



Embedded Systems

Jörg Henkel
Nikil Dutt *Editors*

Dependable Embedded Systems

OPEN ACCESS

 Springer

Embedded Systems

Series Editors

Nikil Dutt, Irvine, CA, USA

Grant Martin, Santa Clara, CA, USA

Peter Marwedel, Informatik 12, TU Dortmund, Dortmund, Germany

This Series addresses current and future challenges pertaining to embedded hardware, software, specifications and techniques. Titles in the Series cover a focused set of embedded topics relating to traditional computing devices as well as high-tech appliances used in newer, personal devices, and related topics. The material will vary by topic but in general most volumes will include fundamental material (when appropriate), methods, designs and techniques.

More information about this series at <http://www.springer.com/series/8563>

Jörg Henkel • Nikil Dutt
Editors

Dependable Embedded Systems



Editors

Jörg Henkel
Karlsruhe Institute of Technology
Karlsruhe, Baden-Württemberg,
Germany

Nikil Dutt
Computer Science
University of California, Irvine
Irvine, CA, USA



ISSN 2193-0155

Embedded Systems

ISBN 978-3-030-52016-8

<https://doi.org/10.1007/978-3-030-52017-5>

ISSN 2193-0163 (electronic)

ISBN 978-3-030-52017-5 (eBook)

© The Editor(s) (if applicable) and The Author(s) 2021. This book is an open access publication.
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*To Wolfgang,
our inspiring colleague, co-initiator of the
SPP 1500 program and a good friend.
We will truly miss him.*

*Prof. Dr. rer. nat. Wolfgang Rosenstiel
5.10.1954–19.08.2020*

Preface

Dependability has become a major issue since Moore's law had hit its limits. While Moore's law has been the pacemaker for the microelectronics age for about four decades, the exponential growth has led to the microelectronic revolution that has changed our lives in multifarious ways starting from the PC through the internet and embedded applications like safety in automotive to today's personal communication/entertainment devices. The positive side effects of this exponential growth were:

- (a) Decreased Costs: This refers to the costs per transistor that decreased exponentially as complexity (i.e., number of transistors per chip) increased. In other words, for the same costs, the customer received far more functionality when migrating from one technology node to the next one.
- (b) Increased Performance: Since transistors shrank, the effective capacitances shrank, too. Hence, signal delays decreased and allowed for higher clocking, i.e., the clock frequency could be raised and significant performance gains could be achieved.
- (c) Decreased Power Consumption: Since smaller transistors have lower effective switching capacitances, the power consumption per transistor and the overall power consumption per chip went significantly down. This opened the opportunity for new application areas like mobile devices, etc.

In summary, Moore's law had provided a win-win situation for four decades in virtually all relevant design constraints (i.e., cost, power consumption, performance, and chip area). However, as Gordon E. Moore had already stated in a talk at ISSCC 2003: "No exponential is forever . . . but we can delay 'forever' . . .," he indicated that the exponential growth cannot be sustained forever but that it may be possible to delay the point when scalability finally comes to an end.

However, systems in the nano-CMOS era are inherently undependable when further advancing from one technology node to the next.

In particular, we can identify the following challenging problems which negatively impact the dependability of future systems. If not addressed properly, the dependability of systems will significantly decrease.

The effects can be divided into two major groups: The first group comprises those effects that stem from fabrication/design time issues, whereas the second group stems from operation/run-time execution.

Fabrication and Design-Time Effects

Yield and Process Variations

Yield defines the number of flaw-free circuits in relation to all fabricated circuits. A high yield is so far considered vital for an economic production line. Unfortunately, the yield will dramatically decrease because feature sizes reach a point where the process of manufacturing underlies statistical variances. Future switching devices may be fabricated through “growing” or “self-assembly.” All known research suggest that these processes cannot be controlled entirely, leading to fabrication flaws, i.e., circuits with faulty devices. As per the definition of yield, it will at a not-that-distant point in time go to zero, i.e., no circuit can be produced without at least a single faulty switching device. The traditional way of sorting out faulty circuits will not work any longer! Rather, faults will be inherent. On the other hand, fabricated circuits (although functionally correct) will continue to exhibit increasing levels of “process variability”: i.e., a high degree of variability in the observed performance, power consumption, and reliability parameters both across manufactured parts and across use of these parts over time in the field. The traditional “guardbanding” approach of overdesigning circuits with a generous margin to hide these process variations will no longer be economically viable nor will fit into a traditional design flow that assumes a rigid specification of operational constraints for the performance, power, and reliability of manufactured circuits. Newer design techniques and methodologies will therefore need to address explicitly the effects of process variation, rather than assuming these are hidden through traditional overdesigned guardbanding margins.

Complexity

In about 10 years from now, the complexity of systems integrated into one single die will amount to basic switching devices. The steadily increasing integration complexity is efficiently exploited by the current trend towards many-core network-on-chip architectures. These architectures introduce hardware and software complexities, which previously were found on entire printed circuit boards and systems down to a single chip and provide significant performance and power advantages in comparison with single cores. A large number of processing and communication

elements require new programming and synchronization models. It leads to a paradigm shift away from the assumption of zero design errors.

Operation and Run-Time Effects

Aging Effects

Transistors in the nano-CMOS era are far more susceptible to environmental changes like heat, as an example. It causes an irreversible altering of the physical (and probably chemical) properties which, itself, lead to malfunctions and performance variability over time. Though effects like electromigration in current CMOS circuits are well known, they typically do not pose a problem since the individual switching device's lifetime is far higher than the product life cycle. In future technologies, however, individual switching devices will fail (i.e., age) earlier than the life cycle of the system (i.e., product) they are part of. Another emergent altering effect is the increasing susceptibility to performance variability resulting in changing critical paths over time. This, for instance, prevents a static determination of the chip performance during manufacturing tests.

Thermal Effects

Thermal effects will have an increasing impact on the correct functionality. Various degradation effects are accelerated by thermal stress like very high temperature and thermal cycling. Aggressive power management can produce opposite effects, e.g., hot spot prevention at the cost of increased thermal cycling. Higher integration forces to extend through the third dimension (3D circuits) which in turn increases the thermal problem since the ratio of surface-area/energy significantly worsens. Devices will be exposed to higher temperatures and increase, among others, aging effects. In addition, transient faults increase.

Soft Errors

The susceptibility of switching devices in the nano age against soft errors will increase about 8% per logic state bit for each technology generation, as recently forecasted. Soft errors are caused by energetic radiation particles (neutrons) hitting silicon chips and creating a charge on the nodes that flips a memory cell or logic latches.

The idea of this book has its origin in several international programs on dependability/reliability:

- The SPP 1500 Dependable Embedded Systems program (by DFG of Germany);
- The NSF Expedition on Variability (by NSF of USA); and
- The Japanese JST program.

While this book is not a complete representation of all of these programs, it does represent all aspects of the SPP 1500 and some aspects of the NSF Expedition on Variability and the Japanese JST program.

The book focuses on cross-layer approaches, i.e., approaches to mitigate dependability issues by means and methods that work across design abstraction layers. It is structured in the main six areas “Cross-Layer from Operating System to Application,” “Cross-Layer Dependability: From Architecture to Software and Operating System,” “Cross-Layer Resilience: Bridging the Gap between Circuit and Architectural Layer,” “Cross-Layer from Physics to Gate- and Circuit-Levels,” and “Cross-Layer from Architecture to Application.” Besides, it contains a chapter in the so-called RAP model: the resilience articulation point (RAP) model aims to provision a probabilistic fault abstraction and error propagation concept for various forms of variability-related faults in deep submicron CMOS technologies at the semiconductor material or device levels. RAP assumes that each of such physical faults will eventually manifest as a single- or multi-bit binary signal inversion or out-of-specification delay in a signal transition between bit values.

The book concludes with a perspective.

We want to thank all the authors who contributed to this book as well as all the funding agencies that made this book possible (DFG, NSP, and JST).

We hope you enjoy reading this book and we would be glad to receive feedback.

Karlsruhe, Baden-Württemberg, Germany

Jörg Henkel

Irvine, CA, USA

Nikil Dutt

Contents

RAP Model—Enabling Cross-Layer Analysis and Optimization for System-on-Chip Resilience	1
Andreas Herkersdorf, Michael Engel, Michael Glaß, Jörg Henkel, Veit B. Kleeberger, Johannes M. Kühn, Peter Marwedel, Daniel Mueller-Gritschneider, Sani R. Nassif, Semeen Rehman, Wolfgang Rosenstiel, Ulf Schlichtmann, Muhammad Shafique, Jürgen Teich, Norbert Wehn, and Christian Weis	
Part I Cross-Layer from Operating System to Application	
Soft Error Handling for Embedded Systems using Compiler-OS Interaction	33
Michael Engel and Peter Marwedel	
ASTEROID and the Replica-Aware Co-scheduling for Mixed-Criticality	57
Eberle A. Rambo and Rolf Ernst	
Dependability Aspects in Configurable Embedded Operating Systems	85
Horst Schirmeier, Christoph Borchert, Martin Hoffmann, Christian Dietrich, Arthur Martens, Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk	
Part II Cross-Layer Dependability: From Architecture to Software and Operating System	
Increasing Reliability Using Adaptive Cross-Layer Techniques in DRPs: Just-Safe-Enough Responses to Reliability Threats	121
Johannes Maximilian Kühn, Oliver Bringmann, and Wolfgang Rosenstiel	

Dependable Software Generation and Execution on Embedded Systems	139
Florian Kriebel, Kuan-Hsun Chen, Semeen Rehman, Jörg Henkel, Jian-Jia Chen, and Muhammad Shafique	
Fault-Tolerant Computing with Heterogeneous Hardening Modes	161
Florian Kriebel, Faiq Khalid, Bharath Srinivas Prabakaran, Semeen Rehman, and Muhammad Shafique	
Thermal Management and Communication Virtualization for Reliability Optimization in MPSoCs	181
Victor M. van Santen, Hussam Amrouch, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf	
Lightweight Software-Defined Error Correction for Memories	207
Irina Alam, Lara Dolecek, and Puneet Gupta	
Resource Management for Improving Overall Reliability of Multi-Processor Systems-on-Chip	233
Yue Ma, Junlong Zhou, Thidapat Chantem, Robert P. Dick, and X. Sharon Hu	
Part III Cross-Layer Resilience: Bridging the Gap Between Circuit and Architectural Layer	
Cross-Layer Resilience Against Soft Errors: Key Insights	249
Daniel Mueller-Gritschneider, Eric Cheng, Uzair Sharif, Veit Kleeberger, Pradip Bose, Subhasish Mitra, and Ulf Schlichtmann	
Online Test Strategies and Optimizations for Reliable Reconfigurable Architectures	277
Lars Bauer, Hongyan Zhang, Michael A. Kochte, Eric Schneider, Hans-Joachim Wunderlich, and Jörg Henkel	
Reliability Analysis and Mitigation of Near-Threshold Voltage (NTC) Caches	303
Anteneh Gebregiorgis, Rajendra Bishnoi, and Mehdi B. Tahoori	
Part IV Cross-Layer from Physics to Gate- and Circuit- Levels	
Selective Flip-Flop Optimization for Circuit Reliability	337
Mohammad Saber Golanbari, Mojtaba Ebrahimi, Saman Kiamehr, and Mehdi B. Tahoori	
EM Lifetime Constrained Optimization for Multi-Segment Power Grid Networks	365
Han Zhou, Zeyu Sun, Sheriff Sadiqbatcha, and Sheldon X.-D. Tan	
Monitor Circuits for Cross-Layer Resiliency	385
Mahfuzul Islam and Hidetoshi Onodera	

Contents	xiii
Dealing with Aging and Yield in Scaled Technologies	409
Wei Ye, Mohamed Baker Alawieh, Che-Lun Hsu, Yibo Lin, and David Z. Pan	
Part V Cross-Layer from Architecture to Application	
Design of Efficient, Dependable SoCs Based on a Cross-Layer-Reliability Approach with Emphasis on Wireless Communication as Application and DRAM Memories	435
Christian Weis, Christina Gimmeler-Dumont, Matthias Jung, and Norbert Wehn	
Uncertainty-Aware Compositional System-Level Reliability Analysis	457
Hananeh Aliee, Michael Gläß, Faramarz Khosravi, and Jürgen Teich	
Robust Computing for Machine Learning-Based Systems	479
Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Mohammad Taghi Teimoori, Florian Kriebel, Jeff (Jun) Zhang, Kang Liu, Semeen Rehman, Theocharis Theocharides, Alessandro Artusi, Siddharth Garg, and Muhammad Shafique	
Exploiting Memory Resilience for Emerging Technologies: An Energy-Aware Resilience Exemplar for STT-RAM Memories	505
Amir Mahdi Hosseini Monazzah, Amir M. Rahmani, Antonio Miele, and Nikil Dutt	
Hardware/Software Codesign for Energy Efficiency and Robustness: From Error-Tolerant Computing to Approximate Computing	527
Abbas Rahimi and Rajesh K. Gupta	
Reliable CPS Design for Unreliable Hardware Platforms	545
Wanli Chang, Swaminathan Narayanaswamy, Alma Pröbstl, and Samarjit Chakraborty	
Power-Aware Fault-Tolerance for Embedded Systems	565
Mohammad Salehi, Florian Kriebel, Semeen Rehman, and Muhammad Shafique	
Our Perspectives	589
Jian-Jia Chen and Joerg Henkel	
Index	593

RAP Model—Enabling Cross-Layer Analysis and Optimization for System-on-Chip Resilience



**Andreas Herkersdorf, Michael Engel, Michael Glaß, Jörg Henkel,
Veit B. Kleeberger, Johannes M. Kühn, Peter Marwedel,
Daniel Mueller-Gritschneider, Sani R. Nassif, Semeen Rehman,
Wolfgang Rosenstiel, Ulf Schlichtmann, Muhammad Shafique, Jürgen Teich,
Norbert Wehn, and Christian Weis**

A. Herkersdorf (✉) · D. Mueller-Gritschneider · U. Schlichtmann
Technical University of Munich, Munich, DE, Germany
e-mail: herkersdorf@tum.de

M. Engel
Department of Computer Science, Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
e-mail: michael.engel@ntnu.no

M. Glaß
University of Ulm, Ulm, DE, Germany

J. Henkel
Karlsruhe Institute of Technology (KIT), Karlsruhe, DE, Germany

V. B. Kleeberger
Infineon Technologies AG, Munich, DE, Germany

J. M. Kühn
Preferred Networks, Inc., Tokyo, JP, Japan

P. Marwedel
Technical University of Dortmund, Dortmund, DE, Germany

S.R. Nassif
Radyalis LLC, Austin, US, United States

S. Rehman · M. Shafique
TU Wien, Vienna, AT, Austria

W. Rosenstiel
University of Tübingen, Tübingen, DE, Germany

J. Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, DE, Germany

N. Wehn · C. Weis
University of Kaiserslautern (TUK), Kaiserslautern, DE, Germany

1 Introduction/Motivation

Conquering System-on-Chip (SoC) architecture and design complexity became a major, if not the number one, challenge in integrated systems development. SoC complexity can be expressed in various ways and different dimensions: Today's single-digit nanometer feature size CMOS technologies allow for multi-billion transistor designs with millions of lines of code being executed on dozens of heterogeneous processing cores. Proving the functional correctness of such designs according to the SoC specifications is practically infeasible and can only be achieved probabilistically within tolerable margins. Further consequences of this ever-increasing hardware/software complexity are: Increasing susceptibility of application- and system-level software codes to security and safety exposures, as well as operational variability of nanometer size semiconductor devices because of environmental or manufacturing variations. The SPP1500 Dependable Embedded Systems Priority Program of the German Research Foundation (DFG) [8] focused on tackling the latter class of exposures. NBTI (negative-bias temperature instability) aging, physical electromigration damage and intermittent, radiation induced bit flips in registers (SEUs (single event upsets)) or memory cells are some manifestations of CMOS variability. The Variability Expedition program by the United States National Science Foundation (NSF) [6] is a partner program driven by the same motivation. There has been and still is a good amount of bi- and multilateral technical exchange and collaboration between the two national-level initiatives.

Divide and conquer strategies, for example, by hierarchically layering a system according to established abstraction levels, proved to be an effective approach for coping with overall system complexity in a level by level manner. Layering SoCs bottom-up with semiconductor materials and transistor devices, followed by combinatorial logic, register-transfer, micro-/macro-architecture levels, and runtime environment middleware, as well as application-level software at the top end of the hierarchy, is an established methodology used both in industry and academia. The seven layer Open Systems Interconnection (OSI) model of the International Organization for Standardization provides a reference framework for communication network protocols with defined interfaces between the layers. It is another example of conquering the complexity of the entire communication stack by layering.

Despite these merits and advantages attributed to system layering, a disadvantage of this approach cannot be overlooked. Layering fosters specialization by focusing the expertise of a researcher or developer to one specific abstraction level only (or to one layer plus certain awareness for the neighboring layers at best). Specialization and even sub-specialization within one abstraction layer became a necessity as the complexity within one layer raises already huge design challenges. However, the consequence of layering and specialization for overall system optimization is that such optimizations are typically constrained by the individual layer boundaries. Cross-layer optimization strives to pursue a more vertical approach, taking the perspectives of two or more, adjacent or non-adjacent, abstraction levels for certain system properties or qualities into account. A holistic approach (considering all abstraction levels for all system properties) is not realistic because of the overall sys-

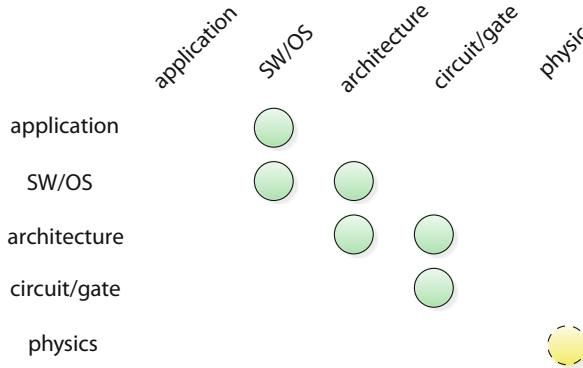


Fig. 1 RAP covers probabilistic error modeling and propagation of physics induced variabilities from circuit/logic up to application level

tem complexity. Nevertheless, for some properties, cross-layer approaches proved to be effective. Approximate computing, exploiting application-level tolerance to on-purpose circuit level inaccuracies in arithmetic operations for savings in silicon area and a lower power dissipation, is a widely adopted example of cross-layer optimization. Cross-layer approaches have also been suggested as a feasible technique to enhance reliability of complex systems [21, 26].

A prerequisite for effective cross-layer optimization is the ability to correlate the causes or events happening at one particular level with the effects or symptoms they will cause at other abstraction levels. Hierarchical system layering and specialization implies that subject matters and corresponding terminology are quite different between levels, especially when the levels of interest are several layers apart. The objective of the presented Resilience Articulation Point (RAP) model is to provision probabilistic fault abstraction and error propagation concepts for various forms of variability induced phenomena [9, 28]. Or, expressed differently, RAP aims to help annotate how variability related physical faults occurring at the semiconductor material and device levels (e.g., charge separation in the silicon substrate in response to a particle impact) can be expressed at higher abstraction levels. Thus, the impact of the low-level physical faults onto higher level fault tolerance, such as instruction vulnerability analysis of CPU core microarchitectures, or fault-aware real-time operating system middleware, can be determined without the higher level experts needing to be aware of the fault representation and error transformation at the lower levels. This cross-layer scope and property differentiates RAP from traditional digital logic fault models, such as stuck-at [18] or the conditional line flip (CLF) model [35]. These models, originally introduced for logic testing purposes, focus on the explicit fault stimulation, error propagation and observation within one and the same abstraction level. Consequently, RAP can be considered as an enabler for obtaining a cross-layer perspective in system optimization. RAP covers all SoC hardware/software abstraction levels as depicted in Fig. 1.

2 Resilience Articulation Point (RAP) Basics

In graph theory, an articulation point is a vertex that connects sub-graphs within a bi-connected graph, and whose removal would result in an increase of the number of connecting arcs within the graph. Translated into our domain of dependability challenges in SoCs, spatially and temporally correlated bit flips represent the single connecting vertex between lower layer fault origins and the upper layer error and failure models of hardware/software system abstraction (see Fig. 2).

The RAP model is based on three foundational assumptions: First, the hypothesis that every variability induced fault at the semiconductor material or device level will manifest with a certain probability as a permanent or transient single- or multi-bit signal inversion or out-of-specification delay in a signal transition. In short, we refer to such signal level misbehavior in terms of logic level or timing as a bit flip error, and model it by a probabilistic, location and time dependent error function $\mathcal{P}_{\text{bit}}(x, t)$. Second, probabilistic error functions $\mathcal{P}_L(x, t)$, which are specific to a certain abstraction layer and describe how layer characteristic data entities and compositional elements are affected by the low-level faults. For example, with what probability will a certain control interface signal on an on-chip CPU system bus, or a data word/register variable used by an application task be corrupted in response to a certain NBTI transistor aging rate. Third, there has to be a library of transformation functions \mathcal{T}_L converting probabilistic error functions $\mathcal{P}_L(x_1, t)$ at abstraction level L into probabilistic error functions $\mathcal{P}_{L+i}(x_2, t + \Delta t)$ at level(s) $L + i$ ($i \geq 1$) (see Fig. 3).

$$\mathcal{P}_{L+1}(x_2, t + \Delta t) = \mathcal{T}_L \circ \mathcal{P}_L(x_1, t) \quad (1)$$

Please note, although the existence of such transformation functions is a foundational assumption of the RAP model itself, the individual transformation functions

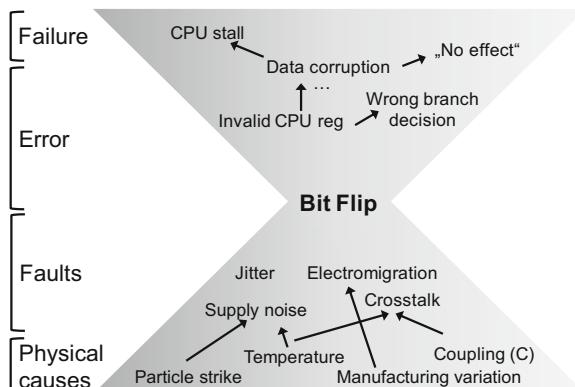


Fig. 2 Fault, error, and failure representations per abstraction levels

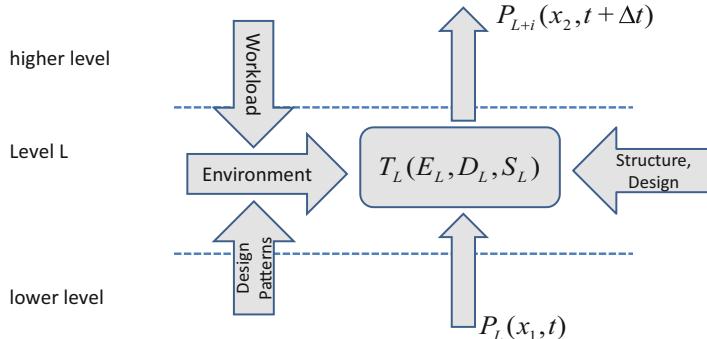


Fig. 3 Error transformation function depending on environmental, design, and system state conditions

T_L cannot come from or be a part of RAP. Transformation functions are dependent on a plurality of environmental, design and structure specific conditions, as well as implementation choices ($\mathcal{E}_L, \mathcal{D}_L, \mathcal{S}_L$) within the specific abstraction layers that are only known to the respective expert designer. Note further, the location or entity x_2 affected at a higher abstraction level may not be identical to the location x_1 , where the error manifested at the lower level. Depending on the type of error, the architecture of the system in use, and the characteristic of the application running, the error detection latency Δt during the root cause analysis for determining the error source at level L typically represents a challenging debugging problem [17].

3 Related Work

Related approaches to describe the reliability of integrated circuits and systems have been developed recently.

In safety-critical domains and to ensure reliable systems, standards prescribing reliability analysis approaches and MTTF (mean time to failure) calculations have been in existence for many decades (e.g., RTCA/DO-254—Design Assurance Guidance for Airborne Electronic Hardware, or the Bellcore/Telcordia Predictive Method, SR-332—Reliability Prediction Procedure for Electronic Equipment, in the telecom area [33]). These approaches, however, were not developed with automation in mind, and do not scale well to very complex systems.

The concept of reliability block diagrams (RBDs) has also been used to describe the reliability of systems [19]. In RBDs, each block models a component of the considered system. A failure rate is associated to each block. The RBD's structure describes how components interact. Components in parallel are redundant, whereas for serially connected components the failure of any one component causes the entire system to fail. However, more complex situations are difficult to model

and analyze. Such more complex situations include parametric dependencies (e.g., reliability dependent on temperature and/or voltage), redundancy schemes which can deal with certain failures, but not other (e.g., ECC which, depending on the code and number of redundant bits, can either deal with the detection and correction of single-bit failure, or detect, but not correct, multi-bit failures), or state-dependent reliability characteristics.

In 2012, RIIF (Reliability Information Interchange Format) was presented [4]. RIIF does not introduce fundamentally new reliability modeling and analysis concepts. Rather, the purpose is to provide a format for describing detailed reliability information of electronic components as well as the interaction among components. Parametric reliability information is supported. State-dependent reliability (modeled by Markov reliability models) is planned to be added. By providing a standardized format, RIIF intends to support the development of automated approaches for reliability analysis. It targets to support real-world scenarios in which complex electronic systems are constructed from legacy components, purchased IP blocks, and newly developed logic.

RIIF was developed in the context of European projects, driven primarily by the company IROC Technologies. The original concept was developed mostly within the MORV (Modeling Reliability under Variation) project. Extensions from RIIF to RIIF2 were recently developed in collaboration with the CLERECO (Cross-Layer Early Reliability Evaluation for the Computing Continuum) project. RIIF is a machine-readable format which allows the detailed description of reliability aspect of system components. The failure modes of each component can be described, depending on parameters of the component. The interconnection of components to a system can be described. RIIF originally focused only on hardware. RIIF2 has been proposed to extend the basic concepts of RIIF to also take software considerations into account [27].

4 Fault Abstraction at Lower Levels

The RAP model proposes modeling the location and time dependent error probability $\mathcal{P}_{\text{bit}}(x, t)$ of a digital signal by an error function \mathcal{F} with three, likewise, location and/or time dependent parameters: Environmental and operating conditions \mathcal{E} , design parameters \mathcal{D} , and (error) state bits \mathcal{S} .

$$\mathcal{P}_{\text{bit}}(x, t) = \mathcal{F}(\mathcal{E}, \mathcal{D}, \mathcal{S}) \quad (2)$$

This generic model has to be adapted to every circuit component and fault type independently. Environmental conditions \mathcal{E} , such as temperature and supply voltage fluctuations, heavily affect the functionality of a circuit. Device aging further influences the electrical properties, concretely the threshold voltage. Other environmental parameters include clock frequency instability and neutron flux density.

System design \mathcal{D} implies multiple forms of decisions making. For example, shall arithmetic adders follow a ripple-carry or carry-look-ahead architecture (enumerative decision)? What technology node to choose (discrete decision)? How much area should one SRAM cell occupy (continuous decision)? Fixing such design parameters \mathcal{D} allows the designer to make trade-offs between different decisions, which all influence the error probability of the design in one way or the other.

In order to model the dependence of the error probability on location, circuit state, and time, it is necessary to include several state variables. These state variables \mathcal{S} lead to a model which is built from conditional probabilities $\mathcal{P}(b_1|b_2)$, where the error probability of the bit b_1 is dependent on the state of the bit b_2 . For example, the failure probability of one SRAM cell depends on the error state of neighboring SRAM cells due to the probability of multi-bit upset (MBU) [8]. For an 8T SRAM cell it also depends on the stored value of the SRAM cell as the bit flip probability of a stored “1” is different from a stored “0.”

Finally, the error function \mathcal{F} takes the three parameter sets \mathcal{E} , \mathcal{D} , and \mathcal{S} and returns the corresponding bit error probability \mathcal{P}_{bit} . The error function \mathcal{F} is unique for a specific type of fault and for a specific circuit element. An error function can either be expressed by a simple analytical formula, or may require a non-closed form representation, e.g., a timing analysis engine or a circuit simulator.

In the sequel, we show by the example of SRAM memory technology, how the design of an SRAM cell (circuit structure, supply voltage, and technology node) as well as different perturbation sources, such as radiating particle strikes, noise and supply voltage drops, will affect the data bit error probability \mathcal{P}_{bit} of stored data bits.

4.1 SRAM Errors

The SRAM is well known to have high failure rates already in current technologies. We have chosen two common SRAM architectures, namely the 6-transistor (6T) and 8-transistor (8T) bit cell shown in Fig. 4. For the 6T architecture we have as design choices the number of fins for the pull-up transistors (PU), the number of fins for the

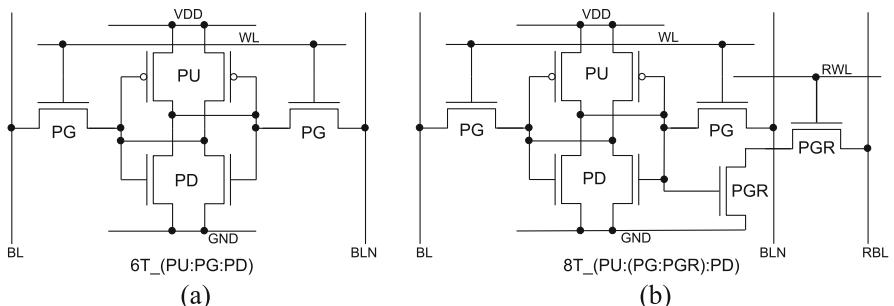


Fig. 4 Circuit schematics for standard 6T (a) and 8T (b) SRAM bit cells

pull-down transistors (PD), and the number of fins for the access transistors (PG). The resulting architecture choice is then depicted by $6T_{-}(PU:PG:PD)$. For the 8T architecture we have additionally two transistors for the read access (PGR). Hence, the corresponding architecture choice is named $8T_{-}(PU:(PG:PGR):PD)$.

An SRAM cell can fail in many different ways, for example:

- **Soft Error/Single Event Upset (SEU) failure:** If the critical charge Q_{crit} is low, the susceptibility to a bit flip caused by radiation is higher.
- **Static Voltage Noise Margin (SVNM) failure:** An SRAM cell can be flipped unintentionally when the voltage noise margin is too low (stability).
- **Read delay failure:** An SRAM cell cannot be read within a specified time.
- **Write Trip Voltage (WTV) failure:** The voltage swing during a write is not high enough at the SRAM cell.

We selected these four parameters, namely Q_{crit} , SVNM, Read delay, and WTV as resilience key parameters. To quantify the influence of technology scaling (down to 7 nm) on the resilience of the two SRAM architectures we used extensive Monte-Carlo simulations and predictive technology models (PTM) [12].

4.1.1 SRAM Errors due to Particle Strikes (Q_{crit})

Bit value changes in high density SRAMs can be induced by energetic particle strikes, e.g., alpha or neutron particles [34]. The sensitivity of digital ICs to such particles is rapidly increasing with aggressive technology scaling [12], due to the correspondingly decreasing parasitic capacitances and operating voltage.

When entering the single-digit fC region for the critical charge, as in current logic and SRAM devices and illustrated in Fig. 5a, lighter particles such as alpha and proton particles become dominant (see Fig. 5b). This increases not only error rates, but also their spread, as the range of lighter particles is much longer compared to residual nucleus [10].

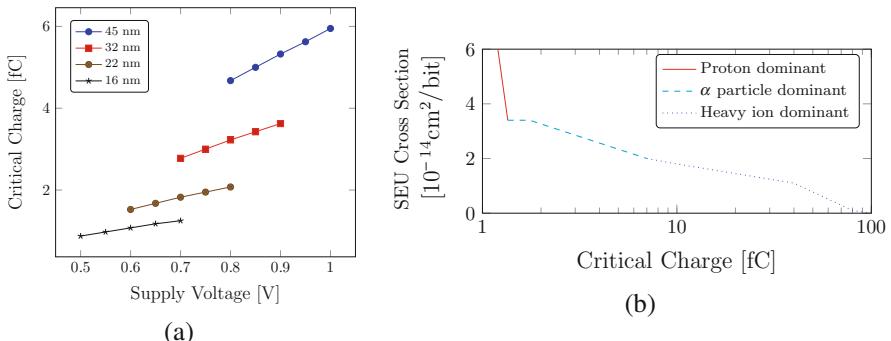


Fig. 5 Technology influence on SRAM bit flips: (a) Critical charge dependency on technology node and supply voltage for 6T SRAM cell, (b) Particle dominance based on critical charge (adapted from [10])

These technology-level faults caused by particle strikes now need to be abstracted into a bit-level fault model, so that they can be used in later system-level resilience studies. In the following this is shown for the example of neutron particle strikes. Given a particle flux of Φ , the number of neutron strikes k that hit a semiconductor area A in a time interval τ can be modeled by a Poisson distribution:

$$P(N(\tau) = k) = \exp(-\Phi \cdot A \cdot \tau) \frac{(\Phi \cdot A \cdot \tau)^k}{k!} \quad (3)$$

These neutrons are uniformly distributed over the considered area, and may only cause an error if they hit the critical area of one of the memory cells injecting a charge which is larger than the critical charge of the memory cell. The charge Q_{injected} transported by the injected current pulse from the neutron strike follows an exponential distribution with a technology dependent parameter Q_s :

$$f_Q(Q_{\text{injected}}) = \frac{1}{Q_s} \exp\left(-\frac{Q_{\text{injected}}}{Q_s}\right) \quad (4)$$

The probability that a cell flips due to this charge can then be derived as

$$P_{\text{SEU}}(Q \geq Q_{\text{crit}} | V_{\text{cellout}} = V_{DD}) = \int_{Q_{\text{crit}}}^{\infty} f_Q(Q) dQ \quad (5)$$

With increasing integration density, the probability of multi-bit upsets (MBU) also increases [16]. A comparison of the scaling trend of Q_{crit} between the 6T and 8T SRAM bit cell is shown in Fig. 6. The right-hand scale in the plots shows the 3 sigma deviation of Q_{crit} in percent to better highlight the scaling trend. The 8T-cell has a slightly improved error resilience due to an increased Q_{crit} (approximately 10% higher). However, this comes at the cost of a 25–30% area increase.

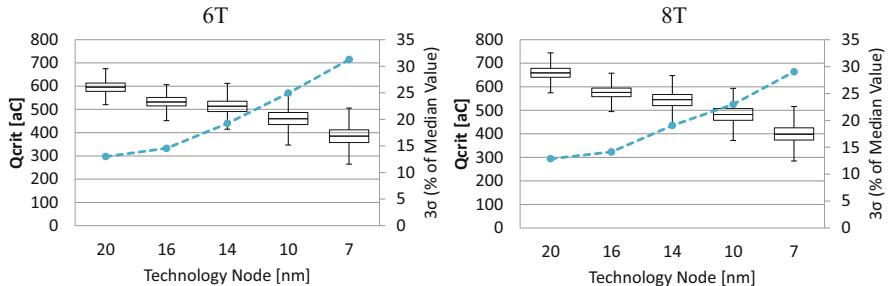


Fig. 6 Q_{crit} results for a 6T_(1:1:1) high density (left) and an 8T_(1:(1:1):1) (right) SRAM cell

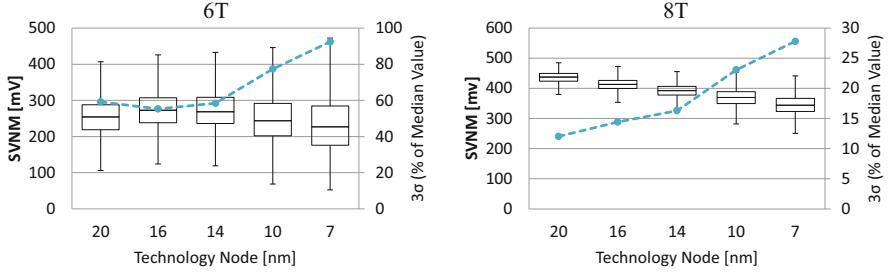


Fig. 7 SVNM results for a 6T_(1:1:1) and an 8T_(1:(1:1):1) SRAM cell

4.1.2 SRAM Errors due to Noise (SVNM)

The probability of an SRAM error (cell flip) due to noise is given by

$$P_{\text{noise_error}}(V_{\text{noise}} \geq V_{\text{SVNM}}) = \int_{V_{\text{SVNM}}}^{\infty} f_{V_{\text{noise}}}(V) dV \quad (6)$$

The distribution function $f_{V_{\text{noise}}}$ is not directly given as it depends largely on the detailed architecture and the environment in which the SRAM is integrated. Figure 7 plots the scaling trend for SVNM for both SRAM cell architectures. Due to its much improved SVNM the 8T_(1:(1:1):1) cell has an advantage over the 6T_(1:1:1) cell. Not only is the 8T cell approximately 22% better in SVNM than the 6T cell, but it is also much more robust in terms of 3σ variability (28% for 8T 7 nm compared to 90% for 6T 7 nm).

4.1.3 SRAM Errors Due to Read/Write Failures (Read Delay/WTV)

The probability of SRAM read errors can be expressed by the following equation:

$$P_{\text{read_error}}(t_{\text{read}} < t_{\text{read_delay}}) = \int_0^{t_{\text{read_delay}}} f_{t_{\text{read}}}(t) dt \quad (7)$$

In Fig. 8 the trend of the read delay for the two SRAM cell architectures is shown. Although the read delay decreases with technology scaling, which theoretically enables a higher working frequency, its relative 3σ variation can be as high as 50% at the 7 nm node. This compromises its robustness and diminishes possible increases in frequency.

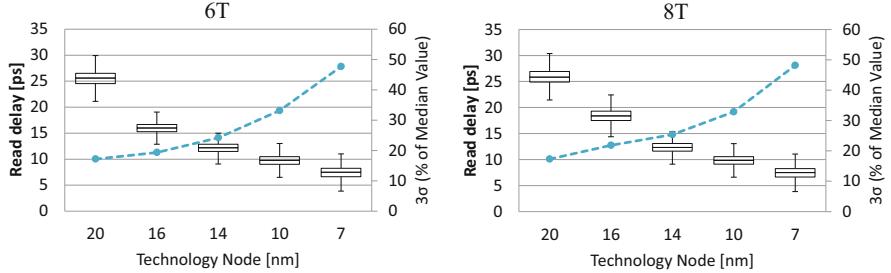


Fig. 8 Read delay results for a 6T_(1:1:1) and an 8T_(1:(1:1):1) SRAM cell

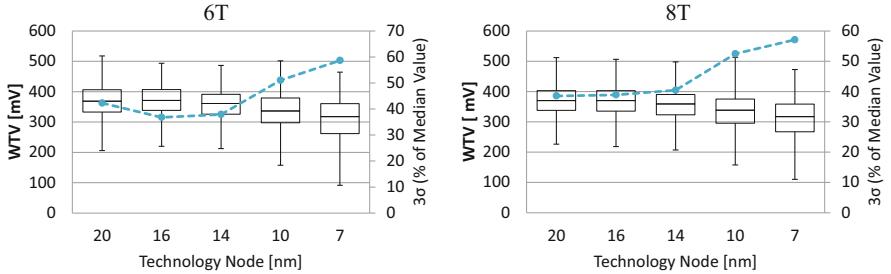


Fig. 9 WTV results for a 6T_(1:1:1) and an 8T_(1:(1:1):1) SRAM cell

If the actual applied voltage swing V_s is not sufficient to flip the content of a SRAM cell, then the data is not written correctly. The probability of such a write failure is given by

$$P_{\text{write_error}}(V_s < V_{\text{swing_min}}) = \int_0^{V_{\text{swing_min}}} f_{V_s}(V) dV \quad (8)$$

Similar to $f_{V_{\text{noise}}}$ both distribution functions for t_{read} and V_s depend strongly on the clock frequency, the transistor dimensions, the voltage supply, and the noise in the system. Figure 9 plots the scaling trend of WTV for 6T and 8T cells. The results for 6T and 8T cells are similar due to the similar circuit structure of 6T and 8T cells regarding write procedure.

4.1.4 SRAM Errors due to Supply Voltage Drop

Figure 10 shows the failure probability of a 65 nm SRAM array with 6T cells and 8T cells for a nominal supply voltage of 1.2 V. When the supply voltage drops below 1.2 V the failure probability increases significantly. Obviously, the behavior is different for 6T and 8T cells. The overall analysis of the resilience key parameters (Q_{crit} , SVNM, read delay, WTV, and V_{DD}) shows that the variability increases rapidly as

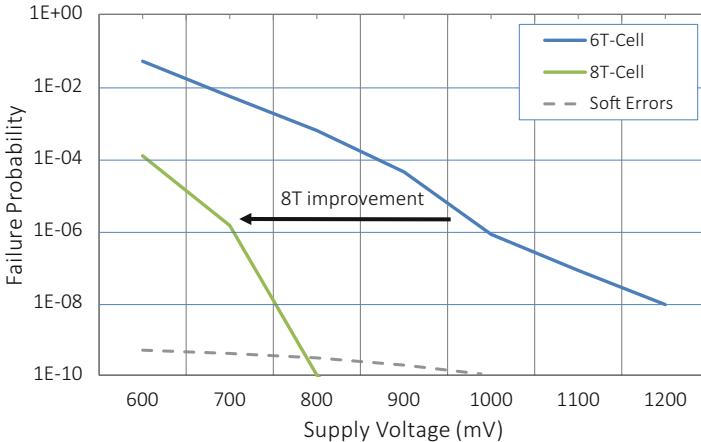


Fig. 10 Memory failure probability (65 nm technology) [1]

technology is scaled down. Investigations considering the failure probabilities of memories (SRAMs, DRAMs) in a system context are described in chapter “Design of Efficient, Dependable SoCs Based on a Cross-Layer-Reliability Approach with Emphasis on Wireless Communication as Application and DRAM Memories”.

5 Architecture Level Analysis and Countermeasures

5.1 Instruction Vulnerability

Due to the wide variety in functionality and implementation of different application softwares as well as changes in the system and application workload depending on the application domain and user, a thorough yet sufficiently abstracted quantification of the dependability of individual applications is required. Even though all application software on a specific system operate on the same hardware, they use the underlying system differently, and exhibit different susceptibility to errors. While a significant number of software applications can tolerate certain errors with a relatively small impact on the quality of the output, others do not tolerate errors well. These types of errors, as well as errors leading to system crashes, have to be addressed at the most appropriate system layer in a cost-effective manner. Therefore, it is important to analyze the effects of errors propagating from the device and hardware layers to all the way up to the application layer, where they can finally affect the behavior of the system software or the output of the applications, and, therefore, become visible to the user. This implies different usage of hardware components, e.g., in the pipeline, as well as different effects of masking at the software layers while considering individual application accuracy requirements.

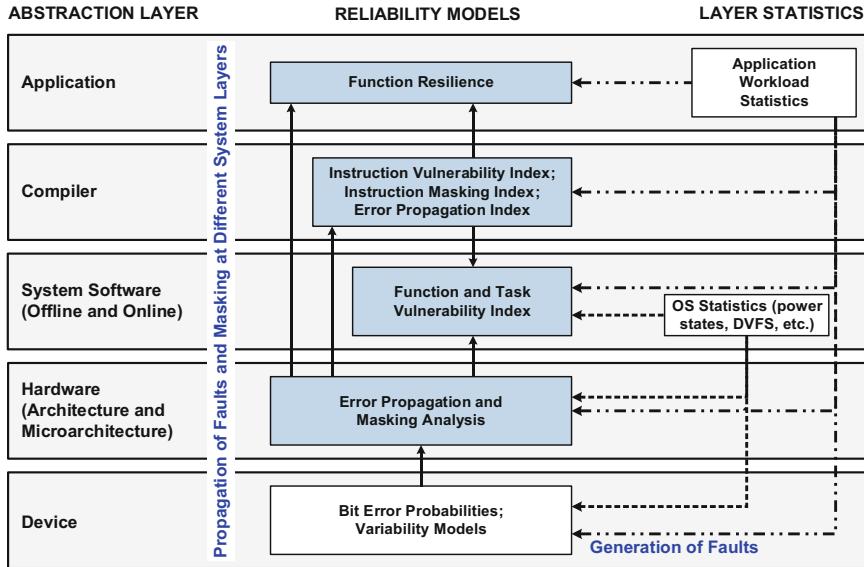


Fig. 11 Cross-layer reliability modeling and estimation: an instantiation of the RAP model from the application software's perspective

These different aspects have to be taken into account in order to accurately quantify the susceptibility of an application towards errors propagating from the lower layers.

