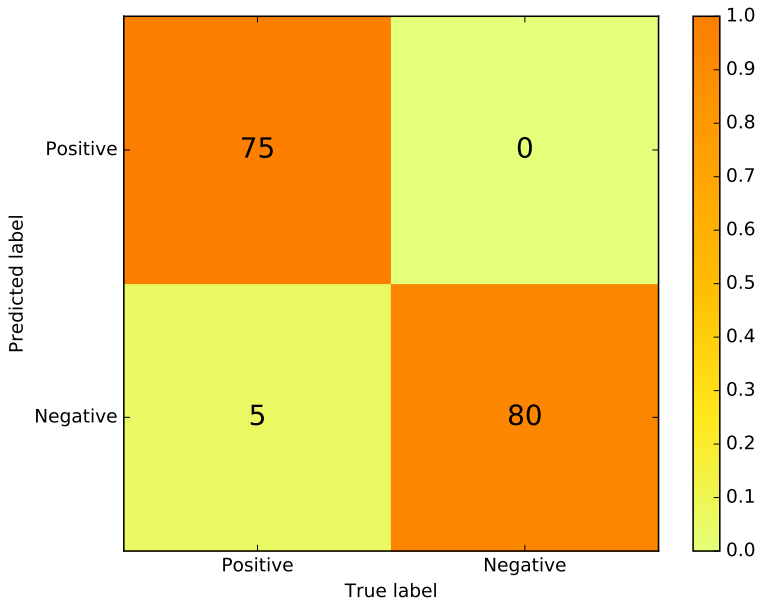


Figure 6.8: The confusion matrix showing the number of true and false predictions corresponding to two classes, (1) positive (Trojan activated) and negative (no Trojan activated).

positive class samples, five samples are predicted incorrectly (5 false negatives). These false negatives originate due to the limitation of workload characterization. The number of false negatives can be reduced by including more workloads in training the classifier such that it can encounter a larger spectrum of sub-workload samples or workload phases.

The prediction accuracy of the proposed method was also evaluated using accuracy score (S_A), defined as follows:

$$S_A(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i), \tag{6.4}$$

where $\hat{y}_i$ is the predicted class label of the i-th sample, y_i is the corresponding true class label, and $1(\hat{y}_i = y_i)$ is the indicator function, i.e., $1(\hat{y}_i = y_i) = 1$ if $\hat{y}_i = y_i$ and $1(\hat{y}_i = y_i) = 0$ if $\hat{y}_i \neq y_i$. In other words, S_A is the fraction of correct predictions over n_{samples}. A value of '1' for S_A shows a perfect match between the actual class labels and the predicted class labels. Therefore, the closer the value of S_A is to '1', the higher is the prediction accuracy.

The obtained prediction metrics are shown in Table 6.2. The values of accuracy score and F_1 score show that the Trojan activation can be predicted with high accuracy by monitoring only 12 representative flip-flops. The threshold score can be adjusted to trade-off between precision and recall. In our method, the threshold score is tuned to maximize precision and eliminate false Trojan alarms by tolerating a few false negatives.

The accuracy score quantifies the prediction performance of the trained model as a ratio of true predictions to the total number of prediction events. An accuracy score of 0.909 means that out of 1000 test workload samples, around 909 samples are expected to be correctly classified as Trojan-infected/Trojan non-infected. With only normal samples in training data, the anomaly detection model could achieve an overall accuracy score (S_A) of 0.969 as shown in Table 6.2. For 160 test samples, overall accuracy score (S_A) for unsupervised learning with both Trojan-infected and Trojan non-infected samples in the training data is obtained as 0.920.

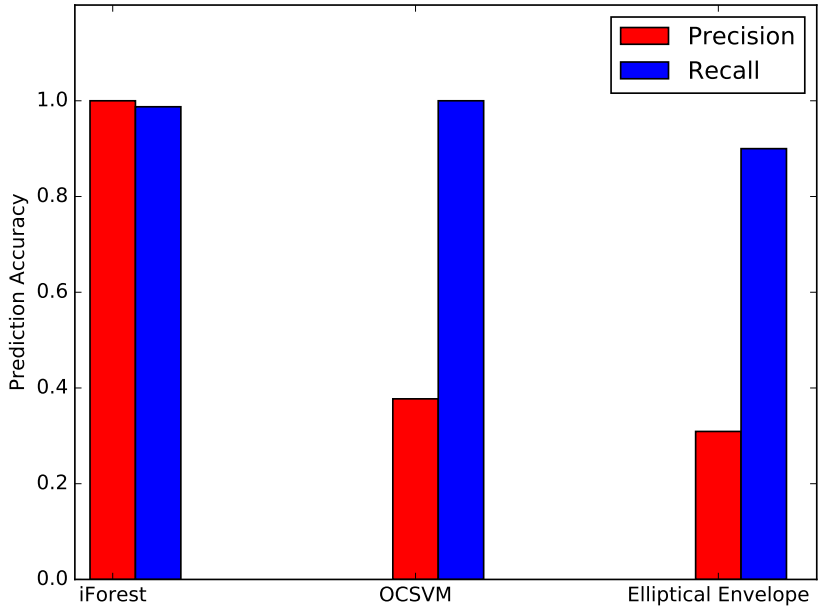


Figure 6.9: Comparison of performances of different anomaly detection algorithms on Trojan detection.

We conducted cross-validation across workloads by calculating the average accuracy score for six sets of train-test pairs by changing the workload in the test set. The average accuracy score is estimated as 0.909. Hence, we obtained significant accuracy in our prediction even when the workload encountered during runtime is significantly different from the workloads involved in training.

We also compared the performance of *i*Forest algorithm with other commonly used anomaly detection algorithms such as one-class SVM (OCSVM) [219] and elliptical envelope [220] as shown in Fig. 6.9. Although recall for OCSVM is slightly better than *i*Forest, precision of *i*Forest is significantly better than both the other anomaly detection methods.

6.5.4 Representative Flip-flop Selection

The number of representative flip-flops can be reduced with a compromise in the accuracy of Trojan detection. The reduction in accuracy with the variation in number of flip-flops is shown in Fig. 6.10. The results show that the accuracy is not significantly dropped when the number of representative flip-flops is reduced from 4% to 0.5% of the total number of flip-flops.

