UNIVERSITY OF CALIFORNIA,
IRVINE

Aging-induced Performance Degradation: Monitoring and Mitigation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Zana Ghaderi

Dissertation Committee:
Professor Nader Bagherzadeh, Chair
Professor Isaac Scherson
Professor Jean-Luc Gaudiot

2017

DEDICATION

To my family; Zaher, Maryam, Dana, and Arina

"Listen," said Pugatschew, with a sort of wild enthusiasm, "I shall tell you a fable which I have heard in my infancy, from an old Kalmouk woman. Once upon a time the eagle asked the raven:–'tell me my good raven how is it that your race lives to the age of three hundred years, whilst we eagles only live to the age of thirty three?'–' The reason, my good sir, is, because you devour your prey alive whilst we live upon dead carcasses.' The eagle thought, well, I shall try that too. They thereupon went out together and found a dead horse. They alighted upon it. The raven began to feed and enjoyed it. The eagle tried several morsels, then shook his pinions, and said to the raven:'–Nay, brother raven, rather than live three hundred years upon dead horses, I once feast upon a live victim, and then let fate do its worst.
Alexander Pushkin, The Captain's Daughter. Ch. XI, P. 117. 1836.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ACKNOWLEDGMENTS

I would like to thank my advisor professor Nader Bagherzadeh, who gave me the opportunity to finish my thesis successfuly. I also would like to express my gratitude to my committee members professor Isaac D. Scherson and professor Jean-luc gaudiot for the comments on my thesis.

I owe everything to my family. I would like to express my everlasting gratitude to my parents, Zaher Ghaderi-Baneh and Maryam Jahangiri for their believe in me and their perpetual support. I dedicate very special thanks to my lovely brother and sister Dana Ghaderi and Arina Ghaderi, for their endless kindness and their special role in my life.

Many thanks to my friends and coauthors for their thoughts, helps, and companionship. I am thankful to my colleagues Tiago R. Muck, Ayed Alqahtani, Mohammad Ebrahimi and Ahmad Albaqsami.

I am very fortunate to have the supprts of great friends. I would specifically like to thank Marzieh Tousi, Kevin Mcquown, Mohmmad Kamandi, Mahsa Darvish, Ahmad Shirazi, Esmaeil Babakrpour, Ayed Alqahtani, Tiago Muck, Hamid Mirzaei, Saba Mohammadi, and Isaac Ramsey.

I would like to express my speciall thanks to my masters degree advisor at Sharif University of Technology, professor Seyed-Ghassem Miremadi who I lost him lately.

I wish to thank NSF for the financial support. Chapter 4 contains material taken from "Aging-aware high-level physical planning for reconfigurable systems" which appears in ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC) 2016, which is partially supported by NSF.

CURRICULUM VITAE

Zana Ghaderi

EDUCATION

Doctor of Philosophy in Computer Science — **Sep. 2013 - May 2017**
University of California Irvine — *Irvine, California*

Master of Science in Computer Engineering — **Sep. 2010 - Sep. 2012**
Sharif University of Technology — *Tehran*

Bachelor of Science in Computer Engineering — **Sep. 2003 - Sep. 2008**
University of Science and Technology — *Tehran*

RESEARCH EXPERIENCE

Graduate Research Assistant — **2013–2017**
University of California, Irvine — *Irvine, California*

Graduate Research Assistant — **2011–2013**
Sharif University of Technology — *Tehran*

Undergraduate Research Assistant — **Jan. 2008–Sep. 2008**
University of Science and Technology — *Tehran*

TEACHING EXPERIENCE

Teaching Assistant — **Sep. 2015–March 2017**
University of California, Irvine — *Irvine, California*

Teaching Assistant — **Jan. 2012–Jun. 2012**
Sharif University of Technology — *Tehran*

WORK EXPERIENCE

R&D Engineering Intern — **Jun. 2015–Sep. 2015**
Abreezio LLC. (Acquired by Qualcomm) — *San Diego, California*

AWARDS

Design and Automation Conference (DAC) Young Student Award **2015**
San Francisco, California

ICS Dean's Fellowship for 4 years of support **2013–2017**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

AROMa: Aging-aware adaptive routing algorithm and online monitoring in 3D NoCs — **Under review-2017**

IEEE Transactions on Parallel and Distributed Systems

STABLE: Stress-aware boolean matching to mitigate BTI-induced SNM reduction in SRAM-based FPGAs — **Under review-2017**

IEEE Transactions on Computers

SENSIBLE: A highly scalable sensor design for path-based age monitoring in FPGAs — **2017**

IEEE Transactions on Computers

Exploiting heterogeneity for aging-aware load balancing in mobile platforms — **2017**

IEEE Transactions on Multi-scale Computing Systems

HAFTA: Highly availavle fault-tolerant architecture to protoect SRAM-based FPGAs agains multiple bit us-pets — **2013**

IEEE Transactions on Device and Materials Reliability

REFEREED CONFERENCE PUBLICATIONS

SLAPUF: Highly scalable low area delay-based physical unclonable function — **Under review-2017**

IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Online monitoring and adaptive routing for aging mitigation in NoCs — **March 2017**

IEEE/ACM Conference on Design, Automation & Test in Europe (DATE)

Path selection and sensor insertion flow for age monitoring in FPGAs — **March 2016**

IEEE/ACM Conference on Design, Automation & Test in Europe (DATE)

Harvesting-aware adaptive energy management in solar-powered embedded systems — **March 2016**

IEEE International Symposium on Quality Electronic Design (ISQED)

Aging-aware high-level physical planning for reconfigurable systems **January 2016**

IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)

A non-intrusive portable fault injection framework to assess reliability of FPGA-based designs **September 2013**

IEEE International Conference on Field-Programmable Technology (FPT)

ABSTRACT OF THE DISSERTATION

Aging-induced Performance Degradation: Monitoring and Mitigation

By

Zana Ghaderi

Doctor of Philosophy in Computer Science

University of California, Irvine, 2017

Professor Nader Bagherzadeh, Chair

One of the fundamental challenges to the performance gain in advanced semiconductor technology is aging-induced delay degradation of transistors, which consequently increases the logic gates delays and eventually critical paths delays. Hence, designers have to add significant time margin as guardband to the main critical path, which imposes considerable performance degradation to the system. Temperature and stress (or usage) are the major sources of transistor's aging, which vary for different applications and are highly workload dependent. This thesis covers challenges and opportunities in monitoring aging, its effects and methods to combat the imposed performance and lifetime degradation in nanometer scales semiconductor platforms.

We devise methods to monitor and mitigate aging in computing platforms ranging from the conventional reconfigurable architectures to the contemporary 3D network-on-chips and many-core systems. To monitor aging-induced delay degradation on critical paths, we proposed SENSIBLE, a highly scalable aging sensor design that can help system-level designers to detect aging and react accordingly. Additionally, we proposed an applicaiton-dependant filtering methodology to select *Representative Critical Paths* (RCPs) among a large pool of critical paths for aging monitoring in reconfigurable architectures. Furthermore, two proactive methods are presented to mitigate aging impacts on application's critical paths and

SRAM cells in reconfigurable architectures. The former is a high-level physical planning coupled with a reconfiguration policy and the latter is STABLE, a post-synthesis stress-aware Boolean matching technique. To mitigate and monitor aging on *Network on Chip* (NoC) components in both 2D and 3D IC designs we proposed AROMa, which is an aging-aware adaptive routing algorithm. To this end, we devised *Centralized Aging Table* (CAT) to convert transistor level aging phenomenon to the workloads' behavior in NoC-based many-core systems. Finally, an aging-aware task mapping, ADAMANT, is proposed to balance aging in many-core heterogeneous architectures' components.

Chapter 1

Introduction

Performance degradation due to runtime (temporal) variations is a fundamental challenge in the advanced semiconductor technology [2]. Extremely low feature size in CMOS technology leads to higher temperature, which as time passes, increases the threshold voltage (V_{th}) of under stress transistors. Hence, these delay degradations at the transistor level increase critical paths' delays, manifesting itself as performance degradation at the system level. The aforementioned phenomenon is the so called transistor aging or briefly aging as a new reliability concern in advanced silicon technology. Accordingly, designers have to add aging guradband to the circuit's main critical path which imposes significant performance overhead [2, 51, 132, 149, 155]. Consequently, aging imperils the performance gain from the movement toward many-core systems and 3D IC design in nano-era technology.

1.1 Aging background

Reliability is a perpetual concern in nanoscale circuits. Among various reliability challenges aging can significantly impact the dependable operation of applications. Aging can cause

timing failure in the running application and/or considerably diminish the operational lifetime of the circuit. Different aging mechanisms are classified as following [2]:

- *Bias Temperature Instability* (BTI)
- *Hot Carrier Injection* (HCI)
- *Time Dependent Dielectric Breakdown* (TDDB)
- *Electromigration* (EM)

BTI

BTI is a two phases mechanism which includes *stress* phase and partial *recovery* phase. Negative BTI (NBTI) happens in PMOS transistors and Positive BTI (PBTI) happens in NMOS transistors. The transistors cross sections during NBTI and PBTI mechanisms are shown in Fig.1.1 and Fig.1.2, respectively [86, 91, 132, 143]. As shown in Fig.1.1.a, when the PMOS transistor is ON, the stress voltage brakes the covalent bound of Si-H at interface; this process is so called *reaction*. The separated hydrogen atoms combine to form H2, which diffuses toward the gate of the transistor. These broken Si-H bonds generate positively charged traps for holes and leads to increase in transistor's V_{th}. Interestingly, as illustrated in Fig.1.1.b, when the PMOS switches off and stress is removed the recovery phase starts where some of the traps are released and the broken Si-H bounds heal. PBTI in NMOS transistors was not a challenge in silicon dioxide dielectrics. However, as the high-k dielectric stacks started from the 32 nm technology node, its contribution to the aging also becomes considerable [164, 165]. Similarly, Fig.1.2 illustrates the NMOS transistor cross section during PBTI mechanism. In contrary to NBTI, as shown in Fig.1.2.a, PBTI is generally happens due to the electrons trapping within the gate dielectric or at the interface when the NMOS transistor is ON. During its recovery phase, these electrons are released and partially

go back to the channel which leads to partial improve in transistor's V_{th}. Temperature accelerates BTI mechanism.

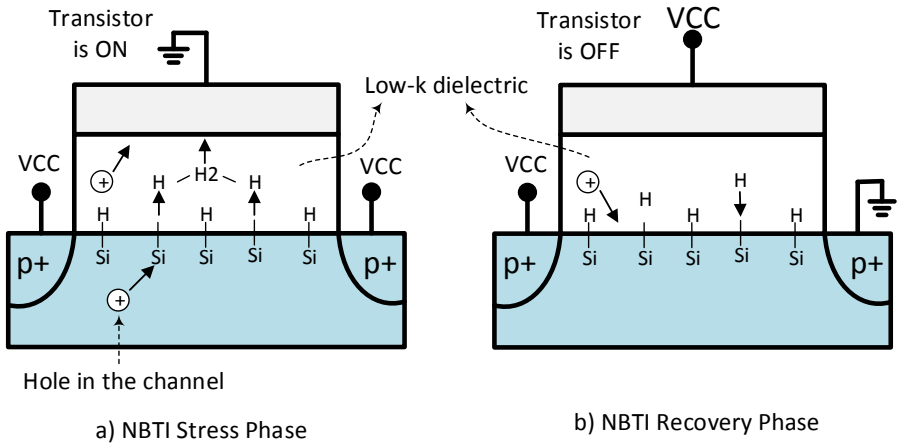


Figure 1.1: PMOS transistor cross section illustration during NBTI.

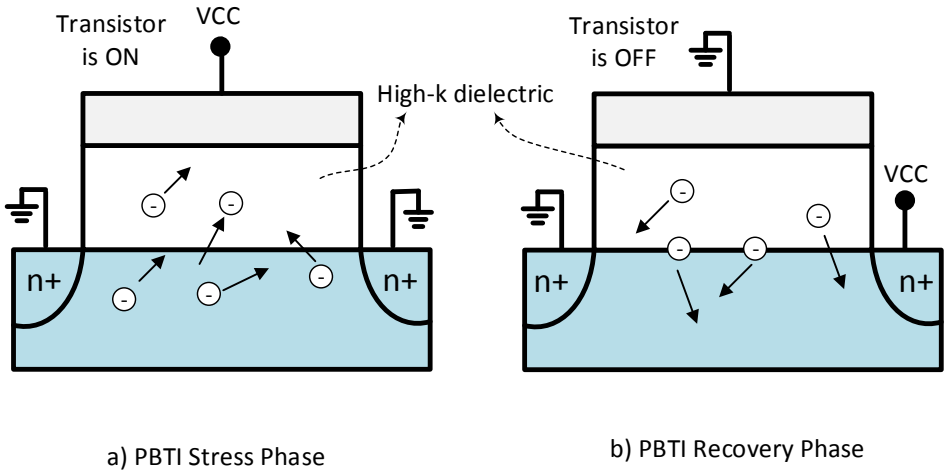


Figure 1.2: NMOS transistor cross section illustration during PBTI.

HCI

HCI mechanism is more prominent in NMOS transistors. The transistor cross section during HCI mechanism is shown in Fig.1.3. When the transistor switches to turn ON the high energy

carriers, that are called hot-carriers, collide with other atoms and carriers in the transistor's channel. If their energies are higher than impact-ionization threshold then electron-hole pairs are generated. Some of these carriers are injected to the gate oxide and as a result V_{th} increases [91, 119, 145].

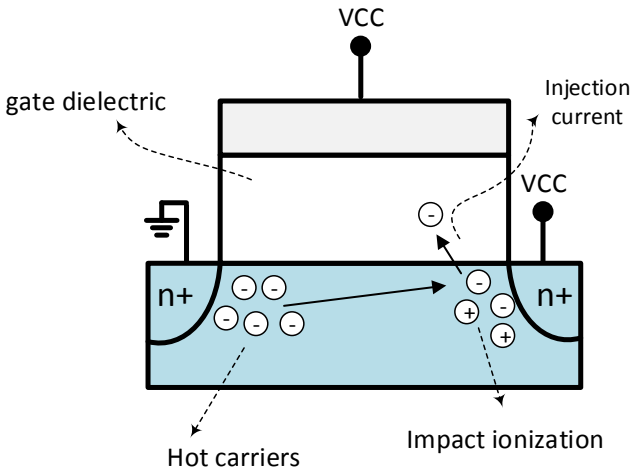


Figure 1.3: NMOS transistor cross section illustration during HCI.

TDDB

TDDB mechanism occurs when the voltage across gate generates traps in the dielectric. Accumulation of these carriers increases the electron density in the transistor dielectric. As shown in Fig.1.4 when the electron density is high a breakdown path (i.e. conductive path) between the transistor's gate and channel is generated [46, 91, 94]. The generated breakdown path increases leakage power of the transistor, shifts its V_{th}, and eventually fails its correct functionality [94, 158].

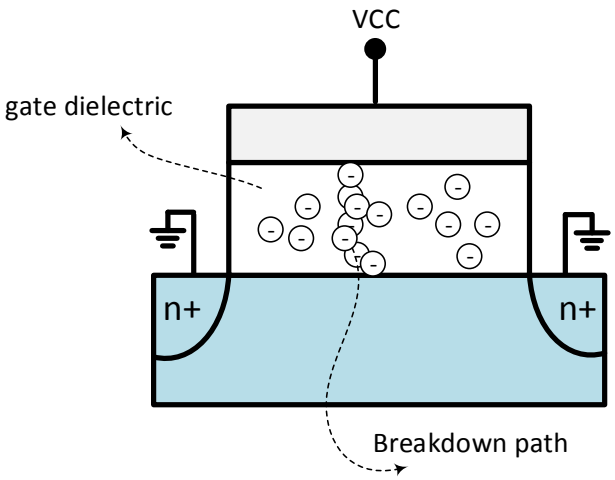


Figure 1.4: NMOS transistor cross section illustration during TDDB.

EM

EM occurs along conductors in the direction of electron flows which cause to diffuse metal atoms. For example, the metal connection between transistors suffers from EM mechanism. The mass migration of metal atoms along with the electrons depletes a part of conductor and makes it thinner. Consequently, thinning of the metal increases the resistance of the connection and eventually leads to an open circuit in the wire [27, 36].

Aging sources and design time solution

The main source of aforementioned aging mechanisms in transistors or wires is usage, or the so called stress. For example, when the transistor is ON or is switching or a current flows inside a wire stress happens. Aging rate increases in high temperatures. BTI and HCI that increase V_{th} and switching delay of transistors are the dominant aging mechanisms [2, 45, 64, 65, 105]. In [18], it is shown that BTI and HCI impacts are 15% and 5%, respectively in 65nm technology. This shows that BTI is the dominant mechanism and its impact is

three times higher than HCI. These get worse for technologies beyond 65nm and the timing variation are expected to aggravate with technology scaling [2, 35]. Furthermore, In [115], BTI and HCI impacts are reported 9.4% and 8.5%, respectively. Additionally, [17] reports about 10% degradation for 32nm in 3 years due to BTI. Based on these reports, in this work, we assume BTI and HCI in worst case degrades critical paths by 12% and 4% in 3 years, respectively, given a fixed temperature of $380K$. In this dissertation, we study and consider the impact of BTI and HCI aging mechanisms on delay of a circuit (i.e. performance).

The path with the highest delay in a circuit (i.e. an implemented application) is the main critical path, which determines the circuit's clock frequency or performance. Aging increases the critical paths' delays which demands for higher period of clock (or lower clock frequency). Sequential elements (e.g. Flip-flops) are connected at the end of each path to store the state of the circuit. Any variation on path delay because of aging causes timing violation and error which results in incorrect state of the circuit. This could lead to failure at the system level. Therefore, designers have to add pessimistic timing margins to the circuit's critical paths (i.e. clock frequency) as guardbands. In addition, dynamic variations at run time due to running workloads' variations lead to distinct stress as well as temperature between resources. For that, designers must consider the worst case scenario due to the unpredictability of running workloads' behaviors at runtime on the circuit. Besides, due to process variation, temperature, and voltage variation other guardbands are required to be added. These timing guardbands' lengths are increasing rapidly as technology advances, which account for almost 40% of the target performance [2]. For example, 20% guardband is required to avoid failure due to aging in 65nm technology within the first 10 years [51, 155]. Guardbanding constraints those advantages that are because of technology scaling [2].

The static variations (i.e. process variation) effects can be mitigated by pre-silicon and post-silicon tuning techniques. While aging as a temporal variation requires after fabrication solutions, during runtime and at system level to compensate for the performance degradation

related to guardbands and help designers to avoid adding considerable guardbands to the system at fabrication time. In all, not only design time analysis and adding guardbands but also runtime techniques can help to cope with aging effects.

Since stress and temperature change broadly for different transistors in a circuit, aging rates of logic gates and the paths along them are not balanced [51, 66, 153]. Therefore, the aging-induced delay increments of different paths vary, which may cause generation of new critical paths by converting non-critical path to critical path. The imbalance aging rate becomes more severe in many-core systems and 3D ICs, where temperature is a fundamental challenge. Beside that, various components (e.g. cores, interconnection infrastructures, and etc.) inside devices experience different temperatures and stresses. Similarly, reconfigurable architectures components where not only the running application on them but also the implemented logics among them change. This also causes imbalance and unfair aging of components.

1.2 Selected platforms and motivation

Runtime reconfigurable architectures enable dynamic adaptation of changing workloads, that allows to optimize area, performance and power [61, 67]. They comprise general purpose cores and a reconfigurable architecture which is implemented as SRAM-based Field Programmable Gate Array (FPGA) to allow swift runtime reconfigurations. Fig. 1.5 illustrates the contemporary island style SRAM-based reconfigurable architecture. The ever-increasing usage of FPGAs in mainstream applications such as acceleration in data-centers as well as safety critical application such as space, automotive, and medical made us to investigate aging in such a platform.

As shown in Fig. 1.5 the application's logics are implemented in tiles which are known as

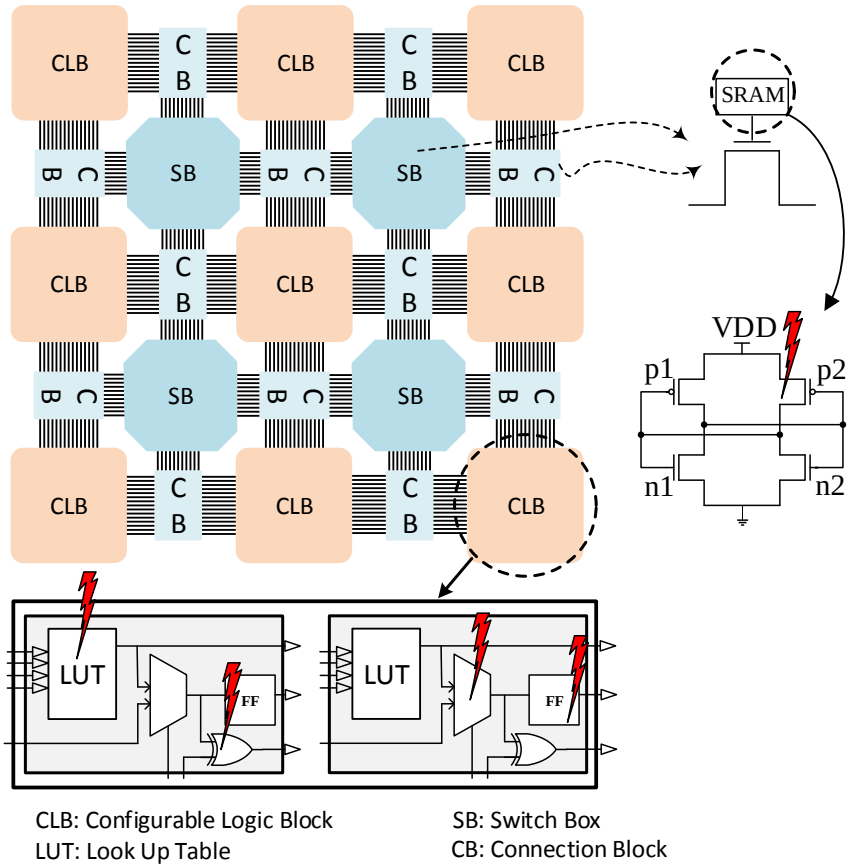


Figure 1.5: Island style SRAM-based reconfigurable architecture.

Configurable Logic Blocks (CLBs) is composed of *Look-up-Tables* (LUTs), *Flip-Flops* (FFs), Carry chain, logic gates and etc. These components are connected to each other through *Connection Blocks* (CBs) and also connected to other CLBs through *Switch Boxes* (SB). SRAM cells inside these components are configured to implement the desired logics and their required routings. Fig. 1.5 shows that all the components in an FPGA suffer from aging-induced delay degradation (red thunders). For example, the delay of implemented critical paths along a reconfigurable architecture components can increase due to aging mechanisms such as BTI or HCI. Additionally, aging impact on SRAM cells in a reconfigurable architecture manifests itself as the SRAM's *Static Noise Margin* (SNM) reduction. SNM is the SRAM cell reliability measurement. Hence, SNM reduction in an SRAM cell decreases its

stability and increases its *Soft Error Rate* (SER).

There have been noticeable numbers of studies in the literature about aging in reconfigurable architectures [6, 87, 88, 123, 138, 142, 167, 169]. These techniques tried to mitigate aging through changes in placement or routing of the implemented design on FPGAs. It is required to be noted that temperature map of the implemented circuit cannot change drastically by these local changes. Therefore, the local tuning not only cannot change the aging map of the implemented design but also induces performance overheads due to stringent constraints of placed and routed design. Furthermore, these methods only focused on the performance degradation of implemented design on an FPGAs and the impact of aging on SRAM cells of reconfigurable architectures is ignored.

Additionally, the movement toward many-core systems and the demand for lower area, power, and scalability as well as higher performance leads computer architects to 3D IC and NoC designs [47]. Furthermore, the broad diversity of application requires many-core heterogeneous architectures, as well. Fig. 1.6 illustrates the system evolution toward many core systems and their on-chip interconnections. It can be seen that routers are the main components to connect cores in a platform with 80 cores [47] which can suffer from aging. Aging-induced performance degradation in routers endangers the reliability and scalability of such platforms. Furthermore, cores in heterogeneous architectures have different clock frequencies which increases the chance of higher imbalanced aging among cores.

All of these lead us to investigate aging in NoCs, as a promising solution for scalable and high performance many-core systems, and heterogeneous architectures. It is shown in Fig. 1.6 that all the components in a many core system such as cores, routers, and links are susceptible to aging-induced performance degradation. It can be concluded that in many-core systems different components experience various temperatures and stresses related to the running workload on them. This leads to aging imbalance among them which causes premature lifetime degradation of highly aged components (i.e. cores or routers).

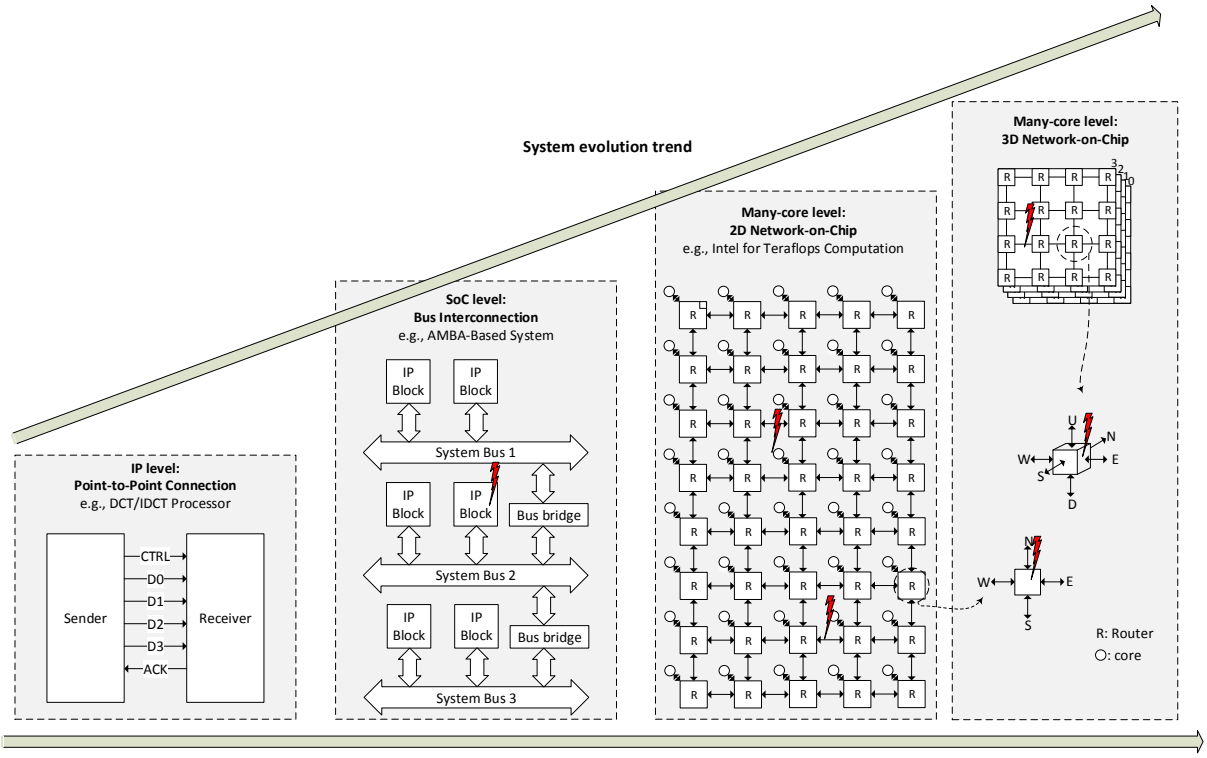


Figure 1.6: The trend of many core systems and on-chip interconnection.

The related works in [11, 12, 19, 20] focused on aging degradation in NoC components.The focus of their works was to mitigate aging by changing the routing between source-destination pairs through profiling aging information at offline. These techniques are not able to catch the runtime behavior of workloads and their impact on temperature and aging. Also, non of these related works studied the impact of aging for 3D NoCs. Additionally, the impact of aging on processing units in many-core systems are studied in [121, 104, 69]. These techniques tried to mitigate aging through task mapping for homogeneous architectures. We observed that for heterogeneous architectures age imbalance between the highest aged core and the lowest aged core is almost 55% higher than homogeneous architectures.

In this dissertation we chose reconfigurable architectures and NoC in many-core system as our platforms for aging study. The usage of both platforms in embedded systems such as safety critical applications as well as main stream applications is inevitable. Performance

and reliability demands for the new application of digital systems such as *Internet-of-Things* (IoT) requires careful investigation of aging to avoid loss in performance, energy, area, power, and scalability.

1.3 Dissertation contribution

This dissertation focuses on mitigating performance degradation for aging in both reconfigurable architectures and many-core system components (i.e. cores and NoC). We devise methods to monitor aging on critical paths, mitigate aging impact on performance in reconfigurable architectures as well as their SRAM cells' SNM. In addition, we investigate aging for both 2D and 3D NoCs and heterogeneous cores in many-core architectures. Our proposed methods objective is to balance aging among available resources and minimizes the delay degradation of highly aged components.

We classify our aging investigation methods for reconfigurable architectures as reactive and proactive methods. In the reactive category, we proposed:

- A highly scalable sensor design for late transition detection in FPGA-based platforms.
- A two-step aging-aware methodology for *Representative Critical Paths* (RCPs) selection from a large number of *Critical Paths* (CPs) in programmable logic devices.

At first, we propose a sensor clock (SCLK) that is a function of minimum slack time of a set of paths selected for age monitoring. There will be one such clock for many sensors as are needed in an entire FPGA. Our proposed sensor architecture makes it possible for a single SCLK to be shared by all sensors. Additionally, the proposed sensor occupies one slice (basic FPGA logic block), which leads to low area, power, and performance overhead.

In addition, we proposed a two-step filtering methodology to select RCPs for aging monitoring in a reconfigurable architecture. In the first step, nomination of CPs is based on delay, temperature, and lexicographic function of duty cycle and switching activity filtering, which are the major causes in BTI and HCI aging mechanisms. Secondly, RCPs will be selected based on Fan-out (FO) and physical location of *Logic Blocks* (LBs) along a CP to decrease

aging propagation and sensor distribution fairness, respectively. We then present a sensor insertion algorithm that will be used during design placement to avoid sensors' inaccuracy. Implementation steps of sensor insertion are performed automatically with a limited human interaction.

In the proactive category, we proposed:

- A high-level physical planning with reconfiguration strategy in order to mitigate the aging-induced delay degradation in FPGA resources.
- A three-step post-synthesis stress-aware technique, to reduce the impact of BTI-induced SNM reduction in FPGA LUTs using SAT-based Boolean Matching (BM) algorithm.

The first method is an offline framework composed of an aging-aware floorplanner coupled with a proactive aging-aware reconfiguration policy which generates checkpoints aperiodically for runtime reconfiguration. We consider BTI and HCI aging mechanisms and consider the BTI-based aging recovery during idle periods using aging history map.

Secondly, our proposed methodology partitions *Data-Flow-Graph* (DFG) of the implemented design into different cones. First, our SAT-based BM algorithm finds a new configuration for each cone in DFG while all SRAMs are flipped and its functionality is preserved. Secondly, cones that did not pass step one can benefit from unused SRAMs in their partially-used LUTs. Hence, we store the flipped configuration of such LUTs in their unused SRAMs. Finally, flipped configurations of fully-used LUTs are stored in unused LUTs. The main configuration of implemented design on FPGA will be swapped by the new flipped configuration, periodically.

To cope with the delay degradation in 2D and 3D NoCs we proposed an online monitoring method through a *Centralized Aging Table* (CAT) for routers in NoCs. To capture inter-layer temperature variations in 3D NoC we exploit Distributed CAT (D-CAT). Router's

capacity in flits, which are the main stimuli in routers, is predictable and limited for a given period of time. Consequently, stress rate and temperature, which are the major sources of aging mechanisms such BTI and HCI, will be in the predictable ranges, as well. Our methodology uses CAT which is populated by values that represent aging degradation for each different pairs of stress and temperature ranges during a given period of time. In 3D NoC, each layer has its own D-CAT to capture the difference between temperature of different layers even with same stress. Furthermore, utilizing D-CAT, we propose AROMa, an online adaptive aging-aware routing algorithm in order to avoid highly aged routers which eventually leads to age balancing between routers. Our proposed routing algorithm reduces maximum age of routers by changing the shortest paths between source-destination pairs adaptively, considering routers' ages across them in each given period of time.

The pervasiveness of heterogeneous multiprocessors (HMP) in the mobile domain enables more energy efficient systems. Current approaches to exploit the energy efficiency of HMPs results in unbalanced usage of resources, which leads to higher aging rates and delay degradation when compared to homogeneous architectures. Hence, we propose ADAMANT, an aging-aware task mapping algorithm for HMPs. ADAMANT exploits on-chip sensing of aging, performance, and power in order to enable online workload characterization to select task-to-core mappings that yield both increased system lifetime and energy efficiency.

1.4 Dissertation organization

The rest of this dissertation is organized as follows. First, Chapter 2 presents SENSIBLE a highly scalable sensor design for reconfigurable architectures. Chapter 3 elaborates on our two-step methodology to find RCPs among large pool of critical paths to insert aging sensors (e.g. SENSIBLE) on them. We propose our high-level physical planning method along with reconfiguration policy for aging mitigation in FPGAs in Chapter 4. Chapter 5 proposes STABLE, a three-step post-synthesis technique to mitigate SNM reduction due to BTI in SRAM cells in FPGAs using Boolean matching technique. In Chapter 6 we elaborate on AROMa, an adaptive aging-aware routing algorithm for 3D NoCs and 2D NoCs along with our novel online aging monitoring system. Next, in Chapter 7, we propose ADAMANT, an aging-aware task mapping technique for heterogeneous architectures.

Chapter 2

SENSIBLE, a novel scalable low-overhead aging sensor design

FPGAs are broadly deployed to accelerate computation in various applications ranging from embedded applications such as automotive, vision, and medical devices to data-center applications. Fabricated in latest advanced silicon technology and deployed for highly computationally intensive kernels, FPGAs face reliability challenges such as aging [18, 30, 161]. BTI and HCI aging mechanisms induce delay degradation to FPGA resources and increase leakage power consumption [142, 148, 7, 138, 50]. The delay degradation in logic and routing resources along *Critical Paths* (CPs) of a design on FPGA results in late transitions that can cause timing failures.

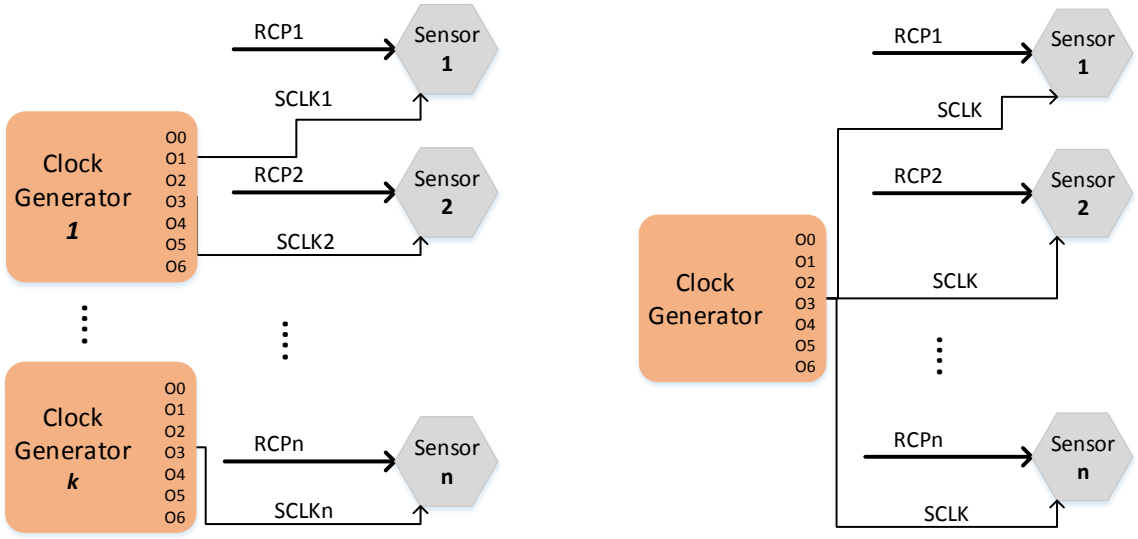
Delay degradation (aging rate) in BTI and HCI mechanisms is exponential function of temperature and nonlinear function of stress time. Each resource in FPGA can be possibly in a critical path of target applications with different temperature and *Stress Rate* (SR) maps. This means different CPs experience different temperatures and stresses, thus different aging rates. Therefore, delay of CPs with longer slack times may exceed smaller ones (including

longest critical path) [50]. This chapter focuses on on-chip aging sensors for critical path delay monitoring in FPGAs.

On-chip aging sensors that are referred to as application dependent sensor [7, 118, 148], are deployed along the paths in a circuit to detect any increase in delay of the paths for system level aging mitigation techniques [118, 160]. A path has a slack time relative to the longest critical path or operating clock period (MCLK). Any transition in slack time is an indication of aging that needs to be detected by aging sensors. While critical paths with negligible slack times are more vulnerable to timing failure due to aging, it is shown that some of near-critical paths with relatively small non-zero slack time often have higher aging rates [50]. Hence, by selecting and monitoring such *near-critical* paths as *Representative Critical Paths* (RCPs), we can potentially detect aging pro-actively to avoid functional failures caused by timing failures in the entire system [50]. Such RCPs even tend to exceed the longest CP's delay due to aging.

Inserting aging sensors on such RCPs' endpoint avoids performance loss due to the added delay of sensors to RCPs. To find these RCPs we deployed and improved version of our RCP selection algorithm in Chapter 3. This algorithm finds RCPs that have higher aging rates than CPs with smaller slack time. However, if aging rates of CPs with smaller slack times (including longest CP) are high enough that may result in timing failure before detection of aging on other selected RCPs, the algorithm selects them for monitoring. Hence, negligible performance loss is unavoidable.

Slack-based aging sensor architecture and design for such RCPs is challenged by various factors. Because of a large number of RCPs in a design, the number of required sensors will be high [50]. Each aging sensor requires a sensor clock (SCLK) and state-of-the-art aging sensors cannot share a single clock source [7, 148]. Given that the number of clock generation modules is limited for an FPGA, existing aging sensor designs are not scalable to be deployed in a large scale for RCPs (Fig. 2.1a). This becomes more important in multiple



(a) Traditional approaches.

(b) SENSIBLE.

Figure 2.1: Multiple aging sensor architecture in FPGAs.

clock domain designs. Another challenge is sensor placement at the RCPs' endpoints. Sensors with higher resource utilization (logic and clock resources) incur more stringent placement constraints. If aging sensors utilize fewer logic resources and share a clock resource, they are more likely to be inserted near RCPs' endpoints. This leads to higher scalability and more accurate operation of sensors with less overhead. Higher accuracy (or sensitivity) leads to faster aging detection than previous works [148, 7]. This helps system level aging mitigation methods to react better by having more precise aging information in RCPs.

This chapter presents a highly scalable sensor architecture by utilizing the minimum slack time of selected RCPs to build a shared sensor clock (SCLK) called SENSIBLE. The proposed multi-sensor clock design has lower area and power overhead in comparison with state-of-the-art aging sensors. As illustrated in Fig. 2.1b, using SENSIBLE we can utilize the same clock generator for a group of sensors. Additionally, the proposed sensor only occupies one basic FPGA logic block (e.g., a slice in Xilinx Artix FPGAs).

Programmable routing resources between programmable logic blocks (CLBs) on FPGAs impose significant delay overhead in a sensor design, which may sometimes exceed the aging-induced delay degradation. By containing the sensor in one logic block and avoiding programmable routing resources, our proposed aging sensors are able to detect aging earlier (i.e. higher accuracy) than existing aging sensors, and yet, they have lower area and power overheads. Earlier (more accurate) aging detection before happening of timing failure helps the system level aging mitigation techniques to react sooner and properly. For example, by changing the placement (configuration bits) of more aged regions we are able to mitigate aging in reconfigurable architectures [65].

The experimental results on Artix7-based board show that the SENSIBLE is a scalable low-overhead aging sensor and it can be inserted in designs on FPGAs in large scale with negligible impact on design performance. Our experimental results support this claim that unlike previous works [148, 7], our proposed sensor design with lower logic resource utilization can fit as closely as possible to path endpoint CLB so as to avoid programmable routing resources. Due to higher accuracy, the aging along CPs are detected earlier than using aging sensors in [148, 7]. To the best of our knowledge this is the first attempt for scalable design of aging sensors on FPGAs which can be applied for ASIC design as well.

2.1 State-of-the-art aging sensors for FPGAs

Logic-based aging sensors for ASIC designs have been proposed in the literature. For instance, sensors presented in [118, 117] measure delay degradation in a circuit due to transistor aging. In [118], two approaches for detection and correction of late transition due to NBTI are proposed, which either impose performance or area overhead to the circuit. The sensor introduced in [117] is based on two ring oscillators, one as a reference, and the other one under stress conditions. The difference between their frequencies is a representation of

their delay difference. In [170], a ring oscillator based multi-purpose sensor on FPGA is presented. Although their method is suitable to extract the FPGA delay, they cannot be used for monitoring aging on CPs (i.e., not application dependent).

Application dependent aging sensors (i.e. by monitoring aging along CPs) for FPGA-based designs are proposed in [148, 7]. In order to detect transitions on path endpoints near the active clock edge, they monitor RCPs. In [7], an FPGA-based aging sensor is proposed which occupies more than one slice. A similar approach is used in [148], with a difference that their proposed sensor is able to adjust the observation interval dynamically. Since such aging sensors require different clocks, multiple clock sources are required (See more details in Section 2.4). This not only reduces their scalability due to limited numbers of clock generation modules, but also increases the area and power overheads. Additionally, the aging sensors occupy more than a basic logic block (e.g. slice) and they are forced to connect to the RCPs' endpoint through programmable routing resources (outside the slice), with a higher delay. In addition, due to higher resource utilization (clock resources and logic), it is less likely to be able to place the sensors close to the RCPsâĂŹ endpoints. Detailed comparison in Section 2.4 is provided.

Some of the related works try to mitigate aging in re-configurable architecture at different levels of granularity by changing the placement of design inside an FPGA [138, 65, 32]. [138, 32] proposed offline aging mitigation techniques by profiling the aging rates of different blocks in an application to increase chip lifetime or avoid timing failure at runtime. However, the applications' behavior (temperature and stress) at runtime might be different and result to impaired solutions which leads to online aging mitigation techniques [32]. Aging sensors on RCPs will be one way of online monitoring. Therefore, sensors with lower overheads and higher sensitivity are needed. Lower overheads (area, power, performance, number of required clock generators, clock routing) leads to higher scalability and lower design complexity. In this work, we amended our proposed RCP selection method in [50] to find required

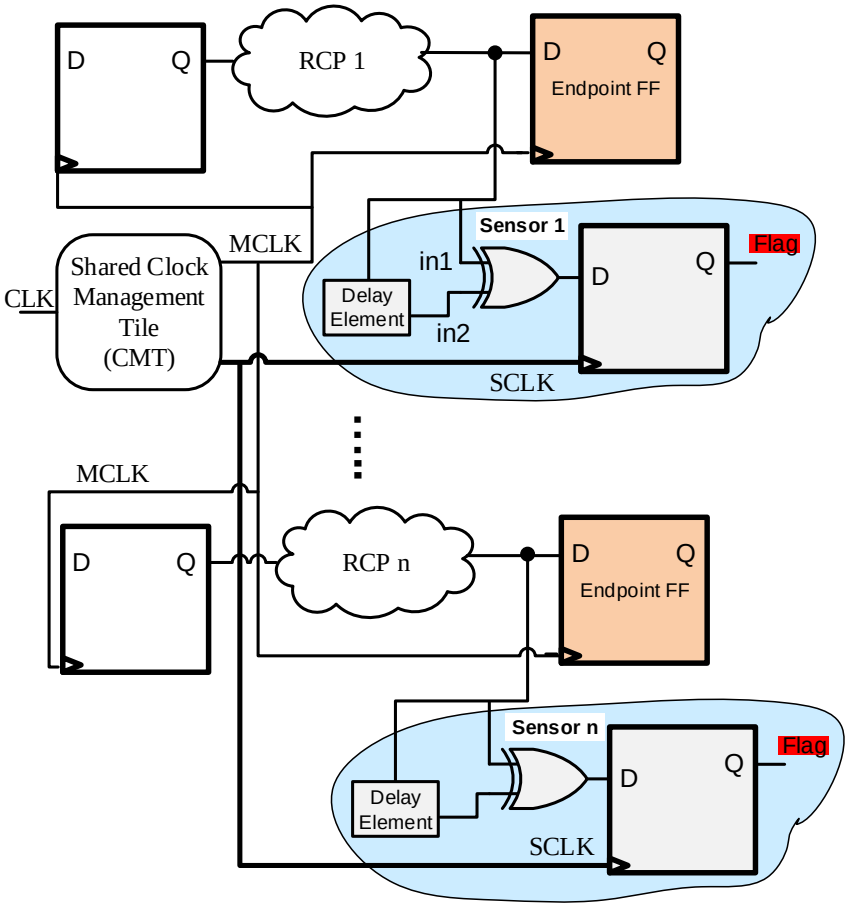


Figure 2.2: Block diagram of SENSIBLE.

RCPs for aging monitoring in FPGAs (Section 2.4).

2.2 Our novel proposed aging sensor (SENSIBLE)

Operation of SENSIBLE is based on the fact that several paths (selected RCPs) in a circuit have non-zero slack times that are relatively close to the slack time of the critical paths. A transition on such path's endpoints during the slack time is known as an erroneous behavior. Detection of any transition on a path's endpoint during the slack time caused by aging mechanism, is considered as an aging effect.

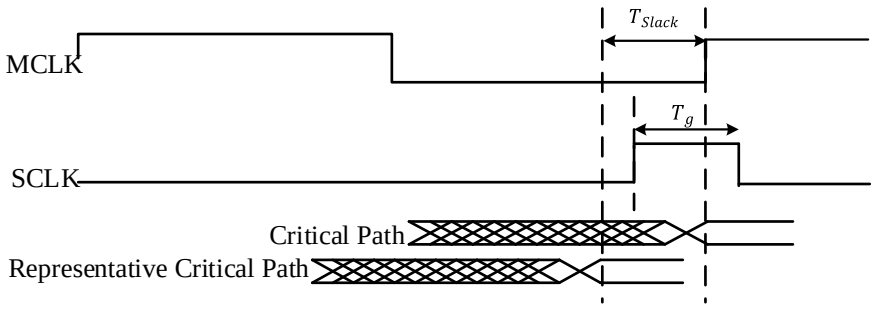


Figure 2.3: MCLK and SCLK representation.

The design of the proposed method is shown in Fig. 2.2. The aging sensor is potentially placed on a near-critical path's (RCP1) endpoint. The sensor consists of an FF, an XOR gate, and a *Delay Element* (DE). For clocking the sensors, a sensor clock (SCLK) is generated that is synchronized with the main system clock (MCLK) using *Clock Management Tiles* (CMTs). Fig. 2.3 shows the timing diagram of SCLK and MCLK. The SCLK frequency is the same as that of MCLK. SCLK makes a 0-to-1 (active edge) transition during the slack time before the active edge of MCLK.

Today's FPGAs have a feature for generating different clock signals using embedded standard resources called *Clock Management Tiles* (CMTs), which consist of *Mixed-Mode Clock Manager* (MMCM) and/or PLLs. For a given clock period T_{clk}, CMTs allow a configurable phase shift to take any discrete value; $phase\_shift = N \times T_{clk}/256$, where N is an integer in the range $-255 < N < 255$. In addition, CMTs allow generation of various duty cycles for a given clock period. Therefore, we can generate a shifted clock signal with a desired duty cycle for the sensor, as required by the proposed aging sensor in this work.

Fig. 2.3 shows the timing diagram of the sensor presented in Fig. 2.2 (for RCP1). As shown, a late transition generates a positive pulse with width of T_{DE} at the output of the XOR gate. The sensor's FF is initialized to '0'. The XOR gate receives data from the near CP (in1) (RCP1), and its time-shifted data (in2) by delay element. Because of the delay element,

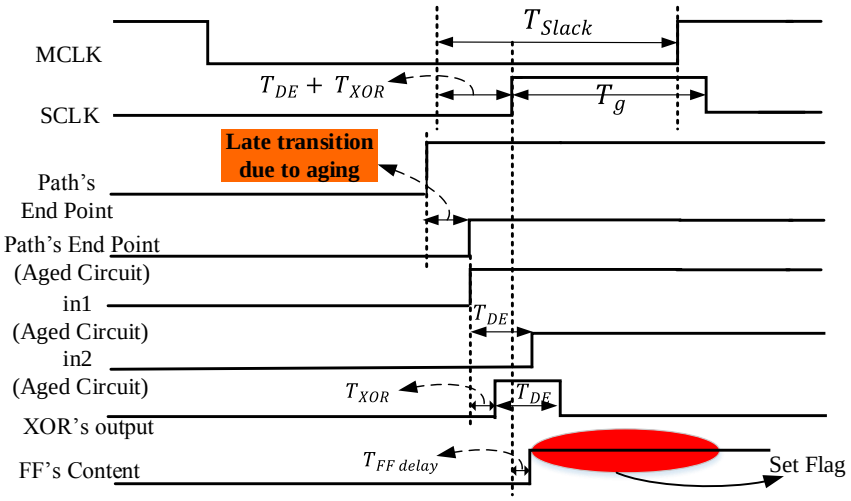


Figure 2.4: The sensor operation in the presence of late transition happened due to aging.

any transition at the output of the RCP driving this sensor will cause a positive pulse on the XOR output. If this pulse happens outside of the slack time, it will safely pass without affecting the FF value. On the other hand, due to the induced delay by aging in the RCP, the aging sensor's FF will capture the pulse at XOR output during slack time window. This occurs on the positive edge of SCLK where FF setup and hold times are also considered.

This positive pulse causes a '1' at the FF output. When there is no aging effect, the path's endpoint does not have any transition that can be captured by SCLK during slack time and there will be no difference between the XOR inputs. XOR output remains at '0' and hence, the FF remains at '0' indicating no late transition.

In order to ensure correct operation of the sensor, T_{Slack} must be chosen based on slack times of selected paths sharing the same clock. Given a set of RCP i with slack time Si, T_{Slack} is the minimum of all slack times.

$$T_{Slack} = min(S_1, S_2, ..., S_n) \tag{2.1}$$

This condition guarantees that none of the selected paths has any transition in T_{Slack}. Valid changes at in1 input of XOR gate can occur up to the start of T_{Slack}. Since in2 is formed by delaying in1 by T_{DE}, we can expect in2 to change T_{DE} after the start of T_{Slack}. Considering T_{XOR} gate delay, the clocking of the FF must be after $T_{DE} + T_{XOR}$ in order to capture slack time violations. It should be noted that as long as the aforementioned condition for SCLK start point is met, its duty cycle could exceed the duty cycle of MCLK. This leads to the easier design of SCLK for our proposed sensor.

To determine the delay of the delay element T_{DE}, the timing difference between the XOR gate inputs must be long enough such that the XOR gate can propagate this difference to its output (e.g., generating positive pulse at its output). Hence, the first condition to be satisfied by T_{DE} is:

$$T_{DE} \geq T_{XOR} \tag{2.2}$$

This pulse propagates to the output of XOR gate (input of FF). Also, T_{DE} must be long enough so that FF setup and hold times are not violated, i.e.,

$$T_{DE} \geq T_{FF\_setup\_hold\_time} \tag{2.3}$$

From (2.2) and (2.3), we can conclude:

$$T_{DE} \geq Max(T_{XOR}, T_{FF\_setup\_hold\_time}) \tag{2.4}$$

The implementation of the proposed sensor on FPGA resources is shown in Fig. 2.6c. In Xilinx Artix (or Virtex) FPGAs, our sensor occupies only one slice. Recall that slice is the basic logic block in Xilinx FPGA devices. Our proposed sensor only uses the intra-slice interconnect resources for connectivity and does not require programmable routing resources.

The proposed sensor has higher accuracy in comparison to available aging sensors in the literature and the delay element (DE) is implemented using two LUTs connected serially.