An overview of the different models as well as their respective system layer is shown in Fig. 11 [30]. A key feature is that the software layer models consider the lower layer information while being able to provide details at the requested granularity (e.g., instruction, function, or application). To achieve that, relevant information from the lower layers has to be propagated to the upper layers for devising accurate reliability models at the software layer. As the errors originate from the device layer, a bottom-up approach is selected here. Examples for important parameters at the *hardware layer* are fault probabilities (i.e., $P_E(c)$) of different processor components ($c \in C$), which can be obtained by a gate-level analysis, as well as spatial and temporal vulnerabilities of different instructions when passing through different pipeline stages (i.e., IVI_{ic}). At the *software layer*, for instance, control and data flow information has to be considered as well as separation of critical and non-critical instructions. In addition, decisions at the OS layer (e.g., DVFS levels, mapping decisions) and application characteristics (e.g., pipeline usage, switching activity determined by data processed) can have a significant impact on the hardware. Towards that, different models have been developed on each layer and at different granularity as shown in Fig. 11. The individual models are discussed briefly in the following.

One building block for quantifying the vulnerability of an application is the *Instruction Vulnerability Index (IVI)* [22, 24]. It estimates the spatial and temporal

vulnerabilities of different types of instructions when passing through different microarchitectural components/pipeline stages $c \in C$ of a processor. Therefore, unlike state-of-the-art program level metrics (like the program vulnerability factor: PVF [32]) that only consider the program state for reliability vulnerability estimation, the *IVI* considers the probability that an error is observed at the output $P_E(c)$ of different processor components as well as their area A_c .

$$IVI_i = \frac{\sum_{\forall c \in C} IVI_{ic} \cdot A_c \cdot P_E(c)}{\sum_{\forall c \in C} A_c}$$

For this, the vulnerability of an instruction i in a distinct microarchitectural component c has to be estimated:

$$IVI_{ic} = \frac{v_{ic} \cdot \beta_{c(v)}}{\sum_{\forall c \in C} \beta_c}$$

The IVI_{ic} is itself based on an analysis of the vulnerable bits $\beta_{c(v)}$ representing the spatial vulnerability (in conjunction with A_c) as well as an analysis of the normalized vulnerable period v_{ic} representing the temporal vulnerability. Both capture the different residence times of instructions in the microarchitectural components (i.e., single vs. multi-cycle instructions) as well as the different usage of components (e.g., adder vs. multiplier) while combining information from the hardware and software layers for an accurate vulnerability estimation. An example for different spatial and temporal vulnerabilities is shown in Fig. 12a: Comparing an “add”- with a “load”-instruction, the “load” additionally uses the data cache/memory component (thus having a higher spatial vulnerability) and might also incur multiple stall cycles due to the access to the data cache/memory (thus having a higher temporal vulnerability).

The *IVI* can further be used for estimating the vulnerabilities of functions and complete application softwares. An option for a more coarse-grained model at the function granularity is the *Function Vulnerability Index (FVI)*. It models the

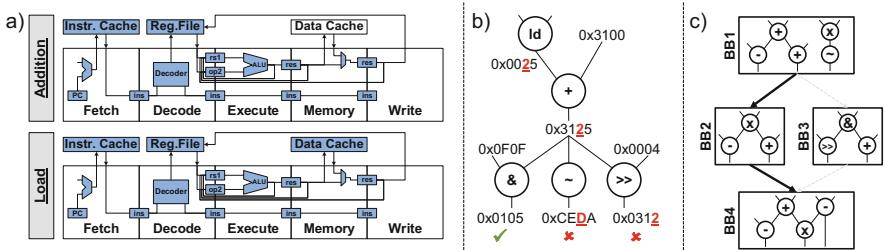


Fig. 12 (a) Temporal and spatial vulnerabilities of different instructions executing in a processor pipeline; (b) Examples for error propagation and error masking due to data flow; (c) Example for error masking due to control flow

vulnerability of a function as the weighted average of its susceptibility towards application failures and its susceptibility towards giving an incorrect output. In order to achieve this, critical instructions (i.e., instructions potentially causing application failures) and non-critical instructions (i.e., instructions potentially causing incorrect application outputs) are distinguished.

The quantification of the error probability provided by the *FVI* is complemented by capturing the masking properties of an application. The *Instruction Error Masking Index (IMI)* [31] estimates the probability that an error at instruction i is masked until the last instruction of all of its successor instruction paths. At the software layer, this is mainly determined by two factors: (a) Masking due to control flow properties, where a control flow decision might lead to an erroneous result originating from instruction i not being used (see example in Fig. 12c); (b) Masking due to data flow properties, which means that a successor instruction might mask an error originating from i due to its instruction type and/or operand values (e.g., the “and”-instruction in Fig. 12b). On the microarchitectural layer, further masking effects may occur due to an error within a microarchitectural component being blocked from propagating further when passing through different logic elements.

Although masking plays an important role, there are still significant errors which propagate to the output of a software application. To capture the effects of an error not being masked and quantify the consequences of its propagation, the *Error Propagation Index (EPI)* of an instruction can be used [31]. It quantifies the error propagation effects at the instruction granularity and provides an estimate of the extent (e.g., number of program outputs) an error at an instruction can affect the output of a software application. This is achieved by analyzing the probability that an error becomes visible at the program output (i.e., its non-masking probability) by considering all successor instructions of a given instruction i . An example of an error propagating to multiple instructions is shown in Fig. 12b.

An alternative for estimating the software dependability at the function granularity is the *Function Resilience* model [23], which provides a probabilistic measure of the function’s correctness (i.e., its output quality) in the presence of faults. In order to avoid exposing the software application details (as it is the case for *FVI*), a black-box model is used for estimating the function resilience. It considers two basic error types: Incorrect Output of an application software (also known as Silent Data Corruption) or Application Failure (e.g., hangs, crashes, etc.). Modeling Function Resilience requires error probabilities for basic block outputs¹ and employs a Markov Chain technique; see details in [23].

As timely generation of results plays an important role, for instance, in real-time systems, it is not only important to consider the functional correctness (i.e., generating the correct output) of a software application, but also to account for the timing correctness (i.e., whether the output is provided in time or after the

¹One potential method to obtain these error probabilities is through fault-injection experiments in the underlying hardware during the execution of these basic blocks

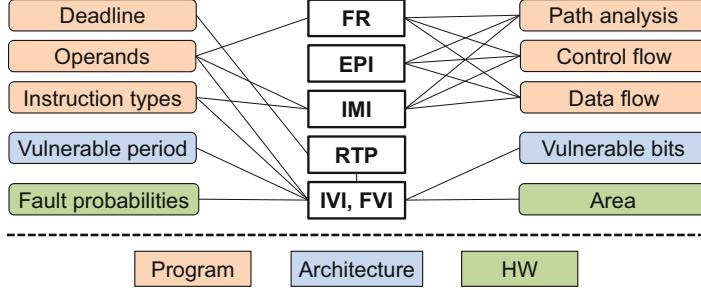


Fig. 13 Composition and focus of the different modeling approaches

deadline). This can be captured via the *Reliability-Timing Penalty (RTP)* model [25]. It is defined as the linear combination of functional reliability and timing reliability:

$$RTP = \alpha \cdot R + (1 - \alpha) \cdot miss_rate$$

where R is the reliability penalty (which can be any reliability metric at function granularity like *FVI* or *Function Resilience*) and *miss_rate* represents the percentage of deadline misses for the software application. Via the parameter α ($0 \leq \alpha \leq 1$), the importance of the two components can be determined: if α is closer to 0, the timing reliability aspect is given a higher importance; when α is closer to 1, the functional reliability aspect is highlighted. The tradeoff formulated by the *RTP* is particularly helpful when selecting appropriate mitigation techniques for errors affecting the functional correctness, but which might have a significant time-wise overhead.

A summary of the different modeling approaches discussed above is shown in Fig. 13, where the main factors and corresponding system layers are highlighted.

5.2 Data Vulnerability Analysis and Mitigation

A number of approaches to analyze and mitigate soft errors, such as ones introduced by memory bit flips or logic errors in an ALU, rely on annotating *sections of code* as to their vulnerability to bit flips [2]. These approaches are relatively straightforward to implement, but regularly fail to capture the *context* of execution of the annotated code section. Thus, the worst-case error detection and correction overhead applies to all executions of, e.g., an annotated function, no matter what the relevance of the data processed within that function to the execution of the program (stability or quality of service effects) may be.

The SPP 1500 Program project FEHLER [29], in contrast, bases its analyses and optimizations on the notion of *data vulnerability* by performing joint *code and data flow analyses*. Here, the foremost goal is to ensure the stability of program

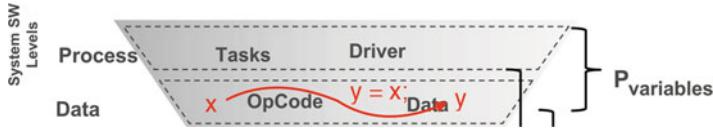


Fig. 14 Horizontal propagation of an error in the RAP model

execution while allowing a system designer to trade the resulting quality of service of a program for optimizations of different non-functional properties such as real-time adherence and energy consumption.

However, analyses on the level of single bit-flips are commonly too fine-grained for consideration in a compiler tool flow. Rather, the level of analysis provided by FEHLER allows the developer to introduce semantics of error handling above the level of single bit-flips. In the upper half of the RAP model hourglass [9], this corresponds to the “data” layer.

The seminal definition of the RAP model provides the notion of a set of bits that belong to a word of data. This allows the minimum resolution of error annotations to represent basic C data types such as `char` or `int`.² In addition, FEHLER allows annotations of complex data types implemented as consecutive words in memory, such as C structures or arrays.

In terms of the RAP model, data flow analyses enable the tracking of the effects of bit flips in a different dimension. The analyses capture how a hardware-induced bit error emanating in the lower half of the RAP hourglass propagates to different data objects on the same layer as an effect of arithmetic, logic, and copy operations executed by the software. As shown in Fig. 14, a bit error on the data layer can now *propagate horizontally* within the model to different memory locations. Thus, with progressing program execution, a bit flip can eventually affect more than one data object of an application.

In order to avoid software crashes in the presence of errors, affected data objects have to be classified according to the worst-case impact an error in a given object can have on a program’s execution.

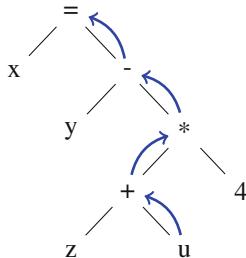
Using a bisecting approach, this results in a binary classification of the worst-case error impact of a data object on a program’s execution. If an error in a data object could result in an application crash, the related piece of data is to be marked as critical to the system stability. An example for this could be a pointer variable which, in case of a bit error, might result in a processor exception when attempting to dereference that pointer. In turn, all other errors are classified as non-critical, which implies that we can ensure that a bit flip in one of these will never result in a system crash.

²Single bit annotations could be realized by either using C bit fields or bit banding memory areas. However, the use of bit fields is discouraged due to portability issues, whereas bit banding is not generally available on all kinds of processors and the compiler possesses no knowledge of aliasing of bit banding areas with regular memory, which would result in more complex data flow analyses.

```
unreliable int x;
reliable int y;
```

Listing 1.1 Reliability type qualifiers in FEHLER

In the FEHLER system, this classification is indicated by reliability type qualifiers, an addition to the C language that allows a programmer to indicate the worst-case effect of errors on a data object [3]. An example for possible annotations is shown in Listing 1.1. Here, the classification is implemented as extensions to the C language in the ICD-C compiler. The `reliable` type qualifier implies that the annotated data object is critical to the execution of the program, i.e., a bit flip in that variable might result in a crash in the worst case, whereas the `unreliable` type qualifier tells the compiler that the worst-case impact of a bit flip is less critical. However, in that case the error can still result in a significant reduction of a program's quality of service.



```
unreliable int u, x;
reliable int y, z;

...
x = y - (z + u) * 4;
```

Listing 1.2 Data flow analysis of possible horizontal error propagation and related AST representation

It is unrealistic to expect that a programmer is able or willing to provide annotations to each and every data object in a program. Thus, the task of analyzing the *error propagation throughout the control and data flow* and, in turn, providing reliability annotations to unannotated data objects, is left to the compiler.

An example for data propagation analysis is shown in Listing 1.2. Here, data flow information captured by the static analysis in the abstract syntax tree is used to propagate reliability type qualifiers to unannotated variables. In addition, this information is used to check the code for *invalid assignments* that would propagate permissible bit errors in unreliable variables to ones declared as reliable. Here, the `unreliable` qualifier of variable `u` propagates to the assignment to the left-hand side variable `x`. Since `x` is also declared `unreliable`, this code is valid.

```

unreliable int u, pos, tmp;
reliable int r, a[10];
u = 10;
r = u;           // invalid assignment
pos = 0;
while ( pos < r ) { // invalid condition
    tmp = r / u;   // invalid division
    a[ pos++ ] = tmp; // invalid memory access
}

```

Listing 1.3 Invalid assignments

Listing 1.3 gives examples for invalid propagation of data from unreliable (i.e., possibly affected by a bit flip) to reliable data objects, which are flagged as an error by the compiler.

However, there are specific data objects for which the compiler is unable to automatically derive a reliability qualifier for. Specifically, this includes input and output data, but also possibly data accessed through pointers for which typical static analyses only provide imprecise results.

The binary classification of data object vulnerability discussed above is effective when the objective is to avoid application crashes. If the quality of service, e.g., measured by the signal-to-noise ratio of a program's output, is of relevance, additional analyses are required.

FEHLER has also been applied to an approximate computing system that utilizes an ALU comprised of probabilistic adders and multipliers [7]. Here, the type qualifiers discussed before are used to indicate if a given arithmetic operation can be safely executed on the probabilistic ALU or if a precise result is required, e.g., for pointer arithmetics. The impact of different error rates on the output of an H.264 video decoder using FEHLER on probabilistic hardware is shown in Fig. 15. Here, lowering the supply voltage results in an increased error probability and, in turn, in more errors in the output, resulting in a reduced QoS as measured by the signal-to-noise ratio of the decoded video frames.

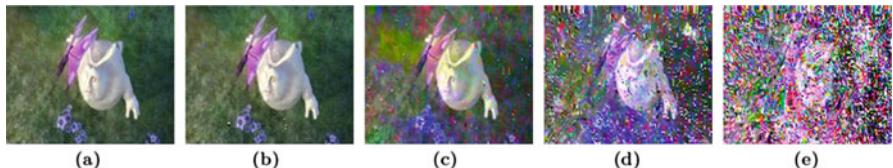


Fig. 15 Effects of different error rated on the QoS of an H.264 video decoder using FEHLER. (a) $V_{DD} = 1.2$ V. (b) $V_{DD} = 1.1$ V. (c) $V_{DD} = 1.0$ V. (d) $V_{DD} = 0.9$ V. (e) $V_{DD} = 0.8$ V

5.3 Dynamic Testing

Architectural countermeasures that prevent errors from surfacing or even only detect their presence come at non-neglectable costs. Whether a specific cost is acceptable or not, in turn, depends on many factors, most prominently criticality. The range of associated costs is also extensive, on one end triple modular redundancy (TMR) or similar duplication schemes such as duplication with comparison (DWC) or on the other end of the spectrum time-multiplexed methods such as online dynamic testing proposed by Gao et al. [5]. In the former examples, the costs directly correlate to the kind of assurance each technique can provide, i.e., TMR can not only continuously monitor a given component like DWC, but it can also mask any detected errors. Using TMR in the right manner, it virtually guarantees the absence of errors, but also comes at a 50% increase in both area and power consumption when compared to DWC.

Whether such cost is sensible or not depends on a complex probabilistic tradeoff with the probability of an error to occur at a specific point in time, and the criticality of an application, on the other hand, also expressed as a probabilistic term, e.g., the maximum tolerable error probability per time, often expressed as failure rate per time λ . While some applications cannot tolerate any errors such as banking transactions (or so we hope), many embedded applications have surprisingly large margins such as applications for entertainment or comfort purposes. For such applications, rather than giving absolute assurances in terms of error detection and masking (e.g., TMR or DWC), temporal limits with confidence levels are far more usable and have much higher utility for the engineering of architectural countermeasures.

Dynamic testing is a probabilistic testing scheme which can exploit such limits as its primary metric is by definition latency detection, that is the time a given dynamic testing configuration requires to detect an error with a given probability. Dynamic testing periodically samples inputs as well as associated outputs of known algorithms implemented in designated components of a SoC in a time-multiplexed fashion. Thereby obtained samples are then recomputed online on a component, the checker core, which is presumed to be more reliable. If the output sample of the device under test (DUT) does not match the recomputed sample, an error on the DUT is assumed. This testing method offers many ways to be tuned towards a specific scenario and to meet particular reliability requirements. By specifying how often a DUT is checked, how many samples per time window are being checked as well as how many such DUTs are checked using the same checker core, effort and the achievable level of assurance can be fine-tuned. Furthermore, depending on the properties of the checker core, even more ways to tailor dynamic testing towards a concrete scenario emerge.

In the presented research as demonstrated in [15], specially hardened Dynamically Reconfigurable Processors (DRPs) have been used to implement the checker functionality (See chapter ‘Increasing Reliability Using Adaptive Cross-Layer Techniques in DRPs’). DRPs are similar to FPGAs as they are reconfigurable

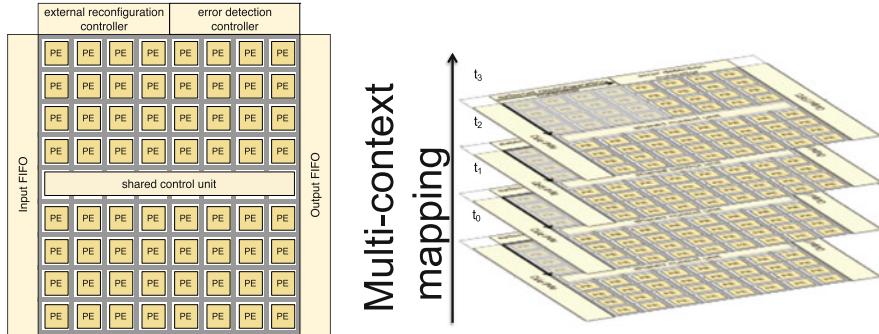


Fig. 16 General DRP structure (left) and temporal application mapping in DRPs (right)

architectures. In terms of functionality, however, they are much closer to many-core architectures, as they consist of an array of processing elements (PE) (Fig. 16 left) which operate on word granularity and possess an instruction concept combined with processor-like cycle-by-cycle internal reconfiguration. Therefore, DRPs do not only allow applications to be mapped spatially like FPGAs but also offer an extensive temporal domain to be used for better area utilization using so-called multi-context application mappings (Fig. 16 right).

For dynamic testing, this means that a DRP as a checker core is more suitable than, e.g., an embedded field programmable gate array (eFPGA) as conventional error detection ensures that the hardened DRP itself is checked regularly during non-checker operation. Furthermore, the high structural regularity also allows workloads to be shifted around on the PE array, adding additional assurances that if a DUT checks out faulty on several different PEs, the likelihood of false-positives decreases. Most importantly, however, it does not need to be dedicated to dynamic testing, but dynamic testing could be executed alongside regular applications. In turn, this, of course, also means that checker computations take longer to complete, reducing the number of samples computed per time window.

While this adaptability makes DRPs and dynamic testing an interesting match, for this combination to be useful, realistic assumptions about the error probability P are essential. If we can obtain P through, e.g., the RAP model, there are two significant advantages. Firstly, P is not constant over the lifetime of a SoC and knowledge about its distribution can help reduce testing efforts with dynamic testing. At a less error-prone time, dynamic testing allows for trade-offs such as increased time to react to errors if the error is unlikely enough to only affect a small minority of devices. Secondly, for an error with probability P to have any effect, it needs to be observable, and, thus, for all practical purposes we equate P and observation probability q which then allows us to use P to fine-tune dynamic testing to a resource minimum while meeting an upper bound for detection latency.

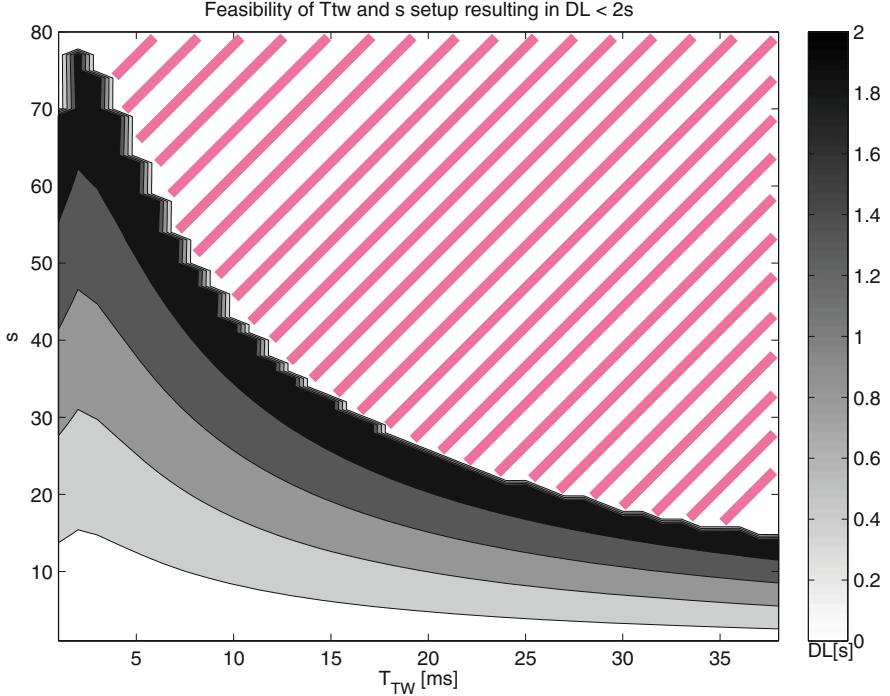


Fig. 17 Feasibility region for an error to be detected within 2 s, with $N = 4$ running at 100 MHz, an observable error probability of $P = 10^{-5}$, a reconfiguration and setup overhead of 1 ms and different scaling factors s and time windows T_{TW}

Assume a dynamic testing setup with $N = 4$ DUTs, a reconfiguration and general setup overhead of $T_{OV} = 1$ ms and time windows of $T_{TW} = \{1, \dots, 40\text{ ms}\}$, one round of checking requires between 8 ms and 44 ms for all DUTs. Now let s denote the scaling factor by which the temporal domain is used to map the checker functionality, e.g., $s = 3$ means using a third of the original spatial resources and, instead, prolonging the time to compute one sample by a factor of three. Consequently, a scaling factor of $s = 3$ divides the number of samples checked within one time window by three.

Now consider Fig. 17 which depicts the feasibility region by time window size T_{TW} and scaling factor s . The area which is not marked by the red dashes means that in this region, a reliability goal of a maximum detection latency DL of 2 s can be guaranteed with two-sigma confidence. However, apart from all adaptability, dynamic testing may be also waived or reduced to a minimum during times of low error probability (after early deaths in the bath tub curve). Ideally, we would only start with serious testing once the error probability is high enough to be concerned and then also only as much that the expected detection latency is within the prescribed limit. In other words, without detailed knowledge of vulnerability P , the only possibility is to guess the probabilities and add margins. If, however, P

can be estimated close enough, dynamic testing using DRPs as checker core offers a near resource optimal time and probability based technique.

Furthermore, if the characteristics of P and its development over time is understood well enough, dynamic testing could pose an alternative to DWC or even TMR for certain applications. The better P can be modeled, the smaller the margins become that have to be added to give assurances with high enough confidence. Especially for more compute intensive applications without 100% availability requirements, dynamic testing could serve as a low-cost alternative.

6 Application-Level Optimization—Autonomous Robot

Autonomous transportation systems are continuously advancing and become increasingly present in our daily lives [37]. Due to their autonomous nature, for such systems often safety and reliability are a special concern—especially when they operate together with humans in the same environment [11]. In [13], we studied the effect of soft errors in the data cache of a two-wheeled autonomous robot. The robot acts as a transportation platform for areas with narrow spacing. Due to safety reasons, the autonomous movement of the robot is limited to a predefined path. A red line on the ground, which is tracked by a camera mounted on the robot, defines the path which the robot should follow.

Since we want to study the impact of single event upsets in the data cache, the whole system memory hierarchy including accurate cache models is included in the simulation environment. We utilized in this example Instruction Set Simulation (ISS) to emulate the control SW, which consists of three main tasks: (1) the extraction of the red line from the camera frames, (2) the computation of orientation and velocity required to follow the line, and (3) evaluation of the sensor data to control the left and right motor torques to move the robot autonomously. The last task has especially hard real-time constraints because the robot must constantly be balanced. In this setup we used a fault model based on neutron particle strike induced single event upsets as shown in Sect. 4.1.1. Further, to make the fault-injection experiment feasible we used Mixture Importance Sampling to avoid simulation of irrelevant scenarios [14].

In this experiment the processor of the robot is modeled in a 45 nm technology together with a supply voltage of 0.9 V. Further, we assume a technology dependent parameter Q_s of 4.05 fC and a flux Φ of 14 Neutrons/cm²/h (New York, Sea Level) [20, 36]. In our fault injection experiment we start with an unprotected, unhardened data cache to find the maximal resilience of the application to soft errors.

Figure 18 depicts traces of position, velocity, and orientation of the robot while it autonomously follows a line for 10 s. The injected faults lead to two types of changed system behavior:

1. strong deviations in orientation and velocity where the robot eventually loses its balance (crash sites are marked with crosses in the $x - y$ plane graph).

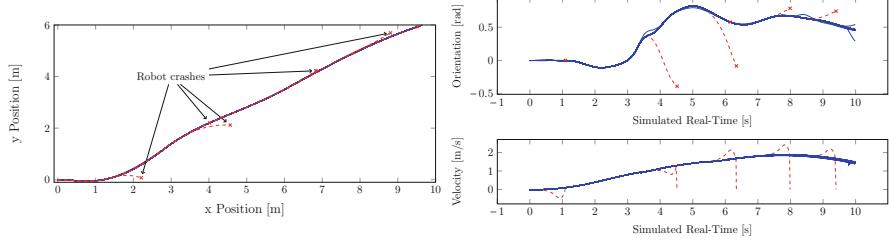


Fig. 18 Robot movement in $x - y$ plane together with velocity and orientation angle. Dashed lines indicate crashes by CPU stalls

2. slight deviations, e.g., temporarily reduced velocity or changed orientation, where the robot still rebalances due to its feed-back control loop and still reaches its goal at the end of the line.

Further investigations showed, that for the more severe failures in (1) the simulator always reported a CPU stall. This led finally to the crash of the robot in the simulation as the balancing control was not executed any longer. Such failures are much more severe compared to (2). Still, such problems are detectable on microarchitectural level. In (2), silent data corruption (SDC) in the control algorithm happens. SDC is a severe problem for an application because it typically cannot easily be detected. Interestingly for our experiment, the algorithm shows a very high fault tolerance and often moves the robot back on its original path. This, possibly, guarantees a safe movement dependent on how narrow the robot's movement corridor is specified. The inherent error resilience of the application, thus, mitigates the SDC effect.

Based on these insights an overall cross-layer design approach for this application could look as follows: The severe crashing failures in (1) are handled by additional protection solution which detects such problems and causes a restart of the application and hence the balancing control. One typical solution to this problem is the addition of a watchdog timer to the system or a small monitoring application to key state variables of the control loop. The silent data corruption in (2) can be accepted in a certain frequency and limit according to the overall system constraints. Hence, further system design techniques and resilience actuators can be used to tune this into the required limits. This is further described in chapter 'Cross-Layer Resilience Against Soft Errors: Key Insights'.

A further use case for applying the RAP model to the cross-layer evaluation of temperature effects in MPSoC systems is presented in chapter 'Thermal Management and Communication Virtualization for Reliability Optimization in MPSoCs'.

References

1. Chang, I., Mohapatra, D., Roy, K.: A priority-based 6t/8t hybrid SRAM architecture for aggressive voltage scaling in video applications. *Trans. Circuits Syst. Video Technol.* **21**(2), 101–112 (2011)
2. de Kruijf, M., Nomura, S., Sankaralingam, K.: Relax: an architectural framework for software recovery of hardware faults. In: Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10), pp. 497–508. ACM, New York (2010). <https://doi.org/10.1145/1815961.1816026>
3. Engel, M., Schmoll, F., Heinig, A., Marwedel, P.: Unreliable yet useful—reliability annotations for data in cyber-physical systems. In: Informatik 2011, Berlin, Germany, p. 334 (2011)
4. Evans, A., Nicolaidis, M., Wen, S.J.: Riif—reliability information interchange format. In: IEEE 18th International On-Line Testing Symposium (IOLTS) (2012)
5. Gao, M., Chang, H.M., Lisherness, P., Cheng, K.T.: Time-multiplexed online checking. *IEEE Trans. Comput.* **60**(9), 1300–1312 (2011)
6. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. CAD of Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
7. Heinig, A., Mooney, V.J., Schmoll, F., Marwedel, P., Palem, K., Engel, M.: Classification-based improvement of application robustness and quality of service in probabilistic computer systems. In: Proceedings of the 25th International Conference on Architecture of Computing Systems (ARCS'12), pp. 1–12. Springer, Berlin (2012)
8. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWEEK '11 Seventh Embedded Systems Week, Taipei, Taiwan, 9–14 October 2011, pp. 69–78 (2011). <https://doi.org/10.1145/2039370.2039384>
9. Herkersdorf, A., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectron. Reliab.* **54**(6–7), 1066–1074 (2014). <https://doi.org/10.1016/j.microrel.2013.12.012>
10. Ibe, E., et al.: Spreading diversity in multi-cell neutron-induced upsets with device scaling. In: IEEE Custom Integrated Circuits Conference (CICC) (2006)
11. ISO: ISO/PAS 21448: Road vehicles—Safety of the intended functionality. International Organization for Standardization, Geneva (2019)
12. Kleeberger, V., Weis, C., Schlichtmann, U., Wehn, N.: Circuit resilience roadmap. In: Circuit Design for Reliability, pp. 121–143. Springer, Berlin (2015)
13. Kleeberger, V., et al.: A cross-layer technology-based study of how memory errors impact system resilience. *IEEE Micro.* **33**(4), 46–55 (2013)
14. Kleeberger, V., et al.: Technology-aware system failure analysis in the presence of soft errors by mixture importance sampling. In: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS) (2013)
15. Kühn, J.M., Schweizer, T., Peterson, D., Kuhn, T., Rosenstiel, W., et al.: Testing reliability techniques for SOCS with fault tolerant CGRA by using live FPGA fault injection. In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 462–465. IEEE, New York (2013)
16. Lee, S., Baeg, S., Reviriego, P.: Memory reliability model for accumulated and clustered soft errors. *IEEE Trans. Nucl. Sci.* **58**(5), 2483–2492 (2011)
17. Lin, D., Hong, T., Li, Y., Fallah, F., Gardner, D.S., Hakim, N., Mitra, S.: Overcoming post-silicon validation challenges through quick error detection (QED). In: Design, Automation and Test in Europe (DATE 13), Grenoble, France, 18–22 March 2013, pp. 320–325 (2013). <https://doi.org/10.7873/DATE.2013.077>

18. Millman, S.D., McCluskey, E.J.: Detecting bridging faults with stuck-at test sets. In: Proceedings International Test Conference 1988, Washington, D.C., USA, September 1988, pp. 773–783 (1988). <https://doi.org/10.1109/TEST.1988.207864>
19. Modarres, M., Kaminskiy, M., Krivtsov, V.: Reliability Engineering and Risk Analysis: A Practical Guide. CRC Press, New York (1999)
20. Mukherjee, S.: Architecture design for soft errors. Morgan Kaufmann, Burlington (2011)
21. Quinn, H.M., De Hon, A., Carter, N.: CCC visioning study: system-level cross-layer cooperation to achieve predictable systems from unpredictable components. Tech. rep., Los Alamos National Laboratory (LANL) (2011)
22. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. *IEEE Trans. CAD Integr. Circuits Syst.* **33**(11), 1597–1610 (2014). <https://doi.org/10.1109/TCAD.2014.2341894>
23. Rehman, S., Shafique, M., Aceituno, P.V., Kriebel, F., Chen, J., Henkel, J.: Leveraging variable function resilience for selective software reliability on unreliable hardware. In: Design, Automation and Test in Europe (DATE 13), Grenoble, France, 18–22 March 2013, pp. 1759–1764 (2013). <https://doi.org/10.7873/DATE.2013.354>
24. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Reliable software for unreliable hardware: embedded code generation aiming at reliability. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, Taiwan, 9–14 October 2011, pp. 237–246 (2011). <https://doi.org/10.1145/2039370.2039408>
25. Rehman, S., Toma, A., Kriebel, F., Shafique, M., Chen, J., Henkel, J.: Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations. In: 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2013), Philadelphia, PA, USA, 9–11 April 2013, pp. 273–282 (2013). <https://doi.org/10.1109/RTAS.2013.6531099>
26. Robinson, W., Alles, M., Bapty, T., Bhuvva, B., Black, J., Bonds, A., Massengill, L., Neema, S., Schrimpf, R., Scott, J.: Soft error considerations for multicore microprocessor design. In: IEEE International Conference on Integrated Circuit Design and Technology, pp. 1–4. IEEE, New York (2007)
27. Savino, A., Di Carlo, S., Vallero, G., Politano, G., Gizopolous, D., Evans, A.: RIIF-2—toward the next generation reliability information interchange format. In: IEEE 22nd International On-Line Testing Symposium (IOLTS) (2016)
28. Schlichtmann, U., et al.: Connecting different worlds—technology abstraction for reliability-aware design and test. In: Design, Automation & Test in Europe Conference & Exhibition (DATE 2014), Dresden, Germany, 24–28 March 2014, pp. 1–8 (2014). <https://doi.org/10.7873/DATE.2014.265>
29. Schmoll, F., Heinig, A., Marwedel, P., Engel, M.: Improving the fault resilience of an H.264 decoder using static analysis methods. *ACM Trans. Embed. Comput. Syst.* **13**(1s), 31:1–31:27 (2013). <https://doi.org/10.1145/2536747.2536753>
30. Shafique, M., Axer, P., Borchert, C., Chen, J., Chen, K., Döbel, B., Ernst, R., Härtig, H., Heinig, A., Kapitza, R., Kriebel, F., Lohmann, D., Marwedel, P., Rehman, S., Schmoll, F., Spinczyk, O.: Multi-layer software reliability for unreliable hardware. *it—Inf. Technol.* **57**(3), 170–180 (2015). <https://doi.org/10.1515/itit-2014-1081>
31. Shafique, M., Rehman, S., Aceituno, P.V., Henkel, J.: Exploiting program-level masking and error propagation for constrained reliability optimization. In: The 50th Annual Design Automation Conference 2013 (DAC '13), Austin, TX, USA, May 29–June 07 2013, pp. 17:1–17:9 (2013). <https://doi.org/10.1145/2463209.2488755>
32. Sridharan, V., Kaeli, D.R.: Eliminating microarchitectural dependency from architectural vulnerability. In: 15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14–18 February 2009, Raleigh, North Carolina, USA, pp. 117–128 (2009). <https://doi.org/10.1109/HPCA.2009.4798243>
33. TelCordia Technologies: Reliability Prediction Procedure for Electronic Equipment, SR-332 (2016)

34. Wirth, G., Vieira, M., Neto, E., Kastensmidt, F.: Generation and propagation of single event transients in CMOS circuits. In: IEEE Design and Diagnostics of Electronic Circuits and systems (2006)
35. Wunderlich, H.J., Holst, S.: Generalized fault modeling for logic diagnosis. In: Models in Hardware Testing—Lecture Notes of the Forum in Honor of Christian Landrault. Springer, Berlin (2010)
36. Zhang, M., Shanbhag, N.: Soft-error-rate-analysis (sera) methodology. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **25**(10), 2140–2155 (2006)
37. Ziegler, J., et al.: Making Bertha drive—an autonomous journey on a historic route. *IEEE Intell. Transp. Syst. Mag.* **6**(2), 8–20 (2014)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part I

Cross-Layer from Operating System to Application

Horst Schirmeier

In the embedded systems domain—particularly in highly competitive areas such as the automotive industry—per-device monetary cost reduction is often a first-class optimization goal. The prevailing approach to keeping costs low is to implement more and more functionality in software instead of hardware, with side effects also on other non-functional properties such as total system weight, energy consumption or in-system upgradability. However, this trend towards software shines in the disconcerting light of ever-increasing soft-error rates: As the industry moves towards single-digit nanometer semiconductor structure sizes, the circuits' susceptibility to soft errors continuously increases—unless countered with costly hardware measures that can diminish the gains achieved through scaling.

In consequence, embedded software must cope with increasingly unreliable hardware. In the stack of software layers, the operating system is the layer closest to the hardware, and figuratively the first line of defense against soft errors that are on their way of propagating to the application layer. But as software countermeasures against unreliable hardware tend to be even more costly than their hardware counterparts, they must be used sparingly and tailored for a particular application and use-case scenario.

The three chapters within this area of the book address this cross-layer connection between the operating system and the application layer, and offer different approaches of constructing a reliable system from unreliable hardware components that exhibit erroneous behavior triggered by soft errors.

The first chapter by Engel and Marwedel addresses the problem of error-correction overhead in the context of embedded real-time systems (chapter “Soft Error Handling for Embedded Systems using Compiler-OS Interaction”). The described FEHLER approach introduces *error semantics*, which provides information about the criticality of data objects. A combination of explicit source-code

H. Schirmeier (✉)

Embedded System Software Group, TU Dortmund, Dortmund, Germany

e-mail: horst.schirmeier@tu-dortmund.de

annotations and static data-flow analysis, which derives information for other, non-annotated variables, is used on the application layer to generate a database that allows to classify data objects at runtime. The gathered information is handed across layers to the operating system: When an error is detected—by a mechanism outside of FEHLER’s scope—the proposed system-software stack can use this classification database to assess the actual damage, and to flexibly choose one of several possible error-handling methods. Among the information incorporated in this decision is the current real-time scheduler’s state, e.g., whether immediately scheduling an error-correction task would cause a deadline miss and should therefore be either delayed or completely skipped. This chapter also introduces the concept of the *Reliable Computing Base* (RCB), a part of the system that is critical in ensuring that the error-handling mechanism is effective, and that must not be affected by soft errors itself.

The second chapter, contributed by Rambo and Ernst, describes how to achieve the goal of application-specific and selective fault tolerance at a much coarser granularity (chapter “*ASTEROID* and the Replica-Aware Co-scheduling for Mixed-Criticality”). Set in a scenario with a given set of mixed-critical applications, the *ASTEROID* approach requires a manual criticality classification of application tasks—information that, similar to the FEHLER approach, is passed cross-layer from the application to the operating system. Critical tasks are executed redundantly by the *Romain* system service, exploiting future manycore platforms for the increased system load, and coexist with non-critical tasks. Unlike FEHLER, the *ASTEROID* approach also comprises a concrete error-detection solution: a microarchitecture-level pipeline fingerprinting mechanism that allows *Romain* to compare replicas with low overhead, facilitated through a cross-layer design involving both the hardware and operating-system layers. Quantifying the minimized overhead, the chapter puts a special focus on the performance of replicated execution, introducing a replica-aware co-scheduling strategy for mixed-critical applications that outperforms the state of the art.

The third chapter by Schirmeier et al. describes the *DanceOS* approach, focusing on application-specific operating-system construction techniques, similar to FEHLER aiming at fine-grained fault-tolerance approaches (chapter “Dependability Aspects in Configurable Embedded Operating Systems”). The chapter first investigates the general reliability limits of static system-software stacks, and demonstrates a technique to reduce the proverbial “attack surface” of a newly constructed, AUTOSAR-compliant operating system by exploiting knowledge from static task descriptions. By additionally applying classic fault-tolerance techniques to the remaining dynamic kernel data structures, the *DanceOS* approach yields a highly reliable software system. The second part of the chapter addresses the problem how a pre-existing, legacy dynamic operating-system codebase can be hardened against soft errors in an application-specific way. Using programming-language and compiler-based program transformation techniques—in particular aspect-oriented programming—this part shows how generic fault-tolerance mechanisms can be encapsulated in separate modules, and applied to the most critical data structures identified, e.g., by fault-injection experiments. In both operating-system scenarios—

and similar to FEHLER and ASTEROID—the application drives the fault-tolerance hardening process: In the AUTOSAR-compliant static OS scenario, statically known structural application knowledge is handed to an operating-system tailoring and minimization process; in the dynamic legacy-OS scenario, the application’s runtime behavior while being exposed to injected faults provides the information relevant for targeted, selective fault-tolerance hardening. In the third and last part, the chapter expands the considered fault model to whole-system power outages, and demonstrates that persistent memory—combined with transactional memory—can be used for state conservation.

To conclude, all three chapters share the common insight that a cross-layer combination of application layer knowledge and operating-system layer fault tolerance—in the case of ASTEROID additionally involving the hardware layer—enables overhead minimization and optimal, application-specific hardening against soft errors.

Soft Error Handling for Embedded Systems using Compiler-OS Interaction



Michael Engel and Peter Marwedel

1 New Requirements for Fault Tolerance

The ongoing downscaling of semiconductor feature sizes in order to integrate more components on chip and to reduce the power and energy consumption of semiconductors also comes with a downside. Smaller feature sizes also lead to an increasing susceptibility to *soft errors*, which affect data stored and processed using semiconductor technology. The amount of disturbance required to cause soft errors, e.g. due to the effects of cosmic particles or electromagnetic radiation on the semiconductor circuit, has declined significantly over the last decades, thus increasing the probability of soft errors affecting a system's reliable operation.

2 Semantics of Errors

Traditionally, system hardening against the effects of soft errors was implemented using hardware solutions, such as error-correcting code circuits, redundant storage of information in separate memories, and redundant execution of code on additional functional units or processor cores. These protection approaches share the property that they protect all sorts of data or code execution, regardless of the requirement to

M. Engel (✉)

Department of Computer Science, Norwegian University of Science and Technology (NTNU),
Trondheim, Norway

e-mail: michael.engel@ntnu.no

P. Marwedel

Department of Computer Science, TU Dortmund, Dortmund, Germany
e-mail: peter.marwedel@tu-dortmund.de

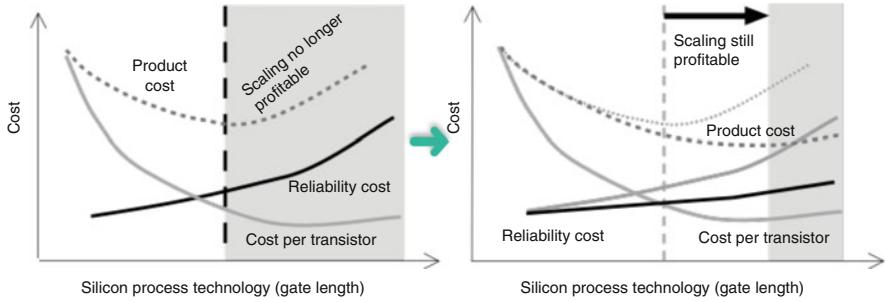


Fig. 1 Enabling profitable scaling using software-based fault tolerance [1]

actually enforce protection. In other terms, they do not possess knowledge about the *semantics* of data and code related to the reliable operation of a system.

As shown by Austin [1], reproduced on the left-hand side of Fig. 1, together with a rising probability of soft errors, this results in a significantly increasing overhead in hardware required to implement error protection. As technology progresses, at a certain point in time, the cost of this overhead will exceed the savings due to the utilization of more recent semiconductor technologies, resulting in diminishing returns that render the use of these advancements unattractive.

The fundamental idea applied by the FEHLER project is to reduce the amount of error handling required in a system by introducing semantic knowledge. We enable a system's software to dynamically decide at runtime whether an error that occurred is critical to the system's operation, e.g. it might result in a crash in the worst case, or is not critical, e.g. an error might only result in an insignificant disturbance of a system's output. In turn, this enables the system to handle only *critical errors* and ignore the others. This *flexible error handling* results in a significantly reduced hardware overhead for implementing fault tolerance, which leads to an increased profitability window for semiconductor scaling, as shown on the right-hand side of Fig. 1.

One important consideration when designing such a selective approach to fault tolerance is which components of a system actually have to be protected from errors. Inspired by the concept of the trusted computing base in information security, we introduced the *Reliable Computing Base* (RCB) [6] to indicate the hardware and software components of a system that are critical in ensuring that our flexible error handling approach is effective.