The representative flip-flops selected for monitoring are analyzed to find their significance in anomaly detection. Some of these flip-flops are related to integer condition code (icc) bits of program status register (PSR), shift count (shcnt) bit, restart register bit, pipeline register bit in the decode stage, middle bit of program counter etc. These flip-flops could implicitly represent the workload pattern even if their forward logic cone is manipulated by the adversary.

6.5.5 Overheads

The Leon3 processor design with the monitoring hardware attached is synthesized using Synopsys Design Compiler with Nangate 45nm library to analyze the power and area overhead. The monitoring hardware consists of N-bit counters attached to each flip-flop in the design. The size of the counter depends on the size of workload segment that needs to be analyzed at any point of time during workload execution. For example, to monitor at the granularity of 1000 clock cycles of a real MiBench workload, one 10-bit counter will be used for each

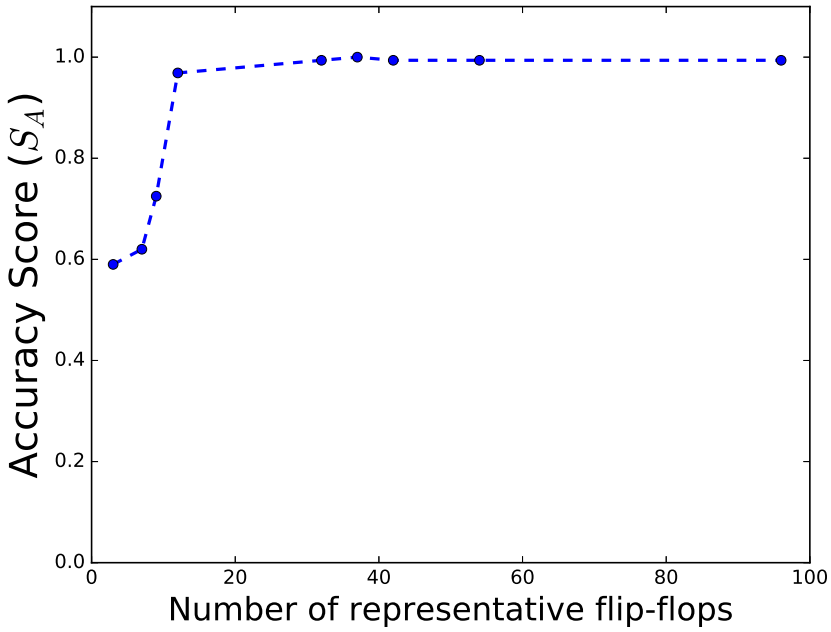


Figure 6.10: The variation in prediction accuracy of Trojan activation in Leon3 with the number of flip-flops selected for monitoring.

representative flip-flop. For 12 representative flip-flops with a prediction accuracy of 0.909, the area overhead due to the extra monitoring hardware is estimated as 0.42% and the power overhead is estimated as 0.40%. The power overhead can be further reduced by implementing time-sampling in calculation of L_p of flip-flops [24].

The execution of anomaly detection software causes a performance overhead on the trusted core and this can be evaluated based on how often the software needs to be executed. The *i*Forest classifier involves only decision tree traversals with simple comparison operations and has linear runtime complexity. The frequency of activation of anomaly detection software will be decided based on the usage scenario (criticality) of the IP. In the case of noncritical systems, this software can also be infrequently executed to reduce performance overhead. If the trusted core on which the software is executed is assumed to be working at 1GHz with similar specifications of an embedded processor like Leon3, the runtime of the anomaly detection algorithm with the number of *i*Trees as 10 and a maximum depth of the *i*Forest as 10 is estimated as around 80 μs.

6.5.6 Limitations of the proposed approach

The proposed Trojan detection framework exploits dispersed information at logic-level to tackle tailored attacks by adversaries on semantically significant observables. However, our method is not aimed at detecting Trojans that cause minimal workload footprint in logic level (e.g. Trojans causing leakage of information). To detect such Trojans, side-channel measurement based runtime Trojan detection techniques [203, 221] can be employed along with our technique. Moreover, we assume that the 3PIP is provided either in the form of RTL or gate-level netlist. Our technique cannot be applied on encrypted or black-box IPs. For such Trojans, detection techniques based on hardware property checkers [190] can be used.

The Trojans causing leakage-of-information were not considered in our threat model. However, many Trojans belonging to this category can still be potentially captured by our technique. If a Trojan causes a change in a control bit in the processor, our method can potentially

detect it. For example, the Trojan PIC16F84-T300 [61] manipulates data lines to the external EEPROM. To achieve this, a select signal of a multiplexer needs to be switched, and this can be identified as an anomaly by our prediction model. In s35932-T100 Trojan, value of an internal signal is leaked through a test output pin. In this case, scan enable of part of a scan chain should be enabled, and this can be detected as an anomaly. In short, our method can also potentially detect leakage-of-information Trojans if the prediction model is trained with appropriate workloads expected in the field-of-operation. In short, several Trojan detection techniques should be employed in parallel to ensure safe in-field operation of a chip.

6.6 Summary

We have proposed a runtime Trojan identification method based on an anomaly detection technique by analyzing the workload profile of a small set of representative flip-flops in the untrusted design. The flip-flops are selected offline and the workload profile of flip-flops are analyzed online on a trusted core to make predictions on Trojan activation. The experiments conducted on four Trojan-inserted designs of Leon3 processor shows an accuracy of above 0.9 with no false positives by monitoring less than 0.5% of the flip-flops.

7 Conclusion

7.1 Summary and conclusion

With continuous technology scaling advancement and globalization of semiconductor design and fabrication stages, a series of vulnerabilities aggravate in reliability and security aspects that affect dependable operation of integrated circuits. The reliability mechanisms include dynamic aging of transistors, accelerated aging scenarios, radiation-induced soft errors and the security vulnerabilities include the possible inclusion of malicious hardware in the form of hardware Trojans from third-party vendors. Since these dependability attributes are dependent on workload and various runtime parameters, design-time solutions are inadequate. Because of the increasing design and runtime complexity, deterministic models are insufficient to guide runtime adaptation actions. In this scenario, runtime monitoring schemes are required to enable protection mechanisms against these dependability issues. The runtime monitoring for dependability is usually achieved in the state-of-the-art techniques with existing micro-architectural level monitors and performance counters. These monitors can only be accessed in a coarse-grained manner and the information available at the micro-architectural level is limited. Hence, new techniques are required to improve accuracy of prediction and also to enable fine-grained mitigation schemes.