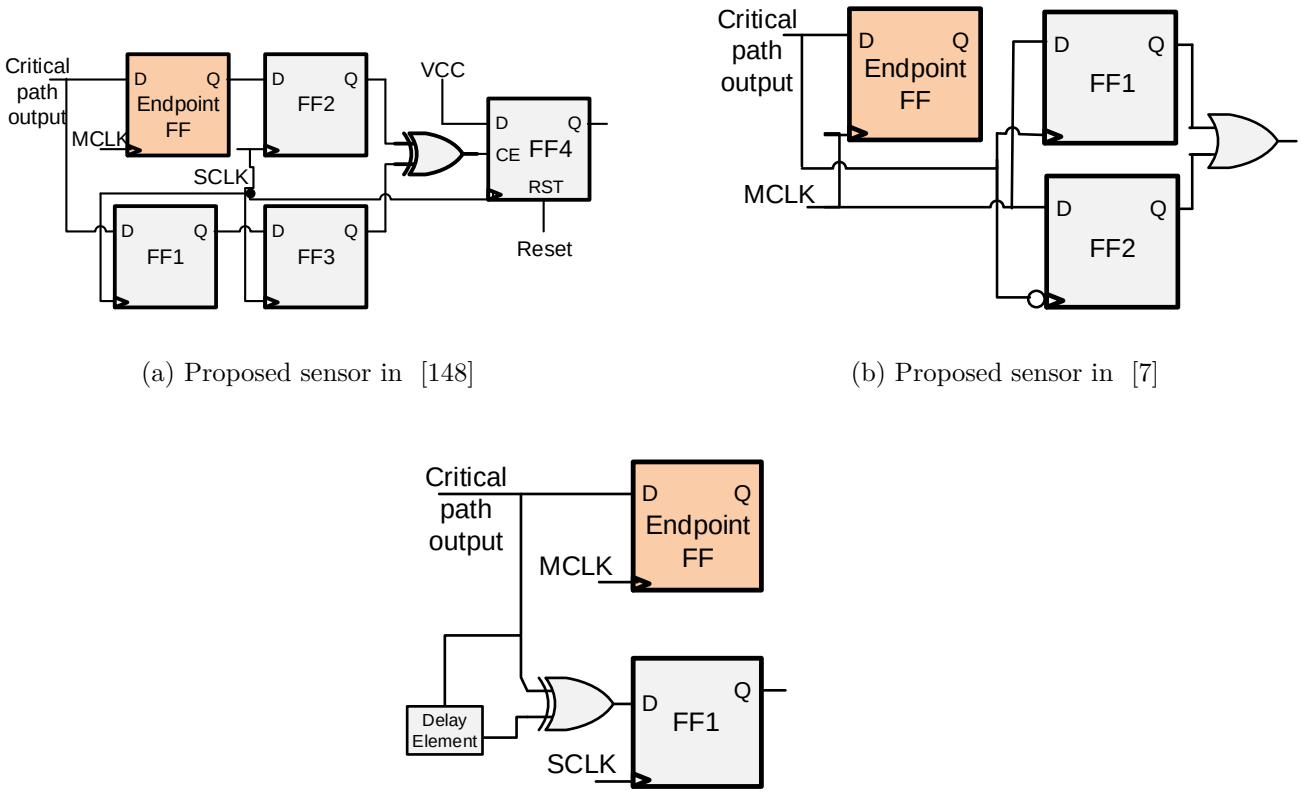
When there is no unused slice inside the endpoint CLB, an unused slice in the nearby CLBs will be selected as the sensor slice. In this case, the programmable routing resources are used. To avoid such scenarios, the placement tools can reserve a slice in the endpoint CLB for sensor insertion. Unlike previous works that propose aging sensors with two slices [148, 7], our proposed sensor only occupies one slice and hence, the probability of finding an unused slice inside the endpoint CLB without changing original design placement is higher. We use a greedy local search to find the closest unused slices (more details in Section 2.5.2).

Since aging is a gradual mechanism and happens in a long term we can turn on sensors in steps of time to avoid aging in sensor's components (Delay element, FF, and XOR). This can be easily done by disconnecting the clock source (SCLK) from them (e.g. by setting a flag). Next, we present a comprehensive comparison between our proposed sensor and existing sensors presented in [148] and [7].

2.3 SENSIBLE vs. state-of-the-art works

As shown in Fig. 2.5a and Fig. 2.6a the proposed sensor in [148] utilizes the observation (guard-band) interval (T_g) of the clock period for detecting any unwanted late transition due to aging. SCLK is the negative-shifted MCLK by T_g to store the correct data before late transition. Then, the faulty data after late transition will be stored in endpoint flip-flop for comparison. By using a T_g shift of MCLK for generation of SCLK, it only detects aging on the targeted RCP accurately (with same T_g).

For multi-sensor insertion on different RCPs multiple clock generators are required because they have different T_g. We can use shared SCLK for this sensor too but in cost of losing



(a) Proposed sensor in [148]

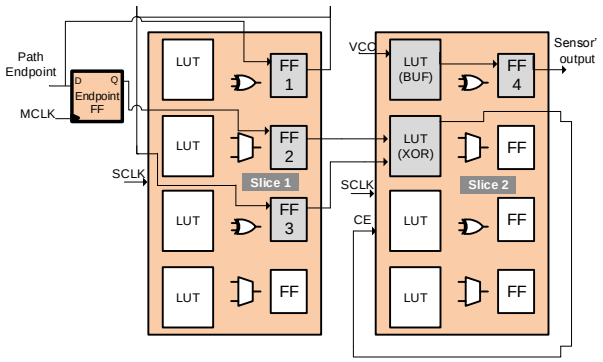
(b) Proposed sensor in [7]

(c) SENSIBLE

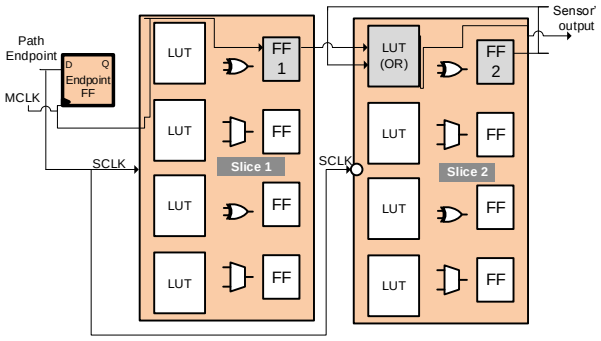
Figure 2.5: Sensor design.

sensitivity and accuracy significantly due to its described way of detecting timing violation. Furthermore, to avoid sensor sensitivity-loss FF1 and endpoint FF must be placed in the same CLB to avoid programmable inter-CLB routing resources with higher delay overhead. Clock enable (CE) input of FF4 is different from other FFs inside the sensor. There is a need for at least two slices (sometimes three) to implement this sensor since flip-flops inside a slice have the same CE inputs (only one CE).

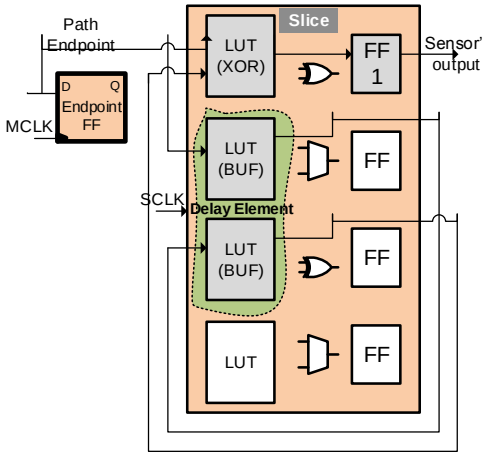
As shown in Fig. 2.5b and Fig. 2.6b, the sensor proposed in [7] uses the signal on the main CP's output as its SCLK and MCLK as its input. When critical path has late rising (falling) transition due to aging, the positive level of MCLK will be latched in FF1 (FF2) as the sign of aging. By using MCLK as sensor's input, it only detects aging on the main critical



(a) Proposed sensor in [148]



(b) Proposed sensor in [7]



(c) SENSIBLE

Figure 2.6: Sensor implementation on FPGA resources.

path. Hence, for multi-sensor insertion on different RCPs (with different delays), we are not able to share SCLK between different sensors and they need multiple clock generators for different detection windows. In addition, FF1 is a positive edge-triggered flip-flop and FF2 is negative, this means we need two slices to implement this sensor because flip-flops inside a slice can only be triggered either on the positive or negative edge.

In summary, the aforementioned sensors have two main drawbacks. For multi-sensor insertion, they need multiple clock generators which impact the sensor design scalability because of limited number of clock generators on FPGAs and increasing clock routing complexity.

This becomes worse in multi clock domain designs in FPGAs. Designing and tweaking different SCLKs for these sensors will be a challenge for designers as well. Since these sensors cannot be placed at the same path endpoint CLB, the inter-CLB routing resources are required to connect the sensor to the path endpoint. This delay overhead impacts the sensitivity of such sensors or may even introduce new critical path to the circuit. Furthermore, using two slices (one CLB) for a sensor induces area and power overheads, which again impacts their scalability for large circuits.

2.4 Aging-aware representative critical paths selection

Due to limited number of resources in reconfigurable architectures, we are not able to monitor the large number of critical paths for aging mitigation techniques. We use the amended version of our proposed algorithm in Chapter 3 to find the minimum number of RCPs. Algorithm 1 shows our filtering based RCP selection pseudo code. The algorithm inputs are the activity matrix $\overrightarrow{\alpha}$, duty cycle matrix $\overrightarrow{Y}$, power consumption matrix $\overrightarrow{P}$, clock frequency f, and matrix of each node (i.e slice) Fan-out along each critical path $\overrightarrow{FO}$.

At the beginning, the list of RCP (RCPlist) is equal to list of CPs. For each CP, we calculate the delay, stress, and fan-out using their node level information (matrices) (line 2-6). Temperature is the dominant factor for BTI and HCI aging mechanisms. Since each CP goes across different regions of implemented design on FPGA the nodes (i.e. slice) along it will experience different temperatures and stresses (Section 2.6, Fig. 2.11). We call HotSpot [137] to extract temperature map at node level as well. Then we remove CPs from the RCPlist that experiences lower temperature than the temperature threshold (line 7-15). After that, we extract the stress map using activity and duty cycle matrices at same level. Then CPs from RCPlist that suffer less than the stress threshold are removed (line 16-20).

Algorithm 1 RCP selection

Input: Critical paths list CP, Delay matrix $\overrightarrow{D}$, Activity matrix $\overrightarrow{\alpha}$, Duty cycle matrix $\overrightarrow{Y}$, Power consumption matrix $\overrightarrow{P}$, Clock frequency f, Fan-out matrix $\overrightarrow{FO}$, Path endpoints list {Ph}

Output: List of representative critical paths {RCP}

$RCPlist = \{\}$
for all $Path_i \in \{CP\}$ **do**
  $D_i \leftarrow CalDelay(Path_i, d)$;
  $Stress_i \leftarrow CalStress(Path_i, \alpha_i, f, Y_i)$;
  $FO_i \leftarrow CalFO(Path_i, \overrightarrow{FO})$;
end for
$delay_{range} \leftarrow CalDelayRange()$;
for all $Path_i \in \{CP\}$ **do**
  if $Path_{i-delay} \in delay_{range}$ **then**
    $RCPlist.Add(Path_i)$
  end if
end for
$\overrightarrow{T} \leftarrow FindTemp(P)$;
for all $Path_i$ **do**
  $Temp_i \leftarrow CalAvgTemp(Path_i, T)$;
end for
$T_{th} \leftarrow CalTempThreshold()$;
for all $Path_i \in RCPlist$ **do**
  if $Temp_i < T_{th}$ **then**
    $RCPlist.Remove(Path_i)$
  end if
end for
$Stress_{th} \leftarrow CalStressThreshold()$;
for all $Path_i \in RCPlist$ **do**
  if $Stress_i < Stress_{th}$ **then**
    $RCPlist.Remove(Path_i)$
  end if
end for
$FO_{th} \leftarrow CalFOThreshold()$;
for all $Path_i \in RCPlist$ **do**
  if $FO_i < FO_{th}$ **then**
    $RCPlist.Remove(Path_i)$
  end if
end for
$RCPlist.Remove(Ph)$

At this step, the RCPlist contains CPs that age faster than removed CPs. To reduce the performance overhead due to sensor insertion on RCPs, it is better to remove CPs with

smaller slack times and lower aging rates. Therefore, we find RCPs that have higher aging rates than CPs with smaller slack times in comparison with the main CP of the circuit. By monitoring such paths, we can detect aging sooner in order to react accordingly. This is shown in Chapter 3 that selected RCPs may have higher aging rates than CPs with the higher delay (i.e. smaller slack time). We define a delay range ($delay_{range}$) for RCPs selection. The delay upper bound (β) is computed by the negation of critical path delay by the sensor delay and the lower bound (γ) delay is defined by the user (e.g. 90% of critical path delay). To avoid timing failure of CPs beyond upper bound (β) (including main CP), we consider the aging rates. If their aging rates are slower than selected RCPs, then it is sufficient (guarantee the timing failure will not happen, because aging will be captured sooner in the already selected RCPs), otherwise that path will be in the RCPlist (line 21-26).

To fairly distribute the sensors among the chip and to reduce the number of RCPs we consider their endpoint physical location (Ph) and average number of fanouts (FO) along a pathâĂŹs nodes (i.e. slices) (line 27-32). The paths with higher average fanout will have higher impact on the reliability of implemented circuit (aging propagation probability is higher). More detailed explanation of Algorithm 1 and RCP selection method is in Chapter 3. In this work, we improved the algorithm by changing the filtering steps in order to avoid removing CPs with high aging rates. As mentioned earlier, we first keep CPs with higher aging rates (line 11-20) then remove those ones from the list that do not satisfy the delay range threshold (line 21-26). By online monitoring RCPs we can detect aging to react accordingly (change the configuration bits) before timing failure happens. For instance, related works in [65, 170] can use sensors aging monitoring.

2.5 Experiment

2.5.1 Setup

Experiments using Aritx7-based FPGA board have been performed in order to evaluate, validate, and analyze SENSIBLE. The implementation of the proposed sensor is feasible with different versions of Xilinx tools (ISE) but can be adapted easily to other vendor or third-party tools. Three benchmarks (DCT (large), AES (medium) and FIR (small)) have been used in our experiments to evaluate the impact of SENSIBLE on area, power, and performance, while assessing the results in term of accuracy and sensitivity. Using Algorithm 1 we find maximum number of required RCPs (sensors), which are 35, 21, 17 sensors (RCPs) for AES, DCT and FIR, respectively. In our experiments, different numbers of sensors (5, 10, and 15) are placed in the top selected RCPs of the circuits with the highest aging rate using our placement tool that uses *Xilinx Design Language* (XDL). To extract power consumption and performance overhead Xpower Analyzer and TRACE by Xilinx are utilized, respectively.

2.5.2 Sensor insertion

Our automated sensor insertion flow is shown in Fig. 2.7. First, a synthesis tool (Xilinx ISE) gets the circuit description in a standard HDL format. Design parameters including path delays (by timing analyzer), wire and node activities for aging estimation (by Xilinx Power Analyzer), and nodes locations (by Xilinx PlanAhead) is extracted after the place and route step. Now, paths are ranked based on their aging rates extracted by these parameters. Using these analyses, RCPs selection will be done by Algorithm 1.

A greedy local search algorithm finds unused resources in the closest slice (for our sensor) or closest CLB near the endpoint slices of selected RCPs. Hence, *Data Flow Graph* (DFG)

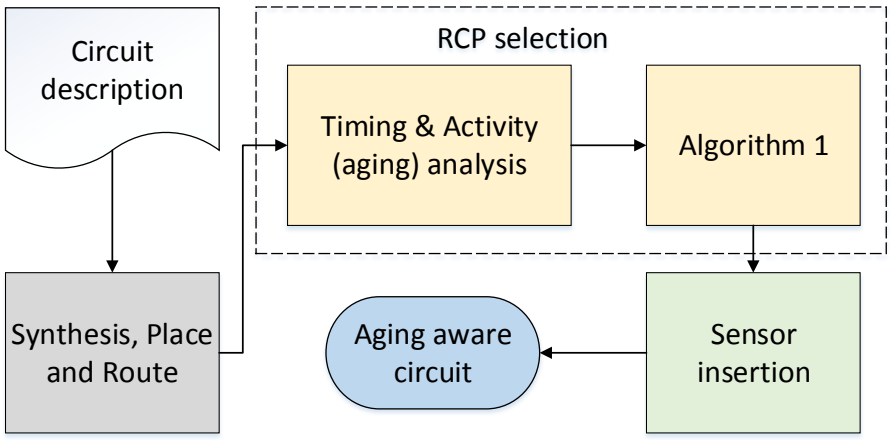


Figure 2.7: Sensor insertion flow.

of the placed and routed design is extracted from the XDL file. Each node of the DFG is a logic block (i.e. slice) that retains information about used and unused resources inside it (e.g. LUT, FF, and etc.). Furthermore, each logic block (or node) is distinguished by physical placement on the FPGA through its x- and y-coordinates. Using this information, we can find the used or unused resources in neighbor logic blocks. Our greedy local search algorithm looks for the first found closest unused resources for placing the sensors required resources (e.g. FF and LUT) [67].

After placement of the sensor, we use the router of commercial tool (ISE®) to route our sensors' resources. Before using the router, our tool routes the placed resource partially, meaning it determines which inputs or outputs will be connected. This new DFG, after the placement and routing of sensors (i.e. insertion), will be converted back to the XDL format for generating the bit-stream of the FPGA based design. All the insertion steps are done on the XDL file, which is compatible with commercial tools. Therefore, our sensor insertion is independent of the implemented FPGA design and can be automated and embedded to ISE or any third-party tools.

As shown in Fig. 2.7, sensor insertion will be done after original design implementation

by commercial vendor tools which consider the worst case (considering process variation) in their placement and routing algorithms. In other word, the sensor insertion flow is in-place and tries not to change the optimized placement and routing by using unused resources after placement and routing. Additionally, FPGA vendors do not release their device level information in order to extract process variation details at the logic level. There are few techniques to extract process variation information of FPGAs using Ring Oscillator (RO) [162, 32]. Using these techniques, we can extract process variation information and consider it in our RCP selection algorithm.

2.5.3 Results and discussion

Table 2.1 shows the area (total number of slices), power (total dynamic and static power), and performance overheads (sensor insertion on RCPs may introduce new critical path, thus we will have performance overhead) for the proposed sensor architecture and design. For fair comparison, area and speed are chosen as optimization goals in the synthesis phase for area and performance overhead calculation, respectively. As shown in Table 2.1, the average power overhead of AES, FIR, and DCT benchmarks for 5, 10, and 15 sensors is 0.90%, 1.03%, and 1.17%, respectively. This low power overhead is not only due to the fact that the proposed sensor only occupies one slice but also regardless of number of sensors, we only use one CMT (MMCM) for generating SCLK as well as MCLK (each MMCM can generate 7 different clock frequencies at the same time).

When ten sensors are inserted for monitoring aging in AES, FIR, and DCT, the area overhead is 0.70%, 3.00%, and 2.30%, respectively. Our aging sensors are inserted in selected RCPsâĂŹ endpoints with high aging rates and are designed using resources inside one slice only. Hence, they are most likely placed in the same endpoint CLB (using available unused slice). The performance overhead of sensor insertion is negligible (i.e., it does not introduce longer

Table 2.1: SENSIBLE overheads for different numbers of inserted sensors.

	Power(%)			Area(%)			Performance(%)
Benchmark	5	10	15	5	10	15	5,10,15
AES	0.61%	0.53%	0.65%	0.30%	0.70%	1.20%	0.30%
FIR	0.83%	1.10%	1.24%	1.50%	3.00%	4.60%	0.00%
DCT	1.26%	1.46%	1.61%	1.10%	2.30%	3.10%	0.00%
AVG.	0.90%	1.03%	1.17%	0.97%	2.00%	2.96%	0.10%

critical path to the circuit).

In Table 2.2, the average area, power, and performance overheads using SENSIBLE and two other implemented aging sensors [148, 7] are reported for selected benchmarks when 15 sensors are inserted. Results show that our sensor has lower overhead compared to the two other sensors. This is because that our sensor regardless of number of required sensors only uses one CMT (MMCM) for generation of MCLK and SCLK (shared), while other proposed sensors [148, 7], as shown in Table 2.2, require 2 MMCMs for 15 sensors. This number increases by increment in the number of sensors (different SCLKs are required). Additionally, our sensor only occupies one slice while other sensors require two slices (a CLB). This also leads to lower overheads in terms of area, power and performance. If we use larger designs in comparison to the mentioned benchmarks, they may need higher number of sensors. Additionally, if we decide to monitor higher number of CPs, we need higher number of sensors as well. Since SENSIBLE overhead is minimum (one slice and one clock generator), it outperforms previous aging sensor design in every aspect of above-mentioned overheads.

In order to measure and compare the sensitivity (accuracy) of SENSIBLE with prior works, we implemented the circuits and inserted the aging sensors in the implemented benchmarks on Artix7 FPGA and connected the sensorsâĂŹ flag to LEDs on the board. As shown in the flow in Fig. 2.8, to emulate aging impact on the implemented benchmarks on FPGA, we

Table 2.2: AVG. overheads comparison(for 15 inserted sensors).

	Power(%)	Area(%)		Performance(%)
Architecture		# slices	#MMCM	
SENSIBLE	1.17%	2.96%	1	0.10%
Proposed sensor in [148]	3.1%	7.20%	2	1.14%
Proposed sensor in [7]	4.43%	6.13%	2	2.26%

increase the delay on RCPs by small steps of Δd. Next, we translate the delay degradation ($\Delta d = d_{aged} - d_0$) to frequency and increase clock frequency (MCLK) by $\Delta f = \Delta \frac{d}{d_0} \times d_{aged}$. This occurs iteratively until the connected sensor outputs to the LED are set and the LED turns on. By this means, we can find the amount of (Δd), or frequencies, at which the implemented sensors will be triggered. Using HotSpot [137] for temperature simulation, extracting the stress rate (duty cycle and activity) after running benchmarks, and delay degradation Δd resulting from on-board experiment, we calculate the earliest time the sensors detect aging-induced delay degradation of Δd using Eq. 2.5 and Eq. 2.6 in Section 2.6. Please consider that we emulated aging by increasing clock frequency of the implemented design to find the "relative" (as opposed to exact) amount of improvement in accuracy of SENSIBLE in comparison to previous works.

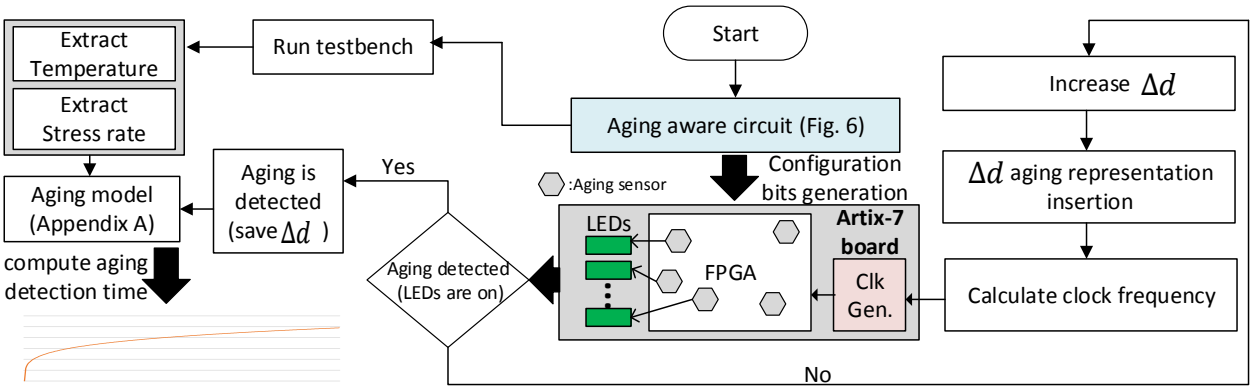
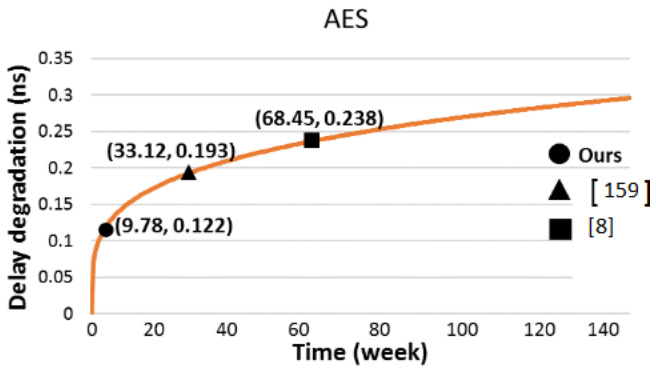
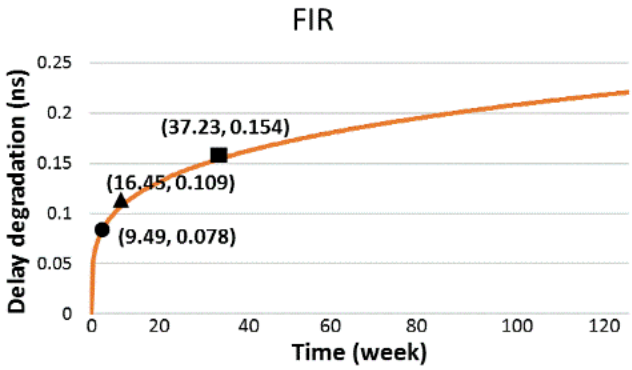


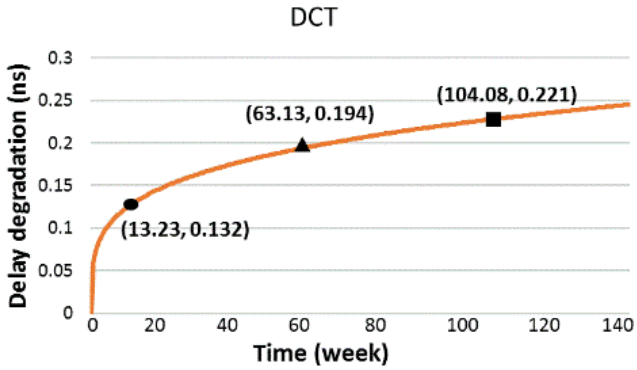
Figure 2.8: On board sensor sensitivity-test flow.



(a) AES benchmark



(b) FIR benchmark



(c) DCT benchmark

Figure 2.9: Aging sensitivity comparison.

Fig. 2.9 demonstrate the aging along top selected RCP in each application. The highlighted points on the curve shows the earliest time and the amount of delay degradation detected by three different aging sensors from the flow in Fig. 2.8. For example, in AES, our proposed aging sensors will be triggered after 9.78 weeks when the induced delay is 122ps (point (9.78, 0.122)). The proposed sensors in [148] detect aging after almost 33 weeks when the degraded delay is 193ps. These values for sensors in [7] are 38 weeks and 238ps, respectively. The results show that our sensor is more sensitive to aging induced delay degradation and therefore, is more accurate.

Fig. 2.10 summarizes the distance between aging sensors and the endpoint flip-flop of the paths under monitoring. While in our flow, we provide stringent constraints to design tools

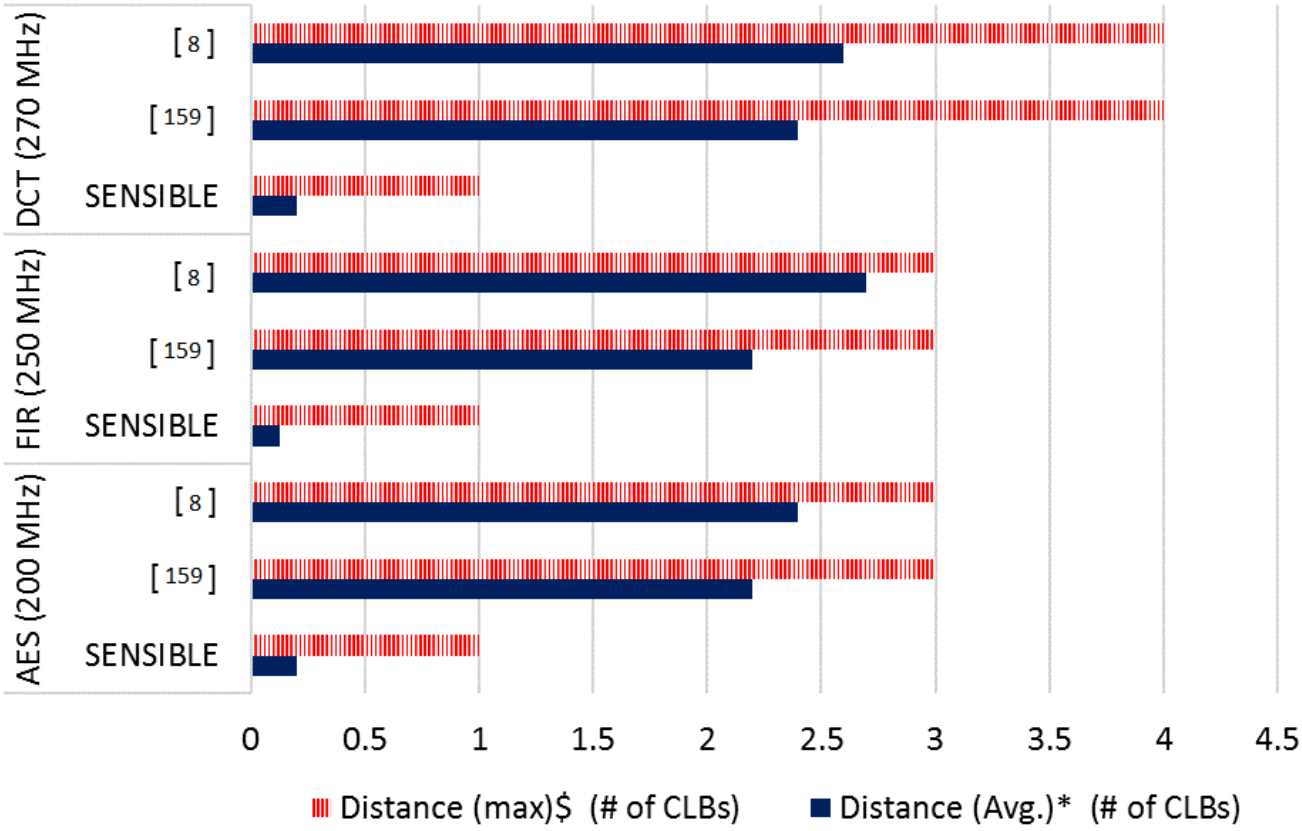


Figure 2.10: Aging sensor displacement (15 inserted sensors).

to place the aging sensors in the same CLB as endpoint flip-flop (distance of zero), the tool will apply best effort but cannot guarantee and hence, in some cases, places the sensors in nearby CLBs (so called sensor displacement). The results show that we can decrease this displacement to zero by forcing RCP endpoint CLB to leave a slice unused. We are not able to do this for sensors in [148, 7] since they need at least a CLB due to their multiple clock design that cannot be supported using one slice. In the worst case, our proposed sensor is placed in the adjacent CLB (max distance = 1 CLB).

Results show that aging sensor displacement in our design is significantly lower than displacement for aging sensors in [148, 7]. The aging displacement results correlate with the sensor sensitivity results as shown in Fig. 2.10. The higher is the average distance, the longer

is the detection time. Since prior works need at least two slices for their implementation, this forces to use inter-CLB routing (switch-boxes) resources to connect them to selected RCP's endpoint which impacts their sensitivity (accuracy). It should be noted that aging induced delay degradation is very small that can be easily compensated by routing delay in aging sensors.

Earlier aging detection of SENSIBLE in comparison to previous works results in better balance of aging among resources on an FPGA and decreases the guardband of critical path. As a result, earlier aging detection can be a better guidance to system level task dispatching on implemented designs' block in FPGAs. Furthermore, placement and routing tools (e.g. ISE) can allocate smaller guardband which increases the implemented designs' performance drastically. In all, higher sensitivity results in faster online reaction in system level aging mitigation techniques and higher system reliability.

As discussed in Algorithm 1, aging sensors will be inserted in different regions of the FPGAs and the clock delay (clock skew) may occur to sensors. While, for our experiments on the Artix-7 board we have not encountered such a problem, there are few techniques in the literature to avoid this issue (so called clock deskew) [48, 59]. Furthermore, for the state-of-the-art FPGAs clock network deskew is provided [75].

2.6 Utilized aging-induced delay degradation model

Delay degradation at transistor level due to BTI and HCI mechanisms results in delay degradation in logic resources of FPGAs such as LUTs, FFs, and carry chains [142, 138, 8]. BTI is a static mechanism and occurs when temperature (T) is high and transistor is constantly under stress (ON). BTI-based degradation depends on transistor's duty cycle (Y) and includes two phases. When the transistor is ON, it is in its stress phase, hence, increasing

the threshold voltage (V_{th}) of transistor. This manifests itself as delay degradation on the FPGA resources (i.e. FF, LUT, MUX, routing resources). When transistor is OFF, it is in its recovery phase and V_{th} will partly decrease. This process manifests itself as partial delay recovery of FPGA resources. The delay degradation due to BTI at time t is a function of duty cycle (Y) and temperature (T) for intrinsic delay (d_0) [114, 31]:

$$\Delta d_{BTI} = A_{BTI} \times (Y)^n \times (t)^n \times e^{\frac{-Ea}{kT}} \times d_0 \tag{2.5}$$

where, A_{BTI} is technology dependent factor, n is a constant depending on fabrication process, k is Boltzmann's constant, Ea is activation energy.

HCI is a dynamic mechanism that occurs when temperature is high and transistor is toggling. HCI-based degradation depends on transistor switching activity. It changes the current-voltage characteristic of transistor that increases V_{th}. The delay degradation due to HCI at time t is a function of clock frequency (f), activity (α), and temperature (T) [114, 31]:

$$\Delta d_{HCI} = A_{HCI} \times \alpha \times f \times t^{(0.5)} \times e^{\frac{-Ea}{kT}} \times d_0 \tag{2.6}$$

where, A_{HCI} is technology dependent factor, k is Boltzmann's constant, Eb is activation energy. BTI aging effect on delay degradation is more dominant than HCI aging effect [50, 31]. Duty cycle (Y) and clock frequency multiplied by activity ($\alpha \times f$) are known as stress in BTI and HCI, respectively.

Since the device level information of the FPGA is not disclosed by vendors, we deployed the proposed method in [50, 31] to calculate aging of nodes along a CP. Node is a basic logic element and its corresponding routing resources (Slice and NET in Xilinx Artix). We assume that all transistors inside a node experience similar stress and temperature. Delay degradation along each path is dependent on the activity and temperature of the resources along the path. As shown in Fig. 2.11, due to different temperatures and stress rates (duty

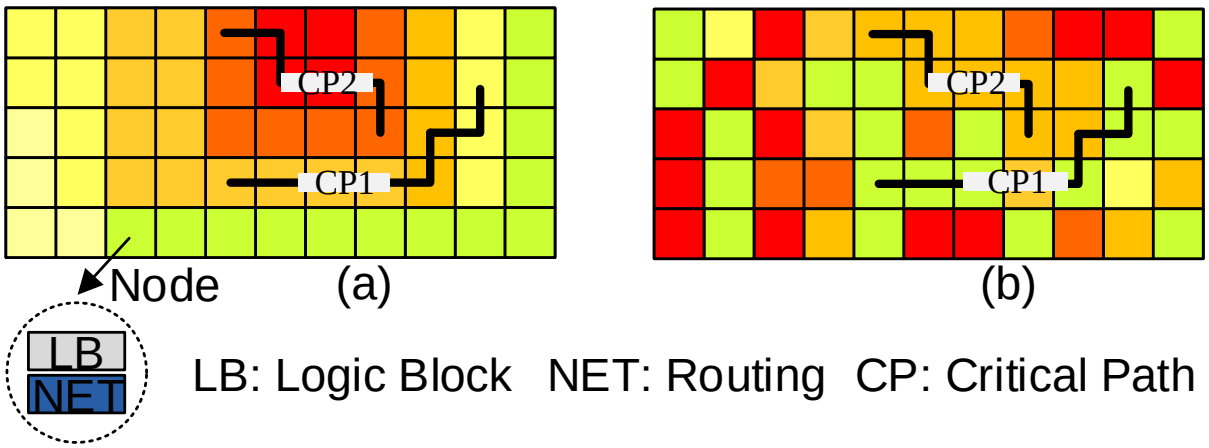


Figure 2.11: Temperature map impact on critical path aging.

cycle, activity and clock frequency) of resources on FPGA, RCPs experience different aging rates [50]. It is observed that such paths may experience higher aging rates than critical paths and their delay may exceed critical paths' due to aging.

2.7 Chapter summary

In this chapter, SENSIBLE, the architecture and design of a low-overhead aging (timing) sensor was proposed along with the strategy for its clock design to increase its scalability in multi-sensor applications. Then, we discussed and compared SENSIBLE with two available aging sensors for FPGAs. We implemented these sensors in selected benchmarks on Artix7 FPGA board for real world comparisons. SENSIBLE exceeds prior works by lower overheads and earlier detection of aging in designs (higher accuracy).

Chapter 3

Aging-aware representative critical path selection for FPGAs

By aggressive down scaling semiconductor technology, designing a reliable system becomes challenging. BTI and HCI are two important phenomena causing accelerated transistor aging [65, 7], which increase the magnitude of the transistor threshold voltage and reduce the effective carrier mobility over time. Moreover, the rate at which supply voltage is scaling is lower than that of transistor size scaling. This results in an increase in current density and temperature, which causes acceleration in device degradation in future semiconductor technologies. Aging effects not only considerably reduce the lifetime of the chip, but also cause timing violations due to delay degradation (δdd) in transistor delay. Once the delay of *Critical Paths* (CPs) exceeds the clock period, the correct functionality of the circuit is affected and timing failures happen. Designers consider avoiding such a failure by on-chip critical path delay monitoring using low overhead on-chip sensors in reconfigurable architectures [7, 148].

The impact of aging and various solutions have been introduced in the context of ASIC-

based design. However, limited research has been done for FPGA devices [142, 170]. As opposed to ASIC designs, where designers are able to place sensors on a chip at the design time, placement of sensors to monitor critical paths in FPGAs needs to be customized for each design (rather than fixed locations). Each resource on FPGA can be potentially in a critical path of target applications due to different temperature and *Stress Rate* (SR) maps. Hence, sensor insertion and allocation need to be applied during place and route phase. Aging sensor insertion in FPGAs is challenged by various factors: 1) Due to high resource utilization locally (e.g., in a slice in Xilinx Virtex FPGA), sensor allocation at nearest location to a critical path may not be possible, and 2) Due to a large pool of critical paths in a design, it is impossible to allocate an aging sensor for each critical path, and hence, only a subset of CPs must be selected for age monitoring.

This chapter presents a two-step methodology to find a list of *Representative Critical Paths* (RCPs) among CPs. At first, paths with delay values close to critical path delay are selected. This set of critical paths, named as *Pseudo Critical Path* (PCP) in the rest of this chapter, are selected to have higher activity than CPs. Due to higher temperature and stress rate along PCPs, aging is manifested sooner on these paths. In addition, the delay overhead of sensors along PCPs does not cause timing violations compared to sensors inserted on critical paths due to their intrinsic timing slack. In our proposed RCP selection algorithm, aging prone path candidates for age monitoring, are first selected. This is based on path delay (of PCPs), temperature, duty cycle, and switching activity. In the next step, a subset of candidates (RCPs) will be selected based on Fan-out (FO), and physical location of path endpoint in *Logic Blocks* (LBs). For Fan-outs, paths with larger FO will be selected. In other words, while path delays, temperature, duty cycles, and activities are given higher priorities to identify aging-prone paths, FOs and sensor distribution play a secondary role in selecting representative paths for age monitoring. We will then present a sensor insertion algorithm that will be used during placement phase to improve sensor accuracy to monitor the path delay. To the best of our knowledge this is the first work that considers different

characteristics of implemented circuit to find reliable numbers of RCPs among a large number of CPs for aging monitoring in reconfigurable architectures.

3.1 Related works on RCP selection for age monitoring

Sensor-based online monitoring of aging-induced delay degradation in reconfigurable architectures have been proposed in [7, 148], while no strategy is presented for critical path selection and sensor insertion in FPGAs. Aging-aware path selection strategies for ASIC design have been proposed in [57, 153]. The approach proposed in [153] selects a small set of CPs considering the delay degradation of the entire circuit due to aging, and not the local causes of aging mechanisms such as temperature or stress rate. Two machine learning-based feature selection approaches are utilized in [57] to find RCPs for aging monitoring in ASIC design. Since the device level information of FPGAs is not available, these proposed methods cannot be deployed in the reconfigurable system design. Besides, these methods do not propose any placement strategy for sensor insertion to tackle the challenges in reconfigurable system design.

Some approaches against aging-induced delay degradation in reconfigurable systems at runtime are proposed in [96, 13]. The method [96] does not propose any path selection strategy and placement. To guide the online placement of modules on FPGAs, an algorithmic based approach determines the aging of a region instead of using sensors [141], which not only imposes performance overhead but also will not be as accurate as using sensors for online monitoring. Proposed methods for offline aging mitigation in reconfigurable systems [65, 141, 32] are out of scope of this chapter.

3.2 Aging model in reconfigurable architectures

Aging impact on transistors manifests as delay degradation (δd). The delay degradation model of the BTI mechanism on transistor's switching delay (d) can be simplified as [8, 114, 21]:

$$\Delta d_{BTI}(t) = A_{BTI} \times (Y)^n \times (t)^n \times e^{(\frac{-E_a}{kT})} \times d_0 \tag{3.1}$$

Where d_0 is the pre-aged delay of transistor, A_{BTI} is technology dependent factor, t is the transistor age, Y is the duty cycle of transistor, T is temperature, E_a is activation energy, k is Boltzmann's constant, and n is constant depending on fabrication process. Delay degradation model of HCI mechanism on transistor's switching delay (d) can be simplified as [8, 114, 21]:

$$\Delta d_{HCI}(t) = A_{HCI} \times \alpha \times f \times t^{0.5} \times e^{(\frac{-E_b}{kT})} \times d_0 \tag{3.2}$$

Where d_0 is the pre-aged delay of transistor, A_{HCI} is technology dependent factor, t is the transistor age, Îś is the activity factor of transistor, f is clock frequency, T is temperature, E_b is activation energy, and k is Boltzmann's constant. According to Eq. 3.1 and Eq. 3.2, delay degradation in both BTI and HCI mechanisms is an exponential function of temperature and a nonlinear function of stress (duty cycle in BTI or switching activity in HCI). Since the transistor level information of the FPGA is not disclosed by vendors, we deployed the proposed method in [142] to calculate aging of nodes along a CP. Node is a basic logic element and its corresponding routing resources (Slice and NET in Xilinx Virtex). We assume that all transistors inside a node experience similar stress and temperature.

As shown in Fig. 3.1, each node along a path may experience different temperatures (the darker the color, the higher the temperature), hence leading to different aging rates for the

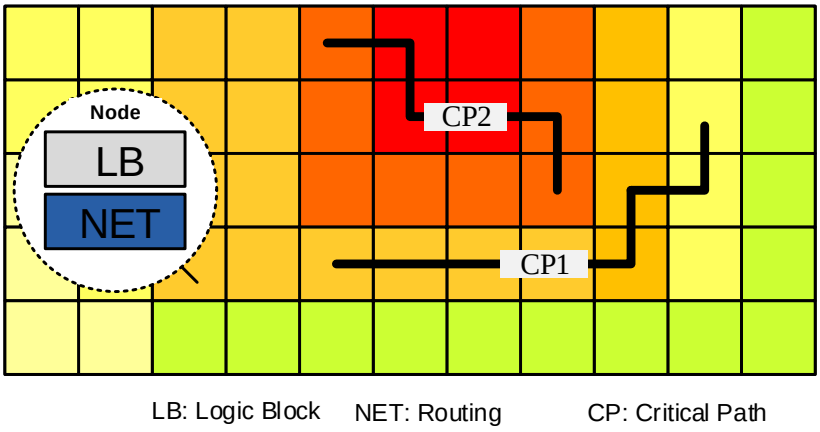


Figure 3.1: Temperature map impact on critical path aging.

nodes along the path. For example, if all nodes have equal delay (one unit of time) and experience equal stress rate, path CP2's delay may exceed CP1's over time due to different temperature maps even if at the beginning $Delay_{CP1} > Delay_{CP2}$.

3.3 The proposed path selection methodology

3.3.1 Path characteristic

Aging-induced delay degradation of each path originates from the delay degradation of each node along it. Each node ages differently from other nodes for various reasons. The aging in each node depends on functional complexity in logic delay as well as Fan-out (FO) in corresponding net delay. In addition, given that temperature and stress on chip varies from one region to another, the temperature and stress at each node varies as well. In this section, we summarize the aging-related path characteristics that are deployed for our representative aging-prone critical path selection algorithm.

Considering Fig. 3.1, assume each CP i consists of m_i nodes. Each node i can be interpreted as a sextuple $(d_{ij}, T_{ij}, \alpha_{ij}, Y_{ij}, Ph_i, FO_{ij})$ where d is delay, T is temperature, α is the activity factor, Y is duty cycle, Ph is physical location, and FO is Fan-out. These parameters play an important role in the aging rate and are computed as follow:

Delay:

The total delay of a critical path is equal to:

$$\textbf{Delay}(CP_i) = \sum_{j=1}^{m_i} d_{ij} \tag{3.3}$$

Because of resource limitation and exponential number of paths, it is impossible to monitor all of the circuit's paths. In addition, power and area overhead for each sensor should be taken into account. Hence, we need to select paths which are more prone to aging and the aging-induced delay degradation along those paths are more likely to cause timing violations. The pseudo-critical paths are those with delay in the range of $\alpha\%$ to $\beta\%$ of minimum clock period. These bounds may change based on the sensor delay to avoid performance loss, implemented circuit delay, and number of CPs.

Temperature:

The exponential impact of temperature on both BTI and HCI mechanisms are unavoidable. In order to compute the aging rate at each node along CPs, we need to consider the temperature in the corresponding region. Therefore, the temperature map of implemented design on FPGA is extracted adapting HotSpot tool [137]. Unlike ASIC design, temperature map of the chip changes by different configurations on the FPGA. This leads to the fact that the resources on FPGA based on the implemented configuration experience different aging

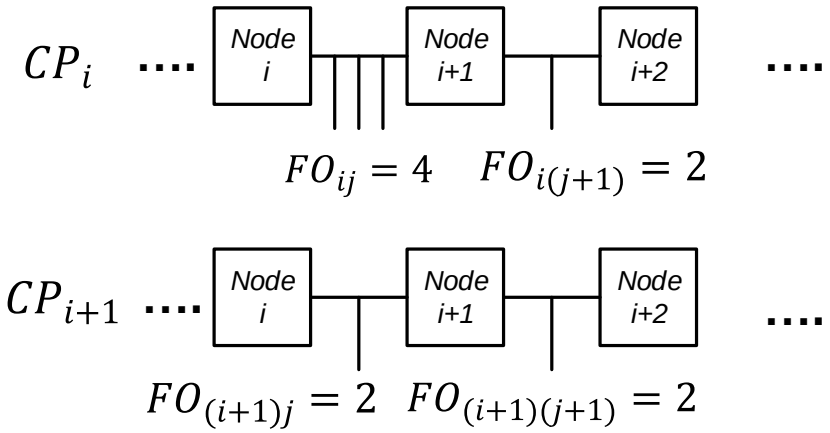


Figure 3.2: Impact of Fan-out (FO) for path selection .

rates at different times. As shown in Eq. 3.4, the temperature of a path is the average temperature of its nodes.

$$\textbf{Temp}(CP_i) = \frac{\sum_{j=1}^{m_i} T_{ij}}{m_i} \tag{3.4}$$

Stress rate:

Based on Eq. 3.1 and Eq. 3.2 *Stress Rate* (SR) is equal to Y in BTI mechanism and equal to $\alpha \times f$ in HCI mechanism. BTI is a static mechanism and is triggered when transistor in ON. HCI is a dynamic mechanism and occurs when transistor toggles. In both cases, transistor is under stress and hence, aging causes delay degradation. Each node along a path may experience different stress rates which results to different stress maps among different configurations implemented on the FPGA. This consideration comes from the fact that paths close to critical path delay but with relatively lower stress rate may not be prone to aging as aggressively as others. As shown in Eq. 3.5 each path's stress rate is a lexicographical function of activity rate ($\alpha \times f$) and duty cycle (Y). C_1 and C_2 are chosen based on relation

impact of BTI and HCI on aging-induced delay (C_2 is three times greater than C_1).

$$\textbf{Stress}(CP_i) = \frac{\sum_{j=1}^{m_i} C_1 \times (\alpha_{ij} \times f) + C_2 \times Y_{ij}}{m_i} \tag{3.5}$$

Fan-out:

In addition to closeness between CPs, another factor to be considered in our path selection algorithm is Fan-out (FO) of nodes along the path. Assume CP_i and CP_{i+1} in Fig. 3.2 experience similar aging rate while based on Eq. 3.6 $FO(CP_i)$ is larger than $FO(CP_{i+1})(FO(CP_i) = 6)$. The probability of aging propagation in paths with larger FO is higher. FO calculated based on Eq. 3.6.

$$\textbf{FO}(CP_i) = \sum_{j=1}^{m_i} FO_{ij} \tag{3.6}$$

Physical location:

To fairly distribute the limited number of sensors on the chip, physical location of nodes along the path, especially the node that contains the endpoint flip-flop, is required to be considered. Location filtering is for avoiding redundant and unnecessary sensor. Someone may argue all the filtering phases are not for aging mechanism such as location but it needs to be considered to further optimize the number of required aging sensors.

3.3.2 Path selection algorithm

Algorithm 2 shows the pseudo code for the proposed critical path selection method. The algorithm inputs are the activity matrix $\overrightarrow{\alpha}$, duty cycle matrix $\overrightarrow{Y}$, power consumption matrix $\overrightarrow{P}$, clock frequency f, and matrix of each node Fan-out along each critical path $\overrightarrow{FO}$. The

objective of this algorithm is to find the near minimum number of RCPs. The algorithm is decomposed of two major steps. First, finding the aging-prone paths (finding PCPs, temperature filtering, and stress rate filtering), followed by finding representative critical paths for sensor insertion (Fan-out and endpoint physical location filtering).

Based on the fact that sensor inserting on *Pseudo Critical Paths* (PCPs) does not lead to performance loss (no increase on CP delay), our proposed algorithm concentrates on PCPs instead of critical paths. The pseudo-code in line 8-11 identifies PCPs. $delay_{range}$ is the range of critical path delay allowed for selecting CPs in the first round. The delay upper bound is computed by the negation of critical path delay by the sensor delay and the lower bound delay is defined by the user (85% of critical path delay).

At the next filtering, we consider temperature at each node along the selected paths (line 12) which is a dominant factor in both BTI and HCI mechanism. The FindTemp() function calls HotSpot tool [15] for extracting the circuit's temperature map. Temperature will be calculated at node level as shown in Fig. 3.1, a path may route through different temperature zones, which leads to different aging rates along a path. If average temperature along $Path_i$ (line 14) is less than a threshold (T_{th}), the path will be filtered and removed from the candidates (lines 17-20). Selecting the appropriate value for T_{th} is a crucial factor that heavily depends on temperature distribution of paths and varies from application to application. For example, Fig. 3.3 shows the path temperature distribution of S38417 benchmark.