Accordingly, we define the RCB as follows:

The Reliable Computing Base (RCB) is a subset of software and hardware components that ensures the reliable operation of software-based fault-tolerance methods. Hardware and software components that are not part of

(continued)

the RCB can be subject to uncorrected errors without affecting the program's expected results.

To design efficient fault-tolerant systems, it is essential to *minimize* the size of the reliable computing base. In the case of FEHLER, this implies that the number and size of hardware and software components required to ensure that upcoming *critical errors* will be corrected are to be reduced as far as possible.

Commonly, code-based annotations such as [13] are used to indicate sections of code to be protected against errors regardless of the data objects handled by that code. This implies an overhead in runtime—protection of the executed section of code applies to all its executions without considering its execution semantics—as well as in programmer effort, since error propagation analyses using control and data flow information would have to consider all data objects handled in annotated code sections. In order to increase the efficiency of this approach, additional manual annotations seem indispensable.

A more efficient approach from a software point of view is to identify the minimal amount of *data objects* that have to be protected against soft errors. Data flow analyses provided by FEHLER allow to determine the worst-case propagation of errors throughout a program's execution, thus determining the precise set of data objects requiring protection against errors. Additional savings at runtime are achieved by employing a microkernel system tailored to exclusively address error handling, leaving the remaining operating system functions to a traditional embedded kernel running on top of it. An analysis of the possible savings for a real-world embedded application is given in Sect. 7.

3 FEHLER System Overview and Semantic Annotations

Based on the observations described above, one central objective of the FEHLER system is to enable the provision of semantics describing the worst-case effects of errors on data objects.

Commonly, the hardware of a system only has very limited knowledge about the semantics of data that it processes.¹ More semantic information, such as the types of data objects, is available on the source code level. However, this information is commonly discarded by the compiler in later code generation and optimization stages when it is no longer required to ensure program correctness. Some of this information can already be utilized to provide error semantics. For example, pointer

¹For example, a processor could distinguish between integer and floating point data due to the use of different registers and instructions to process these, but a distinction between pointer and numeric data is often not possible on machine code level.

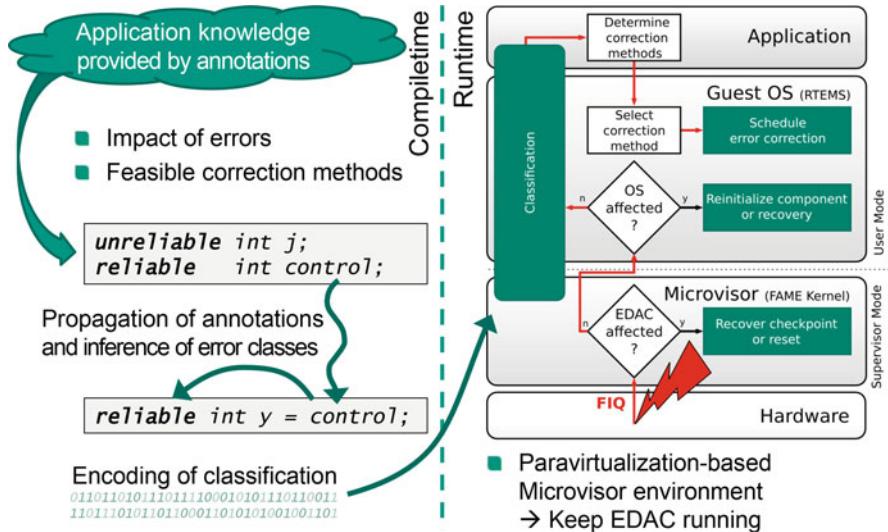


Fig. 2 Interaction of compile time and runtime components of FEHLER

data types and variables influencing the control flow (e.g. used to conditionally execute of code or control loops) are deemed essential in ensuring correct program execution. Accordingly, *static analyses* performed during compile time are able to extract this information.

However, additional information about the relevance of data with regard to the correct behavior of a system in its intended context, e.g. in embedded systems where an incorrect output controlling a peripheral might result in damaging effects, is not expressed *explicitly* in the code. Hence, we have to provide additional information in order to enable static analyses to derive more information about the worst-case criticality of a data object.

This additional semantic information allows the system to classify errors. Data objects which are deemed critical to a program's execution, i.e. may cause the program to crash, are annotated with a `reliable` type qualifier. All objects for which errors in data will result only in an insignificant deviation of the program's behavior in the worst case are provided with an `unreliable` type qualifier.

Classifying data objects into only these two classes is a rather coarse approach. However, as shown later, this minimalistic approach is effective and efficient for systems experiencing normal error rates, i.e. applications not exposed to radiation-rich environments, such as space and aviation systems. Approaches for improved QoS assessment are discussed in Sect. 10.

The interaction of compile time and runtime components of a FEHLER-based system is shown in Fig. 2. Here, the compile time component, realized as a compiler performing static analyses and transformations in addition to code generation, extracts semantic information on the criticality of data objects, analyzes the

program's control and data flow to determine possible error propagation paths and to generate appropriate type qualifiers, and encodes this information along with the generated program binary.

This information lies dormant as long as no error affects the system's operation. Since error detection is outside of the scope of FEHLER, the system is prepared to interface with a number of different error detection methods. In the example in Fig. 2, we assume that a simple hardware mechanism, such as memory parity checks, is employed. When an incorrect parity is detected during a memory access, a special interrupt is raised that informs the system of the error.

Here, our runtime component, the Fault-aware Microvisor Environment (FAME) [11] comes into play. FAME is intentionally restricted to only provide functionality that enables decisions about the necessity of error handling, relegating all other functionality typically found in system software to components outside of the microkernel. This reduced functionality is an additional contribution to RCB minimization. FAME provides a handler for the given error signalization method, which is able to determine the *address of the affected memory location*. As soon as the microkernel is able to ensure that itself is not affected, which can be ensured by RCB analysis and minimization, it determines whether the embedded OS kernel running on top or the application is affected. If this is the case, error handling is initiated. In case of an error affecting the application, FAME consults the semantic information provided by the compile time components and determines if error correction is required or if the error can be safely ignored. Further details of FAME are described in Sect. 6.

Like error detection, specific correction methods are not the focus of FEHLER. Instead, FEHLER is enabled to interface with different standard as well as application-specific correction methods. An example for a standard error correction would be the application of checkpointing and rollback. An application-specific method would be a function that corrects an affected macro block in a video decoder by interpolating its contents from neighboring blocks instead of redecoding the complete video frame, thus saving a considerable amount of compute time.

4 Timing Behavior

Figure 3 shows possible scheduling orders in case of a detected error. In an approach that neglects to use criticality information (“naive approach”), the detection of an error implies an immediate correction action in hardware or software. This potentially time-consuming recovery delays the execution of subsequent program instructions, which may result in a deadline miss.

The flexible approach enabled by FEHLER allows the system to react to an error in a number of different ways. Here, the classifications described above come into play. Whenever an error is detected, the system consults the classifications provided alongside the application (“C” in Fig. 3). This lookup can be performed quickly

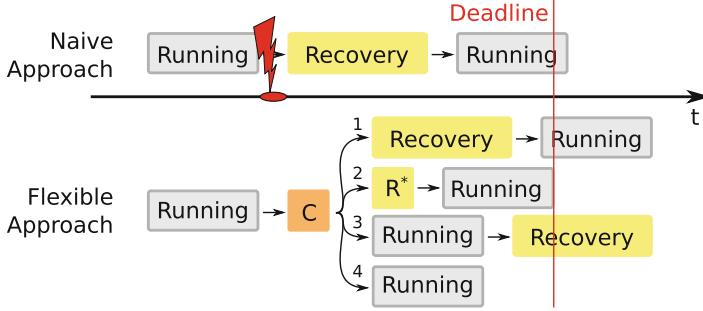


Fig. 3 Different scheduling possibilities for error handling

and provides information on how to handle the error at hand. Specifically, it can be determined *if*, *how*, and *when* the error needs to be corrected:

- *if*: Whether errors have to be handled or not depends mainly on the error impact. If an error has a high impact, error recovery will be mandatory. In contrast, if an error only has a low impact at all, e.g. the color of a single pixel in a frame buffer is disturbed, further handling can be omitted. Handling errors in the latter case will improve the quality of service at the cost of requiring additional compute time. Error impacts are deduced using static analysis methods as described below.
- *how*: Error handling depends on the available error correction methods, the error impact, and the available resources. In FEHLER, commonly a bit-precise correction method such as checkpoint-and-recovery as well as an “ignore” method (case 4 in Fig. 3) doing nothing is available. In addition, the programmer can provide application-specific correction methods, denoted by “R*”. Such a method may be preferable, since it can be faster than the generic correction method provided.
- *when*: Error scheduling can decide when an error correction method has to be scheduled. In a multitasking system, often, the task with the highest priority is executed. Hence, if a high priority task is affected, error correction has to be scheduled immediately (cases 1 and 2). If a low priority task is affected, the high priority task can continue execution and the error handling will be delayed (case 3). In order to enable the mapping of errors to different tasks, a subscriber-based model can be employed [12].

Overall, this flexibility allows a system to improve its real-time behavior in case of errors. While this may not be acceptable for hard real-time systems, the behavior of soft real-time applications, such as media decoding, can be significantly improved. The example of an H.264 video decoder is used in Sect. 7 to evaluate the effectiveness and efficiency of FEHLER.

To enable the flexible handling of errors at runtime, the runtime system requires the provision of detailed, correct meta information about the data objects in the given

application. The static analyses employed to obtain this information are described in the following section.

5 Static Analyses

The correct determination of all data objects critical to a program’s execution, which form part of the RCB, is crucial to ensure that errors threatening to result in a system crash are corrected before they affect its operation.

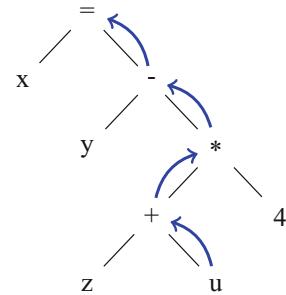
Our static analysis is based on the concept of subtyping [7]. In FEHLER, additional semantic information on the criticality of a data object to the application’s stability is provided by extending the C language using *reliability type qualifiers*. These qualifiers enable a developer to indicate to the static analysis stages whether a data object is deemed critical to a program’s execution (`reliable` classification) or if errors in data will result only in an insignificant deviation of the program’s behavior in the worst case (using the `unreliable` type qualifier).

Accordingly, we have to ensure that reliable data objects must not be modified unpredictably to guarantee that the application will not crash. In contrast, the application can tolerate deviations from the expected values in unreliable data objects.

Rules for the use of our new type qualifiers applied by our static code analysis fall into two groups: *prohibit* and *propagation rules*. Prohibit rules ensure that operations on the annotated data objects are executed error-free, whereas propagation rules reflect the possible propagation of errors from an affected data object to others throughout the control and data flow.

Errors in certain data objects may result in a large deviation in the control flow or even an unintended termination of the application. Prohibit rules ensure that those data objects are annotated with the `reliable` type qualifier; accordingly, errors affecting that data are classified as fatal errors. Data objects serving as a reference to a memory address, i.e. pointers in C, are an important example for this. An error in a pointer that is used for reading will result in either a different address that is read, possibly resulting in the loading of a completely unrelated data object, or even an access to a non-existing memory location, resulting in a processor exception that terminates the application. Pointers used for writing data can result in correct data being written to an unintended memory location, resulting in unexpected error propagation that is especially hard to diagnose. Indexes for arrays behave in a similar way, resulting either in a write to a different array element or, due to the lack of bounds checking for array indexes in C, a write to an arbitrary memory location. Other critical data types include controlling expressions for loops and conditional statements, divisors, branch targets, and function symbols. For details, we refer the reader to the description in [15].

```
unreliable int u, x;
reliable int y, z;
```



...

$x = y - (z + u) * 4;$

Listing 1 Data flow analysis of possible horizontal error propagation and related AST representation

The content of a data object annotated as unreliable may be affected by an uncorrected error. In turn, that error can propagate to other data objects whenever its content is copied or used in an arithmetic or logic expression, as shown in Listing 1. Here, the curved arrows indicate that an error can propagate from one subexpression to the following along the edges of the syntax tree. Accordingly, the content of a resulting data object cannot be considered reliable and thus has to be qualified as unreliable. The dependencies between type qualifiers of different data objects are modeled by the FEHLER propagation rules. In addition to calculations and assignments, other uses of data objects affected by error propagation are the copying of parameters to functions using call-by-value semantics and cast expressions.

```

int step(int x) {
    return x << 2;
}

void main(void) {
    int a, b, c;
    unreliable int w;
    int v;

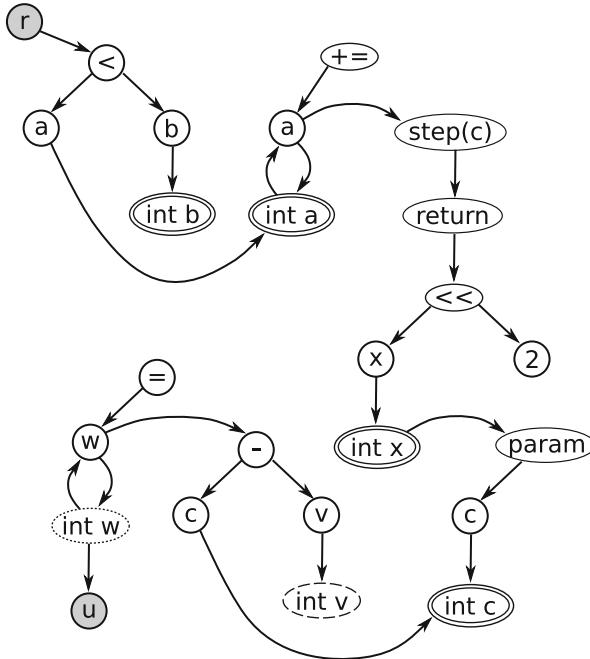
    // Initializations
    // omitted for brevity
    while (a < b)
        a += step(c);

    w = c - v;
}

```

Listing 2 Code example and related type deduction graph

Propagation rules not only help in detecting erroneous data flow from unreliable to reliable data objects, but also reduce the overhead required by the programmer



to create annotations. Since it is unrealistic to expect that each and every data object in a complex program will be annotated manually, our static analyses help to deduce correct type qualifiers for unannotated data. This deduction is enabled by the construction of a *type deduction graph* (TDG), as shown in Listing 2. Here, the shaded special vertices marked \textcircled{r} and \textcircled{u} represent an enforcement of the type qualifiers reliable and unreliable by prohibit rules or explicitly stated annotations. The set of edges of the TDG then reflects the dependencies between the type qualifiers, data objects, operations, and assignments.

```

unreliable int u, pos, tmp;
reliable int r, a[10];
u = 10;
r = u;           // invalid assignment
pos = 0;
while (pos < r) {    // invalid condition
    tmp = r / u;      // invalid division
    a[pos++] = tmp;    // invalid memory access
}

```

Listing 3 Invalid assignments

Accordingly, the use of the TDG enables the compiler to flag invalid data propagation from unreliable to reliable data objects. An example containing a number of such invalid propagations is given in Listing 3.

Overall, the static analyses provided by the FEHLER compiler toolchain enable programmers to state reliability requirements that cannot be deduced from the program itself while ensuring that these manual annotations do not accidentally provide a way to propagate unreliable data to reliable data objects. During runtime, the annotations are then used to enable flexible error handling by allowing the operating system to ignore errors in data objects marked as unreliable, thus enabling a tradeoff between the obtained quality of service and the required error correction overhead, e.g. in terms of time or energy.

6 FEHLER Runtime System

Viewed from the top, as shown in Fig. 6, an application with integrated classification information is running on a virtualized guest OS. The guest OS is linked against the FAME Runtime Environment (FAMERE). FAMERE is responsible for the flexible error handling as well as the interfacing with the microvisor. The microvisor runs low-level error correction and ensures the feasibility of software-based error handling (Fig. 4).

The FAMERE runtime is based on our specialized microvisor component which has control over the hardware components relevant to error handling. The main purpose of the microvisor is to isolate critical system components from possible error propagation and schedule the error handling if required. Critical components in this context are resources required to keep error detection and correction running. Depending on the underlying hardware, the actual critical resources vary. If, for example, errors are signaled via interrupts, the interrupt controller will be an element of the critical resource set.

Since the microvisor itself can be affected by errors, it is considered to be a part of the RCB. The microvisor is incapable of protecting itself, since it implements the basic error handling routines. In order to ensure the effectiveness of error

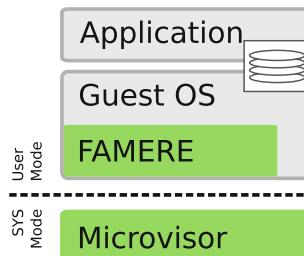


Fig. 4 The runtime software stack of FEHLER. The microvisor is only involved in case of an error, whereas all other resources are administered by the paravirtualized guest OS. The guest OS is extended by FAMERE, the system component responsible for evaluating compiler-provided information on the criticality of errors

handling, fault-free hardware components are required to execute the software-based fault-tolerance mechanisms. In turn, these hardware components also have to be considered part of the RCB. Reducing the code and data size of the microvisor itself is, thus, an optimization objective required to reduce the overall size of the RCB.

To shield the RCB from error propagation, our microvisor uses paravirtualization [16]. The microvisor is tailored to the needs of embedded systems and fault tolerance. To keep the virtualization overhead low, it supports only a single guest operating system. This removes the requirement to provide virtual CPUs and CPU multiplexing. In addition, caches and TLB entries need not be switched between different guest OS instances. An additional responsibility of the microvisor is the creation of full system checkpoints. These are used to restore a valid system state in case of a severe error affecting the FAMERE runtime. FAMERE is a library in the guest OS that combines compile and runtime information required to implement flexible error handling [12].

Error handling is the central task of FAMERE. Figure 5 gives a detailed view of the error-handling procedure at runtime (the right-hand side of Fig. 2). In order to enable a prioritization of error handling, tasks affected by an error in the OS running on top of the microvisor have to be identified. FAMERE determines affected tasks using a memory subscriber model [12] in which tasks explicitly subscribe to and unsubscribe from data objects prior resp. after their use. Accordingly, each data object is annotated with a set of tasks currently using the object, enabling FAMERE to assign a memory address to the set of tasks using the address at the current moment.

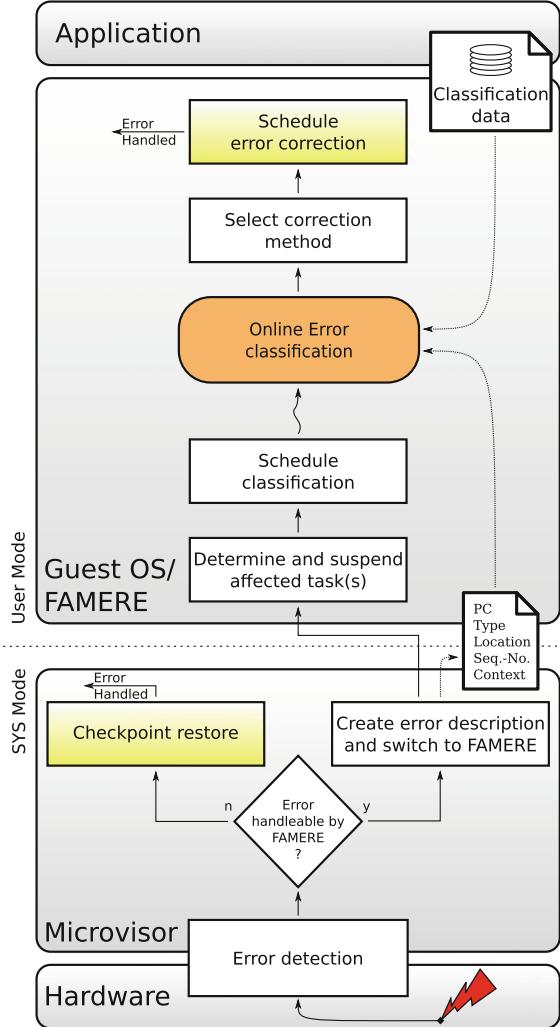
If there are higher prioritized tasks not affected by current error, further error handling will be delayed until all higher prioritized tasks finish execution. Error classification will then be performed when the error handling is scheduled again by the microvisor, thus minimizing the impact on system timing when an error occurs.

Together with classification information for data objects, our microvisor and the FAMERE library enable the FEHLER system to implement the envisioned flexible error-handling principles. By keeping the amount of functionality and the related code and data sizes of the microvisor low, the RCB size could be reduced significantly.

7 Use Case: A Fault-Tolerant QoS-Aware Soft Real-time Application

In order to assess the effectiveness and efficiency of the selective error correction approach enabled by FEHLER, we analyzed typical embedded applications in the presence of errors. Since microbenchmarks only tend to give a restricted view of the effects of errors, we used a real-world application to evaluate the possible reduction in overhead.

Fig. 5 Error handling in the runtime software stack of FEHLER. If an error is signaled (red flash symbol), the microvisor checks whether the fault affects the RCB. If the RCB is affected, the microvisor automatically restores the last system checkpoint. Otherwise, error handling is delegated by sending a message to FAMERE, which includes an error description containing information about the occurred error as well as the user space context



As mentioned above, the class of applications that we expect to benefit most from our flexible error-handling approach are soft real-time applications that are able to accept—or even make use of—varying levels of QoS in their output. Thus, we used a constrained baseline profile H.264 video decoder application comprising ca. 3500 lines of ANSI C code as a real-world benchmark to assess the effectiveness and efficiency of FEHLER [9].

The evaluation is performed on a simulated embedded system using Synopsys' CoMET cycle-accurate simulator as well as a physical platform based on a Marvell ARM926-based SoC. CoMET is configured to resemble the real system by simulating a 1.2 GHz ARM926 system with 64 MiB RAM, 16 MiB ROM, and 128 KiB

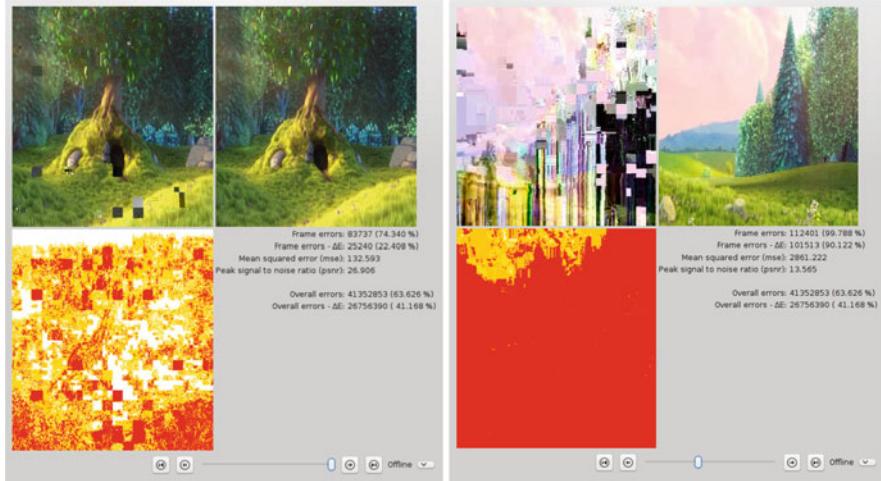


Fig. 6 Analysis of error impacts on the H.264 video decoder using different injection rates

reliable RAM (using ECC-based hardware error protection). All components are considered reliable, except the 64 MiB of RAM.

The H.264 video decoder is configured to create a checkpoint after every displayed frame. In each experiment, we decoded 600 frames in total at a rate of 10 frames per second and a resolution of 480×320 pixels.²

We were primarily interested in evaluation results showing the impact of the injected errors themselves on the achievable QoS of the decoded video, the possible reduction of the RCB size using flexible error handling as well as the impact of error handling on system timing.

To assess the impact on the QoS, we developed the quality assessment tool shown in Fig. 6 [8]. It receives video frames decoded by the target ARM system under the influence of errors using FEHLER's flexible error correction and compares these frames to the correctly decoded reference frames (indicated by the yellow and red squares in the lower left pictures—the more red, the larger the difference between the two frames is). For each frame, the tool then calculates several different metrics indicating the QoS, e.g. the peak signal-to-noise ratio (PSNR) and the ΔE color distance metrics. The left-hand side of Fig. 6 shows a moderate error injection rate resulting in some visible defects in the output, whereas the right-hand side shows an artificially high injection rate which renders the output unusable.

For evaluation, we injected uniformly distributed transient faults into RAM. For each memory access, error detection in hardware is simulated. If the processor

²Although resolution and frame rate seem rather low, this setup leads to a CPU utilization of more than 65%, since we decode H.264 in software only. However, higher resolutions and frame rates will be possible if more computing power is available.

Table 1 Peak Signal-to-Noise Ratio (PSNR) for different error injection rates [dB]

	$\lambda = 1 \cdot 10^{-16}$	$\lambda = 1 \cdot 10^{-15}$	$\lambda = 1 \cdot 10^{-14}$
All errors handled	36.19	36.15	n/a
Flexible error handling	36.19	36.18	29.01
Flexible + application-specific	36.20	36.12	28.95

accesses an erroneous word, an interrupt will be raised. The number of faults to be injected is determined by a Poisson distribution with a configurable parameter λ .³

Table 1 shows QoS results given as PSNR values for different injection rates and correction approaches. We compare a standard correction approach—correcting all errors irrespective of the worst-case outcome—with two approaches based on FEHLER, one which only uses generic error correction such as checkpoint-and-restore and one which, in addition, applies more efficient, application-specific error correction methods. It can be seen that for low error injection rates ($\lambda = 1 \cdot 10^{-16}$ and $1 \cdot 10^{-15}$), uncorrected errors result in a PSNR of about 36 dB, which is still a reasonable quality for lossy compressed media and is similar to the quality of VHS video. For the high error rate ($\lambda = 1 \cdot 10^{-14}$), however, the PSNR drops below 30 dB.⁴

It is important to notice that, although high injected error rates can lead to a significant degradation of the perceived QoS, the primary objective of the binary classification of error impacts employed by FEHLER is achieved—we were unable to provoke the system to crash no matter what the used error injection rate was.

Based on the configuration described above, we analyzed the fraction of memory that the compiler annotated as `unreliable`, implying no protection against errors is required. This fraction is a direct indicator of the reduction of the size of memory that has to be protected, i.e., the RAM memory component of the RCB. In traditional software-based error correction approaches, all of the RAM would be considered part of the RCB. Table 2 shows the results of this evaluation for different video resolutions. It can be observed that for low resolutions, the amount of data classified as `reliable` dominates the memory usage. However, the share of this type of memory is reduced when decoding videos with higher resolutions. For a 720p HD video, already 63% of the RAM used by the H.264 decoder can remain unprotected using FEHLER classifications.

The remaining interesting evaluation is the impact of flexible error handling on the soft real-time properties of the video decoder application. In the first two

³Not all injected faults are visible by the application, since faults are only detected when the corresponding memory cell is accessed.

⁴To control the amount of faults to inject, a Poisson distribution with configurable parameter λ is used. The time base used for the Poisson distribution is memory bus ticks. Faults are randomly injected and are equally distributed over the memory. Hence, the locations of the accesses have no influence on the fault distribution.

Table 2 Reduction of the amount of reliable memory required by the H.264 video decoder

Video resolution	Memory size of reliable data	Percentage	Memory size of unreliable data	Percentage
176 × 144	90 kB	55%	74 kB	45%
352 × 288	223 kB	43%	297 kB	57%
1280 × 720	1585 kB	37%	2700 kB	63%

Table 3 Average deadline misses for different error-handling configurations

Error rate		Naive error handling		Flexible error handling		Flexible + application-specific	
λ	[s^{-1}]	# Avg. Miss	Avg Missed by	# Avg. Miss	Avg Missed by	# Avg. Miss	Avg Missed by
$\lambda = 1 \cdot 10^{-16}$	0.14	0.00	0.00 ms	0.00	0.00 ms	0.00	0.00 ms
$\lambda = 1 \cdot 10^{-15}$	1.44	2.86	8.15 ms	0.52	7.93 ms	0.36	4.89 ms
$\lambda = 1 \cdot 10^{-14}$	35.84	–	–	1937.87	10,268.98 ms	1887.12	9346.16 ms

columns of Table 3, the observed average error rates (of detected faults) are given, ranging from several faults per minute to an artificially high rate of 36 per second.

We analyzed three different scenarios. In naive error handling, the system treats every error as an error which cannot be handled by FAMERE. Hence, a checkpoint is immediately restored. For this scenario, columns three and four in Table 3 show the average amount of missed deadlines and the average duration of a deadline miss, respectively. For the lowest error rate, no deadline misses occur since enough slack time is available for the recovery of checkpoints. If the error rate increases by an order of magnitude, deadline misses can be observed. On average, deadlines were missed by 8.15 ms. For the highest error rate, no run of the experiment terminated within a set limit of 2 h of simulation time, thus no results are given here.

The results for flexible error handling are shown in columns five and six. Here, only errors affecting reliable and live data are handled by checkpoint recovery. Errors affecting other data are ignored. Flexible error handling reduces the number of deadline misses significantly (81.75%). The time by which a deadline is missed is reduced as well (2.70%). For the artificially increased rate of 35.84 errors per second, however, significant deadline misses could be observed.

The final timing evaluation scenario augmented flexible error handling by including an application-specific error-handling method. For data objects with a special annotation, this method is able to transform a corrupted motion vector into a valid state. For these cases, a time-consuming rollback to a valid system state is not required, reducing the overhead for error correction. Accordingly, using this approach, deadline misses could be reduced by 87.37% for the second highest error rate.

To conclude the overview of our evaluation, we provide an overview of a possible use of application-specific error correction approaches for our H.264 video decoder.

	Impact on QoS	Urgency	Fault handling methods
High Impact	Program termination	fix immediately	rollback
	Corrupted frame input	until frame is displayed	redisplay last frame
	Disturbance in frame header	until frame is displayed	rollback, spare frame, redisplay last frame
Medium Impact	Disturbance in DCT coefficients	fix immediately	rollback, ignore
Low Impact	Disturbance in macro block	until frame is displayed	rollback, copy neighbor block, ignore,
	Disturbance in single pixel	until frame is displayed	rollback, copy neighbor pixel, ignore
		:	
No Impact	none		ignore

Fig. 7 Classification of error impacts for the H.264 video decoder

As shown in Fig. 7, errors can occur in different data structures, such as frame header or macro blocks. These can be handled by a number of efficient application-specific error correction approaches.

8 Use Case: Adaptive Error Handling in Control Applications

Control-based systems are the basis of a large number of applications for embedded real-time systems. The inherent safety margins and noise tolerance of control tasks allow that a limited number of errors might be tolerable and might only downgrade control performance; however, such limited errors might not lead to an unrecoverable system state. In control theory literature, techniques have been proposed to enable the stability of control applications even if some signal samples are delayed [14] or dropped [2]. Accordingly, we expect that our idea of flexible fault tolerance as described for the video decoder case will also be applicable to control applications.

As described above, software-based fault-tolerance approaches such as redundant storage or code execution may lead to system overload due to execution time overhead. For control tasks, an adaptive deployment of related error correction is desired in order to meet both application requirements and system constraints.

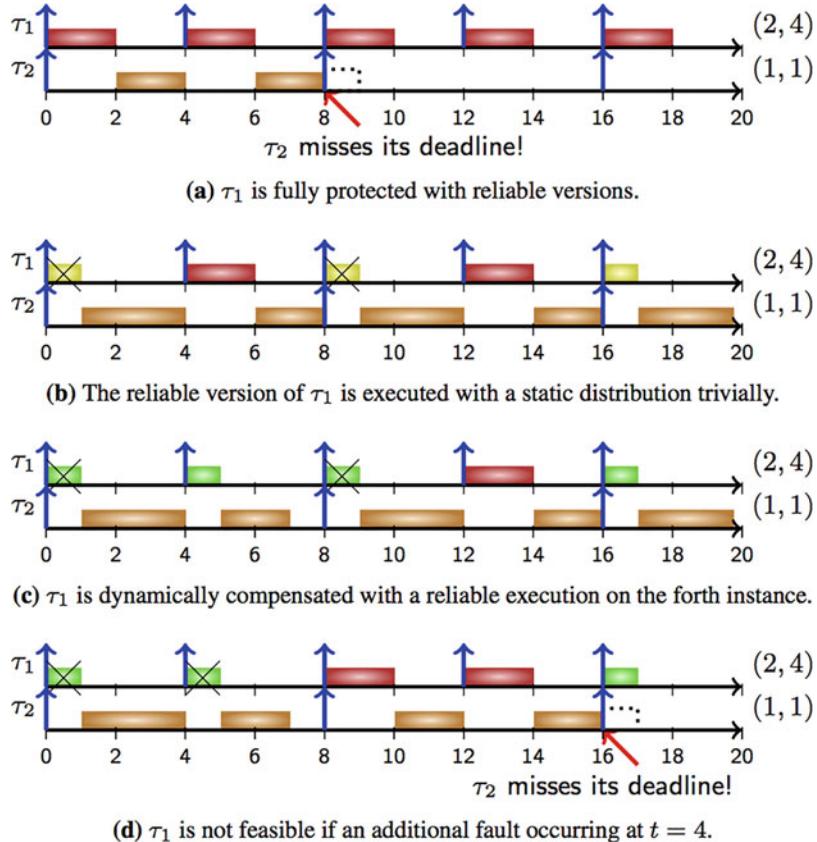


Fig. 8 Different ways to deal with soft errors: red blocks represent reliable executions, green blocks represent executions with error detection, while yellow blocks represent unreliable version (deadlines are implicit in the schedules shown)

Thus, it has to be investigated how and when to compensate, or even ignore errors, given a choice of different techniques. In an initial case study, we observed that a control task can tolerate limited errors with acceptable performance loss [5].⁵

The general approach used to investigate the effectiveness of this approach is to model the fault tolerance of control applications as a (m, k) -constraint which requires at least m correct runs out of any k consecutive runs to be correct. We investigate how a given (m, k) constraint can be satisfied by adopting patterns of task instances with individual error detection and compensation capabilities. Figure 8 shows four different ways to handle soft errors. Some of the presented schedules are infeasible, since they lead to deadline misses.

⁵This section is based on joint work with Kuan-Hsun Chen, Björn Böninghoff and Jian-Jia Chen, TU Dortmund.

A static approach to ensure this property is Static Pattern-Based Reliable Execution. In this approach, we enforce the (m_i, k_i) constraints by applying (m, k) static patterns to allocate the reliable executions for task τ_i . While the adopted pattern will affect the schedulability, stability, and flexibility, deciding the most suitable pattern is out of scope of this work.

Due to its inability to react dynamically to changes at runtime, it is obvious that this approach has to be overprovisioning. Thus, we introduce a runtime adaptive approach called Dynamic Compensation that enhances Static Pattern-Based Reliable Execution by recognizing the need to execute reliable instances dynamically instead of having a static schedule.

It is too pessimistic to allocate the reliable instances strictly due to the fact that soft errors randomly happen from time to time. To mitigate the pessimism, we propose an adaptive approach, called Dynamic Compensation, to decide the executing task version on-the-fly by enhancing Static Pattern-Based Reliable Execution and monitoring the erroneous instances with sporadic replenishment counters.

The idea is to execute the unreliable instances and exploit their successful executions to postpone the moment that the system will not be able to enforce an (m, k) constraint, in which the resulting distribution of execution instances still follows the string of static patterns in the worst case.

With Dynamic Compensation, we prepare a mode indicator Π for each task to distinguish the behaviors of dynamic compensation for different status of tasks, i.e., $\Pi \in \{\text{tolerant}, \text{safe}\}$. If a task τ_i cannot tolerate any error in the following instances, the mode indicator will be set to safe and the compensation will be activated for the robustness accordingly. If it can tolerate error in the next instance, the mode indicator will be set to tolerant and execute the unreliable version with fault detection.

Our investigation showed that in embedded systems used for control applications which are liable to both hard real-time constraints and fulfillment of operational objectives, the inherent robustness of control tasks can be exploited when applying error-handling methods to deal with transient soft errors induced by the environment. When expressing the resulting task requirement regarding correctness as a (m, k) constraint, scheduling strategies based on task versions with different types of error protection become applicable. We have introduced both static- and dynamic-pattern-based approaches, each combined with two different recovery schemes. These strategies drastically reduce utilization compared to full error protection while adhering to both robustness and hard real-time constraints. To ensure the latter for arbitrary task sets, a schedulability test is provided formally. From the evaluation results, we can conclude that the average system utilization can be reduced without any significant drawbacks and be used, e.g., to save energy. This benefit can be increased with further sophistication; however, finding feasible schedules also becomes harder.

For an in-depth discussion in the context of a follow-up investigation of this topic, we refer the reader to [17].

9 Application of FEHLER to Approximate Computing

Whereas the work described above concentrates on handling bit flips in memory, more recently, *approximate computing* approaches have been investigated to design energy-efficient systems that trade result precision for energy consumption.

One of the novel semiconductor technologies at the basis of approximate computing is Probabilistic CMOS [3] (PCMOS). Figure 9 shows the general layout of a ripple-carry adder based on PCMOS technology (PRCA). While traditional energy-conserving circuits use uniform voltage scaling (UVOS), PCMOS employs biased voltage scaling (BIVOS), which provides different single-bit full adder components with differing supply voltages that increase from the least to the most significant bit in multiple steps. As a consequence, the delay required to calculate a bit decreases from the LSB to the MSB; accordingly, the probability p_c of bit errors due to carry bits arriving too late is larger in the least significant bits. Using the PCMOS voltage scaling approach, we also employed a probabilistic Wallace-tree multiplier (PWTM) component and added a related energy model and instructions enabling the use of the probabilistic components to our ARMv4 architecture simulator.

We investigated whether FEHLER reliability annotations would also be applicable to determine which arithmetic operations of a program could be executed on PCMOS-based arithmetic components instead of a less energy-efficient traditional ALU without sacrificing the program's stability [10]. A first evaluation using floating point data objects showed that the use of PCMOS technology has the potential for significant energy conservation. Accordingly, we investigated the possible conservation potential for a real-world embedded application. FEHLER type qualifiers were used to indicate data which accepts precision deviation (*unreliable*). Accordingly, our compiler backend generated instructions using probabilistic arithmetic instructions operating on these data objects.

Table 4 shows that a significant fraction of arithmetic ARM machine instructions of our H.264 video decoder could be executed safely on probabilistic components.⁶

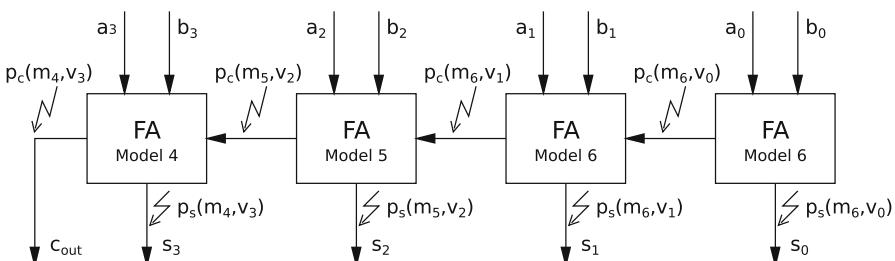
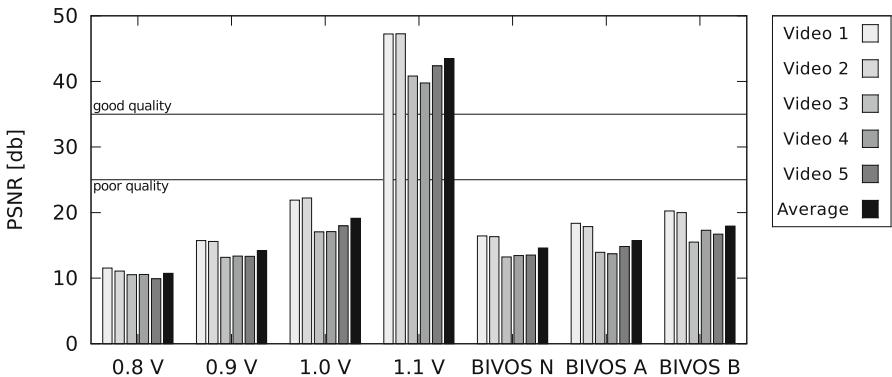


Fig. 9 Probabilistic ripple-carry adder

⁶`rsb` is the ARM reverse subtract instruction.

Table 4 Instructions executed using probabilistic components

Instruction type	Add	Sub	rsb	Mul	Overall
Executed using PRCA/PWTM	18.59%	18.60%	43.01%	76.27%	13.36%

**Fig. 10** PSNR (peak signal-to-noise ratio) values for different supply voltage configurations

Surprisingly, our results indicated that for our typical embedded H.264 decoder application, the use of CMOS components did not result in energy conservation for the identical level of QoS compared to uniform voltage scaling.⁷ This result contradicts the microbenchmarks described in [4]. Figure 10 shows the PSNR of the H.264 decoder output for different video clips decoded with circuits using four different UVOS (0.8 V–1.1 V) as well as three BIVOS schemes with similar energy consumption to the UVOS schemes. It can be observed that the PSNR of the BIVOS-decoded videos does not increase, which is a counterintuitive result at first.

A subsequent investigation of the differences between our H.264 decoder and the code used in the microbenchmarks gave insights into the observed effects. Whereas the microbenchmarks employed floating point numbers, our video decoder is a typical embedded application that employs integer and fixed-point numbers.

```
void enter(unreliable uchar *ptr, unreliable int q_delta) {
    unreliable int i = *ptr + ((q_delta + 32) >> 6);
    *ptr=Clip(i);
}
```

Listing 4 H.264 decoder clipping code

This difference in data representation is one of the reasons for the observed phenomenon. The H.264 specification requires a special behavior when copying 32 bit integer values into an eight bit value in the frame buffer. Here, a saturating clipping function (cf. Listing 4) is used. This function restricts the value to 255 if

⁷This only concerns the static and dynamic energy consumption of the CMOS components. The additional static energy required by the traditional ALU has not been considered here.

the input is larger than that. Accordingly, the shift operation used has the ability to eliminate bit errors in the least significant bits, diminishing the gains of BIVOS scaling.

In contrast, floating point values are always normalized after arithmetic operations. This implies that the bits most relevant to a floating point number's value—sign, exponent, and the MSBs of the mantissa—are always the MSBs of the memory word. In this case, the BIVOS approach to construct arithmetic components that show larger error probabilities in the LSBs is beneficial.

Since it is unrealistic to assume that separate adders for different data widths and data types will be provided in future architectures, an analysis of the number of bits actually used in arithmetic operations is required. However, this implies further complications. One idea for future compiler-based analyses is an approach that combines bit-width analysis methods for arithmetic operations and code transformations to use bits with optimal supply voltage for the operation at hand. The effectiveness of this approach, however, requires further implementation and analysis work.

10 Summary and Outlook

The results of the FEHLER project have shown that for a large class of embedded applications, software-based fault tolerance is a feasible way to reduce the overhead of error handling. The results, as demonstrated using real-world applications, show that already the simple binary classification employed so far is able to avoid crashes due to soft errors while reducing the size of the reliable computing base, i.e. the amount and size of hard- and software components requiring protection from errors.

The technologies developed in the context of FEHLER suggest a number of ways to further improve on the ideas and design of the approach. One constraint of the current design is that the current version of reliability type qualifiers is too coarse-grained. Correcting only errors that affect `reliable` data objects will result in avoiding program crashes. However, a sufficiently high error rate affecting `unreliable` data might still result in a significant reduction of the QoS, rendering its output useless.

The existing static analysis in FEHLER is based on subtyping. Accordingly, to provide a more fine-grained classification of errors, additional error classes have to be introduced. These classes would have to be characterized according to a given total order, so that an error can be classified with the correct worst-case effect. If, for example, the impact of errors is measured in the degradation of a signal-to-noise ratio, a total order can be determined by the resulting amount of degradation.

However, for the overall assessment of a program's QoS, the resulting overall error visible in the output that accumulated throughout the data flow is relevant. Here, one can imagine setting an acceptable QoS limit for the output data and backtracking throughout the arithmetical operations in the program's data flow to

determine the *worst-case deviation* that an error in a given variable can cause in the output. Here, we intend to employ approaches related to numerical error propagation analysis.

We expect that approximate computing approaches will be able to directly benefit from these analyses. Since the approximations already trade precision for other non-functional properties, such as energy consumption, a Pareto optimization of the differing objectives could benefit from worst-case QoS deviation analyses. Here, our initial analysis of the use of binary classifiers for the PC莫斯 case has already given some interesting preliminary insights.