This thesis tackles the challenges in ensuring dependability by identifying logic-level workload observables that can enable fine-grained runtime monitoring and higher prediction accuracy of dependability metrics. The modeling complexity of dependability mechanisms is dealt with appropriate machine learning methods that build prediction models based on different workload scenarios. We also propose several approaches to reduce the cost of implementation of these logic-level runtime monitors. The specific implementations of this approach for different reliability and security mechanisms are as follows.

- A machine-learning based runtime model was developed to predict aging-induced delay degradation of a circuit. The predicted value serves as a comparison metric to compare the aging rate of different workloads executed on a processor and can guide mitigation actions such as task mapping to mitigate aging effect. Our experiments on two embedded processors (Leon3 and OpenRISC 1200) show that the accuracy of prediction is high, even in the case when the prediction is based only on monitoring of a small number of flip-flops in the design.
- An accelerated aging scenario due to static aging effect occurring in specific workload phases in a design was identified. A low cost monitoring circuit to capture such corner cases was designed by identifying flip-flops that undergo correlated static aging phases. A mitigation technique based on a software subroutine was also developed with no additional circuit added. We report significant lifetime improvement by applying our proposed technique to two processor designs.
- We propose a technique to estimate the online soft-error vulnerability of memory arrays and logic cores based on the monitoring of a small set of flip-flops in the design. We exploit learning techniques to develop a predictive model for estimating vulnerability factor. This model is subsequently utilized at runtime as a software thread to evaluate online vulnerability factor. We also showed that the proposed prediction method is significantly more accurate when compared to prediction based on performance counters.

- An anomaly detection technique was proposed to detect the payload activation of a hardware Trojan by identifying anomalous workload signatures. The experiments conducted on Trojan-inserted designs of a processor show high Trojan detection accuracy with no false positives by monitoring a small percentage of the flip-flops in the design.

The results of this work show that the logic-level analysis of runtime data provides a new dimension for online monitoring and mitigation of dependability effects. We demonstrate the potential of machine learning techniques in workload compaction and representation to predict the rate of degradation mechanisms such as static and dynamic aging, transient soft error vulnerability, and also the anomalous behavior due to hardware Trojan activation. Our analysis demonstrates the possibility to extract relevant observables from lower levels of abstraction for low cost and fine-grained monitoring. This methodology can be reused to solve other workload-dependent reliability mechanisms such as voltage droop, and also other runtime security vulnerabilities.

Bibliography

[1] 40 Years of Microprocessor Trend Data. [Online]. Available: https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

[2] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[3] C. Berry, J. Warnock, J. Isakson, J. Badar, B. Bell, F. Malgioglio, G. Mayer, D. Hamid, J. Surprise, D. Wolpert *et al.*, "Ibm z14™: 14nm microprocessor for the next-generation mainframe," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 36–38.

[4] C. F. Webb, "Ibm z10: The next-generation mainframe microprocessor," *IEEE micro*, vol. 28, no. 2, pp. 19–29, 2008.

[5] K. Flamm, "Measuring moore's law: Evidence from price, cost, and quality indexes," National Bureau of Economic Research, Tech. Rep., 2018.

[6] R. D. Arnott, C. R. Harvey, and H. Markowitz, "A backtesting protocol in the era of machine learning," *Available at SSRN*, 2018.

[7] W. Arden, M. Brillouët, P. Cogez, M. Graef, B. Huizing, and R. Mahnkopf, "More-than-moore white paper," *Version*, vol. 2, p. 14, 2010.

[8] A. Avizienis, J.-C. Laprie, B. Randell *et al.*, *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[9] M. White, "Scaled cmos technology reliability users guide," Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space . . . , Tech. Rep., 2010.

[10] International Technology Roadmap for Semiconductors (ITRS), 2013. [Online]. Available: http://www.itrs2.net/2013-itrs.html

[11] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[12] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, 2010.

[13] E. Mintarno, V. Chandra, D. Pietromonaco, R. Aitken, and R. W. Dutton, "Workload dependent nbti and pbti analysis for a sub-45nm commercial microprocessor," in *2013 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2013, pp. 3A–1.

[14] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 2009, pp. 117–128.

[15] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, p. 6, 2016.

[16] B. Li and E. Leon, "Understanding power measurement capabilities on zaius power9," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2018.

[17] G. Xie, Z. Li, N. Yuan, R. Li, and K. Li, "Toward effective reliability requirement assurance for automotive functional safety," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 5, p. 65, 2018.

[18] F. Oboril and M. B. Tahoori, "Mttf-balanced pipeline design," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 270–275.

[19] L. Zhang and R. P. Dick, "Scheduled voltage scaling for increasing lifetime in the presence of nbti," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. IEEE Press, 2009, pp. 492–497.

[20] Z. Qi and M. R. Stan, "Nbti resilient circuits using adaptive body biasing," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*. ACM, 2008, pp. 285–290.

[21] T. R. Mück, Z. Ghaderi, N. D. Dutt, and E. Bozorgzadeh, "Exploiting heterogeneity for aging-aware load balancing in mobile platforms," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 1, pp. 25–35, 2017.

[22] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 7–18.

[23] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter, "Architecting for power management: The ibm® power7™ approach," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–11.

[24] A. Vijayan, A. Koneru, S. Kiamehr, K. Chakrabarty, and M. B. Tahoori, "Fine-grained aging-induced delay prediction based on the monitoring of run-time stress," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[25] A. Vijayan, S. Kiamehr, F. Oboril, K. Chakrabarty, and M. B. Tahoori, "Workload-aware static aging monitoring of timing-critical flip-flops," in *Asia and South Pacific Design Automation Conference (ASP-DAC), 2017*, pp. 176–181.

[26] A. Vijayan, S. Kiamehr, F. Oboril, K. Chakrabarty, and M. B. Tahoori, "Workload-aware static aging monitoring and mitigation of timing-critical flip-flops," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2098–2110, 2018.

[27] A. Vijayan, S. Kiamehr, M. Ebrahimi, K. Chakrabarty, and M. B. Tahoori, "Online soft-error vulnerability estimation for memory arrays and logic cores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 499–511, 2018.