Among the paths, 131 paths have average temperature close to 361 and 148 paths have average temperature of 362 (Kelvin). T_{th} is set to 361 Kelvin to include both sets of paths with high temperature. In other words, aging rate of these paths (with respect to exponential effect of temperature) can potentially exceed the others. After this filtering, CPs are further filtered based on their stress (lines 22-25), another important factor in both BTI and HCI mechanisms. In order to filter appropriate paths, value of $Stress_{th}$ is chosen based on stress factor distribution of paths (similar procedure as temperature filtering). In the last

Algorithm 2 The proposed RCP selection

Input: Critical paths list CP, Delay matrix $\overrightarrow{D}$, Activity matrix $\overrightarrow{\alpha}$, Duty cycle matrix $\overrightarrow{Y}$, Power consumption matrix $\overrightarrow{P}$, Clock frequency f, Fan-out matrix $\overrightarrow{FO}$, Path endpoints list {Ph}

Output: List of representative critical paths {RCP}

1: $RCPlist = \{\}$
2: **for all** $Path_i \in \{CP\}$ **do**
3: $D_i \leftarrow CalDelay(Path_i, d)$;
4: $Stress_i \leftarrow CalStress(Path_i, \alpha_i, f, Y_i)$;
5: $FO_i \leftarrow CalFO(Path_i, \overrightarrow{FO})$;
6: **end for**
7: $delay_{range} \leftarrow CalDelayRange()$;
8: **for all** $Path_i \in \{CP\}$ **do**
9: **if** $Path_{i-delay} \in delay_{range}$ **then**
10: $RCPlist.Add(Path_i)$
11: **end if**
12: **end for**
13: $\overrightarrow{T} \leftarrow FindTemp(P)$;
14: **for all** $Path_i$ **do**
15: $Temp_i \leftarrow CalAvgTemp(Path_i, T)$;
16: **end for**
17: $T_{th} \leftarrow CalTempThreshold()$;
18: **for all** $Path_i \in RCPlist$ **do**
19: **if** $Temp_i < T_{th}$ **then**
20: $RCPlist.Remove(Path_i)$
21: **end if**
22: **end for**
23: $Stress_{th} \leftarrow CalStressThreshold()$;
24: **for all** $Path_i \in RCPlist$ **do**
25: **if** $Stress_i < Stress_{th}$ **then**
26: $RCPlist.Remove(Path_i)$
27: **end if**
28: **end for**
29: $FO_{th} \leftarrow CalFOThreshold()$;
30: **for all** $Path_i \in RCPlist$ **do**
31: **if** $FO_i < FO_{th}$ **then**
32: $RCPlist.Remove(Path_i)$
33: **end if**
34: **end for**
35: $RCPlist.Remove(Ph)$

stage, to recognize the representative CPs, those paths that consist of nodes with high FO will be chosen (lines 27-30). Finally, considering the location of endpoints, the paths with

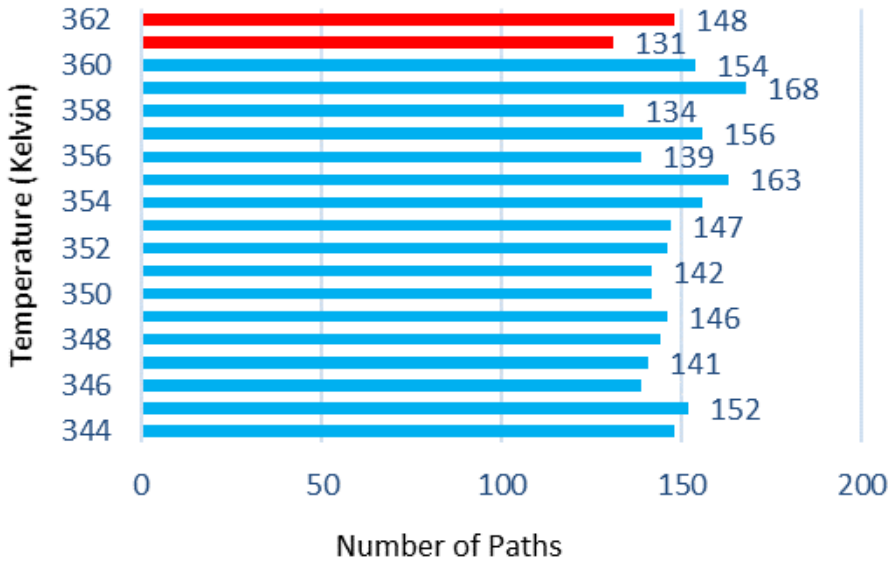


Figure 3.3: S38417 benchmark temperature histogram.

same endpoints but one will be removed from the final list, also at this filtering phase the spatial distribution of the sensors will be considered to fairly distribute them around the implemented circuit's bounding box (line 31).

Algorithm for sensor placement

The proposed algorithm is based on a greedy local search (i.e. *Breadth First Search* (BFS)) algorithm that localizes unused slices near the *Endpoint Slice* (EPS) including the destination flip-flop of the path under consideration for sensor insertion. We explain the algorithm using an example shown in Fig. 3.4. The used resources are colored as black (Slices) or red (Switch boxes) and the unused resources are left as white. Finally, Resources labeled SSi, are utilized to place an aging sensor (SS).

In the placement algorithm, the best slice for sensor insertion is when the other slice in the endpoint CLB is empty (CLB4 situation in Fig 3.4). When the other slice in the endpoint CLB has been occupied, one of the unused slices in the nearby CLBs will be selected as the

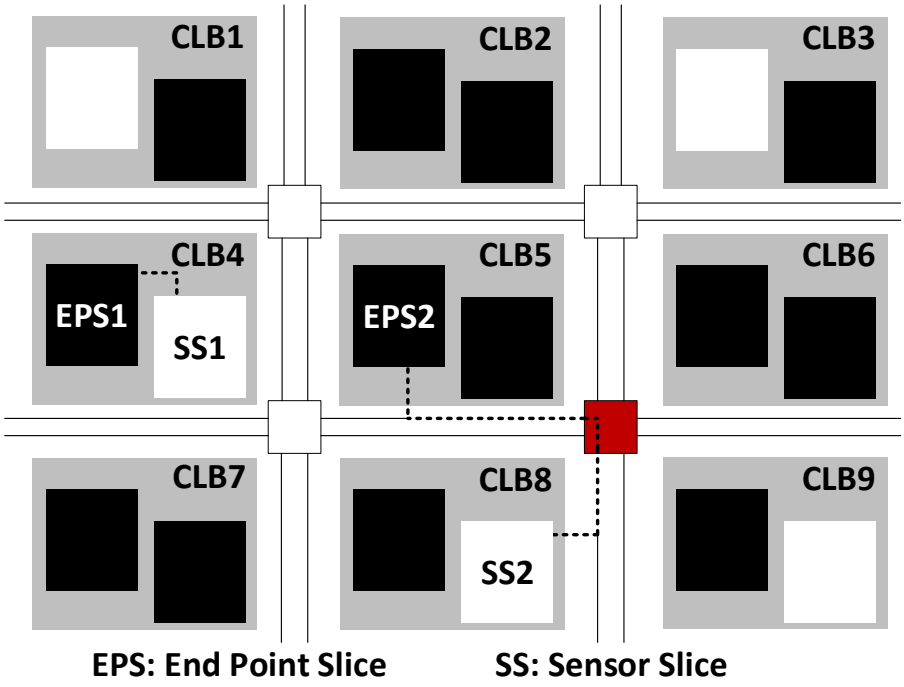


Figure 3.4: Sensor placement.

sensor slice (SS2 in CLB8 is utilized to place the sensor for EPS2). In this case, the general interconnect through a switch box is utilized (red resource in Fig 3.4). We assume an aging sensor occupies only one slice similar to SENSIBLE in Chapter 2.

Sensor placement challenge

Although, the placement algorithm can finally place each aging sensor, but its distance from the corresponding endpoints of the path is a crucial factor. The sensor must be placed on a slice in the same CLB of the destination slice or in the worst case, on a slice of adjacent CLBs. Placing the sensor far from the destination slice leads to a significant increase in the delay of the RCP which is monitored and affects the correct functionality of the sensor. In such a situation we should reserve a slice for the sensor during place and route phase followed by our proposed sensor insertion flow.

3.4 Experiment

3.4.1 Insertion flow and setup

The implementation of sensor insertion is divided to several steps (Fig. 3.5):

1. The benchmark circuits are synthesized using synthesis tools (ISE 14.7). The synthesizer generates post route information without considering aging effects.

2. Timing analysis is performed using Xilinx PlanAhead tool in order to obtain timing information including path delays. The path delays are extracted using a timing analysis tool. For example, PlanAhead tool can generate a timing report file (.twr) containing all of the logic objects and interconnects and their associated delay.

3. Activity rates and duty cycle of circuits are extracted using a power analyzer tool. *Xilinx Power Analyzer* (XPA) receives implemented design description in ISE (using the .ncd file generated after the place and route phase) and simulation activity files (.saif or .vcd) generated by a simulation tool (ISim or Modelsim). The XPA generates a power report file (.pwr) containing signal and I/O activity rates. Path stress rates are calculated based on net activity rates extracted from power report file using a C-based tool.

4. Our proposed critical path selection and sensor placement are applied to design.

5. PlanAhead tool provides user various placement constraints to be applied during place-and-route tools. Using this tool, we reserve some locations in order to place the aging sensors.

Table 3.1: Number of sensors after applying filtering algorithm.

Benchmark	S15850	S38417	B20	FIR	AES
# of LUTs	582	1997	1912	108	3974
# of FFs	431	1387	430	400	261
# of Slices	245	929	557	83	1282
CLK frequency (MHz)	166	125	101	100	136
Delay filtering	12322	31359	58326	10235	12568
Temperature filtering	516	2804	6716	200	1122
Stress rate filtering	130	292	1754	182	757
Fanout filtering	82	121	735	72	301
Physical location filtering	71	60	291	53	263
# of Sensors	**16**	**32**	**28**	**17**	**38**

3.4.2 Results and discussion

In our aging model, the values for ABTI and AHCI are chosen such that the maximum delay degradation in 5 years is 15% in worst case (PMOS transistors always ON, the maximum frequency (AR=500 MHz) at temperature 380 Kelvin). The target FPGA device is Virtex6 at 40 nm technology.

To evaluate the impact of proposed filtering, five different benchmarks are selected from different applications. The characteristic of each benchmark is shown in Table 3.1. For each benchmark the number of selected RCPs is extracted by proposed algorithm 2. Table 3.1 shows that the number of paths filtered in each filtering phase of the algorithm. The results show that the number of paths are reduced significantly at each filtering phase and the impact of each filtering phase varies from one benchmark to another. For example, in AES, the number of paths (delay filtering in Table 3.1) is very high, but the number of pseudo-critical paths (PCPs) is only 1122. The other four filtering characteristics (temperature, stress rate, FO, and physical location) reduce the number of selected paths and at the end, 38 sensors are required to monitor aging for this circuit.

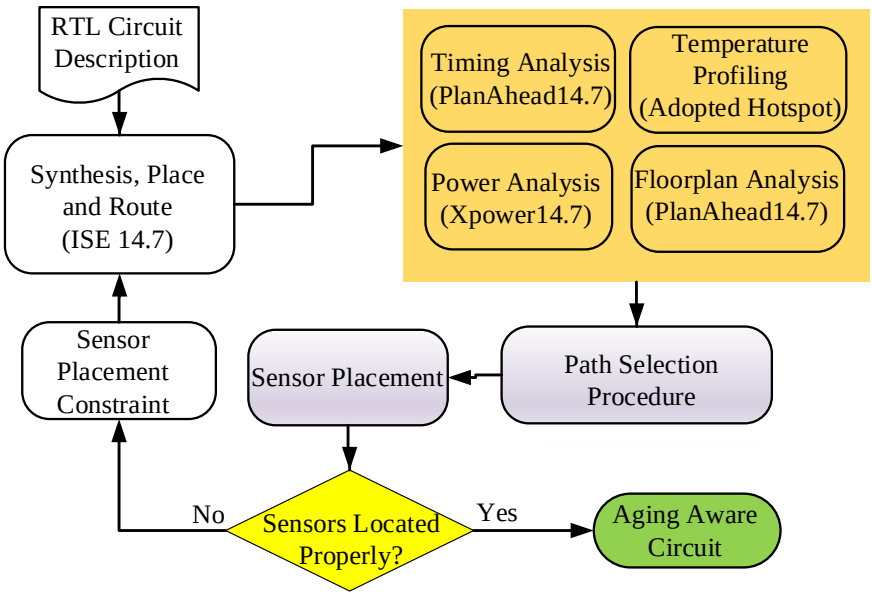
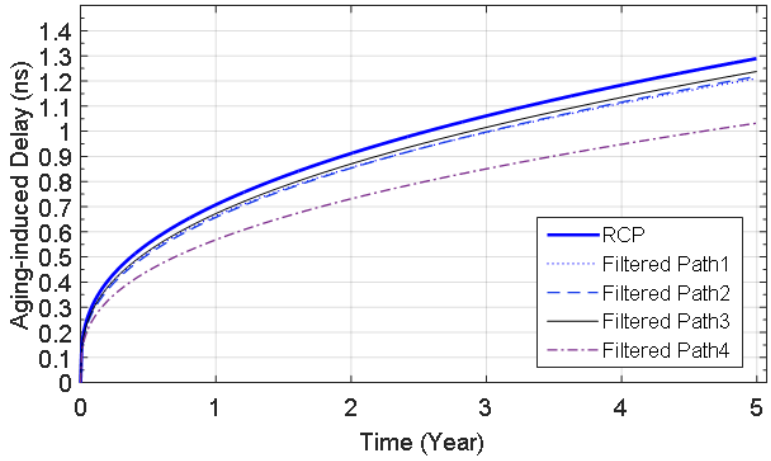


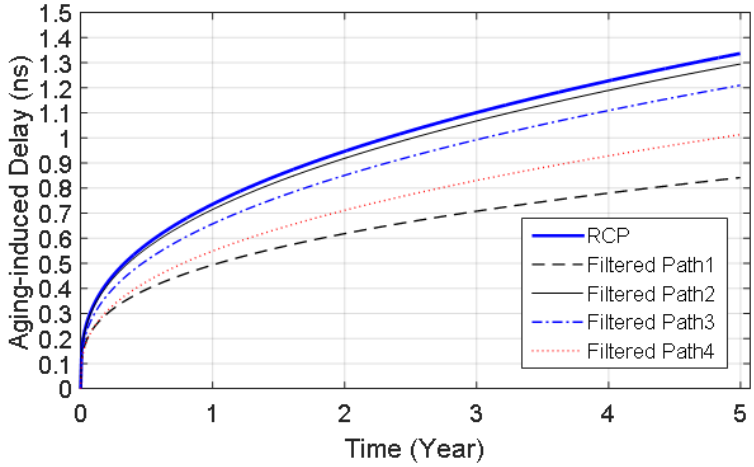
Figure 3.5: Temperature map impact on critical path aging.

Similarly, results are shown for S15850, S38417, B20, and FIR. The number of sensors gradually increases when the circuits' sizes are increasing. For S15850, S38417, and B20 the numbers of critical paths that are required to be monitored are 16, 32, and 28, respectively. For example B20 is a larger circuit with higher number of critical paths at first phase of filtering (delay) but due to low stress rate and temperature of nodes (due to its behavior) most of the critical paths are filtered out at temperature and stress filtering.

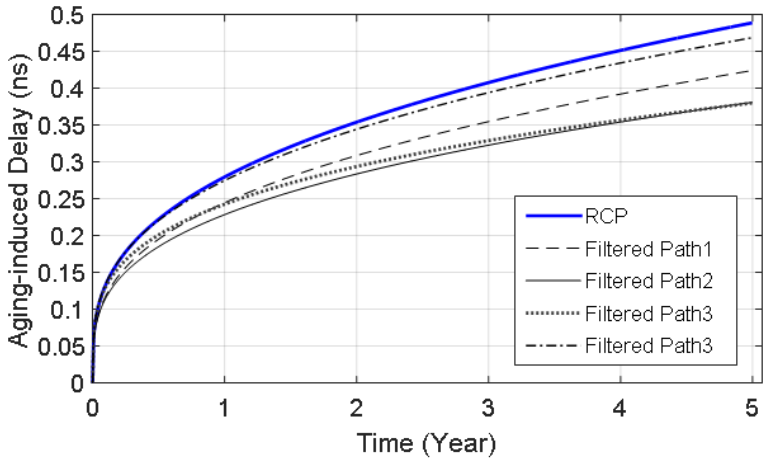
Finding empty slices for sensors determined for each circuit, is the next step. Table 3.2 shows the output of greedy search algorithm representing number of sensor slices which placed in the same CLB including endpoint slices (Case1) and the adjacent CLBs (Case2). In our experiments, there was always an available slice for sensor insertion either in the same CLB or adjacent CLB. Hence, there was no need to apply placement constraints and re-synthesize the circuit in selected benchmarks (Case3).



(a) B20



(b) S38417



(c) S15850

Figure 3.6: Aging-rate comparison of last selected RCP and top filtered out paths.

Table 3.2: Number of sensors placed by the algorithm.

	S15850	S38417	B20	FIR	AES
Case1	9	25	15	14	30
Case2	7	7	13	3	8
Case3	0	0	0	0	0

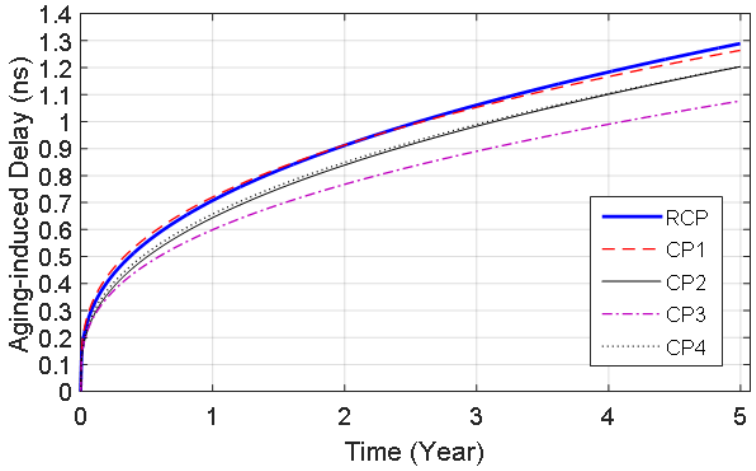
3.4.3 Evaluation of the proposed filtering algorithm

To evaluate the effectiveness of the proposed method, we compared aging rates of RCPs with those paths that are filtered out and removed from our proposed algorithm. As shown, in Fig. 3.6, aging rate of the last selected RCP is higher than aging rate of the filtered paths. In other words, RCPs will be aged sooner than the filtered paths. Hence, the inserted aging sensors detect aging-induced delay degradation of circuit sooner while inserting sensors on removed paths will only cost to the circuit without any meaningful usage.

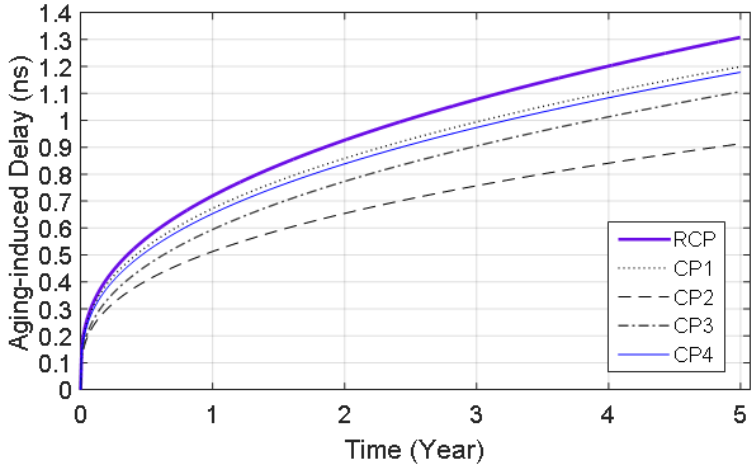
Fig. 3.7 shows aging rate of the last selected RCP in comparison with aging rate of the top-ranked critical paths (based on delay including main critical path, CP1). This result shows aging rate of the last RCP is higher than them, although its pre-aged delay is lower. Consequently, aging manifestation on these RCPs happens sooner than that of the top-ranked critical paths. Hence, by monitoring RCPs we are being able to detect aging before any timing violations.

3.5 Chapter summary

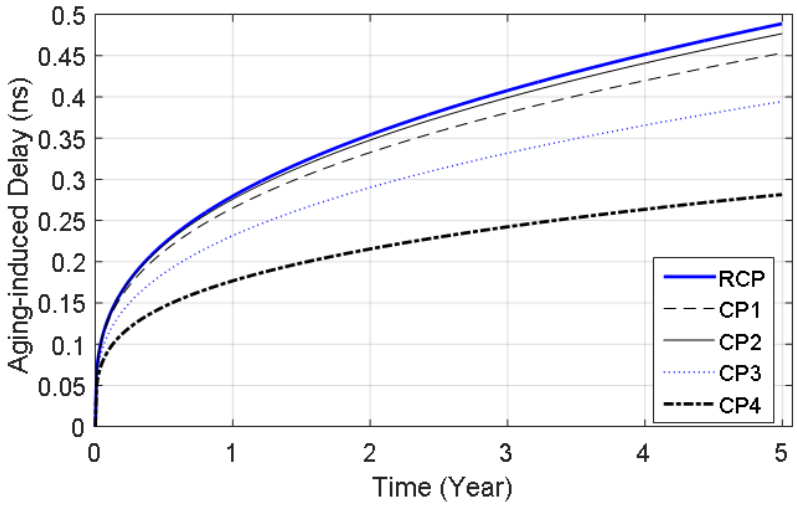
In this chapter, we proposed an aging-aware critical path selection methodology for online aging monitoring using sensors in reconfigurable architecture. Our objective is to reduce the number of required sensors and to avoid unnecessary sensor insertion due to its cost and overhead. We first present a novel representative critical path selection based on major causes



(a) B20



(b) S38417



(c) S15580

Figure 3.7: Aging-rate comparison of last selected RCP and top critical paths.

in aging mechanisms such as temperature, duty cycle and switching activity. Furthermore, to avoid performance loss due to sensor insertion a performance-aware placement is proposed. To the best of our knowledge this is the first work that considers path selection for aging monitoring in FPGAs, considering not only aging effects, but also delay, physical location and impact of aging on net delay paths.

Chapter 4

Aging-aware physical planning for FPGAs

While aging mechanisms and process variation lead to performance degradation, permanent faults, and increase in power dissipation, reconfigurable systems such as FPGA-based embedded systems can benefit from underutilization of resources and *runtime reconfiguration* to mitigate aging effects. Researchers have proposed various techniques to exploit such flexibility in FPGAs [138, 167, 102]. This chapter focuses on aging-induced delay degradation in FPGA logic and routing resources and presents a physical planning and reconfiguration scheme to reduce *aging-rate* and delay degradation in FPGA resources and to slow down the aging effect on application performance.

Although, aging mechanisms decrease the *Signal Noise Margin* (SNM) of the SRAM cell configuration bits, but the focus of previous works and the proposed method here as well is to decrease aging-induced delay degradation of transistors inside FPGA logic and/or routing resources. We study the impact of aging on SNM in Chapter 5.

Among various mechanisms of aging, i.e., *Negative/Positive Biased Temperature Instability*

(NBTI/PBTI), *Hot Carrier Injection* (HCI), *Electromigration* (EM), and *Time-dependent Dielectric Breakdown* (TDDB), BTI and HCI are dominant mechanisms on delay degradation of resources [8, 142]. *Aging rate* in BTI and HCI mechanisms are highly dependent on *Temperature* (T) as well as *Stress Time* (ST)(when transistors are toggled). Since aging-induced delay degradation is an exponential function of temperature and depends on ST in non-linear fashion, ignoring any of the two has significant impact on accuracy of *aging rate* estimation and may lead to impaired solutions [115].

Spatial distribution of temperature and critical paths of a design is affected by mapping and physical planning on FPGA resources. Runtime reconfiguration enables multiple physical mappings on resources temporally and hence, it has a critical role in balancing *aging rate*. In system-level design flow for FPGAs, high-level physical planning tools such as *floorplanner* maps each component at block- (e.g., datapath components) or IP-level (e.g., soft processor) on the device. High level physical planning tools are more effective and powerful than placement tools (CLB-level and/or logic level) [138, 167, 141, 32] to migrate thermal hot spots or highly active blocks globally across the chip. Hence, aging mitigation can be achieved more effectively using an aging-aware floorplanner while aging-aware placement tools due to stringent timing constraints of blocks are only limited to swap used and unused resources locally.

In this chapter, we propose an aging-aware floorplanner tool, which allows global migration of blocks in design to underutilized resources. Moreover, a *fine-grain Aged-delay Map* ($\overrightarrow{ADmap(t)}$) is proposed to capture the *aging delay history* effects of underlying resources. Our tool finds a new configuration with minimum *aging rate* given the aging history of the resources from the past configurations. Also, $\overrightarrow{ADmap(t)}$ leads our tool for surpassed performance awareness by intensifying aging effect in wirelength cost.

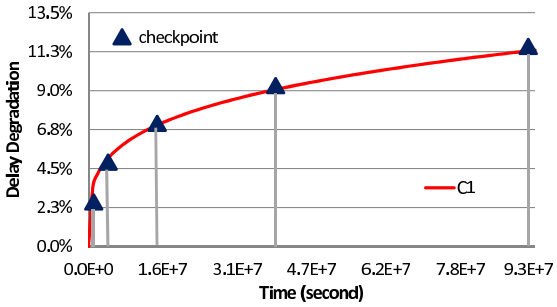
Using aging-aware floorplanner, the proposed framework generates a sequence of configurations to be mapped on FPGA in order to decelerate aging-induced performance and delay

degradation. Fig. 4.1a demonstrates that *aging rate* of resources in a configuration is faster at the beginning of execution. Hence, checkpoints are generated aperiodically (unequal timing steps) to capture fast aging rate early in time as well as to avoid unnecessary checkpoints later in time. At each checkpoint, if the new floorplan has a lower *aging rate* than the current floorplan and the critical path delay of design is not degraded beyond given performance constraint, the new configuration is accepted. Generating the configurations sequentially using the aging history ($\overrightarrow{ADmap(t)}$) not only allows the flooplanner to balance aging among resources but also to consider recovery phase in BTI-induced aging.

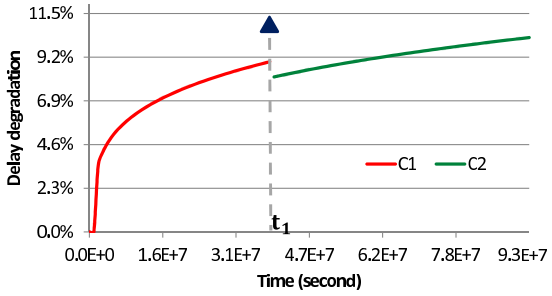
Experimental results show improvement in the delay degradation of underlying FPGA resources and in the degradation of critical-path delay on average by 53.2% and 17.5%, respectively compared to *aging-aware floorplanner* without reconfiguration in 3 years of execution.

4.1 Related work

Generating multiple configurations within the same region for balancing stress is proposed in [138, 167, 141, 32]. [138] proposes periodic swapping between two configurations. [167] presents multiple placements which swap between unused CLBs and used ones. Due to performance constraints, the swap is local and is limited to adjacent used and unused CLBs. [141] proposes various placement ideas to generate new configurations but mostly are local changes without using aging estimation models. A process variation and NBTI-aware placement strategy is proposed in [32]. The aforementioned works aim to reduce *stress time* without considering the impact of *temperature* on *aging rate*. Reconfiguration scheme in [167] does not provide any schedule for reconfiguration. Since aging history affects delay degradation, sequentially generating the configurations and scheduling each configuration is necessary to provide effective aging mitigation.



(a) Aperiodic checkpoints for 3 years.



(b) HCI-induced delay degradation mechanism.

Figure 4.1: Aging-induced Average Delay Degradation, $\Delta d(t)$, in FPGA (BTI and HCI).

4.2 Aging-induced delay degradation estimation in FPGAs

The delay degradation of transistors due to aging is modeled as: $d(t) = d_0 + \Delta d(t)$. d_0 is the pre-aged delay of transistor before any stress (never used before) and $\Delta d(t)$ is delay degradation function due to HCI and BTI aging effects until time t. Next, we describe how induced $\Delta d(t)$ due to BTI or HCI is computed for FPGA resources.

4.2.1 BTI aging effect

BTI has two phases named as *stress phase* and *recovery phase*. When a transistor is ON, it is in *stress phase*. At high temperature, BTI suggests trap generation which manifests as gradual increase in V_{th}. When a transistor is OFF, it is in *recovery phase*, which suggests trap recovery to decrease V_{th} gradually. Based on [8, 114, 21], the switching delay degradation (Δd) due to BTI is:

$$\Delta d_{BTI}(t) = A_{BTI} \times (ST_{t_0 \to t})^n \times e^{(\frac{-E_a}{kT})} \times d_0 \tag{4.1}$$

where, A_{BTI} is technology dependent factor, n is a constant depending on fabrication process, k is Botlzmann's constant, E_a is activation energy, T is temperature, d_0 is the pre-aged switching delay of the transistor, and $ST_{t_0 \to t}$ is the *stress time* till time t. $ST_{t_0 \to t}$ in BTI is equal to $SR \times (t - t_0)$. Essentially, *Stress Rate (SR)* for BTI is equal to Y, which is the duty cycle, i.e. the ratio of time that transistor is ON. Given, *Signal Probability* (SP), Y is equal to $1 - SP$ in NBTI, and equal to SP in PBTI. Since the *stress phase* is faster than *recovery phase*, increasing the duration of *recovery phase* helps to heal defected transistors better. $\Delta d_{BTI}(t)$ can be expressed as $\Delta d_{BTI}(ST_{t_0 \to t}, T)$.

4.2.2 HCI aging effect

HCI changes the current-voltage characteristic of transistor caused by acceleration of carriers within the electric field inside transistor channel. This increases V_{th}. Based on [8, 114, 21], delay degradation (Δd) due to HCI is:

$$\Delta d_{HCI}(t) = A_{HCI} \times (ST_{t_0 \to t}) \times t^{-0.5} \times e^{(\frac{-E_b}{kT})} \times d_0 \tag{4.2}$$

where, A_{HCI} is technology dependent factor, t is time, k is Botlzmann's constant, E_b is activation energy, T is temperature, d_0 is the pre-aged switching delay of the transistor, and $ST_{t_0 \to t}$ is the *stress time* till time t. $ST_{t_0 \to t}$ in HCI is equal to $SR \times (t - t_0)$. SR refers to activity rate, i.e., $\alpha \times f$, where α is activity factor of transistors and f is clock frequency. $\Delta d_{HCI}(t)$ can be expressed as $\Delta d_{HCI}(ST_{t_0 \to t}, T)$.

4.2.3 Aged-delay map

We introduce Aged-Delay Map matrix ($\overrightarrow{ADmap(t)}$) to record delay history of underlying nodes inside region R in FPGA. According to high level aging model for FPGA resources

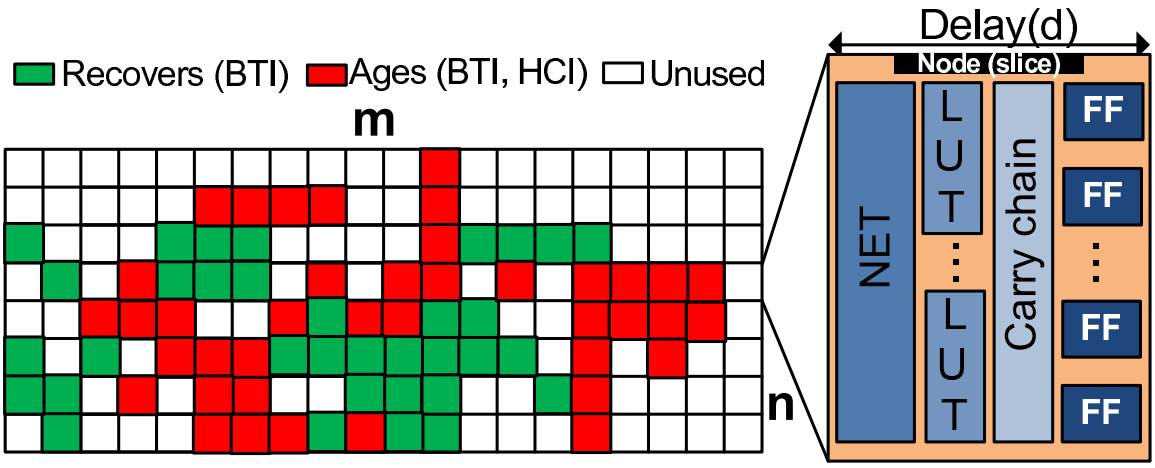


Figure 4.2: Aged-Delay Map, ADmap(t), of region R at time t_i

presented in [8], a resource node is defined as a basic logic element and its corresponding routing resources in FPGA architectures (e.g. slices in Xilinx Virtex). Each node is composed of set of *LUTs*, *FFs*, *NETs*, and a *Carry chain*. As shown in Fig. 4.2, $\overrightarrow{ADmap(t)}$ shows aged delay at time t corresponding to each resource node j in region R.

$$\overrightarrow{ADmap(t)} = \overrightarrow{d(t_0)} + \overrightarrow{\Delta d(t)} \tag{4.3}$$

The method in [8], along with Eq. 4.1 and Eq. 4.2, is deployed to compute $\Delta d_j(t)$ for each node j. Delay of each *used* node j is the longest path delay before aging ($d_j(t_0)$) inside the node plus its delay degradation until time t. Delay of each *used* node (colored as red or green in Fig. 4.2) either increases (*ages*) due to BTI and HCI mechanisms or decreases (*recovers*) if it is in recovery mode (only for BTI). *Unused* node (colored as white in Fig. 4.2) is referred to a node that has never been used ($ST = 0$). Its delay does not degrade (Eq. 4.1 & Eq. 4.2) until it is used for the first time ($ST \neq 0$). Delay degradation at each node j at time t is:

$$\Delta d_j(t) = \Delta d_{BTI_j}(t) + \Delta d_{HCI_j}(t) \tag{4.4}$$

As shown in Fig. 4.1a, when a node is used in one configuration (C_1), delay degradation is computed based on Eq. 4.4. When the node is used in two configurations (C_1 for $t < t_1$ and C_2 for $t > t_1$ in Fig. 1.b), delay degradation at t_1 is:

$$\Delta d_j(t_1) = \Delta d_{BTI_j}(ST_{t_0 \to t_1}, T_1) + \Delta d_{HCI_j}(ST_{t_0 \to t_1}, T_1) \tag{4.5}$$

where $ST_{t_0 \to t_1}$ and T_1 refers to *stress time* and temperature of the nodes under first configuration (C_1). Delay degradation at time $t > t_1$ is:

$$\Delta d_j(t) = \Delta d_{BTI_j}(ST_{t_0 \to t}, T_1, T_2) + \Delta d_{HCI_j}(ST_{t_1 \to t}, T_2) \quad + \Delta d_{HCI_j}(ST_{t_0 \to t_1}, T_1) \tag{4.6}$$

where T_1 and T_2 refer to the node temperature under configurations C_1 and C_2, respectively. $ST_{t_0 \to t_1}$ and $ST_{t_1 \to t}$ are *stress time* under C_1 and C_2, respectively. $\Delta d_{HCI_j}(t)$ calculation is divided for C_1 and C_2. In order to consider recovery effect from BTI aging effect, we compute delay degradation from t_0 (before *stress time*). Hence, we estimate Δd_{BTI_j} as:

$$\Delta d_{BTI_j}(t) = \Delta d_{BTI_j}(\frac{t_1}{t} ST_{t_0 \to t_1} + (1 - \frac{t_1}{t}) ST_{t_1 \to t}), T_{Avg.}) \tag{4.7}$$

where $T_{Avg.}$ refers to average temperature from $t_0 \to t$. If a node is idle in a configuration, the weighted average ST in Eq. 4.7 can capture the recovery during idle time. Assume t and t_1 are 8 and 6, respectively. If $ST_{0 \to 6}$ and $ST_{6 \to 8}$ are 3 and 0, respectively, $ST_{0 \to 8}$ is 2.25. This decrement in ST helps the node's delay to recover.

$\overrightarrow{ADmap(t)}$ is deployed to guide the floorplanner to recognize the nodes that are aging faster due to high temperature and/or higher ST so that the floorplanner would avoid mapping the new configuration to such stressed nodes. $\overrightarrow{ADmap(t)}$ needs to be updated at each time a new floorplan (i.e., configuration) is generated according to equations provided in this section. Given that the critical path in a design is composed of a series of the nodes, similar equations are deployed to compute delay degradation in critical paths. However, when a

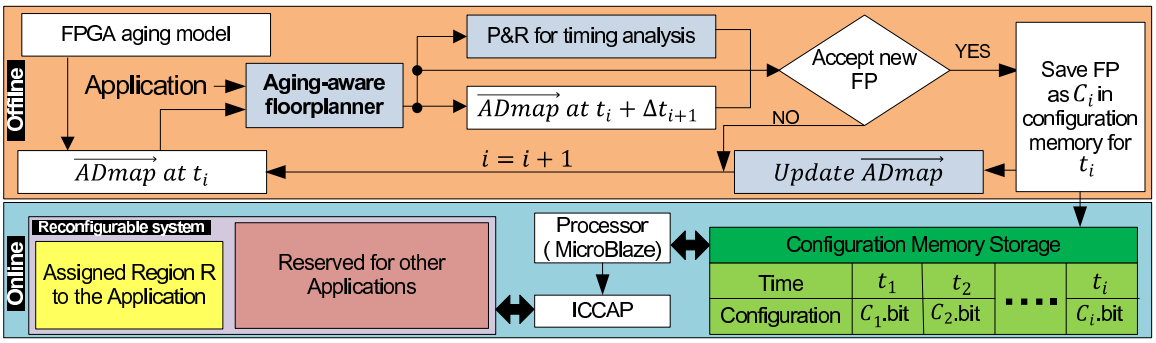


Figure 4.3: Aging-aware high-level physical planning and reconfiguration framework

new configuration is implemented, critical paths are changed and hence, the function in Fig. 4.1b is a discontinuous function at time t_1.

4.3 Problem formulation

- Given application description (block level), stress rate (node level), aging model (BTI and HCI), region R on FPGA and its corresponding $\overrightarrow{ADmap(t)}$ at t_0, and time interval $[t_0, t_n]$,
- Generate and schedule multiple configurations such that average aged-delay in region R at t_n is minimized and critical path delay degradation is bounded by α.

4.4 The proposed aging-aware physical planning and reconfiguration policy

Fig. 4.3 provides an overview of proposed framework which is composed of offline high level physical planning (floorplanning) and reconfiguration policy coupled with runtime reconfiguration to mitigate aging effects in region R. In FPGA device, region R is allocated for mapping the target application. The size of region R is assumed to be slightly larger than

minimum area required for application (e.g., 10%-20%). In offline phase, region R is checked for delay degradation of underlying resources (chip lifetime) as well as aging-induced critical path (performance) degradation of application running in this region. At each checkpoint, t_i, a new floorplan is generated by aging-aware floorplanner. The new floorplan is evaluated based on its impact on *delay aging rate* of resources as well as delay degradation along critical paths in design. The decision step decides whether to reconfigure region R to the new floorplan or to continue the execution on current configuration.

According to aging-induced delay model in Section 4.2, delay degradation is at much higher pace shortly after t_0 and is at much lower pace later in time (Fig. 4.1). Instead of periodic checkpoints (equal timing steps), we introduce checkpoints when delay degradation of resources or critical path degradation has reached a threshold (*aperiodic checkpoints*). After checkpoint t_i, the next checkpoint is at $t_i + \Delta t_{i+1}$ referred to as checkpoint t_{i+1}. $\overrightarrow{ADmap(t)}$ is updated at each checkpoint t_{i+1} based on the accepted configuration.

The outcome of offline phase is a sequence of *aging-aware configurations* and *reconfiguration schedule time table* for execution time window of $[t_0, t_n]$. During runtime, reconfigurable system is reconfigured according to time table sequentially. Generating the configurations sequentially enables to include aging history and current delay degradation in generating the next configuration. This method is more effective to avoid highly stressed resources and to migrate hotspots across the region R.

The proposed aging-aware floorplanner considers aging effect in two ways:

1. It is guided by $\overrightarrow{ADmap(t)}$ to avoid resources that are currently aged higher and to prefer under utilized resources;

2. The thermal simulation inside the tool allows to select configurations with fewer hotspots and more balanced thermal behavior which leads to decrease in *aging rate*.

Algorithm 3 Aging-aware floorplanner and reconfiguration policy

Input: $\overrightarrow{ADmap(t)}$, Power consumption matrix $\overrightarrow{P}$, stress-rate matrix $\overrightarrow{SR}$, Clock frequency f, Duration of application execution D, number of checkpoints n

Output: Aging-aware floorplans list $\{FP\}$, Scheduling time-table $\{t\}$

1: $i \leftarrow 1$;
2: **while** i $\leq$ n **do** ▷ **SA-based floorplanner with adapted cost function (Step 1)**
3: $fp \leftarrow$ FindFp (FP_{i-1});
4: $cost \leftarrow$ FindCst(fp, $\overrightarrow{ADmap(t)}$) ▷ Eq. 4.10
5: **while** T is not Frozen **do**
6: $fp \leftarrow$ FindFp(fp);
7: $newcost \leftarrow$ FindCst(fp, $\overrightarrow{ADmap(t)}$) ▷ Eq. 4.10
8: **if** newcost < cost or exp(-(newcost − cost)/T) > Random[0,1) **then**
9: $cost = newcost$;
10: **end if**
11: $\{kbestFP\} \leftarrow$FindkBestFps($fp, cost$); ▷ Select k best candidates floorplans
12: **end while**
13: **for all** Floorplan FP_k in $\{kbestFP\}$ list **do** ▷ **Find the best aging-aware floorplan (Step 2)**
14: $tmp_k \leftarrow$ FindTmp(FP_k, $\overrightarrow{P}$) ▷ call HotSpot
15: $agcost_k \leftarrow$ CalAgCst(FP_k, tmp_k, $\overrightarrow{SR}$, f, D) ▷ Eq. 4.5 and Eq. 4.6
16: $Qcost_k \leftarrow \frac{agcost_k}{n}$;
17: **end for**
18: $bestFP_{t_i} \leftarrow$ FindBestFp($\{kbestFP\}$, $\overrightarrow{tmp}$, $\overrightarrow{Qcost}$);
19: $\Delta t_i \leftarrow$FindRecTime($bestFP_{t_i}$, $Qcost_{bestFP_{t_i}}$);
20: $chagcost_{t_i+\Delta t_i}^{bestFP_{t_i}} \leftarrow$ CalChpAgCst($bestFP_{t_i}$, $\overrightarrow{ADmap(t)}$) ▷ Eq. 4.11
21: **if** i > 1 **then**
22: **if** $chagcost_{t_i+\Delta t_i}^{bestFP_{t_i}} < chagcost_{t_i+\Delta t_i}^{bestFP_{t_{i-\Delta t_{i-1}}}}$ **then**
23: $frequency \leftarrow$ExFreq($bestFP_{t_i}$);
24: **if** f − frequency <= threshold **then**
25: Add $bestFP_{t_i}$ to $\{FP\}$ list;
26: UpADmap($bestFP_{t_i}$, $\overrightarrow{ADmap(t)}$, $t_i + \Delta t_i$);
27: **else**
28: UpADmap($bestFP_{t_{i-\Delta t_{i-1}}}$, $\overrightarrow{ADmap(t)}$, $t_i + \Delta t_i$) ;
29: **end if**
30: **else**
31: UpADmap($bestFP_{t_{i-\Delta t_{i-1}}}$, $\overrightarrow{ADmap(t)}$, $t_i + \Delta t_i$) ;
32: **end if**
33: **end if**
34: $t_i \leftarrow t_i + \Delta t_i$;
35: $\{t\} \leftarrow t_i$;
36: $i \leftarrow i + 1$; ▷ **Go to next period**
37: **end while**

Algorithm 3 shows the pseudo code for the proposed method. The algorithm inputs are $\overrightarrow{ADmap(t)}$, matrix of power consumption ($\overrightarrow{P}$) of nodes, $\overrightarrow{SR}$, clock frequency (f), Duration of application execution D, and number of checkpoints n. The objective function is a lexicographic function. It first minimizes total *aging-aware wirelength*, and next, it minimizes the average delay degradation in region R. The algorithm's output is a sequence of floorplans and the corresponding reconfiguration schedule time table.

The traditional cost function of floorplanners (Eq. 4.8) does not consider aging influence on underlying nodes inside the chip. Hence, finding a floorplan based on this cost function may cause timing violation in critical paths over time. The nodes along a path age differently due to different temperature, stress time and aging history. The accumulation of heterogeneously aged delays of nodes causes timing failure. This implies that *avoiding* highly-aged nodes can reduce *aging rate* along critical paths.

$$cost = \alpha \times area + \beta \times TotalWirelength\,(or\,timing) \tag{4.8}$$

An *aging-aware coefficient*, defined as γ, is proposed as new weight for wirelength (timing) in cost function. This guides the floorplanner to avoid highly aged nodes. Also, highly aged nodes may get chance to recover partly due to BTI recovery mechanism. γ is formulated as:

$$\gamma_l^{t=t_i} = (\frac{1}{L}\sum_{j=1}^{L}\frac{d_j^{t=t_i}}{d_j^{t=t_0}}) \tag{4.9}$$

where, L is number of nodes along each net l and $d_j^{t=t_i}$ is delay of node j at time t_i. $\gamma_l^{t=t_i}$ is average *aging rate* of nodes along net l at time t_i. To intensify the impact of $\gamma_l^{t_i}$, it is powered by m. Hence, the aging-aware new cost function is:

$$AgeAwareCost(t_i) = \alpha \times area + \beta \times \sum_{l \in Nets}(\gamma_l^{t_i})^m \times wirelength\,(timing) \tag{4.10}$$

The aging-aware floorplanner is a *Simulated Annealing* floorplanner (lines 3-16), which is composed of two steps:

1. Find k best floorplan candidates based on Eq. 4.10 using aging history, $\overrightarrow{ADmap(t)}$, until t_i (lines 3-11).

2. Find the one between candidates that minimizes the aging-rate delay for t_{i+1} (lines 12-16).

Additionally, the amount of delay degradation for each application not only depends on its behavior (signal activities) that affects its power consumption and accordingly its temperature but also depends on where each block is floorplanned considering $\overrightarrow{ADmap(t)}$. In other words, different floorplans for same application with same behavior (signal activities and inputs) have different temperature maps considering the history of the chip. Different power consumption of blocks inside the floorplan and their relative position to each other will change the temperature map of the design. For instance, assume a trivial floorplan with four blocks inside it that are named as A, B, C, D at positions $PS^A_{x,y}, PS^B_{x,y}, PS^C_{x,y}, PS^D_{x,y}$ respectively. Any acceptable permutation of the positions lead to different temperature maps of the final floorplan.

The selected aging-aware floorplan candidates, $\{kbestFP\}$, in step 1 are sorted based on aging-aware cost function (Eq. 4.10). However, they have different permutation of blocks' position that results to different temperature maps and different *aging rates*. Using temperature maps ($\overrightarrow{tmp}$), the tool is able to find the best aging-aware floorplan between the candidates. Due to time consuming temperature extraction, k best floorplan candidates are selected by $FindkBestFps()$ (line 10). To find the best *aging-aware floorplan* between candidates aging costs (Eq. 4.6) are calculated. The $FindTmp()$ function uses HotSpot tool [137] for thermal simulation. Temperature matrix ($\overrightarrow{tmp}$) is used by $CalAgCst()$ to calculate the aging cost of each candidate.

To find the next checkpoint, *quantized* aging costs for each candidate is required to be calculated. Dividing the aging cost of each candidate floorplan ($agcost_k$) by the number of demanded checkpoints (n) results $Qcost_k$. Based on these quantized costs, $Qcost_k$, the best *aging-aware floorplan*, $bestFP_{t_i}$, for period i is found using $FindBestFp()$ function.

Finally, a reconfiguration policy (lines 17-30) decides whether to stay on current configuration or reconfigure to the new one. The quantized cost of best floorplan at time t_i, $Qcost_{bestFP_{t_i}}$ is passed to $FindRecTime()$ to find the time of next checkpoint, Δt_i (line 17). Next, $CalChpAgCst()$ calculates the *average aging cost* of the new floorplan in Region R as follows:

$$chipagcost_{t_i}^{FP} = (\frac{1}{No.\,used\,nodes}) \sum_{j=1}^{No.\,used\,nodes} (d_j^{t_i}) \tag{4.11}$$

where, $No.\,used\,nodes$ refers to the total number of used nodes by different floorplans till time t_i, and $d_j^{t_i}$ is the delay of node j at t_i. If the new floorplan aging effect on the chip is less than the current running floorplan (line 20) and critical path delay (maximum clock frequency) meets application timing constraints (line 22), it is accepted for reconfiguration and it is added to time table. Otherwise, current floorplan will continue its execution until next checkpoint ($t_{i+1} = t_i + \Delta t_i$). In both cases, the $\overrightarrow{ADmap(t)}$ of region is updated based on *accepted* new floorplan or current floorplan at (t_{i+1}). The list of checkpoints {t} will be stored at the end of algorithm (line 30).

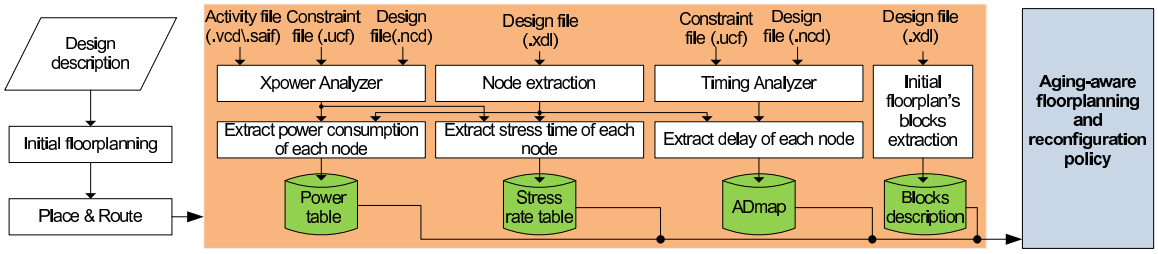


Figure 4.4: Generation of required data for aging-aware floorplanner

4.5 Experiments

4.5.1 Setup

Layout-level information of FPGA resources is extracted using *Xilinx Design Language* (XDL) format of placed-and-routed application. This information along with power and timing information of the application (i.e., switching activity, signal probability, and clock frequency) is fed into algorithm 3 as input in order to generate list of aging-aware floorplans and their timing table.

As illustrated in Fig. 4.4, an initial floorplan is generated by $PlanAhead\,Xilinx$ tool. After place-and-route, the back-annotated *Native Circuit Design* (NCD) into HDL-based language is fed to the logic simulator tool ($ModelSim$ or $ISim$) accompanied with test-bench to extract activity file (.vcd/.saif). Using XDL file, placement and corresponding routing information of *used* nodes (SLICE) are stored in a table. This table is used along with power report by $XpowerAnalyzer$, activity file, and timing report by $TRACE$ to generate matrices $\overrightarrow{P}$, $\overrightarrow{SR}$, and $\overrightarrow{ADmap(t)}$ for each *used* and *unused* node inside a *predetermined* region R in FPGA.

Finally, the description of each block, i.e., the number of *used* nodes, interconnects, size of each block (number of nodes), aspect ratio, and the possibility of rotation during floorplanning is generated from XDL file. These data are fed into the proposed aging-aware

Table 4.1: Selected benchmarks characteristics

App.	# LUTs	# FFs	# Nodes	# Blocks	Power Cons. (mw)	$T_{Crit}(ns)$
AES	2622	530	870	20	460.3	7.3 (136.9 MHz)
DCT	6790	3709	1960	65	850.2	7.8 (128.2 MHz)
JPG	14331	4755	4300	67	1270.2	9.4 (106.4 MHz)

floorplanning and reconfiguration policy to find the list of best aging-aware configurations at each checkpoint and their sequential time-table ordering considering aging history of the chip, temperature, and *stress time* (ST) for each node. In aging model, the values of A_{BTI} and A_{HCI} are defined such that the maximum amount of degradation in 5 years is 15% at worst case (PMOS transistors always ON ($SP = 1$), maximum frequency (AR=500 MHz) at temperature 380 Kelvin). The target FPGA device is Virtex6 (xc6vlx240t) at 40 nm technology.

To investigate the impact of proposed method, three different benchmarks are selected from multimedia and security applications. The characteristics of these benchmarks are shown in Table 4.1. Each block is a datapath component such as multiplier and adder (generated by *CoreGen* in *ISE* 14.7).