References

1. Austin, T., Bertacco, V., Mahlke, S., Cao, Y.: Reliable systems on unreliable fabrics. *IEEE Des. Test Comput.* **25**(4), 322–332 (2008)
2. Bund, T., Slomka, F.: Sensitivity analysis of dropped samples for performance-oriented controller design. In: 2015 IEEE 18th International Symposium on Real-Time Distributed Computing, pp. 244–251 (2015)
3. Chakrapani, L.N., Akgul, B.E.S., Cheemalavagu, S., Korkmaz, P., Palem, K.V., Seshasayee, B.: Ultra-efficient (embedded) SoC architectures based on probabilistic CMOS (PC莫斯) technology. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06), pp. 1110–1115. European Design and Automation Association (2006)
4. Chakrapani, L.N., Muntimadugu, K.K., Lingamneni, A., George, J., Palem, K.V.: Highly energy and performance efficient embedded computing through approximately correct arithmetic: a mathematical foundation and preliminary experimental validation. In: Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '08), pp. 187–196. ACM, New York (2008)
5. Chen, K.H., Bönnighoff, B., Chen, J.J., Marwedel, P.: Compensate or ignore? Meeting control robustness requirements through adaptive soft-error handling. In: Languages, Compilers, Tools and Theory for Embedded Systems (LCTES). ACM, Santa Barbara (2016)
6. Engel, M., Döbel, B.: The reliable computing base—a paradigm for software-based reliability. In: INFORMATIK 2012, pp. 480–493. Gesellschaft für Informatik e.V., Bonn (2012)
7. Foster, J.S., Fähndrich, M., Aiken, A.: A theory of type qualifiers. In: Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation (PLDI '99), pp. 192–203. ACM, New York (1999)
8. Heinig, A., Engel, M., Schmoll, F., Marwedel, P.: Improving transient memory fault resilience of an H.264 decoder. In: Proceedings of the Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia 2010). IEEE Computer Society Press, Scottsdale (2010)
9. Heinig, A., Engel, M., Schmoll, F., Marwedel, P.: Using application knowledge to improve embedded systems dependability. In: Proceedings of the Workshop on Hot Topics in System Dependability (HotDep 2010). USENIX Association, Vancouver (2010)
10. Heinig, A., Mooney, V.J., Schmoll, F., Marwedel, P., Palem, K., Engel, M.: Classification-based improvement of application robustness and quality of service in probabilistic computer systems. In: Proceedings of the 25th International Conference on Architecture of Computing Systems (ARCS'12), pp. 1–12. Springer, Berlin (2012)
11. Heinig, A., Schmoll, F., Bönnighoff, B., Marwedel, P., Engel, M.: Fame: flexible real-time aware error correction by combining application knowledge and run-time information. In: Proceedings of the 11th Workshop on Silicon Errors in Logic-System Effects (SELSE) (2015)
12. Heinig, A., Schmoll, F., Marwedel, P., Engel, M.: Who's using that memory? A subscriber model for mapping errors to tasks. In: Proceedings of the 10th Workshop on Silicon Errors in Logic-System Effects (SELSE), Stanford, CA, USA (2014)

13. de Kruijf, M., Nomura, S., Sankaralingam, K.: Relax: an architectural framework for software recovery of hardware faults. In: Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10), pp. 497–508. ACM, New York (2010)
14. Ramanathan, P.: Overload management in real-time control applications using (m, k)-firm guarantee. IEEE Trans. Parallel Distrib. Syst. **10**(6), 549–559 (1999)
15. Schmoll, F., Heinig, A., Marwedel, P., Engel, M.: Improving the fault resilience of an H.264 decoder using static analysis methods. ACM Trans. Embed. Comput. Syst. **13**(1s), 31:1–31:27 (2013)
16. Whitaker, A., Shaw, M., Gribble, S.: Denali: lightweight virtual machines for distributed and networked applications. In: Proceedings of the 2002 USENIX Annual Technical Conference (2002)
17. Yayla, M., Chen, K., Chen, J.: Fault tolerance on control applications: empirical investigations of impacts from incorrect calculations. In: 2018 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), pp. 17–24 (2018)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



ASTEROID and the Replica-Aware Co-scheduling for Mixed-Criticality



Eberle A. Rambo and Rolf Ernst

1 The ASTEROID Project

1.1 Motivation

Technology downscaling has increased the hardware's overall susceptibility to errors to the point where they became non-negligible [17, 21, 22]. Hence, current and future computing systems must be appropriately designed to cope with errors in order to provide a reliable service and correct functionality [17, 21]. That is a challenge, especially in the real-time mixed-criticality domain where applications with different requirements and criticalities co-exist in the system, which must provide *sufficient independence* and prevent error propagation (e.g., timing, data corruption) between criticalities [24, 42]. Recent examples are increasingly complex applications such as flight management systems (FMS), advanced driver assistance systems (ADAS), and autonomous driving (AD) in the avionics and automotive domains, respectively [24, 42]. A major threat to the reliability of such systems is the so-called soft errors.

Soft errors, more specifically Single Event Effects (SEEs), are transient faults abstracted as *bit-flips* in hardware and can be caused by alpha particles, energetic neutrons from cosmic radiation, and process variability [15, 22]. Soft errors are comprehensively discussed in chapter “Reliable CPS Design for Unreliable Hardware Platforms”. Depending on where and when they occur, their impact on software execution range from masked (no observable effect) to a complete system crash [3, 12, 13]. Soft errors are typically more frequent than hard errors

E. A. Rambo (✉) · R. Ernst

Institute of Computer and Network Engineering (IDA), TU Braunschweig, Braunschweig,
Germany

e-mail: rambo@ida.ing.tu-bs.de; ernst@ida.ing.tu-bs.de

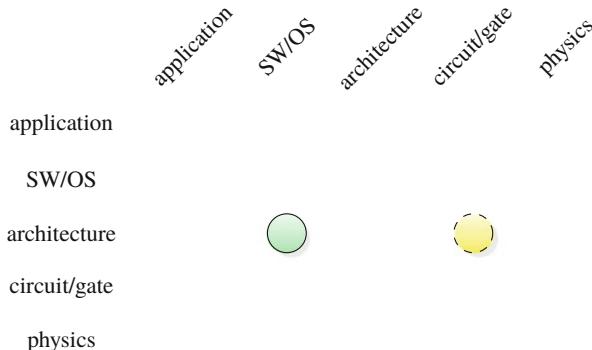


Fig. 1 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

(permanent faults) and often remain undetected, also known as latent error or silent data corruption, because they cannot be detected by testing. Moreover, undetected errors are a frequent source of later system crashes [12]. To handle soft errors, the approaches can vary from completely software-based to completely hardware-based. The former are able to cover only part of the errors [12, 13] and the latter result in costly redundant hardware [22], as seen in lock-step dual-core execution [29]. Cross-layer solutions can be more effective and efficient by distributing the tasks of detecting errors, handling them and recovering from them in different layers of software and hardware [12, 13, 22].

1.2 Overview

The ASTEROID project [5] developed a cross-layer fault-tolerance solution to provide reliable software execution on unreliable hardware. The approach is based on replicated software execution and exploits the large number of cores available in modern and future architectures at a higher level of abstraction without resorting to hardware redundancy [5, 12]. That concentrates ASTEROID’s contributions around the architecture and operating system (OS) abstraction layers, as illustrated in Fig. 1. ASTEROID’s architecture is illustrated in Fig. 2. The reliable software execution is realized by the OS service Romain [12]. Mixed-critical applications may co-exist in the system and are translated into protected and unprotected applications. Romain replicates the protected applications, which are mapped to arbitrary cores, and manages their execution. Error detection is realized by a set of mechanisms whose main feature is the hardware assisted state comparison, which compares the replicas’ state at certain points in time [5, 12]. Error recovery strategies can vary depending on whether the application is running in dual modular redundancy (DMR) or triple modular redundancy (TMR) [3, 5].

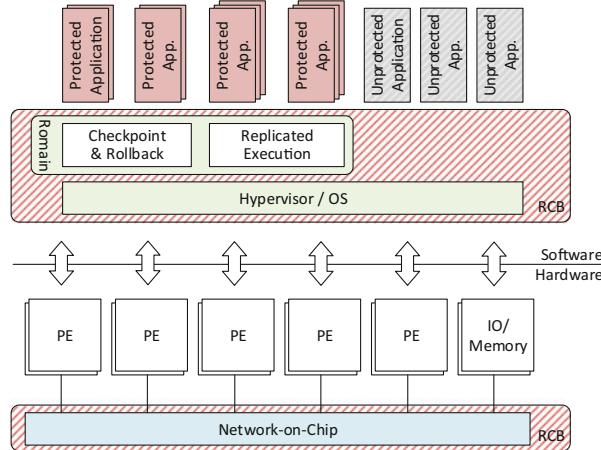


Fig. 2 ASTEROID’s architecture

ASTEROID comprised topics ranging from system-level conceptual modeling, to the OS and all the way down to Application-Specific Integrated Circuit (ASIC) synthesis and gate-level simulation. We summarize selected work that were developed in the project next. An initial overview of the ASTEROID approach was introduced in [5]. Romain, the OS service that provides replicated execution for unmodified binary applications, was introduced in [12]. The vulnerabilities of the system were assessed in [9, 13], giving rise to the reliable computing base (RCB), the minimum set of software and hardware components on which the approach relies. The runtime overheads related with the OS-assisted replication were investigated in [10]. Later, RomainMT extends Romain in [11] to support unmodified multithreaded applications. A systematic design process was investigated in [28], followed by the definition of a trusted component ecosystem in [19].

In terms of modeling, the reliability of replicated execution was modeled and evaluated in [3]. The approach was modeled in Compositional Performance Analysis (CPA), a worst-case performance analysis framework, as fork-join tasks and the performance evaluated in [6] and revised in [2]. Later, co-scheduling was employed to improve the worst-case performance of replicated execution with the replica-aware co-scheduling for mixed-criticality [34]. Off-chip real-time communication under soft errors was modeled in [4] with a probabilistic response-time analysis. On-chip real-time communication with and without soft errors were modeled in CPA and evaluated in [33] and [38], where Automatic Repeat reQuest (ARQ)-based protocols were employed in a real-time Network-on-Chip (NoC). As part of the RCB, the NoC’s behavior under soft errors was further researched with thorough Failure Mode and Effects Analyses (FMEAs) in [35–37]. Based on those findings, a resilient NoC architecture was proposed in [32, 39, 40], which is able to provide a reliable and predictable service under soft errors.

The remainder of this chapter focuses on the performance of replicated execution under real-time constraints, first published in [34]. It is organized as follows. Section 2 introduces the replica-aware co-scheduling for mixed-criticality and its related work. Section 3 describes the system, task, and error models. Section 4 introduces the formal response-time analysis. Experimental results are reported in Sect. 5. Finally, Sect. 6 ends the chapter with the conclusions.

2 Replica-Aware Co-scheduling for Mixed-Criticality

2.1 Motivation

Replicated software execution is a flexible and powerful mechanism to increase the reliability of the software execution on unreliable hardware. However, the scheduler has a direct influence on its performance. The performance of replicated execution for real-time applications has been formally analyzed in [6] and revised in [2]. The work considers the well-known Partitioned Strict Priority Preemptive (SPP) scheduling, where tasks are mapped to arbitrary cores, and assumes a single error model. The authors found that SPP, although widely employed in real-time systems, provides very pessimistic response-time bounds for replicated tasks. Depending on the interfering workload, replicated tasks executing serially (on the same core) present much better performance than when executing in parallel (on distinct cores). That occurs due to the long time that replicated tasks potentially have to wait on each core to synchronize and compare states before resuming execution. That leads to very low resource utilization and prevents the use of replicated execution in practice.

The replica-aware co-scheduling for mixed-criticality explores co-scheduling to provide short response times for replicated tasks without hindering the remaining unprotected tasks. Co-scheduling is a technique that schedules interacting tasks/threads to execute simultaneously on different cores [30]. It allows tasks/threads to communicate more efficiently by reducing the time they are blocked during synchronization. In contrast to SPP [2, 6], the proposed replica-aware co-scheduling approach drastically minimizes delays due to the implicit synchronization found in state comparisons. In contrast to gang scheduling [14], it rules out starvation and distributes the execution of replicas in time to achieve short response times of unprotected tasks. The proposed approach differs from standard Time-Division Multiplexing (TDM) and TDM with background partition [25] in that all tasks have formal guarantees. In contrast to related work, it supports different recovery strategies and accounts for the NoC communication delay and overheads due to replica management and state comparison. Experimental results with benchmark applications show an improvement on taskset schedulability of up to $6.9\times$ when compared to SPP, and $1.5\times$ when compared to a TDM-based scheduler.

2.2 Related Work

L4/Romain [12] is a cross-layer fault-tolerance approach that provides reliable software execution under soft errors. Romain provides protection at the application-level by replicating and managing the applications' executions as an operating system service. The error detection is realized by a set of mechanisms [5, 12, 13] whose main feature is the hardware assisted state comparison, which allows an effective and efficient comparison of the replicas' states. Pipeline fingerprinting [5] provides a checksum of the retired instructions and the pipeline's data path in every processor, detecting errors in the execution flow and data. The state comparison, reduced to comparing checksums instead of data structures, is carried out at certain points in time. It must occur at least when the application is about to externalize its state, e.g., in a *syscall* [12]. The replica generated *syscalls* are intercepted by Romain, have their integrity checked, and their replicas' states compared before being allowed to externalize the state [12].

Mixed-criticality, in the context of the approach, is supported with different levels of protection for applications with different criticalities and requirements (unprotected, protected with DMR¹ or TMR) and by ensuring that timing constraints are met even in case of errors. For instance, Romain provides different error recovery strategies [3, 5]:

- *DMR with checkpoint and rollback*: to recover, the replicas rollback to their last valid state and re-execute;
- *TMR with state copy*: to recover, the state of the faulty replica is replaced with the state of one of the healthy replicas.

This chapter focuses on the system-level timing aspect of errors affecting the applications. We assume thereby the absence of failures in critical components [13, 32], such as the OS/hypervisor, the replica manager/voter (e.g., Romain), and interconnect (e.g., NoC), which can be protected as in [23, 39].

The Worst-Case Response Time (WCRT) of replicated execution has been analyzed in [6], where replicas are modeled as fork-join tasks in a system implementing Partitioned SPP. The work was later revised in [2] due to optimism in the original approach. The revised approach is used in this work. In that approach, with deadline monotonic priority assignment, where the priority of tasks decreases as their deadlines increase, replicated tasks perform worse when mapped in parallel than when mapped to the same core. This is due to the state comparisons during execution, which involves implicit synchronization between cores. With partitioned scheduling, in the worst-case, the synchronization ends up accumulating the interference from all cores to which the replicated task is mapped, resulting in poor performance at higher loads. On the other hand, mapping replicated tasks

¹DMR *per se* can be used for system integrity only. However, DMR augmented with checkpointing and rollback enables recovery and can be used to achieve integrity and availability (state rollback followed by re-execution in both replicas) [3, 5].

to the highest priorities results in long response times for lower priority tasks and rules out deadline monotonicity. The latter causes the unschedulability of all tasksets with at least one regular task whose deadline is shorter than the execution time of a replicated task.

Gang scheduling [14] is a co-scheduling variant that schedules groups of interacting tasks/threads simultaneously. It increases performance by reducing the inter-thread communication latency. The authors in [26] present an integration between gang scheduling and Global Earliest Deadline First (EDF), called the Gang EDF. They provide a schedulability analysis derived from the Global EDF's based on the sporadic task model. In another work, [16] shows that SPP Gang schedulers in general are not predictable, for instance, due to priority inversions and slack utilization. In the context of real-time systems, gang scheduling has not received much attention.

TDM-based scheduling [25] is widely employed to achieve predictability and ensure temporal-isolation. Tasks are allocated to partitions, which are scheduled to execute in time slots. Partitions can span across several (or all) cores and can be executed at the same time. The downside of TDM is that it is not work-conserving and underutilizes system resources. A TDM variant with background partition [25] tackles this issue by allowing low priority tasks to execute in other partitions whenever no higher priority workload is executing. Yet, in addition to the high cost to switch between partitions, no guarantees can be given to tasks in the background partition.

In the proposed approach, we exploit co-scheduling with SPP to improve the performance of the system. The proposed approach differs from [6] in that replicas are treated as gangs and are mapped with highest priorities, and are hence activated simultaneously on different cores. In contrast to gang scheduling [14, 16] and to [6], the execution of replicas is distributed in time with offsets to compensate for the lack of deadline monotonicity, thus allowing the schedulability of tasks with short deadlines. We further provide for the worst-case performance of lower priority tasks by allowing them to execute whenever no higher priority workload is executing. However, in contrast to [25], all tasks have WCRT guarantees. Moreover, we also model the state comparison and the on-chip communication overheads.

3 System, Task, and Error Models

In this work, we use the CPA [20] to provide formal response-time bounds. Let us introduce the system, task, and error models.

3.1 System Model

The system consists of a standard NoC-based many-core composed of processing elements, simply referred to as cores.

There are two types of tasks in our system, as in [2]:

- *independent* tasks τ_i : regular, unprotected tasks; and
- *fork-join* tasks Γ_i : replicated, protected tasks.

The system implements partitioned scheduling, where the operating system manages tasks statically mapped to cores. The mapping is assumed to be given as input. The scheduling policy is a combination of SPP and gang scheduling. When executing only independent tasks, the system's behavior is identical to Partitioned SPP, where tasks are scheduled independently on each core according to SPP. It differs from SPP when scheduling fork-join tasks.

Fork-join tasks are mapped with highest priorities, hence do not suffer interference from independent tasks, and execute simultaneously on different cores, as in gang scheduling. Note that deadline monotonicity is, therefore, only partially possible. To limit the interference to independent tasks, the execution of a fork-join task is divided in smaller intervals called stages, whose executions are distributed in time. At the end of each stage, the states of the replicas are compared. In case of an error, i.e. states differ, recovery is triggered.

Fork-join stages are executed with static offsets [31] in execution slots. One stage is executed per slot. On a core with n fork-join tasks, there are $n + 1$ execution slots: one slot for each fork-join task Γ_i and one slot for recovery. The slots are cyclically scheduled in a cycle Φ . The slot for Γ_i starts at offset $\phi(\Gamma_i)$ relative to the start of Φ and ends after $\varphi(\Gamma_i)$, the slot length. The recovery slot is shared by all fork-join tasks on that core and is where error recovery may take place under a single error assumption (details in Sects. 3.3 and 4.3). The recovery slot has an offset $\phi(recovery)$ relative to Φ and length $\varphi(recovery)$. Lower priority independent tasks are allowed to execute whenever no higher priority workload is executing.

An example is shown in Fig. 3, where two fork-join tasks Γ_1 and Γ_2 and two independent tasks τ_3 and τ_4 are mapped to two cores. Γ_1 and Γ_2 execute in their respective slots simultaneously in both cores. When an error occurs, the recovery of Γ_2 is scheduled and the recovery of the error-affected stage occurs in the recovery slot. The use of offsets enables the schedulability of independent tasks with short periods and deadlines, such as τ_3 and τ_4 . Note that, without the offsets, Γ_1 and Γ_2 would execute back-to-back leading to the unschedulability of τ_3 and τ_4 .

3.2 Task Model

An independent task τ_i is mapped to core σ with a priority p . Once activated, it executes for at most C_i , its worst-case execution time (WCET). The activations of

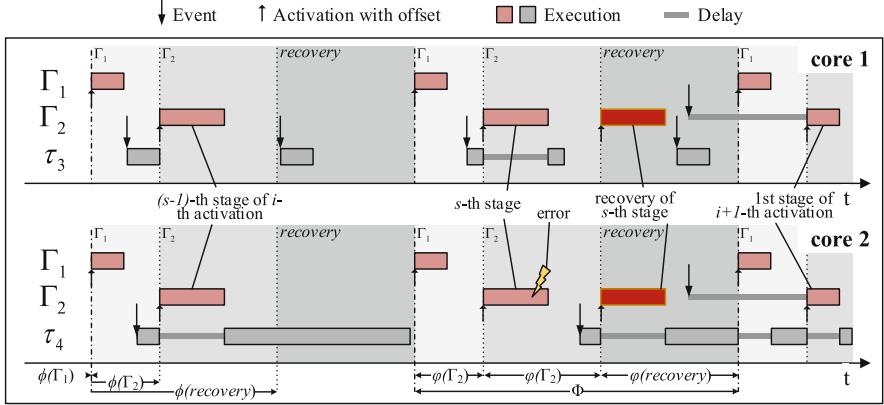


Fig. 3 Execution example with two fork-join and two independent tasks on two cores [34]

a task are modeled with arbitrary *event models*. Task activations in an event model are given by arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum and maximum number of events arriving in any time interval Δt . Their pseudo-inverse counterparts $\delta^+(q)$ and $\delta^-(q)$ return the maximum and minimum time interval between the first and last events in any sequence of q event arrivals. Conversion is provided in [41]. Periodic events with jitter, sporadic events, and others can be modeled with the minimum distance function $\delta_i^-(q)$ as follows [41]:

$$\delta_i^-(q) = \max((q - 1) \cdot d^{min}, (q - 1) \cdot \mathcal{P} - \mathcal{J}) \quad (1)$$

where \mathcal{P} is the period, \mathcal{J} is the jitter, d^{min} is the minimum distance between any two events, and the subscript i indicates the association with a task τ_i or Γ_i .

Fork-join tasks are rigid parallel tasks, i.e. the number of processors required by a fork-join task is fixed and specified externally to the scheduler [16], and consist of multiple stages with data dependencies, as in [1, 2]. A fork-join task Γ_i is a Directed Acyclic Graph (DAG) $G(V, E)$, where vertices in V are subtasks and edges in E are precedence dependencies [2]. In the graph, tasks are partitioned in *segments* and *stages*, as illustrated in Fig. 5a. A subtask $\tau_i^{\sigma,s}$ is the s -th stage of the σ -th segment and is annotated with its WCET $C_i^{\sigma,s}$. The WCET of a stage is equal across all segments, i.e. $\forall x, y : C_i^{x,s} = C_i^{y,s}$. Each segment σ of Γ_i is mapped to a distinct core. A fork-join task Γ_i is annotated with the *static offset* $\phi(\Gamma_i)$, which marks the start of its execution slot in Φ . The offset also admits a small positive jitter j_ϕ , to account for a slight desynchronization between cores and context switch overhead.

The activations of a fork-join task are modeled with *event models*. Once Γ_i is activated, its stages are successively activated by the completion of all segments of the previous stage, as in [1, 2]. Our approach differs from them in that it restricts the scheduling of at most one stage of Γ_i in a cycle Φ , and the stage receives service at the offset $\phi(\Gamma_i)$. Note that the event arrival at a fork-join task is not synchronized

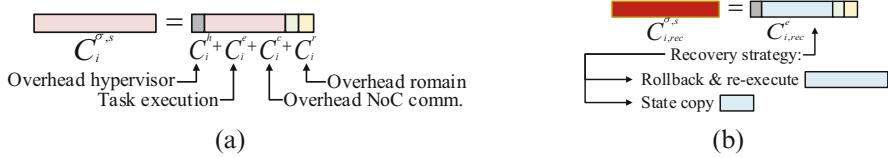


Fig. 4 The composition of WCET of fork-join subtasks [34]. (a) WCET of a fork-join subtask. (b) WCET of recovery

with its offset. The events at a fork-join task are queued at the first stage and only one event at a time is processed (FIFO) [2]. A queued event is admitted when the previous event leaves the last stage.

The interaction with Romain (the voter) is modeled in the analysis as part of the WCET $C_i^{\sigma,s}$, as depicted in Fig. 4a. The WCET includes the on-chip communication latency and state comparison overheads, as the Romain instance may be mapped to an arbitrary core. Those can be obtained, e.g., with [38] along with task mapping and scheduler properties to avoid over-conservative interference estimation and obtain tighter bounds.

3.3 Error Model

Our model assumes a single error scenario caused by SEEs. We assume that all errors affecting fork-join tasks can be detected and contained, ensuring integrity. The overhead of error detection mechanisms is modeled as part of the WCET (cf. Fig. 4a). Regarding independent tasks, we assume that an error immediately leads to a task failure and assume also that its failure will not violate the WCRT guarantees of the remaining tasks. Those assumptions are met, e.g., by Romain.² Moreover, we assume the absence of failures in critical components [13, 32], such as the OS, the replica manager/voter Romain, and the interconnect (e.g., the NoC), which can be protected as in [23, 39].

Our model provides recovery² for fork-join tasks, ensuring their availability. With a recovery slot in every cycle Φ , our approach is able to handle up to one error per cycle Φ . However, the analysis in Sect. 4.3 assumes at most one error per busy window for the sake of a simpler analysis (the concept will be introduced in Sect. 4). The assumption is reasonable since the probability of a multiple error scenario is very low and can be considered as an acceptable risk [24]. A multiple error scenario occurs only if an error affects more than one replica at a time or if more than one error occurs within the same busy window.

²Romain is able to detect and recover from all soft errors affecting user-level applications. For details on the different error impacts and detection strategies, the interested reader can refer to [5, 12].

3.4 Offsets

The execution of fork-join tasks in our approach is based on static offsets, which are assumed to be provided as input to the scheduler. The offsets form execution slots whose size do not vary during runtime, as seen in Fig. 3. Varying the slots sizes would substantially increase the timing analysis complexity without a justifiable performance gain. The offsets must satisfy two constraints:

Constraint 1 *A slot for a fork-join task Γ_i must be large enough to fit the largest stage of Γ_i . That is, $\forall s, \sigma : \varphi(\Gamma_i) \geq C_i^{\sigma,s} + j_\phi$.*

Constraint 2 *The recovery slot must be large enough to fit the recovery of the largest stage of any fork-join task mapped to that core. That is, $\forall i, s, \sigma : \varphi(\text{recovery}) \geq C_{i,\text{rec}}^{\sigma,s} + j_\phi$.*

where a one error scenario per cycle is assumed and $C_{i,\text{rec}}^{\sigma,s}$ is the recovery WCET of subtask $\tau_i^{\sigma,s}$ (cf. Sect. 4.3).

We provide basic offsets that satisfy Constraints 1 and 2. The calculation must consider only overlapping fork-join tasks, i.e. fork-join tasks mapped to at least one core in common. Offsets for non-overlapping fork-join tasks are computed separately as they do not interfere directly with each other. The indirect interference, e.g., in the NoC, is accounted for in the WCETs. First we determine the smallest slots that satisfy Constraint 1:

$$\forall \Gamma_i : \varphi(\Gamma_i) = \max_{\forall \sigma, s} \{C_i^{\sigma,s}\} + j_\phi \quad (2)$$

and the smallest recovery slot that satisfies Constraint 2:

$$\varphi(\text{recovery}) = \max_{\forall \Gamma_i, \tau_j^{\sigma,s} \in \Gamma_i} \{C_{i,\text{rec}}^{\sigma,s}\} + j_\phi \quad (3)$$

The cycle Φ is then the sum of all slots:

$$\Phi = \sum_{\forall \Gamma_i} \{\varphi(\Gamma_i)\} + \varphi(\text{recovery}) \quad (4)$$

The offsets then depend on the order in which the slots are placed inside Φ . Assuming that the slots $\phi(\Gamma_i)$ are sorted in ascending order on i and that the recovery slot is the last one, the offsets are obtained by

$$\phi(x) = \begin{cases} 0 & \text{if } x = \Gamma_1 \\ \phi(\Gamma_{i-1}) + \varphi(\Gamma_{i-1}) & \text{if } x = \Gamma_i \text{ and } i > 1 \\ \Phi - \varphi(\text{recovery}) & \text{if } x = \text{recovery} \end{cases} \quad (5)$$

4 Response-Time Analysis

The analysis is based on CPA and inspired by Aixer [2] and Palencia and Harbour [31]. In CPA, the WCRT is calculated with the busy window approach [43]. The response time of an event of a task τ_i (resp. Γ_i) is the time interval between the event arrival and the completion of its execution. In the busy window approach [43], the event with the WCRT can be found inside the busy window. The busy window w_i of a task τ_i (resp. Γ_i) is the time interval where all response times of the task depend on the execution of at least one previous event in the same busy window, except for the task's first event. The busy window starts at a critical instant corresponding to the worst-case scheduling scenario. Since the worst-case scheduling scenario depends on the type of task, it will be derived individually in the sequel.

Before we derive the analysis for fork-join and for independent tasks, let us introduce the example in Fig. 5 used throughout the section. The taskset consists of four independent tasks and two fork-join tasks, mapped to two cores. The task priority on each core decreases from top to bottom (e.g., $\tau_1^{1,1}$ has the highest priority and τ_4 the lowest).

4.1 Fork-Join Tasks

We now derive the WCRT for an arbitrary fork-join task Γ_i . To do that, we need to identify the critical instant leading to the worst-case scheduling scenario. In case

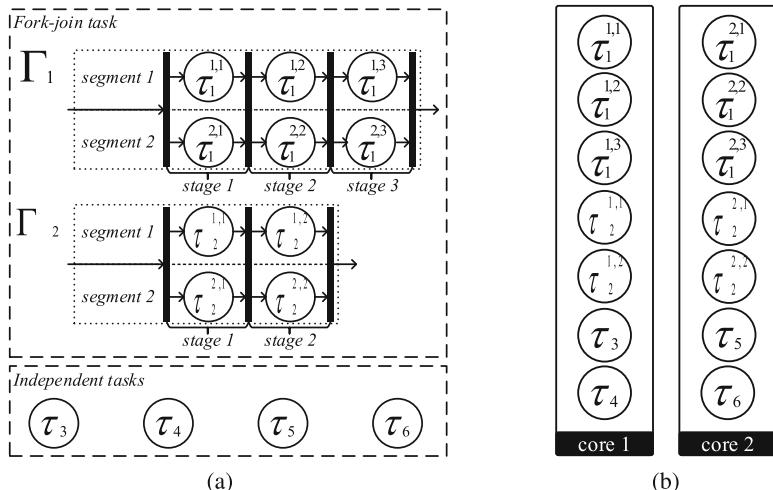


Fig. 5 A taskset with 4 independent tasks and 2 fork-join tasks, and its mapping to 2 cores. Highest priority at the top, lowest at the bottom [34]. (a) Taskset. (b) Mapping

of SPP, the critical instant is when all tasks are activated at the same time and the tasks' subsequent events arrive as early as possible [43]. In our case, the critical instant must also account for the use of static offsets [31].

The worst-case scheduling scenario for Γ_2 on core 1 is illustrated in Fig. 6. Γ_2 is activated and executed at the same time on cores 1 and 2 (omitted). Note that, by design, fork-join tasks do not dynamically interfere with each other. The *critical instant* occurs when the first event of Γ_2 arrives just after missing Γ_2 's offset. The event has to wait until the next cycle to be served, which takes time $\Phi + j_\phi$ when the activation with offset is delayed by a jitter j_ϕ . Notice that the WCETs of fork-join tasks already account for the inter-core communication and synchronization overhead (cf. Fig. 4a).

Lemma 1 *The critical instant leading to the worst-case scheduling scenario of a fork-join task Γ_i is when the first event of Γ_i arrives just after missing Γ_i 's offset $\phi(\Gamma_i)$.*

Proof A fork-join task Γ_i does not suffer interference from independent tasks or other fork-join tasks. The former holds since independent tasks always have lower priority. The latter holds due to three reasons: an arbitrary fork-join task Γ_j always receives service in its slot $\phi(\Gamma_j)$; the slot $\phi(\Gamma_j)$ is large enough to fit Γ_j 's largest subtask (Constraint 1); and the slots in a cycle Φ are disjoint. Thus, the critical instant can only be influenced by Γ_i itself.

We prove by contradiction. Suppose that there is another scenario worse than Lemma 1. That means that the first event can arrive at a time that causes a delay to Γ_i larger than $\Phi + j_\phi$. However, if the delay is larger than $\Phi + j_\phi$, then the event arrived before a previous slot $\phi(\Gamma_i)$ and Γ_i did not receive service. Since that can only happen if there is a pending activation of Γ_i and thus violates the definition of a busy window, the hypothesis must be rejected. \square

Let us now derive the Multiple-Event Queueing Delay $Q_i(q)$ and Multiple-Event Busy Time $B_i(q)$ on which the busy window relies. $Q_i(q)$ is the longest time interval between the arrival of Γ_i 's first activation and the first time its q -th activation

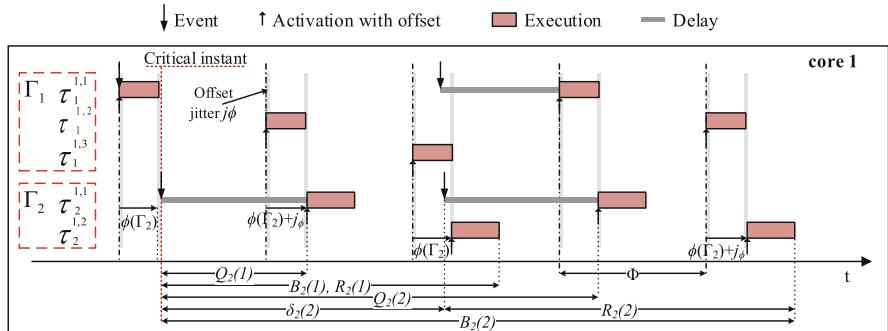


Fig. 6 Worst-case schedule for fork-join gang Γ_2 on core 1 (cf. Fig. 5) [34]

receives service, considering that all events belong to the same busy window [2, 27]. For Γ_i , the q -th activation can receive service at the next cycle Φ after the execution of $q-1$ activations of Γ_i lasting $s_i \cdot \Phi$ each, a delay Φ (cf. (cf. Lemma 1), and a jitter j_ϕ . This is given by

$$Q_i(q) = (q - 1) \cdot s_i \cdot \Phi + \Phi + j_\phi \quad (6)$$

where s_i is the number of stages of Γ_i and Φ is the cycle.

Lemma 2 *The Multiple-Event Queueing Delay $Q_i(q)$ given by Eq. 6 is an upper bound.*

Proof The proof is by induction. When $q=1$, Γ_i has to wait for service at most until the next cycle Φ plus an offset jitter j_ϕ to get service for its first stage, considering that the event arrives just after its offset (Lemma 1). In a subsequent $q+1$ -th activation in the same busy window, Eq. 6 must also consider q entire executions of Γ_i . Since Γ_i has s_i stages and only one stage can be activated and executed per cycle Φ , it takes additional $s_i \cdot \Phi$ for each activation of Γ_i , resulting in Eq. 6. \square

The Multiple-Event Busy Time $B_i(q)$ is the longest time interval between the arrival of Γ_i 's first activation and the completion of its q -th activation, considering that all events belong to the same busy window [2, 27]. The q -th activation of Γ_i completes after a delay Φ (cf. Lemma 1), a jitter j_ϕ , and the execution of q activations of Γ_i . This is given by

$$B_i(q) = q \cdot s_i \cdot \Phi + j_\phi + C_i^{\sigma,s} \quad (7)$$

where $C_i^{\sigma,s}$ is the WCET of Γ_i 's last stage.

Lemma 3 *The Multiple-Event Busy Time $B_i(q)$ given by Eq. 7 is an upper bound.*

Proof The proof is by induction. When $q=1$, Γ_i has to wait for service at most until the next cycle Φ plus an offset jitter j_ϕ to get service for its first stage (Lemma 1), plus the completion of the last stage of the activation lasting $(s_i-1) \cdot \Phi + C_i^{\sigma,s}$. This is given by

$$\begin{aligned} B_i(1) &= (s_i - 1) \cdot \Phi + \Phi + j_\phi + C_i^{\sigma,s} \\ &= s_i \cdot \Phi + j_\phi + C_i^{\sigma,s} \end{aligned} \quad (8)$$

In a subsequent $q+1$ -th activation in the same busy window, Eq. 7 must consider q additional executions of Γ_i . Since Γ_i has s_i stages and only one stage can be activated and executed per cycle Φ , it takes additional $s_i \cdot \Phi$ for each activation of Γ_i . Thus, Eq. 7. \square

Now we can calculate the busy window and WCRT of Γ_i . The busy window w_i of a fork-join task Γ_i is given by

$$w_i = \max_{q \geq 1, q \in \mathbb{N}} \{B_i(q) \mid Q_i(q+1) \geq \delta_i^-(q+1)\} \quad (9)$$

Lemma 4 *The busy window is upper bounded by Eq. 9.*

Proof The proof is by contradiction. Suppose there is a busy window \check{w}_i longer than w_i . In that case, \check{w}_i must contain at least one activation more than w_i , i.e. $\check{q} \geq q + 1$. From Eq. 9, we have that $Q_i(\check{q}) < \delta_i^-(\check{q})$, i.e. \check{q} is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. \square

The response time $R_i(q)$ of the q -th activation of Γ_i in the busy window is given by

$$R_i(q) = B_i(q) - \delta_i^-(q) \quad (10)$$

The worst-case response time R_i^+ is the longest response time of any activation of Γ_i observed in the busy window.

$$R_i^+ = \max_{1 \leq q \leq \eta_i^+(w_i)} R_i(q) \quad (11)$$

Theorem 1 R_i^+ (Eq. 11) provides an upper bound on the worst-case response time of an arbitrary fork-join task Γ_i .

Proof The WCRT of a fork-join task Γ_i is obtained with the busy window approach [43]. It remains to prove that the critical instant leads to the worst-case scheduling scenario, that the interference captured in Eqs. 6 and 7 are upper bounds, and that the busy window is correctly captured by Eq. 9. These are proved in Lemmas 1, 2, 3, and 4, respectively. \square

4.2 Independent Tasks

We now derive the WCRT analysis of an arbitrary independent task τ_i . Two types of interference affect independent tasks: interference caused by higher priority independent tasks and by fork-join tasks. Let us first identify the critical instant leading to the worst-case scheduling scenario where τ_i suffers the most interference.

Lemma 5 *The critical instant of τ_i is when the first event of higher priority independent tasks arrives simultaneously with τ_i 's event at the offset of a fork-join task.*

Proof The worst-case interference caused by a higher priority (independent) task τ_j under SPP is when its first event arrives simultaneously with τ_i 's and continue arriving as early as possible [43].

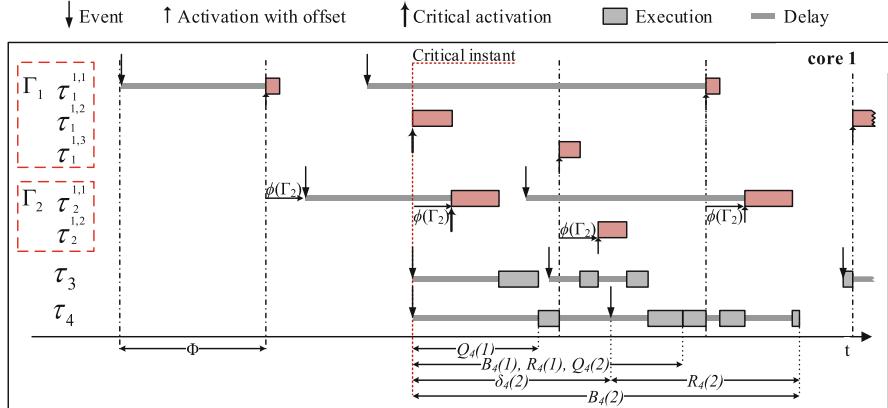


Fig. 7 The worst-case schedule for independent task τ_4 on core 1 (cf. Fig. 5) [34]

The interference caused by a fork-join task Γ_j on τ_i depends on Γ_j 's offset $\phi(\Gamma_j)$ and subtasks $\tau_j^{\sigma,s}$, whose execution times vary for different stages s . Assume a critical instant that occurs at a time other than at the offset $\phi(\Gamma_j)$. Since a task Γ_j starts receiving service at its offset, an event of τ_i arriving at time $t > \phi(\Gamma_j)$ can only suffer less interference from Γ_j 's subtask than when arriving at $t = 0$. \square

Fork-join subtasks have different execution times for different stages, which leads to a number of scheduling scenarios that must be evaluated [31]. Each scenario is defined by the fork-join subtasks that will receive service in the cycle Φ and the offset at which the critical instant supposedly occurs. The scenario is called a critical instant candidate S . Since independent tasks participate in all critical instant candidates, they are omitted in S for the sake of simplicity.

Definition 1 Critical Instant Candidate S : the critical instant candidate S is an ordered pair (a, b) , where a is a critical offset and b is a tuple containing one subtask $\tau_j^{\sigma,s}$ of every interfering fork-join task Γ_j .

Let us also define the set of candidates that must be evaluated.

Definition 2 Critical Instant Candidate Set \mathcal{S} : the set containing all possible different critical instant candidates S .

The worst-case schedule of the independent task τ_4 from the example in Fig. 5 is illustrated in Fig. 7. In fact, the critical instant leading to τ_4 's WCRT is at $\phi(\Gamma_1)$ when $\tau_1^{1,2}$ and $\tau_2^{1,1}$ receive service at the same cycle Φ , i.e. $S = (\phi(\Gamma_1), (\tau_1^{1,2}, \tau_2^{1,1}))$. Events of the independent task τ_3 start arriving at the critical instant and continue arriving as early as possible.

Let us now bound the interference $I_i^I(\Delta t)$ caused by equal or higher priority independent tasks in any time interval Δt . The interference $I_i^I(\Delta t)$ can be upper bounded as follows [27]:

$$I_i^I(\Delta t) = \sum_{\forall \tau_j \in hp_I(i)} \eta_j^+(\Delta t) \cdot C_j \quad (12)$$

where $hp_I(i)$ is the set of equal or higher priority independent tasks mapped to the same core as τ_i .

To derive the interference caused by fork-join tasks we need to define the Critical Instant Event Model. The critical instant event model $\check{\eta}_i^{\sigma,s}(\Delta t, S)$ of a subtask $\tau_i^{\sigma,s} \in \Gamma_i$ returns the maximum number of activations observable in any time interval Δt , assuming the critical instant S . It can be derived from Γ_i 's input event model $\eta_i^+(\Delta t)$ as follows:

$$\check{\eta}_i^{\sigma,s}(\Delta t, S) = \min \{ \eta_i^+(\Delta t_S + \Phi - \phi(\Gamma_i)), \psi \} - gt(s^S, s, \phi^S, \phi(\Gamma_i)) \quad (13)$$

$$\psi = \left\lfloor \frac{\Delta t_S}{\Phi \cdot s_i} \right\rfloor + ge(\Delta t_S \bmod (\Phi \cdot s_i), \Phi \cdot (s - 1)) \quad (14)$$

$$\Delta t_S = \Delta t + \underbrace{\Phi \cdot (s^S - 1)}_{\text{critical instant stage}} + \underbrace{\phi^S}_{\text{critical instant offset}} \quad (15)$$

where s is the stage of subtask $\tau_i^{\sigma,s}$; s_i is the number of stages in Γ_i ; ϕ^S is the offset in S ; s^S is the stage of Γ_i in S ; $gt(a, b, c, d)$ is a function that returns 1 when $(a > b) \vee (a = b \wedge c > d)$, 0 otherwise; and $ge(a, b)$ is a function that returns 1 when $a \geq b$, 0 otherwise.

Lemma 6 $\check{\eta}_i^{\sigma,s}(\Delta t, S)$ (Eq. 13) provides a valid upper bound on the number of activations of $\tau_i^{\sigma,s}$ observable in any time interval Δt , assuming the critical instant S .

Proof The proof is by induction, in two parts. First let us assume $s^S = 1$ and $\phi^S = 0$, neutral values resulting in $\Delta t_S = \Delta t$ and $gt(s^S, s, \phi^S, \phi(\Gamma_i)) = 0$. The maximum number of activations of $\tau_i^{\sigma,s}$ seen in the interval Δt is limited by the maximum number of activations of the fork-join task Γ_i because a subtask $\tau_i^{\sigma,s}$ is activated once per Γ_i 's activation, and limited by the maximum number of times that $\tau_i^{\sigma,s}$ can actually be scheduled and served in Δt . This is ensured in Eq. 13 by the minimum function and its first and second terms, respectively.

When $s^S > 1$ and/or $\phi^S > 0$, the time interval $[0, \Delta t)$ must be moved forward so that it starts at stage s^S and offset ϕ^S . This is captured by Δt_S in Eq. 15 and by the last term of Eq. 13. The former extends the end of the time interval by the time it takes to reach the stage s^S and the offset ϕ^S , i.e. $[0, \Delta t_S]$. The latter pushes the start of the interval forward by subtracting an activation of $\tau_i^{\sigma,s}$ if it occurs before the stage s^S and the offset ϕ^S , resulting in the interval $[\Delta t_S - \Delta t, \Delta t_S)$. Thus Eq. 13. \square

The interference $I_i^{FJ}(\Delta t, S)$ caused by fork-join tasks on the same core in any time interval Δt , assuming a critical instant candidate S , can then be upper bounded

as follows:

$$I_i^{FJ}(\Delta t, S) = \sum_{\forall \tau_j^{\sigma,s} \in hp_{FJ}(i)} \check{\eta}_j^{\sigma,s}(\Delta t, S) \cdot C_j^{\sigma,s} \quad (16)$$

where $hp_{FJ}(i)$ is the set of fork-join subtasks mapped to the same core as τ_i .