[28] A. Vijayan, A. Koneru, M. Ebrahimit, K. Chakrabarty, and M. B. Tahoori, "Online soft-error vulnerability estimation for memory arrays," in *2016 IEEE 34th VLSI Test Symposium (VTS)*. IEEE, 2016, pp. 1–6.

[29] A. Vijayan, M. B. Tahoori, and K. Chakrabarty, "Runtime identification of hardware trojans by feature analysis on gate-level unstructured data and anomaly detection," ***under Review in*** *IEEE Transactions on Information, Foren- sics and Security*, 2019.

[30] J. Gaisler, "The leon processor user's manual," *Gaisler research*, 2001.

[31] D. Lampret, "Openrisc 1200 ip core specification," *September June*, 2001.

[32] A. R. Lebeck and D. A. Wood, "Cache profiling and the SPEC benchmarks: A case study," *Computer*, vol. 27, no. 10, pp. 15–26, 1994.

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

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] J.-C. Laprie, "Dependability: Basic concepts and terminology," in *Dependability: Basic Concepts and Terminology*. Springer, 1992, pp. 3–245.

[36] K. Sutaria, A. Ramkumar, R. Zhu, R. Rajveev, Y. Ma, and Y. Cao, "Bti-induced aging under random stress waveforms: Modeling, simulation and silicon validation," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

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

[38] J. Fang and S. S. Sapatnekar, "The impact of BTI variations on timing in digital logic circuits," *Transactions on Device and Materials Reliability*, vol. 13, no. 1, pp. 277–286, 2013.

[39] K. B. Sutaria, J. B. Velamala, C. H. Kim, T. Sato, and Y. Cao, "Aging statistics based on trapping/detrapping: Compact modeling and silicon validation," *Transactions on Device and Materials Reliability*, vol. 14, no. 2, pp. 607–615, 2014.

[40] H. Reisinger, T. Grasser, K. Ermisch, H. Nielen, W. Gustin, and C. Schlünder, "Understanding and modeling ac bti," in *International Reliability Physics Symposium (IRPS), 2011*, pp. 6A–1.

[41] J. B. Velamala, V. Ravi, and Y. Cao, "Failure diagnosis of asymmetric aging under NBTI," in *International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 428–433.

[42] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl, "A physical alpha-power law mosfet model," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 10, pp. 1410–1414, 1999.

[43] R. Chadha and J. Bhasker, *Static timing analysis for nanometer designs*. Springer US, 2009.

[44] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "An analytical model for negative bias temperature instability," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 493–496.

[45] T.-H. Kim, R. Persaud, and C. H. Kim, "Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 874–880, 2008.

[46] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *2009 Symposium on VLSI Circuits*. IEEE, 2009, pp. 112–113.

[47] S. Wang, J. Chen, and M. Tehranipoor, "Representative critical reliability paths for low-cost and accurate on-chip aging evaluation," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2012, pp. 736–741.

[48] F. Paterna, A. Acquaviva, and L. Benini, "Aging-aware energy-efficient workload allocation for mobile multimedia platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1489–1499, 2013.

[49] M. Basoglu, M. Orshansky, and M. Erez, "Nbti-aware dvfs: A new approach to saving energy and increasing processor lifetime," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. ACM, 2010, pp. 253–258.

[50] E. Mintarno, J. Skaf, R. Zheng, J. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, "Optimized self-tuning for circuit aging," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 586–591.

[51] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Adaptive techniques for overcoming performance degradation due to aging in digital circuits," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. IEEE Press, 2009, pp. 284–289.

[52] S. Mukherjee, *Architecture design for soft errors*. Morgan Kaufmann, 2011.

[53] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 243–247.

[54] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee, "Computing accurate avfs using ace analysis on performance models: A rebuttal," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 21–24, 2008.

[55] H. Asadi and M. B. Tahoori, "Soft error derating computation in sequential circuits," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 497–501.

[56] N. Miskov-Zivanov and D. Marculescu, "Mars-c: modeling and reduction of soft errors in combinational circuits," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 767–772.

[57] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online estimation of architectural vulnerability factor for soft errors," in *ISCA*. IEEE, 2008, pp. 341–352.

[58] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor," in *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2. IEEE Computer Society, 2004, p. 264.

[59] X. Ni, E. Meneses, N. Jain, and L. V. Kalé, "Acr: Automatic checkpoint/restart for soft and hard error protection," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 7.

[60] S. Bhunia and M. Tehranipoor, *The Hardware Trojan War*. Springer, 2018.

[61] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[62] M. Beaumont, B. Hopkins, and T. Newby, "Hardware trojans-prevention, detection, countermeasures (a literature review)," DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) COMMAND ..., Tech. Rep., 2011.

[63] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *2009 IEEE International high level design validation and test workshop*. IEEE, 2009, pp. 166–171.

[64] M. Abramovici and P. Bradley, "Integrated circuit security: new threats and solutions." *CSIIRW*, vol. 9, pp. 1–3, 2009.

[65] G. Bloom, B. Narahari, R. Simha, and J. Zambreno, "Providing secure execution environments with a last line of defense against trojan circuit attacks," *computers & security*, vol. 28, no. 7, pp. 660–669, 2009.

[66] T. M. Mitchell, *Machine learning, International Edition*, ser. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. [Online]. Available: http://www.worldcat.org/oclc/61321007

[67] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007.

[68] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer New York Inc., 2001.

[69] E. Mintarno, J. Skaf, R. Zheng, J. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, "Optimized Self-Tuning for Circuit Aging," in *Design, Automation and Test in Europe Conference and Exhibition*, 2010, pp. 586–591.

[70] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, "Bubble Razor: An Architecture-Independent Approach to Timing-Error Detection and Correction," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*. IEEE, 2012, pp. 488–490.

[71] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit Failure Prediction and its Application to Transistor Aging," in *IEEE VLSI Test Symposium*, 2007, pp. 277–286.

[72] K. Bowman *et al.*, "Circuit Techniques for Dynamic Variation Tolerance," in *ACM/IEEE Design Automation Conference*, 2009, pp. 4–7.

[73] N. Shah, R. Samanta, M. Zhang, J. Hu, and D. Walker, "Built-In Proactive Tuning System for Circuit Aging Resilience," in *IEEE Defect and Fault Tolerance in VLSI Systems*, 2008, pp. 96–104.