4.5.2 Results and discussion

For each benchmark, the *average* node delay degradation in region R (Δd) and performance (application critical path degradation) (ΔT_{crit}) in 3 years (9.3E+7 seconds) of execution are evaluated. The following four aging-aware schemes are compared:

1. Our proposed *aging-aware floorplanner* without reconfiguration (FP, w/o,Rec.): The first configuration runs for 3 years without any reconfiguration. The aging-aware floorplan is executed for 3 years in region R (Scheme 1).

2. Our proposed *aging-aware floorplanner* and reconfiguration scheme (FP,w/, Seq.,Rec.,and, Aper.,Chk.): The proposed method in this paper (Scheme 2).

3. Our proposed *aging-aware floorplanner* and reconfiguration scheme (FP,w/, Seq.,Rec.,and, Per., Chk.): The proposed method in this paper, but with *periodic* checkpoints at every 6 months (Scheme 3).

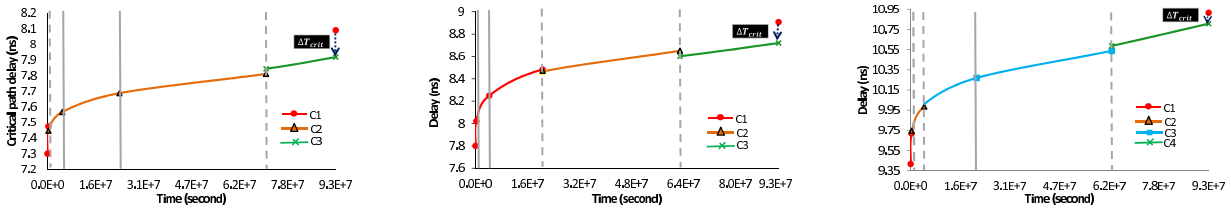
4. Multiple configuration generation using floorplanner and LP-based combinational configuration scheduling (FP, w/, Com.,Rec.): Using conventional floorplanner, we first generate multiple configurations. Next, using their *Stress Rate* (SR) matrices, we apply the LP-based optimization as presented in [168] to find the percentage of total execution time assigned to each configuration for 3 years (Scheme 4).

Table 4.2 summarizes the results. The results show that our proposed approach (scheme 2) slows down node delay degradation (chip lifetime) by 53.2%, on average, compared to *aging-aware floorplan* without reconfiguration (scheme 1). Hence, the proposed reconfiguration scheme using multiple aging-aware floorplans mitigates the *aging rate* further, even when an *aging-aware floorplan* has been used in scheme 1. In addition, the proposed method improves the *maximum* aged node delay of AES, DCT and JPG by 15.7%, 17.8%, and 18.0%, respectively.

Furthermore, scheme 3, in comparison to scheme 1, reduces node delay degradation (chip lifetime) by 46.7%, on average. Moreover, this scheme improves the *maximum* aged node delay of AES, DCT and JPG by 14.1%, 16.2%, and 16.6%, respectively. Since the checkpoints are predetermined steps (every 6 months) by user in scheme 3, it is not able to detect the high *aging rate* at the beginning of stress and it also applies redundant checkpoints later in lower *aging rate* phase.

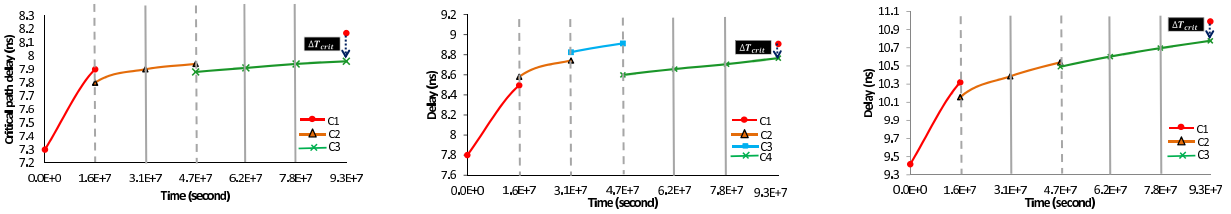
In comparison with scheme 1, the average node delay degradation has reduced only 11%, on average, in scheme 4, while the reduction in our approach (scheme 2) is 53.25%. In scheme 4,

the combinational reconfiguration scheme does not impose any ordering on the configurations and hence, it does not consider the history of aged nodes on the chip to generate the next configurations. In addition, the recovery from BTI-induced aging cannot be captured using this technique. As a result, our *aging-aware* framework is able to mitigate the aging effect on the delay degradation more effectively.



(a) AES (b) DCT (c) JPG

Figure 4.5: Critical path delay with aperiodic checkpoints for reconfiguration



(a) AES (b) DCT (c) JPG

Figure 4.6: Critical path delay with periodic checkpoints for reconfiguration

The last five columns in Table 4.2 show the effectiveness of our proposed method on critical path delay degradation over 3 years of execution. Compared to the scheme 1, our proposed methods (scheme 2 and scheme 3) show 17.5% and 16% less degradation in critical path delay on average. Given that scheme 4 does not consider critical path delay during reconfiguration, the results show that the critical path delay degradation after 3 years is higher than scheme 1. To the best of our knowledge, this is the first work that has considered and investigated the impact of aging on critical path in aging-aware physical planning.

To further investigate the impact of proposed method on the performance of different designs,

Table 4.2: Results on aging induced average delay degradation improvement and performance improvement

	$\Delta d(ns)$							$Performance\ or\ \Delta T_{crit}(ns)$						
Bench	FP w/o Rec.	FP w/ Seq. Rec. Per. Chk. (ours)		FP w/ Seq. Rec. Aper. Chk. (ours)		FP w/ Comb. Rec. [168]		FP w/o Rec.	FP w/ Seq. Rec. Per. Chk. (ours)		FP w/ Seq. Rec. Aper. Chk. (ours)		FP w/ Comb. Rec. [168]	
AES	0.0622	0.0350	43.72%	0.0312	49.84%	0.0554	10.94%	0.8700	0.6600	24.13%	0.6180	28.96%	0.9375	-6.75%
DCT	0.0765	0.0423	44.70%	0.0370	51.63%	0.0679	11.24%	1.1078	0.9660	12.80%	0.9200	16.95%	1.2085	-9.09%
JPG	0.0917	0.0455	50.38%	0.0394	57.03%	0.0813	11.34%	1.5738	1.3600	13.58%	1.3902	11.66%	1.6404	-4.23%
AVG.	0.0768	0.0409	**46.74%**	0.0359	**53.25%**	0.0682	**11.19%**	1.1839	0.9953	**15.92%**	0.9760	**17.56%**	1.2621	**-6.61%**

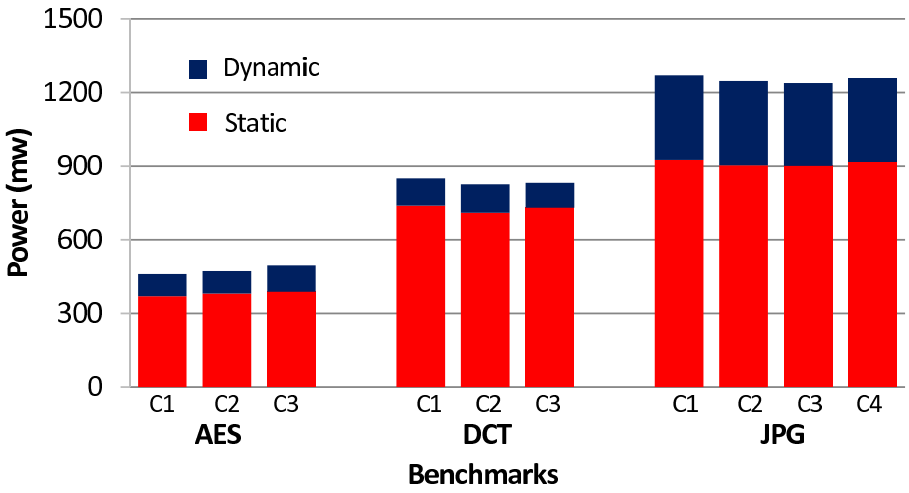


Figure 4.7: Power consumption of different accepted configurations in each benchmark

Fig. 4.5 and Fig. 4.6 illustrate critical path delay for aperiodic (scheme 2) and periodic (scheme 3) checkpoint approaches respectively. The vertical lines show the checkpoint times, while the dotted lines show when a new floorplan is accepted based on proposed algorithm 1 and decision making for reconfiguration. In scheme 2, the required number of reconfigurations for AES, DCT and JPG benchmarks are 2, 2 and 3, respectively. As a result, 3 (AES), 3 (DCT), and 4 (JPG) sequences of configurations are accepted for runtime reconfiguration. Although newly accepted configuration at each checkpoint may have a higher critical path delay at the beginning, the *aging rate* is lower and hence, the critical path delay degrades more slowly.

As shown in Fig. 4.5, our proposed scheme results in less critical path delay degradation (ΔT_{crit}) compared to scheme 1 after 3 years. In comparison with scheme 3 (as shown in Fig. 6), our proposed scheme is able to slow down *aging rate* earlier, which results in less delay degradation with fewer reconfigurations. In Fig. 5.a, the configurations generated at second and third checkpoints are rejected due to degradation in performance. At fourth checkpoint, a configuration with lower aging rate and acceptable performance is implemented on FPGA.

Fig. 4.7 shows the power consumption of different configurations of each benchmarks in

aperiodic checkpoint approach. This graph shows that transition to different aging aware configurations does not impose significant power cost. The reconfiguration time using ICAP (32-bitwidth), at 60 MHz is around 2.3 ms on average for selected benchmarks, which is a negligible time for 3-year continues operation on FPGA. Same trend happens for regular checkpoint approach.

4.6 Chapter summary

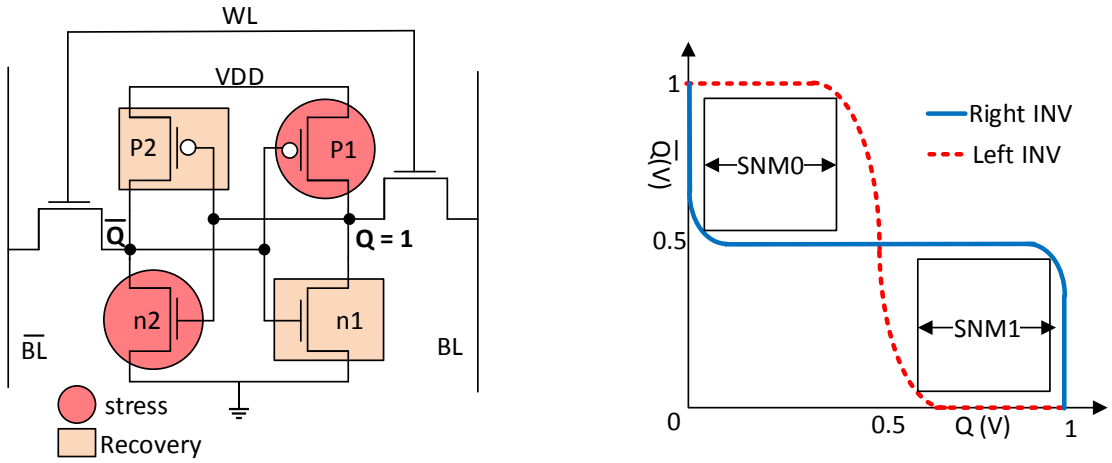
In this chapter, an aging-aware floorplanner and a reconfiguration policy to mitigate aging on the chip lifetime and performance are proposed. Aged Delay Map matrix, $\overrightarrow{ADmap(t)}$, is introduced to record aging history of FPGA resources considering temperature and stress time in order to guide the aging-aware floorplanner and reconfiguration policy. Our method is able to mitigate *aging-rate* of resources and critical path delay by 53.25% and 17.56%, respectively, in selected computationally intensive benchmarks.

Chapter 5

Stress-aware Boolean matching to protect SRAM cells in FPGAs

Runtime reconfigurable architectures are broadly employed to accelerate applications ranging from datacenters, vision, and safety critical applications such as medical, automotive, and space applications [67, 66, 65, 50, 168]. Aging mechanism in transistors, as a major challenge in advanced silicon technology, jeopardizes the reliability, lifetime, power, and performance of SRAM-based reconfigurable architectures, as well [2, 138, 169, 65]. The amount of stress (S) in BTI is measured as the time ratio that transistor is ON to the total time (i.e. it is the duty cycle of the transistor in a time period and $0 \leq S \leq 1$).

BTI leads to *Static Noise Margin* (SNM) reduction of SRAM cell. SNM, which is a function of V_{th} [133, 38], is the stability and reliability measurement of SRAM robustness against noise. As shown in Fig. 5.1a, if the SRAM content is one, Q=1, then (p1, n2) are under stress (i.e. SNM1 reduces) and if the SRAM content is zero, Q=0, then (p2, n1) are under stress (i.e. SNM0 reduces). We defined S1 and S2 as the amount of stresses on (p1, n2) and (p2, n1) transistor pairs, respectively. Additionally, SNM of an SRAM cell is equal



(a) Under stress and recovery transistor pairs in SRAM when Q=1.

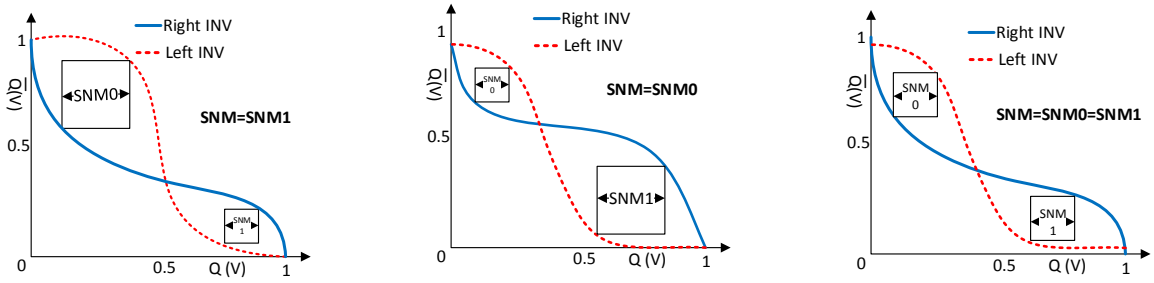
(b) SNM=Min(SNM0, SNM1).

Figure 5.1: Architecture of SRAM and its SNM.

to $min(SNM1, SNM0)$. This means that the balanced stress, i.e. S1=S2=0.5, leads to optimal and balanced SNM reduction during a period of time (Fig. 5.1b). To this end, flipping the content of SRAM balances stress as well as SNM. Otherwise, as shown in Fig. 5.2, imbalanced stress ($S1 \neq S0$) results in imbalanced SNM1 and SNM0 and consequently SNM reduction of the SRAM cell. The higher the SNM, the more stable and reliable the SRAM.

SRAM cell is the main element in an FPGAs as configuration bit to implement the desired logic inside *Look-up-Tables* (LUTs). The SNM reduction in SRAM cells leads to data corruption and *Soft Error Rate* (SER) raise in FPGAs [84, 95, 157, 34]. This makes FPGAs more susceptible to high energetic particles and noise, that can flip the content of SRAMs and change the implemented logics in LUTs.

This chapter presents STABLE, a three-step post-synthesis stress aware method to reduce the impact of BTI on the SNM of SRAM cells in FPGAs. To this end, we extract the *Data Flow Graph* (DFG) of the placed-and-routed design. We then partition the DFG to different cones in a cone construction phase. Each cone represents a function and is composed of a



(a) SNM imbalance when Q=1 for a long time (S1=1), SNM=SNM1.

(b) SNM imbalance when Q=0 for a long time (S0=1), SNM=SNM0.

(c) Balanced SNM when Q=0 and Q=1 equal amount of times (S1=S0=0.5), SNM=SNM1=SNM0.

Figure 5.2: SNM reduction in different scenarios.

root LUT and the connected LUTs of that root. Through SAT-based Boolean matching, we find a new configuration ($C2$) as apposed to the main configuration ($C1$), where all the bits are flipped ($C2 = \overline{C1}$) and the application's functionality is maintained ($F2 = F1$). $C2$ will be swapped with $C1$, through periodical FPGA reconfiguration, in order to recover and balance SNM reduction fo SRAMs.

At step 1, the possibility of flipping all configuration bits inside each cone is explored, if their functionalities are preserved. Obviously, configuration bits of all the cones cannot be flipped due to change in their functionality. In addition, all LUTs could not be included into a cone in the cone construction phase. Therefore, at second step, for the remaining LUTs, we check if they have unused SRAM cells (i.e. they are partially-used LUTs). These unused SRAM cells can be utilized to store flipped configuration bits of such LUTs that can be swapped with them. Till the end of step two, LUTs' configurations are flipped without impacting the optimized design on FPGA. This is due to either STABLE flips LUT's configuration bits inside the cones (step 1) or swaps them with flipped ones in their unused cells (step 2). At third step, if an LUT has not passed any of the two previous steps (i.e. they are fully-used LUTs), we look for a nearest unused LUT as spare, where its flipped configuration bits is stored.

Our proposed methodology is deployed after placement and routing (i.e. post-synthesis) of the optimized design by commercial tools on FPGA. Therefore, the area, power, and performance overheads are negligible. Our extensive experimental analysis on Xilinx Virtex-6 (40nm technology) for several benchmarks demonstrates 66.32% and 67.23% improvements on average in SNM reduction (ΔSNM) and Soft-Error-Rate raise (ΔSER), respectively. To the best of our knowledge, this is the first effort to reduce the impact of BTI on SRAMs in FPGA LUTs through Boolean matching and unused cells inside each LUT.

In next section, the BTI impact on SRAM cells is detailed. Section 5.1 elaborates susceptibility of FPGA to BTI. In Section 5.3, we review the related works. Section 5.4 and 5.5 discuss the Boolean matching preliminaries and the problem formulation, respectively. In Section 5.6 we present STABLE, our stress-aware post-synthesis method. After that, the experimental analysis comes in Section 5.7. Finally, Section 5.9 summarizes the chapter.

5.1 BTI aging impact on SRAM

In this section the BTI aging analytical model is elaborated, which shows how BTI can increase the V_{th} of a transistor. Moreover, we show that how delay degradation at transistor level due to BTI can impact the SRAM cell characteristic and how aging manifests itself at SRAM's logic level.

5.1.1 BTI aging modeling

BTI is the dominant aging mechanism in the advanced silicon technology [2, 138, 169, 65], which increases the threshold voltage (V_{th}) of a transistor where it will be manifested as delay degradation at logic level. BTI is a static mechanism that happens when the transistor is ON for a prolonged time. It is composed of two phases: a stress phase and a recovery phase [156,

21]. During the stress phase (i.e. when transistor is ON), and in high temperature, BTI occurs due to the generation of traps at the Si-SiO2 interface, which gradually increase V_{th}. While, during the recovery phase (i.e. when transistor is OFF), some of these traps are eliminated and partially recover the shift on V_{th}. Based on [156, 151, 143], the delay degradation due to BTI stress phase can be simplified as:

$$\Delta V_{th\_S} = (K_v\sqrt{t_S} + \sqrt[2n]{V_{th_{t_0}}})^{2n} \tag{5.1}$$

where, t_S is the amount of time that transistor is under stress; K_v is dependent to temperature (T), V_{dd}, and electrical field; n is the time exponent parameter which is 1/6 for H2 diffusion; and $V_{th_{t_0}}$ is the intrinsic V_{th} of transistor at time zero. The delay recovery is:

$$\Delta V_{th\_R} = V_{th\_S}(1 - \frac{2\zeta_1 t_e + \sqrt{\zeta_2 C t_R}}{(1+\delta)t_{ox} + \sqrt{Ct}}) \tag{5.2}$$

where, t_R is the time that transistor is OFF and under recovery, t_{ox} is the oxide thickness, t_e is the effective oxide thickness, t is the total time, C is dependent to the temperature, and ζ_1, ζ_2, and δ are constants. A long term cycle-to-cycle model is derived in [21]. Hence, the V_{th} degradation and recovery for clock cycle i based on Eq. 5.1 and Eq. 5.2 can be computed as $\Delta V_{th\_S_i}$ and $\Delta V_{th\_R_i}$ as (3) and (4), respectively:

$$\Delta V_{th\_S_i} = (K_v\sqrt{\alpha T_{clk}} + \sqrt[2n]{V_{th\_R_{i-1}}})^{2n} \tag{5.3}$$

and,

$$\Delta V_{th\_R_i} = V_{th\_S_i}(1 - \frac{2\zeta_1 t_e + \sqrt{\zeta_2 C(1-\alpha)T_{clk}}}{(1+\delta)t_{ox} + \sqrt{iCT_{clk}}}) \tag{5.4}$$

where, α is the duty cycle of the transistor for the time period of T_{clk} and $i = t/T_{clk}$. The duty cycle of a transistor is the ratio of time that it is ON and under stress (S), i.e.

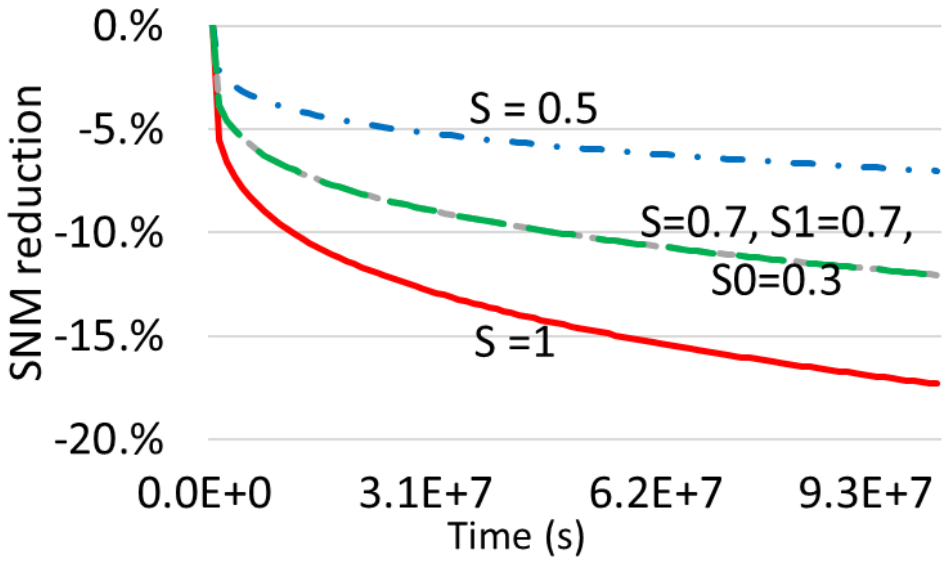


Figure 5.3: Impact of different stress (S) amount on SNM reduction at worst case (T=375K) after three years in 40nm technology. $SNM_{t_0} = 250$mV.

$S = \alpha, 0 \leq \alpha \leq 1$. It should be noted that NMOS transistors are affected by *Positive BTI* (PBTI) and PMOS transistors are affected by *Negative BTI* (NBTI). The Table 2 in Appendix A shows more details about BTI.

5.1.2 BTI-induced SNM reduction in SRAM

As shown in Fig. 5.1a, 6-transistor SRAM cells is composed of two inverters which store complementary values of a bit (Q in Fig. 5.1a). Increase in V_{th} of SRAM's transistors, reduces its SNM [84, 95, 157, 34]. SNM is the minimum DC noise voltage that is able to change the state of the SRAM cell (i.e. flip it). SNM can be estimated by the proposed graphical method in [133]. As shown in Fig. 5.2, SNM is computed as the maximum side of largest square enclosed between the two static characteristics curves of SRAM cells. i.e. $SNM = Min(SNM0, SNM1)$.

Based on Fig. 5.1, when the SRAM value is one (Q=1), p1 and n2 transistors are under stress. Therefore, SNM1 reduces due to BTI. Similarly, when the SRAM value is zero (Q=0),

p2 and n1 transistors are under stress. Therefore, SNM0 reduces due to BTI. Based on Eq. 5.3 and Eq. 5.4, S (or α) is a measurement for stress of a transistor. We define S1 and S0 as the duty cycle of under stress transistors when Q=1 and Q=0, respectively. Since the two pairs of transistors in SRAM cell (Fig. 5.1a) are complement of each others, we can conclude that $S1 = 1 - S0$ for a period of time t and $S = Max(S0, S1)$. For example if S1 is 0.25 then S0 is 0.75, which means 25% of the time SRAM value is one and 75% of the time SRAM value is zero. In this scenario, SNM0 is less than SNM1 and SRAM's SNM will be equal to SNM0 (Fig. 5.2b). It can be concluded that when S1 and S0 are equal (i.e. 0.5) is the fair, and optimum, case which all transistors experience the same amount of stress as well as recovery (Fig. 5.2c). In such as case, SNM has its maximum *possible* value which is equal to SNM1 (or SNM0). In all, balanced stress leads to balanced SNM0 and SNM1 and maximum amount of SRAM's SNM.

Therefore, the reduction of SNM is the sign of aging in SRAM cells, which decreases their hold, read and write stability that may result in timing failures. This leads to a *Soft Error Rate* (SER) increment in SRAM-based FPGAs by a factor of two [156]. *âĂIJThe SER is measured in Failure in Time (FIT) given by the number of errors per billion hours of device operationâĂİ* [150]. SER and SNM correlation due to variation in V_{th} of SRAM's transistors is evaluated using the Pearson correlation coefficient [150]:

$$SER = -a \times SNM + b \tag{5.5}$$

where $a = 1.2 \times 10^3$ and $b = 1.3 \times 10^3$ for 40nm technology.

In sum, flipping the value of SRAM gives an opportunity to stressed transistors to recover and balance aging by transferring stress to other pair transistors (Fig. 5.1 and Fig. 5.2). However, as detailed in [33, 71], with the balanced stress, in 40nm technology, 12% to 16% reduction in the SNM is unavoidable after 5 years, which leads to 3.6% to 4.8% raise in

SER. This is worse when SNM reduction is imbalanced, which necessitates a proper stress leveling. For example, in our experiments as shown in Fig. 5.3, in the worst case scenario (temperature = $375K$) when the stress is balanced (S=0.5) the SNM reduction is more than 7% after 3 years. Additionally, when the stress is imbalanced S=S1=0.7 (i.e. S0=0.3) the SNM reduction is more than 12%. This amount is more than 17% if S=S1=1 (i.e. S0=0).

5.2 FPGA susceptibility to aging

Millions of SRAM cells are used to store configuration bits in the state-of-the-art reconfigurable architectures (e.g. FPGA) for customizing logic in LUTs and switch boxes. As shown in Fig. 5.4, each LUT is composed of configuration bits, which are basically SRAM cells to store the logic, and a Mux-tree to select the proper bit as an output. The illustrated LUT has two inputs and four SRAM cells as configuration bits, which are programmed as an XOR logic. Both components, Mux-tree and SRAM cells, are susceptible to aging mechanism (red thunders in Fig. 5.4). Aging impact on Mux-tree increases the delay of LUT which can affect the critical path delay of implemented circuit along it. This will result in lower performance or timing failure. While aging in SRAM leads to lower SNM and higher SER. Moreover, the implemented logic by SRAMs inside LUT does not change for a prolonged time unless a reconfiguration happens. This makes them even more susceptible to BTI aging mechanism, as elaborated in Section 5.1. The constant stress (S=1) on SRAM cell configuration bits reduces their SNMs and increases SER.

Runtime reconfiguration in FPGAs gives the opportunity to the designer to change the implemented design on FPGA. This specification has been utilized to increase the reliability of FPGAs [67, 65]. As discussed in Section 5.1, by flipping the SRAM content we can balance stress in a given period of time t. To this end, the SRAM content should be altered for half of the time, $\frac{t}{2}$. Since by flipping SRAM values we basically invert the logic inside LUT, the

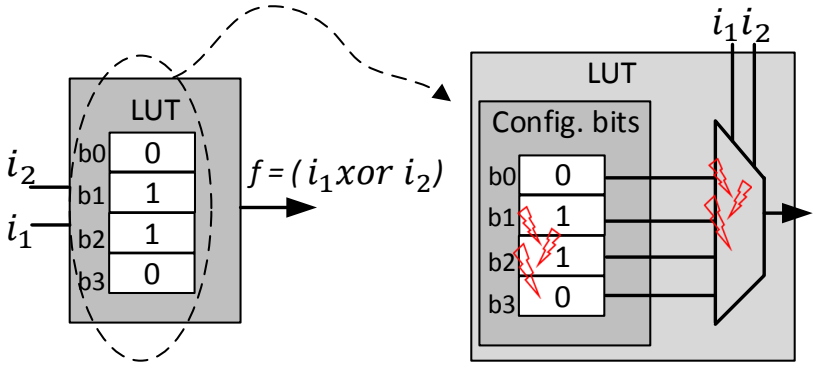


Figure 5.4: LUT components and their sensitivity to aging.

question is how to preserve the circuit's functionality at the same time.

5.3 Related works and motivation

Soft error happens due to high energetic particles and noise in SRAM cell of FPGAs and changes implemented application's functionality, which can cause failure in the system. This necessitates FPGA reconfiguration for scrubbing the faults [140]. When *Soft Error Rate* (SER) is high in harsh environments such as space or automotive applications we need to reconfigure FPGAs more frequently. This impose significant overheads to the system. There have been extensive studies for reliable design on SRAM-based FPGAs against soft error to reduce the overhead and cost of fault detection and correction [67, 139, 28, 55]. For example, authors in [67] proposed HAFTA, which duplicates the application on FPGA and stores its latest correct state in history FFs. When *Multiple Bit Upset* (MBU) happens in SRAM cells and are manifested in main FFs, HAFTA detects them and reconfigure the FPGA without loosing the last correct state of the implemented application. RoRA in [139] is a reliability aware placement and routing technique to protect *Triple Modular Redundancy* (TMR)-based designs on FPGAs against *Single Bit Upset* (SBU).

In addition, the authors in [138, 169, 65] proposed offline or online reconfiguration techniques to mitigate aging-induced delay degradation in FPGAs to avoid timing failure in the system. An aging-aware floorplanner and reconfiguration policy for FPGAs is proposed in [65]. This offline method reduces the joint impact of BTI and *Hot Carrier Injection* (HCI) aging mechanism on the performance of the implemented application. Furthermore, the proposed technique in [138] swaps two different configurations for same application in two different regions of the FPGA, which impose area and power overheads. Similarly, a stress-aware placement proposed in [169] for acceleration application on different regions of the FPGA. Additionally, effects of different LUT logics on the BTI-induced delay degradation in LUT's Mux-tree is inspected in[123]. [88] investigates the BTI impact on different structures for Mux-tree of an LUT. Also, [6] investigates the BTI impact on different switch box structures of routing resources in an FPGA. In sum, the studies on aging impact on FPGAs more concentrated on performance degradation, that happens due to aging in Mux-tree (Fig. 5.4) or routing switch boxes. However, as discussed in Section 3, BTI can impact the SRAM cells inside the LUT and reduce their stabilities and SNMs.

Furthermore, the impact of BTI on SRAM cells in memory block has been studied in [135, 112, 126]. For example, in [135], a periodic flipping technique is proposed for mitigating aging in SRAM-based memories. [112] proposed an aging aware register file allocator for GPGPU to improve SNM. An adaptive cache size technique is proposed in [126] to mitigate the SNM reduction of their SRAM cells. In addition, some related works studied the impact of aging on an SRAM cell, such as read and write stability reduction, and increase in leakage power [95, 157, 34, 91, 54, 16, 151]. These works show that BTI can significantly reduce the SNM of an SRAM which results in lower SRAM stability, reliability and performance.

Frequent recovery phase (removing stress) in transistors leads to their lower age in long term [76]. Therefore, the impact of BTI on SRAMs in FPGAs is more severe than memory because the logic inside SRAM of an FPGA does not change until next reconfiguration (i.e.

S=1). Whilst, the logic inside SRAM cells in block memories changes more frequently in every several cycles (i.e. $S < 1$). This may corrupt the SRAM value and\or raise the SER in FPGAs by a factor of two [84]. For example, in [84], the impact of voltage scaling and aging mechanisms on SER of SRAM configuration bits in FPGAs is studied. This study shows that SER in FPGAs increases by a factor of two. Additionally, [87] proposed a stress aware routing to preserve SRAMs' switch boxes in FPGAs against aging. As occlusion, aging in SRAM cells not only jeopardizes the usage of FPGAs in safety critical applications, but also in mainstream applications. Hence, we proposed a stress-aware Boolean matching technique for LUTs in order to increase FPGA's lifetime and decrease SER.

5.4 SAT-based Boolean matching (BM) overview

Boolean matching is a well-known technique that has been deployed in logic re-synthesis and technology mapping for FPGAs [42, 127]. This technique is utilized to investigate the equivalence of two functions $f(X)$ and $g(Y)$ using input permutation and logic or wiring reconfiguration. For solving this problem, we need to search a possible mapping ψ in such a way that $f(\psi(X)) = g(Y)$. However, searching all possible mappings due to the huge time complexity $O(n! \times 2^{n+1})$ is not practical, where n is the number of input variables. Therefore, search space pruning techniques are required to reduce the time complexity [81].

For instance, consider a *cone* ζ as a reconfigurable template with inputs $x_1, ..., x_k$, output G, LUT configurations $c_1, ..., c_k$, and their internal wires $w_1, ..., w_m$ as shown in Fig. 5.6. Additionally, let F be a Boolean function of k inputs, given as a truth table. Let $\Psi(\zeta)$ be a set of constraints that define the cone ζ characteristics, e.g. internal and output wires, number of LUTs, end etc. Hence, the Boolean matching problem for (ζ, F) can be formalized to a Boolean formula question that asks is there any setting for LUT configuration $c_1, ..., c_k$

such that for all inputs $x_1, ..., x_k$, the output G of ζ is equivalent to F:

$$\begin{aligned} &\exists c_1 ... c_n \forall x_1 ... x_k \exists w_1 ... w_m \\ &\omega(\overrightarrow{x}, \overrightarrow{x'}) \wedge \Psi(\zeta) \wedge \Psi(F) \wedge (G \leftrightarrow F) \end{aligned} \tag{5.6}$$

where $\omega(\overrightarrow{x}, \overrightarrow{x'})$ expresses a mapping function for input permutation $\overrightarrow{x}$ of the Boolean function to the input pins $\overrightarrow{x'}$ of the cone. It needs to be noted that the mapping functionω can be altered. For example, instead of input permutation we can change the LUT configurations $c_1, ..., c_k$, change the internal wiring $w_1, ..., w_m$, or combination of them to G satisfy (SAT) F. In sum, SAT-based Boolean matching is an effective solution to map a function to a given reconfigurable template such as cones.

5.5 Problem formulation

In this work, our goal is to generate a new configuration for already placed and routed design on FPGA in such a way that all the LUT configuration bits values are flipped but functionality is preserved. The main configuration and the new configuration are exchanged periodically (i.e. each P time) through FPGA reconfiguration to mitigate BTI-induced SNM reduction of SRAM cells in FPGAs' LUTs. Therefore, the data flow graph, DFG, of the placed and routed design is extracted. Each node in DFG is an LUT. After that, the cone construction step partitions DFG to different cones with specific functionalities. Through SAT-based Boolean matching a flipped configuration for each cone is found in such a way the cone's functionality is preserved. The problem is stated formally as:

- Given the placed and routed design, and time interval of $[t_0, t_n]$,
- Extract the Data Flow Graph (DFG) of the design, partition it to cones, and then generate a new configuration such that all LUTs' bits are flipped, while the functionality

of each cone is preserved. The main configuration will be replaced by the flipped configuration at each period of time, P, from t_0 to t_n.

For instance, consider the DFG Γ is partitioned to set of cones $\zeta_1, ..., \zeta_m$ with the Boolean function of $F_1, ...F_m$ and configuration of $\pi_1, ..., \pi_m$. Each cone's configuration π_r is a set of LUT configurations $c_1^r, ..., c_k^r$, as well. Additionally, as elaborated in previous section, let $\Psi_1(\zeta_1), ..., \Psi_m(\zeta_m)$ be a set of constraints sets that define the cones' characteristics, such as their number of inputs, its number of LUTs, the internal connections, and its output G. Then:

$$\begin{aligned} &\forall \zeta_1 ... \zeta_m \exists F_1 ... F_m \\ &\varphi(\pi, \pi') \wedge \Psi(\zeta) \wedge \Psi(F) \wedge (G \leftrightarrow F) \end{aligned} \tag{5.7}$$

where $\varphi(\pi, \pi')$ expresses a Boolean matching procedure on each cone ζ that flips all LUT configurations $c_1, ..., c_k$ to $\overline{c_1}, ..., \overline{c_k}$ considering that its output G is SAT (i.e. equivalent to F). In the next section we elaborate our proposed stress-aware Boolean matching (STABLE) technique.

5.6 STABLE, stress-aware Boolean matching

The key idea behind the STABLE is to utilize SAT-based Boolean matching technique at cone level or LUT level to flip LUT contents while functionality is preserved, without changing the placement and routing of the optimized application on FPGA. This method is applied after placement and routing of the desired application on FPGA using commercial tools (e.g. Xilinx ISE©) in order to induce minimal overheads in term of rerouting, placement, area, performance, and power. STABLE consists of three steps, which are elaborated in the following subsections. After these steps, the generated flipped configuration (i.e. bitstream)

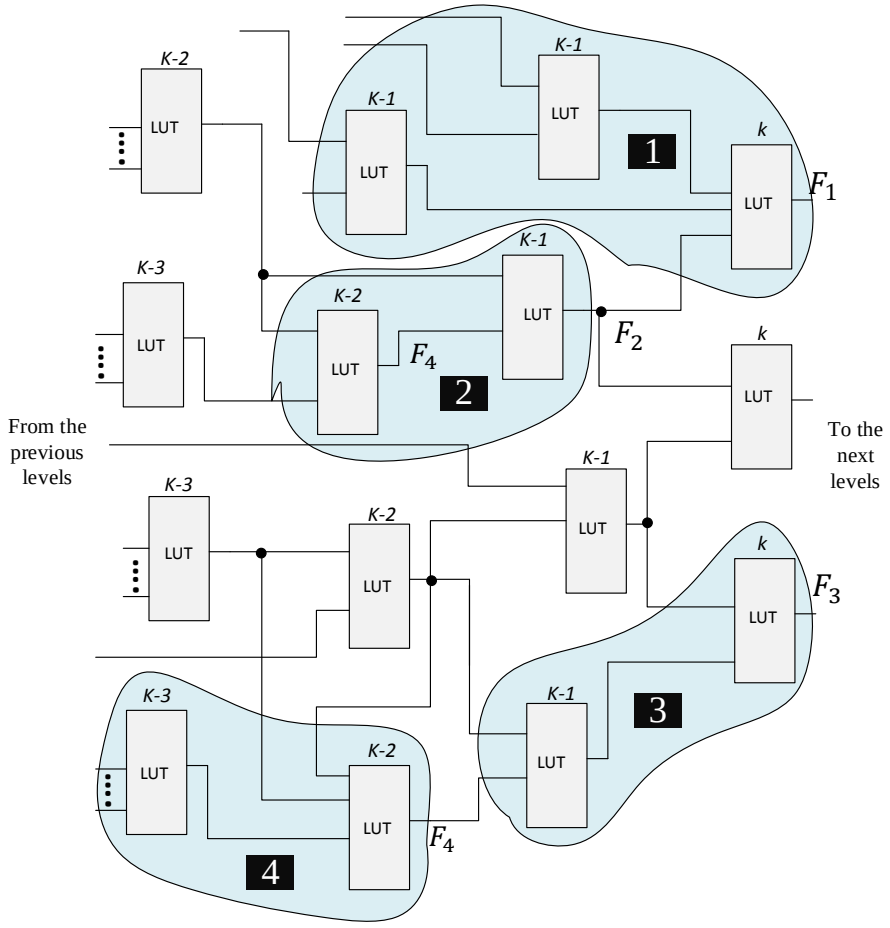


Figure 5.5: Partitioning the Data Flow Graph (DFG) to cones.

is stored to be used in periodical reconfiguration of the system.

5.6.1 Cone construction

We partition the implemented design to a set of cones. For example, the DFG in Fig. 5.6.1 is portioned to four different cones $\{\zeta_1, \zeta_2, \zeta_3, \zeta_4\}$. Each cone consists of a set of LUTs with known functionality, $\{F_1, F_2, F_3, F_4\}$. A cone has only one output and consists of a root LUT and its fan-in LUTs. By definition, a cone must have at least two levels of LUTs (its root and its fan-in LUTs) and represents one and only one function. Note the number of levels can be n. A cone with m inputs is an m-cone. For instance, in Fig. 5.6.1 the cone 1, which

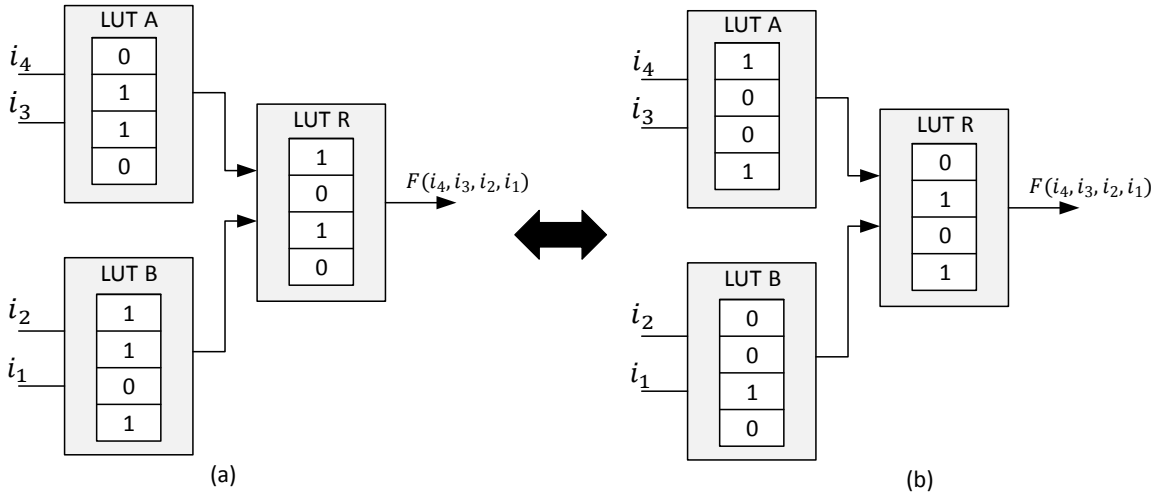


Figure 5.6: Cone flip motivation; one cone with two complemented configuration bits, (a) and (b), and same function..

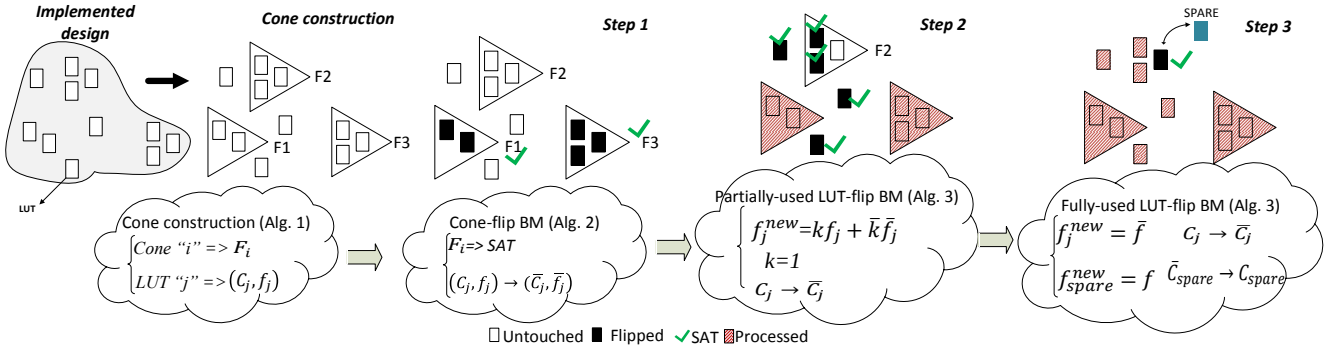


Figure 5.7: The STABLE, stress-aware Boolean Matching.

includes three LUTs (one root at level K and two LUTs at level $K-1$) has 5 inputs, thus it is a 5-cone that represents function F_1. As shown, all LUTs inside a DFG cannot be part of a cone due to the cone characteristics and limitations. For example, a should have cone only one output and avoids LUTs with more than one fanout to be included into a cone, unless it is a root. Additionally, if the cone size (number of LUTs inside it and number of inputs to it) is high the Boolean matching search space will be huge (As elaborated in Section 5). Hence, we have to limit the size of cone to at most three or four LUTs. An LUT only belongs to one cone.

The motivation behind partitioning a DFG to cones is illustrated in Fig. 5.6. As can be seen, the cone consists of three LUTs (one root LUT and two fan-in LUTs). The cone has 4 inputs which means it is a 4-cone that represents function $F(i_4, i_3, i_2, i_1)$. The LUT configuration bits in (a) and (b) are full complement of each other while both configuration represent the same function for a cone. Therefore, using in-place Boolean matching (i.e. without interfering with the placement and routing), we find alternative flipped configuration bits for each cone and preserve the cone's function. Briefly, the first step in STABLE looks for an opportunity of finding the maximum hamming distance (i.e. flip all configuration bits) without changing the placement and routing of LUTs inside the cone. LUTs that are not included in a cone or their cone cannot be flipped (due to not maintaining the coneâĂŹs function) will be considered in the second or third steps in the STABLE.

Fig. 5.7 illustrates the STABLE. It is shown that each cone ζ_i represents function F_i and each LUT L_j is represented by a tuple (c_j, f_j), where c_j is its configuration bits and f_j is its function. Algorithm 4 shows the pseudo-code for cone construction phase. The algorithm's input is the DFG (i.e. list of LUTs (or nodes) in the implemented design and their connections (or edges)) and its output is the list of cones. The algorithm looks for those LUTs that only have one fan-out using $RetFanOut()$ function (line 4). This function returns number of fan-outs of an LUT. Then, using $FindOutput()$, the connected LUTs to the target LUT's output will be set as the root of its cone (Line 5). The root LUT can have more than one fan-outs, obviously. Basically, at this step of the algorithm we tag LUTs with more than one fan-out as roots. Next, the target LUT, and its root, will be added to the cone list (line 6). This process continues until all the LUTs are investigated. As previously discussed, all LUTs cannot be included in a cone. However, the LUT inside a cone has a higher chance of flipping without any impact on the implemented design. Section 5.8 elaborates more on cone construction step.

Algorithm 4 Cone Construction

Input: DFG as an LUT list $\{LUT\}$
Output: List of Cones $\{Conelist\}$

```
1: Conelist = {};
2: for all LUT_i ∈ {LUT} do
3:    if LUT_i ∉ {Conelist} then
4:       if RetFanOut(LUT_i) = âĂIJ1âĂİ then
5:          Root_j ← FindOutput(LUT_i);
6:          Conelist.Add(Root_j, LUT_i);
7:       end if
8:    end if
9: end for
10: Return {Conelist}
```

5.6.2 Cone-flip Boolean matching (Step one)

As shown in Fig. 5.7, in the first step of STABLE, we want to flip a cone configuration bits entirely based on the motivation in Fig. 5.6 using Boolean matching technique. Algorithm 5 elaborates this step. The algorithm's input is the generated list of cones $\{Conelist\}$ from construction phase and its output is the list of flipped cones $\{FlippedConelist\}$. Hence, using $GetFunc()$, the function of each cone is found for each cone (line 2). Then, all LUTs inside the cone are flipped by $FlipConf()$ (line 3-5). After that, if the new flipped configuration bits of the cone satisfy (SAT) the original Boolean function of the cone (line 6), then the cone is added to the flipped cone list and removed from the original list of cones (line 8-9). Otherwise, configuration bits are reset to the original setting of the cone (line 11-13).

The core of Algorithm 5 is expressed in lines 6-9, where determines that flipped configuration's function is SAT or not (Section 6). Since we want the new configuration bits be the complement of main configuration bits thus it is not required to explore the whole (or part of) the search space. This is not usually a case in SAT-solvers. Therefore, here, the time complexity of SAT-based Boolean matching is decreased considerably and we only look for a known configuration bits, the flipped one. Through that, a flipped configuration for each LUT inside the cones with 100% hamming distance are obtained. This step impose no over-

Algorithm 5 Cone-flip Boolean Matching (Step 1)

Input: Cone list $\{Conelist\}$
Output: List of flipped Cones $\{FlippedConelist\}$

```
1: for all Cone_i ∈ {Conelist} do
2:    F ← GetFunc(Conei);
3:    for all LUT_j ∈ {Cone_i} do
4:       FlipConf(LUTj);
5:    end for
6:    F_new ← GetFunc(Cone_i);
7:    if F_new = F then
8:       FlippedConelist.Add(Cone_i);
9:       Conelist.Remove(Cone_i);
10:   else
11:      for all LUT_j ∈ {Cone_i} do
12:         FlipConf(LUT_j);
13:      end for
14:   end if
15: end for
16: Return {FlippedConelist}
```

head to the system, while all SRAM cells in cones' LUTs are flipped and cones functionality is preserved. As shown in Fig. 5.7, the flipped LUTs inside the cone are colored as black and the cone's that their functionalities is SAT are determined. For next steps these cones and their LUTs are classified as processed (red lines hachure in Fig. 5.7).

5.6.3 Partially-used LUT-flip Boolean matching (Step two)

Based on Fig. 5.7, in this step, we investigate partially-used LUTs, that do not belong to any cone (Fig. 5.6.1) or were inside a cone but have not been covered during step 1. Partially-used LUTs have one or more unused inputs, that are connected to either V_{dd} or ground. For example, in a 6-input LUT (contains 64 configuration bits), if one of the inputs is not used then there are 32 unused SRAM cells, inside the LUT, that can be utilized for reliability purposes (i.e. as redundancy). This technique is a low-cost, effective, and efficient usage of wasted SRAM cells inside partially-used LUTs. Due to optimization purposes or application

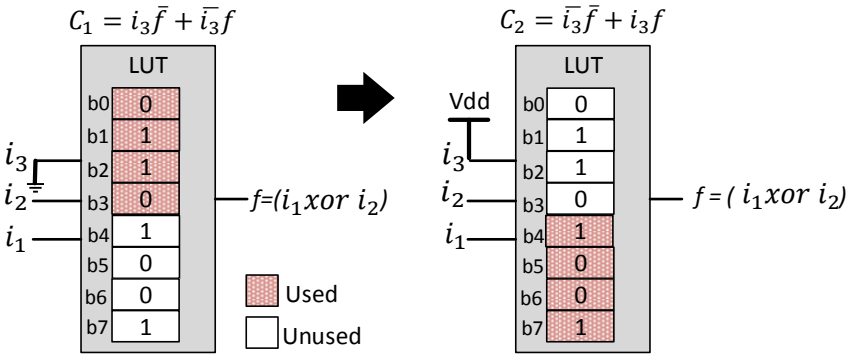


Figure 5.8: C1 and C2 represent the same function using different configuration bits inside on LUT.

characteristics, academic and/or commercial synthesis tools left behind many LUTs that are partially used, that some of their SRAM cells are wasted.

Assuming the original LUT's configuration bits' functionality is f and input k is one of the unused inputs, we change the configuration from $kf + \bar{k}\bar{f}$ to $k\bar{f} + \bar{k}f$ and k is set to V_{dd}. For example, as illustrated in Fig. 5.8, we connect the unused inputs (i_3) to a flip state (V_{dd}) to access unused cells and configure them with the complement (C_2) of the original configuration bits (C_1). Both configurations represent the same function (i.e. $i_1 xor i_2$) while different parts of the LUT are used and configuration bits are flipped entirely. In comparison to step 1, we have to rewire (connect) the free input (here i_3) from ground to V_{dd}. The first part in algorithm 5.8 (line 2-16) elaborates on this step. The inputs are the remaining list of cones $\{Conelist\}$ from the previous step and the list of LUTs $\{LUTlist\}$ that are not part of any cone.