The Multiple-Event Queueing Delay $Q_i(q, S)$ and Multiple-Event Busy Time $B_i(q, S)$ for an independent task τ_i , assuming a critical instant candidate S , can be derived as follows:

$$Q_i(q, S) = (q - 1) \cdot C_i + I_i^I(Q_i(q, S)) + I_i^{FJ}(Q_i(q, S), S) \quad (17)$$

$$B_i(q, S) = q \cdot C_i + I_i^I(B_i(q, S)) + I_i^{FJ}(B_i(q, S), S) \quad (18)$$

where $q \cdot C_i$ is the time required to execute q activations of task τ_i .

Equations 17 and 18 result in fixed-point problems, similar to the well-known busy window equation (Eq. 9). They can be solved iteratively, starting with a very small, positive ϵ .

Lemma 7 *The Multiple-Event Queueing Delay $Q_i(q, S)$ given by Eq. 17 is an upper bound, assuming the critical instant S .*

Proof The proof is by induction. When $q = 1$, τ_i has to wait for service until the interfering workload is served. The interfering workload is given by Eqs. 12 and 16. Since $\eta_j^+(\Delta t)$ and C_j are upper bounds by definition, Eq. 12 is also an upper bound. Similarly, since $\check{\eta}_j^{\sigma,s}(\Delta t, S)$ is an upper bound (cf. Lemma 6) and $C_j^{\sigma,s}$ is an upper bound by definition, 16 is an upper bound for a given S . Therefore, $Q_i(1, S)$ is also an upper bound, for a given S .

In a subsequent $q + 1$ -th activation in the same busy window, $Q_i(q, S)$ also must consider q executions of τ_i . This is captured in Eq. 17 by the first term, which is, by definition, an upper bound on the execution time. From that, Lemma 7 follows. \square

Lemma 8 *The Multiple-Event Busy Time $B_i(q, S)$ given by Eq. 18 is an upper bound, assuming the critical instant S .*

Proof The proof is similar to Lemma 7, except that $B_i(q, S)$ in Eq. 18 also captures the completion of the q -th activation. It takes additional C_i , which is an upper bound by definition. Thus Eq. 18 is an upper bound, for a given S . \square

The busy window $w_i(q, S)$ of an independent task τ_i is given by

$$w_i(S) = \max_{q \geq 1, q \in \mathbb{N}} \{B_i(q, S) \mid Q_i(q+1, S) \geq \delta_i^-(q+1)\} \quad (19)$$

Lemma 9 *The busy window is upper bounded by Eq. 19.*

Proof The proof is by contradiction. Suppose there is a busy window $\check{w}_i(S)$ longer than $w_i(S)$. In that case, $\check{w}_i(S)$ must contain at least one activation more than $w_i(S)$,

i.e. $\check{q} \geq q + 1$. From Eq. 19, we have that $Q_i(\check{q}, S) < \delta_i^-(\check{q})$, i.e. \check{q} is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. \square

The response time R_i of the q -th activation of a task in a busy window is given by

$$R_i(q, S) = B_i(q, S) - \delta_i^-(q) \quad (20)$$

Finally, the worst-case response time R_i^+ is found inside the busy window and must be evaluated for all possible critical instant candidates $S \in \mathcal{S}$. The worst-case response time R_i^+ is given by

$$R_i^+ = \max_{S \in \mathcal{S}} \left\{ \max_{1 \leq q \leq \eta_i^+(w_i(S))} \{R_i(q, S)\} \right\} \quad (21)$$

where the set \mathcal{S} is given by the following Cartesian products:

$$\mathcal{S} = \{\phi(\Gamma_j), \phi(\Gamma_k), \dots\} \times \{\sigma_i(\Gamma_j) \times \sigma_i(\Gamma_k) \times \dots\} \quad (22)$$

where $\Gamma_j, \Gamma_k, \dots$ are all fork-join tasks mapped to the same core as τ_i and $\sigma_i(\Gamma_j)$ is the set of subtasks of Γ_j that are mapped to that core. When no fork-join tasks interfere with τ_i , the set $\mathcal{S} = \{(0, 0)\}$.

Theorem 2 R_i^+ (Eq. 21) returns an upper bound on the worst-case response time of an independent task τ_i .

Proof We must first prove that, for a given S , R_i^+ is an upper bound. R_i^+ is obtained with the busy window approach [43]. It returns the maximum response time $R_i(q, S)$ among all activations inside the busy window. From Lemmas 7 and 8 we have that Eqs. 17 and 18 are upper bounds for a given S . From Lemma 9 we have that the busy window is captured by Eq. 19. Since the first term of Eq. 20 is an upper bound and the second term is a lower bound by definition, $R_i(q, S)$ is an upper bound. Thus R_i^+ is an upper bound for a given S . Since Eq. 21 evaluates the maximum response time over all $S \in \mathcal{S}$, R_i^+ is an upper bound on the response time of τ_i . \square

4.3 Error Recovery

Designed for mixed-criticality, our approach supports different recovery strategies for different fork-join tasks (cf. Sect. 2.2). For instance, in DMR augmented with checkpointing and rollback, recovery consists in reverting the state and re-executing the error-affected stage in both replicas. In TMR, recovery consists in copying and replacing the state of the faulty replica with the state of a healthy one. The different

strategies are captured in the analysis by the recovery execution time, which depends on the strategy and the stage to be recovered. The recovery WCET $C_{i,rec}^{\sigma,s}$ of a fork-join subtask $\tau_i^{\sigma,s}$ accounts for the adopted recovery strategy as illustrated in Fig. 4b. Once an error is detected, error recovery is triggered and executed in the recovery slot of the same cycle Φ . Figure 3 illustrates the recovery of the s -th stage of Γ_2 's i -th activation.

Let us incorporate the error recovery into the analysis. For a fork-join task Γ_i , we must only adapt the Multiple-Event Busy Time $B_i(q)$ (Eq. 7) to account for the execution of the recovery:

$$B_i^{rec}(q) = q \cdot s_i \cdot \Phi + j_\phi + \phi(recovery) - \phi(\Gamma_i) + C_{i,rec}^{\sigma,s} \quad (23)$$

where $C_{i,rec}^{\sigma,s}$ is the WCET of the recovery of last subtask of Γ_i . The recovery of another task Γ_j does not interfere with Γ_i 's WCRT. Only the recovery of one of Γ_i 's subtasks can interfere with Γ_i 's WCRT. Moreover, since the recovery of a subtask occurs in the recovery slot of the same cycle Φ and does not interfere with the next subtask, only the recovery of the last stage of Γ_i actually has an impact on its response time. This is captured by the three last terms of Eq. 23.

For an independent task τ_i , the worst-case impact of recovery of a fork-join task Γ_j is modeled as an additional fork-join task Γ_{rec} with one subtask $\tau_{rec}^{\sigma,1}$ mapped to the same core as τ_i and that executes in the *recovery* slot. The WCET $C_{rec}^{\sigma,1}$ of $\tau_{rec}^{\sigma,1}$ is chosen as the maximum recovery time among the subtasks of all fork-join tasks mapped to that core:

$$C_{rec}^{\sigma,1} = \max_{\forall \tau_j^{\sigma,s} \in hp_{FJ}(i)} \{C_{i,rec}^{\sigma,s}\} \quad (24)$$

with Γ_{rec} mapped, Eq. 21 finds the critical instant, where the recovery $C_{rec}^{\sigma,1}$ has the worst impact on the response time of τ_i .

5 Experimental Evaluation

In our experiments we evaluate our approach with real as well as synthetic workloads, focusing on the performance of the scheduler. First we characterize MiBench applications [18] and evaluate them as fork-join (replicated) tasks in the system. Then we evaluate the performance of independent (regular) tasks. Finally we evaluate the approach with synthetic workloads when varying parameters of fork-join tasks.

Table 1 MiBench applications' profile [34]

	WCET [ms]	Observed stages		Grouped stages	
		#stages	Max WCET [ms]	#stages	Max WCET [ms]
basicmath	32.48	19,738	0.02	5	6.50
bitcount	24.42	30	15.16	3	15.16
susan	9.63	12	9.59	1	9.63
blowfish	0.11	7	0.09	1	0.11
rijndael	13.17	93	0.37	3	5.91
sha	3.49	51	0.11	2	1.90

5.1 Evaluation with Benchmark Applications

5.1.1 Characterization

First we extract execution times and number of stages from MiBench automotive and security applications [18]. They were executed with small input on an ARMv7@1 GHz and a memory subsystem including a DDR3-1600 DRAM [8]. Table 1 summarizes the total WCET, *observed* number of stages, and WCET of the longest stage (max). A stage is delimited by *syscalls* (cf. Sect. 2.2). We report the observed execution times as WCETs. As pointed out in [2], stages vary in number and execution time depending on the application and on the current activity in that stage (computation/IO). This is seen, e.g., in *susan*, where 99% of the WCET is concentrated in one stage (computation) while the other stages perform mostly IO and are on average 3.34 µs long.

In our approach, the optimum is when all stages of a fork-join task have the same WCET. There are two possibilities to achieve that: to *aligned* very long stages in shorter ones or to *group* short, subsequent stages together. We exploit the latter as it does not require changes to the error detection mechanism or to our model. The results with *grouped* stages are shown on the right-hand side of Table 1. We have first grouped stages without increasing the maximum stage length. The largest improvement is seen in *bitcount*, where the number of stages reduces by one order of magnitude. In cases where all stages are very short, we increase the maximum stage length. When increasing the maximum stage length by two orders of magnitude, the number of stages of *basicmath* reduces by four orders of magnitude. We have manually chosen the maximum stage length. Alternatively the problem of finding the maximum stage length can be formulated as an optimization problem that, e.g., minimizes the overall WCRT or maximizes the slack. Next, we map the applications as fork-join tasks and evaluate their WCRTs.

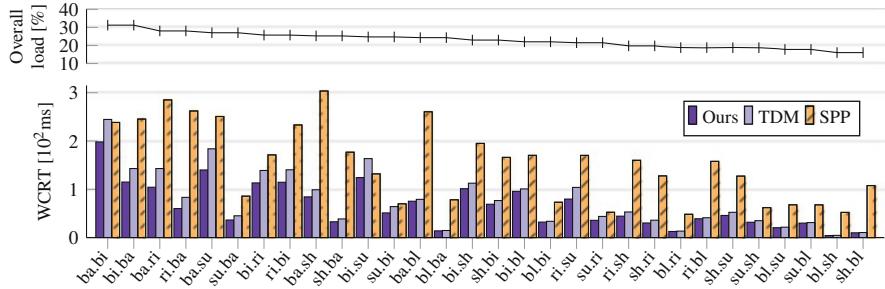


Fig. 8 WCRT of fork-join tasks with two segments derived from MiBench [34]

5.1.2 Evaluation of Fork-Join Tasks

Two applications at a time are mapped as fork-join tasks with two segments (i.e., replicas in DMR) to two cores (cf. Fig. 5). On each core, 15% load is introduced by ten independent tasks generated with UUniFast [7]. We compare our approach with a TDM-based scheduler and Axer’s Partitioned SPP [2]. In TDM, each fork-join task executes (and recovers) in its own slot. Independent tasks execute in a third slot, which replaces the recovery slot of our approach. The size of the slots is derived from our offsets. For all approaches, the priority assignment for independent tasks is deadline monotonic and considers that deadline equals period. In SPP, the deadline monotonic priority assignment also includes fork-join tasks.

The results are plotted in Fig. 8, where *ba.bi* gives the WCRT of *basicmath* when mapped together with *bitcount*. Despite the low system load, our approach also outperforms SPP in all cases, with bounds 58.2% lower, on average. Better results with SPP cannot be obtained unless the interfering workload is removed or highest priority is given to the fork-join tasks [2], which violates DM. Despite the similarity of how our approach handles fork-join tasks with TDM, the proposed approach outperforms TDM in all cases, achieving, on average, bounds 13.9% lower. This minor difference is because TDM slots must be slightly longer than our offsets to fit an eventual recovery. Nonetheless, not only our approach can guarantee short WCRT for replicated tasks but also provides for the worst-case performance of independent tasks.

5.1.3 Evaluation of Independent Tasks

In a second experiment we fix *bitcount* and *rijndael* as fork-join tasks and vary the load on both cores. The generated task periods are in the range [20, 500] ms, larger than the longest stage of the fork-join tasks. The schedulability of the system as the load increases is shown in Fig. 9. Our approach outperforms TDM and SPP in all cases, scheduling 1.55× and 6.96× more tasksets, respectively. Due to its non-work conserving characteristic, TDM’s schedulability is limited

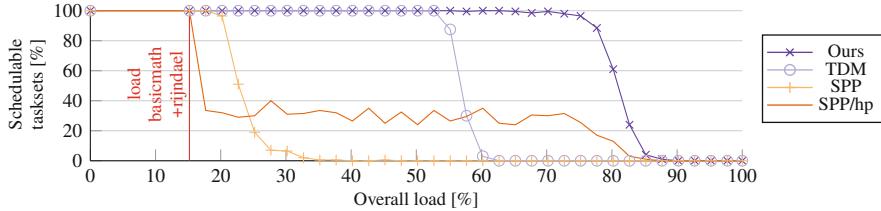


Fig. 9 Schedulability as a function of the load of the system. *Basicmath* and *rijndael* as fork-join tasks with two segments [34]

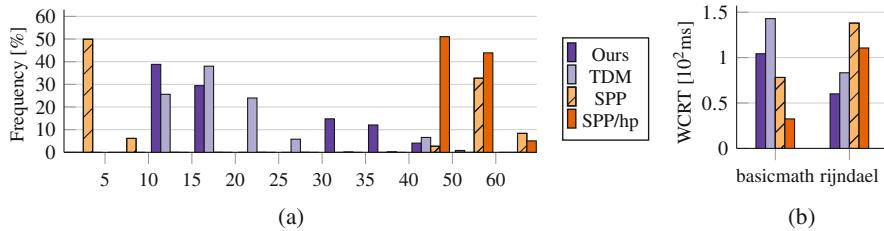


Fig. 10 *Basicmath* and *rijndael* as replicated tasks in DMR running on a dual-core configuration with 20.2% load (5% load from independent tasks) [34]. (a) WCRT of independent tasks [ms]. (b) WCRT of FJ tasks

to medium loads. SPP provides very short response times with lower loads but, as the load increases, the schedulability drops fast due to high interference (and thus high WCRT) suffered by fork-join tasks. For reference purposes, we also plot the schedulability of SPP when assigning the highest priorities to the fork-join tasks (SPP/hp). The schedulability in higher loads improves but losing deadline monotonicity guarantees renders the systems unusable in practice. Moreover, when increasing the jitter to 20% (relative to period), schedulability decreases 14.2% but shows the same trends for all schedulers.

Figure 10 details the tasks’ WCRTs when the system load is 20.2%. Indeed, when schedulable, SPP provides some of the shortest WCRTs for independent tasks, and SPP/hp improves the response times of fork-join tasks at the expense of the independent tasks’. Our approach provides a balanced trade-off between the performance of independent tasks and of fork-join tasks, and achieves high schedulability even in higher loads.

5.2 Evaluation with Synthetic Workload

We now evaluate the performance of our approach when varying parameters such as stage length and cycle Φ .

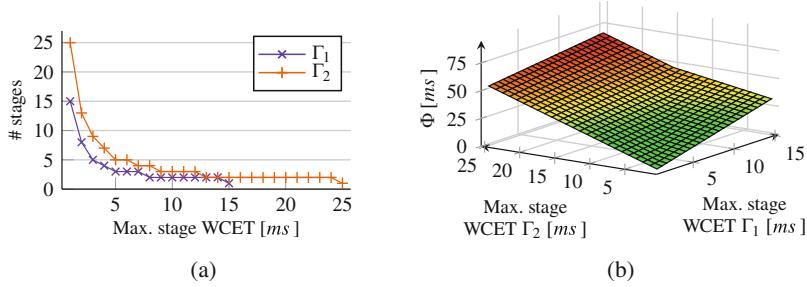


Fig. 11 Parameters of two fork-join tasks Γ_1 and Γ_2 with two segments each (i.e., replicas in DMR) are mapped to two cores. The total WCETs³ of Γ_1 and Γ_2 are 15 and 25ms, respectively. Both tasks are sporadic, with a minimum distance of 1s between activations. The number of stages of Γ_1 and Γ_2 is varied as a function of the maximum stage WCET, as depicted in Fig. 11a. The length of the cycle Φ , depicted in Fig. 11b, varies with the maximum stage WCET since it is derived from them (cf. Sect. 3.4).

5.2.1 Evaluation of Fork-Join Tasks

Two fork-join tasks Γ_1 and Γ_2 with two segments each (i.e., replicas in DMR) are mapped to two cores. The total WCETs³ of Γ_1 and Γ_2 are 15 and 25ms, respectively. Both tasks are sporadic, with a minimum distance of 1s between activations. The number of stages of Γ_1 and Γ_2 is varied as a function of the maximum stage WCET, as depicted in Fig. 11a. The length of the cycle Φ , depicted in Fig. 11b, varies with the maximum stage WCET since it is derived from them (cf. Sect. 3.4).

The system performance as the maximum stage lengths of Γ_1 and Γ_2 increase is reported in Fig. 12. The WCRT of Γ_1 increases with the stage length (Fig. 12a) as it depends on the number of stages and Φ 's length. In fact, the WCRT of Γ_1 is longest when the stages of Γ_1 are the shortest and the stages of the interfering fork-join task (Γ_2) are the longest. Conversely, WCRT of Γ_1 is shortest when its stages are the longest and the stages of the interfering fork-join task are the shortest. The same occurs to Γ_2 in Fig. 12b. Thus, there is a trade-off between the response times of interfering fork-join tasks. This is plotted in Fig. 13 as the sum of the WCRTs of Γ_1 and Γ_2 . As can be seen in Fig. 13, low response times can be obtained next and above to the line segment between the origin (0, 0, 0) and the point (15, 25, 0), the total WCETs¹ of Γ_1 and Γ_2 , respectively.

5.2.2 Evaluation of Independent Tasks

To evaluate the impact of the parameters on independent tasks, we extend the previous scenario introducing 25% load on each core with ten independent tasks generated with UUniFast [7]. The task periods are within the interval [15, 500] ms for the first experiment, and the interval [25, 500] ms for the second. The priority

³The sum of the WCET of all stages of a fork-join task.

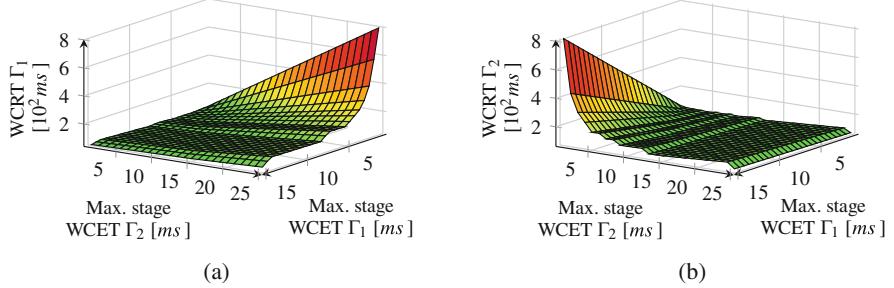


Fig. 12 Performance of fork-join tasks Γ_1 and Γ_2 as a function of the maximum stage WCET [34]. (a) WCRT of Γ_1 . (b) WCRT of Γ_2

Fig. 13 WCRT trade-off between interfering fork-join tasks [34]

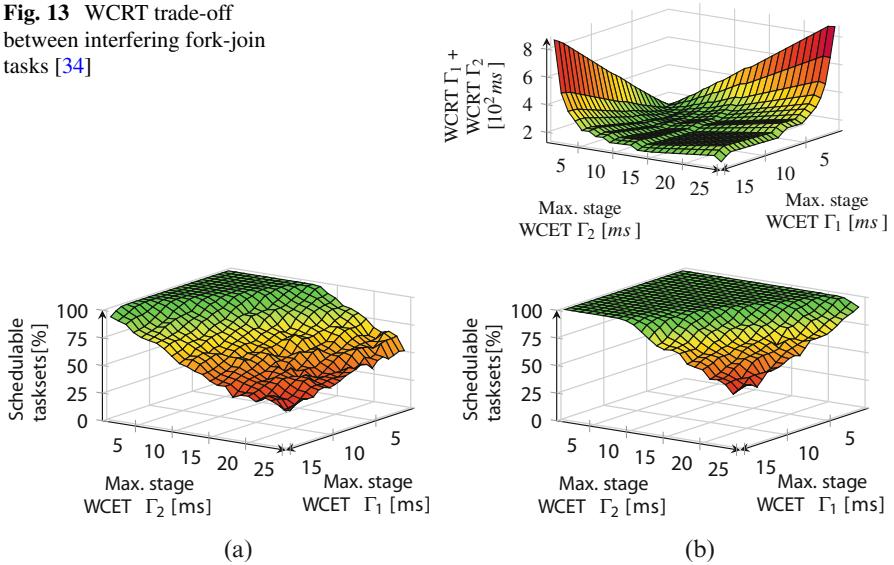


Fig. 14 Schedulable tasksets as a function of the maximum stage WCET of fork-join tasks Γ_1 and Γ_2 with 25% load from independent tasks [34]. (a) Task period interval [15–500] ms. (b) Task period interval [25–500] ms

assignment is deadline monotonic and considers that the deadline is equal to the period.

The schedulability as a function of the stage lengths is shown in Fig. 14. Sufficiently long stages cause the schedulability to decrease as independent tasks with short periods start missing their deadlines. This is seen in Fig. 14a when the stage length of either fork-join task reaches 15 ms, the minimum period for the generated tasksets. Thus, when increasing the minimum period of generated tasks to 25 ms, the number of schedulable tasksets also increases (Fig. 14b).

The maximum stage length of a fork-join task has direct impact on the response times and schedulability of the system. For the sake of performance, shorter stage

lengths are preferred. However, that is not always possible because it would result in a large number of stages or because of the application, which restricts the minimum stage length (cf. Sect. 5.1.1). Nonetheless, fork-join tasks still are able to perform well with appropriate parameter choices. Additionally, one can formulate the problem of finding the stage lengths according to an objective function, such as minimize the overall response time or maximize the slack. The offsets can also be included in the formulation, as long as Constraints 1 and 2 are met.

6 Conclusion

This chapter started with an overview of the project ASTEROID. ASTEROID developed a cross-layer fault-tolerance approach to provide reliable software execution on unreliable hardware. The approach is based on replicated software execution and exploits the large number of cores available in modern and future architectures at a higher level of abstraction without resorting to the inefficient hardware redundancy. The chapter then focused on the performance of replicated execution and the replica-aware co-scheduling, which was developed in ASTEROID.

The replica-aware co-scheduling for mixed-critical systems, where applications with different requirements and criticalities co-exist, overcomes the performance limitations of standard schedulers such as SPP and TDM. A formal WCRT analysis was presented, which supports different recovery strategies and accounting for the NoC communication delay and overheads due to replica management and state comparison. The replica-aware co-scheduling provides for high worst-case performance of replicated software execution on many-core architectures without impairing the remaining tasks in the system. Experimental results with benchmark applications showed an improvement on taskset schedulability of up to $6.9 \times$ when compared to Partitioned SPP and $1.5 \times$ when compared to a TDM-based scheduler.

References

1. Andersson, B., de Niz, D.: Analyzing Global-EDF for multiprocessor scheduling of parallel tasks. In: International Conference On Principles Of Distributed Systems, pp. 16–30. Springer, Berlin (2012)
2. Axer, P.: Performance of time-critical embedded systems under the influence of errors and error handling protocols. Ph.D. Thesis, TU Braunschweig (2015)
3. Axer, P., Sebastian, M., Ernst, R.: Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2011)
4. Axer, P., Sebastian, M., Ernst, R.: Probabilistic response time bound for CAN messages with arbitrary deadlines. In: 2012 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1114–1117. IEEE, Piscataway (2012)

5. Axer, P., Ernst, R., Döbel, B., Härtig, H.: Designing an analyzable and resilient embedded operating system. In: Proceedings of Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES'12), Braunschweig (2012)
6. Axer, P., Quinton, S., Neukirchner, M., Ernst, R., Dobel, B., Hartig, H.: Response-time analysis of parallel fork-join workloads with real-time constraints. In: Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'13) (2013)
7. Bini, E., Buttazzo, G.C.: Measuring the performance of schedulability tests. *Real-Time Syst.* **30**(1–2), 129–154 (2005)
8. Binkert, N., Beckmann, B., Black, G., et al.: The Gem5 simulator. *SIGARCH Comput. Archit. News* **39**(2) (2011). <https://doi.org/10.1145/2024716.2024718>
9. Döbel, B., Härtig, H.: Who watches the watchmen? – protecting operating system reliability mechanisms. In: International Workshop on Hot Topics in System Dependability (HotDep'12) (2012)
10. Döbel, B., Härtig, H.: Where have all the cycles gone? Investigating runtime overheads of OS-assisted replication. In: Proceedings of Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES'13), pp. 2534–2547 (2013)
11. Döbel, B., Härtig, H.: Can we put concurrency back into redundant multithreading? In: Proceedings of the International Conference on Embedded Software (EMSOFT'14) (2014)
12. Döbel, B., Härtig, H., Engel, M.: Operating system support for redundant multithreading. In: Proceedings of the International Conference on Embedded Software (EMSOFT'12) (2012)
13. Engel, M., Döbel, B.: The reliable computing base-a paradigm for software-based reliability. In: Proceedings of Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES'12), pp. 480–493 (2012)
14. Feitelson, D.G., Rudolph, L.: Gang scheduling performance benefits for fine-grain synchronization. *J. Parallel Distrib. Comput.* **16**(4), 306–318 (1992)
15. Gaillard, R.: Single Event Effects: Mechanisms and Classification. In: Soft Errors in Modern Electronic Systems. Springer, New York (2011)
16. Goossens, J., Berten, V.: Gang ftp scheduling of periodic and parallel rigid real-time tasks (2010). arXiv:1006.2617
17. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., et al.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**(1), 8–23 (2012)
18. Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R.: Mibench: a free, commercially representative embedded benchmark suite. In: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization (WWC-4). 2001 (2001)
19. Härtig, H., Roitzsch, M., Weinhold, C., Lackorzynski, A.: Lateral thinking for trustworthy apps. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1890–1899. IEEE, Piscataway (2017)
20. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System Level Performance Analysis—the SymTA/S Approach. *IEE Proc.-Comput. Digit. Tech.* **152**, 148–166 (2005)
21. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M., Teich, J., Wehn, N., Wunderlich, H.J.: Design and architectures for dependable embedded systems. In: Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '11, pp. 69–78. ACM, New York (2011). <http://doi.acm.org/10.1145/2039370.2039384>
22. Herkersdorf, A., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectron. Reliab.* **54**(6–7), 1066–1074 (2014)
23. Hoffmann, M., Lukas, F., Dietrich, C., Lohmann, D.: dOSEK: the design and implementation of a dependability-oriented static embedded kernel. In: 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'15) (2015)
24. International Standards Organization: ISO 26262: Road Vehicles – Functional Safety (2011)

25. Kaiser, R., Wagner, S.: Evolution of the PikeOS microkernel. In: First International Workshop on Microkernels for Embedded Systems (2007)
26. Kato, S., Ishikawa, Y.: Gang EDF scheduling of parallel task systems. In: Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS'09) (2009)
27. Lehoczky, J.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings 11th Real-Time Systems Symposium (RTSS'90) (1990)
28. Leuschner, L., Küttler, M., Stumpf, T., Baier, C., Härtig, H., Klüppelholz, S.: Towards automated configuration of systems with non-functional constraints. In: Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotDep'17), pp. 111–117. ACM, New York (2017)
29. NXP MPC577xK Ultra-Reliable MCU Family (2017). <http://www.nxp.com/assets/documents/data/en/fact-sheets/MPC577xKFS.pdf>
30. Ousterhout, J.K.: Scheduling techniques for concurrent systems. In: International Conference on Distributed Computing (ICDCS), vol. 82, pp. 22–30 (1982)
31. Palencia, J.C., Harbour, M.G.: Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings 19th IEEE Real-Time Systems Symposium (RTSS'98) (1998)
32. Rambo, E.A., Ernst, R.: Providing flexible and reliable on-chip network communication with real-time constraints. In: 1st International Workshop on Resiliency in Embedded Electronic Systems (REES) (2015)
33. Rambo, E.A., Ernst, R.: Worst-case communication time analysis of networks-on-chip with shared virtual channels. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'15) (2015)
34. Rambo, E.A., Ernst, R.: Replica-aware co-scheduling for mixed-criticality. In: 29th Euromicro Conference on Real-Time Systems (ECRTS 2017), vol. 76, pp. 20:1–20:20 (2017). <https://doi.org/10.4230/LIPIcs.ECRTS.2017.20>
35. Rambo, E.A., Ahrendts, L., Diemer, J.: FMEA of the IDAMC NoC. Tech. Rep., Institute of Computer and Network Engineering – TU Braunschweig (2013)
36. Rambo, E.A., Tschiene, A., Diemer, J., Ahrendts, L., Ernst, R.: Failure analysis of a network-on-chip for real-time mixed-critical systems. In: 2014 Design, Automation Test in Europe Conference Exhibition (DATE) (2014)
37. Rambo, E.A., Tschiene, A., Diemer, J., Ahrendts, L., Ernst, R.: FMEA-based analysis of a network-on-chip for mixed-critical systems. In: Proceedings of the Eighth IEEE/ACM International Symposium on Networks-on-Chip (NOCS'14) (2014)
38. Rambo, E.A., Saidi, S., Ernst, R.: Providing formal latency guarantees for ARQ-based protocols in networks-on-chip. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'16) (2016)
39. Rambo, E.A., Seitz, C., Saidi, S., Ernst, R.: Designing networks-on-chip for high assurance real-time systems. In: Proceedings of the IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC'17) (2017)
40. Rambo, E.A., Seitz, C., Saidi, S., Ernst, R.: Bridging the gap between resilient networks-on-chip and real-time systems. IEEE Trans. Emer. Top. Comput. **8**, 418–430 (2020). <http://doi.org/10.1109/TETC.2017.2736783>
41. Richter, K.: Compositional scheduling analysis using standard event models. Ph.D. Thesis, TU Braunschweig (2005)
42. RTCA Incorporated: DO-254: Design Assurance Guidance For Airborne Electronic Hardware (2000)
43. Tindell, K., Burns, A., Wellings, A.: An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Syst. **6**(2), 133–151 (1994)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Dependability Aspects in Configurable Embedded Operating Systems



Horst Schirmeier, Christoph Borchert, Martin Hoffmann, Christian Dietrich,
Arthur Martens, Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk

1 Introduction

Future hardware designs for embedded systems will exhibit more parallelism and energy efficiency at the price of being less reliable, due to shrinking structure sizes, increased clock frequencies, and lowered operating voltages [9]. In embedded control systems, the handling of soft errors—e.g., transient bit flips in the memory hierarchy—is becoming mandatory for all safety integrity level (SIL) 3 or SIL 4 categorized safety functions [30, 35]. Established solutions stem mostly from the avionics domain and employ extensive hardware redundancy or specifically hardened hardware components [55]—both of which are too costly to be deployed in commodity products.

H. Schirmeier (✉)

Embedded System Software Group, TU Dortmund, Dortmund, Germany
e-mail: horst.schirmeier@tu-dortmund.de

C. Borchert · O. Spinczyk

Embedded Software Systems Group, Osnabrück University, Osnabrück, Germany
e-mail: christoph.borchert@uos.de; olaf.spinczyk@uos.de

M. Hoffmann

System Software Group, FAU Erlangen, Erlangen, Germany
e-mail: hoffmann@cs.fau.de

C. Dietrich · D. Lohmann

Systems Research and Architecture Group, Leibniz University Hannover, Hannover, Germany
e-mail: dietrich@sra.uni-hannover.de; lohmamn@sra.uni-hannover.de

A. Martens · R. Kapitza

Institute of Operating Systems and Computer Networks, TU Braunschweig, Braunschweig,
Germany
e-mail: martens@ibr.cs.tu-bs.de; kapitza@ibr.cs.tu-bs.de

Software-based redundancy techniques, especially redundant execution with majority voting in terms of TMR, are well-established countermeasures against soft errors on the application level [24]. By combining them with further techniques—such as arithmetic codes—even the voter as the single point of failure (SPOF) can be eliminated [53]. However, all these techniques “work” only under the assumption that the application is running on top of a soft-error-resilient system-software stack.

In this chapter, we address the problem of software-stack hardening for three different points in the system-software and fault-tolerance technique design space:

- In Sect. 3 we investigate soft-error hardening techniques for a statically configured OS, which implements the automotive OSEK/AUTOSAR real-time operating system (RTOS) standard [5, 40]. We answer the research question what the *general reliability limits* in this scenario are when aiming at *reliability as a first-class design goal*. We show that harnessing the static application knowledge available in an AUTOSAR environment, and protecting the OS kernel with AN-encoding, yields an extremely reliable software system.
- In Sect. 4 we analyze how programming-language and compiler extensions can help to modularize fault-tolerance mechanisms. By applying the resulting fault-tolerance modules to a *dynamic* commercial off-the-shelf (COTS) embedded OS, we explore how far reliability can be pushed when a legacy software stack needs to be maintained. We show that aspect-oriented programming (AOP) is suitable for encapsulating generic software-implemented hardware fault tolerance (SIHFT) mechanisms, and can improve reliability of the targeted software stack by up to 79%.
- Looking beyond bit flips in the memory hierarchy, in Sect. 5 we investigate how a system-software stack can survive even more adverse fault models such as whole-system outages. Using persistent memory (PM) technology for state conservation, our findings include that software transactional memory (STM) facilitates maintaining state consistency and allows fast recovery.

These works have been previously published in conference proceedings and journals [8, 29, 36], and are presented here in a summarized manner. Section 6 concludes the chapter and summarizes the results of the *DanceOS* project, which was funded by the German Research Foundation (DFG) over a period of 6 years as part of the priority program SPP 1500 “Dependable Embedded Systems” [26] (Fig. 1).

2 Related Work

Dependable Embedded Operating Systems While most work from the dependable-systems community still assumes the OS itself to be too hard to protect, the topic of RTOS reliability in case of transient faults has recently gained attention. The C³ μ-kernel tracks system-state transitions at the inter-process communication (IPC) level to be able to recover system components in case of a fault [50]. Their approach,

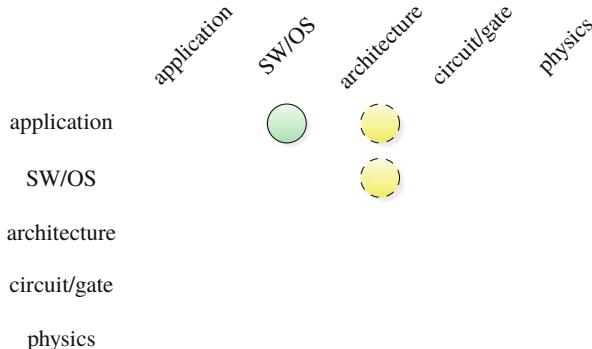


Fig. 1 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

however, assumes that faults are detected immediately and never turn into silent data corruptions (SDCs), and that the recovery functionality itself is part of the RCB. L4/Romain [19] employs system-call interception to provide transparent thread-level TMR—and, hence, error detection—but still requires a reliable μ -kernel. The hypervisor approach of Quest-V [34] reduces the *software*-part of the RCB even further—at the price of increasing the *hardware*-part for the required virtualization support. In the end, however, all these approaches assume early and reliable detection of faults and their strict containment inside the RCB, which our three approaches provide.

Software-Based Soft-Error Detection and Correction The concept of AN-encoding has been known for quite a while and has been taken up in recent years in compiler- and interpreter-based solutions [45]. Yet, these generic realizations are not practicable for realizing a RCB—not only due their immense runtime overhead of a factor of 10^3 up to 10^5 , but also due to the specific nature of low-level system software. Thus, following our proven CoRed concept [28], we concentrate the encoded execution to the *minimal necessary points*. Besides AN-encoding, several more generic *error detection and recovery mechanisms* (EDMs/ERMs) exist and have been successfully deployed. Shirvani et al. [48] evaluate several software-implemented error-correcting codes for application in a space satellite to obviate the use of a low-performance radiation-hardened CPU and memory. Read-only data segments are periodically scrubbed to correct memory errors, whereas protected *variables* must be accessed manually via a special API to perform error correction. Similarly, *Samurai* [41] implements a C/C++ dynamic memory allocator with a dedicated API for access to replicated heap memory. Programmers have to manually invoke functions to check and update the replicated memory chunks. The latter approach exposes the heap allocator as single point of failure, which is not resilient against memory errors. To automate the hardening process, some works extend *compilers* for transforming code to add fault tolerance [44]. These

approaches are based on duplicating or even triplicating important *variables* of single-threaded user-level programs. Our work differs in that we use the general-purpose AspectC++ compiler that allows us to focus on the implementation of software-based EDM/ERMs in the OS/application layer, instead of implementing special-purpose compilers. AOP also allows to separate the “business logic” from fault-tolerance implementations, which has, e.g., been pioneered by Alexandersson et al. [2]—however at the cost of 300% runtime overhead.

State Consistency in Non-volatile Memories Maintaining state consistency in persistent memory has been achieved on the level of process-wide persistence [10, 39] and specialized file systems [13, 20]. Our *DNV Memory* approach shares the most similarities with libraries that provide safe access to a persistent heap [6, 12, 54]. Mnemosyne [54] shows the overall steps that are needed to build a persistent heap, while NV-Heaps [12] focuses mainly on usability aspects. Both libraries rely on a transactional-memory model that stores logs in persistent memory and executes expensive flush operations to ensure data consistency in presence of power failures. In order to improve performance, the memory allocator of Makalu [6] guarantees the consistency of its own meta data without the need of transactions. However, it does not extend this ability to the data stored within. Thus, library support, similar to Mnemosyne [54], is still needed to enforce durability. *DNV Memory* shares with these approaches the transactional model and the goal to provide a persistent heap, but aims at improving performance and lifetime of persistent applications by reducing the amount of writes to persistent memory. Additionally, *DNV Memory* provides transparent dependability guarantees that none of the previous work has covered.

3 dOSEK: A Dependable RTOS for Automotive Applications

In the following, we present the design and implementation of *dOSEK*, an OSEK/AUTOSAR-conforming [5, 40] RTOS that serves as reliable computing base (RCB) for safety-critical systems. *dOSEK* has been developed from scratch with dependability as the first-class design goal based on a two-pillar design approach: First we aim for strict *fault avoidance*¹ by an in-depth static tailoring of the kernel towards the concrete application and hardware platform—without restricting the required RTOS services. Thereby, we constructively minimize the (often redundant) vulnerable runtime state. The second pillar is to then constructively reintegrate redundancy in form of dependability measures to eliminate the remaining SDCs in the essential state. Here, we concentrate—in contrast to others [4, 50]—on reliable *fault detection* and fault containment within the kernel execution path (Sect. 3.2) by

¹Strictly speaking, we aim to avoid *errors* resulting from transient hardware faults.

employing arithmetic encoding [23] to realize self-contained data and control-flow error detection across the complete RTOS execution path.

We evaluate our hardened *dOSEK* against ERIKA [21], an industry-grade open-source OSEK implementation, which received an official OSEK/VDX certification (Sect. 3.3). We present the runtime and memory overhead as well as the results of extensive fault-injection campaigns covering the *complete* fault space of single-bit faults in registers and volatile memory. Here, *dOSEK* shows an improvement of four orders of magnitude regarding the SDC count, compared to ERIKA.

3.1 Development of a Fault-Avoiding Operating System

Essentially, a transient fault can lead to an error inside the kernel only if it affects either the kernel’s control or data flow. For this, it has to hit a memory cell or register that carries *currently alive* kernel state, such as a global variable (always alive), a return address on the stack (alive during the execution of a system call), or a bit in the status register of the CPU (alive only immediately before a conditional instruction). Intuitively, the more long-living state a kernel maintains, the more prone it is to transient faults. Thus, our first rule of fault-avoiding OS development is: **① Minimize the time spent in system calls and the amount of volatile state, especially of global state that is alive across system calls.**

However, no kernel can provide useful services without any runtime state. So, the second point to consider is the containment and, thus, detectability of data and control-flow errors by local sanity checks. Intuitively, bit flips in pointer variables have a much higher error range than those used in arithmetic operations; hence, they are more likely to lead to SDCs. In a nutshell, any kind of indirection at runtime (through data or function pointers, index registers, return addresses, and so on) impairs the inherent robustness of the resulting system. Thus, our second rule of fault-avoiding operating-system development is: **② Avoid indirections in the code and data flow.**

In *dOSEK*, we implement these rules by an extensive static analysis of the application code followed by a subsequent dependability-oriented “pointer-less” generation of the RTOS functionality. Our approach follows the OSEK/AUTOSAR system model of static tailoring [5, 40], which in itself already leads to a significant reduction of state and SDC vulnerability [27]. We amplify these already good results by a flow-sensitive analysis of all application–RTOS interactions [17, 18] in order to perform a partial specialization of system calls: Our system generator specializes each system call *per invocation* to embed it into the particular application code. This facilitates an aggressive folding of parameter values into the code. Therefore, less state needs to be passed in volatile registers or on the stack (rule ①). We further achieve a pointer-less design by allocating all system objects statically as global data structures, with the help of the generator. In occasions where pointers would be used to select one object out of multiple possible candidates, an array at a constant address with small indices is preferred (rule ②).

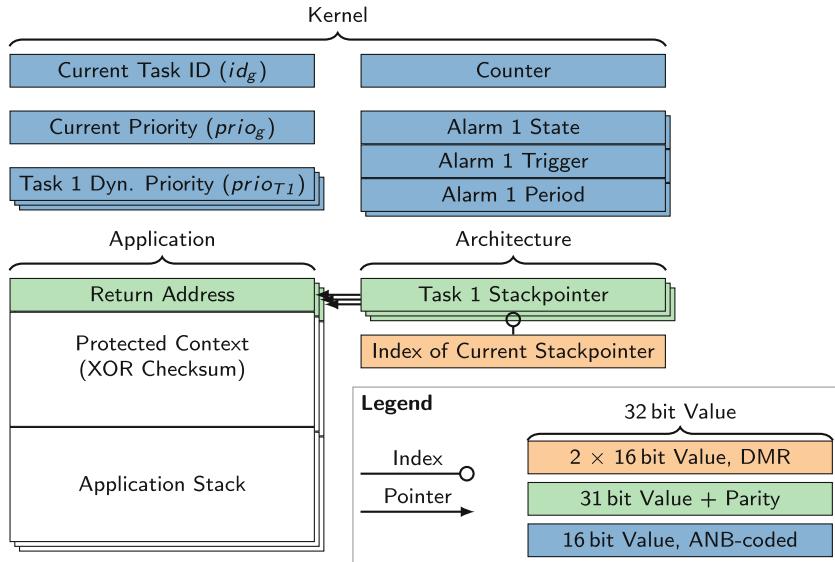


Fig. 2 Overview of the OS data kept in RAM of an example system composed of three tasks and two alarms. Each box represents a 32-bit memory location. All kernel data are hardened using an ANB-Code. The remaining application- and architecture-specific values are safeguarded by dual modular redundancy (DMR) or parity bits

Figure 2 depicts the resulting state of this analysis by the example of a system consisting of three tasks and two alarms: The remaining volatile state variables are subsumed under the blocks *Application*, *Architecture*, and *Kernel*. The architecture-independent minimal *Kernel* state is condensed to two machine words for the current task's priority, its id, and one machine word per task for the task's dynamic priority according to the priority ceiling protocol. Depending on the requirements of the application, the kernel maintains the current state of additional resources: in this case two alarms (three machine words each) and one counter (one machine word). The *Architecture* blocks are related to the dispatching mechanism of the underlying processor. In case of the IA-32, this is reduced to the administration of one stack pointer per task.