[74] S. Corbetta and W. Fornaciari, "NBTI Mitigation in Microprocessor Designs," in *Proceedings of the Great Lakes Symposium on VLSI*. ACM, 2012.

[75] T.-B. Chan, J. Sartori, P. Gupta, and R. Kumar, "On the Efficacy of NBTI Mitigation Techniques," in *Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 2011, pp. 1–6.

[76] H. Mostafa, M. Anis, and M. Elmasry, "Adaptive Body Bias for Reducing the Impacts of NBTI and Process Variations on 6T SRAM Cells," *Transactions on Circuits and Systems*, 2011.

[77] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging Analysis of Circuit Timing Considering NBTI and HCI," in *IEEE International On-Line Testing Symposium*, 2009, pp. 3–8.

[78] K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy, "Efficient Transistor-Level Sizing Technique Under Temporal Performance Degradation Due to NBTI," in *IEEE International Conference on Computer Design*, 2007, pp. 216–221.

[79] B. C. Paul, K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy, "Temporal Performance Degradation Under NBTI: Estimation and Design for Improved Reliability of Nanoscale Circuits," in *Design, Automation and Test in Europe Conference and Exhibition*, 2006, pp. 780–785.

[80] D. Sylvester, D. Blaauw, and E. Karl, "Elastic: An adaptive Self-Healing Architecture for Unpredictable Silicon," *Design & Test*, vol. 23, no. 6, pp. 484–490, 2006.

[81] K. Kang, S. P. Park, K. Roy, and M. A. Alam, "Estimation of Statistical Variation in Temporal NBTI Degradation and its Impact on Lifetime Circuit Performance," in *International Conference on Computer-Aided Design*, 2007.

[82] "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation."

[83] J. W. McPherson, "Reliability Challenges for 45 nm and Beyond," in *ACM/IEEE Design Automation Conference*, 2006, pp. 176–181.

[84] B. Kapoor, S. Hemmady, S. Verma, K. Roy, and M. A. D'Abreu, "Impact of SoC Power Management Techniques on Verification and Testing," in *International Symposium on Quality Electronic Design*, 2009, pp. 692–695.

[85] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Chandrakasan, and V. De, "Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Variations on Microprocessor Frequency and Leakage," *Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1396–1402, 2002.

[86] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Multi-Mechanism Reliability Modeling and Management in Dynamic Systems," *Transactions on VLSI Systems*, vol. 16, no. 4, pp. 476–487, 2008.

[87] B. Guenin, K. C. Gross, A. Gribok, and A. Urmanov, "A New Sensor Validation Technique for the Enhanced RAS of High End Servers," in *International Conference on Machine Learning: Models, Technologies and Applications*, 2004.

[88] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," in *ACM International Symposium on Computer architecture*, vol. 32, 2004, p. 276.

[89] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Lifetime Reliability: Toward an Architectural Solution," *Micro, IEEE*, vol. 25, pp. 70–80, 2005.

[90] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Exploiting Structural Duplication for Lifetime Reliability Enhancement," in *ACM International Symposium on Computer architecture*, June 2005, pp. 520–531.

[91] "Facelift: Hiding and slowing down aging in multicores."

[92] S. Gupta and S. S. Sapatnekar, "Employing circadian rhythms to enhance power and reliability," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 3, p. 38, 2013.

[93] B. Choi and Y. Shin, "Lookup Table-Based Adaptive Body Biasing of Multiple Macros," in *International Symposium on Quality Electronic Design*, 2007, pp. 533–538.

[94] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "NBTI-Aware Synthesis of Digital Circuits," in *ACM/IEEE Design Automation Conference*, 2007, pp. 370–375.

[95] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De, "Adaptive Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging," in *International Solid-State Circuits Conference*, 2007, pp. 292–604.

[96] E. Mintarno, J. Skaf, R. Zheng, J. B. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, "Self-Tuning for Maximized Lifetime Energy-Efficiency in the Presence of Circuit Aging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 760–773, 2011.

[97] G. Gammie, A. Wang, M. Chau, S. Gururajarao, R. Pitts, F. Jumel, S. Engel, P. Royannez, R. Lagerquist, H. Mair, J. Vaccani, G. Baldwin, K. Heragu, R. Mandal, M. Clinton, D. Arden, and U. Ko, "A 45 nm 3.5G Baseband-and-Multimedia Application Processor Using Adaptive Body-Bias and Ultra-Low-Power Techniques," in *International Solid-State Circuits Conference*, 2008, pp. 258–611.

[98] T. Fischer, J. Desai, B. Doyle, S. Naffziger, and B. Patella, "A 90-nm Variable Frequency Clock System for a Power-Managed Itanium Architecture Processor," *Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 218–228, 2006.

[99] C. Poirier, R. McGowen, C. Bostak, and S. Naffziger, "Power and Temperature Control on a 90-nm Itanium Family Processor," *Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 229–237, 2006.

[100] H. Mair, A. Wang, G. Gammie, D. Scott, P. Royannez, S. Gururajarao, M. Chau, R. Lagerquist, L. Ho, M. Basude, N. Culp, A. Sadate, D. Wilson, F. Dahan, J. Song, B. Carlson, and U. Ko, "A 65-nm Mobile Multimedia Applications Processor With an Adaptive Power Management Scheme to Compensate for Variations," in *IEEE Symposium on VLSI Circuits*, 2007.

[101] S. Wang, J. Chen, and M. Tehranipoor, "Representative Critical Reliability Paths for Low-Cost and Accurate On-Chip Aging Evaluation," in *IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 736–741.

[102] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori, "Aging-and variation-aware delay monitoring using representative critical path selection," *ACM Transactions on Design Automation of Electronic Systems*, 2015.

[103] J. Keane, X. Wang, D. Persaud, and C. H. Kim, "An all-in-one silicon odometer for separately monitoring HCI, BTI, and TDDB," *IEEE Journal of Solid-State Circuits*, 2010.

[104] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[105] A. Jain and D. Zongker, "Feature Selection: Evaluation, Application, and Small Sample Performance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153–158, Feb 1997.

[106] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson Correlation Coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[107] A. K. Jain, "Data Clustering: 50 years Beyond K-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[108] S. Gupta and S. S. Sapatnekar, "Variation-aware variable latency design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 5, pp. 1106–1117, 2014.

[109] D. Pelleg and A. Moore, "Accelerating Exact K-means Algorithms with Geometric Reasoning," in *ACM International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 277–281.