Our algorithm iterates through each LUT, inside the remaining cones, and checks if they have unused (free) inputs using the $InputIsFree()$ function (line 4). If the LUT has unused cells, the configuration bits will be flipped and the main configuration will be stored in the unused cells using $FlipConfUnused()$ based on Fig. 5.8. LUTs that pass this step will be added to the list of flipped LUTs and will be removed from the cone (line 6-7). Same

Algorithm 6 LUT-flip Boolean Matching (step two and step three)

Input: List of Cones $\{Conelist\}$, List of LUTs that not in any cone $\{LUTlist\}$
Output: List of flipped LUTs $\{FlippedLUTlist\}$

```
1: FlippedLUTlist = {};
 //Flip partially-used LUTs (step 2)
2: for all Cone_i ∈ {Conelist} do
3:    for all LUT_j ∈ {Cone_i} do
4:       if InputisFree(LUT_j) then
5:          FlipConfUnused(LUT_j);
6:          FlippedLUTlist.Add(LUT_j);
7:          Conelist.RemoveLUT(Cone_i, LUT_j);
8:       end if
9:    end for
10: end for
11: for all LUT_j ∈ {LUTlist} do
12:    if InputIsFree(LUT_j) then
13:       FlipConfUnsed(LUT_j);
14:       FlippedLUTlist.Add(LUT_j);
15:    end if
16: end for
 //Flip fully-used LUTs (step 3)
17: for all Cone_i ∈ {Conelist} do
18:    for all LUT_j ∈ {Cone_i} do
19:       SpareLUT ← SearchUnsedLUT();
20:       StoreLUT(LUT_j, SpareLUT);
21:       FlipConf(LUT_j);
22:       FlippedLUTlist.Add(LUT_j);
23:    end for
24: end for
25: for all LUT_j ∈ LUTlist do
26:    SpareLUT ← SearchUnsedLUT();
27:    StoreLUT(LUT_j, SpareLUT);
28:    FlipConf(LUT_j);
29:    FlippedLUTlist.Add(LUT_j);
30: end for
31: Return {FlippedConelist}
```

investigation and operation are also done to the list of LUTs which do not belong to any cone (line 10-14). In all, at this step by using the unused SRAM cells inside the LUT we flip the LUT configuration while the Boolean functionality of each cone is satisfied. Furthermore, it can be concluded that till this step the LUTs are flipped with negligible overhead on

the design without affecting the optimized placement and routing. The remained LUTs are fully-used LUTs that were not included to any cone at the cone construction phase. Hence, in step 3 we cover remained fully-used LUTs. As shown in Fig. 5.7, at the end of step two partially-used LUTs are flipped (colored as black). Whilst, their functionalities are SAT by swapping configuration bits inside the LUTs with their complement.

5.6.4 Fully-used LUT-flip Boolean matching (Step three)

To flip fully-used LUTs inside the cone, which are remained from step 1, we determine the fully-unused LUTs inside each neighboring logic block (i.e. slice in Xilinx) as a spare for them (Fig. 5.7, step 3). Each LUT belongs to a logic block. Logic block is a reconfigurable component in FPGAs that is a set of LUTs, FFs, Carry chain, and etc. Usually, each two logic blocks belong to a *Configurable Logic Block* (CLB) that is connected to other CLBs through switch boxes. FPGA is a 2D matrix of CLBs. Therefore, we can find the physical location of each LUT and its neighboring LUTs.

The second part in algorithm 5.8 (line 17-30) elaborates this step. Using a greedy *Breadth First Search* (BFS) algorithm we find the closest fully-unused LUT and configure it as the spare LUT by the flipped configuration of main LUT (line 19). Then, using $StoreLUT()$, we store original configuration bits to the spare LUT (line 20) and the original configuration will be flipped (line 21). Similarly, for fully-used LUTs that do not belong to any cone the same process is done (line 25-30). In all, at this step we flip the fully-used LUTs using fully-unused neighboring spare LUTs, that are left behind after placement and routing of the design inside the logic blocks, with negligible rewiring overhead.

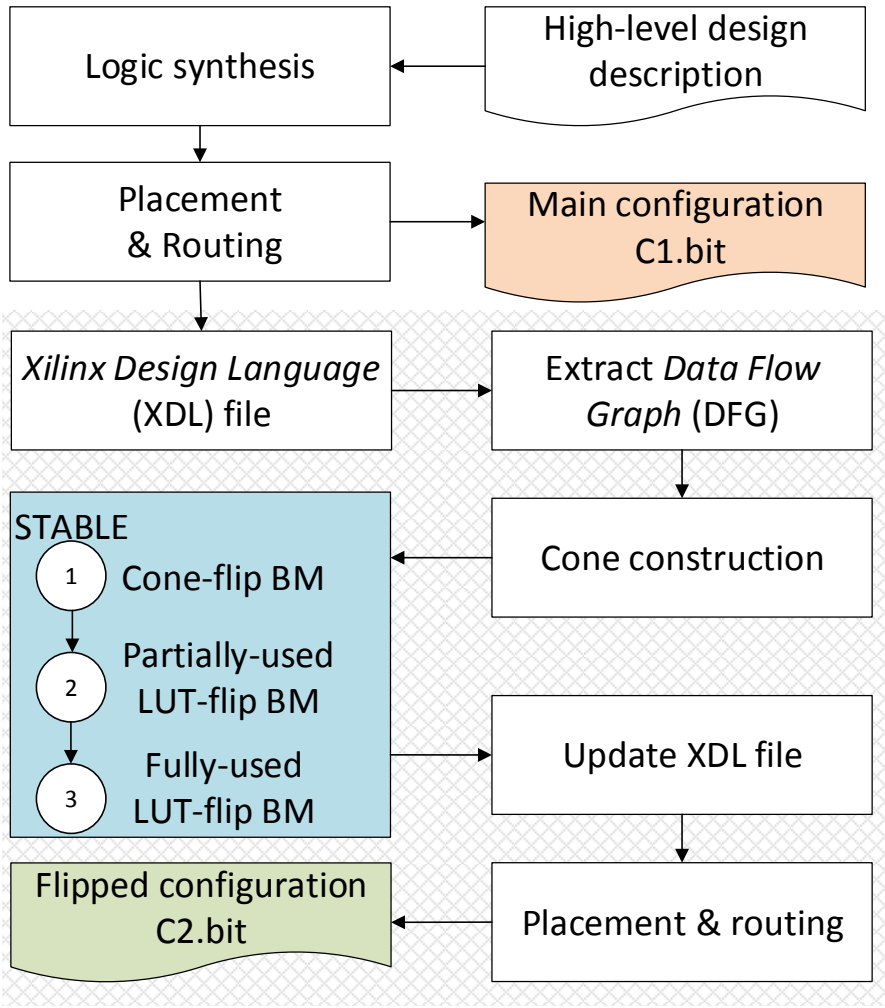


Figure 5.9: STABLE implementation flow.

5.7 Experimental evaluation

In this section we review our experimental setup for verification of STABLE and the extracted result for different sets of selected benchmarks. At the end we discuss and analyze STABLE.

5.7.1 Setup

Implementation flow of the proposed method, STABLE, is abstracted in Fig. 5.9. At first, the benchmark's description (e.g. as HDL format) is synthesized using the Xilinx synthesis tool, ISE 14.7©. Our target FPGA device is Virtex6 (xc6vlx240t) which is fabricated in 40nm technology. At next step, the placement and routing is done. After that, we can generate the *.bit* format of the main configuration (C1) that can be loaded into the FPGA. In addition, the synthesizer generates post-route information in *Xilinx Design Language* (XDL) format. From XDL we can extract the DFG of the implemented design and find the required information such as each configuration of LUTs, functionalities, their connections, and unused resources. Then, the cone construction phase is done (Section 5.6.1).

Next, our implemented tool (in Python) for the STABLE methodology (Section 5.6) applies the three Boolean Matching (BM) steps. The XDL file is required to be updated based on the new DFG. For example, LUTs configuration bits are flipped and the new connection and LUTs are added for spare LUTs in step 3. Then we run the placement and routing tool for these new added resources. Finally, the flipped configuration bits is generated ($C2.bit$). C2 has the exact functionality as C1, while its configuration bits are flipped to put their SRAM cells in the recovery phase. Through periodical FPGA reconfiguration, (e.g. using ICAP in Xilinx board) we can swap between C1 and C2 to mitigate BTI-induced SNM reduction in SRAM cells (i.e. configuration bits).

To evaluate the impact of STABLE, 14 benchmarks are selected from different suites of applications (including ISCAS89, ITC99, multimedia, security, and etc.). Each benchmark's floorplan along with power information (i.e. after running test benches on them), from *Xilinx XpowerAnalyzer*, are fed into HotSpot[83] to extract the temperature map. Based on Eq. 5.3 and Eq. 5.4 for BTI, temperature analysis and stress (S) is required to extract the delay degradation to consequently compute SNM using proposed method in[133]. In SNM

analysis, we considered the worst case for temperature. Therefore, the extracted SNMs for each benchmark is based on the maximum temperature of the application. This is because for reliability analysis and *Mean Time to failure* (MTTF) we consider the worst case. Eq. 5.5 is used to extract the SER based on the proposed method in [150]. Another way of exacting SNM for an SRAM cell is proposed here [38].

Table 5.1: Cone construction result for each benchmark

Bench.	# LUT	# Cone	# LUT in Cone	Coverage (%)
AES	3729	652	2125	56.99
DCT	9862	800	1887	19.13
JPEG	20141	1545	4225	20.98
AVA	4609	512	1247	27.06
FPU	4774	535	1371	28.72
S641	60	6	14	23.33
S5378	325	42	104	32.00
S38417	2454	237	727	29.63
S15850	723	88	223	30.84
b10	40	7	16	40.00
b14	1013	200	532	52.52
b18	16325	2530	6137	37.59
b20	2336	340	881	37.71
b22	3393	463	1241	36.58
Total	69784	7957	20730	**29.70**

5.7.2 Results

As discussed earlier we partition DFG of the implemented design to cones. Table 5.1 shows the number of LUTs in each benchmark and the number of cones that are constructed using Algorithm 1, in the second and third columns, respectively. The last two columns of Table 5.1 report the coverage of cones in number of LUTs and percentage, respectively.

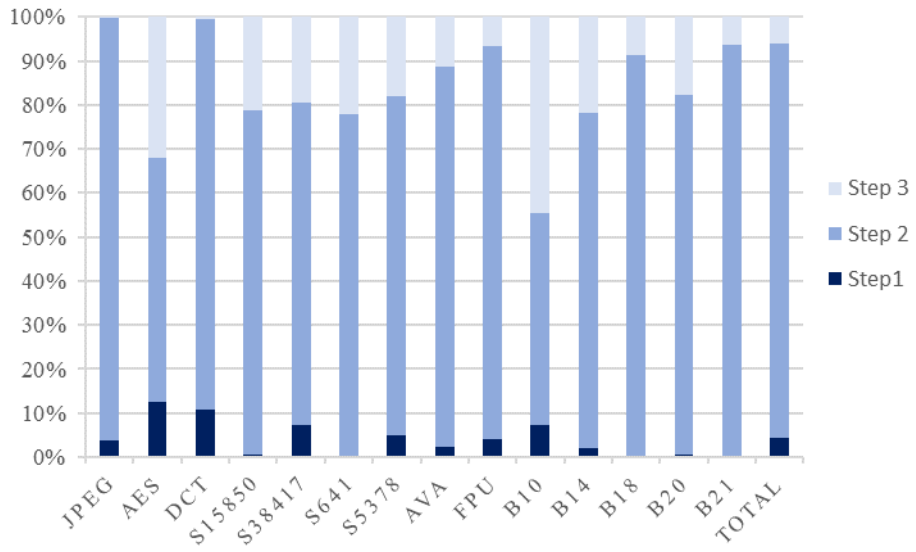


Figure 5.10: LUT Coverage in each step of STABLE.

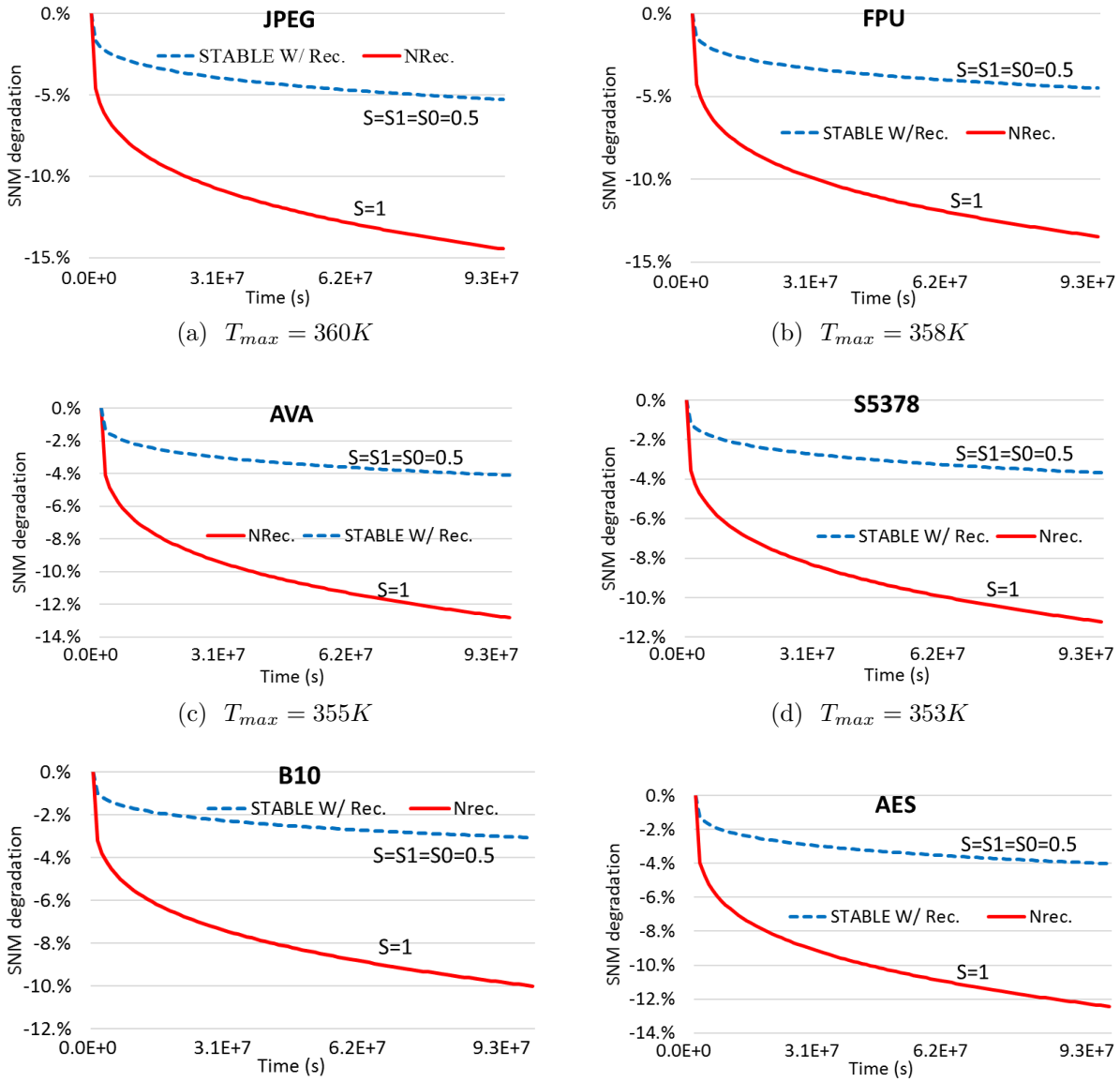
Our proposed algorithm for cone construction covers from $\sim$ 19% up to $\sim$ 57% of LUTs in different benchmarks. The average coverage in all 14 benchmarks is 29.70%.

In Fig. 5.10, the LUT coverage of STABLE in different steps for different benchmarks is illustrated. The selected benchmarks due to diverse characteristics have different amounts of LUT coverages at each step. For the selected benchmarks, $\sim$ 93% of LUTs are covered till the end of steps one ($\sim$ 4.2% is covered) and step two ($\sim$ 89% is covered). The remained fully-used LUTs ($\sim$ 7%) are covered at step three. For instance, in DCT, 1026 LUTs out of 9862 LUTs ($\sim$ 10%) are covered at step one. ($\sim$ 86%) of LUTs are covered at step two. The remaining ($\sim$ 4%) are covered during step three. It can be concluded that most of the LUTs are partially-used LUTs and will be covered during the second step. The second step in STABLE is an effective and straight forward technique which imposes negligible overhead to the system.

Fig. 5.11 demonstrates the SNM rate reduction of SRAM configuration bits for different benchmarks for 3 years of execution (i.e. 9.6E+7 seconds) in their worst case situation (maximum temperature). The dotted blue line (STABLE W/ Rec.) is when the application

Table 5.2: SNM (mV) reduction and SER (FIT/Mbit) raise comparisons in 3-year of execution (At time zero, SNM= 250mV and SER=1000FIT/Mbit)

	Bench.	AES	DCT	JPEG	AVA	FPU	S641	S5358	S38417	S15850	B10	B14	B18	B20	B22	AVG.
ΔSNM	STABLE	10	11	14	10	11	8	9	14	10	8	9	11	11	14	10.71
	NRec.	31	33	36	32	34	24	28	36	33	25	28	33	34	36	31.64
	Imp.(%)	67.74	66.67	61.11	68.75	67.65	66.67	67.86	61.11	69.70	68.00	67.86	66.67	67.65	61.11	**66.32**
ΔSER	STABLE	11	12	16	12	13	9	11	16	12	9	11	12	13	16	12.36
	NRec.	37	39	43	37	40	29	33	43	39	30	33	39	40	43	37.50
	Imp.(%)	70.27	69.23	62.79	67.57	67.50	68.97	66.67	62.79	69.23	70.00	66.67	69.23	67.50	62.79	**67.23**



(a) $T_{max} = 360K$ (b) $T_{max} = 358K$

(c) $T_{max} = 355K$ (d) $T_{max} = 353K$

(e) $T_{max} = 350K$ (f) $T_{max} = 356K$

Figure 5.11: SNM degradation comparison in 3 years (9.6E+7 seconds).

main configuration (C1) will be reconfigured by the generated flipped configuration (C2) by STABLE. Red line is when there is no reconfiguration technique (NRec.). For example, in JPEG benchmark using our proposed technique with periodical reconfiguration the SNM reduction is $\sim 5.6\%$ as opposed to $\sim 14.4\%$ in non-reconfiguration (NRec.) at $T_{max} = 360K$. As mentioned in Subsection 5.6.1, the temperature map of each benchmark is extracted using HotSpot tool. SNM reduction is improved by $\sim 63\%$, $\sim 46.27\%$, and $\sim 69\%$ in JPEG, FPU,

and B10, respectively. In Table 5.2, the amount of SNM is reported for different benchmarks after 3 years of simulated execution. As shown, SNM for SRAM in the 40 nm technology (in Xilinx Virtex 6) is calculated to be $\sim 250mV$ at time zero (i.e. before aging happens) [150]. SNM reduction is improved by $\sim 66\%$, on average after employing STABLE along with the periodical FPGA reconfiguration to swap between two complement configurations.

Furthermore, utilizing Eq. 5.5, we extracted *Soft Error Rate* (SER) of SRAM which is 1000 FIT/Mbit (0.001FIT/bit) at time zero for 40nm technology. After three years of simulated execution, after using STABLE, SER degradation, as compared to time zero, is improved by $\sim 67\%$ on average for different benchmarks. Table 5.2 summarizes this result, as well. For example, in B22, SER increases 16FIT/Mbit and 43FIT/Mbit in STABLE W/ Rec. and NRec., respectively. Higher FIT rate results in lower reliability and lower MTBF.

5.7.3 Analysis and discussion

The first two steps, cone-flip Boolean matching and partially-used LUT-flip Boolean matching, impose negligible overhead to the design. This is because they either flip LUTs inside a cone or use unused SRAM cells of partially-used LUTs. In step three, we find the nearest fully-unused LUTs as a spare for the remained fully-used LUTs using a greedy BFS search algorithm to reduce overhead. Let's assume that only step three is utilized to find a spare for each LUT to store their flipped configurations, then area overhead will be 2X. Because each LUT requires a spare. This imposes power overhead to the system since we need two times of main benchmark's assigned region on FPGA. This even imposes performance overhead to the system, since we need to pair each LUT to a spare LUT that might change the critical path of the design.

Additionally, STABLE is deployed after placement and routing of the design on FPGA. Therefore, the area, power, and performance overheads are negligible as compared to opti-

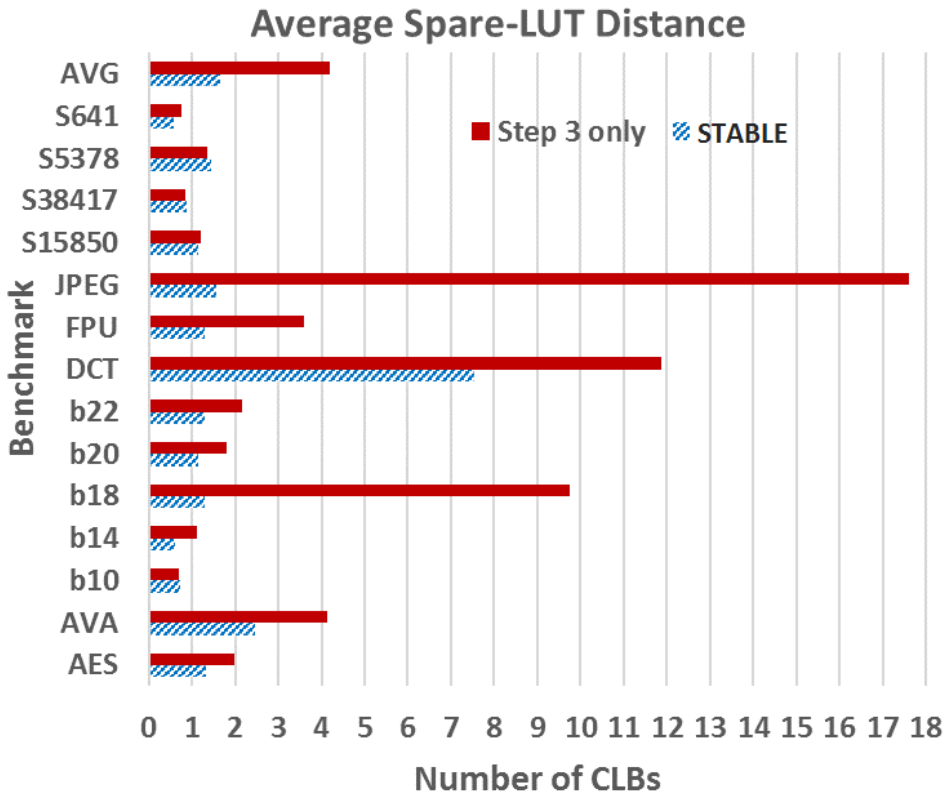


Figure 5.12: Comparison of average distance of spare LUT from main LUT between Step 3 and STABLE (Step 1, Step 2, and Step 3).

mized and obtained design by commercial tools. The STABLE overhead to the system may occur at step three for fully-used LUTs that could not be covered at step one (cone-flip). Therefore, spare fully-unused LUTs are found at the nearest distance using the greedy BFS on the DFG. Fig. 5.12 illustrates the average number of CLB in the way from main LUT to the identified spare LUTs, for each benchmark. This number on average is 1.66 *Configurable Logic Blocks* (CLBs) for different benchmarks in our proposed method. If we eliminate the first two steps and only use step three the distance overhead will be 4.2 CLBs. Moreover, in DCT benchmark step 3 only is required for less than 1% of the LUTs (Fig. 5.10), while average spare-LUT distance is $\sim$ 7.5 CLBs (Fig. 5.12). On the other hand, these numbers are $\sim$ 45% and $\sim$ 1, respectively. Although, the *percentage* of required LUTs in B10 is higher than DCT, but it seems our greedy BFS algorithm finds spare LUTs inside same slice as the main LUTs for step 3. This is not the case in DCT benchmark that the unused spared LUTs

are not available in nearer CLBs.

Table 5.3: The required time for STABLE steps in seconds

Bench.	Cone construction	Step 1	Step 2	Step 3
AES	23.75	100.20	0.02	0.02
DCT	0.73	9.69	0.06	0.02
JPEG	4.56	18.06	0.14	0.04
AVA	0.53	65.86	0.03	0.02
FPU	0.59	92.40	0.03	0.01
S641	0.01	1.05	0.02	0.02
S5378	0.07	8.06	0.03	0.01
S38417	0.35	40.85	0.01	0.01
S15850	10.32	17.67	0.01	0.01
b10	0.03	1.57	0.01	0.01
b14	0.05	31.70	0.01	0.02
b18	1.03	378.60	0.10	0.06
b20	0.14	78.67	0.01	0.03
b22	0.35	119.80	0.02	0.01
Amean	**3.04**	**68.87**	**0.03**	**0.02**
Gmean	**0.41**	**27.19**	**0.01**	**0.01**

Furthermore, Fig. 5.12 proves that our greedy BFS algorithm which looks for spare LUTs in the nearest possible distance always finds required spare LUTs in step 3. Although, implemented designs on FPGAs are optimized from placement and routing points of view, but usually there are unused resources inside the CLBs. Our experiment shows that the performance overhead is less than $\sim 1\%$ in different benchmarks on average.

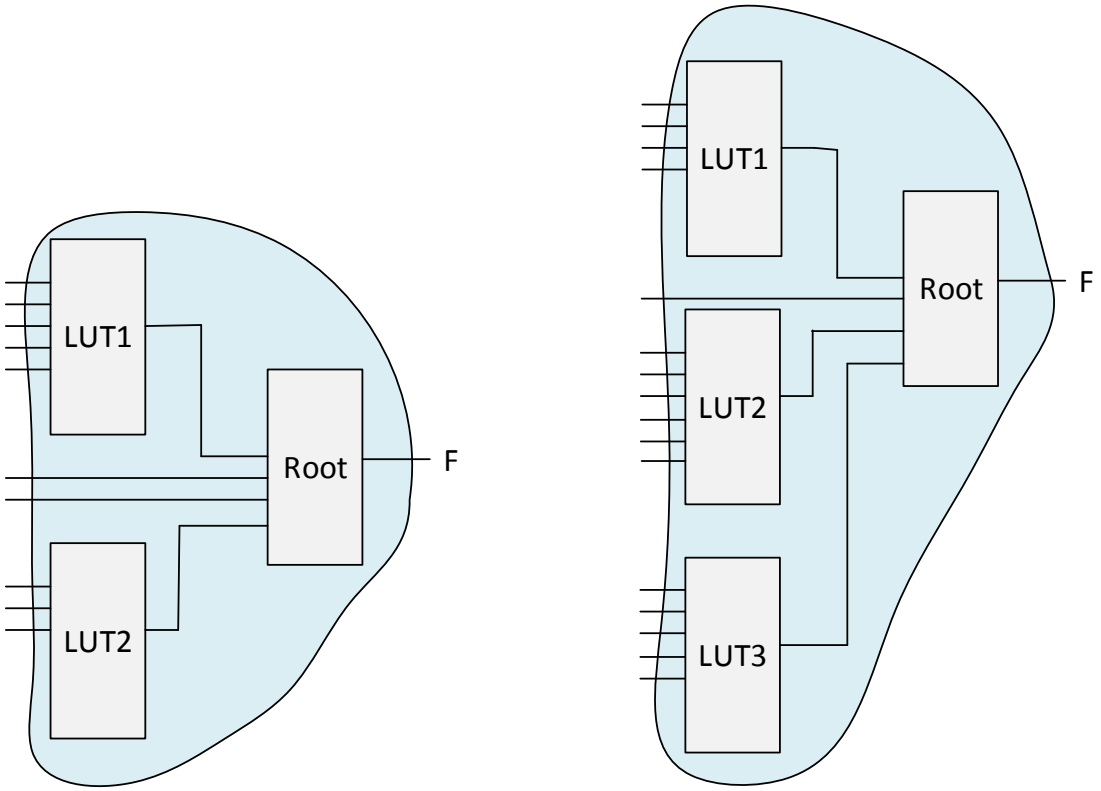
The long term BTI impact is independent of switching frequency in transistors [149, 9]. For example, as long as the stress value is optimum (S=0.5) then the switching frequency of an SRAM state will not impact the long term aging impact (e.g. SNM reduction). This means during the lifetime of T if SRAM's content flips at each T/2 or T/4 and so on the long term

BTI impact is fixed and is independent of different waveforms with same duty cycle (same S). hence, it can be concluded that only one time reconfiguration of design on the FPGA at time T/2 is enough to reach the optimal S. However, the designer can choose different period of the reconfiguration to swap between two configurations based on desired constraints. The only constraint from BTI point of view is to keep the stress optimal (S=0.5).

The required execution time of STABLE for each benchmark is shown at Table 5.3 in seconds. The average required time for cone construction phase, step 1, step 2, and step 3 are 3.04, 68.87, 0.03, 0.02, respectively. As it was expected step 1, cone-flip BM, has the lion's share of the running time. It needs to be noted that SAT-based Boolean matching is usually an expensive step in logic design, while in STABLE it runs in a reasonable speed. The main reason is that we do not need to investigate the whole search space and we only need to find the flipped configuration bits in a constant time, as explained in Algorithm 1. In all, STABLE runtime overhead is negligible in comparison to the whole process of implementing a design on FPGA from synthesis level all the way down to placement and routing. Therefore, STABLE can be easily embedded into the commercial tools.

5.8 Cone size impact on the run time of step 1

Each cone has an LUT as its root as well as a set of connected LUTs to it. Algorithm 1 elaborates on the cone construction phase. The cone size is defined as number of inputs to it plays an important role on the running time of step 1 in STABLE. Here we want to show how increase in the number of inputs to a cone can drastically increase the SAT-based Boolean matching process on it. Therefore, we need to put a constraint on the size of a cone to not exceed a certain size of it and avoid high running time overhead. For instance as shown in Fig. 5.13, the required time to run SAT-based BM on cone a with 10 inputs is around 0.2 seconds while in cone b is more than 20 seconds.



(a) 10-cone, a cone with 10 inputs

(b) 16-cone, a cone with 16 inputs

Figure 5.13: Cones with different size

The reason behind is that by increasing number of inputs the required time to extract the functionality of the flipped cone increases exponentially. Therefore, we limit the cone size to avoid such cases and if adding an LUT to a root LUT cause to exceed the number of inputs' threshold, which is 11 inputs in our experiment, we do not add that LUT to the cone. Table 5.4 summarizes the approximate required running time for different cone sizes in seconds.

Table 5.4: BM running time on cones with different number of inputs in seconds

#inputs	10	11	12	13	14	15	16
running time	0.2	0.5	1.2	2.3	4.7	11.8	20

5.9 Chapter summary

In this chapter, a post-synthesis three-step flipping LUTs' configuration bits' methodology named as STABLE is proposed. STABLE finds an entirely flipped configuration for the implemented design on FPGA. The new configuration is swapped by the main configuration to reduce the impact of aging on SRAM cell configuration bits in FPGAs for half of the lifetime. Flipping SRAM content periodically helps to mitigate BTI-induced SNM reduction by putting SRAM transistors in recovery phase. Our objective is to flip bits inside each LUT while preserving the original functionality of the implemented design using SAT-based BM. Therefore, we partition DFG of the implemented design to cones. Each cone is investigated for flipping their LUTs' configuration bits. At first, the possibility of flipping all the LUTs' inside the cones are explored, while maintaining the cone's functionality. Next, unused SRAM cells inside partially-used LUTs are utilized for flipping original configuration bits' of remaining LUTs from previous step. At the end, we can leverage nearest unused LUTs as flipped configuration spares for the main fully-used LUTs that did not pass the first two steps. We chose 14 different set of benchmarks to show the effectiveness of STABLE. Our results show 66% and 67% improvement in SNM reduction (ΔSNM) and SER increase (ΔSER), respectively, with negligible overhead on the implemented design.

Chapter 6

ADAMANT, aging-aware task mapping in heterogeneous multiprocessor architectures

Present day mobile devices (e.g., Smartphones, laptop, wearables, IoT platforms) continue to execute an increasingly wider range of applications and workloads. In this scenario, *single-ISA heterogeneous multiprocessor architectures* (HMPs) (e.g. ARM's big.LITTLE [73] with both simple and complex single-ISA core types in the same chip) are becoming pervasive due to their ability to simultaneously provide performance for computationally intensive applications (e.g. gaming, data mining) and power efficiency for IO-driven or interactive applications (e.g. web, email, body sensing).

Effectively exploiting power-performance tradeoffs requires smart task management mechanisms that are able to properly map workloads to the appropriate core type [14, 130, 99] (e.g. mapping a task to the most power efficient core type that satisfies its performance requirements). This necessity, however, has the side effect of overutilizing specific core types,

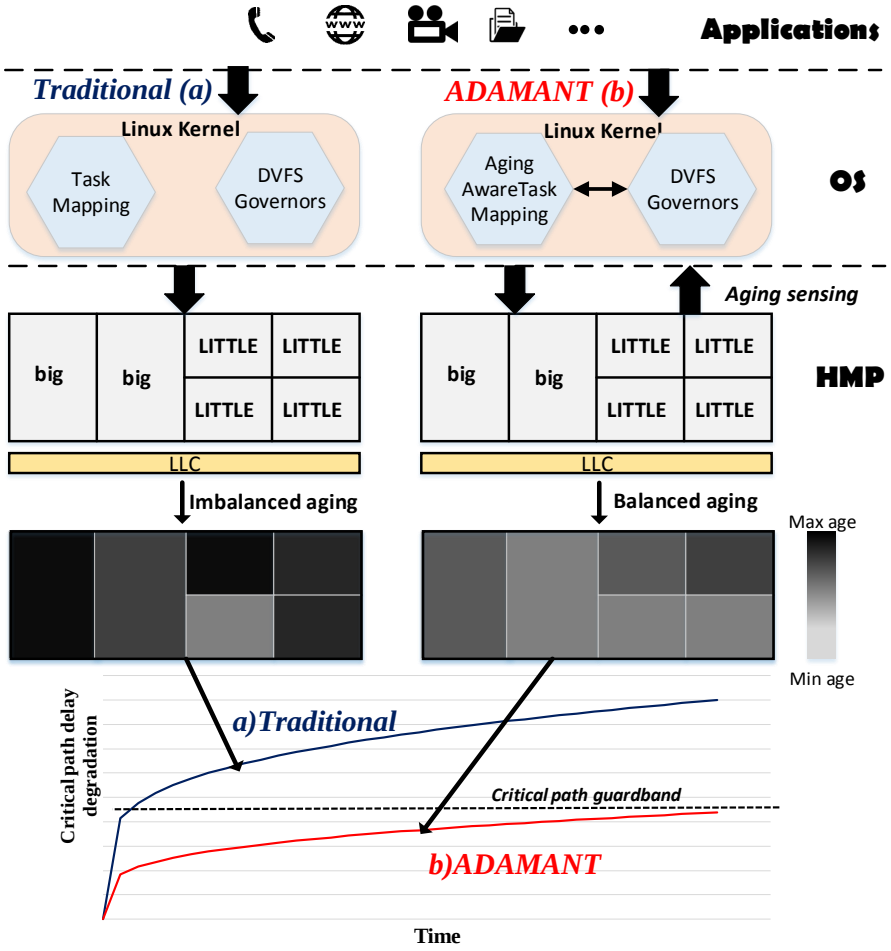


Figure 6.1: (a) Traditional task mapping. (b) Aging-aware task mapping with DVFS. Due to wearout caused by the aging imbalance in (a), the circuit critical path delay may violate its operating frequency guardband.

which may lead to decreased system reliability due to excessive stress on that core type. For instance, [147] shows that the current model for task load balancing in Linux leads to premature aging and wearout of overutilized cores in multicore platforms as shown in Fig. 6.1 (a). This happens due to device-level aging mechanisms such as *Bias Temperature Instability* (BTI) [22] and *Hot Carrier Injection* (HCI) [31]. These mechanisms are highly dependent on factors such as temperature and core utilization (i.e. stress), which causes transistor-level delay degradation and reduces core performance throughout its lifetime. Several works have attempted to mitigate these issues and proposed runtime aging-

aware *dynamic voltage/frequency scaling* (DVFS) and load balancing schemes to increase platform reliability [39, 147, 121, 106, 70]. However, these approaches are limited to homogeneous architectures and cannot be directly applied to HMPs since they do not consider the workload affinity towards specific core types.

This chapter closes this gap by applying aging-aware load balancing to effectively explore the power-efficiency of HMPs. We propose, ADAMANT, an Aging Driven tAskMAppiNg tool that uses aging, performance, and power sensing and prediction in order to map tasks to the most appropriate core type while balancing out aging towards increased lifetime as shown in Fig. 6.1 (b). In contrast to previous works, ADAMANT is designed only to replace the load balancing mechanisms in Linux-like runtime systems and works in tandem with other components such as the widely used *Completely Fair Scheduler* (CFS) and on-demand DVFS governor. In summary, this chapter makes the following contributions:

- ADAMANT leverages performance/power predictive models as well as core-level aging models to perform online characterization of the tasks' workload across all core types in the HMP.
- ADAMANT's run-time task mapping algorithm finds energy efficient mappings that simultaneously meets the task's performance requirements and reduce platform aging.
- Experimental results on typical mobile workloads executing on a big.LITTLE architecture demonstrate up to 2x improvement in lifetime with negligible overheads.

6.1 Related work and motivation

In multicore scenarios, task mapping/scheduling policies that are unaware of aging may lead to reduced lifetime due to the over-utilization of the same core. In current heterogeneous architectures such as ARM's big.LITTLE [74], task mapping policies attempt to

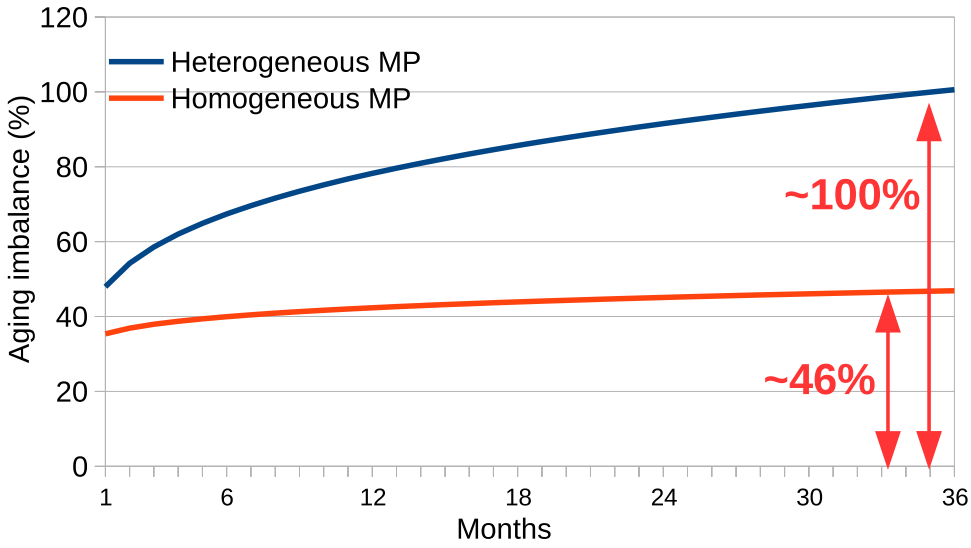


Figure 6.2: Relative difference between the most and the least aged cores in homogeneous and heterogeneous architectures.*typical* workload (Table 6.2). GTS scheduling [14]

improve energy efficiency by establishing an affinity between workloads and different core types [14, 130, 99, 163], which further intensifies the aging imbalance. An example of such policy is the Global Task Scheduling (GTS)[14] targeting ARM's big.LITTLE. GTS migrates tasks between big and little cores when the task load reaches a certain threshold. GTS-based policies has been implemented on Linux and deployed by multiple SoC vendors that support the big.LITTLE technology (e.g., Linaro's/ARM's big.LITTLE MP implementation used by Samsung Exynos[122], MediaTek's CorePilot[101], Qualcomm's Energy Aware Scheduling[110]). Fig. 6.2 illustrates this by showing the difference of delay degradation (aging imbalance) between the most and the least aged cores in an 8-core big.LITTLE heterogeneous architecture and a 4-core (big-only) homogeneous architecture (refer to Section 6.6.1, Table 6.1 for platform details). After three years, the difference in degradation between the most aged core and the least aged core is $\sim 46\%$ on the homogeneous platform, while on the heterogeneous platform this difference reaches 100% for the same workload, despite having a larger number of cores to map tasks to.

Previous works [121, 106, 70] are able to reduce aging imbalance by prioritizing the least aged cores during task mapping in homogeneous architectures. [121] proposed an adaptive runtime

task allocation to meet performance constraints while minimizing energy and maximizing system lifetime. They focused on reducing BTI aging impact for soft realtime multimedia applications and do not consider scenarios with DVFS in which the workload is not known. Furthermore, [121] does not consider the effect of HCI which is one of the dominant aging factors. [106] proposes a joint task mapping/DVFS algorithm for multicore mobile scenarios. They use a combination of aging sensors and aging models to define which cores can run foreground performance-demanding tasks (and the target frequency) Their goal is to balance out cores' aging induced delay degradation while maintaining performance. [70] proposes a similar strategy assuming a dark silicon scenario [77], i.e., not all cores can be active at the same time due to power/thermal constraints. They leverage this availability of unused cores to dynamically redistribute the workloads according to the chip thermal profile and estimated aging rates. [124] proposes a task mapping approach in which tasks are compiled to multiple versions tradeoffs performance and error resilience. The code version to use is selected at run time considering the cores' soft error rates and performance variations. [44] also proposes a compile-time approach. Multiple task maps are statically generated for tasks modeled as directed cyclic graphs (DAGs) or synchronous data flow (SDF) models considering different sets of available cores and their *mean time to failure*. At runtime, one of pre-generated mapping is chosen for each application depending on the total number of active applications. [128] improves over [44] by detecting intermittent faults at run time in order to make mapping decisions.

The aforementioned works perform aging mitigation through task mapping, however, they are not suitable for heterogeneous architectures since they do not consider the difference in power/performance tradeoffs that different core types can provide. Balancing out workloads without considering these tradeoffs may lead to impaired energy efficiency solutions [130]. [104] assumes that all cores provide the same performance (given the same frequency) which is unsuitable for HMPs. Furthermore, their approach assumes core-level DVFS domains, while current mobile platforms employ cluster-level DVFS [74]. [70] requires offline profiling

of the target applications, which limits its applicability to general purpose mobile devices (e.g. Smartphones); DVFS is also not supported, instead, the task mapping is chosen based on the cores' maximum supported frequency given the current aging state and temperature constraints. [44] and [128] also target scenarios in which the tasks are known and their formal DAG/SDF models are available. [124] has similar limitations and uses custom compiler support. ADAMANT, on the other hand supports generic workload without prior characterization and is orthogonal to DVFS (core-level and cluster-level). When compared to state-of-the-art works that address task mapping for heterogeneous architectures [130, 99, 163], ADAMANT is the first to consider online sensing of BTI and HCI aging mechanisms. Also, current solutions for performance and power prediction [108, 163] currently do not consider dynamic frequency variability due to DVFS. ADAMANT addresses this issue with predictive DVFS models for synergistically coupling task mapping with frequency scaling to mitigate aging effects.

6.2 ADAMANT framework overview

As shown in Figure 6.3, the ADAMANT framework for HMPs is composed of three parts: 1) **Sensing** exploits readily accessible *hardware performance counters* (HPCs), power sensors and aging sensors distributed throughout the chip to monitor the workload behavior and chip aging; 2) this information is aggregated at the end of each mapping epoch and is used as input for **performance/power prediction and estimation** models to characterize tasks' performance and power across different core types in the HMP; 3) a **task mapping** phase uses the predicted values to select the task mapping that best satisfies the aging-aware system goal.

An important aspect of ADAMANT is that its design considers the current scenario of HMP-powered mobile platforms that use Linux/Unix-like operating systems such as Android and

iOS. In such models, applications can enter and leave the system at any time and their total execution time is unknown. Therefore, as opposed to previous works that assume a fixed set of known applications [121, 70], we consider the case of dynamic task-to-core assignments. In ADAMANT, we use a dynamic task assignment model in which thread-to-core mapping decisions are fixed and re-evaluated at periodic time intervals or control *epochs*, similar to [99, 130].

In the context of Linux-like operating systems, ADAMANT replaces the load balancing mechanisms and is orthogonal to the *completely fair scheduling* (CFS) policy which is used to distribute processing time amongst tasks mapped to the same core. Unlike previous works that attempt to select both task-to-core assignments and core's operation frequency, ADAMANT leverages Linux's *ondemand DVFS governor* to set the core frequency according to the current processing time utilization of each core. ADAMANT works synergistically with DVFS by deploying performance estimation models that predict the DVFS governor behavior and uses this information during task mapping.

Section 6.3 provides a detailed definitions of ADAMANT workload model, performance metrics, and execution platform assumptions. Section 6.5 describes the ADAMANT predictive task mapping approach.

6.3 System model and metrics

Workload model: We assume a set of applications composed of a single or multiple threads. For uniformity, in this paper the term *task* is used interchangeably for both single-threaded processes and for individual threads of the same process. Tasks can be dynamically created and there is no prior knowledge of their requirements or execution time. As described previously, we perform task mapping at fixed periods called epochs. We denote the duration

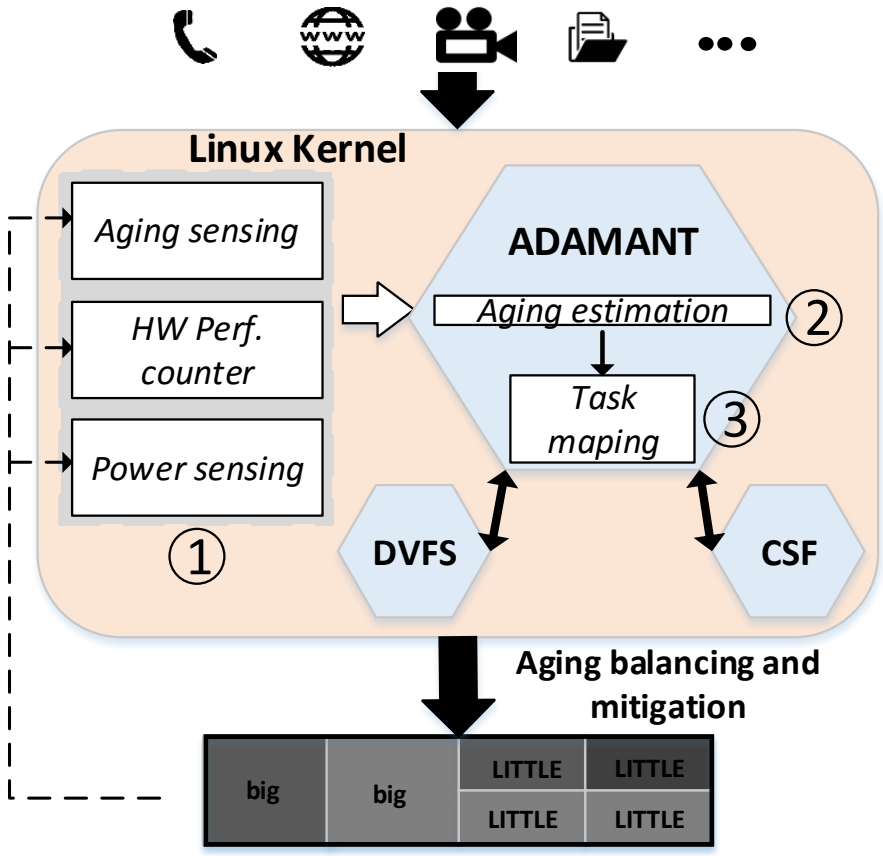


Figure 6.3: ADAMANT aging-aware tasks mapping for heterogeneous architectures.

of an epoch as τ_{map} and define the set $P(t_k) = \{p_1, p_2, \ldots, p_m\}$ of the tasks active at any given time, where $t_k = \tau_{map} * k$ denotes the time at the k^{th} mapping epoch.

Platform model: In this work, we consider HMP platforms consisting of multiple cache-coherent cores that share the same ISA and memory address space. Throughout the paper, we define the set of cores as $C = \{c_1, c_2, \ldots, c_n\}$ and the microarchitecture type of a core c_j as $y(c_j)$. Cores are grouped into clusters, such that all cores in the same cluster support the same voltage/frequency pairs and share the same frequency domain, therefore DVFS is always performed at cluster granularity. This is the model used by current implementations of the ARM's big.LITTLE architecture. We define the set of clusters as $Z = \{z_1, z_2, \ldots, z_s\}$. Fine-grained (per-core) DVFS is supported in our model by defining a different cluster for each core ($|C| = |Z|$). We use $F^{c_j}(t)$ and $V^{c_j}(t)$ to refer to the average frequency and voltage,

respectively, of core c_j during a time period t.

Performance and power sensing model: Our approach does not require a prior profile of the applications, so we assume that each core provides *hardware performance counters*(HPCs) and *power sensors*, which allow us to characterize task's workload and system efficiency. We consider that the following generic metrics can be extracted from HPCs and sensors to characterize the task's workload and system state during a time period t: $IPS^{p_i c_j}(t)$ is the average throughput of p_i when executing at c_j in terms of instructions per second; $U^{p_i c_j}(t)$ is the average task processing time utilization; $IPS^{c_j}(t)$ is the average core throughput across all tasks that executed on c_j; $U^{c_j}(t)$ is the average core utilization; and $W^{c_j}(t)$ is the average core power consumption.

Aging sensing model: We assume the availability of per-core critical path aging sensors [85] which enable us to evaluate aging in terms of the relative critical path delay degradation at any given time t: $\Delta D^{c_j}_{rel}(t) = D^{c_j}(t)/D^{c_j}_0$, where $D^{c_j}_0$ is the initial critical path delay (measured when the system is first deployed) and $D^{c_j}(t)$ is the current critical path delay of core c_j at time t, such that $D^{c_j}(t) < D^{c_j}_{gb}$, where D_{gb} is the maximum delay in which the core can operate at its maximum frequency. Throughout the text, we use the terms *frequency guardband* or *guardband slack* to refer to the difference $D^{c_j}_0 - D^{c_j}_{gb}$; we also refer to $D^{c_j}_{gb}$ as the core's *guardband delay*. Critical paths may age at different rates, therefore multiple aging sensors may be deployed [154]. In the remainder of this paper, we assume that $\Delta D^{c_j}_{rel}(t)$ corresponds to the most aged critical path.

6.4 System-level delay degradation model

For platforms that do not feature critical path sensors or for offline estimation of aging effects, aging can be assessed using aging models for critical path delay degradation. Using

aging models at runtime to replace sensors is usually overhead-less since transistors age at a slow rate and therefore it is not necessary to update $\Delta D_{rel}^{c_j}(t)$ at the granularity of mapping epochs (typically between $50ms$-$300ms$). Throughout the rest of this paper, τ_{aging} refers to the period of core aging assessment, i.e., the aging epoch, while τ_{aging}^{k} denotes the time at the k^{th} aging epoch. When using this aging estimation model, we assume the availability of temperature sensor that all allow us to obtain the average core temperature $T^{c_j}(t)$.

Aging-induced delay degradation in transistors is caused mostly due to BTI and HCI effects [22, 31], therefore we can define the delay degradation during the period of duration t as $D^{c_j}(t) = D_0^{c_j} + \Delta D_{BTI}^{c_j}(t) + \Delta D_{HCI}^{c_j}(t)$, where $\Delta D_{BTI}^{c_j}$ and $\Delta D_{HCI}^{c_j}$ is the delay degradation due to BTI and HCI since $D_0^{c_j}$ was measured. Given that technology and fabrication-dependent factors are known, the degradation of the delay of the cores' critical paths can be estimated at runtime using the BTI models from [22] and HCI models from [31] by determining duty cycle δ, switching activity α, and temperature for all transistors in the critical path. Since such information is not available at system-level, we follow a similar approach to [115], in which we assume all transistors in a component experience similar stress rate and temperature, thus similar aging rates. For core-level aging estimation, temperature can be obtained from sensors, while the average $\delta_{c_j}(t)$ and $\alpha_{c_j}(t)$ for a core c_j throughout a period of duration t can be defined as [115]:

$$\delta_{c_j}(t) = \frac{time_{active}^{c_j}(t) + time_{idle}^{c_j}(t)}{time_{active}^{c_j}(t) + time_{idle}^{c_j}(t) + time_{pg}^{c_j}(t)} \tag{6.1}$$

$$\alpha_{c_j}(t) = F^{c_j}(t) * \frac{time_{active}^{c_j}(t)}{time_{active}^{c_j}(t) + time_{idle}^{c_j}(t) + time_{pg}^{c_j}(t)} \tag{6.2}$$

where $time^{c_j}_{active}(t)$ denotes the amount of time the core is running a task, $time^{c_j}_{idle}(t)$ denotes the amount of time the core is clock gated, and $time^{c_j}_{pg}(t)$ denotes the amount of time the core is on a power gating state during the period t. This models the core at the worst case aging since we assume that duty cycle $\delta_{c_j}(t) = 1$ during the entire period the core is not power-gated and that switching activity $\alpha_{c_j}(t) = F^{c_j}(t)$ during the entire period the core is not clock-gated. Next, we describe in more details how BTI and HCI aging are periodically computed as a function of δ and α.