The most frequently used (but far less visible) pointers are the stack pointer and the base pointer. Albeit less obvious, they are significant: A corrupted stack pointer influences all local variables, function arguments, and the return address. Here, we eliminated the indirection for local variables by storing them as static variables at fixed, absolute addresses, while keeping isolation in terms of visibility and memory protection (rule ②). Furthermore, by aggressively inlining the specialized system calls into the application code, we reduce the spilling of parameter values and return addresses onto the vulnerable stack, while keeping the hardware-based spatial isolation (MPU/MMU-based AUTOSAR memory protection) between applications and kernel using inline traps [15] (rule ①).

3.2 Implementing a Fault-Detecting Operating System

dOSEK's *fault-detection* strategies can be split up into two complementary concepts: First, coarse-grained hardware-based fault-detection mechanisms, mainly by means of MPU-based memory and privilege isolation. Second, fine-grained software-based concepts that protect the kernel-internal data/control flows.

Hardware-based isolation by watchdogs and memory protection units (MPUs) are a widely used and a proven dependability measure. Consequently, *dOSEK* integrates the underlying architecture's mechanisms into its system design, leveraging a coarse-grained fault detection between tasks and the kernel. We furthermore employ hardware-based isolation to minimize the set of kernel-writable regions during a system call, which leverages additional error-detection capabilities for faulty memory writes from the kernel space. With our completely generative approach, all necessary MPU configurations can be derived already at compile time and placed in robust read-only memory (ROM).

The execution of the *dOSEK* kernel itself is hardened with a fine-grained arithmetic encoding. All kernel data structures are safeguarded using a variant of an AN-code [23] capable of detecting both data- and control-flow errors. The code provides a constant common key A , allowing to uncover errors when calculating the remainder, and a variable-specific, compile-time constant signature B_n detecting the mix-up of two encoded values as well as the detection of faulty control flows—the ANB-Code:

$$n_{enc} = A \cdot n + B_n$$

↑ ↑ ↑ ↑
Encoded Value Key Value Signature

A particular feature of arithmetic codes is a set of code-preserving arithmetic operations, which allow for computation with the encoded values. Hence, a continuous sphere of redundancy is spanned, as the corresponding operands remain encoded throughout the entire kernel execution.

In addition to the existing elementary arithmetic operations, *dOSEK* also requires an encoded variant of the mandatory OSEK/AUTOSAR fixed-priority scheduling algorithm [40]: The encoded scheduler is based on a simple prioritized task list. Each task's current dynamic priority is stored at a fixed location (see also Fig. 2), with the lowest possible value, an encoded zero, representing the suspended state. To determine the highest-priority task, the maximum task priority is searched by comparing all task priorities sequentially. Thus, the algorithm's complexity in space and time is linear to the constant number of tasks. Figure 3 shows the basic concept for three tasks: The sequence processes a global tuple of ANB-encoded values storing the current highest-priority task id found so far, and the corresponding priority ($\langle id_g, prio_g \rangle$, see Fig. 2). Sequential compare-and-update operations, based on an encoded greater-equal decision on a tuple of values (ge_tuple), compare

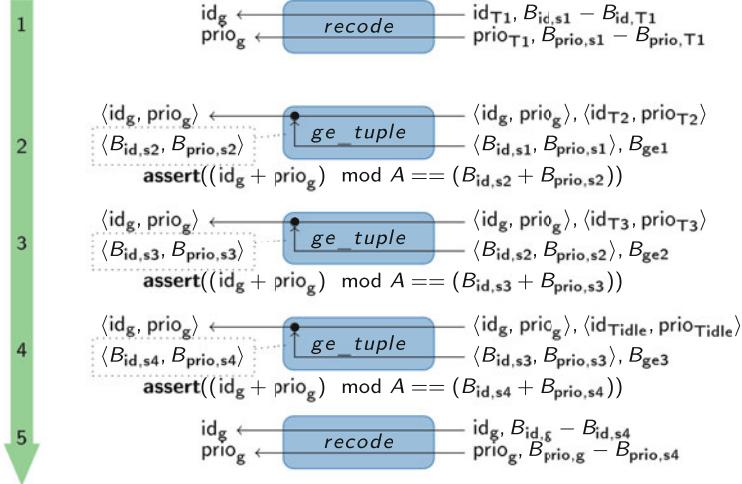


Fig. 3 General sequence of the encoded scheduling operation on the example of three tasks (T1, T2, T3). All operations on signatures B are calculated already at compile time

the tuples' priority value and update the global values, if necessary. The sequence consists of five steps, as shown in Fig. 3:

- (1) Initialize prio_g and id_g to the first task.
- (2–3) For all further tasks, compare the task's priority to prio_g : If greater or equal, update $\langle \text{id}_g, \text{prio}_g \rangle$.
- (4) Repeat the last step for the idle task.
- (5) Recode the results to their original signatures.

The idle task priority is constantly bound to an encoded zero that is representing a suspended state. Thus, if all previous tasks are suspended, the last comparison (in step 4) will choose the idle task halting the system until the next interrupt.

Aside from the actual compare-and-update operation on fully encoded values, the *ge_tuple* function additionally integrates control-flow error detection. For each step, all signatures of the input operands ($B_{\text{id},s1..s4}$, $B_{\text{prio},s1..s4}$) and the signature of the operation itself ($B_{\text{ge}1..4}$) are merged into the resulting encoded values of the global tuple. Each corresponding signature of a step is then applied in the next operation accordingly. Thus, the dynamic values of the result tuple accumulate the signatures of all preceding operations. As the combination of these compile-time constant signatures is known before runtime, interspersed assertions can validate the correctness of each step. Even after the final signature recode operation (step 5), any control-flow error is still detectable by the dynamic signature. Thus, the correctness of the encoded global tuple can be validated at any point in time. In effect, fault detection is ensured, as all operations are performed on encoded values.

The remaining dynamic state highly depends on the underlying architecture. Regarding the currently implemented IA-32 variant, we were able to reduce this

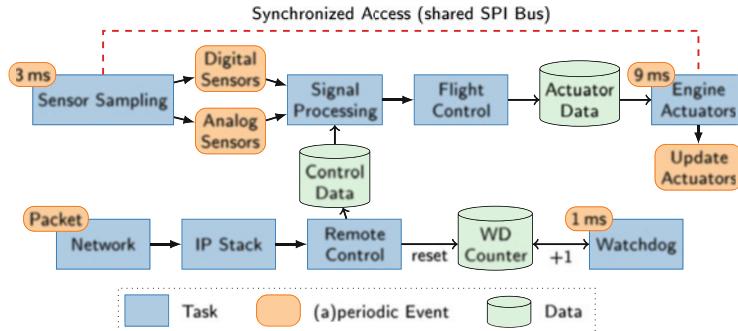


Fig. 4 Simplified representation of the *I4Copter* task and resource constellation used as evaluation scenario

runtime state to an array storing the stack pointers of preempted tasks, and an corresponding index variable, as shown in Fig. 2. The variables are used within each interrupt entry as well as during the actual dispatch operation. As they are not involved in any arithmetic calculations, but only read and written, we can avoid the overhead of the ANB-encoding in these cases and protect them by DMR or parity checks, respectively.

3.3 Evaluation

For comparison, we chose ERIKA Enterprise [21], an industry-grade (i.e., formally certified) open-source implementation of the automotive OSEK standard [40].

The evaluation is based on a realistic system workload scenario considering all essential RTOS services, resembling a real-world safety-critical embedded system in terms of a quadrotor helicopter control application (cf. Fig. 4). The scenario consists of 11 tasks, which are activated either periodically or sporadically by one of four interrupts. Inter-task synchronization is done with OSEK resources and a watchdog task, observing the remote control communication. We evaluated several variants of ERIKA and *dOSEK*, all running the same task set. As ERIKA does not provide support for hardware-based memory protection, we also disabled the MPU in *dOSEK*:

ERIKA Standard version of ERIKA with enabled sanity checks (SVN r3274).

dOSEK (unprotected) For the *dOSEK* base version only the indirection avoidance and the generative approach are used against SDCs.

dOSEK (FT) The safeguarded kernel execution with encoded operations.

dOSEK (FT+ASS) Like FT, but with additional assertions obtained by a flow-sensitive global control-flow analysis [18].

The application flow is augmented with 172 checkpoints. Every RTOS under test executes the application for three hyper periods, while, at the same time a trace of visited checkpoints is recorded. It is the mission of the systems under test to reproduce this sequence, without corrupting the application state. If the sequence *silently* diverges in the presence of faults, we record a silent data corruption.² The application state (task stacks) is checked for integrity at each checkpoint. To evaluate the fault containment within the kernel execution, we further recorded an SDC in case of violated integrity. Both SDC detection mechanisms were realized externally by the FAIL* fault-injection framework [47] without influencing the runtime behavior of the systems under test. Since FAIL* has the most mature support for IA-32, we choose this architecture as our evaluation platform. FAIL* provides elaborate fault-space pruning techniques that allow to cover the *entire* space of effective faults, while keeping the total number of experiments manageable. The evaluated fault space includes all *single-bit faults* in the *main memory*, in the *general-purpose registers*, the *stack pointer*, and *flags registers*, as well as the *instruction pointer*.

3.3.1 Fault-Injection Results

All OS variants differ in code size, runtime, and memory consumption—parameters that directly influence the number of effective injected faults. To directly compare the robustness independent of any other non-functional properties, we concentrate on the resulting absolute SDC count, which represents the number of cases in which the RTOS did not provide the expected behavior. Figure 5 shows, on a logarithmic scale, the resulting SDC counts.

The results show that, compared to ERIKA, the *unprotected dOSEK* variant already faces significantly fewer control-flow and register errors. This is caused by the means of constructive fault avoidance, particularly the avoidance of indirections in the generated code. The activation of fault tolerance measures (*dOSEK FT*) significantly reduces the number of memory errors, which in total reduces the SDC count compared to ERIKA by **four orders of magnitude**. The remaining SDCs can further be halved by adding static assertions (*dOSEK FT+ASS*).

3.3.2 Memory- and Runtime Costs

On the downside, aggressive inlining to avoid indirections, but especially the encoded scheduler and kernel execution path leads to additional runtime and memory costs, which are summarized in Table 1. Compared again to ERIKA, the SDC reduction by four orders of magnitude is paid for with a 4× increase in runtime and a 20× increase in code size. As most of the code bloat is caused by the inlining

²Faults that lead to a hardware trap are *not* counted as silent, as they are handled by the kernel.

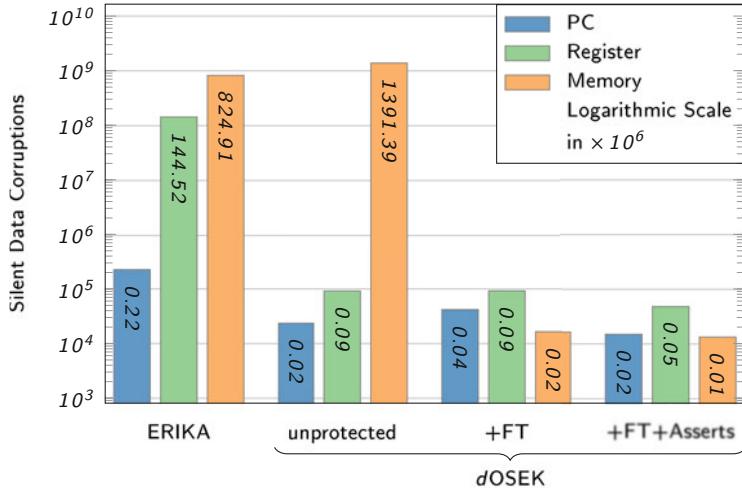


Fig. 5 SDC distribution for the evaluated variants of the *I4Copter* scenario (Fig. 4 on a logarithmic scale; pruned experiments are factored in). The encoded *dOSEK* system achieves an improvement in the SDC count by four orders of magnitude compared to ERIKA (base)

Table 1 Memory- and runtime cost

System	Code size (bytes)	Runtime (instructions)
ERIKA	3782	38,912
<i>dOSEK</i> (unprotected)	14,985	29,223
<i>dOSEK</i> FT	53,956	110,524
<i>dOSEK</i> FT+ASS	71,049	121,583
<i>dOSEK</i> FT+ASS+OPT	24,955	90,106

of the encoded scheduler at each call site, we have added a fifth variant (*dOSEK FT+ASS+OPT*) that employs further whole-program static optimizations to exclude unnecessary scheduler invocations (see [17] for further details). This version is still $10^4 \times$ less vulnerable to SDCs, but reduces the runtime overhead to $2.5 \times$ and the code overhead to $8 \times$.

4 Modularizing Software-Based Memory Error Detection and Correction

The *dOSEK* approach in the previous section showed the general reliability limits when designing a static OS from scratch, focusing on reliability as a first-class design goal. However, a different and quite common use case is that the requirements entail using a preexisting COTS embedded OS, which is often *dynamic* in the sense that it provides an interface for creating and destroying threads or memory

allocations at runtime. To protect this class of system-software stacks against transient hardware faults—e.g., bit flips—in memory, we propose a software-based memory-error recovery approach that exploits application knowledge about memory accesses, which are analyzed at compile time and hardened by compiler-generated runtime checks.

A central challenge is the *placement* of these runtime checks in the control flow of the software, necessitating an analysis that determines which program instructions access which parts of the memory. In general, this is an undecidable problem for pointer-based programming languages; however, if we assume an *object-oriented programming model*, we can reason that non-public data-structure members are accessed only within member functions of the same class. Consequently, data structures—or, *objects*—can be examined for errors by inserting a runtime check *before* each member-function call.

In this section, we describe our experiences with devising such an object-level error recovery in AspectC++ [51]—an AOP extension to C++—and applying it to the embedded Configurable operating system (eCos) [37]. Our software-based approach, called Generic Object Protection (GOP), offers the flexibility to choose from an extensible toolbox of error-detecting and error-correcting codes, for example, CRC and Hamming codes.

4.1 Generic Object Protection with AspectC++

Our experience with the embedded operating system eCos shows that OS kernel data structures are highly susceptible to soft errors in main memory [8]. Several kernel data structures, such as the process scheduler, persist during the whole OS uptime, which increases the chance of being hit by a random soft error.

As a countermeasure, OS kernel data structures can contain redundancy, for example, a separated Hamming code [48]. Before an instance of such a data structure—an *object* in object-oriented jargon—is used, the object can be examined for errors. Then, after object usage, the Hamming code can be updated to reflect modifications of the object.

Manually implementing such a protection scheme in an object-oriented programming language is a tedious and error-prone task, because *every* program statement that operates on such an object needs careful manipulation. Therefore, we propose to integrate object checking into existing source code by AOP [32]. Over the last 19 years, we have developed the general-purpose *AspectC++* programming language and compiler [51] that extends C++ by AOP features. A result of the SPP-1500’s *DanceOS* project is *AspectC++ 2.0*, which provides new language features that allow for a completely modular implementation of the sketched object protection scheme—the GOP. In the following, we describe these programming-language features taking the example of GOP.

```

1 aspect GenericObjectProtection {
2     pointcut critical() = "Cyg_Scheduler" || "Cyg_Thread"; // list of types
3     advice critical() : slice class { // generic class extension
4         HammingCode<JoinPoint> code; // template meta-program
5     };
6
7     advice construction(critical()) : after() {
8         tjp->target()->code.update(); // generic advice
9     }
10
11    pointcut trigger_check() = call(member(critical())) ||
12        get(member(critical())) || set(member(critical()));
13    pointcut trigger_update() = call(member(critical())) ||
14        set(member(critical()));
15
16    advice trigger_check() : before() {
17        if (tjp->that() != static_cast<void*>(tjp->target())) {
18            tjp->target()->code.check(); // check callee
19        }
20    advice trigger_update() : after() {
21        if (tjp->that() != static_cast<void*>(tjp->target())) {
22            tjp->target()->code.update(); // update callee
23        }
24
25    pointcut any_call() = call("% ...::%(...)");
26    advice any_call() && within(member(critical())) : before() {
27        if (tjp->that() != static_cast<void*>(tjp->target())) {
28            tjp->that()->code.update(); // update caller
29        }
30    advice any_call() && within(member(critical())) : after() {
31        if (tjp->that() != static_cast<void*>(tjp->target())) {
32            tjp->that()->code.check(); // check caller
33        }
34    };
}

```

Fig. 6 A simplified implementation of the GOP mechanism written in AspectC++

4.1.1 Generic Introductions by Compile-Time Introspection

Figure 6 shows the source code for a highly simplified implementation of the GOP. The keyword `aspect` in the first line declares an entity similar to a C++ class that additionally encompasses `pointcut` expressions and pieces of advice. A `pointcut` expression is a reusable alias for names defined in the program. For example, the `pointcut critical()` in line 2 lists two classes, namely “`Cyg_Scheduler`” and “`Cyg_Thread`”, from the eCos kernel. This `pointcut` is used by the following line that defines `advice` that those two classes get extended by a `slice` introduction, which inserts an additional member into these classes. The inserted member “`code`” is an instance of the template class `HammingCode<typename>`, whose template argument is bound to the built-in type `JoinPoint`. This type is only available in the body of advice code and offers an interface to a compile-time introspection API.

AspectC++’s introspection API [7] provides the programmer with information on the class type that is being extended by the `slice` introduction. We use this information within the template class `HammingCode` to instantiate a generative

C++ template metaprogram [14] that compiles to a tailored Hamming code for each class. In particular, we use the number of existing data members (`MEMBERS`) *prior* to the slice introduction, their types (`Member<I>::Type`) to obtain the size of each member, and a typed pointer (`Member<I>::pointer (T *obj)`) to each data member to compute the actual Hamming code. Furthermore, for classes with inheritance relationships, we recursively iterate over all base classes that are exposed by the introspection API. To simplify the iteration over this API, we implemented a Join-Point Template Library (JPTL) that offers compile-time iterators for each API entry.

4.1.2 Advice for Control Flow and Data Access

Once the Hamming code is introduced into the classes, we need to make sure that the code is checked and updated when such an object is used. At first, the Hamming code needs to be computed *whenever an object of a protected class is instantiated*. The advice for construction in line 7 implements this requirement: after a constructor execution, the `update()` function is invoked on the “code” data member. The built-in pointer `tjp->target()` yields the particular object being constructed (`tjp` is an abbreviation for `this join point`).

The lines 11–14 define further pointcuts that describe situations where the objects are used. The pointcut function `member(...)` translates the existing pointcut `critical()` into a set of all data members and member functions belonging to classes matched by `critical()`. Thus, `call(member(critical()))` describes all procedure calls to member functions of the particular classes. Likewise, the pointcut function `get(...)` refers to all program statements that read a member variable, and the other way around, `set(...)` matches all events in the program that write to a particular member variable. The `get/set` pointcut functions are new features of the AspectC++ language that notably allow observing access to data members declared as `public`.

The advice in line 16 invokes the `check()` routine on the Hamming-code sub-object based on the `trigger_check()` pointcut, that is, *whenever a member function is called, or a member variable is read or written*. Similarly, the advice in line 20 invokes the `update()` function after member-function calls or writing to a member variable. Both pieces of advice invoke these routines only if the caller object (`tjp->that()`) and the callee object (`tjp->target()`) are not identical. This is an optimization that avoids unnecessary checking when an already verified object invokes a function on itself.

A call to *any* function is matched by the wild-card expression in line 25. Therewith, the advice definition in line 26 updates the Hamming code *whenever a function call leaves a critical object*, as specified by `within(member(critical()))`, and when the caller object is not identical to the callee object. When the function returns, the Hamming code gets checked by the advice in line 30.

By defining such generic pieces of advice, AspectC++ enables a modular implementation of the GOP mechanism, completely separated from the remaining

source code. More advice definitions exist in the complete GOP implementation, for instance, covering `static` data members, non-blocking synchronization, or virtual-function pointers [8].

4.2 Implementation and Evaluation

In the following, we describe the implementation of five concrete EDMs/ERMs based on the GOP mechanism. Subsequently, we demonstrate their configurability on a set of benchmark programs bundled with eCos. We show that the mechanisms can easily be adapted to protect a specific subset of the eCos-kernel data structures, e.g., only the most critical ones. After applying a heuristic that benchmark-specifically chooses this data-structure subset, and protecting the corresponding classes, we present fault injection (FI) experiment results that compare the five EDMs/ERMs. Additionally, we measure their static and dynamic overhead, and draw conclusions on the overall methodology.

4.2.1 EDM/ERM Variants

We implemented the five EDMs and ERMs listed in Table 2 to exemplarily evaluate the GOP mechanism. For instance, a template metaprogram generates an optimal Hamming code tailored for each data structure and we applied a bit-slicing technique [48] to process 32 bits in parallel. Thereby, the Hamming-code implementation can correct multi-bit errors, in particular, all burst errors up to the length of a machine word (32 bits in our case). Besides burst errors, the CRC variants (see Table 2) cover all possible 2-bit and 3-bit errors in objects smaller than 256 MiB by the CRC-32/4 code [11]. Each EDM/ERM variant is implemented as a generic module and can be configured to protect any subset of the existing C++ classes of the target system.

In the following subsections, we refer to the acronyms introduced in Table 2, and term the unprotected version of each benchmark the “Baseline.”

Table 2 EDM/ERM variants, and their effective line counts (determined by `cloc`)

Variant	Description (mechanisms applied <i>on data member granularity</i>)	LOC
CRC	CRC-32, using SSE 4.2 instructions (EDM)	163
TMR	Triple modular redundancy: two copies + majority voting (EDM/ERM)	124
CRC+DMR	CRC (EDM) + one copy for error correction (ERM)	210
SUM+DMR	32-Bit two’s complement addition checksum (EDM) + one copy (ERM)	198
Hamming	SW-implemented Hamming code (EDM/ERM), processing 32 bits in parallel	355
Framework	GOP infrastructure, basis for all concrete EDM/ERM implementations	2371

4.2.2 Evaluation Setup

We evaluate the five EDM/ERM variants on eCos 3.0 with a subset of the benchmark and test programs that are bundled with eCos itself, namely those 19 implemented in C++ and using threads (omitting CLOCK1 and CLOCKTRUTH due to their extremely long runtime). More details on the benchmarks can be found in previous work [8]. Because eCos currently does not support x64, all benchmarks are compiled for i386 with the GNU C++ compiler (GCC Debian 4.7.2–5), and eCos is set up with its default configuration.

Using the FAIL* FI framework [47], we simulate a fault model of uniformly distributed transient single-bit flips in data memory, i.e., we consider *all* program runs in which one bit in the data/BSS segments flips at some point in time. Bochs, the IA-32 (x86) emulator back end that FAIL* currently provides, is configured to simulate a modern 2.666 GHz x86 CPU. It simulates the CPU on a behavior level with a simplistic timing model of one instruction per cycle, also lacking a CPU cache hierarchy. Therefore the results obtained from injecting memory errors in this simulator are pessimistic, as we expect a contemporary cache hierarchy would mask some main-memory bit flips.

4.2.3 Optimizing the Generic Object Protection

As described in Sect. 4.1.1, the generic object-protection mechanisms from Table 2 can be configured by specifying the classes to be protected in a pointcut expression. Either a wild-card expression selects all classes automatically, or the pointcut expression lists a subset of classes by name. In the following, we explore the trade-off between the subset of selected classes and the runtime overhead caused by the EDM/ERMs.

We cannot evaluate all possible configurations, since there are exponentially many subsets of eCos-kernel classes—the power set. Instead, we compile each benchmark in *all* configurations that select only a single eCos-kernel class for hardening. For these sets that contain exactly one class each, we measure their simulated runtime, and subsequently order the classes from the least to most runtime overhead individually for each benchmark. This order allows us to *cumulatively* select these classes in the next step: We compile each benchmark again with increasingly more classes being protected (from one to all classes, ordered by runtime). Observing the cumulative runtimes of the respective class selections [8], the benchmarks can be divided into two categories, based on their absolute runtime:

- 1. Long runtime (more than ten million cycles):** For any subset of selected classes, the runtime overhead stays negligible. The reason is that the long-running benchmarks spend a significant amount of time in calculations on the application level or contain idle phases.

2. **Short runtime (less than ten million cycles):** The EDM/ERM runtime overhead notably increases with each additional class included in the selections. These benchmarks mainly execute kernel code.

After conducting extensive FI experiments on each of the cumulatively protected programs, it turns out that for our set of benchmarks, the following heuristic yields a good trade-off between runtime and fault tolerance: *We only select a particular class if its protection incurs less than 1 percent runtime overhead.* Using this rule of thumb can massively reduce the efforts spent on choosing a good configuration, as the runtime overhead is easily measurable without running any costly FI experiments. However, in 6 of the initial 19 benchmarks, there are *no* classes that can be protected with less than 1% overhead. Those programs are most resilient without GOP (see Sect. 4.3 for further discussion).

4.2.4 Protection Effectiveness and Overhead

Using this optimization heuristic, we evaluate the EDM/ERM mechanisms described in Table 2. Omitting the aforementioned six benchmarks that our heuristic deems not protectable, Fig. 7 shows FI results from an FI campaign entailing 46 million single experiment runs, using the extrapolated absolute failure count (EAFC) as a comparison metric that is proportional to the unconditional failure probability [46]. The results indicate that the five EDM/ERMs mechanisms are similarly effective in reducing the EAFC, and reduce the failure probability by up to 79% (MBOX1 and THREAD1, protected with CRC) compared to the baseline. The total number of system failures—compared to the baseline without GOP—is reduced by 69.14% (CRC error detection), and, for example, by 68.75% (CRC+DMR error correction). Note that some benchmarks (e.g., EXCEPT1 or MQUEUE1) show very little improvement; we will discuss this phenomenon in Sect. 4.3.

Of course, the increase in system resiliency comes at different static and dynamic costs. With the GOP in place, the static binary sizes (Fig. 8) can grow quite significantly by on average 57% (CRC) to 120% (TMR) (up to 229% in the case of TMR and the KILL benchmark)—showing increases in the same order of magnitude as those observed in the dOSEK evaluation (Sect. 3.3.2). Looking closer, the DATA sections of all baseline binaries are negligibly tiny (around 450 bytes) and increase by 5% up to 79%. The BSS sections are significantly larger (in the tens of kilobytes), and vary more between the different benchmarks. They grow more moderately by below 1% up to 15%. In contrast, the code size (TEXT) is even larger in the baseline (23–145 kiB), and the increases vary extremely between the different variants: While CRC increases the code by an average of 114%, CRC+DMR on average adds 204%, SUM+DMR 197%, Hamming 200%, and TMR is the most expensive at an average 241% code-size increase.

But although the static code increase may seem drastic in places, low amounts of code are actually executed at runtime, as we only protected classes that introduce

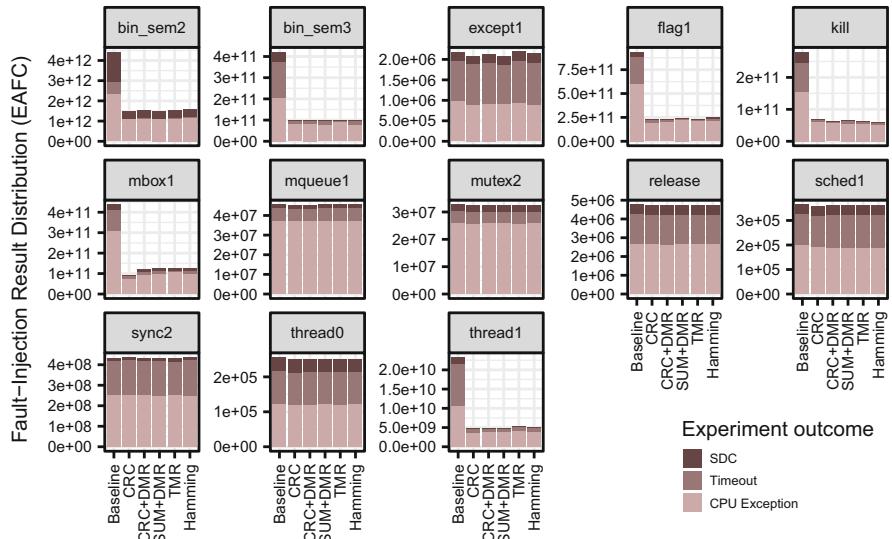


Fig. 7 Protection effectiveness for different EDM/ERM variants

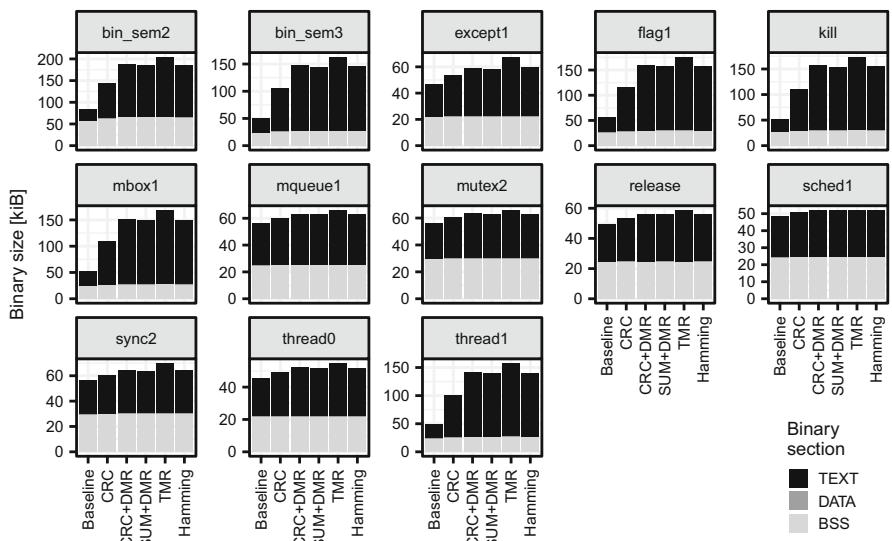


Fig. 8 Static code and data/BSS segment size of the EDM/ERM variants: the code (TEXT) segment grows due to additional CPU instructions, with CRC (detection only) being the most lightweight

less than 1% runtime overhead (see Sect. 4.2.3). Verifying the runtime on real hardware (an Intel Core i7-M620 CPU running at 2.66 GHz), we confirm that the real-world runtime overhead totals at only 0.36% for all variants except for TMR (0.37%). The results indicate that the GOP—when configured appropriately— involves negligible runtime overhead on real hardware.

4.3 Discussion

As software-implemented error detection and correction always introduces a runtime overhead, protected variants naturally run longer than their unprotected counterparts, increasing the chance of being hit by memory bit flips (assuming them to be uniformly distributed). Consequently, there exists a break-even point between, metaphorically, quickly crossing the battlefield without protection (and a high probability that a hit is fatal), and running slower but with heavy armor (and a good probability to survive a hit). The benchmarks in our initial analysis [8] we identified to be *not* effectively protectable with the GOP are on the unfavorable side of this break-even point: The additional attack surface from the runtime and memory overhead outweighs the gains from being protected for all configurations. Also, some benchmarks are just *barely* profiting from the GOP, such as, e.g., EXCEPT1 or MQUEUE1 (see Fig. 7).

A more detailed analysis of what distinguishes these benchmarks from the others reveals that they actually represent the *pathologic worst case* for GOP: Unlike “normal” applications that spend a significant amount of time in calculations on the application level, or waiting for input or events from the outside, this subset of benchmarks *only* executes eCos system calls. This reduces the time frame between an `update()` after the usage of a system object, and the `check()` at the begin of the next usage (cf. Sect. 4.1.2), to a few CPU cycles. The fault resilience gains are minimal, and the increased attack surface all in all increases the fault susceptibility significantly. Nevertheless, we do not believe the kernel-usage behavior of these benchmarks is representative for most real-world applications, and do not expect this issue to invalidate our claim that GOP is a viable solution for error detection and correction in long-living data structures.

For the remaining benchmarks, the analysis in Sect. 4.2.4 shows that the EDM/ERMs mainly differ in their static overhead. CRC is clearly the best choice when detection-only suffices. For error correction, the Hamming code turns out best. The high redundancy of the DMR variants and TMR are overkill—at least unless much more adverse fault models are considered.

5 Conserving Consistent State in Persistent Memory with Software Transactional Memory

Recent advances in persistent memory (PM) enable fast, byte-addressable main memory that maintains its state across power-cycling events. To survive power outages and prevent inconsistent application state, current approaches introduce persistent logs and require expensive cache flushes. In fact, these solutions can cause a performance penalty of up to $10\times$ for write operations on PM. With respect to wear-out effects, and a significantly lower write performance compared to read operations, we identify this as a major flaw that impacts performance and lifetime of PM. Being already persistent, data corruptions in PM cannot be resolved by simply restarting a system. Without countermeasures this limits the usability of PM and poses a high risk of a permanently inconsistent system state.

In this section, we present *DNV Memory*, a library for PM management. For securing allocated data against power outages, multi-bit faults that bypass hardware protection and even usage violations, *DNV Memory* introduces *reliable transactions*. Additionally, it reduces writes to PM by offloading logging operations to volatile memory, while maintaining *durability on demand* by an early detection of upcoming power failures. Our evaluation shows a median overhead of 6.5%, which is very low considering the ability to repair up to 7 random bit-errors per word. With durability on demand, the performance can be even improved by a factor of up to 3.5 compared to a state-of-the-art approach that enforces durability on each transaction commit.

5.1 System Model

We assume that hybrid system architectures equipped with both, volatile and persistent main memory, will become a commodity. This implicates that the execution state of processes will be composed of volatile and persistent parts.

While Phase Change Memory (PCM) is the most promising PM technology today, PM modules can also be built using resistive random-access memory (RRAM), spin-transfer-torque magnetoresistive random-access memory (STT-MRAM), or even battery-backed DRAM. Thereby, all processes in a system should be able to access PM directly through load and store operations in order to achieve optimal performance.

CPU caches can be used to further speed up access to persistent data. However, in order to survive power failures, cache lines containing data from PM must be flushed and the data must reach the *Durability Domain* of the PM module before the machine shuts down due to a power loss. This requires platform support in form of an asynchronous DRAM refresh (ADR) [49] or a *Flush Hint Address* [1]. Under these premises, we assume that word-level power failure atomicity is reached.

Depending on the used main-memory technology, various effects exist that may cause transient faults as previously outlined. Additionally, PCM and RRAM have

a limited write endurance that lies in the range of 10^6 up to 10^{10} operations [31]. Once worn out, the cell's value can only be read but not modified anymore.

We assume that all static random-access memory (SRAM) cells inside the CPU are guarded by hardware fault tolerance and are sufficiently reliable to ensure correct operation. Of course reliable DRAM supporting hardware error correction code (ECC) exists and PM can be protected by hardware solutions too. However, the common hardware ECC mechanisms only provide single-bit-error correction, double-bit-error detection (SECDED) capabilities, which is not always sufficient [52]. We assume that due to economic reasons not every PM module will support the highest possible dependability standard, leaving a fraction of errors undetected. Some PM modules may even lack any hardware protection. This paves the way for software-based dependability solutions.

5.2 Concepts of DNV Memory

The main goal of our design is to provide the familiar `malloc` interface to application developers for direct access to PM. At the same time, we want data stored in PM to be robust against power failures, transient faults, and usage errors.

Our core API functions (see Table 3(a) and (b)) resemble the interface of `malloc` and `free`. The only additional requirement for making legacy volatile structures persistent with *DNV Memory* is using our API functions and wrapping all persistent memory accesses in atomic blocks (see Table 3(e)).

These atomic blocks provide ACID³ guarantees for thread safety, and additionally preserve consistency in case of power failures. Furthermore, *DNV Memory* combines software transactional memory (STM) with the allocator to manage

Table 3 Overview of the *DNV Memory* application programming interface (API)

Category	Function	Description	Ref.
Core API	<code>void* dnv_malloc(size_t sz)</code>	Allocates persistent memory like <code>malloc(3)</code>	(a)
	<code>void dnv_free(void* ptr)</code>	Releases persistent memory like <code>free(3)</code>	(b)
Static Variables	<code>DNV POD variable</code>	Statically places <i>plain old data</i> in PM at definition	(c)
	<code>DNV_OBJ variable</code>	Statically places the object in PM at definition	(d)
Transactions	<code>__transaction_atomic{...}</code>	Atomic block with ACID guarantees and reliability	(e)

³Atomicity, consistency, isolation, durability.

software-based ECC. Every data word that is accessed during a transaction is validated and can be repaired if necessary.

In order to store entry points to persistent data structures that survive process restarts, *DNV Memory* provides the possibility to create static persistent variables (Table 3(c) and (d)). On top of this core functionality, *DNV Memory* introduces the concepts *durability on demand* and *reliable transactions* that are explained in the following.

If a power failure occurs during the update of persistent data structures, the *DNV Memory* might be in an inconsistent state after restart. To prevent this, *DNV Memory* follows the best practices from databases and other PM allocators [12, 54] and wraps operations on PM in atomic blocks. This can be achieved with STM provided by modern compilers or libraries like TinySTM [22]. The transactions must also be applied to the allocator itself, as its internal state must be stored in PM as well.

Different to previous works, *DNV Memory* aims at minimizing write accesses to PM. We store all transaction logs in volatile memory and utilize a power-failure detection to enforce *durability on demand*. When a power outage is imminent, the operating system copies the write-back logs back to PM in order to prevent state inconsistency. Therefore, every thread has to register its volatile memory range for the write-back log at our kernel module, which in turn reserves a PM range for a potential backup copy. After restart, the write-back logs are restored from PM, and every unfinished commit is repeated.

Since durability is actually required only in case of a power failure or process termination, memory fences and cache flushing can be performed on demand. This preserves persistent data inside the CPU cache and consequently reduces writes to PM. Additionally, since memory within a CPU is well protected by hardware, persistent data inside the cache is less susceptible to transient faults and can be accessed faster.

Enforcing durability on demand requires the ability to detect power failures in advance. For embedded devices, the power-outage detection is a part of the *brownout* detection and state of the art [43]. On servers and personal computers, power outages can be detected via the PWR_OK signal according to the ATX power supply unit (PSU) design guide [3]. Although the PWR_OK signal is required to announce a power outage at least 1 ms in advance, much better forecasts can be achieved in practice. For instance, some Intel machines provide a power-failure forecast of up to 33 ms [39]. An even better power-failure detection can be achieved by inspecting the input voltage of the PSU with a simple custom hardware [25]. With this approach, power failures can be detected more than 70 ms in advance, which leaves more than enough time to enforce durability and prevent further modification of persistent data.

Crashes that are not caused by power failures can be handled just like power failures if durability can be secured. For instance, our kernel module is aware of any process using PM that terminates and enforces durability in that case. Crashes in the operating-system kernel can be handled either as part of a kernel-panic procedure, or by utilizing a system like Otherworld [16].

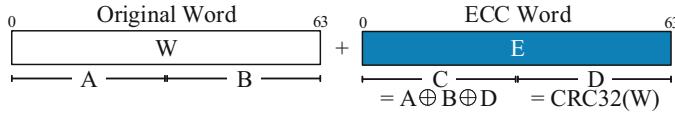


Fig. 9 DNV Memory ECC

In order to protect persistent data from corruption, *DNV Memory* reserves additional memory in each allocation that is meant to store ECC data. Afterwards fault tolerance is provided through *reliable transactions*.

As described in the previous section, all accesses to PM should be wrapped by atomic blocks in order to protect persistent data from power failures. These atomic blocks simply wrap all read and write operations in TM_LOAD and TM_STORE functions provided by the STM library, which in consequence control every word access. In combination with support from the memory allocator, this can be exploited to provide transparent fault tolerance.

Essentially, any ECC can be used to provide fault tolerance in software. For instance, we considered the SECDED Hamming code that is common in hardware protected memory. It protects 64-bit words with additional 8 bits, resulting in a 12.5% memory overhead. However, if implemented in software, the Hamming code would highly impact the performance of the application. Additionally, as already mentioned, we do not think that SECDED is enough to protect persistent data. Consequently, we decided to implement an ECC that provides a high multi-bit error correction with a memory overhead no more than dual modular redundancy. In addition, we want a fast error detection in software by exploiting commonly available hardware support. In general, whenever a *data word* W is written inside an atomic block, an *ECC word* E is created and stored in the additional space that the allocator has reserved. In theory, any fault-tolerant encoding is possible as long as error detection can be conducted in a few CPU cycles.

For *DNV Memory* we combine cyclic redundancy check (CRC) for fast error detection with an error location hint. Thus, we subdivide E into two halves C and D as shown in Fig. 9. The *error detection* half word D is generated with CRC32c ($D = \text{CRC32c}(W)$). We chose CRC as hardware support is available on many architectures, including most commodity CPUs. Additionally, with CRC32c—which is supported by SSE 4.2—a Hamming distance of 8 is achieved on a word length of 64 bits [33]. Without further assistance, error correction of up to 3 bits can be achieved by guessing the error location. However, by augmenting the CRC-based error detection with an *error location hint* C , less trials are needed and more bit-errors can be corrected. Inspired by RAID level 5 [42], we subdivide the data word W into two halves A and B and compute C according to Eq. (1).

$$C = A \oplus B \oplus D \quad (1)$$

The data validation takes place during a transaction whenever a word W is read for the first time. At that point, we recompute E' from W and compare its value with E . Normal execution can continue if both values match. Otherwise error correction is initiated.

Since errors can be randomly distributed across W and E , we start the error correction by narrowing the possible locations of errors. Therefore, we compute the *error vector* F via Eq. (2), which indicates the bit position of errors.

$$F = A \oplus B \oplus C \oplus D \quad (2)$$

This information is, however, imprecise, as it is unknown whether the corrupted bit is located in A , B , C , or D . Thus, for f errors detected by F , 4^f repair candidates R_i are possible, and are computed via Eq. (3). The *masking vectors* M_a , M_b , M_c , M_d are used to partition F between all four half words.

$$\begin{aligned} R_i &= W_i \parallel E_i \\ W_i &= A \oplus (F \wedge M_a) \parallel B \oplus (F \wedge M_b) \\ E_i &= C \oplus (F \wedge M_c) \parallel D \oplus (F \wedge M_d) \end{aligned} \quad (3)$$

To find the repair candidate R_s that contains the right solution, each R_i needs to be validated by recomputing E'_i from W_i and compare it to E_i . In order to repair all errors, exactly one R_s must be found with matching E'_i and E_i . For instance, if all errors are located in A , the repair candidate using $M_a = F$ and other masking vectors set to zero will be the correct result. Additionally, all combinations need to be considered that have an error at the same bit position in two or all half words, as these errors extinguish each other in C .

Please note that the set of repair candidates may yield more than one solution that can be successfully validated if more than three errors are present. To prevent a false recovery, all repair candidates must be validated for up to n errors. As an optimization step, we estimate n by counting the population in $E \oplus E'$ and limit the result to a maximum of $n = 7$.

To optimize the performance in a cache-aware way, we store the ECC words interleaved with the original words W as presented in Fig. 10. However, this interleaved data layout cannot be accessed correctly outside atomic blocks because the original layout is always expected here. Unfortunately, omitting atomic blocks around PM access is a very common mistake. We encountered such usage errors in every single STAMP benchmark [38], and whenever we ported or wrote persistent applications ourselves. Since the access to PM outside atomic blocks should be prevented to keep data consistent during power failures, we introduce the concept of a *transaction staging (TxStaging) section* as shown in Fig. 10. All memory that is allocated by *DNV Memory* has addresses belonging to the TxStaging section. The same applies to the location of persistent static variables. The TxStaging section is only a reserved virtual address space without any access rights. Consequently, any access to this segment will cause a segmentation fault that is easy to debug.

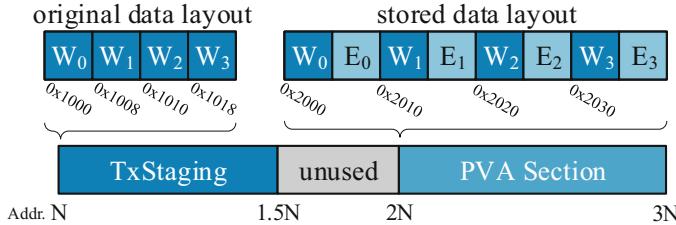


Fig. 10 *DNV Memory* persistent data layout and memory sections

However, inside an atomic block every access to the TxStaging section is intercepted by the STM library and redirected to the persistent virtual address (PVA) section where the actual persistent data is stored. To simplify the address transformation, the PVA section should be located at the address of the TxStaging section multiplied by 2. For instance, assuming the TxStaging section begins at address 0x1000 the PVA section should be placed at 0x2000. In that case a 32-byte object that is located in the address range from 0x1000 to 0x101f will be transformed into the address space 0x2000 to 0x203f as shown in Fig. 10.

5.3 Evaluation

We implemented *DNV Memory* on Linux in the form of a user-space library with a small companion kernel module and a hardware power-failure detector. Our design does not require any changes to the operating-system kernel or the machine itself. All components are pluggable and can be replaced by more extended solutions if needed. All user-space code is written in C++ and compiled with an unmodified GCC 5.4.0. A small linker-script extension provides additional sections like the TxStaging or the PVA section as shown in Fig. 10.