[110] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. J. Drake, L. Pesantez, T. Gloekler, J. A. Tierno, P. Bose, and A. Buyuktosunoglu, "Introducing the Adaptive Energy Management Features of the POWER7 Chip," *IEEE/ACM International Symposium on Microarchitecture*, vol. 31, no. 2, pp. 60–75, March 2011.

[111] "Nangate 45 nm open cell library v1.3," http://www.nangate.com.

[112] "ITRS," http://www.itrs.net.

[113] H. Amrouch, S. Mishra, V. van Santen, S. Mahapatra, and J. Henkel, "Impact of bti on dynamic and static power: From the physical to circuit level," in *International Reliability Physics Symposium (IRPS), 2017*, pp. CR–3.

[114] K. Ramakrishnan, X. Wu, N. Vijaykrishnan, and Y. Xie, "Comparative analysis of NBTI effects on low power and high performance flip-flops," in *International Conference on Computer Design (ICCD), 2008*, pp. 200–205.

[115] A. K. Deb, B. Vermeulen, and L. van Dijk, "Overview of health monitoring techniques for reliability." in *ERMAVSS@ DATE*, 2016, pp. 30–33.

[116] J. B. Velamala, K. B. Sutaria, T. Sato, and Y. Cao, "Aging statistics based on trapping/detrapping: Silicon evidence, modeling and long-term prediction," in *International Reliability Physics Symposium, (IRPS), 2012*, pp. 2F–2.

[117] D. R. Bild, R. P. Dick, and G. E. Bok, "Static NBTI reduction using internal node control," *Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 17, no. 4, p. 45, 2012.

[118] Y. Wang, X. Chen, W. Wang, Y. Cao, Y. Xie, and H. Yang, "Gate replacement techniques for simultaneous leakage and aging optimization," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 328–333.

[119] V. G. Rao and H. Mahmoodi, "Analysis of reliability of flip-flops under transistor aging effects in nano-scale cmos technology," in *International Conference on Computer Design (ICCD), 2011*, pp. 439–440.

[120] C. Nunes, P. F. Butzen, A. I. Reis, and R. P. Ribas, "BTI, HCI and TDDB aging impact in flip-flops," *Microelectronics Reliability*, vol. 53, no. 9, pp. 1355–1359, 2013.

[121] H. Abrishami, S. Hatami, and M. Pedram, "Multi-corner, energy-delay optimized, NBTI-aware flip-flop design," in *International Symposium on Quality Electronic Design (ISQED), 2010*, pp. 652–659.

[122] M. S. Golanbari, S. Kiamehr, M. B. Tahoori, and S. Nassif, "Analysis and optimization of flip-flops under process and runtime variations," in *International Symposium on Quality Electronic Design (ISQED), 2015*, pp. 191–196.

[123] Y. Wang, X. Chen, W. Wang, Y. Cao, Y. Xie, and H. Yang, "Leakage power and circuit aging cooptimization by gate replacement techniques," *Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 4, pp. 615–628, 2011.

[124] M. S. Golanbari, S. Kiamehr, M. Ebrahimi, and M. B. Tahoori, "Aging guardband reduction through selective flip-flop optimization," in *European Test Symposium (ETS), 2015*, 2015, pp. 1–6.

[125] C. Lin, Y.-T. Wang, S.-S. Yang, and Y.-L. Wu, "Analyzing the BTI Effect on Multi-bit Retention Registers," in *Intelligent Systems and Applications: Proceedings of the International Computer Symposium (ICS) Held at Taichung, Taiwan, December 12–14, 2014*, vol. 274. IOS Press, 2015, p. 269.

[126] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, 2001.

[127] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-based self-testing of embedded processors," *Transactions on Computers*, vol. 54, no. 4, pp. 461–475, 2005.

[128] L. Lai, V. Chandra, R. Aitken, and P. Gupta, "BTI-Gater: An aging-resilient clock gating methodology," *Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 2, pp. 180–189, 2014.

[129] M. Chen, V. Reddy, S. Krishnan, V. Srinivasan, and Y. Cao, "Asymmetric aging and workload sensitive bias temperature instability sensors," *IEEE Design & Test of Computers*, vol. 29, no. 5, pp. 18–26, 2012.

[130] M. Denais, V. Huard, C. Parthasarathy, G. Ribes, F. Perrier, N. Revil, and A. Bravaix, "Interface trap generation and hole trapping under NBTI and PBTI in advanced CMOS technology with a 2-nm gate oxide," *IEEE transactions on device and materials reliability*, vol. 4, no. 4, pp. 715–722, 2004.

[131] R. M. Karp, *Reducibility among combinatorial problems.* Springer, 1972.

[132] P. Bernardi, L. M. Ciganda, E. Sanchez, and M. S. Reorda, "MIHST: A hardware technique for embedded microprocessor functional on-line self-test," *Transactions on Computers*, vol. 63, no. 11, pp. 2760–2771, 2014.

[133] "SPARC," http://www.gaisler.com/doc/sparcv8.pdf.

[134] F. Oboril, J. Ewert, and M. B. Tahoori, "High-resolution online power monitoring for modern microprocessors," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 265–268.

[135] A. Cortex, "A9 processor technical reference manual," *Revisio r4p1 ed., ARM*, 2012.

[136] M. Ershov, S. Saxena, H. Karbasi, S. Winters, S. Minehane, J. Babcock, R. Lindley, P. Clifton, M. Redford, and A. Shibkov, "Dynamic recovery of negative bias temperature instability in p-type metal–oxide–semiconductor field-effect transistors," *Applied physics letters*, vol. 83, no. 8, pp. 1647–1649, 2003.

[137] B. Kaczer, T. Grasser, J. Franco, M. Toledano-Luque, P. J. Roussel, M. Cho, E. Simoen, and G. Groeseneken, "Recent trends in bias temperature instability," *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 29, no. 1, p. 01AB01, 2011.

[138] V. B. Kleeberger, M. Barke, C. Werner, D. Schmitt-Landsiedel, and U. Schlichtmann, "A compact model for NBTI degradation and recovery under use-profile variations and its application to aging analysis of digital integrated circuits," *Microelectronics Reliability*, vol. 54, no. 6, pp. 1083–1089, 2014.

[139] L. P. U. Manual, "Gaisler research," *Version*, vol. 1, p. 23, 2004.