BTI aging impact: BTI is a two-phase effect: *stress* and *recovery*. During *stress* phase the transistor is ON and traps may be generated at the interface between channel and gate oxide. This effect is accentuated at higher temperatures and gradually increase the threshold voltage (V_{th}). On the other side, when the transistor is OFF, a *recovery* phase starts in which some traps are filled, thus leading to a partial decrease of V_{th}. Based on the models from [22, 115, 65], we define the delay degradation due to BTI during period of duration t as:

$$\Delta D^{c_j}_{BTI}(t) = A_{BTI} \times (\delta(t) \times t)^n \times e^{(\frac{-E_a}{k \times T^{c_j}(t)})} \times (V^{c_j}(t) - V_{th}) \times e^{(\frac{V^{c_j}(t) - V_{th}}{E_0})} \times D^{c_j}_0 \quad (6.3)$$

where A_{BTI} and E_0 are technology dependent factors, n is a constant depending on the fabrication process, k is a Boltzmann's constant, and E_a is the activation energy.

In order to capture the recovery phase in BTI degradation, $\Delta D^{c_j}_{BTI}(t)$ is always computed for the period t since the system was first deployed ($D^{c_j}_0$ was measured) and the current aging epoch τ^k_{aging}, averaging out the stress and temperature. Therefore, we need to consider the aging history from the previous aging epoch τ^{k-1}_{aging} for each core c_j to compute the delay due to changes in temperature and stress rate. Equation 6.4 shows that BTI degradation is a weighted function of stress and temperature for current aging epoch τ^k_{aging} based on history

(previous aging epoch τ_{aging}^{k-1}):

$$\Delta D_{BTI}^{c_j}(\tau_{aging}^{k}) = \Delta D_{BTI}^{c_j}(\frac{\tau_{aging}^{k-1}}{\tau_{aging}^{k}} \times \delta(\tau_{aging}^{k-1}) + \frac{\tau_{aging}^{k} - \tau_{aging}^{k-1}}{\tau_{aging}^{k}} \times \delta(\tau_{aging}^{k}), \\ \frac{\tau_{aging}^{k-1}}{\tau_{aging}^{k}} \times T^{c_j}(\tau_{aging}^{k-1}) + \frac{\tau_{aging}^{k} - \tau_{aging}^{k-1}}{\tau_{aging}^{k}} \times T^{c_j}(\tau_{aging}^{k})) \tag{6.4}$$

where τ_{aging}^{k} denotes the time period since the system was first deployed and the k^{th} aging epoch. Using this equation we can keep track of aging history as well as recovery phase in case that stress or temperature are decreased in the new epoch.

HCI aging impact: HCI is a dynamic mechanism that happens when the transistor is switching, when accelerated electrons inside the channel collide with the oxide interface, creating electron-hole pairs. Hence, the current-voltage characteristic of the transistor changes and leads to increase in V_{th}. The models from [31, 115, 65] define the delay degradation due to HCI for the period t since the system was first deployed as:

$$\Delta D_{HCI}^{c_j}(t) = A_{HCI} \times t^{0.5} \times \alpha(t) \times F^{c_j}(t) \times e^{(\frac{-E_b}{k \times T^{c_j}(t)})} \times e^{(\frac{V^{c_j}(t) - V_{th}}{E_0})} \times D_0^{c_j} \tag{6.5}$$

where A_{HCI} is a technology dependent factor and E_a is the activation energy. For HCI degradation there is no recovery period, so define $\Delta D_{HCI}^{c_j}(t)$ as the HCI delay degradation of the current epoch and the accumulated aging saved from previous epochs:

$$\Delta D_{HCI}^{c_j}(\tau_{aging}^{k}) = \Delta D_{HCI}^{c_j}(\tau_{aging}^{k-1}) + \Delta D_{HCI}^{c_j}(\tau_{aging}^{k} - \tau_{aging}^{k-1}) \tag{6.6}$$

where τ_{aging}^{k} denotes the time period since the system was first deployed and the k^{th} aging epoch.

6.5 Aging-aware task mapping

The objective of ADAMANT mapping phase is to find a task mapping that satisfies each task's performance requirements while reducing the total system power consumption and balancing aging across cores.

Problem formulation: Given the set of tasks $P(t_k) = \{p_1, p_2, \ldots, p_m\}$ that are active at the k^{th} mapping epoch, our goal is to find the set of all tasks p_i to be mapped to a core c_j during the next mapping epoch, defined as X^{c_j}, such that the total power consumption is minimized, tasks performance constraints are satisfied, guardbands are not violated, and all tasks are mapped to cores. The objective function is defined by Equation 6.7:

$$
\begin{aligned}
&minimize \sum_{\forall c_j \in C} W^{c_j}(t) \\
&constr : IPS^{p_i c_j}(t) \times U^{p_i c_j}(t) \leq IPS^{p_i}_{max}, \forall p_i \in X^{c_j}, \forall c_j \in C \\
&constr : D^{c_j}(t) < D^{c_j}_{gb}, \forall c_j \in C \\
&constr : P = \bigcup_{c_j \in C} X^{c_j}
\end{aligned}
\tag{6.7}
$$

where t is the time period until the next mapping epoch. We consider the performance constraint met if the task's *effective throughput*, defined as $IPS^{p_i c_j} \times U^{p_i c_j}$ (i.e. the average throughput including periods the task was not executing), cannot be increased by moving the tasks to a faster, i.e. the effective throughput is saturated (defined as $IPS^{p_i}_{max}$). This saturation is the typical case for interactive tasks with *computation—IO/sleep* cycles. By providing a faster core to such task, its IPS during the computation cycles ($IPS^{p_i c_j}$) increases, which in turn decreases the computation time and the core utilization ($U^{p_i c_j}$), thus limiting the task's effective throughput. For tasks that we do not observe IPS saturation, we assume that in the best case its performance constraint can only be met by the fastest core. This

is the typical case for non-interactive compute-intensive tasks with only computation cycles. Our choice of IPS saturation metric as the performance target is motivated by the fact that in our workload model tasks do not explicitly define deadlines or performance targets. Some works in [111, 63] use additional libraries such as the Heartbeat framework [79] to define throughput targets for task mapping. In ADAMANT we chose to use the implicit throughput target defined by the IPS saturation point in order to support unmodified Linux applications.

6.5.1 Task mapping algorithm

Optimally solving Equation 6.7 is an NP-hard combinatorial problem of complexity $\mathcal{O}(|C|^{|P|})$ (multiple tasks can be mapped to the same core in any order)[62]. Therefore, heuristic solution to efficiently find a task assignment at runtime is developed. The rationale of the ADAMANT task mapping algorithm is to map each task to a core that satisfies the target throughput $IPS_{max}^{p_i}$ and results in the smallest increase in total power while keeping the relative aging of cores balanced. This is achieved using a *list scheduling* [41] heuristic. The general idea of list scheduling is to order the processes/task to be scheduled/mapped, according to their priorities and map them in order.

Our heuristic is described in Algorithm 7. First, the throughput of all tasks across all core types is predicted (section 6.5.2) and the saturation IPS is set as the throughput constraint (line 3). Note that we correct the throughput constraint by a constant $0 < \omega < 1$. We currently use $\omega = 0.95$ to account for prediction errors. We compute the *availability* A^{p_i} for a thread p_i as the number of core the task can be mapped to without violating its performance constraint. Tasks are added to a list which is sorted according to the maximum energy efficiency that can be obtained (in terms of IPS/Watt) and the given task availability (lines 5, 7), so tasks that are more constrained are mapped first. Finally, the tasks

Algorithm 7 ADAMANT task mapping algorithm

1: ▷ Obtain tasks max. IPS as performance target
2: **for all** $p_i \in P$ **do**
3: $IPS^{p_i}_{max} \leftarrow maximum(IPS^{p_i c_j}(t) \times U^{p_i c_j}(t), \forall c_j \in C) \times \omega$
4: $A^{p_i} \leftarrow |\{c : IPS^{p_i}_{max} \leq IPS^{p_i c_j}(t) \times U^{p_i c_j}(t)\}|$
5: $task\_list.add(p_i)$
6: **end for**

7: $sort^{\frac{IPS/Watt}{A^{p_i}}}(task\_list)$
▷ Power minimization given aging penalty and perf. constraint
8: **for all** $p_i \in task\_list$ **do**
9: $c \leftarrow c_j : IPS^{p_i}_{max} \leq IPS^{p_i c_j}(t) \times U^{p_i c_j}(t) \times \Delta D^{c_j}_{norm}(t) \wedge W^{system}(t)$ is minimized ▷ If perf. constraint cannot be met. Find the best effort perf. given the aging penalty
10: **if** $c = \emptyset$ **then**
11: $c \leftarrow c_j : IPS^{p_i c_j}(t) \times U^{p_i c_j}(t) \times \Delta D^{c_j}_{norm}(t)$ is maximized
12: **end if**
13: $X^c \leftarrow p_i$
14: **end for**

are mapped in the order defined by the sorted list to the core that satisfies the described conditions above (lines 8–14).

In order to account for aging we apply a penalty $0 \leq \Delta D_{norm}(t) \leq 1$ in the computed performance when optimizing towards the objective. The aging penalty is defined by Equation 6.8 as the core's relative delay degradation normalized to the difference between the maximum and minimum relative degradation across all cores in the system. Equation 6.8 computes the final penalty by balancing the normalized delay degradation with the difference between the current delay $D^{c_j}(t)$ and the guardband delay $D^{c_j}_{gb}$. The $D^{c_j}_{gb}$ ratio is taken into account in order to avoid an excessive penalty for cases when there is aging imbalance but the guardband slack is high. The computation of the overall system power also incurs an aging penalty as shown in Equation 6.9. σ is a constant that can be used to control the tradeoff between aging balancing and power optimization as well as to set an upper bound to the power penalty applied. We employ σ to avoid excessively high W^{system} values as $\Delta D^{c_j}_{norm}$ approaches 0. The rationale is to avoid having a single core completely dominating the value of W^{system}, which may have a significant adverse impact on the mapping energy efficiency since power

variations caused by other cores would be negligible. We empirically set σ to 0.1.

$$\Delta D^{c_j}_{norm}(t) = \frac{1 - \frac{\Delta D^{c_j}_{rel}(t) - min(\Delta D_{rel})}{max(\Delta D_{rel}) - min(\Delta D_{rel})}}{2} + \frac{1 - \frac{D^{c_j}(t) - D^{c_j}_0}{D^{c_j}_{gb} - D^{c_j}_0}}{2} \tag{6.8}$$

$$W^{system}(t) = \sum_{\forall c_j \in C} \frac{W^{c_j}(t)}{max(\Delta D^{c_j}_{norm}(t), \sigma)} \tag{6.9}$$

6.5.2 Iterative performance/power prediction

As described in Sections 6.2 and 6.3, during the sensing phase hardware performance counters (HPC), power sensors and aging sensors are sampled and all tasks are characterized in terms of average IPS and power, i.e, $IPS^{p_i c_j}(t)$, $U^{p_i c_j}(t)$, $U^{c_j}(t)$, and $W^{c_j}(t)$. In the mapping phase, we explore candidate mappings for the next epoch. In this case, task execution metrics such as $IPS^{p_i c_l}(t), \forall c_l \neq c_j$ are therefore required. These values are not directly available (we only have p_i information regarding c_j and core types may differ thus yielding different performance) and must be estimated during the prediction phase.

We use the models proposed by [108] in order to obtain these performance metrics for any core type based on measurements from the sensing phase. $IPS^{p_i c_l}(t)$ is predicted using a linear regression model of the form $IPS^{p_i c_l}(t) = A^{y(c_l)}_{y(c_j)} * B^{c_j}_{p_i}(t)$, where $B^{c_j}_{p_i}(t)$ is a characterization vector containing the performance counter measurements of p_i at c_j. $A^{y(c_l)}_{y(c_j)}$ is a coefficient vector for predicting from core type $y(c_j)$ to $y(c_l)$ obtained through offline profiling. The prediction of $W^{c_l}(t)$ is performed in a similar manner.

$U^{p_i c_l}(t)$ and, $U^{c_l}(t)$, on the other hand, are not predicted using a regression model since they are a function of $IPS^{p_i c_l}(t)$, the temporal activity of the other tasks to be mapped to the

Algorithm 8 DVFS prediction algorithm

1: $iter \leftarrow 0$
2: **repeat**
3: $\quad \widehat{F^{c_l}}(t) \leftarrow F^{c_l}(t)$
4: $\quad$ Predict $\{U^{c_j}(t), \forall c_j \in z_s\}$ given $\widehat{F^{c_l}}(t)$ $\quad \triangleright$ z_s is c_l's DVFS cluster
5: $\quad F^{c_l}(t) \leftarrow$ predict frequency for cluster z_s using ondemand threshold for the $\{U^{c_j}(t)\}$ loads
6: $\quad iter \leftarrow iter + 1$
7: **until** $iter > \tau_{map}/\tau_{dvfs}$ or $\widehat{F^{c_l}} = F^{c_l}$

same core c_l, and the scheduling policy adopted by the operating system[108]. In this paper, we assume that Linux's CFS policy is used to schedule tasks mapped to the same core and use the CFS performance estimation model proposed by [108] to predict $U^{c_l}(t)$. However, in contrast to [108], we consider the use of DVFS, therefore we extended [108] method in order to adjust $U^{c_l}(t)$ according to frequency variations. In this paper we assume Linux's on demand DVFS governor is used. Algorithm 8 is used to estimate the frequency $F^{c_l}(t)$ that will be set by the governor in the next epoch assuming τ_{map} as the length of the task mapping epoch and τ_{dvfs} as the length of the DVFS epoch. The frequency prediction is performed based on the previous core load prediction. Since the updated frequency prediction might affect the core load, this process is repeated until the predicted frequency becomes stable or a maximum number of predictions are made (defined by τ_{map}/τ_{dvfs}).

It is important to note that the prediction routines are continuously called in Algorithm 7(lines 8–14), since mapping decisions from previous iterations affect the available processing time on each core (for cases when multiple tasks are mapped to the same core) and the predicted frequency due to DVFS. Total system power also has to be recalculated to capture power changes resulted from coarse-grained DVFS (e.g. cluster-level).

6.5.3 Complexity analysis

Given the number of cores $|C|$, core types $|Y|$, and tasks $|P|$, Algorithm 7 has complexity of $|P| \times |Y|$ for the target throughput detection (lines 2–6), $|P|^2$ for sorting, and $|P| \times |C|$ for the mapping phase (lines 8–14). If we assume $|P| >> |Y|$, an average case performance of $|P| \times log(|P|)$ for sorting (if quick sort is used), and that $|C| \geq |Y|$, then the runtime is bound by $\mathcal{O}(|P| \times |C|)$ defined by the mapping phase.

6.6 Experimental evaluation

6.6.1 Setup

We developed a trace-based aging simulation framework to enable the evaluation of the effects of task scheduling in long term wearout (Fig. 6.4) without long periods of live execution. Applications are executed using gem5 [25] integrated with McPAT [98] to simulate an HMP consisting of the core types described in Table 6.1. Each application executes individually as a single thread in gem5 full system mode on every possible core type and frequency. Temporal traces with periodic simulation statistics and power estimation are produced (every 1ms of simulated time), and used as input for the trace-based aging simulator.

The trace-based simulator works at the granularity of DVFS epochs and emulates the execution of each core in four steps: **1)** The trace information produced by gem5/McPAT is used to obtain maximum amount of processing time a task would use during the DVFS epoch and the average length of the task's *computation—IO/sleep*, i.e., the task's duty cycle; **2)** the current task-to-core assignment and the tasks' duty cycles are used as input for to which simulates the behavior of the Linux CFS scheduler and produces the exact runtime allotted by the OS scheduler to each task; **3)** the execution of each task is emulated (using

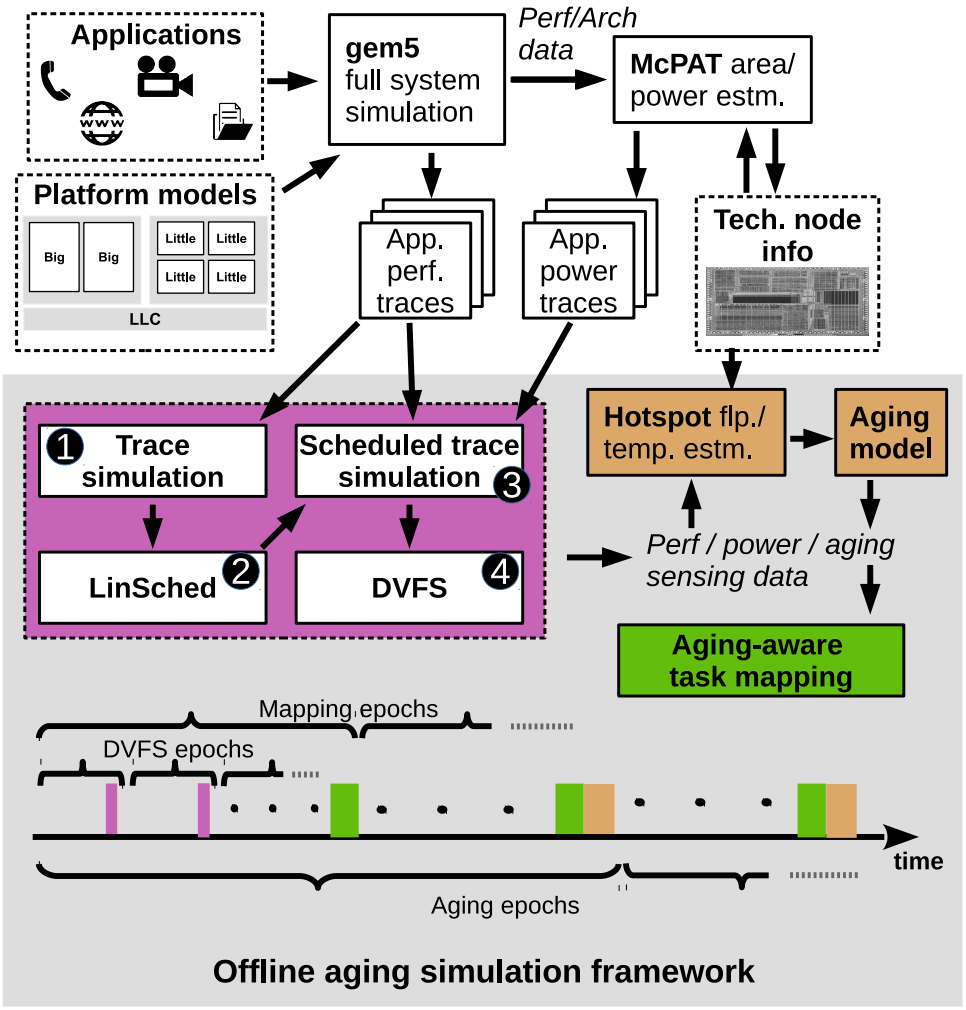


Figure 6.4: Relative difference between the most and the least aged cores in homogeneous and heterogeneous architectures.*typical* workload (Table 6.2). GTS scheduling [14]

the performance/power traces) for the time allotted by LinSched; **4)** The DVFS governor algorithm sets the frequency for the next epoch according to the emulated load.

The trace-based simulation is repeated τ_{map}/τ_{dvfs} times until the mapping epoch, when Algorithm 7 uses the information collected from the trace simulation to find a new task mapping for the next epoch. This process is repeated until the end of an aging epoch (τ_{aging}), when the critical path delays are updated. At this time critical path aging sensors are simulated using the aging model described in Section 6.4. We use Hotspot [137] to obtain the temperature estimation required for aging estimation.

In this paper we consider an 8-core platform with *two core types*: large 4-way out-of-order (OoO) cores with speculative execution (*Big cores*) and small in-order cores (*Little cores*), which are representative of current HMPs for mobile devices. However, our model is not restrictive with respect the degree of heterogeneity (i.e. $|Y| > 2$).

Table 6.1: Heterogeneous Core Parameters

Parameter[1]	Big	Little
Issue width	4 (OoO)	2 (in-order)
LQ/SQ size	16/16	8/8
IQ size	32	16
ROB size	128	64
Int/float Regs	128	64
L1 I$/D$ size (KB)	32/32	32/32
L2 $ size (KB)[2]	512	128
VF pairs	2GHz,1V	1.5GHz,0.8V
	1.5GHz,0.8V	1GHz,0.7V
	1GHz,0.7V	500MHz,0.6V

6.6.2 Workloads

In our experimental evaluation, we define workload based on the analysis performed by [60], which characterizes popular mobile applications in terms of potential for thread-level parallelism and core utilization. They verified that in a typical scenario only one core is utilized 70% of the time in an 8-core big.LITTLE platform, even when multiple applications are running simultaneously. Based on the observations from [60], we used the *x264 (x2)*, *bodytrack (bt)*(compute intensive) and *blackscholes*(bs) (memory intensive) applications from PARSEC [23] to devise the mobile-like core utilization scenarios shown in Table 6.2. The applications are executed periodically in order to generate the average load shown in Table 6.2 during an aging epoch. We define two typical mobile scenarios: *typical* and *typical(heavy)*, while the other cases represent more extreme scenarios in terms of both stress and system

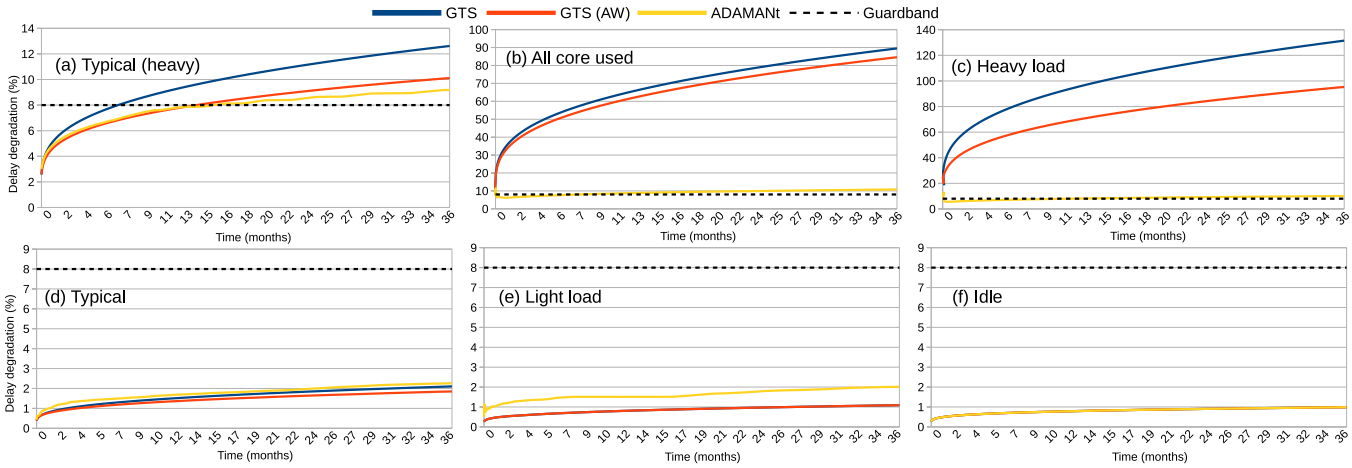


Figure 6.5: Delay degradation. In the (f)*Idle* case, only background OS services are running with no active tasks

idleness.

Table 6.2: Workload patterns: Benchmark Name/Number of Tasks/Max

Typical	Typical(heavy)	Light load	Heavy load	All cores used
bt/5/0.04/0.17	**bt**/5/0.04/0.17			**x2**/4/0.97/1.00
x2/1/0.30/0.96	**x2**/1/0.30/0.96	**bc**/8/0.06/0.11	**x2**/8/0.97/1.00	**x2**/4/0.30/0.96
	x2/4/0.97/1.00			

6.6.3 Results and discussion

We used our framework to simulate three years of aging on our 8-core platform composed of a cluster of four *Big* cores and a cluster of four *little* cores. We compare ADAMANT against Linux GTS scheduling for heterogeneous architectures. For fairness of comparison, we also extend *GTS with aging-awareness* (GTS AW). GTS AW uses our aging penalty ΔD_{norm} as a baseline *virtual load* when performing load balancing (i.e., aged cores tend to have less compute intensive tasks mapped to them). GTS AW illustrates the case in which aging mitigation is applied only within clusters of cores of the same type. For these experiments we assume a guardband of 8% with respect to the maximum frequency of each core. The

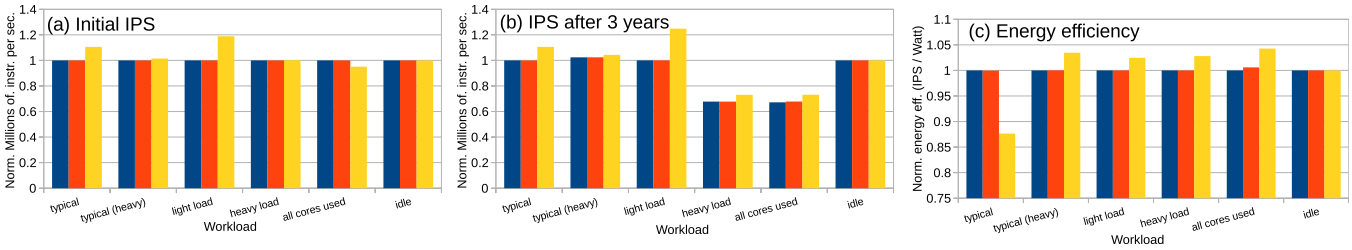


Figure 6.6: Performance degradation under DVFS frequency capping

aging epoch length is 1 *day*, task mapping epoch is 200ms, and the DVFS epoch is 50ms. σ (Equation 6.9) is set to 0.1. We simulate aging for a time frame of 3 *years*.

Fig. 6.5 shows the relative delay degradation for the representative workloads described in Table 6.2. For the *Typical(heavy)* workload (Figure 5.a), GTS violates the guardband within less than a year of runtime. With ADAMANT, we are able to increase the time to guardband violation $\sim 100\%$ (about 8 months). Figures 6.5.b-c show aging scenarios under high stress. For these cases, the guardband is violated at the very beginning of execution due the simultaneous utilization of all *Big* cores. ADAMANT is able to adapt and mitigate aging by prioritizing the use of the *little* cores. It's worth mentioning that under nominal behavior, the system would be throttled due to violation of thermal constraints [136]. Figures 6.5.d-f show the low stress scenarios. ADAMANT delay degradation is slightly higher than GTS since the $rel_{norm}\Delta d$ penalty is also proportional to the distance of the sensed delay and the guardband violation delay. For these scenarios, ADAMANT is able to adapt to the available guardband slack and prioritizes other metrics for task mapping (such as IPS and energy efficiency) as shown in Figure 6.6.

Figure 6.6 shows the effects of aging when aging-aware DVFS is applied, and also compares ADAMANT with GTS in terms of performance and energy efficiency. For these cases, the frequency is reduced when the guardband is violated in order to allow continuous operation. In general ADAMANT yields better performance and energy efficiency. For example, for the *typical* case (no guardband violation) ADAMANT improves performance by $\sim 10\%$.

For the case when frequency capping was applied, ADAMANT improves performance by $\sim$ 7% after three years of aging. One exception can be seen for the *typical* case, in which ADAMANT improved performance by trading off energy efficiency. In this case the ***bt****/5/0.04/0.17* should have been mapped to the little core cluster, however, it is possible that due to $IPS_{max}^{p_i}$ mispredictions, some tasks were moved to the big cluster, thus reducing the energy efficiency.

Figure 6.6 also shows that GTS and GTS AW yield the same performance and energy for both aging scenarios. GTS first chooses to which cluster a task will be migrated and then performs either load balancing (GTS) or aging-aware load balancing (GTS AW) within the cluster, therefore performance and/or power degradation resulting from forcing a task to a different cluster is not expected since GTS AW won't take this decision. For the cases GTS AW is able to reduce the aging rate (Figures 6.5.a-c), we also observe identical performance after 3 years for both GTS and GTS AW since both will have the same frequency capping in the Big cluster (from 2GHz to 1.5GHz — Table 6.1) after 3 years.

Overheads: The total overhead of each phase in ADAMANT is negligible when compared to the $200ms$ mapping epoch and the 1 *day* aging epoch. In order to measure sensing overhead of ADAMANT, we modified the `schedule()` function of the Linux kernel used in the gem5 full system simulator to sample performance counters every time there is a task context switch, which allows us to collect the per-task performance metrics required in the subsequent ADAMANT's phases. The total *sensing* overhead varies between $13\mu s$ (on a *Big* core) and $22\mu s$ (on a *Little* core) per epoch and per core. We also executed our aging simulation framework within gem5 in order to measure the latency of the prediction and task mapping phases. Across the workload runs described previously, the average measured latency of these phases is $12\mu s$ (measured on a *Big* core 2GHz).

In a platform that utilizes critical path aging sensors, delay degradation monitoring time is very fast and imposes negligible overhead [65, 50]. Otherwise, induced aging delay degrada-

tion can be estimated using the models from Section 6.4. Estimating the aging for 8 cores in our experimental platform takes $60\mu s$ on average. If temperature sensors are not available, temperature must be estimated by using, for instance, Hotspot [137], which would incur an additional overhead of $10ms$. The latency of aging estimation can be high when compared to the latency of the mapping phase, but, as mentioned previously, aging only needs to be assessed at the end of an aging epoch (1 *day*), therefore we consider this overhead to be negligible.

Limitations: As mentioned previously, we implemented a trace-based offline simulator in order to allow us to simulate long periods of aging (e.g., 3 years) without long periods of live execution. Therefore, it is important to mention that this approach has limitations when compared to live execution. Our trace-based offline simulator does not model contention on shared data and thread synchronization, thus only workloads with independent threads are simulated in order to avoid these effects. Additionally, our offline simulator does not capture cache warmup latency caused by thread migration as well as runtime voltage/frequency switching overhead when DVFS is applied. However, at the granularity that task migration and DVFS (every 200ms and 50ms, respectively) are performed, we assume their impacts in the accuracy of the results are negligible. The main source of inaccuracy that we do not capture with our offline simulations is contention on the main memory bus. In the simulated 8-core big.Little configuration with all cores running identical workloads (which maximizes the contention on the main memory due to coincidental workload memory phases), we observed a maximum performance difference of 5% (across the PARSEC workloads x264, bodytrack, and blacksholes) when we compare performance of offline traces against a live 8-core gem5 simulation (which captures all memory contention affects). We believe this to be a worst-case performance difference since during the aging simulations not all cores are active and/or with coincidental memory phases, therefore we believe this would not have a significant impact on the observed results and conclusions.

6.7 Chapter summary

Heterogeneity in multicore systems provides better energy efficiency but are more prone to unbalanced workload distribution and premature wear-out in mobile platforms. To mitigate this issue, ADAMANT, an aging-aware task mapping approach for HMP is proposed. ADAMANT is orthogonal to other system-level mechanisms such as DVFS by employing sensing and predictive models for runtime workload characterization. Our experimental evaluation using workloads derived from realistic mobile scenarios shows that ADAMANT improves both lifetime and performance when compared to both the vanilla and an aging-aware implementation of Linux's GTS load balance for HMPs. For typical workloads, ADAMANT improves lifetime by up to 2x without performance degradation. Under DVFS frequency capping scenarios, ADAMANT improves performance by about 7% when compared to the modified aging-aware GTS.

Chapter 7

AROMa, a novel aging-aware adaptive routing and online monitoring for 3D and 2D NoCs

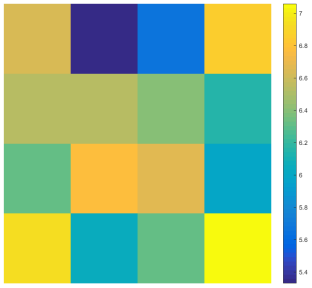
To cope with the complex on-chip interconnection issues in many-core systems, the three dimensional NoC has been proposed by applying die stacking technology for performance, energy efficiency and power consumption gains. Furthermore, 3D NoCs become a promising solution to many-core systems for its scalability which integrate large number of homogeneous or heterogeneous intellectual properties (IP)s, e.g. processing units [4, 15, 80]. Nevertheless, the reliability challenges in advanced silicon technology may jeopardize the performance gain as well as scalability of many-core systems. One of those arising challenges is aging mechanisms which are escalated in high density stacked die integration, [121, 82, 45, 40].

Aging happens when a transistor is under stress and temperature is high. *Bias Temperature Instability* (BTI) and *Hot Carrier Injection* (HCI) are the dominant aging mechanisms that gradually increase the threshold voltage (V_{th}) of transistors [2, 45, 64, 66]. The shifted V_{th}

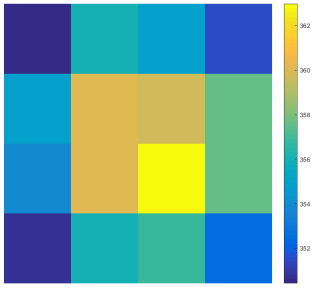
leads to undesired increase in system critical path delay which ultimately exacerbates performance loss and timing failure within the system components in the long run. Therefore, designers have to allocate considerable guardband to the critical path for avoiding timing failure. This imposes power, area, and performance overheads to the system [40, 149, 107]. Hence, NoC requires careful aging investigation to maintain high bandwidth, parallelism, and scalability in many-core systems. Aging-induced routers' performance degradation yielding to timing failure and connectivity loss in the NoC [64, 19, 20, 12, 10]. In addition, temperature is currently a controversial challenge in 3D design which compels further aging investigation as compared to 2D NoC. As shown in Fig. 7.1, we observed that even after running uniform random distribution of tasks, routers in different layers of a $4 \times 4 \times 4$ 3D NoC experience different temperatures and stresses. This leads to imbalanced aging degradation of routers at different layers and cause some routers to age more than the others.

Stress in BTI is the transistor's duty cycle. BTI-induced delay degradation is partially recoverable when transistor switches OFF. Stress in HCI is switching activity of transistors. Therefore, aging is a function of workload that changes both temperature and stress on the routers' critical paths' transistors. Since flits are the only stimuli in a router, both temperature as well as stress are functions of flits. Moreover, the router's capacity of flits for a given period of time is limited and also predictable. This means we can predict temperature and stress as well as aging of a router based on flits. To this end, we count *number-of-flits* (fl) and their *residence-times* (rs) in a given period of time $t = \epsilon$. Therefore, we proposed *Centralized Aging Table* (CAT) in [64]. CAT is populated by the amount of aging degradation for different ranges of fl and rs in a router from zero up to the router's capacity for a specific time ϵ. All in all, various pairs of (fl_i, rs_j) corresponds to different temperature amounts, stress values, and thus aging rates. This allows us to monitor aging independent from the running workload.

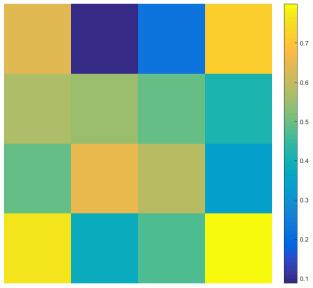
The inter-layer temperature difference of stacked die causes imbalanced aging between them



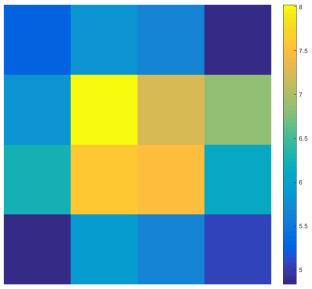
(a) Age in L_0 (%)



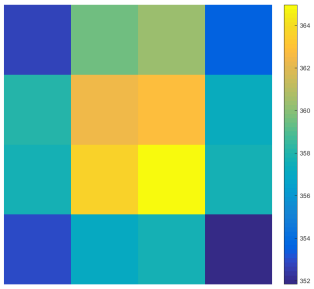
(b) Temp. in L_0 (K)



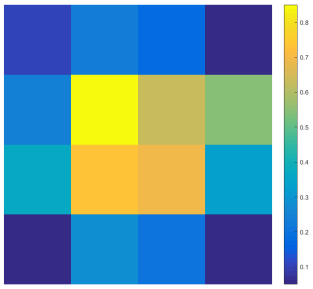
(c) Stress in L_0



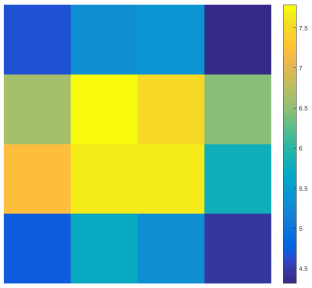
(d) Age in L_1 (%)



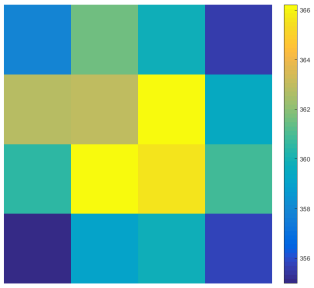
(e) Temp. in L_1 (K)



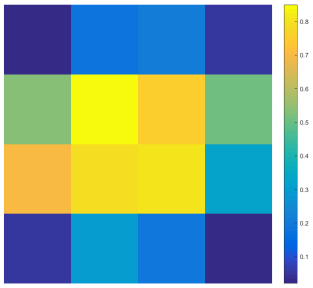
(f) Stress in L_1



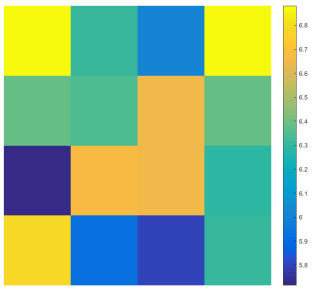
(g) Age in L_2 (%)



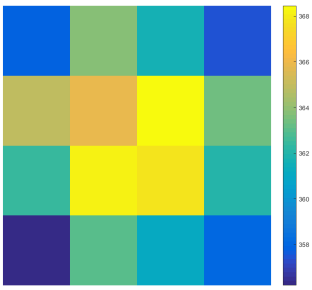
(h) Temp. in L_2 (K)



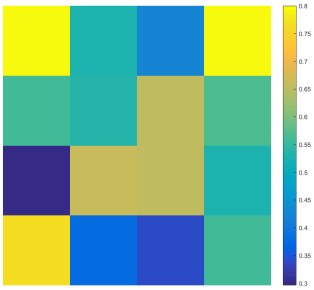
(i) Stress in L_2



(j) Age in L_3 (ns)



(k) Temp. in L_3 (K)



(l) Stress in L_3

Figure 7.1: Age, temperature (Temp.) and stress maps in each layer L_i of 3D NoC ($4 \times 4 \times 4$) for uniform random distribution.

(Fig. 1). In this work, we propose *Distributed CAT* (D-CAT) to quantify the gap due to temperature change in different perpendicular distances of layers from the heat sink. Therefore, each layer has its own D-CAT to increase the aging monitoring system scalability and catch the temperature variation of different layers. This means that the same pair of (fl_i, rs_j) results in different aging rates due to varying temperature amounts of layers in 3D NoCs. Our proposed aging monitoring system is independent from the diverse behavior of running application and is able to monitor aging online for routers at different layers of the network.

In addition, because fl and rs pairs are stimulated by network routing algorithm, routers' ages also have direct correlation to it. Since, there are different shortest paths between source-destination pairs in an NoC, the network can adaptively swap between them to decrease the stress and temperature on the routers. To this end, we proposed AROMa, which is an aging-aware adaptive routing coupled with our novel online monitoring system to avoid aging imbalance in routers and ultimately increase lifetime of the system.

The proposed techniques in [19, 20, 152, 11] focused on BTI-induced aging delay degradation in 2D NoC. These techniques are either offline [19, 20], while aging is significantly affected by runtime operation conditions, or impose large overhead to the systems [152, 11]. The proposed techniques in [19, 20, 152] assign lifetime budget to each router that results in their premature routers failure. Starting with unchanged biased budgets assignment to each router incurs imbalance and unfair aging which results in over-aged routers. Furthermore, the proposed budgeting techniques in [19, 20] are dependent on the benchmark characteristic used in profiling phase which reduce their applicability. Additionally, a complex circuitry is required to compute budgets in [152] as well as a parallel network to propagate budgets in the network that sustain significant overhead. We implemented the proposed **OF**fline budgeting for adaptive **A**ging-aware **R**outing (OFAR) method based on [19, 20, 152], which assign lower budgets to highly utilized routers during profiling to lower their load at runtime

and utilized our online monitoring system for tracking the aging rates of routers.

The main contributions of this work can be summarized as follows:

- We formulate aging effects due to HCI and BTI for 3D NoCs using our proposed online monitoring distributed centralized aging table (D-CAT) in AROMa. D-CAT is able to quantify the gap due to temperature change in different perpendicular distances of layers from the heat sink and system's conditions expressed by stress in 3D NoC. Using D-CAT, routers are able to keep track of their ages at each determined time ϵ.

- We proposed AROMa, an online adaptive aging-aware routing algorithm and online monitoring system for 3D NoCs. AROMa chooses one of k-best shortest paths between each source-destination pairs, which has least aged routers by avoiding the maximum aged ones. This adaptivity happen at each period of time $P = n \times \epsilon$.

- We implemented the proposed strategy using gem5 full system mode and compare it to OFAR and **N**on-**A**ging a**W**are routing (NAW) for both 2D and 3D NoCs.

Our extensive experimental analysis using gem5 full system mode for PARSEC and SPLASH-2 benchmark suits is done for both 2D (4×8) mesh topology and its respective 3D NoCs ($4 \times 4 \times 2$) version. These results for three years of execution show that AROMa outperforms state-of-the-art works (OFAR) when they are compared to non-aging aware XY and XYZ routing (NAW). On average, AROMa improves maximum aging by 33% and 34% in 2D and 3D NoC, respectively, in comparison to NAW while OFAR worsens it by 31% and 51%. Similarly, AROMa improves age imbalance significantly by 61% and 72% in 2D and 3D NoC, respectively, while in OFAR age imbalance is worsened by 69% and 120%. We can conclude that since OFAR assigns budgets offline and is not able to adaptively change between shortest paths. Also, OFAR's main purpose is to transfer traffic to less loaded routers and over-utilize these routers to the point that some of them fail. Although, OFAR shows acceptable

improvements for certain benchmarks, it fails on others. This shows that the previous techniques are application dependent and less flexible. Moreover, the experimental results show that 3D NoCs are more robust against aging as compared to 2D NoCs, since the paths are shorter and routers experience less stress. AROMa imposes negligible overheads in comparison to OFAR.

The rest of this chapter is organized as follows. In Section 7.1, the 3D NoC background is demonstrated. After that, an overview of the impact of aging mechanisms in NoC is detailed in Section 7.2. Section 7.3 discusses problem formulation. AROMa aging monitoring system is proposed in Section 7.4. Section 7.5 elaborates AROMa's adaptive routing. Section 7.6 studies related work. Then, experimental setup and results are discussed in Section 7.7. Finally, the chapter is summarized in Section 7.8.

7.1 3D NoC background

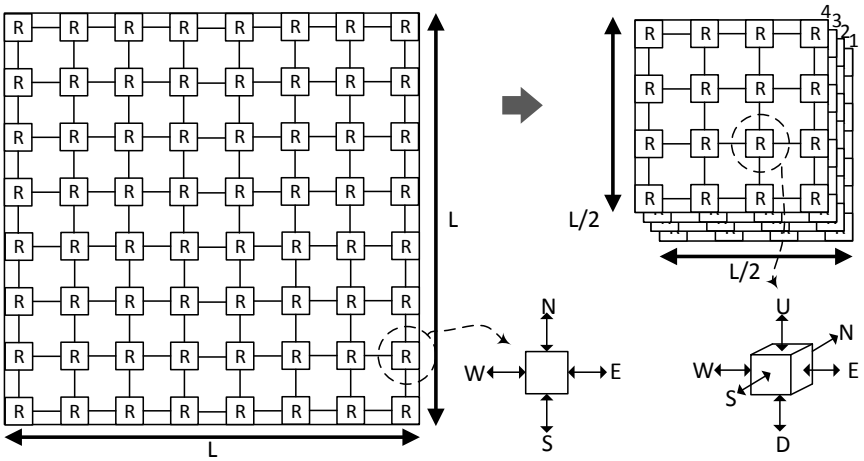


Figure 7.2: Comparison of 2D and 3D NoC area overheads.

Communication between cores in systems with many cores play a significant role [2, 78, 15]. With system design evolution throughout the decades, in order to integrate more cores in

the same chip, the demand for more scalable and structural interconnect grew. Moving to NoC paradigm introduced better alternative to the old crossbar or bus model. Therefore, several researches related to 2D NoC interconnect can be found such as [80, 90].

As shown in Fig. 7.2, the 2D mesh can be converted to 3D corresponding NoC by stacking layers (4 layers in the figure) on top of each other and decrease area overhead roughly by 4× in the horizontal direction [89, 100]. The design of the 2D router has up to five ports in each direction (N, S, E, W and local). A direct extension to 3D NoC is to add two more ports in the up and down direction forming vertical links that connect different layers. These vertical links are shorter compared to horizontal links and are called *Through Silicon Vias* [166, 92]. The use of 3D mesh NoCs is intensively researched for its promising performance gain, less power consumptions, reliability enhancement, design regularity, easy of implementation and heterogeneous system support compared to 2D mesh network [129, 159].

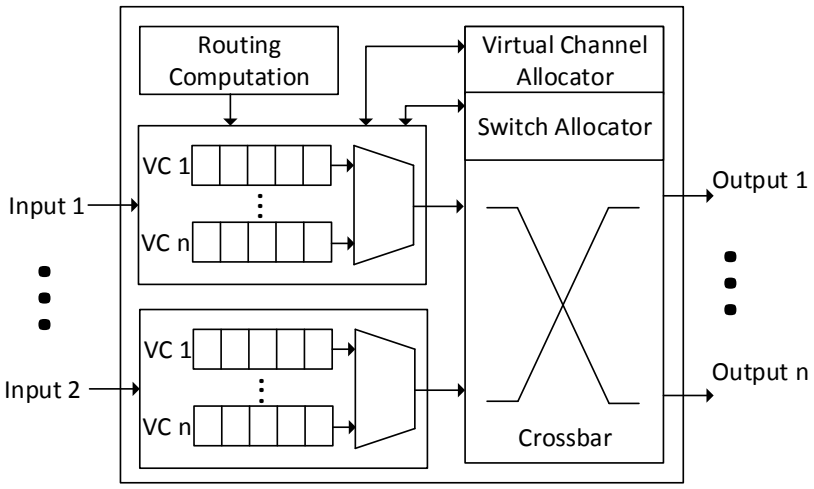


Figure 7.3: Router components and architecture.

In Fig. 7.3, the router architecture is depicted. We assume a pipelined router which is composed of 5 stages [3, 90, 52]. Our router [1] pipeline stages, components and their functionalities are summarized below:

- *Buffer Write and Routing Compute* (RC): in this stage the incoming flits are stored in input ports' buffer slots. Simultaneously, the routing compute logic determine candidate output port and its respective virtual channel using routing table. At this stage, routing can be precomputed at the upstream router if lookahead routing is implemented.

- *Virtual Channel Allocation*(VA): its function is to assign available output virtual channels to waiting flits stored in input buffer respective virtual channel.

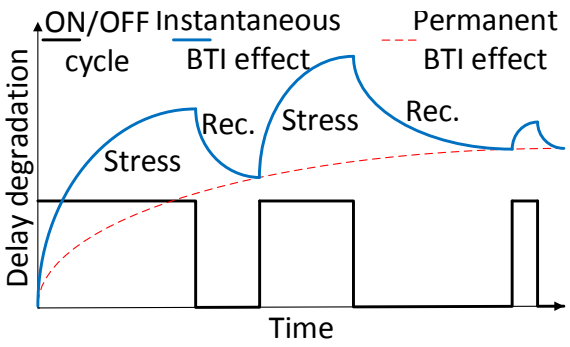
- *Switch Allocation* (SA): at which a flit in one of the input buffer slots which are ready to be received wins the crossbar switch time slot after arbitrating between them.

- *Switch Traversal* (ST): at this stage, flits are sent through crossbar switch to their appropriate output.

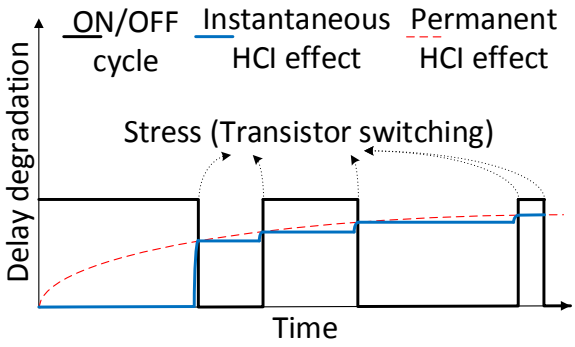
- *Link Traversal* (LT): transferring flits through links to their appropriate next router happens at this stage.

7.2 Aging-induced delay degradation background

BTI and HCI are the most dominant aging mechanisms that cause aging-induced delay degradation in transistors [2, 64, 105, 65, 45]. When a transistor ages, its threshold voltage (V_{th}) increases that leads to slower switching and higher propagation delay. Transistors aging along circuits critical paths deteriorates performance and/or causes timing failure at system level. The delay of an aged transistor at time t can be shown as $d_t = d_0 + \Delta d(t)$, where d_0 is the intrinsic delay of transistor at time $t = 0$ and $\Delta d(t)$ is the amount of delay degradation or transistor's age. In the following, we elaborate BTI and HCI aging mechanisms and their influences on transistor's age, $\Delta d(t)$.



(a) BTI-induced delay degradation mechanism.



(b) HCI-induced delay degradation mechanism.

Figure 7.4: BTI and HCI aging mechanisms dependency on stress.

7.2.1 BTI aging impact

As shown in Fig. 7.4.a, BTI is a two-phase mechanism: stress and recovery. During the stress phase, the transistor is ON and its V_{th} increases gradually, while if the transistor turns OFF recovery phase starts and the threshold voltage decreases gradually but partially. During stress and recovery phases instantaneous BTI effects eventuate [149, 100] which leave permanent decay behind. According to [100, 147], BTI occurs due to interface trap generation, hole trapping in available defects, and oxide bulk trap generation. While breaking in $Si-H$ bonds at Si/SiO_2 interface results in trap generation, hole trapping in preexistent process defects is a fast phenomenon that recovers fully after stress. Whereas, oxide trap generation is dependent upon voltage drop across SiO_2 interlayer. Works in [21, 116] demonstrate additional elaborations. This generation and destruction of traps leads to fluctuations based on stress and recovery phases yielding increase then lesser partial decrease of the V_{th} of transistors. Therefore, BTI is considered a static mechanism that depends on the stress imposed by the so called duty cycle, i.e. the portion of time that transistors are ON.