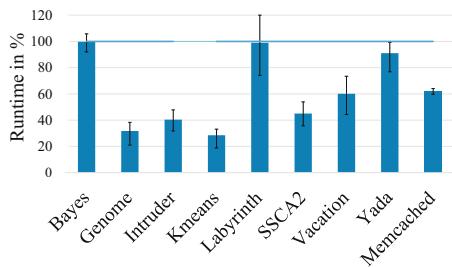
To show the feasibility of *durability on demand*, we artificially introduced power failures and measured the time between the detection of a power failure and the eventual machine shutdown. This period is referred as the shutdown forecast, and the results of 100 experiments are shown in Fig. 11. Additionally, the time of critical tasks in the event of a power failure is shown here. As can be seen, power failures can be detected sufficiently early to conduct all necessary durability measures. Counterintuitively, an idling CPU has a negative impact on the feasibility of the approach because the CPU enters the a power-saving mode with reduced performance. Additionally, less energy is stored within the power supply in the event of a power failure, thus leading to a quicker shutdown.

The performance impact of durability on demand was evaluated with applications from the STAMP benchmark suite [38] and the Memcached key-value store that was retrofitted with transactions. Figure 12 shows for each application the average relative runtime out of 100 measurements together with the 90% quantile that is

Fig. 11 Duration of critical tasks. A Heavy workload is achieved through kernel compilation

Measurement	Workload	Time in ms	
		min	max
Stop CPU and Flush Cache	Heavy	2.3	3.3
	Idle	4.4	5.6
Store Write-Back Log	Heavy	3.8	4.8
	Idle	7.4	8.6
Shutdown Forecast	Heavy	34.6	39.4
	Idle	25.2	36.8

Fig. 12 Application runtime under *durability on demand* in comparison to *durability on commit* (100% baseline)



indicated by the error bars. As the 100% baseline we used the state of the art, which enforces durability on each transaction commit. The results highly correlate with the cache efficiency of the application. For instance, little to no performance impact was achieved for Bayes, Labyrinth, and Yada, which operate on large work sets and show large transactions. If the transactions become large, they do not fit well into the cache and therefore do not benefit from locality, which severely impacts performance. Enforcing durability in this case has a low impact because the overhead from memory barriers and cache flushing becomes negligible. The other benchmarks, however, have moderate to small work sets, therefore a significant performance increase of up to $3.5\times$ can be observed.

To investigate the error detecting and correcting capabilities of *DNV Memory*, we conducted one billion fault-injection experiments, for one to seven-bit errors each. Every fault-injection experiment used a random word and bit-error positions that were randomly distributed over the original data and its corresponding ECC word. Only in the case of 7-bit errors, a small fraction of 0.000012163% fault injections produced ambiguous repair solutions that prevented a correction. In all other cases, including all errors up to 6-bit, a detection and correction was always successful. As can be seen in Fig. 13 the repair time increases exponentially with the number of flipped bits. However, even for correcting seven-bit errors, the mean error-repair time is less than 1.4 ms, which is acceptable considering the low probability of errors. Without any error, the validation only takes 34 ns.

For the performance evaluation of reliable transactions we again used STAMP benchmark applications [38] and Memcached. The bars depicted in Fig. 14 show

Fig. 13 Time to repair bit errors with reliable transactions

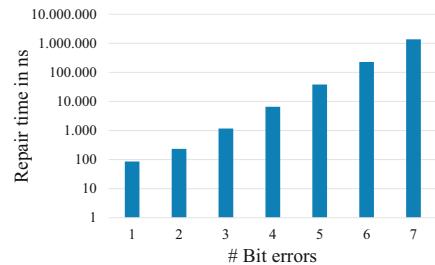
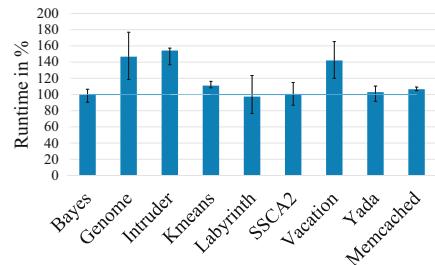


Fig. 14 Performance impact of reliable vs. traditional transactions (100% baseline)



the mean runtime of each benchmark. All values are relative to plain transactional execution (the 100% baseline), and the error bars represent the 95% and the 5% quantile. Over all applications, a median runtime of 106.5% is achieved with reliable transactions. Applications above this median have a workload that is dominated by reads or short transactions, hence the overhead of data verification has a higher impact here. Applications with a balanced or write-driven workload, however, have a higher runtime impact from transactions in general, thus the overhead that comes from reliable transactions is less prevalent. In summary, these results indicate a very acceptable performance impact—especially when considering the error-correcting capabilities of the approach.

5.4 Discussion

DNV Memory provides system support for dependable PM. Unlike previous approaches, *DNV Memory* enforces *durability on demand*, which in turn reduces write operations on PM and therefore improves reliability, lifetime, and performance. For tolerating power failures, *DNV Memory* uses software transactions that also include and secure the allocator itself. Our system even goes one step further and provides fault tolerance via software transactional memory. As our evaluation showed, *DNV Memory* protects data at word granularity, with an ECC word that is capable of detecting and correcting a random distributed seven-bit error, which is by far more than common hardware protection offered by server-

class volatile main memory. We also demonstrated that power failures can be detected early, allowing to conduct all necessary cleanup operations.

6 Summary

The work presented in this chapter has gained high visibility in the international research community. It was on the programme of all major conferences in the field and the authors received a number of best paper, best poster, and best dissertation awards, culminating in the renowned Carter Award for Christoph Borchert.

A reason for this success might be the focus on design principles and methods for hardening the operating system—and only the operating system. Most of previous research did not consider the specific properties of this special execution environment, such as different kinds of concurrent control flows, or assumed the reliable availability of underlying system services.

In our work we made a huge effort to design and implement an embedded operating system from scratch with the goal to explore the limits of software-implemented hardware fault tolerance in a reliability-oriented static system design. As a result we were able to reduce the SDC probability by orders of magnitude and found the remaining spots where software is unable to deal with hardware faults.

For existing embedded operating systems we have developed and evaluated Generic Object Protection by means of “dependability aspects,” which can harden operating systems at low cost without having to change the source code, and also addressed faults that crash the whole system by means of reliable transactions on persistent memory.

Finally, the authors have developed a fault-injection framework for their evaluation purposes that implements novel methods, which also advanced the state of the art in this domain.

Acknowledgments This work was supported by the German Research Foundation (DFG) under priority program SPP-1500 grants no. KA 3171/2-3, LO 1719/1-3, and SP 968/5-3.

References

1. Advanced Configuration and Power Interface Specification (Version 6.1) (2016). http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf
2. Alexandersson, R., Karlsson, J.: Fault injection-based assessment of aspect-oriented implementation of fault tolerance. In: Proceedings of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN ’11), pp. 303–314. IEEE Press, Piscataway (2011). <https://doi.org/10.1109/DSN.2011.5958244>
3. ATX12V Power Supply Design Guide (2005). http://formfactors.org/developer%5Cspecs%5CATX12V_PSDG_2_2_public_br2.pdf

4. Aussagues, C., Chabrol, D., David, V., Roux, D., Willey, N., Tornadre, A., Graniou, M.: PharOS, a multicore OS ready for safety-related automotive systems: results and future prospects. In: Proceedings of the 4th International Conference on Embedded Real Time Software and Systems (ERTS2 '10) (2010)
5. AUTOSAR: Specification of operating system (version 5.1.0). Tech. rep., Automotive Open System Architecture GbR (2013)
6. Bhandari, K., Chakrabarti, D.R., Boehm, H.J.: Makalu: fast recoverable allocation of non-volatile memory. In: Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) (2016)
7. Borchert, C., Spinczyk, O.: Hardening an L4 microkernel against soft errors by aspect-oriented programming and whole-program analysis. In: Proceedings of the 8th Workshop on Programming Languages and Operating Systems (PLOS '15), pp. 1–7. ACM Press, New York (2015). <https://doi.org/10.1145/2818302.2818304>
8. Borchert, C., Schirmeier, H., Spinczyk, O.: Generic soft-error detection and correction for concurrent data structures. IEEE Trans. Dependable Secure Comput. **14**(1), 22–36 (2017). <https://doi.org/10.1109/TDSC.2015.2427832>
9. Borkar, S.Y.: Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. IEEE Micro **25**(6), 10–16 (2005). <https://doi.org/10.1109/MM.2005.110>
10. Cassens, B., Martens, A., Kapitza, R.: The neverending runtime: using new technologies for ultra-low power applications with an unlimited runtime. In: International Conference on Embedded Wireless Systems and Networks, NextMote Workshop (EWSN 2016) (2016)
11. Castagnoli, G., Brauer, S., Herrmann, M.: Optimization of cyclic redundancy-check codes with 24 and 32 parity bits. IEEE Trans. Commun. **41**(6), 883–892 (1993). <https://doi.org/10.1109/26.231911>
12. Coburn, J., Caulfield, A.M., Akel, A., Grupp, L.M., Gupta, R.K., Jhala, R., Swanson, S.: NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In: SIGARCH Computer Architecture News (2011). <https://doi.org/10.1145/1961295.1950380>
13. Condit, J., Nightingale, E.B., Frost, C., Ipek, E., Lee, B., Burger, D., Coetze, D.: Better I/O through byte-addressable, persistent memory. In: Proceedings of the Symposium on Operating Systems Principles (2009)
14. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools and Applications. Addison-Wesley, Boston (2000)
15. Danner, D., Müller, R., Schröder-Preikschat, W., Hofer, W., Lohmann, D.: Safer Sloth: efficient, hardware-tailored memory protection. In: Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '14), pp. 37–47. IEEE Press, Piscataway (2014)
16. Depoutovitch, A., Stumm, M.: “Otherworld” - giving applications a chance to survive OS kernel crashes. In: Proceedings of the European Conference on Computer Systems (EuroSys) (2010)
17. Dietrich, C., Hoffmann, M., Lohmann, D.: Cross-kernel control-flow-graph analysis for event-driven real-time systems. In: Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '15). ACM Press, New York (2015). <https://doi.org/10.1145/2670529.2754963>
18. Dietrich, C., Hoffmann, M., Lohmann, D.: Global optimization of fixed-priority real-time systems by RTOS-aware control-flow analysis. ACM Trans. Embed. Comput. Syst. **16**, 35:1–35:25 (2017). <https://doi.org/10.1145/2950053>
19. Döbel, B., Härtig, H.: Who watches the watchmen?—protecting operating system reliability mechanisms. In: International Workshop on Hot Topics in System Dependability (HotDep) (2012)
20. Dulloor, S.R., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R., Jackson, J.: System software for persistent memory. In: Proceedings of the 9th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '14) (2014)
21. ERIKA Enterprise. <https://erika.tuxfamily.org>. Accessed 29 Sept 2014

22. Felber, P., Fetzer, C., Riegel, T., Marlier, P.: Time-based software transactional memory. *IEEE Trans. Parallel Distrib. Syst.* **21** (2010). <https://doi.org/10.1109/TPDS.2010.49>
23. Forin, P.: Vital coded microprocessor principles and application for various transit systems. In: Proceedings of the IFAC IFIP/IFORS Symposium on Control, Computers, Communications in Transportation (CCCT '89), pp. 79–84 (1989)
24. Goloubeva, O., Rebaudengo, M., Reorda, M.S., Violante, M.: Software-Implemented Hardware Fault Tolerance. Springer, New York (2006). <https://doi.org/10.1007/0-387-32937-4>
25. Heiser, G., Le Sueur, E., Danis, A., Budzynowski, A., Salomie, T.l., Alonso, G.: RapiLog: reducing system complexity through verification. In: Proceedings of the 8th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '13) (2013). <https://doi.org/10.1145/2465351.2465383>
26. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M., Teich, J., Wehn, N., Wunderlich, H.J.: Design and architectures for dependable embedded systems. In: Dick, R.P., Madsen, J. (eds.) Proceedings of the 9th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11), pp. 69–78. ACM Press (2011). <https://doi.org/10.1145/2039370.2039384>
27. Hoffmann, M., Borchert, C., Dietrich, C., Schirmeier, H., Kapitza, R., Spinczyk, O., Lohmann, D.: Effectiveness of fault detection mechanisms in static and dynamic operating system designs. In: Proceedings of the 17th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '14), pp. 230–237. IEEE Press, Piscataway (2014). <https://doi.org/10.1109/ISORC.2014.26>
28. Hoffmann, M., Ulbrich, P., Dietrich, C., Schirmeier, H., Lohmann, D., Schröder-Preikschat, W.: A practitioner's guide to software-based soft-error mitigation using AN-codes. In: Proceedings of the 15th IEEE International Symposium on High Assurance Systems Engineering (HASE '14), pp. 33–40. IEEE Press, Miami (2014). <https://doi.org/10.1109/HASE.2014.14>
29. Hoffmann, M., Lukas, F., Dietrich, C., Lohmann, D.: dOSEK: the design and implementation of a dependability-oriented static embedded kernel. In: Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '15), pp. 259–270. IEEE Press, Piscataway (2015). <https://doi.org/10.1109/RTAS.2015.7108449>
30. IEC: IEC 61508 – functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, Geneva (1998)
31. Kannan, S., Gavrilovska, A., Schwan, K.: pVM: persistent virtual memory for efficient capacity scaling and object storage. In: Proceedings of the 11th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '16), pp. 13:1–13:16. ACM, New York (2016). <https://doi.org/10.1145/2901318.2901325>
32. Kiczales, G., Lampert, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97). Lecture Notes in Computer Science, vol. 1241, pp. 220–242. Springer, Berlin (1997). <https://doi.org/10.1007/BFb0053381>
33. Koopman, P.: 32-Bit cyclic redundancy codes for Internet applications. In: Proceedings of the 32nd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '02) (2002). <https://doi.org/10.1109/DSN.2002.1028931>
34. Li, Y., West, R., Missimer, E.: A virtualized separation kernel for mixed criticality systems. In: Proceedings of the 10th USENIX International Conference on Virtual Execution Environments (VEE '14), pp. 201–212. ACM Press, New York (2014). <https://doi.org/10.1145/2576195.2576206>
35. Mariani, R., Fuhrmann, P., Vittorelli, B.: Fault-robust microcontrollers for automotive applications. In: Proceedings of the 12th International On-Line Testing Symposium (IOLTS '06), 6 pp. IEEE Press, Piscataway (2006). <https://doi.org/10.1109/IOLTS.2006.38>
36. Martens, A., Scholz, R., Lindow, P., Lehnfeld, N., Kastner, M.A., Kapitza, R.: Dependable non-volatile memory. In: Proceedings of the 11th ACM International Systems and Storage

- Conference, SYSTOR '18, pp. 1–12. ACM Press, New York (2018). <https://doi.org/10.1145/3211890.3211898>
- 37. Massa, A.: Embedded Software Development with eCos. Prentice Hall, Upper Saddle River (2002)
 - 38. Minh, C.C., Chung, J., Kozyrakis, C., Olukotun, K.: Stamp: Stanford transactional applications for multi-processing. In: Proceedings of the International Symposium on Workload Characterization (IISWC) (2008)
 - 39. Narayanan, D., Hodson, O.: Whole-System Persistence, pp. 401–410. ACM Press, New York (2012). <https://doi.org/10.1145/2189750.2151018>
 - 40. OSEK/VDX Group: Operating system specification 2.2.3. Tech. rep., OSEK/VDX Group (2005). <http://portal.osek-vdx.org/files/pdf/specs/os223.pdf>. Accessed 29 Sept 2014
 - 41. Pattabiraman, K., Grover, V., Zorn, B.G.: Samurai: protecting critical data in unsafe languages. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '08), pp. 219–232. ACM Press, New York (2008). <https://doi.org/10.1145/1352592.1352616>
 - 42. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (raid). SIGMOD Rec. **17**(3), 109–116 (1988). <https://doi.org/10.1145/971701.50214>
 - 43. Philips Semiconductors: AN468: Protecting Microcontrollers against Power Supply Imperfections (2001)
 - 44. Rebaudengo, M., Sonza Reorda, M., Violante, M., Torchiano, M.: A source-to-source compiler for generating dependable software. In: Proceedings of the 1st IEEE International Workshop on Source Code Analysis and Manipulation, pp. 33–42. IEEE Press (2001). <https://doi.org/10.1109/SCAM.2001.972664>
 - 45. Schiffl, U., Schmitt, A., Süßkraut, M., Fetzer, C.: ANB- and ANBDmem-encoding: detecting hardware errors in software. In: Proceedings of the 29th International Conference on Computer Safety, Reliability and Security (SAFECOMP '10), pp. 169–182. Springer, Berlin (2010)
 - 46. Schirmeier, H., Borchert, C., Spinczyk, O.: Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors. In: Proceedings of the 45th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '15), pp. 319–330. IEEE Press, Piscataway (2015). <https://doi.org/10.1109/DSN.2015.44>
 - 47. Schirmeier, H., Hoffmann, M., Dietrich, C., Lenz, M., Lohmann, D., Spinczyk, O.: FAIL*: an open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance. In: Proceedings of the 11th European Dependable Computing Conference (EDCC '15), pp. 245–255. IEEE Press, Piscataway (2015). <https://doi.org/10.1109/EDCC.2015.28>
 - 48. Shirvani, P.P., Saxena, N.R., Mccluskey, E.J.: Software-implemented EDAC protection against SEUs. IEEE Trans. Reliab. **49**(3), 273–284 (2000). <https://doi.org/10.1109/24.914544>
 - 49. SNIA NVDIMM Messaging and FAQ (2014)
 - 50. Song, J., Wittrock, J., Parmer, G.: Predictable, efficient system-level fault tolerance in C^3 . In: Proceedings of the 34th IEEE International Symposium on Real-Time Systems (RTSS '13), pp. 21–32. IEEE Press (2013). <https://doi.org/10.1109/RTSS.2013.11>
 - 51. Spinczyk, O., Lohmann, D.: The design and implementation of AspectC++. Knowl.-Based Syst. **20**(7), 636–651 (2007). Special Issue on Techniques to Produce Intelligent Secure Software. <https://doi.org/10.1016/j.knosys.2007.05.004>
 - 52. Sridharan, V., DeBardeleben, N., Blanchard, S., Ferreira, K.B., Stearley, J., Shalf, J., Gurumurthi, S.: Memory errors in modern systems: the good, the bad, and the ugly. In: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15), pp. 297–310. ACM Press, New York (2015). <https://doi.org/10.1145/2694344.2694348>
 - 53. Ulbrich, P., Hoffmann, M., Kapitza, R., Lohmann, D., Schröder-Preikschat, W., Schmid, R.: Eliminating single points of failure in software-based redundancy. In: Proceedings of the 9th European Dependable Computing Conference (EDCC '12), pp. 49–60. IEEE Press, Piscataway (2012). <https://doi.org/10.1109/EDCC.2012.21>

54. Volos, H., Tack, A.J., Swift, M.M.: Mnemosyne: lightweight persistent memory. In: SIGARCH Computer Architecture News, vol. 39, pp. 91–104. ACM Press, New York (2011). <https://doi.org/10.1145/1961295.1950379>
55. Yeh, Y.C.: Triple-triple redundant 777 primary flight computer. In: Proceedings of the IEEE Aerospace Applications Conference, vol. 1, pp. 293–307 (1996). <https://doi.org/10.1109/AERO.1996.495891>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part II

Cross-Layer Dependability: From Architecture to Software and Operating System

Michael Engel

Designers of modern embedded systems have to cope with errors in all kinds of system components, such as processing elements, memories, I/O devices, and interconnects. The ever increasing pressure to reduce the size, cost, and energy consumption of a given system has two effects that tend to amplify each other. On the one hand, solutions that are able to mitigate errors on the hardware side are often considered too expensive in terms of product cost or energy consumption and, thus, are frequently left out of the design for not strictly safety-critical systems. On the other hand, the ongoing miniaturization of semiconductor feature sizes and the reduction of supply voltages results in hardware that is increasingly more sensitive to external effects, such as cosmic radiation, thermal effects, or electromagnetic interference, that could cause errors.

As a consequence, those errors are much more likely to affect recent and future designs. The design constraints, thus, require new methods to detect and mitigate errors and allow designers to create more cost- and energy-efficient systems.

Due to the wide spectrum of possible error causes, approaches to mitigate these errors vary significantly. In this book section, a number of approaches that have been developed in the context of SPP 1500 as well as in projects of collaborating researchers are presented that cover a large part of the possible design space. The different projects discussed in the following chapters have one important common property—they are not restricted to work on a single layer of the hardware/software stack, but instead integrate information from different layers of the stack for increased efficiency.

The first chapter of this section, written by Kühn et al., analyzes the opportunities that massively parallel architectures offer in terms of providing a platform for the reliable execution of software. A large number of available processors enable the system designer to make use of selective redundancy for differing requirements

M. Engel (✉)

Norwegian University of Science and Technology, Trondheim, Norway

e-mail: michael.engel@ntnu.no

of the system's software components. The use of online health monitoring allows the system to detect typical reliability issues such as negative-bias temperature instability (NBTI) or hot carrier injection (HCI) and adapt to these by using online hardware reconfiguration before the issues result in an error visible on the software layers.

The second chapter by Kriebel et al. tries to tackle the reliability problem from a different direction. In their approach, the authors use error models and additional information from hardware as well as software layers in order to generate dependable software. By quantifying the error masking and propagation properties of a system, an analysis is performed that determines in which way an application's output will be affected by the assumed errors. An increase in dependability is then achieved by avoiding or mitigating the critical situations by means of software transformation or selective instruction protection.

In the third chapter by Kriebel et al., an approach to protect systems against transient errors using heterogeneous hardware/software hardening is proposed. Here, the authors analyze and exploit masking and error tolerance properties of different levels of the hard- and software stack. By using system components with different reliability properties from the architecture level to the design of caches, systems are enabled to adapt to error properties and reliability requirements of the executed software. The authors also give an outlook onto methods to complement the described heterogeneous hardware approach with compiler-based heterogeneous hardening modes on the software level.

The fourth chapter by van Santen et al. concentrates on reliability optimization for embedded multiprocessor systems on chip (MPSoCs). The problems analyzed are interdependencies of temperature and the reliable operation of MPSoCs. Here, the authors employ measured or estimated thermal values for different cores to determine which measures on system level can be applied to balance the thermal stress. This balancing, in turn, results in an evenly distributed probability of errors throughout the system. To enable the balancing, task migration between different cores based on virtualized interconnects is employed, which enables fast and transparent switch-over of communication channels.

Memory errors are in the focus of the fifth chapter, contributed by Alam and Gupta. The optimization of current memory chips to maximize their bit storage density makes them especially susceptible to soft errors. For cost and efficiency reasons, this process neglects to optimize for additional parameters such as manufacturing process variation, environmental operating conditions, and aging-induced wearout, leading to significant variability in the error susceptibility of memories. To improve memory reliability, the authors propose to replace traditional hardware-based bit-error checking and correction methods by software managed techniques and novel error correction codes to opportunistically cope with memory errors. These techniques can take the architectural or application context into consideration by leveraging semantics information to reduce the cost of error correction.

The final chapter in this section by Ma et al. concentrates on MPSoC systems again. Here, the focus lies on the contrasting requirements of soft-error reliability and lifetime reliability. The authors observe that most existing work on MPSoC fault tolerance only considers one of the described requirements, which in turn

might adversely impact the other. Accordingly, the possible tradeoffs between soft-error reliability and lifetime reliability are analyzed in order to achieve a high overall system reliability. Like some of the previously described approaches, the authors make efficient use of heterogeneous MPSoC architecture properties, such as big-little type same-ISA systems and systems that integrate traditional CPUs with GPGPUs.

Overall, the different approaches discussed in this section cover a large part of typical modern embedded architectures. Solutions for improved reliability of processors, memories, and interconnects are presented. A common theme for all these approaches is that each one operates on several different layers of the hardware/software stack in order to exploit this fused information to reduce the hardware and software overhead for error detection and mitigation.

While this common property is shown to be beneficial for embedded design tasks facing dependability problems, the particular projects show a large variety of detail in their approaches to achieve that goal. One common approach is to operate in a bottom-up way. These systems adapt hardware properties to mask problems for the software level. Other approaches make use of a top-down methodology. Here, software is adapted in order to handle possible errors showing up in the hardware. In general, however, most of the projects described above employ sort of a hybrid approach, in which information from different layers of the system is fused in order to enable optimization decisions at compile time and runtime.

An important additional research direction reflected in this section is based on the idea of accepting certain incorrect behaviors of a system in response to an error. Here, additional semantic information on the relevance of deviating system behavior is employed to determine the criticality of certain errors. In turn, accepting certain imprecisions in a system's results enables more efficient reliable embedded systems.

In general, we can conclude that all of the cross-layer techniques described above show significant improvements in the non-functional properties or design constraints a system designer has to consider when creating dependable embedded systems.

The large variety of analysis and mitigation efforts throughout all layers of the hardware and software stack show that dependability of systems, even if it is one of the earliest research topics in computer engineering, is still a highly relevant and active research topic. Novel challenges due to different hardware components and their properties increased demands on the computational power and energy efficiency as well as additional non-functional properties require innovative methods that combine work on all layers of the hardware and software stack.

However, we have to assess that the current solution landscape, of which we have tried to show a representative profile in this section, today still tends to produce isolated solutions which are not designed for interoperability. Here, an important future research challenge is an overarching effort that allows to flexibly integrate information from various different layers and, in turn, to enable multi-criterial optimizations at design and compile time as well as at runtime to enable general fault tolerant embedded systems. It will be interesting to observe how the different approaches described in this section will be able to contribute to this overall objective.

Increasing Reliability Using Adaptive Cross-Layer Techniques in DRPs: Just-Safe-Enough Responses to Reliability Threats



Johannes Maximilian Kühn, Oliver Bringmann, and Wolfgang Rosenstiel

1 Introduction

The broad deployment, as well as the increasingly difficult manufacturing of inspec semiconductors long make reliable operation and failures across the lifetime of an embedded system one of the industry's main concerns. Since ever-increasing demands do no longer allow us to resort to "robust" technologies, other means than semiconductor technology have to fill the gap left by cutting-edge technologies without resorting to unrealistic mainframe like protection mechanisms. As the operation scenarios become ever more challenging as well (edge computing, intelligent IoT nodes), hardware architects are faced with ever tighter power budgets for continuously increasing compute demands. We, therefore, proposed to exploit the architectural redundancies provided by potent, yet energy efficient massively parallel architectures, modeled using Dynamically Reconfigurable Processors (DRP). Using DRPs, we built an extensive cross-layer approach inspired by the overall project's approach as laid out in [1]. Following the idea of cross-layer reliability approaches, we built interfaces reaching from software layers right down to the transistor level mainly through computer architecture, allowing us to address both the varying reliability requirements and the significant computational demands of prospective workloads.

Figure 1 shows an overview of the layers this project targeted as described in the previous paragraph. While a strong focus has been on architecture, the project's aim was to use computer architecture to connect to the layers above and

J. M. Kühn (✉)
Preferred Networks Inc., Tokyo, Japan
e-mail: kuhn@preferred.jp

O. Bringmann · W. Rosenstiel
Eberhard Karls Universität Tübingen, Tübingen, Germany
e-mail: oliver.bringmann@uni-tuebingen.de

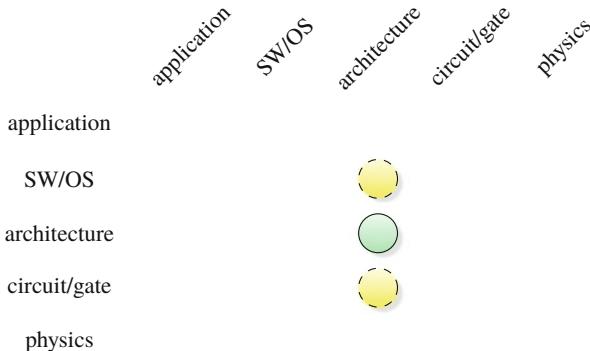


Fig. 1 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

below. We show how DRP architectures can leverage their inherent architectural redundancies to realize various degrees of reliable computing. On one end of the spectrum, we highlight how triple modular redundancy (TMR) and duplication with comparison (DWC) compute modes can be realized to actively secure computations without permanently binding hardware resources and with only slight hardware overheads. On the other end of the spectrum, we show how fault-free operation can be passively ascertained by periodically testing SoC components. Both, active and passive concepts together with the architectural redundancies allow for graceful degradation by pinpoint failure detection and subsequently dynamically remapping applications. Once established, both graceful degradation and low-cost TMR for critical parts of applications can be used to make specific operations in processor cores reliable by using the DRP or the demonstrated concepts as a reliable pipeline within a processor core.

A central point of the proposed methods is an overarching cross-layer approach [1], tying together these methods from the software layers (Application, Operating System) to all hardware layers below down to the semiconductor through the concepts introduced by our DRP architecture. To enable a reach down to the circuit level, we exemplarily used the extensive Body Biasing capabilities of Fully Depleted Silicon on Insulator (FDSOI) processes as a means for transistor-level testing and manipulation. This access down to the transistor level enables continuous monitoring of the precise hardware health and thereby not only reactive measures in case of hardware failure but also proactive measures to prevent system failure and prolong system lifetime if the hardware starts exhibiting signs of wear. Access to the device state also multiplies the reliability and system health options on the software layer. With previously having the choice of using TMR/DWC to minimize the error probability, we also show how DVFS with Body Biasing can offer both high power but highly reliable over spec versus ultra-low-power but risky computing modes. These modes’ long-term effects further multiply the set of operation modes, e.g., slowing down or speeding up degenerative effects such as Hot

Carrier Injection (HCI) or Negative Bias Temperature Instability (NBTI). However, with access to actual transistor parameters, the proposed approach also indicates that even permanent degeneration such as HCI can be temporarily overcome [2] to prolong system lifetime long enough to extend the graceful degradation period beyond conventional physical limits. Or to put it in the spirit of the parallel NSF effort [3], by opportunistically filling the technology gap using cross-layer methods, there are more means to approach and exploit the hardware's sheer physical limits.

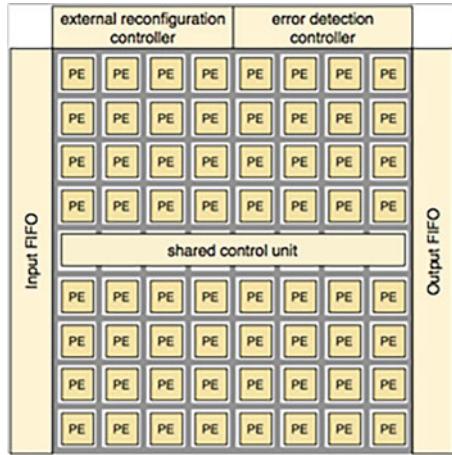
Within this project, we also faced the challenge of how such cross-layer approaches can be realistically validated and evaluated. While Software layers down to the RTL level allow, e.g., fault injection through instrumentation or emulation, the computational effort quickly becomes too large for realistically sized test samples. Furthermore, going below the gate level offers an entirely new set of challenges, both calling for appropriate solutions. For the layers from Software to RTL, we chose to implement the entire system as a prototype on an FPGA. For this FPGA, in turn, we developed a precise fault-injection mechanism so we could emulate the entire SoC with specific faults present. For the gate level and below, we devised a mix of SPICE simulations, and for body bias effect evaluation we ran in-silicon evaluations at the laboratory of Professor Amano at Keio University.

This chapter is structured as follows. Since reliability threats and how such threats surface has been covered in the general introduction, Dynamically Reconfigurable Processors are briefly introduced. The next section directly dives into how the inherent architectural redundancy can be put to use to increase the reliability of computations, as well as how to test these techniques. In the following two sections, the focus then shifts to both ends of the abstraction layers by focussing on how to infer the device state at the transistor level and potentially also recovering from a faulty state using body biasing together with how decisions on the software or operating system level affect the transistor level. The last technical section before wrapping up then brings all levels together by highlighting the interplay between each layer and the synergistic gain thereby achieved.

2 Dynamically Reconfigurable Processors

Dynamically reconfigurable architectures, or short DRP, are a sub-category of so-called coarse-grained reconfigurable architectures (CGRA). Similar to Field Programmable Gate Arrays (FPGA), CGRAs are reconfigurable architectures; however, in contrast to FPGAs, CGRAs are reconfigurable on a far coarser level. That is, while FPGAs can efficiently map per-bit configurability, CGRAs only allow reconfigurability on word-sized units. While this restriction makes CGRAs unfavorable for random bit logic, CGRAs possess a far greater area and energy efficiency as the logic overhead for reconfigurability per bit is far lower. DRPs add the concept of dynamic reconfiguration to CGRAs by having on-chip memories for multiple configurations, or contexts, as instructions are often called in DRPs. As the keyword instruction already hints, DRPs resemble much more simple processors

Fig. 2 Exemplary DRP instance with additional controllers and I/O buffers



than classical reconfigurable architectures, hence this reconfiguration mechanism is also often referred to processor-like reconfiguration.

DRPs at the point of writing date back more than 25 years which makes an exhaustive overview unfeasible. Instead, three different cited surveys shall give both a historical, functional, and up-to-date introduction to the field. De Sutter et al. [4] take a processor-centric view on CGRA architectures using the concept of instruction slots, that is logic where instructions can be executed. These units are connected using a simple form of interconnect like, e.g., nearest neighbor interconnect, and all have shared, or as De Sutter et al. describe them, distributed register files.

On the other hand, Hideharu Amano defines CGRAs and DRPs from a general hardware perspective. In [5], he defines a DRP to be an array of coarse-grained cells as depicted in Fig. 2, so-called PEs, consisting of one or multiple ALU and/or functional units (FU), a register file and a data manipulator [5]. The third and last survey cited for the purpose of an encompassing definition takes a similar approach as the authors of this chapter. In [6], Kiyoung Choi characterizes CGRA and by extension also DRPs via configuration granularity. All authors' definitions encompass an array of PEs and possess dynamic reconfiguration or processor-like execution and thus DRPs as architectural concept range from small reconfigurable DSP like blocks to many-core processors.

In theory, this allows the generalization of findings obtained in DRPs to be extended to far more complex brethren. In practice, however, the definition is restricted by precisely the architectural complexity as DRPs aim to be more energy efficient in more specialized fields other than, e.g., GPGPUs. This becomes also apparent in the general lack of complex caches and big register files, as well as simplistic, spatial interconnects that reduces both register file accesses and long and energy inefficient data transfers [4, 5]. For the purpose of this research project, this minimalism was a welcome attribute as it allowed an abstraction of far more complex architectures while maintaining generality. For this reason, we refer to the cited surveys [4–6] for comprehensive coverage of concrete DRP architectures.

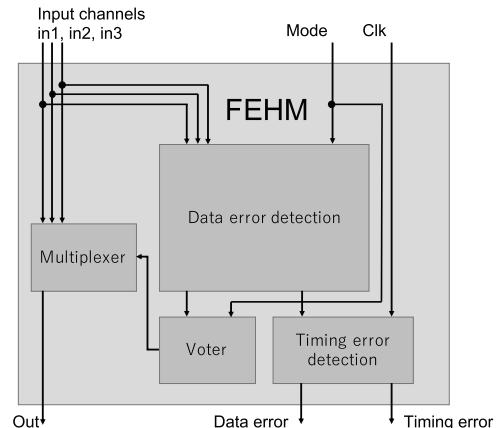
3 Exploiting Architectural Redundancy for Increased Reliability

3.1 Realizing Low-Cost TMR Using PE Clusters

Among the most apparent aspects of DRPs is their regular structure. One of the first investigations published in [7] therefore sought to utilize the structural redundancy to increase DRPs' reliability by implementing the quasi-gold standard of fault-tolerance, triple modular redundancy (TMR). The biggest issue of TMR and also the reason why it is only used in critical systems is the prohibitively high cost, i.e., everything that is secured through TMR is triplicated. These triplicated copies then have to perform the exact same operation, and at given checkpoints or most commonly at the block level of the covered component, the outputs are compared. If an error surfaced, the correct result, as well as the faulty component, are determined through a majority vote. The big drawback of this technique is the high cost, both in circuit size since three copies are required, as well as in power consumption as all have to perform the same operation all the time. This makes TMR unviable for all but the most critical applications. With reconfigurable hardware, such as DRPs, however, hardware resource can be dynamically allocated. Given the addition of error detection components, the penalty of TMR can be severely reduced as resources do not have to be committed in a hard-wired fashion, but can be reassigned temporally, or, TMR could be dynamically used for specially flagged parts of a program only.

Figure 3 depicts a simplified representation of the Flexible Error Handling Module. It consists of an actual data error detection module, containing a three-input comparator. The comparator results are fed to the voter and the timing error detection. The voter determines the correct results through a majority vote and feeds the correct channel selection to the multiplexer which then forwards the result that

Fig. 3 The flexible error handling module (FEHM)



is now presumed to be correct to the next PE or out of the DRP. The timing error detection samples the comparison results in a double buffer on C1k the clock signal as well as on a slightly delayed clock signal. If the double buffer's contents on each sample are not the same, a timing error occurred and will be appropriately signaled. Similarly, if not all comparison results are equal in the first place, it will raise a data error signal. The entire module's functionality is controlled using the Mode signal. Using this signal, the FEHM can be turned off, to Duplicate with Comparison (DWC) mode or to full TMR mode.

This switch is central to the original goal of attaining TMR at lower cost: By making the mode signal part of the instruction word, not only does this free up TMR resources when TMR is not required, but it also allows for some degradation to DWC. Evaluations of this low-cost TMR evaluation showed that even if it is used in relatively primitive DRP architectures with very fine-grained data words, the additional hardware amounts for approximately a 6% increase in area. The power consumption, on the other hand, increased by about 7.5% which can be attributed to the constantly used XOR-OR trees and double buffers used for comparison and timing error detection.

3.2 DRPs as Redundancy for CPU Pipelines

CPUs as central control units in SoCs take a vital role and thus are of great interest for reliability. However, at the same time, they are among the most difficult components to harden against any type of fault if blunt and costly instruments such as TMR are avoided. The extreme degree of dynamism and control involved in CPUs make static redundancy schemes like TMR virtually mandatory if an error-free operation needs to be guaranteed. But if some tradeoffs are permissible, dynamic redundancy schemes can be alternatively used. Such tradeoffs can be for example an absolute time limit until recovery has to complete. In both cases, however, some form of spare component is required.

While DRPs will not be able to take over a CPU's main functions, they certainly could serve as spare compute pipeline [8], thus reducing the parts that need to be hardened using conventional methods. Placing a DRP into a processor's pipeline is not a novel idea such as [9] or [10] demonstrated and makes much sense from an acceleration point of view. However, as this chapter shall highlight, they might be a good pick concerning reliability as well. When used as a static redundancy as depicted in Fig. 4 (left), DRPs can make use of their structural redundancies to provide for additional samples computed in parallel to realize true TMR. The low-cost TMR method proposed in the previous section, on the other hand, can add an additional level of reliability so that the DRP's results can be trusted and false-positives effectively prevented. As dynamic redundancy or as a spare, the DRP can take over functionality if an error has been detected using other means as depicted in Fig. 4 (right). The viability of this approach has been validated in a model implementation inspired by ARM's Cortex-M3 microcontroller. This serves

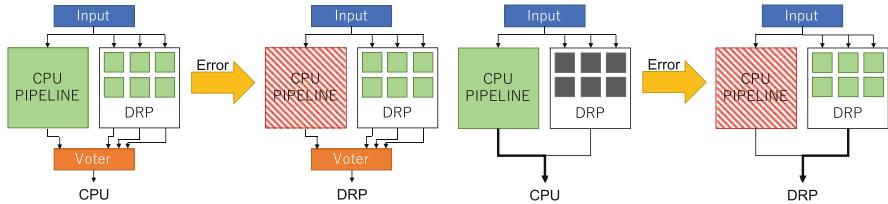


Fig. 4 DRP serving as static redundancy (left) and as dynamic redundancy (right)

as an interesting choice as ARM has its line of cores for safety-critical applications, the so-called ARM Cortex-R series with support for dual-core lock-step [11]. The results of this study as published in [8] showed that as long as support for division units is omitted in the DRP, the area overhead is far lower than the 100% overhead of an additional core, however, while of course leaving out other components to be secured separately. In this particular study, a 2 by 2 PE array, that is 4 PEs have been integrated into the CPU pipeline. Additionally, instructions and infrastructure to utilize the DRP have been added. Comparing the incurred overheads to a single-core implementation without any reliability measures, the area overhead for an implementation without hardware implemented division amounted to 20%. While this might not be an entirely fair comparison, division implementations in DRPs have a greater impact due to the far greater number of processing elements.

3.3 Dynamic Testing

In contrast to critical applications, SoCs often also accommodate non-essential functionality. For these applications, running all parts in TMR mode might be wasteful, yet a certain temporal assurance would be desirable. For example, in case of infotainment, brief dysfunction might be tolerable, but if functionality cannot be restored within a given amount of time, actual damage ensues. To avoid TMR or DWC for all applications and to implement time and probability based levels of reliability, we proposed a dynamic testing scheme for reconfigurable hardware.

Dynamic testing or also often called online testing as defined by Gao et al. [12] describes a testing method where for a known algorithm implemented in a certain component, input samples, and associated output samples are obtained and then recomputed separately. If the recomputation's results match the output samples, no error is present. If there is a mismatch, an error of the tested component is assumed.

Specifically using DRPs for dynamic testing has a big advantage: the choice between utilizing the temporal and spatial domains. Instead of competing with applications for resources on the DRP, dynamic testing resources can be allocated temporally and inserted interleaved with applications' instructions to be executed in a time-multiplexed fashion. By moving and interleaving into the time domain, testing becomes slower. However, for most non-critical applications, a couple of

seconds before a system returns to a functioning state can be tolerated. Furthermore, the spatial domain allows alternating the compute units used to recompute the samples, further making false-positives less likely apart from the error checking conducted during TMR usage.

While these two aspects make DRPs appealing for such testing schemes, time-multiplexing restricting testing to time-windows T_{TW} and further mapping into the temporal domain slowing down testing by a scaling factor s in combination with the probabilistic nature of error occurrence and detection make any estimation rather difficult. Therefore, Monte Carlo simulations can be used to estimate the behavior of dynamic testing accounting for all DRP specific aspects. For example, aspects such as reconfiguration overhead T_{OV} which has to be deducted from time-windows T_{TW} as well as scaling factors which reduces the number of samples that can be computed within one T_{TW} to detect a fault with an observation probability of q .

Consider Fig. 5, depicting a feasibility plot to detect a fault with an observation probability of $q = 10^{-5}$ and a reconfiguration overhead of 1 ms. The goal in this experiment was to detect such a fault within 2 s. The red striped regions indicate that here, it would take more than 2 s to detect the fault, whereas shades from white (fastest) to black indicate increasing detection latency DL . This result shows that even if the temporal domain is massively utilized at e.g. $s = 77$, the deadline of 2 s is still met at $DL = 1.7$ s with a time-window of 2 ms for computations thus allowing to use spatially extremely compact mappings for fault detection. This compaction also allows to further share the DRPs resources to conduct periodic checks of the

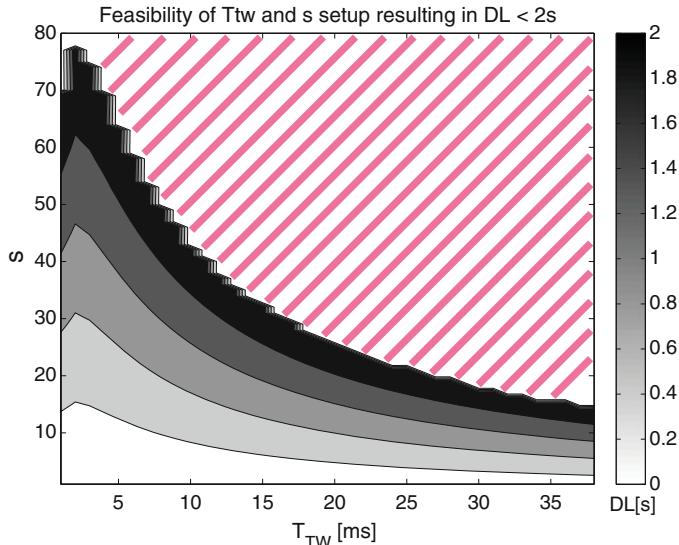


Fig. 5 Dynamic testing feasibility for a detection latency DL of 2 s by scaling factor s and time-window size for a fault with observation probability $q = 10^{-5}$, $T_{OV} = 1$ ms, a clock frequency of $F = 100$ MHz and 2σ confidence

entire surrounding SoC, expanding the reach of a reliable DRP to other system components as well.