[140] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "Fabscalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, 2011, pp. 11–22.

[141] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *WWC*, 2001, pp. 3–14.

[142] A. Dixit, , and A. Wood, "The impact of new technology on soft error rates," in *IRPS*, 2011, pp. 5B–4.

[143] M. Ebrahimi, A. Evans, M. B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi, "Comprehensive analysis of sequential and combinational soft errors in an embedded processor," *TCAD*, 2015.

[144] L. G. Szafaryn, B. H. Meyer, and K. Skadron, "Evaluating overheads of multibit soft-error protection in the processor core," *IEEE Micro*, no. 4, pp. 56–65, 2013.

[145] S. Mitra, T. Karnik, N. Seifert, and M. Zhang, "Logic soft errors in sub-65nm technologies design and cad challenges," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 2–4.

[146] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *International Symposium on Microarchitecture*, 2003.

[147] S. Mirkhani, B. Samynathan, and J. A. Abraham, "In-depth soft error vulnerability analysis using synthetic benchmarks," in *VTS*, April 2015, pp. 1–6.

[148] C.-Y. Cher, K. P. Muller, R. A. Haring, D. L. Satterfield, T. E. Musta, T. M. Gooding, K. D. Davis, M. B. Dombrowa, G. V. Kopcsay, R. M. Senger *et al.*, "Soft error resiliency characterization and improvement on ibm bluegene/q processor using accelerated proton irradiation," in *ITC*, 2014, pp. 1–6.

[149] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *ISCA*, 2005, pp. 532–543.

[150] M. Ebrahimi, L. Chen, H. Asadi, and M. Tahoori, "Class: Combined logic and architectural soft error sensitivity analysis," in *Asia and South Pacific Design Automation Conference*, 2013, pp. 601–607.

[151] M. Fazeli, S. N. Ahmadian, S. G. Miremadi, H. Asadi, and M. B. Tahoori, "Soft error rate estimation of digital circuits in the presence of multiple event transients (mets)," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[152] S. Raasch, A. Biswas, J. Stephan, P. Racunas, and J. Emer, "A fast and accurate analytical technique to compute the avf of sequential bits in a processor," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 738–749.

[153] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 516–527, 2007.

[154] A. Biswas, N. Soundararajan, S. S. Mukherjee, and S. Gurumurthi, "Quantized AVF: A means of capturing vulnerability variations over small windows of time," in *SELSE*, 2009.

[155] N. K. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for bounding vulnerabilities of processor structures," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 506–515, 2007.

[156] X. Fu, J. Poe, T. Li, and J. A. Fortes, "Characterizing microarchitecture soft error vulnerability phase behavior," in *MASCOTS*. IEEE, 2006, pp. 147–155.

[157] Y. Cheng, Y. Wang, Z. Xing, and M. Zhang, "Characterizing time-varying behavior and predictability of cache avf," in *INCoS*. IEEE, 2011, pp. 720–725.

[158] D. Alexandrescu, "A comprehensive soft error analysis methodology for SoCs/ASICs memory instances," in *IOLTS*, 2011, pp. 175–176.

[159] C. Hu and S. Zain, "NSEU mitigation in avionics applications," *Xilinx Application Note XAPP1073*, pp. 1–12, 2011.

[160] M. Ebrahimi, M. Rashvand, F. Kaddachi, M. B. Tahoori, and G. Di Natale, "Revisiting software-based soft error mitigation techniques via accurate error generation and propagation models," in *On-Line Testing and Robust System Design (IOLTS), 2016 IEEE 22nd International Symposium on*. IEEE, 2016, pp. 66–71.

[161] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose *et al.*, "CLEAR: Cross-layer exploration for architecting resilience-combining hardware and software techniques to tolerate soft errors in processor cores," *arXiv preprint arXiv:1604.03062*, 2016.

[162] B. Gill, N. Seifert, and V. Zia, "Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node," in *International Reliability Physics Symposium*, 2009, pp. 199–205.

[163] N. Seifert, B. Gill, S. Jahinuzzaman, J. Basile, V. Ambrose, Q. Shi, R. Allmon, and A. Bramnik, "Soft error susceptibilities of 22 nm tri-gate devices," *IEEE Transactions on Nuclear Science*, vol. 59, no. 6, pp. 2666–2673, 2012.

[164] P. Ramachandran, P. Kudva+, J. Kellington, J. Schumann, and P. Sanda, "Statistical fault injection," in *DSN*, June 2008, pp. 122–127.

[165] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi, "SCFIT: A FPGA-based Fault Injection Technique for SEU Fault Model," in *DATE*, 2012, pp. 586–589.

[166] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: quantified error and confidence," in *DATE*, 2009, pp. 502–506.

[167] Cho, Hyungmin and Mirkhani, Shahrzad and Cher, Chen-Yong and Abraham, Jacob A and Mitra, Subhasish, "Quantitative evaluation of soft error injection techniques for robust system design," in *Design Automation Conference*, 2013.

[168] H. Asadi and M. B. Tahoori, "Soft error modeling and protection for sequential elements," in *International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 463–471.

[169] D. Holcomb, W. Li, and S. A. Seshia, "Design as you see FIT: system-level soft error analysis of sequential circuits," in *Design, Automation and Test in Europe (DATE)*, 2009.

[170] Cho, Hyungmin and Cher, Chen-Yong and Shepherd, Thomas and Mitra, Subhasish, "Understanding soft errors in uncore components," in *Design Automation Conference (DAC)* , 2015.

[171] Mirkhani, Shahrzad and Mitra, Subhasish and Cher, Chen-Yong and Abraham, Jacob, "Efficient soft error vulnerability estimation of complex designs," in *Design, Automation & Test in Europe (DATE)*, 2015.

[172] Gold, Brian T and Ferdman, Michael and Falsafi, Babak and Mai, Ken, "Mitigating multi-bit soft errors in L1 caches using last-store prediction," in *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, 2007.

[173] B. Li, L. Duan, and L. Peng, "Efficient microarchitectural vulnerabilities prediction using boosted regression trees and patient rule inductions," *Transactions on Computers*, vol. 59, no. 5, pp. 593–607, 2010.

[174] Ma, An-Guo and Cheng, Yu and Xing, Zuo-Cheng, "Accurate and simplified prediction of avf for delay and energy efficient cache design," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 504–519, 2011.