The extensively studied literatures in [21, 116] are unanimous that the performance of transistor deteriorates due to temperature and stress in BTI. We utilized the analytical

closed form model based on them:

$$\Delta d_{BTI}(t) = \Delta d_{BTI}(t_{stress}) \times \Big[1 - \sqrt{\eta \times \frac{t_{recovery}}{t}}\Big] \tag{7.1}$$

$$\Delta d_{BTI}(t_{stress}) = C_{BTI} \times t_{stress}^{n} \times e^{(-\frac{E_a}{K \times T})} \times d_0 \tag{7.2}$$

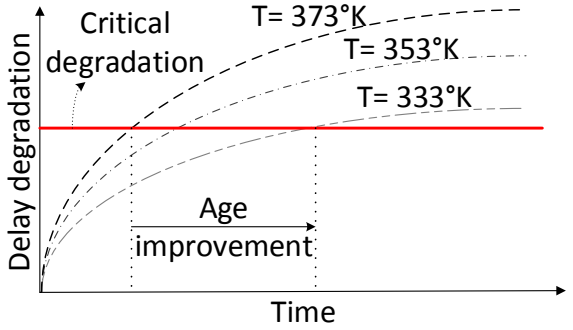
$$t_{stress} = Y \times t \tag{7.3}$$

$$t_{recovery} = (1 - Y) \times t \tag{7.4}$$

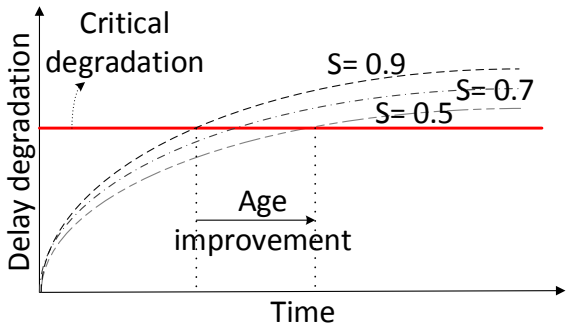
where, T is temperature in Kelvin, Y is duty cycle, t is age in seconds, K is Boltzmann's constant, d_0 is the transistor pre-aged intrinsic delay, E_a is activation energy, n is technology dependent parameter, η is a constant and C_{BTI} is the technology dependent fitting parameter.

7.2.2 HCI aging impact

Unlike BTI, HCI is a dynamic mechanism that depends on the switching activity of the transistor, as demonstrated in Fig 7.4.b. HCI happens when accelerated electrons of the channel collide with the oxide interface and create carriers (i.e. electron-hole pairs). Some of the carriers are deposited into prohibited transistor areas (e.g. the gate oxide). During time, deposited carriers alter the conductive properties of the transistor and eventually lead



(a) Different temperature values at same stress, S=0.5



(b) Different stress values at same temperature, T=333°K

Figure 7.5: Temperature and stress impacts on delay degradation.

to increase in its V_{th}. Similar to BTI, and as depicted in Fig 7.4.b, transistor's performance degrades due to temperature and stress (switching activity) due to HCI. We utilized the following model based on [145, 134]:

$$\Delta d_{HCI}(t) = C_{HCI} \times t_{stress} \times t^{-0.5} \times e^{(-\frac{E_a}{K \times T})} \times d_0 \tag{7.5}$$

$$t_{stress} = \alpha \times f \times t \tag{7.6}$$

where, T is temperature in Kelvin, α is switching activity, f is clock frequency in Hz, t is age in seconds, K is Boltzmann's constant, d_0 is the pre-aged intrinsic delay of transistor, E_a is activation energy, and C_{HCI} is the technology dependent fitting parameter.

7.2.3 Joint impact of BTI and HCI

As described and shown in Eq. 1 and Eq. 5, BTI and HCI aging mechanisms are exponential function of temperature (T) and non-linear function of stress (S). It needs to be noted that stress in BTI (S^{BTI}) is duty cycle (Y) and stress in HCI (S^{BTI}) is switching activity (α).

Fig. 7.5.a shows delay degradation (aging rate) for different temperatures but same stress. Clearly, delay degradation commensurate with higher temperature because the transistor experience increasing aging rate. This results to earlier failure due to missing the critical degradation limit (i.e the so called guardband). Similarly, in Fig. 7.5.b, aging rate increases in proportion with higher stress but same temperature. Based on Fig. 7.5, we can conclude that different temperature and stress pairs leads to dissimilar aging rates. Accordingly, the transistor's age at time t for BTI and HCI aging mechanisms is equal to:

$$d(t) = d_0 + \Delta d_{BTI}(t) + \Delta d_{HCI}(t) \tag{7.7}$$

Since, the aforementioned modelings compute delay degradation for time t, it is required to keep track of the aging history if we want to compute delay degradation in consecutive periods of time. For example, in Fig. 7.6 it is shown that since in each time period t_{i-1} to t_i the running workload behavior and characteristics change, it will consequently affect the temperature and stress as well. For instance, in first time period t_0 to t_1, temperature and stress pair is $< T_1, S_1 >$, while in time period t_1 to t_2, it is $< T_2, S_2 >$. Therefore, Eq. 1 and Eq. 5 for time period t_0 till t_i can be rewritten as Eq. 8 and Eq. 9, respectively:

$$\Delta d_{BTI}(t_i) = \Delta d_{BTI}(T_i, S_i^{BTI}, t_i) \tag{7.8}$$

$$\Delta d_{HCI}(t_i) = \Delta d_{HCI}(T_i, S_i^{HCI}, t_i) \tag{7.9}$$

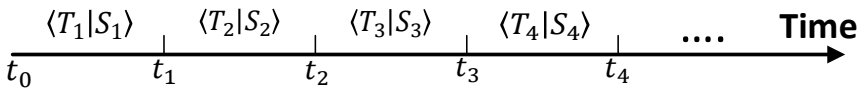


Figure 7.6: Different delay degradation due to temperature and stress in consecutive time periods.

Utilizing Eq. 8, BTI-induced delay degradation can be computed for time period t_i to t_{i+1} based on delay degradation for time period t_0 to t_i as follows:

$$\Delta d_{BTI}(t_{i+1}) = \Delta d_{BTI}(\frac{t_i}{t_{i+1}} \times T_i + \frac{t_{i+1} - t_i}{t_{i+1}} \times T_{i+1}, \frac{t_i}{t_{i+1}} \times S_i^{BTI} + \frac{t_{i+1} - t_i}{t_{i+1}} \times S_{i+1}^{BTI}, t_{i+1}) \tag{7.10}$$

This weighted function of temperature and stress can capture the history and BTI aging recovery due to reduction in stress or temperature. However, HCI does not have recovery phase. Therefore, we can utilize Eq. 9 to compute HCI-induced delay degradation for time period t_i to t_{i+1} based on delay degradation for time period t_0 to t_i as follows:

$$\Delta d_{HCI}(t_{i+1}) = \Delta d_{HCI}(T_i, S_i^{HCI}, t_i) + \Delta d_{HCI}(T_{i+1}, S_{i+1}^{HCI}, t_{i+1}) - \Delta d_{HCI}(T_{i+1}, S_{i+1}^{HCI}, t_i) \tag{7.11}$$

In all,

$$\Delta d(t_i) = \Delta d_{HCI}(T_i, S_i^{HCI}, t_i) + \Delta d_{BTI}(T_i, S_i^{BTI}, t_i) \tag{7.12}$$

Since BTI is the dominant aging mechanism and almost is three times higher than HCI [18, 66, 109], therefore we define S_i as:

$$S_i = m_1 \times S_i^{BTI} + m_2 \times S_i^{HCI} \tag{7.13}$$

where m_1 is three times greater than m_2. From Eq. 12 and Eq. 13:

$$\Delta d(t_i) = \Delta d(T_i, S_i, t_i) \tag{7.14}$$

Eq. 14 concludes that aging is a function of stress and temperature. By finding these two characteristics of a router till time t_i, aging rate can be predicted. Designers assign aging guardband on critical paths by predicting worst case scenario which impose performance, area and power overhead to the system [66, 40, 149]. Guardbands on critical paths also are added due to process variation, voltage droop, and temperature. The focus of this work is to increases lifetime of the NoC by preserving *aging guardband* after chip fabrication and during runtime.

7.3 Problem formulation

The objective of AROMa is to find a set of source-destination paths that satisfies the performance requirement while minimizing the maximum aged router's age and balancing age across all routers in 3D NoCs. Given the set of source-destination pairs $P = \{P_{0,1}, P_{0,2}, ..., P_{0,n-1}, P_{1,0}, ..., P_{n-2,n-1}\}$, the list of routers $R = \{r_0, r_1, ..., r_{n-1}\}$, the list of routers' ages $RAge = \{RAg_0, RAg_1, ..., RAg_{n-1}\}$ at time t, the list of *number-of-flits* (fl) and their *residence-times* (rs) pairs $FLRS = \{(fl_0, rs_0), (fl_1, rs_1), ..., (fl_{n-1}, rs_{n-1})\}$ for each router $r_i \in R$ at time t, the objective is:

$$\begin{aligned} &\text{MinMax} && \text{RAg}(r_i), \forall r_i \in R, \\ &\text{subject to} && RAg(r_i) < RAg_{GB}, \forall r_i \in R, \\ & && SP(P_{i,j}, RAge, t) \in \{KSP_{i,j}\}, \forall P_{i,j} \in P, \end{aligned} \tag{7.15}$$

where, RAg_{GB} is the critical path aging guardband, $SP(P_{i,j}, RAge, t)$ is our proposed function to find shortest paths $P_{i,j}$ among $\{KSP_{i,j}\}$ for each pair of i and j considering the routers ages (i.e. $\{RAge\}$) along it at time t, and $\{KSP_{i,j}\}$ is the list of K-best shortest paths between each pair of i and j.

7.4 Online aging monitoring in 3D NoC

We elaborated in Section 4 and concluded in Eq. 12 that BTI and HCI are functions of temperature and stress. In addition, the only stimuli in a router, as a system, is flits. Hence, temperature and stress are functions of flits. From system point of view flits characteristics in a router are the *number-of-flits* (fl) and the amount of time that they reside inside router, namely *residence-time* (rs), for a given period of time, epsilon (ϵ). These two parameters impact the amount of stress, power, and temperature of a router. For instance, if the number of flits is fl_j and their total residence time is rs_k for a given period of time i, the temperature T_i will be:

$$T_i = T(fl_j, rs_k) \tag{7.16}$$

Similarly, stress S_i for time period i is a function of fl_j and rs_k as follow:

$$S_i = S(fl_j, rs_k) \tag{7.17}$$

Consequently, aging rate also is determined by these two parameters. Eq. 14 can be rewritten as:

$$\Delta d(t_i) = \Delta d(fl_j, rs_k, t_i) \tag{7.18}$$

fl or rs is a range of numbers not certain numbers. If either fl or rs change, stress, temperature and aging rate will change as well. Additionally, router's capacity of flits is limited in a given period of time, which means that maximum number of flits and their residence time cannot exceed a certain amount. Based on the NoC characteristics such as flit injection rate and topology, the maximum number of flits as well as their maximum residence time in a predetermined ϵ is bounded by FL_{max} and RS_{max}, respectively. In all, by monitoring fl and rs, they can be exploited to map their corresponding temperature and stress of a specific router to predict its age.

Additionally, in 3D NoC, it is essential to distinguish between layers in terms of temperature because distinct layers have different temperature maps. Usually, the bottom layer that is next to the heat sink experiences lower range of temperature than the upper layers assuming homogeneous NoC nodes. in other words, the farther distance from the heat sink the higher temperature range. Fig. 7.7 illustrates that routers with same position in each layer have different temperatures even though their stress values are equal. For instance, the average temperature difference between different layers is approximately 2 Kelvin [37]. Consequently, same fl and rs pair values correspond to different temperatures for each layer.

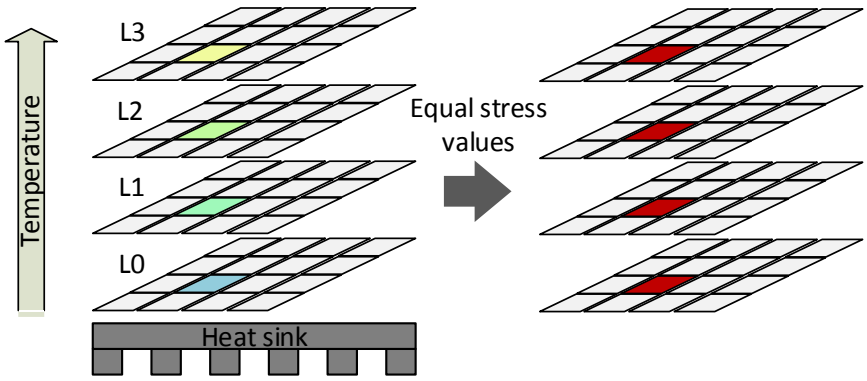


Figure 7.7: Routers in different layers of 3D NoCs with equal amount of stresses have different temperatures.

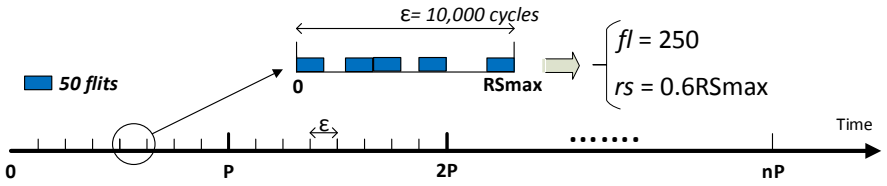


Figure 7.8: Age monitoring at each period of P at each layer L_i.

For example, in Fig. 7.8, it is illustrated that aging is monitored at each period of time P, which is divided to smaller equal periods of ϵ. Therefore, a period P is equal to $n \times \epsilon$. Assuming $RS_{max} = \epsilon$ equals to $10,000$ cycles, fl is equal to 250 and rs is equal to 0.6 of maximum residence time (RS_{max}), which is 6,000 cycles. Their pair of fl and rs, namely 250 and 6000, corresponds to a specific stress value but different temperatures for each layer in 3D NoC (Fig. 7.7). Therefore, same pair of fl and rs corresponds to same aging rate for different routers in a layer but different aging rates for routers in different layers. For example, in layer L_2, stress is equal to H similar to other layers but temperature is equal to C and the corresponding aging is w.

Fig. 7.9 illustrates our proposed architecture in AROMa for monitoring fl and rs. This monitoring system is embedded into each router architecture [1]. Each core i is connected to a router R_i. The upper counter which is a 12-bit counter [144] counts fl for each ϵ. It monitors valid incoming flits to the router from different ports to the router using valid (v) and ready (r) signals. Therefore, whenever a flit enters a router these two signals will be active and the counter can count the number of incoming flits and find fl. This counter will be reseted to zero at each ϵ (i.e. when the timer reaches ϵ).

The other parallel counter depicted in the lower section of Fig. 7.9, is responsible for counting the number of cycles at which flits are residing inside a router. Basically this counter is a timer which computes residence time, rs, of flits for each router during each ϵ. As regards to that RS_{max} or ϵ, it can be represented by 14 bits if $\epsilon = 10,000$ cycles. The counter is

preceded by 14-bit subtractors. Each subtractor subtracts the exit time (Ex) of an outgoing flit, which is the current cycle when the flit exiting the router, from the en-queue time (Eq), which is saved inside the flit when it enters the router. If we assume the maximum rs of a flit inside a 5-stage router is 15 cycles, then a 4-bit MUX connected to the output of each subtractor, in order to drop any possible negative subtractions in the boundaries of each 10,000 cycles. After that, it is fed to the parallel counter to keep accumulating residence time (rs) of all flits exiting the router through all possible seven output ports (five ports in 2D architecture). Moreover, these two counters are reseted after ϵ cycles. We use a timer to count ϵ and whenever it reaches to ϵ a reset signal is sent to the two parallel counters inside each router to be ready for next ϵ. The number of control signals for each counter depends on the position of the router inside the network. For instance, for a centered router in the middle layers we have 7 ports with its corresponding control signals whereas for a router in the corner of the upper layer it has only 4 ports with less control signals.

To minimize the distance between D-CATs and all routers, they must be located in one of the middle routers in each layer. For example, as illustrated in Fig. 7.9 D-CAT3 for layer three resides in core 53, similarly, D-CAT2 for layer two resides in core 42. For the other two bottom layers also their D-CATs can be reside in the corresponding cores as upper layers. The timer, which counts ϵ can be located in one of the middle layers to reset the counters inside all routers when it is required.

Based on Eq. 18, D-CAT in each layer will be accessed using (fl, rs) pair from all routers of that specific layer to read back their age degradation in each ϵ. Therefore, the 26 bits data (14-bit rs and 12-bit fl) is decoded to access corresponding entry in D-CAT. Age degradation of a router can be computed for each temperature and stress (Eq. 10 and Eq. 11). To this end, we determine conditions that may happen to a router. Each condition, $C_{i,j}$, is represented by its respective rs_i and fl_j. Each pair of (rs_i, fl_j) corresponds to temperature $T_{i,j}$ and stress $S_{i,j}$ (i.e. $(T_{i,j}, S_{i,j})$). Hence, each condition is a function of rs_i

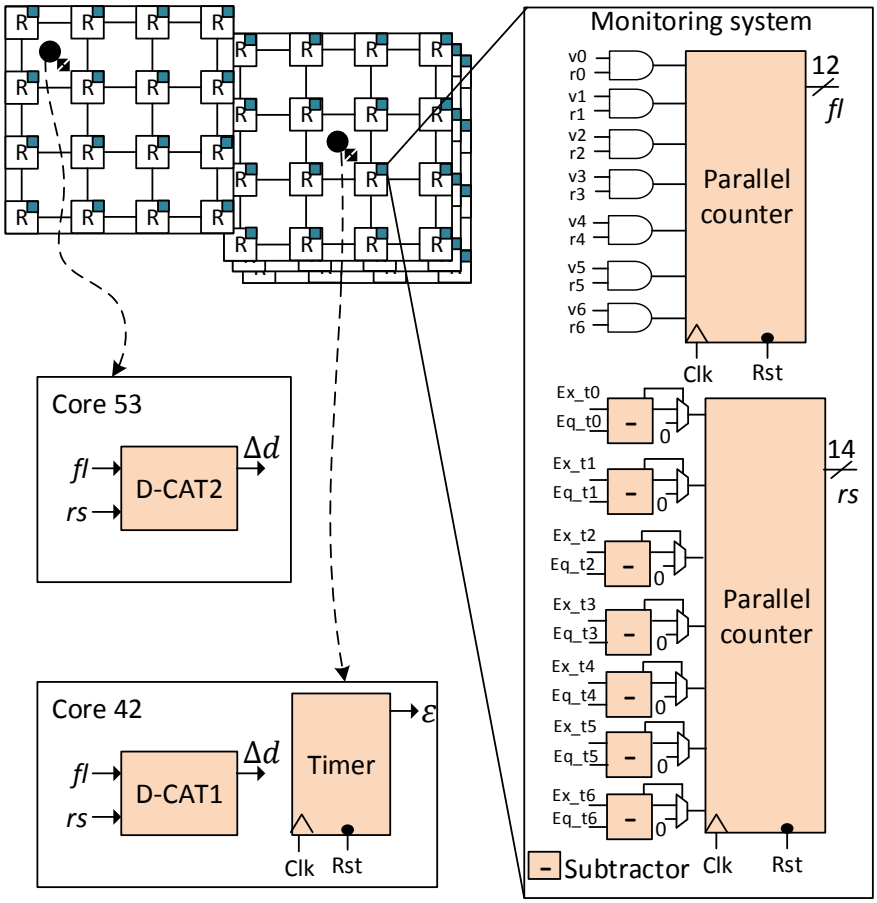


Figure 7.9: AROMa online aging monitoring architecture. Other cores (black circles) and two D-CATs are not shown for clarity).

and fl_j and each condition corresponds to a specific aging rate. For example, in Fig. 7.10 for layer L_0, when number of flits is fl_j and they reside inside the router for queuing, processing and traversal through the router for rs_i cycles out of ϵ cycles, the delay degradation is $\Delta d^0_{i,j}$. Same number of flits fl_j and residence time rs_i in layer L_k leads to $\Delta d^k_{i,j}$ delay degradation. This is because the temperature varies at different layer as discussed earlier (Fig. 7.7). It must be noted that when the router is not busy (fl and rs are equal to zero) and BTI recovery phase happens, D-CATs returns a negative corresponding amount of recovery as stated in their first entry.

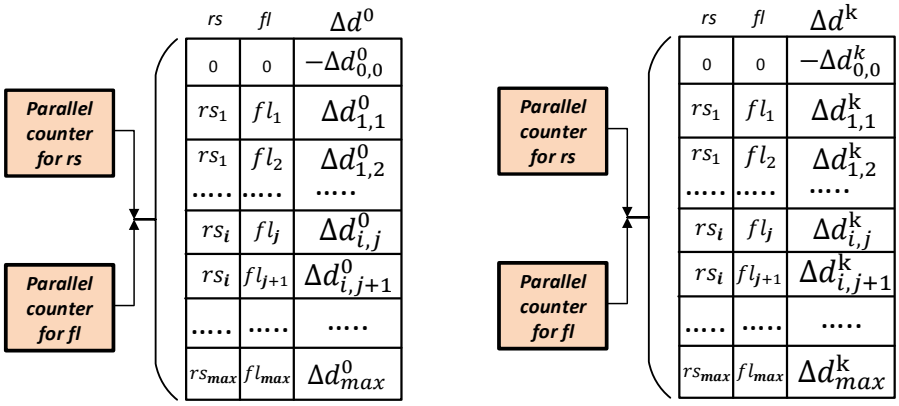


Figure 7.10: D-CATs for layer L_0 and layer L_k.

7.4.1 D-CAT construction algorithm

The pseudo code for constructing D-CATs is shown in Algorithm 9. The inputs to this algorithm are maximum residence time RS_{max} (or the updating time period (ϵ)), the steps for each residence time rs_{steps}, the number of steps for counting flits inside the router fl_{steps}, the injection rate to the system $Ijrate$, the number of layers in the 3D NoC L, and the router's floorplan $RFLP$. The algorithm's output is the list of D-CATs which are found for each layer in the 3D NoC and can be accessed from each router of its corresponding layer to read back its own age based on fl and rs during each ϵ.

At the beginning, the maximum number of flits (FL_{max}) that can occupy a router during RS_{max} (or ϵ) considering maximum $Ijrate$ is extracted (line 1). After that, the list of residence time (rs) and number of flits (fl) will be quantized based on their number of steps (line 2, 3). As we discussed, each layer requires different D-CATs. Therefore, in a loop for each layer l_k, each different residence time rs_i and number of flits fl_j, we have different power maps ($P^k_{i,j}$). We calculate power consumption using Mcpat [98] (line 7) for each pair of rs_i, fl_j values. Consequently, different temperature maps ($T^k_{i,j}$) can be extracted for each layer k using the HotSpot tool [137]. HotSpot takes the corresponding power consumption for rs_i, fl_j, and routers' floorplan $RFLP$ as inputs (line 8). Similarly, the stress will be extract as

Algorithm 9 D-CAT Construction

Input: Maximum resident time RS_{max}, number of resident time steps rs_{steps}, number of flits steps fl_{steps}, injection rate $Ijrate$, number of layers L, Router floorplan $RFLP$

Output: List of {D-CAT}

1: $FL_{max} \leftarrow FindMaxFlit(Ijrate, RS_{max})$;
2: $\{rs\} \leftarrow CreateRsList(RS_{max}, rs_{steps})$;
3: $\{fl\} \leftarrow CreateFlList(FL_{max}, fl_{steps})$;
4: **for all** $l_k \in L$ **do**
5: **for all** $rs_i \in rs$ **do**
6: **for all** $fl_j \in fl$ **do**
7: $P^k_{(i,j)} \leftarrow CalPower(rs_i, fl_j)$;
8: $T^k_{(i,j)} \leftarrow CalTempreture(P^k_{(i,j)}, RFLP)$;
9: $S_{(i,j)} \leftarrow CalStress(rs_i, fl_j)$;
10: $\Delta d^k_{(i,j)} \leftarrow CalDelayDeg(T^k_{(i,j)}, S_{(i,j)})$;
11: D-CAT$^k \leftarrow FillCAT(rs_i, fl_j, \Delta d^k_{(i,j)})$;
12: D-CAT$list.Add$(D-CAT^k);
13: **end for**
14: **end for**
15: **end for**
16: **Return {D-CAT};**

$S_{i,j}$ based on HCI and BTI aging mechanism (line 9). As shown in Eq. 3 and Eq. 6, Stress (S) is a function of duty cycle (Y) in BTI and switching activity (α) multiplied by clock frequency (f) in HCI. In this work, Eq. 13 is utilized to calculate S. For BTI mechanism, Y is equal to the residence time rs and for HCI mechanism α is equal to the ratio between fl and FL_{max}. The delay degradation is extracted for each temperature and stress pair using Eq. 14 (line 10). After that, the D-CAT for each layer k will be filled for each pair of rs_i and fl_j by $\Delta d^k_{i,j}$ using Eq. 18 (line 11). At the end, we add the D-CAT for layer k into the D-CAT list and iterate for other layers in the network till all of them in 3D NoC are covered.

7.5 Adaptive aging-aware routing

Imbalanced aging between routers in a network can lead to performance loss or timing failure in the system. If a highly aged router fails, it impacts the scalability and reliability of the

whole system. However, there are different shortest paths between each pair for source and destination in a network as graph. In addition, there are alternative paths with costs which are very close to the shortest paths in term of delay. These paths use different routers to transfer flits inside the network, which means a router can be along different source-destination pairs' shortest paths.

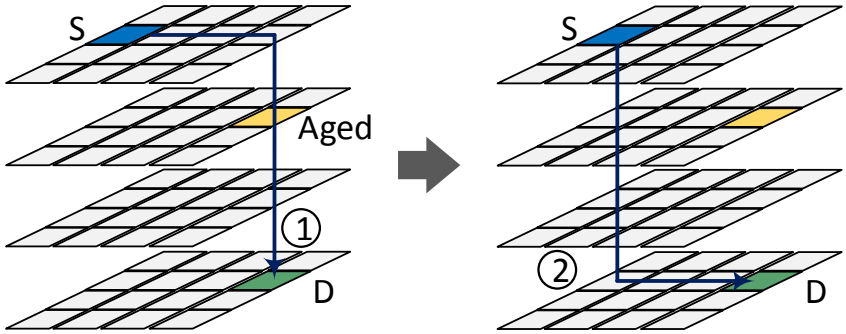


Figure 7.11: Swapping between different shortest paths that have different routers' ages.

As illustrated in Fig. 7.11, two different shortest paths from source S to destination D are chosen among a set of k-best shortest paths. Each of them uses five different routers and there is no shared router between these two paths. Whereas in scenario number 1, one of the routers along the shortest paths is aged more than the others and become the maximum aged router. In this case, we can easily switch the shortest path to scenario number 2 without losing performance while avoiding needless increase to the age of maximum aged router and give it an opportunity to recover (i.e. in BTI). In this example there are ten different shortest paths for the specified source-destination pairs that we can choose from for the purpose of aging mitigation.

For more elaboration, we show how routing happens as it is depicted in Fig. 7.12, where a $4 \times 4 \times 4$ 3D NoC is demonstrated with source router 0 sending flits to destination router 63. Every router maintains a routing table with size of $O(N)$ where N represent the number of routers, as it is shown in the figure for router 47. The first column represent the destination

(DS) and the second one correspond to the next router (NR). For the other routers, only the tuple where $DS = 63$ is shown. The left side of the Fig. 7.12 represent the NoC at period iP. The flits follow the path defined by $0 \rightarrow 16 \rightarrow 32 \rightarrow 48 \rightarrow 52 \rightarrow 56 \rightarrow 60 \rightarrow 61 \rightarrow 62 \rightarrow 63$. After period iP is consumed and assuming router 47 and 52 are aged more that the other routers, they should be avoided in the next period. Therefore, at period $(i+1)P$, the routing tables of the routers are adjusted to new values to avoid the aged routers. Thus, at period $(i+1)P$ as shown in Fig. 7.12 right side, the new path for the flits will be $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 11 \rightarrow 15 \rightarrow 31 \rightarrow 47 \rightarrow 63$.

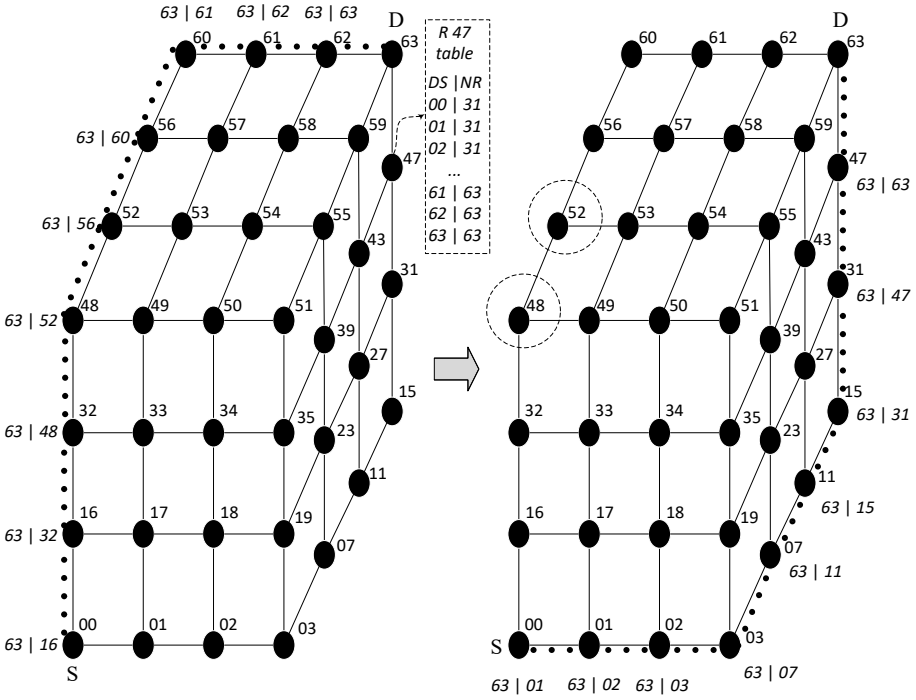


Figure 7.12: Updating routers' routing tables' entries for the new shortest path from S to D considering high aging in routers number 48 and 52. Routing table is detailed for router number 47.

Algorithm 10 Aging aware routing algorithm

Input: Src-Dest pair list $\{(Src, Dest)\}$, Router's age list $\{RAg\}$
Output: List of shortest paths $\{ShortPathPair\}$

```
1: ShortPathPair ={};
2: for all Pair_i ∈ {(Src, Dest)} do
3:    k_ShortPath{} ← CalShortestPath(Pair_i);
4: end for
5: for all Pair_i ∈ {(Src, Dest)} do
6:    for all Path_j ∈ k_ShortPath_i do
7:       if (!MaxAgeR(Path_j, {RAg}) ∧
   MinAge(Path_j, {RAg})) then
8:          ShortPathPair.Add(Path_j);
9:       end if
10:   end for
11: end for
12: Return ShortPathPair;
```

7.5.1 Adaptive aging-aware routing algorithm in AROMa

Since, aging is a gradual and slow mechanism we can swap between different shortest paths in each period of time P (e.g. each week) to avoid highly aged routers in the network. The pseudo code in Algorithm 10 proposes an aging-aware routing algorithm. To this end, we add a tag to each router as its age. This tag will be updated online using their corresponding D-CATs, periodically ($P = n \times \epsilon$). The aging tag is leveraged for choosing best aging aware shortest path between all available k-best shortest paths from each source-destination pairs. When the new aging-aware shortest path is chosen among all k-best shortest path routing table in each router will be updated adaptively at each period of time P (Fig. 12).

The inputs to Algorithm 10 are list of source-destination pairs, $\{(Src, Dest)\}$ and list of routers' age, $\{RAg\}$. The algorithm's output is the list of shortest paths for each source-destination pairs, $\{ShortPathPair\}$. The routing table of each router will be updated based on new shortest paths output from Algorithm 1. Using $CalShortestPath()$ function, we find k best shortest paths list for each pair of source-destination. Dijkstra's shortest path algorithm is leveraged to find this list. There are different algorithms that can be utilized

for this purpose [53, 5]. After that, for each pair we check which paths do not include the maximum aged router by calling $MaxAgeR()$ function and then find the best paths based on minimum summation of ages on their routers using the list of ages by calling $MinAge()$ function (line 7). The new shortest paths for each source-destination pairs are found for the next P and are added to the list of shortest paths (line 8). At the end of each period, the shortest paths are updated between all the pairs in the network using the proposed algorithm.

The age of each router is obtained from their corresponding D-CATs. After that, the list of routers' ages $\{RAg\}$ will be updated. As exemplified in Fig. 12, the routing table are updated to swap to the new aging-aware shortest path, adaptively. In all, we intensified Dijkstra's algorithm to find k-best shortest paths by adding routers ages to it as cost. As formulated in Section 4, in Eq. 15, our goal is to minimize the age of maximum aged router and balance the aging among different routers in the network. It must be noted that when the highly aged router is avoided, its links also are avoided which means our algorithm inherently minimizes the age of links, as well.

7.6 Related work

The routing algorithm's function is to forward the flit that arrives to an input port of a router to one of its output ports. There are numerous routing algorithms for 2D [26, 113] and 3D NoCs [97], each one leads to different performance and cost. Routing algorithms can have three major criteria: decision location (source or distributed routing), the path length (minimal or non-minimal routing), and path definition (deterministic or adaptive routing). In source routing, the complete path is decided at the router connected to the source [29], while in distributed routing each router receives, stores and then defines the direction of a flit [125]. Therefore, source routing requires full knowledge about the network, which results

in area and traffic overheads in routing tables and network. Contrary, routing decision in distributed routing is spread among routers which impose less area and power overheads. In minimal routing the shortest path from source to destination is greedily chosen, e.g. *Breadth First Search* (BFS), Dijkstra, and Floyd-Warshall algorithms [43]. In contrast, non-minimal routing algorithms allow flits to traverse longer source-destination pair distance to meet other network objectives, e.g. aging or congestion avoidance [49, 146].

In deterministic routing, the path is completely specified based on the position of source and destination offline, e.g. XY routing [26]. In adaptive routing, the path is a function of online network variations, e.g. traffic in turn model [68]. Deterministic routing is easy to implement and impose less area overhead while flits are blindly follow the same path without considering the path congestion or age. In adaptive routing, we can enhance the routing to be more intelligent to choose between different shortest-paths and guarantee certain objectives. Plethora of researches are conducted that target the usage of adaptive routing for different objectives such as performance, fault tolerance, congestion avoidances, load balancing and aging mitigation. We review different proposed adaptive routing algorithms in the following.

7.6.1 Congestion aware adaptive routings

Table based adaptive routing is among the method to ensure high degree of adaptivity for better performance [103, 120, 146, 49, 72]. In [120], authors propose a compression technique using graph coloring to shrink the large routing table size. [103] proposes a routing algorithm based on partitioning routers into different regions that could be accessed only through certain routers. HARAQ utilizes Q-learning method to provide alternative paths between each source-destination pairs using Q-tables in each router for local and global congestion information [49]. Additionally, [72] proposes *Region Congestion Awareness* (RCA) technique to improve global network balance using a monitoring network to estimate congestion. The

monitoring network added circuitry to each router to aggregate and propagate congestion information to other routers. A channel pressure model is adopted in [146] to quantify and predicts traffic in order to implement an offline method for designing deadlock free adaptive minimal routing to address local congestion. In all, considering only local congestion information may not choose less congested paths and lead to congestions in other parts of the NoC. However, if global information is considered area and power overheads are imposed due to larger routing tables. In addition, congestion information broadcasting imposes traffic overhead.

7.6.2 Fault tolerant adaptive routings

Adaptive routing algorithms have been implemented for reliability and fault tolerance purposes in NoC [56, 93, 131]. In [56], a routing algorithm for 2D mesh and torus topologies reconfigures the routing table of each router to avoid faulty components. However, they need a fixed amount of hardware per router to achieve that. In addition, [93] proposes an adaptive routing which forces the traffic to be distributed across the whole network. The algorithm distributes traffic uniformly to avoid overloading the links and faulty routers in case of failure. Furthermore, [131] aims to provide connectivity of 2D mesh NoCs even after some network components are out of service by deflective routing using a router design based on nostrum architecture. They use fine grain functional fault model and a methodology to diagnose and determine routers' status using *Cyclic Redundancy Checks* (CRC) hardware components. Also, the adaptive routing algorithm can employ the remaining functionality of partially defective routers. They support graceful degradation by retransmitting messages suffering from transient faults using *Error Correcting Codes* (ECC). In all, these techniques require additional hardware to the router as well as overhead bits (checksum) in each flit to implement EEC/CRC.

7.6.3 Aging-aware adaptive routings and motivation

Adaptive routing can be used for aging mitigation and overcoming delay degradation effects in routers [10, 19, 20, 11, 152]. The authors in [10] propose an aging aware adaptive routing algorithm and router micro-architecture to route flits in paths that are experiencing minimum aging degradation. The routing algorithm have a shortest path selection stage and a recovery cycles insertion stage in overloaded routers. They used a circuitry through a series of delay buffers to measure a router delay degradation, which imposes considerable hardware overhead for that purpose.

Authors in [19, 20] proposed offline methods to avoid highly aged routers in 2D NoC. Deploying *Mixed Integer Linear Programming* (MILP) based on power-performance the optimized routing is obtained [19]. This technique assigns a budget to each router offline by profiling the benchmark traces. A routing algorithm finds the source-destination pairs' shortest paths considering router's budgets to mitigate aging. A similar approach is proposed in [20], where budgets are assigned offline for different epochs of time. These methods not only limits the usage of routers at runtime which can impact the system performance but also still leaves the unbalanced aging among the routers by assigning unbalanced budgets. Another important shortcoming is that aging strictly influenced by runtime variation in the workload that affects the stress and temperature on routers. While profiling and budgeting are done based on some specific benchmarks.

Exploiting architecture level criticality of flits, [11] presents a routing policy for 2D NoC with heterogeneous routers (i.e. routers are either buffered or buffer-less). Utilizing their *Wearout Monitoring System* (WMS), this method monitors aging in routers to deflect non-critical flits. Beside the large area overhead due to the complex WMS, deflecting flits degrades *Quality of Service* (QoS) at system level. Furthermore, flits' criticality designation not only is a challenging issue but also does impose overhead to the system. The performance of

this technique will be questioned in regular NoC. Furthermore, a dynamic programming based aging-aware routing is proposed in [152], which employs lifetime budget computation unit. This technique requires a parallel dynamic programming network to propagate routers' lifetime budgets and a complex circuitry for their calculations, which impose significant overhead to the system. All the proposed methods are either offline or impose large overhead to the system. Besides, the proposed methods are considered in 2D NoC, that can be modified for 3D NoC, but new challenges in 3D NoC such as higher temperature can impact them. In this work, we fill these gaps based on our low overhead online monitoring system that can capture workload behavior and update the routing. Authors in [58, 12] proposed aging-aware router architectures which out of scope of this work.

7.7 Experimental evaluation

In this section we evaluate our methodology. First, our simulation environment setup is explained. After that, we describe our results for AROMa in 2D and 3D NoC as compared to non aging-aware (NAW) technique and state-of-the-art work. Finally, we analyze AROMa from different aspects.

7.7.1 Setup

All of our simulations are done in the full system simulation mode using gem5 [24] that runs on Linux operating system to support scheduling benches for three years (9.3E+7 seconds) of execution time. In addition, we adopt a ruby memory model with 2D and 3D mesh interconnect network. Also, Garnet [3] network model is used with 5-stage routers that is embedded inside gem5. In order to extract power estimation results for these stages, we used Mcpat [98] for different ranges of fl and rs. HotSpot [137] is used to extract temperature

Table 7.1: Simulation platform configuration

item	**Description**
Processor	X86 based 1.0 GHz in order cores.
L1-iCache	private, 32KB, 2-way set associative,
	64B blocks, 4 cycles latency, pseudo LRU replacement.
L1-dCache	private, 32KB, 2-way set associative,
	64B blocks, 4 cycles latency, pseudo LRU replacement.
L2-Cache	private, 16MB, 8-way set associative,
	64B blocks, 12 cycles latency.
Main Memory	512MB. DRAM
NoC	4×8 2D mesh, $4 \times 4 \times 2$ 3D mesh,
	each node consists of 1 router,
	1 core, 1 private L1 i/dcache, and 1 private L2 cache.
	MOESI cache coherence protocol, 5-stage pipeline router.
Flit size	16B
Buffer size	$4 \times 16B$ or 4 flits per virtual channel.

maps of a router for different extracted powers. To get the router's floorplan for temperature analysis, the architecture in [1] is used. The floorplan is extracted for 45nm technology using Cadence tool chain.

SPLASH-2 and PARSEC benchmarks are adopted for our experiments. Each experiment run with 32 cores interconnected via 4×8 2D mesh or $4 \times 4 \times 4$ 3D topology. All routers accept 16-byte flit sizes and assume a virtual channel architecture that has four virtual channels which holds four flits. Each 2D router in the system has five input ports for (N, E, S, W and local). 3D routers have seven input ports for (N, E, S, W and local). The local ports are

Table 7.2: Aging-induced delay degradation for maximum aged router and network age imbalance (Δ) in 2D NoC for 3 years of execution (9.3E+7 seconds).

	NAW		OFAR				AROMa			
Benchmark	MAX(NS)	Δ(NS)	MAX(NS)	Δ(NS)	MAX(%)	Δ(%)	MAX(NS)	Δ(NS)	MAX(%)	Δ(%)
FFT	0.101873	0.0603584	0.074661	0.036366	26.71	39.75	0.067989	0.020266	33.26	66.4
CANNEAL	0.098927	0.057665	0.070327	0.032300	28.91	43.99	0.066582	0.024100	32.67	58.21
LU_NON_CON	0.152819	0.152819	0.172457	0.172457	-12.85	-12.85	0.091096	0.076563	40.39	49.90
RADIX	0.144387	0.063254	0.395172	0.365172	-173.69	-477.31	0.098993	0.013055	31.43	79.36
SWAPTIONS	0.101935	0.060667	0.071693	0.033939	29.67	44.06	0.070238	0.026983	31.09	55.52
X264	0.109525	0.066149	0.080548	0.046149	26.46	30.23	0.072390	0.027666	33.90	58.17
BLACKSCHOLES	0.097498	0.056513	0.069047	0.031285	29.18	44.64	0.061431	0.019239	36.99	65.96
CHOLESKY	0.199071	0.147790	0.349684	0.349683	-75.66	-136.61	0.132243	0.054992	33.57	62.79
LU_CON	0.194385	0.138543	0.291253	0.291218	-49.83	-110.20	0.139464	0.061535	28.258	55.58
AMEAN	0.133380	0.089306	0.174982	0.150952	-31.19	-69.03	0.088936	0.036044	**33.51**	**61.32**
GMEAN	0.128726	0.082123	0.138207	0.094319	N/A	N/A	0.085516	0.031132	**33.37**	**60.87**

connected to one core with one L1 instruction cache, one L1 data cache, and one private L2 cache with sizes of 32kB, 32kB, and 16M, respectively. Since the clock frequency is equal to 1GHz (critical path will be 1 ns) and the guadrband is chosen to be 16% (0.16 ns) for 3-year worst case execution. Worst case happens when the temperature is 380K and the transistor is always ON. The rest of the simulation setup is listed in Table 7.1.

In modeling stage, $RS_{max} = \epsilon$ is assumed to be $10,000$ cycles which can be counted using a 14-bit parallel counter (Fig. 9 & Fig. 10). To get the maximum number of flits, FL_{max}, we use a representative synthetic traffic patterns with flit injection rate equals to 0.05 flits/cycle for ϵ (or RS_{max}), as depicted in Fig. 7.8. we found that FL_{max} cannot exceed $2,400$ which can be counted by 12-bit counter. This injection rate is chosen based on the maximum possible traffic.

7.7.2 Results

For each benchmark, the aging-induced delay degradation of routers as well as the age imbalance (Δd) between routers are extracted for both 2D and 3D NoC in 3 years (9.3E+7 seconds) of execution. The following three schemes are compared:

- XY routing, which is not-adaptive and therefore non-aging aware routing (NAW). This method is intensified by our monitoring system using D-CAT to extract routers' ages online. For 3D case, we used XYZ routing.
- Offline aging-aware routing through assigning budgets (OFAR), which is based on state-of-the-art works in [19, 20] and enhanced by our proposed online aging monitoring system for fairness. It assigns a lifetime budget for each router and defined as the fraction of the traffic that a stressed router should accept. Each routers' budget is predetermined using profiling for each benchmark.

Table 7.3: Aging-induced delay degradation for maximum aged router and network age imbalance (Δ) in 3D NoC for 3 years of execution (9.3E+7 seconds).

	NAW		OFAR				AROMa			
Benchmark	MAX(NS)	Δ(NS)	MAX(NS)	Δ(NS)	MAX(%)	Δ(%)	MAX(NS)	Δ(NS)	MAX(%)	Δ(%)
FFT	0.088736	0.049433	0.072167	0.038652	18.67	21.81	0.062064	0.018803	30.06	61.96
CANNEAL	0.092792	0.054076	0.071092	0.036751	23.39	32.04	0.059354	0.016520	36.04	69.45
LU_NON_CON	0.155106	0.155106	0.265872	0.265872	-71.41	-71.41	0.112998	0.095032	27.15	38.73
RADIX	0.128938	0.055948	0.486709	0.486709	-277.48	-769.94	0.088988	0.010294	30.98	81.60
SWAPTIONS	0.087378	0.047744	0.071672	0.037109	17.98	22.27	0.055814	0.011844	36.12	75.19
X264	0.097211	0.056439	0.077891	0.043849	19.87	22.31	0.062816	0.017808	35.38	68.45
BLACKSCHOLES	0.090148	0.051525	0.069848	0.035965	22.52	30.20	0.056309	0.013341	37.54	74.11
CHOLESKY	0.181058	0.137805	0.364778	0.364778	-101.47	-164.71	0.111719	0.017490	38.30	87.31
LU_CON	0.124743	0.088748	0.269684	0.269684	-116.19	-203.88	0.078841	0.009424	36.80	89.38
AMEAN	0.116234	0.077425	0.194413	0.175485	-51.57	-120.15	0.076545	0.023395	**34.26**	**71.80**
GMEAN	0.112332	0.069812	0.143291	0.100631	N/A	N/A	0.073733	0.017329	**34.05**	**70.02**

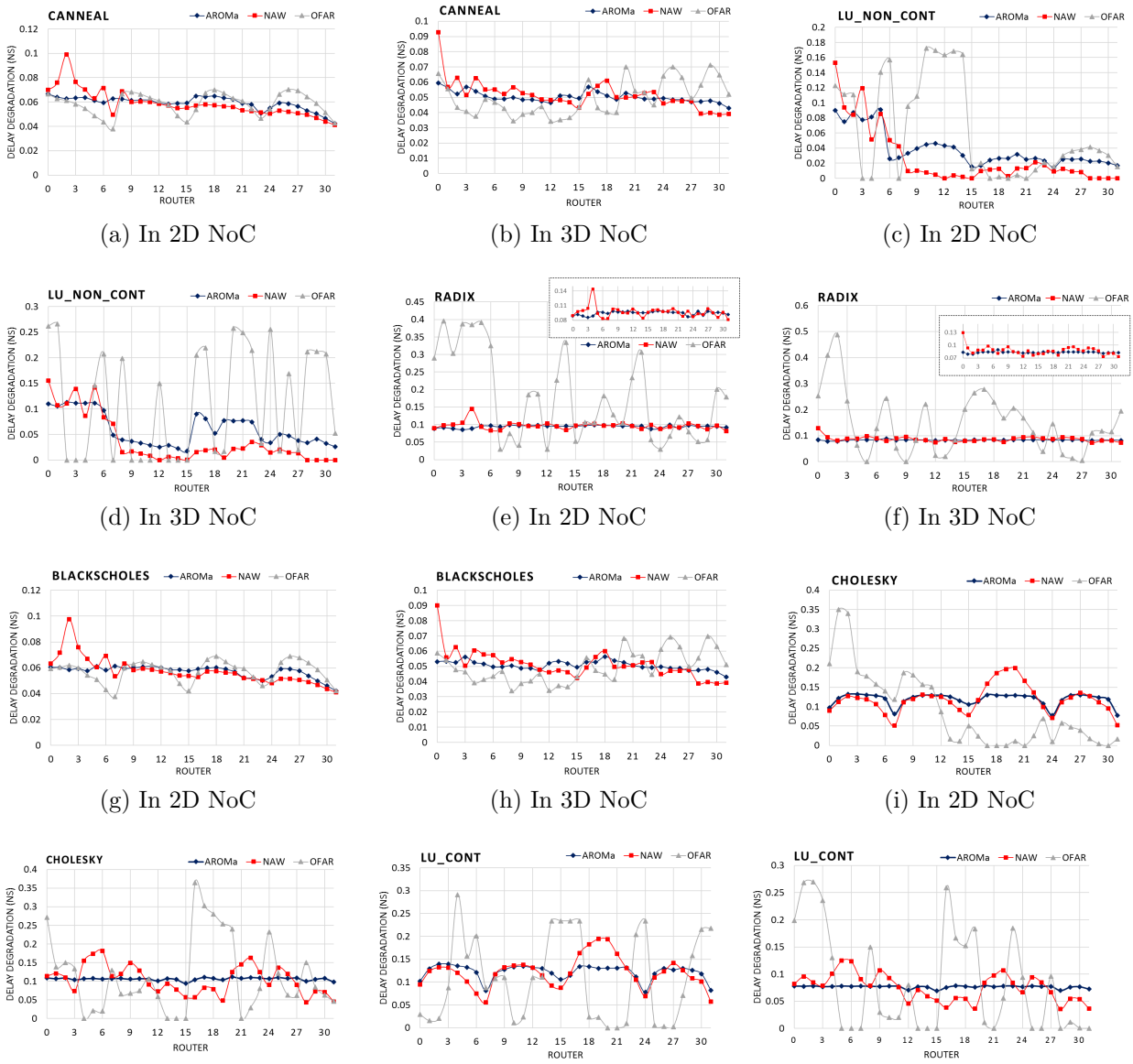
- AROMa, our aging-aware adaptive routing and proposed online aging monitoring system.

Table 7.2 and Table 7.3 summarize the results for 2D NoC and 3D NoC, respectively. As shown in Table 7.2, AROMa improves the age of maximum-aged router by 33.5% on average compared to non-aging aware routing (NAW). For example, in RADIX the age of maximum aged router in NAW scheme is 0.14487 ns while in our proposed method the maximum-aged router age is 0.098993 ns, which is equal to 31.43% improvement. Moreover, the age imbalance (Δ) between the maximum-aged and minimum-aged routers is improved by 61.31% in AROMa as compared to NAW. This shows how AROMa balances routers' ages fairly.

OFAR scheme assigns budgets to each router based on their load through profiling and then finds new source-destination shortest paths. Therefore, OFAR is not necessarily able to balance age properly and reduce maximum-aged routers age. Our results in Table 7.2 also shows that maximum age and age imbalance (Δ) become worse by 31.19% and 69.03%, on average. For example, in X264 benchmark, the maximum age is improved by 29.67% and age imbalanced is improved by 30.23%. On the other hand, in LU_NON_CONT benchmark, there is no improvement and these values are -12.85% and -12.85%, respectively.

Similarly, Table 7.3 shows the results for 3D NoC. It is shown that on average the maximum age and age imbalance are improved by 34.26% and 71.80%, respectively. For example, in LU_CON benchmark the age of maximum aged router is 0.124743 ns in NAW while in AROMa it is 0.076545 ns, which means 36.80% improvement. In addition, the age imbalance (Δ) in NAW is 0.088748 ns and 0.009424 ns in AROMa, which is equal to a significant improvement of 89.38%.