[175] Farahani, Bahar and Safari, Saeed, "A cross-layer approach to online adaptive reliability prediction of transient faults," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)* , 2015.

[176] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. Springer series in statistics Springer, Berlin, 2001, vol. 1.

[177] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[178] C. Li, *A Gentle Introduction to Gradient Boosting*, (accessed November 6, 2016), http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf.

[179] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance avf analysis," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 461–472.

[180] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, "Hardware trojan horse detection using gate-level characterization," in *ACM/IEEE Design Automation Conference (DAC), 2009.*, pp. 688–693.

[181] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Multiple-parameter side-channel analysis: A non-invasive hardware trojan detection approach," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2010*, pp. 13–18.

[182] N. Veeranna and B. C. Schafer, "Trust filter: Runtime hardware trojan detection in behavioral MPSoCs," *Journal of Hardware and Systems Security*, pp. 1–12, 2017.

[183] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a comprehensive and systematic classification of hardware trojans," in *Proceedings of International Symposium on Circuits and Systems (ISCAS),*. IEEE, 2010, pp. 1871–1874.

[184] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M. S. Hsiao, J. Plusquellic, and M. Tehranipoor, "Protection against hardware trojan attacks: Towards a comprehensive solution," *IEEE Design & Test*, vol. 30, no. 3, pp. 6–17, 2013.

[185] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Design Automation Conference (DAC)*. ACM, 2014, pp. 1–6.

[186] J. Zhang, H. Yu, and Q. Xu, "Htoutlier: Hardware trojan detection with side-channel signature outlier identification," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2012*, 2012, pp. 55–58.

[187] X. Zhang and M. Tehranipoor, "Ron: An on-chip ring oscillator network for hardware trojan detection," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[188] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware trojans with gate-level information-flow tracking," *Computer*, vol. 49, no. 8, pp. 44–52, 2016.

[189] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ics: Problem analysis and detection scheme," in *Proceedings of the conference on Design, automation and test in Europe (DATE)*. ACM, 2008, pp. 1362–1365.

[190] X. T. Ngo, J.-L. Danger, S. Guilley, Z. Najm, and O. Emery, "Hardware property checker for run-time hardware trojan detection," in *IEEE European Conference on Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4.

[191] M. Bilzor, T. Huffmire, C. Irvine, and T. Levin, "Evaluating security requirements in a general-purpose processor by combining assertion checkers with code coverage," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2012*, pp. 49–54.

[192] J. Dubeuf, D. Hély, and R. Karri, "Run-time detection of hardware trojans: The processor protection unit," in *European Test Symposium (ETS)*. IEEE, 2013, pp. 1–6.

[193] D. Forte, C. Bao, and A. Srivastava, "Temperature tracking: An innovative run-time approach for hardware trojan detection," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013*, pp. 532–539.

[194] J. Li and J. Lach, "At-speed delay characterization for ic authentication and trojan horse detection," in *IEEE International Workshop on Hardware-Oriented Security and Trust, 2008.*, pp. 8–14.

[195] R. Elnaggar, K. Chakrabarty, and M. Tahoori, "Run-time hardware trojan detection using performance counters," in *Proceedings of the IEEE International Test Conference*, 2017.

[196] Y. Jin, M. Maniatakos, and Y. Makris, "Exposing vulnerabilities of untrusted computing platforms," in *International Conference on Computer Design (ICCD), 2012*. IEEE, 2012, pp. 131–134.

[197] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, 2012.

[198] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 697–708.

[199] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, 2017.

[200] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE Symposium on Security and Privacy (SP), 2010*. IEEE, 2010, pp. 159–172.

[201] S. K. Haider, C. Jin, M. Ahmad, D. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans detection," *Transactions on Dependable and Secure Computing*, 2017.

[202] C. Bao, D. Forte, and A. Srivastava, "On application of one-class svm to reverse engineering-based hardware trojan detection," in *International Symposium on Quality Electronic Design (ISQED), 2014*. IEEE, 2014, pp. 47–54.

[203] D. Jap, W. He, and S. Bhasin, "Supervised and unsupervised machine learning for side-channel based trojan detection," *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2016.

[204] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists based on machine learning," in *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), 2016*, 2016, pp. 203–206.

[205] S. Saha, R. S. Chakraborty, S. S. Nuthakki, D. Mukhopadhyay *et al.*, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2015, pp. 577–596.

[206] A. Kulkarni, Y. Pino, and T. Mohsenin, "Svm-based real-time hardware trojan detection for many-core platform," in *International Symposium on Quality Electronic Design (ISQED), 2016*. IEEE, 2016, pp. 362–367.

[207] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware trojans using machine learning in frequency domain," in *IEEE Global Conference on Consumer Electronics (GCCE), 2015*, 2015, pp. 185–186.

[208] H. Zhao, L. Kwiat, K. A. Kwiat, C. A. Kamhoua, and L. Njilla, "Applying chaos theory for runtime hardware trojan monitoring and detection," *Transactions on Dependable and Secure Computing*, 2018.

[209] L. Pyrgas, F. Pirpilidis, A. Panayiotarou, and P. Kitsos, "Thermal sensor based hardware trojan detection in fpgas," in *Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 268–273.

[210] J. Oberg, T. Sherwood, and R. Kastner, "Eliminating timing information flows in a mix-trusted system-on-chip," *IEEE Design & Test*, vol. 30, no. 2, pp. 55–62, 2013.

[211] D. M. Shila, V. Venugopalan, and C. D. Patterson, "Fides: Enhancing trust in reconfigurable based hardware systems," in *IEEE High Performance Extreme Computing Conference (HPEC), 2015*, 2015, pp. 1–7.

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

[213] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *IEEE International Conference on Data Mining, ICDM.*, 2008, pp. 413–422.

[214] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 485–492.

[215] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, p. 3, 2012.

[216] A. Vijayan, "Trojan-inserted Leon3 designs," https://github.com/arunatkit/Leon_Trojans, 2018, [Online; accessed 24-July-2018].

[217] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *International Conference on Computer Design (ICCD), 2013*. IEEE, pp. 471–474.

[218] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, pp. 1–18, 2017.

[219] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[220] P. J. Rousseeuw and K. V. Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.

[221] F. Karabacak, U. Ogras, and S. Ozev, "Remote detection of unauthorized activity via spectral analysis," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 6, p. 81, 2018.