In OFAR, both of 3D NoC and 2D NoC maximum age and age imbalance (Δ) are worsening by 31.19% and 69.03% as compared to NAW, respectively. Although, maximum age in



(a) In 2D NoC (b) In 3D NoC (c) In 2D NoC

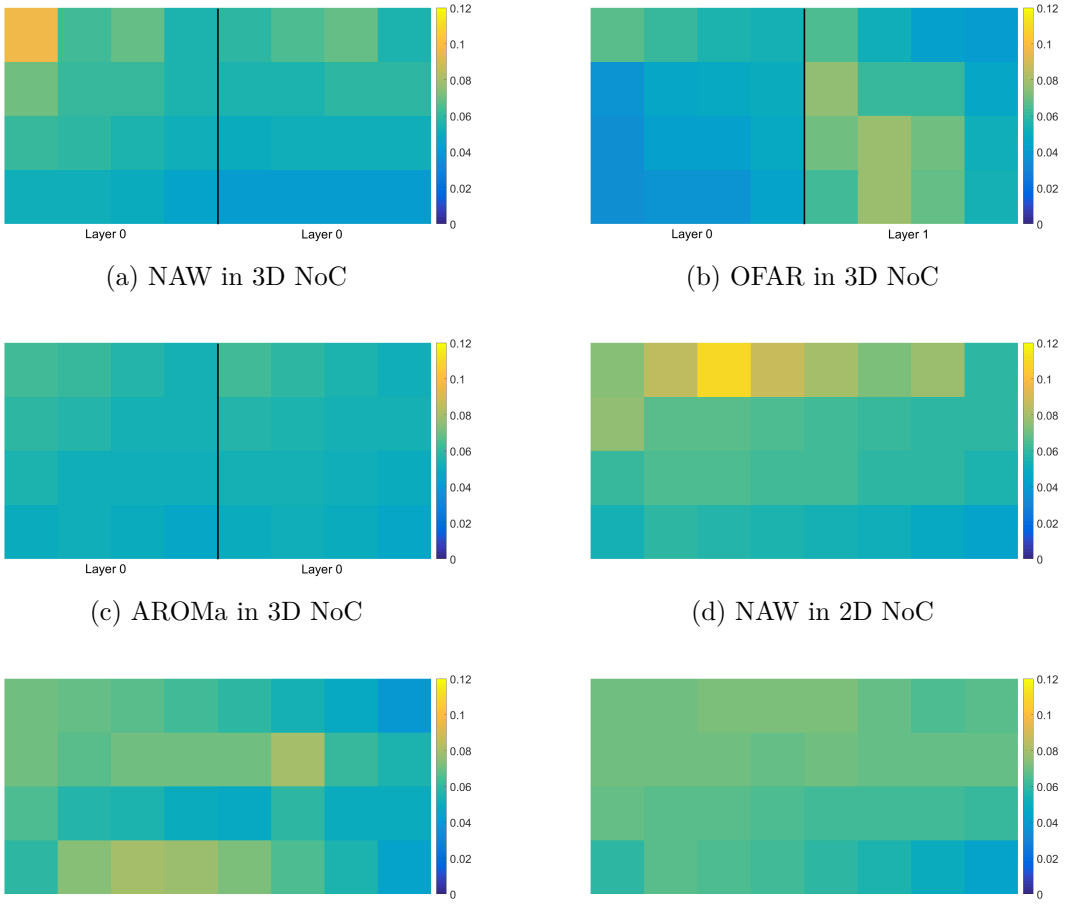
(d) In 3D NoC (e) In 2D NoC (f) In 3D NoC

(g) In 2D NoC (h) In 3D NoC (i) In 2D NoC

(j) In 3D NoC (k) In 2D NoC (l) In 3D NoC

Figure 7.13: Age imbalance in 3 years for different routers in the network.

FFT is improved by 26.71% but it is increased by 75.66% in CHOLESKY. Similarly, age imbalance is improved by 43.99% in CANNEAL while it is worsen by 110.20% in LU_CON benchmark. These results shows that AROMa outperforms state-of-the-art works (OFAR) significantly. The main reason is that AROMa monitor age online and adaptively changes source-destination shortest paths to avoid maximum aged routers. In contrast, OFAR finds shortest paths and assigns age budgets to router offline. Therefore, it cannot balance ages and reduce maximum ages properly. More will be detailed in the next subsection.



(a) NAW in 3D NoC

(b) OFAR in 3D NoC

(c) AROMa in 3D NoC

(d) NAW in 2D NoC

(e) OFAR in 2D NoC

(f) AROMa in 2D NoC

Figure 7.14: Age imbalance between different routers in X264.

7.7.3 Analysis and discussions

AROMa's main purpose is to balance ages between different routers in the network. It is done by avoiding highly aged routers and finding new shortest paths in each period P. This not only reduces the age of maximum aged router (through BTI recovery) but also decreases the difference between different router's ages, so-called age imbalance. Fig. 7.13, illustrates the age imbalance (Δ) for both 3D and 2D NoCs between all the routers in AROMa, NAW, and OFAR. The horizontal axis shows each router number in the network and the vertical axis shows the age (aging-induced delay degradation) of each router in the network. It can be seen that AROMa balance ages properly as compared to NAW and OFAR. For example, in RADIX on 3D NoC all routers ages are around 0.1 ns while in NAW the different routers are aged differently which is not fair. This becomes even worse in OFAR since the age of highly aged routers passed the guardband (0.16 ns). These routers are considered faulty and also pass their assigned offline budgets.

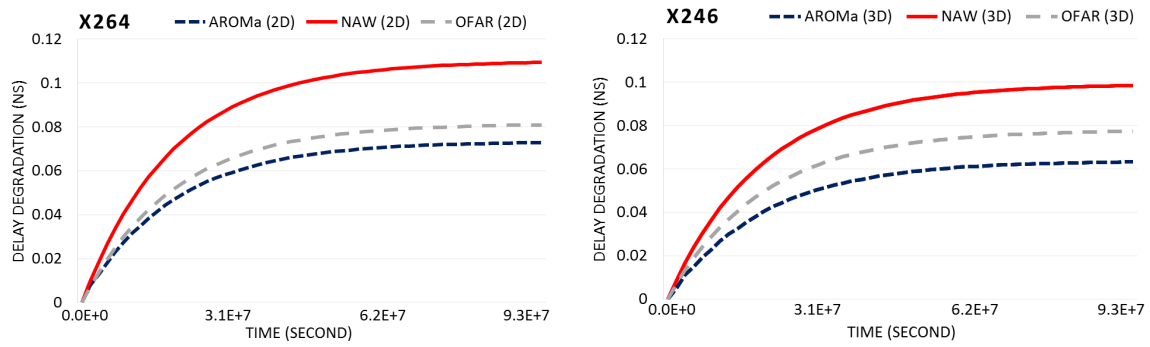
Furthermore, Fig. 7.13 shows that in OFAR scheme some routers for certain benchmarks are overstressed due to offline budgeting. This will cause some of the routers to become faulty and pass the threshold of aging guardband (i.e. 16 ns). The reason is that in the offline profiling stage they were barely utilized. Hence, they are assigned higher budgets to diminish the utilization of highly overloaded routers. This unfair budgeting is not able to balance the age of routers to avoid aging other routers. For example, as shown in Fig. 7.13.j, for CHOLESKY in 3D NoC, seven routers are highly stressed and become faulty. However, some of the routers are rarely used due to unfair budgeting. Nevertheless, we continued the simulation of the network after these routers become faulty, which leads to unpredictable network behavior.

The color map in Fig. 7.14 illustrates the age imbalance of routers in X264, as an example for both 2D and 3D NoC using NAW, OFAR, and AROMa for 3-year execution. Each square

represents a router's age. The lighter color, the higher age. It can be seen that the bottom layer (layer 0) in NAW is aged more than the upper layer (layer 1) (Fig. 7.14.a). Therefore, OFAR assigns more budgets to layer 1 routers to reduce the age in layer 0 but increases the age in layer 1 routers (Fig. 7.14.c). As shown in Fig. 7.14.c, by swapping between different shortest-paths, AROMa balances ages among both layer 0 and layer 1, uniformly. In contrast, routers ages in 2D NoC for both NAW and OFAR schemes are not uniformly distributed (Fig. 7.14.d and Fig. 7.14.e), while AROMa distributes ages between routers in the network properly.

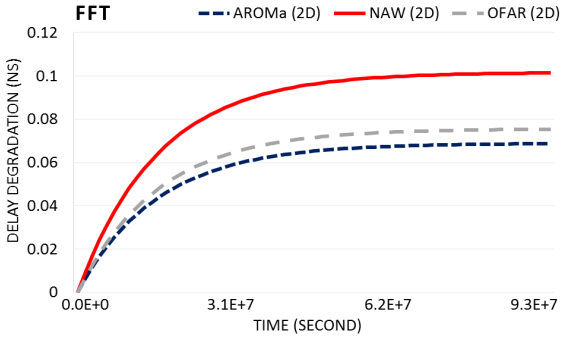
Furthermore, it can be observed from Fig. 7.13 and Fig. 7.14 that routers age more in 2D NoC than 3D NoC even though the temperature is higher in 3D NoC upper layers. The main reason is that in 2D NoC the shortest paths between each source and destination are usually longer. This means that a flit requires more time to traverse the NoC through more routers. This results in more usage of routers and subsequently more delay degradation for them. All in all, it can be concluded that 3D NoC architecture performs better in term of aging as compared to 2D NoC.

Fig. 7.15 illustrates the trend of maximum aged router's delay degradation in 3 years (9.3E+7 seconds) of execution in both 3D and 2D NoCs for three selected benchmarks. It can be seen in all of them that the maximum age after 3 years of execution for 2D NoC is higher than 3D NoC. For example, the maximum age of routers in 2D NoC is 12.89%, 11.24%, and 14.28% higher than 3D NoC for FFT, X264, and SWAPTIONS, respectively. Fig. 7.15.c and Fig. 7.15.d show the delay degradation trend for FFT. As depicted in the figure AROMa outperforms OFAR by 14% and 8.9% in 3D and 2D NoC, respectively. Based on the holistic results in Table 7.2 and Table 7.3 and also the result is Fig. 7.13 and Fig. 7.15, we can conclude that AROMa performs better in 3D NoC and decreases maximum age even more. In contrary, on average OFAR shows better performance in 2D as compared to 3D in term of lowering the maximum age.

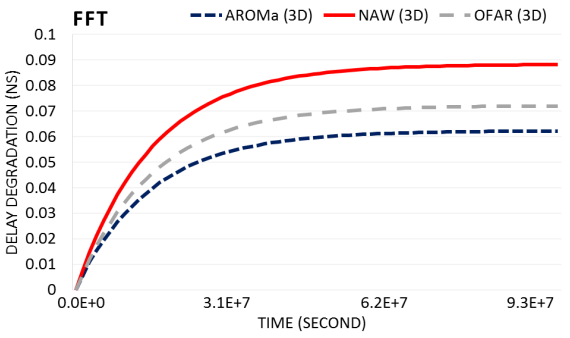


(a) In 2D NoC

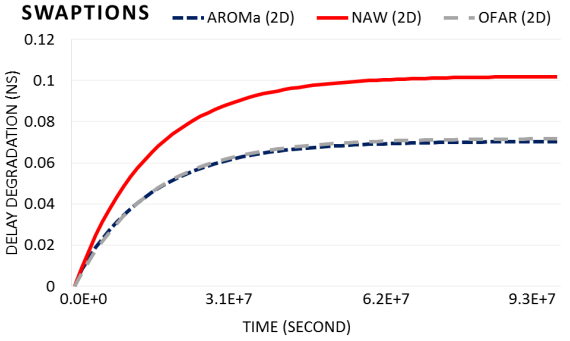
(b) In 3D NoC



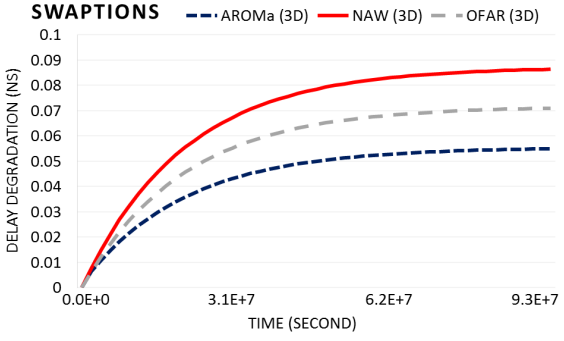
(c) In 2D NoC



(d) In 3D NoC



(e) In 2D NoC



(f) In 3D NoC

Figure 7.15: Delay degradation for 3 years (9.3E+7 seconds) for maximum aged routers in the network.

7.7.4 Overhead

Area

As shown in Fig. 7.9, for online monitoring system we need to add two parallel counters (12-bit and 14-bit), seven 14-bit subtractors (Maximum number of ports in a 3D NoC is seven), seven 4-bit MUXes, and seven AND logic gates. We embedded them into our router architecture from [1] and used Xilinx ISE synthesis tools to extract area overhead. Our analysis shows that the area overhead is negligible ($\sim 0.55\%$). The area overhead for previous methods, that OFAR deployed them, in [20] and [152] is ($\sim 4\%$) and ($\sim 1.2\%$), respectively. This means that our online monitoring system imposes a negligible area overhead to each router as compared to state-of-the-arts.

As shown in Fig. 7.9 and Fig. 7.10, each D-CAT has three columns of information. The first one corresponds to *residence-time* (rs) which can be represented by two bytes. Similarly, the second column represents *number-of-flits* (fl), which is also can be represented by two bytes. In addition, to store the aging-induced delay degradation corresponding to each fl and rs pair four bytes in third column is used. In our experiments, the number of steps is 482 (in Algorithm 2), which means our D-CATs have 482 entries considering that each step is 50 flits. We observed that a change in number of flits by 50 does not change the power consumption and temperature noticeably. Therefore, the total amount of memory that is required for each D-CAT is $482 \times (2 + 2 + 4)$ bytes which is equal to ($\sim 3.8KB$).

Energy-Delay-Product-Per-Flit

In Fig. 7.16, we illustrate the *Energy-Delay-Product-Per-Flit* (EDDPF) of each benchmark for AROMa and OFAR schemes compared to NAW in both 2D and 3D NoCs. The EDDPF on average is 1.53%, 1.88%, 1.67%, and 6.91% for AROMa in 2D NoC, AROMa in 3D,

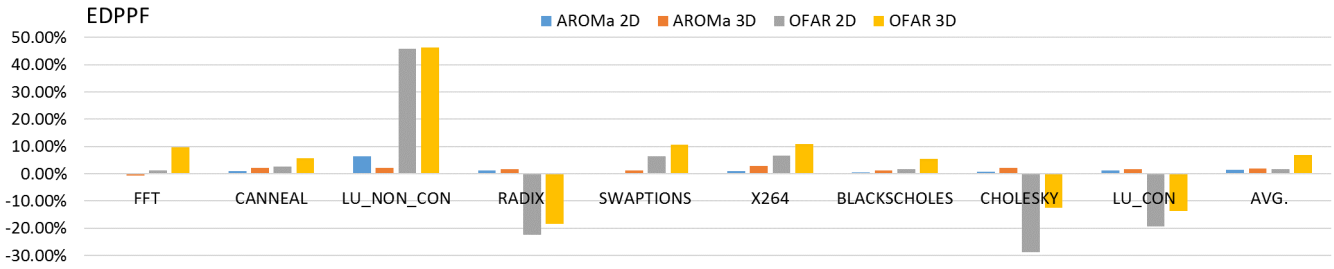


Figure 7.16: EDPPF for 3-year (9.3E+7 seconds) execution time in each benchmark for different schemes.

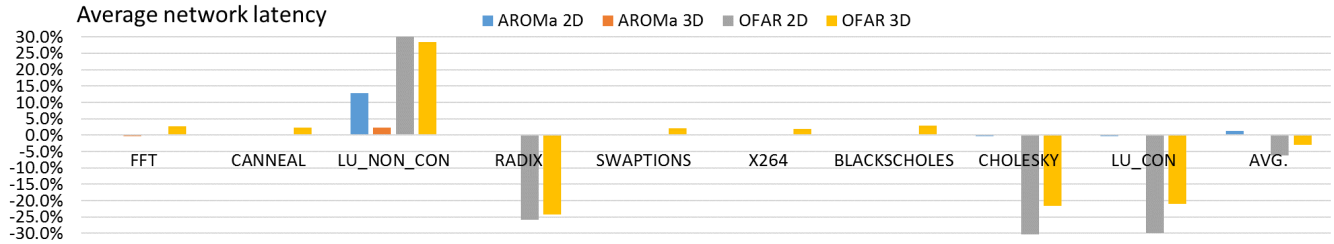


Figure 7.17: Average network latency for 3-year (9.3E+7 seconds) execution time in each benchmark for different schemes.

OFAR in 2D, and OFAR in 3D, respectively. It can be concluded that OFAR has higher EDDPF overhead as compared to AROMa for both 2D and 3D NoCs on average. However, in some of the benchmarks the EDPPF is improved in OFAR scheme. It should be noted that in OFAR, there are faulty routers since they are overstressed (i.e. overaged) due to offline budgeting. As we mentioned in Subsection 8.3, the overstressed routers due to unfair budgeting in OFAR make network behavior unpredictable. For example, in RADIX, the EDDPF overhead is improved while in LU_NON_CON it is increased.

In addition, EDPPF in 2D NoC is performing better than 3D NoC for all benchmarks expect LU_NON_CON in AROMa. Even though for 3D NoC the delay is decreased but due to increase in power and energy consumption the EDDPF will increase as compared to 2D.

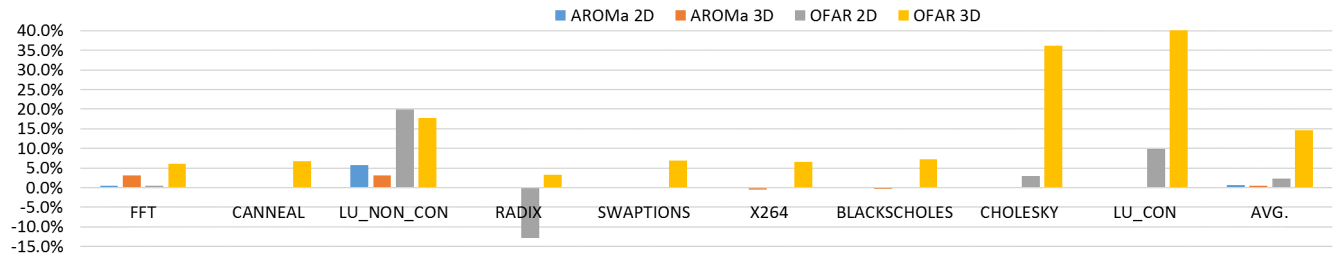


Figure 7.18: Link utilization for 3-year (9.3E+7 seconds) execution time in each benchmark for different schemes.

Network latency

Fig. 7.17 shows the average network latency for different benchmarks in both 2D and 3D NoC for our three different schemes. The average network latency is 1.31%, 0.12%, -6.24%, and -2.96% for AROMa in 2D NoC, AROMa in 3D NoC, OFAR in 2D NoC, and OFAR in 3D NoC, respectively as compared to NAW schemes. However, in the benchmarks such as FFT that do not have any faulty routers, the OFAR average network latency is higher for both 2D and 3D NoCs. In addition, the average network latency for AROMa in 3D NoC is slightly better than 2D NoC, which is expected. However, due to the unfair budgeting in OFAR the flits may traverse longer paths which include highly aged routers and leads to higher average network latency even in 3D as compared to 2D NoC. OFAR is not able to follow the online behavior of the workload in the system which can impact both temperature and stress as well as aging of routers. This is because OFAR does not adaptively change the source-destination pairs' shortest paths considering aging and eventually overstress highly aged routers.

Link utilization

Fig. 7.18 demonstrates link utilization of AROMa and OFAR schemes for 2D and 3D NoC. The average link utilization is 0.7%, 0.53%, 2.32%, and 14.69% for AROMa in 2D NoC, AROMa in 3D NoC, OFAR in 2D NoC, and OFAR in 3D NoC, respectively. Except

LU_NON_CON benchmark OFAR in 3D NoC has higher link utilization as compared to AROMa. Similarly, 2D NoC OFAR has higher link utilization in comparison to AROMa except in RADIX benchmark. Link utilization depends on benchmarks' behaviors as well as the routing algorithm. OFAR uses certain paths to traverse flits to avoid highly utilized routers that are determined by offline budgeting. This leads to lower utilization of connected links to such routers but migrates it to other set of links connected to low utilized routers, that are determined in offline budgeting. Therefore, OFAR may choose longer paths to traverse flits which results in more unnecessary misleading higher link utilization (Fig. 7.18) in comparison to AROMa and NAW for same traffic. In all, OFAR is not able to fairly utilize resources in the NoC and transferring the load (age or problem) from one region to another. As our results shows, AROMa outperforms OFAR in term of fair utilization of network resources which leads to better balance of ages.

7.8 Chapter summary

In this work, we proposed AROMa, an adaptive aging-aware routing algorithm along with an online aging monitoring system for 3D NoCs. Temperature is a fundamental challenge in 3D NoCs which can significantly change both BTI and HCI aging mechanisms. In addition, aging induced delay degradation is a function of stress quantified by usage. We introduce Distributed Centralized Aging Table (D-CAT) which stores delay degradation for each temperature and stress pairs. Our online monitoring system utilizes D-CAT for each layer of the 3D NoC to keep track of each router's age. Moreover, AROMa finds different shortest paths between each source-destination pair to avoid highly aged router at each period of time. Therefore, highly-aged routers may get a chance to recover from BTI-induced delay degradation due to our adaptive and aging-aware routing. Our extensive experimental evaluation for SPLASH-2 and PARSEC benchmarks using gem5 in full system mode for both 2D

and 3D NoC shows that AROMa outperforms state-of-the-art work. AROMa significantly is better than OFAR not only in terms of age imbalance between different routers but also minimizing maximum aged router age. Furthermore, our results shows that 3D NoC is more resistance against aging as compared to 2D NoC even though the temperature may go higher in upper layers. The main reason is that in 3D NoC paths are shorter and router's usage (stress) decreases. AROMa improves age imbalance by 60% and 72% in 2D and 3D NoC, respectively, in comparison to non-aging aware technique. In addition, the maximum age of routers decreases by 33.51% and 34.26% in 2D and 3D NoC, respectively.

Chapter 8

Conclusion

Aging-induced delay degradation of transistor leads to longer critical paths and lower clock frequency or performance of the system. Designer have to add considerable time margin as guardband to the main critical path at design time to avoid timing failure at runtime. The key point is that at design time there is not enough information to have a viable approach. This information includes temperature and stress (or usage), two major sources of aging, which are highly workload dependent and vary at runtime. We believe both design time (i.e. proactive) and runtime (i.e. reactive) approaches are required to cope with aging issues in nanometer scales computing platforms.

The usage of reconfigurable architectures such as FPGAs is increasing in both mainstream and safety critical applications ranging from the data-center acceleration to the space and avionic applications. Also the system evolution toward 3D many-core heterogeneous architectures increases the workload's behavior's variation. The variation in running workload on the system leads to imbalance aging of the circuit's components which results in premature aging of unfair highly aged resources. We studied and devised proactive as well as reactive methods to cope with aging in such systems.

To monitor aging we proposed SENSIBLE, which is a highly scalable and low overhead aging sensor. SENSIBLE only uses one clock generator as timing source for all the required sensors on the system. This and the low area overhead due to occupying only one slice on FPGA lead to higher accuracy and sensitivity of SENSIBLE. To find the number of required *Representative Critical Paths* (RCPs) for age monitoring, we proposed a two-step application dependent methodology. This methodology filters out large pool of critical paths based on the application's behavior such as temperature, stress and user constraints to find minimum number of required RCPs and insert aging sensors on their endpoints. After finding RCPs and inserting aging sensors such as SENSIBLE then system level designers can monitor aging to react accordingly.

Two proactive methods are proposed in this thesis to protect delay degradation of critical paths of implemented applications on reconfigurable architectures and the *Static Noise Margin* (SNM) of SRAM cells in such architectures. The former is a high-level physical design which is able to find the k-best aging-aware floorplans to be altered by each-others at runtime for aging mitigation. The latter is STABLE, a post-synthesis SAT-based Boolean matching to find new configuration for the implemented application while all the logic SRAM cells are flipped but the functionality is preserved. Flipping SRAM cells' contents allows their transistors to turn off and enjoy BTI recovery phase to partially recover their SNMs. SNM degradation reduces SRAM's stability and increases its *Soft Error Rate* (SER).

To monitor and mitigate aging in both 2D and 3D components in NoC we proposed AROMa. AROMa is an adaptive aging-aware routing algorithm which finds k-best source-destination shortest paths to alter them at runtime based on aging information of routers along them. Through this, we are able to balance aging among NoC components and minimize the age of maximum aged router. To find aging information of routers we proposed *Centralized Aging Table* (CAT) and Distributed CAT (D-CAT) for 2D and 3D NOCs, respectively. CAT is able to convert transistor level aging information to the running workloads' behaviors (i.e.

flits) in NoCs. D-CAT is able to capture the inter-layer temperature variations of 3D NoCs.

We proposed ADAMANT, which is an aging-aware task mapping for heterogeneous many-core systems. This technique balances aging between different cores that have various characteristics. ADAMANT exploits on-chip sensing of aging, performance, and power in order to enable online workload characterization to select task-to-core mappings that yield both increased system lifetime and energy efficiency.

8.0.1 Future work

As mentioned earlier both proactive and reactive methods are required to protect computing platforms against aging. As shown in Fig. 8.1, the proposed methods in this dissertation can be applied and integrated in a framework to avoid performance degradation in many-core heterogeneous NoC-based platforms. It is illustrated that all the proposed methods are

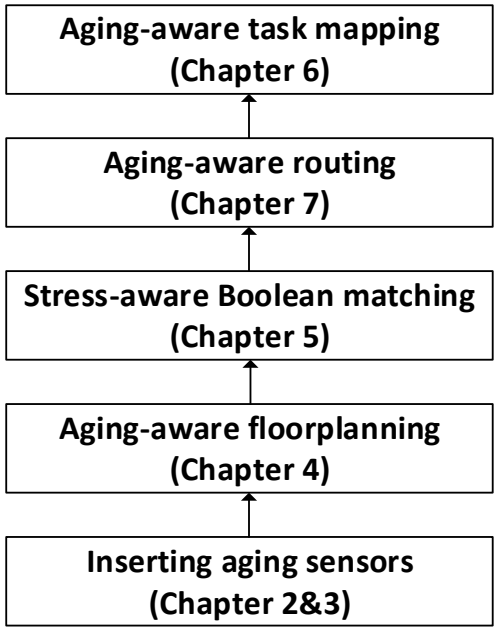


Figure 8.1: The proposed framework for aging mitigation and monitoring in many-core 3D heterogeneous architectures.

required to avoid performance degradation because of aging. This framework connects the aging information across the layers from sensor insertion and synthesis levels all the way up to the system level task mapping.

Another future direction that can be pursued is to consider aging degradation in approximation computing applications. Since adding guardband increases the area, power and performance overheads for approximation computing the imposed inaccuracy due to aging can be leveraged to reduce overheads and energy consumption. This can be performed and studied by reducing guardband and accepting some level of errors.

Bibliography

[1] Open source noc router rtl, http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/resources/router.

[2] International technology roadmap for semiconductors 2.0, edition 2015, http://www.semiconductors.org. 2015.

[3] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42. IEEE, 2009.

[4] M. O. Agyeman, A. Ahmadinia, and N. Bagherzadeh. Performance and energy aware inhomogeneous 3d networks-on-chip architecture generation. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1756–1769, 2016.

[5] H. Aljazzar and S. Leue. KâĄŐ: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, 2011.

[6] A. Amouri, S. Kiamehr, and M. Tahoori. Investigation of aging effects in different implementations and structures of programmable routing resources of fpgas. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 215–219. IEEE, 2012.

[7] A. Amouri and M. Tahoori. A low-cost sensor for aging and late transitions detection in modern fpgas. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 329–335. IEEE, 2011.

[8] A. Amouri and M. Tahoori. High-level aging estimation for fpga-mapped designs. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 284–291. IEEE, 2012.

[9] H. Amrouch, J. Martin-Martinez, V. M. van Santen, M. Moras, R. Rodriguez, M. Nafria, and J. Henkel. Connecting the physical and application level towards grasping aging effects. In *Reliability Physics Symposium (IRPS), 2015 IEEE International*, pages 3D–1. IEEE, 2015.

[10] D. M. Ancajas, K. Bhardwaj, K. Chakraborty, and S. Roy. Wearout resilience in nocs through an aging aware adaptive routing algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):369–373, 2015.

[11] D. M. Ancajas, K. Chakraborty, and S. Roy. Proactive aging management in heterogeneous nocs through a criticality-driven routing approach. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1032–1037. EDA Consortium, 2013.

[12] D. M. Ancajas, J. M. Nickerson, K. Chakraborty, and S. Roy. Hci-tolerant noc router microarchitecture. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–10. IEEE, 2013.

[13] J. Angermeier, D. Ziener, M. Glaß, and J. Teich. Stress-aware module placement on reconfigurable devices. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 277–281. IEEE, 2011.

[14] ARM. big. LITTLE Technology : The Future of Mobile. Technical report, ARM, 2013.

[15] J. H. Bahn and N. Bagherzadeh. A generic traffic model for on-chip interconnection networks. *Network on Chip Architectures*, page 22, 2008.

[16] A. Bansal, R. Rao, J.-J. Kim, S. Zafar, J. H. Stathis, and C.-T. Chuang. Impact of nbti and pbti in sram bit-cells: Relative sensitivities and guidelines for application-specific target stability/performance. In *Reliability Physics Symposium, 2009 IEEE International*, pages 745–749. IEEE, 2009.

[17] M. Basoglu, M. Orshansky, and M. Erez. Nbti-aware dvfs: A new approach to saving energy and increasing processor lifetime. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 253–258. IEEE, 2010.

[18] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM journal of research and development*, 50(4.5):433–449, 2006.

[19] K. Bhardwaj, K. Chakraborty, and S. Roy. An milp-based aging-aware routing algorithm for nocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 326–331. EDA Consortium, 2012.

[20] K. Bhardwaj, K. Chakraborty, and S. Roy. Towards graceful aging degradation in nocs through an adaptive routing algorithm. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 382–391. IEEE, 2012.

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

[22] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula. Predictive Modeling of the NBTI Effect for Reliable Design. In *IEEE Custom Integrated Circuits Conference 2006*, number Cicc, pages 189–192. IEEE, sep 2006.

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, page 72, New York, New York, USA, 2008. ACM Press.

[24] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[25] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1, aug 2011.

[26] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1, 2006.

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

[28] C. Bolchini, A. Miele, and C. Sandionigi. A novel design methodology for implementing reliability-aware systems on sram-based fpgas. *IEEE Transactions on Computers*, 60(12):1744–1758, 2011.

[29] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Qnoc: Qos architecture and design process for network on chip. *Journal of systems architecture*, 50(2):105–128, 2004.

[30] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Ieee Micro*, 25(6):10–16, 2005.

[31] A. Bravaix, C. Guerin, V. Huard, D. Roy, J. Roux, and E. Vincent. Hot-carrier acceleration factors for low power management in dc-ac stressed 40nm nmos node at high temperature. In *Reliability Physics Symposium, 2009 IEEE International*, pages 531–548. IEEE, 2009.

[32] A. A. Bsoul, N. Manjikian, and L. Shang. Reliability-and process variation-aware placement for fpgas. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pages 1809–1814. IEEE, 2010.

[33] A. Calimera, E. Macii, and M. Poncino. Analysis of nbti-induced snm degradation in power-gated sram cells. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 785–788. IEEE, 2010.

[34] E. H. Cannon, A. KleinOsowski, R. Kanj, D. D. Reinhardt, and R. V. Joshi. The impact of aging effects and manufacturing variation on sram soft-error rate. *IEEE Transactions on Device and Materials Reliability*, 8(1):145–152, 2008.

[35] G. Chen, K. Chuah, M. Li, D. S. Chan, C. Ang, J. Zheng, Y. Jin, and D. Kwong. Dynamic nbti of pmos transistors and its impact on device lifetime. In *Reliability Physics Symposium Proceedings, 2003. 41st Annual. 2003 IEEE International*, pages 196–202. IEEE, 2003.

[36] H.-B. Chen, S. X.-D. Tan, X. Huang, and V. Sukharev. New electromigration modeling and analysis considering time-varying temperature and current densities. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 352–357. IEEE, 2015.

[37] K.-C. J. Chen, C.-H. Chao, and A.-Y. A. Wu. Thermal-aware 3d network-on-chip (3d noc) designs: routing algorithms and thermal managements. *IEEE Circuits and Systems Magazine*, 15(4):45–69, 2015.

[38] Q. Chen, A. Guha, and K. Roy. An accurate analytical snm modeling technique for srams based on butterworth filter function. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, pages 615–620. IEEE, 2007.

[39] Cheng Zhuo, D. Sylvester, and D. Blaauw. Process variation and temperature-aware reliability management. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 580–585. IEEE, mar 2010.

[40] M. Cho, S. T. Kim, C. Tokunaga, C. Augustine, J. P. Kulkarni, K. Ravichandran, J. W. Tschanz, M. M. Khellah, and V. De. Postsilicon voltage guard-band reduction in a 22 nm graphics execution core using adaptive voltage scaling and dynamic power gating. *IEEE Journal of Solid-State Circuits*, 52(1):50–63, 2017.

[41] E. G. Coffman and J. L. Bruno. *Computer and job-shop scheduling theory*. A Wiley-Interscience publication. Wiley, 1976.

[42] J. Cong and K. Minkovich. Improved sat-based boolean matching using implicants for lut-based fpgas. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, pages 139–147. ACM, 2007.

[43] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[44] A. Das, A. Kumar, and B. Veeravalli. Reliability-Driven Task Mapping for Lifetime Extension of Networks-on-Chip Based Multiprocessor Systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 689–694, New Jersey, 2013. IEEE Conference Publications.

[45] A. Das, A. Kumar, and B. Veeravalli. Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):869–884, 2016.

[46] R. Degraeve, G. Groeseneken, R. Bellens, J. L. Ogier, M. Depas, P. J. Roussel, and H. E. Maes. New insights in the relation between electron trap generation and the statistical properties of oxide breakdown. *IEEE Transactions on Electron Devices*, 45(4):904–911, 1998.

[47] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, et al. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *IEEE Journal of Solid-State Circuits*, 46(1):184–193, 2011.

[48] C. E. Dike, N. A. Kurd, P. Patra, and J. Barkatullah. A design for digital, dynamic clock deskew. In *VLSI Circuits, 2003. Digest of Technical Papers. 2003 Symposium on*, pages 21–24. IEEE, 2003.

[49] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, J. Plosila, P. Liljeberg, M. Palesi, and H. Tenhunen. Haraq: congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 19–26. IEEE, 2012.

[50] M. Ebrahimi, Z. Ghaderi, E. Bozorgzadeh, and Z. Navabi. Path selection and sensor insertion flow for age monitoring in fpgas. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 792–797. EDA Consortium, 2016.

[51] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. B. Tahoori. Aging-aware logic synthesis. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 61–68. IEEE, 2013.

[52] N. Eisley, L.-S. Peh, and L. Shang. In-network cache coherence. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–332. IEEE Computer Society, 2006.

[53] D. Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

[54] R. Faraji and H. R. Naji. Adaptive technique for overcoming performance degradation due to aging on 6t sram cells. *IEEE Transactions on device and materials reliability*, 14(4):1031–1040, 2014.

[55] Z. Feng, Y. Hu, L. He, and R. Majumdar. Ipr: in-place reconfiguration for fpga fault tolerance? In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 105–108. ACM, 2009.

[56] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw. A highly resilient routing algorithm for fault-tolerant nocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 21–26. European Design and Automation Association, 2009.

[57] 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 (TODAES)*, 20(3):39, 2015.

[58] X. Fu, T. Li, and J. A. Fortes. Architecting reliable multi-core network-on-chip for small scale processing technology. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 111–120. IEEE, 2010.

[59] K. Fujiwara, K. Kawamura, M. Yanagisawa, and N. Togawa. A high-level synthesis algorithm for fpga designs optimizing critical path with interconnection-delay and clock-skew consideration. In *VLSI Design, Automation and Test (VLSI-DAT), 2016 International Symposium on*, pages 1–4. IEEE, 2016.

[60] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C.-J. Wu. A study of mobile device utilization. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, number September 2014, pages 225–234. IEEE, mar 2015.

[61] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP Journal on Embedded Systems*, 2006(1):13–13, 2006.

[62] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[63] F. Gaspar, L. Taniça, P. Tomás, A. Ilic, and L. Sousa. A Framework for Application-Guided Task Management on Heterogeneous Embedded Systems. *ACM Transactions on Architecture and Code Optimization*, 12(4):1–25, dec 2015.

[64] Z. Ghaderi, A. Alqahtani, and N. Bagherzadeh. Online monitoring and adaptive routing for aging mitigation in nocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 2017.

[65] Z. Ghaderi and E. Bozorgzadeh. Aging-aware high-level physical planning for reconfigurable systems. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 631–636. IEEE, 2016.

[66] Z. Ghaderi, M. Ebrahimi, Z. Navabi, E. Bozorgzadeh, and N. Bagherzadeh. Sensible: A highly scalable sensor design for path-based age monitoring in fpgas. *IEEE Transactions on Computers*, 2016.

[67] Z. Ghaderi, S. G. Miremadi, H. Asadi, and M. Fazeli. Hafta: Highly available fault-tolerant architecture to protect sram-based reconfigurable devices against multiple bit upsets. *IEEE Transactions on Device and Materials Reliability*, 13(1):203–212, 2013.

[68] C. J. Glass and L. M. Ni. The turn model for adaptive routing. volume 20, pages 278–287. ACM, 1992.

[69] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel. Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[70] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel. Hayat: Harnessing Dark Silicon and Variability for Aging Deceleration and Balancing. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

[71] N. Gong, S. Jiang, A. Challapalli, M. Panesar, and R. Sridhar. Variation-and-aging aware low power embedded sram for multimedia applications. In *SOC Conference (SOCC), 2012 IEEE International*, pages 21–26. IEEE, 2012.

[72] P. Gratz, B. Grot, and S. W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 203–214. IEEE, 2008.

[73] P. Greenhalgh. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper*, pages 1–8, 2011.

[74] P. Greenhalgh. big. LITTLE Processing with ARM Cortex-A15 & Cortex-A7. Technical Report September 2011, ARM, 2011.

[75] X. U. Guide. seriens fpgas clocking resources,âĂİ. *UG472, Jun*, 12:2015, 7.

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

[77] J. Henkel, H. Khdr, S. Pagani, and M. Shafique. New trends in dark silicon. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, number Figure 1, pages 1–6, New York, New York, USA, 2015. ACM Press.

[78] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, 2001.

[79] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats. In *Proceeding of the 7th international conference on Autonomic computing - ICAC '10*, page 79, New York, New York, USA, 2010. ACM Press.

[80] W.-H. Hu, S. E. Lee, and N. Bagherzadeh. Dmesh: a diagonally-linked mesh network-on-chip architecture. *Network on Chip Architectures*, page 14, 2008.

[81] Y. Hu, V. Shih, R. Majumdar, and L. He. Exploiting symmetry in sat-based boolean matching for heterogeneous fpga technology mapping. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 350–353. IEEE, 2007.

[82] L. Huang, F. Yuan, and Q. Xu. On task allocation and scheduling for lifetime extension of platform-based mpsoc designs. *IEEE Transactions on Parallel and Distributed Systems*, 22(12):2088–2099, 2011.

[83] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam. Compact thermal modeling for temperature-aware design. In *Proceedings of the 41st annual Design Automation Conference*, pages 878–883. ACM, 2004.

[84] F. L. Kastensmidt, J. Tonfat, T. Both, P. Rech, G. Wirth, R. Reis, F. Bruguier, P. Benoit, L. Torres, and C. Frost. Voltage scaling and aging effects on soft error rate in sram-based fpgas. *Microelectronics Reliability*, 54(9):2344–2348, 2014.

[85] J. Keane, T.-h. Kim, X. Wang, and C. H. Kim. On-chip reliability monitors for measuring circuit degradation. *Microelectronics Reliability*, 50(8):1039–1053, aug 2010.

[86] J. P. Keane. *On-Chip Circuits for Characterizing Transistor Aging Mechanisms in Advanced CMOS Technologies*. PhD thesis, UNIVERSITY OF MINNESOTA, 2010.

[87] B. Khaleghi, B. Omidi, H. Amrouch, J. Henkel, and H. Asadi. Stress-aware routing to mitigate aging effects in sram-based fpgas. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–8. IEEE, 2016.

[88] S. Kiamehr, A. Amouri, and M. B. Tahoori. Investigation of nbti and pbti induced aging in different lut implementations. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pages 1–8. IEEE, 2011.

[89] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das. A novel dimensionally-decomposed router for on-chip communication in 3d architectures. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 138–149. ACM, 2007.

[90] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. *ACM SIGARCH Computer Architecture News*, 34(2):4–15, 2006.

[91] T. T.-H. Kim and Z. H. Kong. Impact analysis of nbti/pbti on sram v min and design techniques for improved sram v min. *JSTS: Journal of Semiconductor Technology and Science*, 13(2):87–97, 2013.

[92] C.-T. Ko and K.-N. Chen. Wafer-level bonding/stacking technology for 3d integration. *Microelectronics reliability*, 50(4):481–488, 2010.

[93] A. Kohler, G. Schley, and M. Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):883–896, 2010.

[94] A. T. Krishnan and P. E. Nicollian. Analytic extension of the cell-based oxide breakdown model to full percolation and its implications. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 232–239. IEEE, 2007.

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

[96] C. Leong, J. Semião, I. C. Teixeira, M. B. Santos, J. P. Teixeira, M. Valdes, J. Freijedo, J. J. Rodríguez-Andina, and F. Vargas. Aging monitoring with local sensors in fpga-based designs. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4. IEEE, 2013.

[97] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir. Design and management of 3d chip multiprocessors using network-in-memory. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 130–141. IEEE Computer Society, 2006.

[98] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. The McPAT Framework for Multicore and Manycore Architectures. *ACM Transactions on Architecture and Code Optimization*, 10(1):1–29, apr 2013.

[99] G. Liu, J. Park, and D. Marculescu. Procrustes: Power Constrained Performance Improvement Using Extended Maximize-then-Swap Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 0070(c):1–1, 2015.

[100] N. Madan and R. Balasubramonian. Leveraging 3d technology for improved reliability. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 223–235. IEEE, 2007.

[101] MediaTek. MediaTek CorePilot, Heterogeneous Multi-Processing Technology.

[102] N. Mehta and A. DeHon. Variation and aging tolerance in fpgas. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*, pages 365–380. Springer, 2011.

[103] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. López, R. Holsmark, and J. Duato. Region-based routing: a mechanism to support efficient routing algorithms in nocs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3):356–369, 2009.

[104] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini. A Linux-governor based Dynamic Reliability Manager for android mobile devices. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, pages 1–4, New Jersey, 2014. IEEE Conference Publications.

[105] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. Rosing. Warm: Workload-aware reliability management in linux/android. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[106] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing. Dynamic variability management in mobile multicore processors under lifetime constraints. pages 448–455. IEEE, oct 2014.

[107] 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*, pages 586–591. European Design and Automation Association, 2010.

[108] T. Muck, S. Sarma, and N. Dutt. Run-DMC: Runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 173–182. IEEE, oct 2015.

[109] T. R. Muck, 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*, 2016.

[110] S. Muckle. LCU14-406: A QuIC Take on Energy-Aware Scheduling. In *Linaro Connect USA*, 2014.

[111] T. S. Muthukaruppan, A. Pathania, and T. Mitra. Price theory based power management for heterogeneous multi-cores. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, pages 161–176, New York, New York, USA, 2014. ACM Press.

[112] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta. Argo: aging-aware gpgpu register file allocation. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 30. IEEE Press, 2013.

[113] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[114] M. Noda, S. Kajihara, Y. Sato, K. Miyase, X. Wen, and Y. Miura. On estimation of nbti-induced delay degradation. In *Test Symposium (ETS), 2010 15th IEEE European*, pages 107–111. IEEE, 2010.

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

[116] S. Ogawa and N. Shiono. Generalized diffusion-reaction model for the low-field charge-buildup instability at the si-sio 2 interface. *Physical Review B*, 51(7):4218, 1995.

[117] M. Omaña, D. Rossi, N. Bosio, and C. Metra. Novel low-cost aging sensor. In *Proceedings of the 7th ACM international conference on Computing frontiers*, pages 93–94. ACM, 2010.

[118] M. Omana, D. Rossi, N. Bosio, and C. Metra. Low cost nbti degradation detection and masking approaches. *IEEE Transactions on Computers*, 62(3):496–509, 2013.

[119] M. P. Pagey. *Characterization and modeling of hot-carrier degradation in sub-micron NMOSFETS*. PhD thesis, Citeseer, 2002.

[120] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. Application specific routing algorithms for networks on chip. *IEEE Transactions on Parallel and Distributed Systems*, 20(3):316–330, 2009.

[121] F. Paterna, A. Acquaviva, and L. Benini. Aging-Aware Energy-Efficient Workload Allocation for Mobile Multimedia Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 24(8):1489–1499, aug 2013.

[122] M. Poirier. LCA14-104: GTS- A solution to support ARM's big.LITTLE technology. In *Linaro Connect USA*, 2014.

[123] P. M. Rao, A. Amouri, S. Kiamehr, and M. B. Tahoori. Altering lut configuration for wear-out mitigation of fpga-mapped designs. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8. IEEE, 2013.

[124] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel. dTune: Leveraging Reliable Code Generation for Adaptive Dependability Tuning under Process Variation and Aging-Induced Effects. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, pages 1–6, New York, New York, USA, 2014. ACM Press.

[125] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, and J. Duato. Efficient implementation of distributed routing algorithms for nocs. *IET Computers & Digital Techniques*, 3(5):460–475, 2009.

[126] N. Rohbani, M. Ebrahimi, S.-G. Miremadi, and M. B. Tahoori. Bias temperature instability mitigation via adaptive cache size management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[127] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan. Efficient sat-based boolean matching for fpga technology mapping. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 466–471. IEEE, 2006.

[128] S. S. Sahoo, A. Kumar, and B. Veeravalli. Design and Evaluation of Reliability-oriented Task Re-Mapping in MPSoCs using Time-Series Analysis of Intermittent faults. In *DATE 2016*, 2016.

[129] R. Salamat, M. Khayambashi, M. Ebrahimi, and N. Bagherzadeh. A resilient routing algorithm with formal reliability analysis for partially connected 3d-nocs. *IEEE Transactions on Computers*, 65(11):3265–3279, 2016.

[130] S. Sarma, T. Muck, L. A. D. Bathen, N. Dutt, and A. Nicolau. SmartBalance: A Sensing-Driven Linux Load Balancer for Energy Efficiency of Heterogeneous MPSoCs. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

[131] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel. Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 527–534. IEEE, 2007.

[132] D. K. Schroder and J. A. Babcock. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of applied Physics*, 94(1):1–18, 2003.

[133] E. Seevinck, F. J. List, and J. Lohstroh. Static-noise margin analysis of mos sram cells. *IEEE Journal of solid-state circuits*, 22(5):748–754, 1987.

[134] D. Sengupta and S. Sapatnekar. Estimating circuit aging due to bti and hci using ring-oscillator-based sensors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[135] M. Shafique, M. U. K. Khan, and J. Henkel. Content-aware low-power configurable aging mitigation for sram memories. *IEEE Transactions on Computers*, 65(12):3617–3630, 2016.

[136] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras. Predictive Dynamic Thermal and Power Management for Heterogeneous Mobile Platforms. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 960–965, 2015.

[137] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-Aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1):94–125, mar 2004.

[138] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari. Toward increasing fpga lifetime. *IEEE Transactions on Dependable and Secure Computing*, 5(2):115–127, 2008.

[139] L. Sterpone and M. Violante. A new reliability-oriented place and route algorithm for sram-based fpgas. *IEEE Transactions on Computers*, 55(6):732–744, 2006.

[140] A. Stoddard, A. Gruwell, P. Zabriskie, and M. Wirthlin. A hybrid approach to fpga configuration scrubbing. *IEEE Transactions on Nuclear Science*, 2016.

[141] E. Stott and P. Y. Cheung. Improving fpga reliability with wear-levelling. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 323–328. IEEE, 2011.

[142] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung. Degradation in fpgas: measurement and modelling. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, pages 229–238. ACM, 2010.

[143] 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*, pages 1–6. ACM, 2014.

[144] E. E. Swartzlander. Parallel counters. *IEEE Transactions on computers*, 100(11):1021–1024, 1973.

[145] E. Takeda and N. Suzuki. An empirical model for device degradation due to hot-carrier injection. *IEEE electron device letters*, 4(4):111–113, 1983.

[146] M. Tang, X. Lin, and M. Palesi. An offline method for designing adaptive routing based on pressure model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(2):307–320, 2015.

[147] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 129–140. IEEE, 2008.

[148] M. D. Valdes-Pena, J. F. Freijedo, M. J. M. Rodriguez, J. J. Rodriguez-Andina, J. Semiao, I. M. C. Teixeira, J. P. C. Teixeira, and F. Vargas. Design and validation of configurable online aging sensors in nanometer-scale fpgas. *IEEE Transactions on Nanotechnology*, 12(4):508–517, 2013.

[149] V. M. van Santen, H. Amrouch, J. Martin-Martinez, M. Nafria, and J. Henkel. Designing guardbands for instantaneous aging effects. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[150] E. I. Vatajelu, G. Tsiligiannis, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Todri, A. Virazel, F. Wrobel, and F. Salgne. On the correlation between static noise margin and soft error rate evaluated for a 40nm sram cell. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*, pages 143–148. IEEE, 2013.

[151] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of pmos nbti effect for robust nanometer design. In *Proceedings of the 43rd annual Design Automation Conference*, pages 1047–1052. ACM, 2006.

[152] L. Wang, X. Wang, and T. Mak. Dynamic programming-based lifetime aware adaptive routing algorithm for network-on-chip. In *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE, 2014.

[153] 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*, pages 736–741. ACM, 2012.

[154] 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 - ICCAD '12*, page 736, New York, New York, USA, 2012. ACM Press.

[155] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao. The impact of nbti on the performance of combinational and sequential circuits. In *Proceedings of the 44th annual Design Automation Conference*, pages 364–369. ACM, 2007.

[156] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao. The impact of nbti effect on combinational circuit: modeling, simulation, and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(2):173–183, 2010.

[157] X. Wang, W. Xu, and C. H. Kim. Sram read performance degradation under asymmetric nbti and pbti stress: Characterization vehicle and statistical aging data. In *Custom Integrated Circuits Conference (CICC), 2014 IEEE Proceedings of the*, pages 1–4. IEEE, 2014.

[158] S. Warnock and J. A. del Alamo. Stress and characterization strategies to assess oxide breakdown in high-voltage gan field-effect transistors. In *Compound Semiconductor Manuf. Technol. Conf*, pages 311–314, 2015.

[159] P. M. Yaghini, A. Eghbal, S. S. Yazdi, N. Bagherzadeh, and M. M. Green. Capacitive and inductive tsv-to-tsv resilient approaches for 3d ics. *IEEE Transactions on Computers*, 65(3):693–705, 2016.

[160] G. Yan, Y. Han, and X. Li. Revivenet: A self-adaptive architecture for improving lifetime reliability via localized timing adaptation. *IEEE Transactions on Computers*, 60(9):1219–1232, 2011.

[161] G. Yan, F. Sun, H. Li, and X. Li. Corerank: Redeeming âĂIJsick siliconâĂİ by dynamically quantifying core-level healthy condition. *IEEE Transactions on Computers*, 65(3):716–729, 2016.

[162] H. Yu, Q. Xu, and P. H. Leong. Fine-grained characterization of process variation in fpgas. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 138–145. IEEE, 2010.

[163] J. Yun, J. Park, and W. Baek. HARS: a Heterogeneity-Aware Runtime System for Self-Adaptive Multithreaded Applications. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, New York, New York, USA, 2015. ACM Press.

[164] S. Zafar, Y. Kim, V. Narayanan, C. Cabral, V. Paruchuri, B. Doris, J. Stathis, A. Callegari, and M. Chudzik. A comparative study of nbti and pbti (charge trapping) in sio2/hfo2 stacks with fusi, tin, re gates. In *VLSI Technology, 2006. Digest of Technical Papers. 2006 Symposium on*, pages 23–25. IEEE, 2006.

[165] S. Zafar, A. Kumar, E. Gusev, and E. Cartier. Threshold voltage instabilities in high-/spl kappa/gate dielectric stacks. *IEEE Transactions on Device and Materials Reliability*, 5(1):45–64, 2005.

[166] A. Zeng, J. Lu, K. Rose, and R. J. Gutmann. First-order performance prediction of cache memory with wafer-level 3d integration. *IEEE Design & Test of Computers*, 22(6):548–555, 2005.

[167] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In *Test Conference (ITC), 2013 IEEE International*, pages 1–10. IEEE, 2013.

[168] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, H.-J. Wunderlich, and J. Henkel. Aging resilience and fault tolerance in runtime reconfigurable architectures. *IEEE Transactions on Computers*, 2016.

[169] H. Zhang, M. A. Kochte, E. Schneider, L. Bauer, H.-J. Wunderlich, and J. Henkel. Strap: Stress-aware placement for aging mitigation in runtime reconfigurable architectures. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 38–45. IEEE, 2015.

[170] K. M. Zick and J. P. Hayes. On-line sensing for healthier fpga systems. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, pages 239–248. ACM, 2010.