3.4 Dynamic Remapping

Having various reliable ways to detect errors is vital as any reaction to a false-positive would just turn any reliability mechanism against itself. With low-cost TMR and dynamic testing, we have ways to detect errors and in the TMR case even to mask them. However, once a permanent fault is present and errors surface, TMR degrades to DWC, and dynamic testing is also limited to reasserting the error's presence over and over again. As DRPs are a class of reconfigurable hardware, to restore proper functionality, the applications have to be mapped anew avoiding faulty components. To do this, however, the remapping method and sufficient mapping resources are required.

In case of the FEHM equipped DRP used for our studies, two dimensions of redundancies can be utilized to run the application on unaffected PEs of the DRP. (1) spatially moving the application part of one faulty PE to a fault-free unused PE and (2) temporally adding the application part to an unaffected PE which is used for other application parts but still has the capacity to accommodate this part. As in DRPs the amount of instructions that can be stored and executed without external reconfiguration is limited, compensating for one or more faulty PEs can be a challenge in highly utilized scenarios. However, even if utilization is not critical, just moving parts around on the DRP will yield sub-optimal results, which is why the application mapping, that is resource allocation and scheduling needs to be rerun. This task, however, needs to be run on the SoCs CPU without obstructing normal operation.

To reduce the work-load of the SoC's CPU, we proposed an incremental remapping algorithm in [13]. First, the architecture graph is adjusted by removing the faulty components. Then, from this architecture graph, we extract a subgraph containing the affected PE and its vicinity. Similarly, the application graph is used to extract a subgraph containing only the application nodes mapped to the affected nodes in the architecture subgraph. With these two subgraphs, the mapping is then attempted as exemplarily depicted in Fig. 6. The mapping algorithm will try to first utilize the spatial dimension before resorting to the temporal dimension, i.e. prolonging execution time. If both dimensions do not have the resources to accommodate the application subgraph on the nodes of the pruned architecture subgraph, the architecture subgraph is enlarged by adding further neighboring nodes and remapping is retried until a new mapping has been found or the process fails altogether. If the process succeeds, the application now can run again without any errors occurring, even in non-TMR modes.

This prioritization of subgraph size over runtime, i.e., increasing subgraph size only if both dimensions cannot accommodate the application subgraph is arbitrary and other tradeoffs might be preferable. In this specific case, the priority was CPU

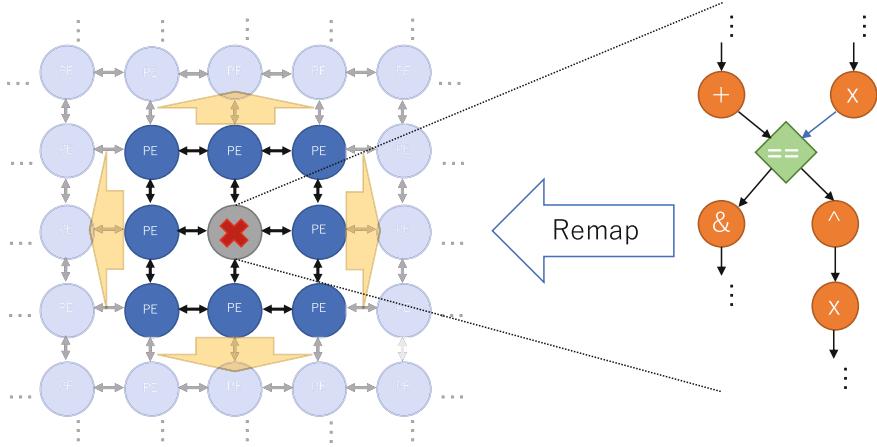


Fig. 6 Incremental remapping flow on architecture and application subgraphs to avoid faulty components. Starting on the direct neighborhood first, expanding if resources do not suffice

usage minimization, and therefore runtime and memory usage were prioritized by using the smallest subgraphs first at the expense of increased runtimes of the new mappings. For real-world applications, this needs to be carefully weighted as increased runtimes might not be viable.

3.5 Testing Reliability Schemes in Hardware

One of the big challenges of hardware manufacturing and particularly of implementing hardware-based countermeasures to reliability issues is testing and verification. Given the enormous number of input vectors and states, exhaustive testing via simulation is entirely unfeasible. While big commercial hardware emulators allow for a much greater design size and ease of use, they are also very costly. For small to medium-sized designs, FPGAs offer a sweet spot for prototype implementations. While simulations allow for easy fault injection but very slow simulation speeds, FPGAs offer speeds close to ASIC implementations but fault injection was virtually unfeasible.

To develop a prototyping platform, the Gaisler LEON3 SoC [14] served as a template into which the hardened DRP has been integrated. Parallel to this effort, different techniques for FPGA fault injection have been studied [15], culminating in the Static Mapping Library (StML) approach [16]. While instrumentation, i.e. RTL level insertion of faulty behavior allows unlimited choice in fault type and temporal behavior, it also requires for the RTL to be recompiled after each change. As the entire compilation and mapping process of our SoC took more than 4 h, this approach was abandoned. On the other hand, directly inserting faults into FPGA

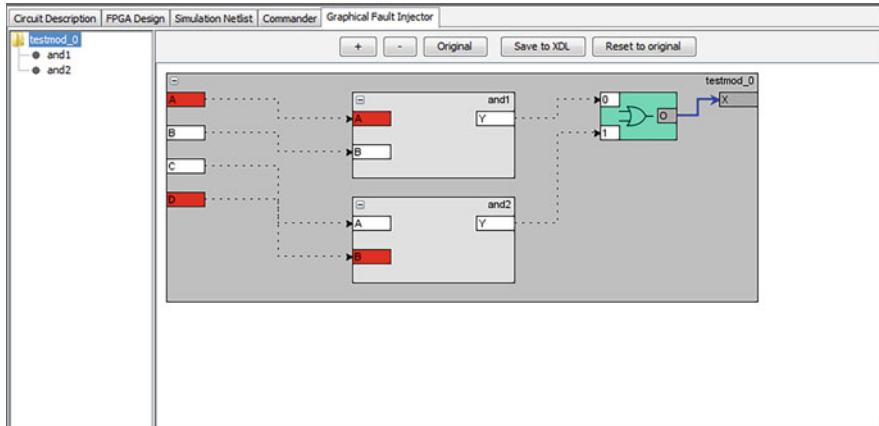


Fig. 7 StML GUI view of a sample logical AND component with fault injectable ports in each module

mappings or even the bitstream offers a simplistic way to create faulty versions of an FPGA mapping, but it offers no control over the type or location of the injected fault. This approach would not even guarantee that the FPGA mapping would behave in a faulty manner. Ideally, the exact fault location should be specifiable on RTL level to fully qualify the efficiency of the proposed architectural methods. To realize this, different intermediate results were utilized, primarily the FPGA's simulation netlist containing both RTL level structural information and FPGA mapping names in combination with the Xilinx Design Language (XDL) file containing the concrete FPGA mapping. By establishing a bidirectional link between the simulation netlist and the XDL file, StML enabled to pinpoint ports of module's implementation right down to the logic level to insert a stuck-at-zero or stuck-at-one fault. As the placed and routed XDL file can be directly altered, the only remaining step after fault injection is bitstream generation. A user-friendly GUI (Fig. 7) offering graphical representations of the implementation as well as a powerful command line interface allowed for both smooth experiment and extensive testing. Using this approach, we were able to reduce the fault-injection experiment time from hours to below 5 min, with most experiments done in below 2 min.

To showcase the viability of the proposed techniques, low-cost TMR, dynamic testing, dynamic remapping, and the FPGA prototype combined with the fault injection techniques have been successfully demonstrated at ICFPT in 2013 [17].

4 Device-Level State and Countermeasures

Below the architectural level, we studied opportunities to determine the state of semiconductor devices. Additionally, we also considered specific device-level

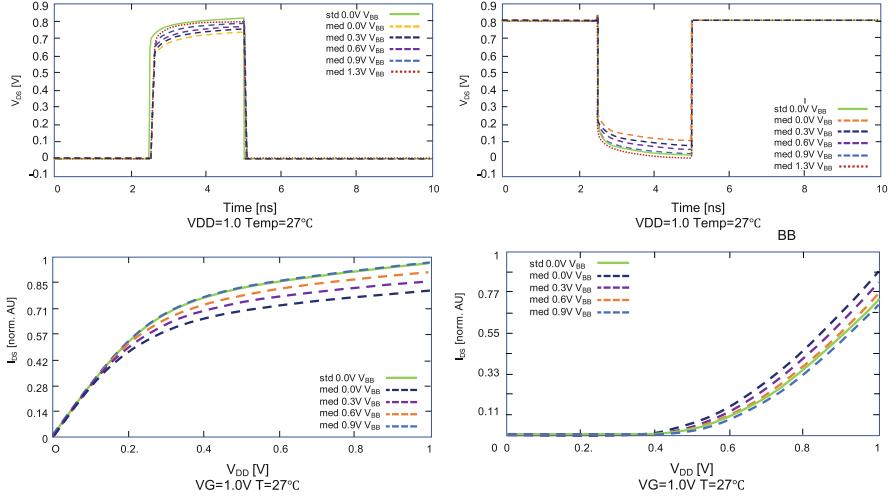


Fig. 8 Transient simulation for a single switching NMOS (upper left) and a single PMOS (upper right), as well as DC sweep of I_{DS} over $V_{DD} = V_{GS}$ for a NMOS (bottom left) as well as a PMOS transistor (bottom right), all using a medium degeneration model

countermeasures and their effects to put hardware into a more reliable state for tasks that require higher levels of reliability.

When considering how to obtain information on the state at the device level, a transistors' threshold voltage V_{TH} is a central variable [18] to consider. While of course, not all reliability phenomena manifest as an actual shift in V_{TH} , they can be modeled as such. For example, stuck-at faults are either a reduction to 0 V or ∞ V of V_{TH} or even changes in the drive current and subsequent timing faults can be viewed as such. With the semiconductor world moving either towards FinFET or FDSOI technologies, we investigated the options of FDSOI processes such as ST Microelectronics Ultra Thin Body and Box Fully Depleted Semiconductor on Insulator (UTBB-FDSOI) technology [19]. While being a planar technology, it is manufactured in a triple well process, shielding the transistor body against the substrate using a diode in reverse direction. The transistor is manufactured using a fully depleted channel which allows for further scaling to compete with FinFET processes. One of the main advantages of FDSOI technologies is that the insulated transistor body allows for very high biasing voltages previously unfeasible as it would have shorted the transistor to the substrate. As this thin body with the thin box construction equipped with a separate body electrode acts as a second gate, it is ideal to adjust V_{TH} dynamically after manufacturing. The adjustment of a transistor via this second gate is also called body biasing.

To study the possibilities to use body biasing to detect faults or even faults building up, SPICE level models have been considered. Figure 8 depicts the transient and DC analysis of a medium degeneration transistor-level model. The left side in Fig. 8 depicts an NMOS transistor whereas the right side depicts a PMOS

transistor. In each graph, there are several plots: std 0.0V V_{BB} , that is a perfectly functioning transistor without any body bias applied, med 0.0V V_{BB} transistor with a medium V_{TH} increase and no body biasing, and then several variants of the defective transistor with increasing levels of body biasing. When comparing std 0.0V V_{BB} to med 0.0V V_{BB} , it immediately becomes apparent that there is a significant gap in the rate at which the signal rises (top two graphs) and signal level, as well as a strong difference in drive current (bottom two graphs). The effects of a V_{TH} (about 45 mV NMOS and 40 mV PMOS) shift of this magnitude are, of course, relative to the operating conditions. If e.g. a couple of such transistors would be used somewhere on a critical path within a high-performance circuit, it would surely fail. On the other hand, if the circuit is used far from timing limits or if only a single transistor is considered, the effect might be barely noticeable. Given an on-chip test circuit or a known critical path, they can be used in conjunction with body biasing to measure degenerative effects. To perform such post-manufacturing bias, ideally each chip should be tested after production with a sweep over body bias levels as described in [20], with the minimum body bias, that is the maximum reverse body bias (the circuit's timing is intentionally slowed down), at which the circuit checked out functional written to a non-volatile memory. Later on, this minimum bias point can be used as a reference, i.e., if the chip or the tested component needs a higher level of body bias corrected for temperature, then some degeneration occurred. If the circuit is designed with reasonable margins, a build-up until an actual fault occurs can be thereby detected.

Similarly, body bias also allows pushing the circuit back conforming to specification. The effect depicted in Fig. 8 would be catastrophic for any performance-oriented component. However, this medium degeneration case has been chosen specifically so that corrective measures can be taken without special electrical precautions, which is up to a V_{BB} of 1.3 V in most processes. However, it should be noted that this also leads to significantly increased leakage levels and would be unfeasible for an entire chip. This being said, it neatly complements DRPs' architectural granularity, i.e. one PE would be coarse enough to mitigate the overheads of an individual body bias domain, yet it is small enough to keep the leakage overhead of strong forward biases down [21]. Additionally, finer steps of, e.g., 100 mV should be used to detect shifts in V_{TH} early on.

5 Synergistic Effects of Cross-Layer Approaches

The question following from the previous section is whether to use architectural approaches or device-level countermeasures to achieve a certain reliability objective is an extremely complex and multivariate problem. Beyond the question whether or not to use a specific technique, there are additional variables such as time, i.e. when to use these techniques, extend, that is in what parts to use them and also in regard to criticality, what techniques and with which parameters could be used at all and to what end?

In a very insightful collaboration with the FEHLER project (chapter “Soft Error Handling for Embedded Systems using Compiler-OS Interaction”), their static analysis of program criticality provided powerful means to determine key portions for reliable execution at the application level [22]. By annotating source code with keywords indicating the respective criticality, only those parts marked as critical will be additionally secured using reliability techniques. It thus was not only a great fit for selective low-cost TMR on the hardened DRP, but beyond that offered a proof-of-concept of mixed-criticality applications along with the means to identify portions critical for reliability. In [22], the targeted application was an h264 decoder. As an entertainment application, the primary metric is whether the service is provided at a certain perceived quality level above which actually occurring errors are irrelevant as they are imperceivable.

On the other end of the scale, device state monitoring allows to assess the physical state of a SoC and also its progression over time. On the architectural level, low-cost TMR or DWC allows for continuous checking, whereas dynamic testing makes sure that errors are not left undetected indefinitely, both providing vital information to potential agents. However, as shall be explored below, reactive measures cannot be determined on one layer alone.

Once the device-level state is known, this information can be used on every abstraction layer above. If for example degradation has been detected, this information can be used to minimize physical stresses by using a combination of supply voltage V_{DD} and body bias V_{BB} [2]. As proposed in the previous subsection, a concrete proposal is to counter V_{TH} drift, that is usually V_{TH} becoming larger, by using a forward body bias. As, however, Federspiel et al. found in [2], this will also increase effects like Hot Carrier Injection (HCI) stress which in turn can cause a decrease in drive current. This could lead to a feedback loop as with the method described in Sect. 4, this would appear like a V_{TH} increase and cause more forward bias to be applied, further increasing HCI stress. Thus, such action needs to be a concerted effort on the operating system level with a full view of the system state and the resources available.

For this reason, countermeasures could encompass several different options from the set of available countermeasures with the primary distinction on lifetime extension or securing error-free functionality in the presence of faults. In both cases, it should be noted that both distinctions are only two different takes on graceful degradation. In case the primary goal of reactive measures is lifetime extension, measures which incur less physical stress should be taken. If e.g. the application allows for some degeneration of the service level such as the aforementioned h264 decoding, less effort can be spent on uncritical parts of an application or it could be mapped alongside another application on a DRP. If error-free functionality is the primary goal because, e.g., the application is critical and does not allow for any degeneration, there is a two-step cascade. If the application can be remapped to fault-free components, this should be prioritized. If the resources do not permit remapping or if no resources are left, the SoC can attempt to mitigate the fault through, e.g., a forward body bias at the expense of a potentially shortened lifetime.

In all cases, however, it is clear that information from the application layer, the operating system, i.e. knowledge about what else is running on the SoC, the

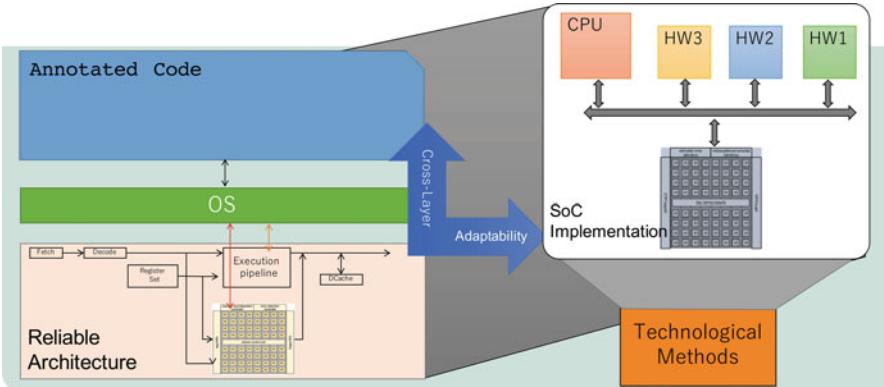


Fig. 9 Using information of all abstraction layers to realize more reliable and efficient SoC

architectural layer, what resources are available, which resources are inoperable, etc., as well as the device level, are all key to determine the optimal response. In the specific example visualized in Fig. 9, we start at the application layer by assuming reliability annotated source code. Using this source code, an appropriate mapping for example with low-cost TMR onto the DRP can be determined. Additionally, the OS might then go ahead to issue its execution without any special circuit-level tuning, i.e. increasing supply voltage or forward bias to add timing margins. Similarly, a mapping onto the CPU pipeline could be more suitable where the OS then might opt for extra forward bias as some degradation has been previously detected and the application is realizing important functionality. Not only does such a cross-layer approach as visualized in Fig. 9 help to achieve the reliability objectives, but it also is capable of more than what can be achieved on one layer at a time [23]. In this concrete example, the incorporation of multiple layers and multiple methods at specific layers allows to tailor reliability measures to requirements. Device-level information enables the system to act proactively as many phenomena can be detected at this layer in the build-up phase. Once the device layer degenerates, actors such as body biasing allow a system to restore or prolong functionality in the presence of faults.

6 Conclusion

Over a generous 6-year period in which this project was funded, the possibilities to use DRPs for increased reliability were extensively studied and also tested in prototype implementations at a functional level. This research revealed that DRPs are not only well suited for tasks that require TMR like reliability, but they can be used in numerous ways to improve the reliability of entire SoCs as well. Their simple and efficient structure allowed to research new and efficient concepts such as

dynamic remapping or body biasing for device-level sensing and countermeasures. While DRPs are still undeservingly viewed as a kind of fringe architecture concept, most of the insights gained through such architectures are easily transferable to multi- or many-core SoCs. This project showed that far more can be done in regard to reliability if multiple abstraction layers are considered in a cross-layer approach. While common wisdom still is to use TMR whenever software people use terms such as error-free or fault-tolerant, this project showed multiple options how to incorporate more specific application requirements and how to translate this into adequate reliability measures. Or in simpler terms, just-safe-enough responses to the reliability threats.

Acknowledgments As the funding agency of this project, the authors express their gratitude to the DFG for the generous funding throughout the project duration under RO-1030/13 within the Priority Program 1500. The authors also would like to thank STMicroelectronics for the cooperation on FDSOI technology. In regard to the exploration of technological aspects, the authors would also like to express their deep gratitude to the lab of Prof. Hideharu Amano at Keio University and its partnering institutions. It was a tremendous help to see to possibilities of FDSOI in silicon very early on. Finally, the authors would like to thank all the collaborating projects, persons, and especially all the alumni that were involved in this project for the fruitful discussions and interesting exchanges.

References

1. Herkersdorf, A., Aliee, H., Engel, M., Gläß, M., Gimmier-Dumont, C., Henkel, J., Kleeberger, V.B., Kochte, M.A., Kühn, J.M., Mueller-Gritschneider, D., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectron. Reliab.* **54**(6–7), 1066–1074 (2014)
2. Federspiel, X., Angot, D., Rafik, M., Cacho, F., Bajolet, A., Planes, N., Roy, D., Haond, M., Arnaud, F.: 28 nm node bulk vs FDSOI reliability comparison. In: 2012 IEEE International Reliability Physics Symposium (IRPS) pp. 3B–1. IEEE, Piscataway (2012)
3. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., et al.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**(1), 8–23 (2012)
4. De Sutter, B., Raghavan, P., Lambrechts, A.: Coarse-grained reconfigurable array architectures. In: *Handbook of Signal Processing Systems*, pp. 427–472. Springer, Berlin (2019)
5. Amano, H.: A survey on dynamically reconfigurable processors. *IEICE Trans. Commun.* **89**(12), 3179–3187 (2006)
6. Choi, K.: Coarse-grained reconfigurable array: Architecture and application mapping. *IPSJ Trans. Syst. LSI Des. Methodol.* **4**, 31–46 (2011)
7. Schweizer, T., Schlicker, P., Eisenhardt, S., Kuhn, T., Rosenstiel, W.: Low-cost TMR for fault-tolerance on coarse-grained reconfigurable architectures. 2011 International Conference on Reconfigurable Computing and FPGAs, pp. 135–140. IEEE, Piscataway (2011)
8. Lübeck, K., Morgenstern, D., Schweizer, T., Peterson, D., Rosenstiel, W., Bringmann, O.: Neues Konzept zur Steigerung der Zuverlässigkeit einer ARM-basierten Prozessorarchitektur unter Verwendung eines CGRAs. In: *MBMV*, pp. 46–58 (2016)
9. Mei, B., Vernalde, S., Verkest, D., De Man, H., Lauwereins, R.: ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In: *International Conference on Field Programmable Logic and Applications*, pp. 61–70. Springer, Berlin (2003)

10. Hoy, C.H., Govindarajuz, V., Nowatzki, T., Nagaraju, R., Marzec, Z., Agarwal, P., Frericks, C., Cofell, R., Sankaralingam, K.: Performance evaluation of a DySER FPGA prototype system spanning the compiler, microarchitecture, and hardware implementation. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 203–214. IEEE, Piscataway (2015)
11. Arm Cortex-R Series Processors. <https://developer.arm.com/products/processors/cortex-r>. Accessed 8 March 2019
12. Gao, M., Chang, H.M., Lisherness, P., Cheng, K.T.: Time-multiplexed online checking. *IEEE Trans. Comput.* **60**(9), 1300–1312 (2011)
13. Eisenhardt, S., Küster, A., Schweizer, T., Kuhn, T., Rosenstiel, W.: Spatial and temporal data path remapping for fault-tolerant coarse-grained reconfigurable architectures. In: 2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, pp. 382–388. IEEE, Piscataway (2011)
14. LEON3 Processor. <https://www.gaisler.com/index.php/products/processors/leon3?task=view&id=13>
15. Schweizer, T., Peterson, D., Kühn, J.M., Kuhn, T., Rosenstiel, W.: A fast and accurate fpga-based fault injection system. In: 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, pp. 236–236. IEEE, Piscataway (2013)
16. Peterson, D., Bringmann, O., Schweizer, T., Rosenstiel, W.: StML: bridging the gap between FPGA design and HDL circuit description. In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 278–285. IEEE, Piscataway (2013)
17. Kühn, J.M., Schweizer, T., Peterson, D., Kuhn, T., Rosenstiel, W., et al.: Testing reliability techniques for SoCs with fault tolerant CGRA by using live FPGA fault injection, In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 462–465. IEEE, Piscataway (2013)
18. Maricau, E., Gielen, G.: CMOS reliability overview. In: Analog IC Reliability in Nanometer CMOS, pp. 15–35. Springer, Berlin (2013)
19. Pelloux-Prayer, B., Blagojević, M., Thomas, O., Amara, A., Vladimirescu, A., Nikolić, B., Cesana, G., Flatresse, P.: Planar fully depleted SOI technology: The convergence of high performance and low power towards multimedia mobile applications. In: 2012 IEEE Faible Tension Faible Consommation, pp. 1–4. IEEE, Piscataway (2012)
20. Okuhara, H., Ahmed, A.B., Kühn, J.M., Amano, H.: Asymmetric body bias control with low-power FD-SOI technologies: modeling and power optimization. *IEEE Trans. Very Large Scale Integr. Syst.* **26**(7), 1254–1267 (2018)
21. Kühn, J.M., Ahmed, A.B., Okuhara, H., Amano, H., Bringmann, O., Rosenstiel, W.: MuCCRA4-BB: a fine-grained body biasing capable DRP. In: 2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX), pp. 1–3. IEEE, Piscataway (2016)
22. Schmoll, F., Heinig, A., Marwedel, P., Engel, M.: Improving the fault resilience of an H.264 decoder using static analysis methods. *ACM Trans. Embed. Comput. Syst.* **13**(1s), 31:1–31:27 (2013). <http://doi.acm.org/10.1145/2536747.2536753>
23. Herkersdorf, A., Engel, M., Glaß, M., Henkel, J., Kleeberger, V.B., Kochte, M., Kühn, J.M., Nassif, S.R., Rauchfuss, H., Rosenstiel, W., et al.: Cross-layer dependability modeling and abstraction in system on chip. In: Workshop on Silicon Errors in Logic-System Effects (SELSE) (2013)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Dependable Software Generation and Execution on Embedded Systems



Florian Kriebel, Kuan-Hsun Chen, Semeen Rehman, Jörg Henkel,
Jian-Jia Chen, and Muhammad Shafique

1 Overview

An overview of the chapter structure and the connection of the different sections is illustrated in Fig. 1. Soft error mitigation techniques like [17, 29] have shown that the software layer can be employed for enhancing the dependability of computing systems. However, to effectively use them, their overhead (e.g., in terms of power and performance) has to be considered. This also includes the option of adapting to different output accuracy requirements and inherent resilience against faults of different applications, for which appropriate metrics considering information from multiple system layers are required. Therefore, we start with a short overview of reliability and resilience modeling and estimation approaches, which not only focus on the functional correctness (like application reliability and resilience) but also consider the timeliness, i.e., determining the change of the timing behavior according to the run-time dependability, and providing various timing guarantees for real-time systems. They are used to evaluate the results of different dependable

F. Kriebel (✉) · M. Shafique

Technische Universität Wien (TU Wien), ECS (E191-02), Institute of Computer Engineering,
Wien, Austria

e-mail: florian.kriebel@tuwien.ac.at; [muhammad.shafique@tuwien.ac.at](mailto:muhmmad.shafique@tuwien.ac.at)

K.-H. Chen · J.-J. Chen

Technische Universität Dortmund, Lehrstuhl Informatik 12, Dortmund, Germany
e-mail: kuan-hsun.chen@tu-dortmund.de; jian-jia.chen@cs.uni-dortmund.de

S. Rehman

Technische Universität Wien (TU Wien), ICT (E384), Wien, Austria
e-mail: seemeen.rehman@tuwien.ac.at

J. Henkel

Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: henkel@kit.edu

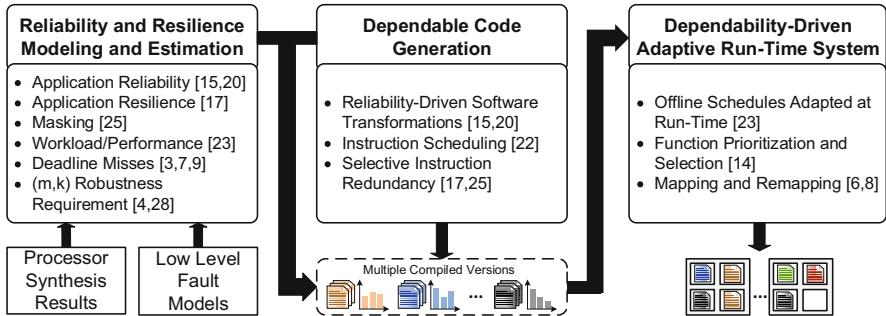


Fig. 1 Overview

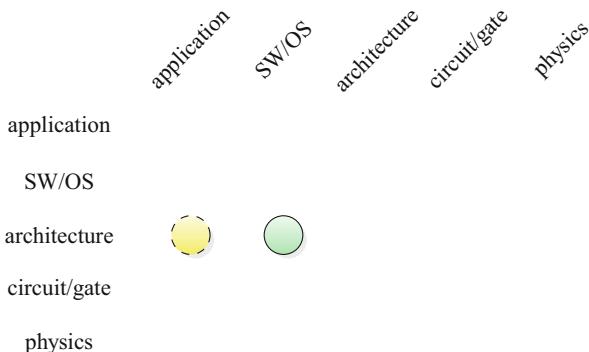


Fig. 2 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

code generation approaches, like *dependability-driven software transformations* and *selective instruction redundancy*. This enables generation of multiple compiled code versions of an application realizing different performance/energy vs. dependability trade-offs. The evaluation results and the different versions are then used by a *dependability-driven adaptive run-time system*. It considers *offline* and *online* optimizations, for instance, for selecting appropriate application versions and adapting to different workloads and conditions at run-time (like fault rate, aging, and process variation). Thereby, it finally enables a dependable execution of the applications on the target system.

As, however, not all systems are general-purpose, towards the end of the chapter an example design of a video processing system is included, which illustrates different approaches for application-specific dependability.

Embedding this chapter's content in the scope of this book and the overall projects [12, 14], the main contributions lie on the application, SW/OS, and architectural layers as illustrated in Fig. 2.

2 Dependability Modeling and Estimation

Modeling dependability at the software layer is a complex task as parameters and effects of different systems layers have to be taken into account. For an accurate yet fast evaluation of application dependability, information from lower system layers has to be considered, while abstracting it in a reasonable way to also allow for a fast estimation. For this purpose, different aspects have been separated into distinct metrics focusing on individual phenomena, as discussed below.

- The **Instruction Vulnerability Index (IVI)** [19, 25] focuses on the error probability of each instruction when being executed on different components/pipeline stages of a processor by analyzing their *spatial and temporal vulnerabilities*. This requires an analysis of *vulnerable bits* as well as *vulnerable time period*, i.e., the residence times of instructions in different components, while considering micro-architecture dependent information from the lower layers like the *area consumption* of different components and the *probability* that an error is observed at their output (see Fig. 1). The IVI of individual instructions can then be combined to estimate the vulnerability at higher granularity (e.g., Function Vulnerability Index—FVI). In this case, the susceptibility towards application failures and incorrect application outputs can be considered as well, for instance by classifying instructions into *critical* and *non-critical* ones, which is important if deviations in the application output can be tolerated.
- As not all errors occurring during the execution of an application become visible to the user due to *data flow and control flow masking*, the **Instruction Error Masking Index (IMI)** [31] provides probabilistic estimates whether the erroneous output of an instruction will be masked until the visible output of an application.
- The **Instruction Error Propagation Index (EPI)** [31] captures the effects of errors not being masked from the time of their generation until the final output of an application. It analyzes the propagation effects at instruction granularity and quantifies the impacts of the error propagation and how much it affects the final output of an application.
- Based on the information theory principles, the **Function Resilience** model [24] provides a probabilistic measure of the function’s correctness (i.e., its output quality) in the presence of faults. In contrast to the *IVI/FVI*, it avoids exposing the application details by adopting a black-box modeling technique.
- The **Reliability-Timing Penalty (RTP)** [23] model jointly accounts for the *functional correctness* (i.e., generating the correct output) and the *timing correctness* (i.e., timely delivery of an output). In this work, we studied RTP as a linear combination of functional reliability and timing reliability, where the focus (functional or timing correctness) can be adjusted. However, it can also be devised through a non-linear model depending upon the design requirements of the target system.
- The **(m,k) robustness constraint** model [4, 35] quantifies the potential inherent safety margins of control tasks. In this work, several error-handling approaches

guarantee the minimal frequency of correctness over a static number of instances while satisfying the hard real-time constraints in the worst-case scenario.

- The **Deadline-Miss Probability** [3, 9, 34] provides a statistical argument for the probabilistic timing guarantees in soft real-time systems by assuming that after a deadline miss the system either discards the job missing its deadline or reboots itself. It is used to derive the **Deadline-Miss Rate** [7], which captures the frequency of deadline misses by considering the backlog of overrun tasks without the previous assumption of discarding jobs or rebooting the system.

A more detailed description of the different models as well as their corresponding system layers are presented in chapter “Reliable CPS Design for Unreliable Hardware Platforms”.

3 Dependability-Driven Compilation

Considering the models and therewith the main parameters affecting the dependability of a system, several mitigation techniques are developed, which target to improve the system dependability on the software layer. Three different approaches are discussed in the following.

3.1 Dependability-Driven Software Transformations

Software transformations like loop unrolling have mainly been motivated by and analyzed from the perspective of improving performance. Similarly, techniques for improving dependability at the software level have mainly focused on error detection and mitigation, e.g., by using redundant instruction executions. Therefore, the following dependability-driven compiler-based software transformations [19, 25] can be used to generate different application versions, which are identical in terms of their functionality but which provide different dependability-performance trade-offs.

- *Dependability-Driven Data Type Optimization:* The idea is to implement the same functionality with different data types, targeting to reduce the number of memory load/store instructions (which are critical instructions due to their potential of causing application failures) and their predecessor instructions in the execution path. However, additional extraction/merging instructions for the data type optimization have to be taken care of when applying this transformation.
- *Dependability-Driven Loop Unrolling:* The goal is to find an unrolling factor (i.e., loop body replications), which minimizes the number of critical instructions/data (e.g., loop counters, branch instructions) that can lead to a significant deviation in the control flow causing application failures. This reduction, however, needs to be balanced, e.g., with the increase in the code size.

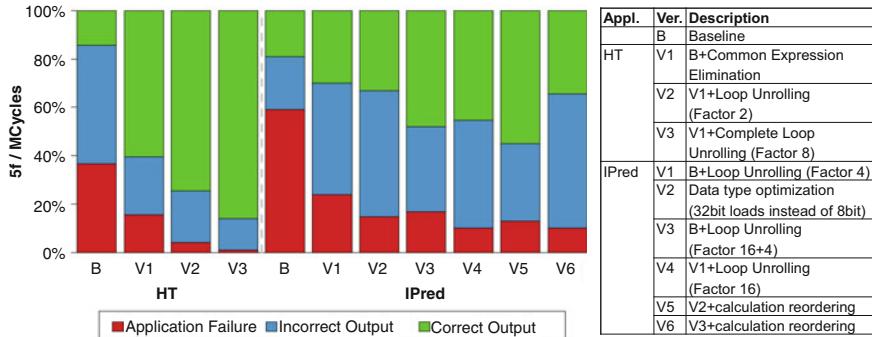


Fig. 3 Fault injection results for two applications and the generated application versions (adapted from [25])

- *Reliability-Driven Common Expression Elimination and Operation Merging:* The idea of eliminating common expressions is to achieve performance improvement due to less instructions being executed and therefore less faults being able to affect an application execution. However, excessively applying this transformation might lead to register spilling or longer residence times of data in the registers. Therefore, it needs to be evaluated carefully whether eliminating a common expression leads to a vulnerability reduction or whether the redundancy implied by a re-execution provides a benefit.
- *Reliability-Driven Online Table Value Computation:* The goal of the online table value computation is to avoid long residence times of pre-computed tables in the memory, where the values can be affected by faults and can therefore affect a large set of computations. This needs to be traded off against the performance overhead (and therefore increased temporal vulnerability) of online value computation.

As the transformations listed above also imply certain side effects (e.g., increased code size, additional instructions), they need to be applied carefully. We evaluate the above techniques using an *instruction set simulator-based fault injection approach*, where faults can be injected in different processor components (e.g., register file, PC, ALU, etc.) considering their area. It supports injecting a single or multiple faults per experiment, where each fault can itself corrupt a single or multiple bits. The results for two example applications from the *MiBench* benchmark suite [13] are shown in Fig. 3. They illustrate the effectiveness of the proposed transformations, e.g., for the “HT” application by the reduction of the application failures and incorrect outputs generated when comparing the *Baseline* application version and V3.

Finally, the dependability-driven software transformations are not only useful as a standalone technique, but can also be combined with other error mitigation techniques. For example, by reducing the number of instructions accessing the

memory, they can help reduce the required checking instructions in [29], and thereby lead to a performance improvement.

3.2 Dependability-Driven Instruction Scheduling

Instruction scheduling can significantly affect the temporal vulnerability of instructions and data, as it determines their residence time in different processor components. To improve the dependability of an application, several problems have to be addressed, which usually do not have to be considered for a performance-oriented instruction scheduling:

1. *Critical instructions should not be scheduled after multi-cycle instructions* or instructions potentially stalling the pipeline as this increases their temporal vulnerability;
2. *High residence time* (and therefore temporal vulnerability) of data in registers/memory;
3. *High spatial vulnerability*, e.g., as a consequence of using more registers in parallel.

Therefore, the dependability-driven instruction scheduling in [21, 22] estimates the vulnerabilities, and separates the instructions into *critical* and *non-critical* ones statically at compile-time before performing the instruction scheduling. Afterwards, it targets minimizing the application dependability by minimizing the spatial and temporal vulnerabilities while avoiding scheduling critical instructions after multi-cycle instructions to reduce their residence time in the pipeline. These parameters are combined to an evaluation metric called *instruction reliability weight*, which is employed by a *lookahead-based heuristic* for scheduling the instructions. The scheduler operates at the basic block level and considers the reliability weight of an instruction in conjunction with its dependent instructions to make a scheduling decision. In order to satisfy a given performance overhead constraint, the scheduler also considers the performance loss compared to a performance-oriented instruction scheduling.

3.3 Dependability-Driven Selective Instruction Redundancy

While the dependability-driven software transformations and instruction scheduling focus on reducing the vulnerability and critical instruction executions, certain important instructions might still have to be protected in applications being highly susceptible to faults. Therefore, it is beneficial to *selectively protect important instructions using error detection and recovery techniques* [24, 31], while saving the performance/power overhead of protecting every instruction.

To find the most important instructions, the error masking and error propagation properties as well as the instruction vulnerabilities have to be estimated. These results are used afterwards to prioritize the instructions to be protected, considering the performance overhead and the reliability improvement. For this, a *reliability profit function* is used, which jointly considers the protection overhead, error propagation and masking properties and the instruction vulnerabilities. The results of this analysis are finally used to select individual or a group of instructions, which maximize the total reliability profit considering a user-provided tolerable performance overhead.

4 Dependability-Driven System Software

Based on the dependability modeling and estimation approaches and the dependability-driven compilation techniques, *multiple code versions* are generated. These code versions exhibit distinct performance and dependability properties while providing the same functionality. They are then used by the **run-time system** for exploring different *reliability-performance trade-offs* by selecting appropriate application versions while adapting to changing run-time scenarios (e.g., different fault rates and workloads) for single- and multi-core systems.

4.1 Joint Consideration of Functional and Timing Dependability

The key requirement of many systems is producing correct results, where a (limited) time-wise overhead is oftentimes acceptable. However, for real-time (embedded) systems both the *functional dependability* (i.e., providing correct outputs even in the presence of hardware-level faults) and the *timing dependability* (i.e., providing the correct output before the deadline) play a central role and need to be considered jointly trading-off one against the other [23, 27]. To enable this, multiple system layers (i.e., compiler, offline system software, and run-time system software) need to be leveraged in a *cross-layer framework* to find the most effective solution [15]. For an application with multiple functions, the problem is to compose and execute it in way that jointly optimizes the functional and timing correctness. For this, the *RTP* (see Sect. 2) is used as an evaluation metric.

Figure 4a presents an overview of our approach. It is based on multiple function versions generated by employing the approaches described in Sect. 3, where additionally even different algorithms might be considered. As an example, a *sorting* application is illustrated in Fig. 4b, where the vulnerability of different algorithms and implementations as well as their execution times are compared showing different trade-offs. For generating the versions, a dependability-driven

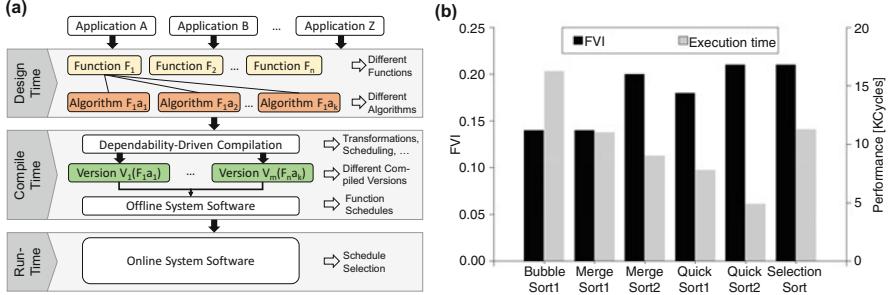


Fig. 4 (a) Overview of the design-time, compile-time, and run-time steps for generating different function/application versions. (b) Different algorithms and implementations for sorting (adapted from [23])

compilation process is used, given different implementations and a tolerable performance overhead to limit the design space. Then, only a limited number of versions from the *pareto-frontier* are selected, representing a wide spectrum of solutions.

In the next step, a *Dependability-Driven Offline System Software* generates **schedule tables** by minimizing the expected RTP. For the execution time, a probability distribution is considered, since it is not constant for all functions. For applications with only one function, the version minimizing the RTP (based on a weighting parameter) can be found by analyzing its probability for deadline misses and its reliability. For applications with multiple functions, it is required to consider that the selected version of a function is dependent on the functions executed earlier, e.g., if they finish early, a high-reliability version with a longer execution time can be selected. Therefore, a *dynamic version selection scheme* is adopted, where schedule tables are prepared offline and the scheduler selects appropriate function versions depending on the run-time behavior. Selecting a version for a particular function depends on both the functions executed *earlier*, and the functions executed *afterwards* (i.e. the predecessor and the successor functions in the execution path). The schedule tables are filled from the last function to be executed and remaining entries are added successively later, where the properties of earlier functions have to be explored and later functions can be captured by a lookup in the already filled parts of the table.

At run-time, a *Dependability-Driven Run-time System Software* selects an appropriate function version from the schedule table depending on the RTP. To execute the corresponding function, dynamic linking can be used. At the start of an application, the RTP is zero and the remaining time is the complete time until the deadline, as no function has been executed so far. With these parameters, the entry is looked up in the schedule table and the corresponding function version is executed. When one of the following functions need to be executed, the RTP observed so far is accumulated and the remaining time until the deadline is calculated. Afterwards, the corresponding table lookup is performed and a version is selected. To ensure

the correctness of the schedule tables, they should be placed in a protected memory part. As they, however, might become large, the size of the table can be reduced by removing redundant entries and entries where the RTP difference is too small. However, in this chapter, we assume that the system software is protected (for instance, using the approaches described in the OS-oriented chapters) and does not experience any failures.

In case the ordering of function executions is (partially) flexible, i.e., no/only partial precedence constraints exist, this approach can be extended by a function prioritization technique [27].

4.2 Adaptive Dependability Tuning in Multi-Core Systems

While Sect. 4.1 mainly focused on single-core systems and transient faults, the following technique will extend the scope towards multi-core systems and reliability threats having a permanent impact on the system (like process variation and aging). Thereby, different workloads on the individual cores might further aggravate the imbalance in core frequencies, which already preexists due to process variation. Consequently, *a joint consideration of soft errors, aging, and process variation is required* to optimize the dependability of the system. The goal is to achieve resource-efficient dependable application execution in multi-core systems under core-to-core frequency variation.

In a multi-core system, the software layer-based approaches can be complemented by *Redundant Multithreading (RMT)*, which is a hardware-based technique that executes redundant threads on different cores. An application can be executed with either Dual Modular Redundancy (DMR) or Triple Modular Redundancy (TMR). This broadens the mitigation solutions against the above-mentioned dependability threats, but also demands for the following problems to be solved [26].

1. The activation/deactivation of RMT has to be decided based on the properties (i.e., vulnerability, masking, performance) of the concurrently executing applications, the allowed performance overhead, and the error rate.
2. Mapping of (potentially redundant) threads to cores at run-time needs to consider the cores' states.
3. A reliable code version needs to be selected based on the performance variations of the underlying hardware and the application dependability requirements.

These problems are addressed by employing two key components: (1) a *Hybrid RMT-Tuning* technique, and (2) a *Dependability-Aware Application Version Tuning and Core Assignment* technique.

The **Hybrid RMT-Tuning** technique considers the performance requirements and vulnerability of the upcoming applications in combination with the available cores and history of encountered errors. It estimates the RTP of all applications, activates RMT for the one with the highest RTP in order to maintain the history, and

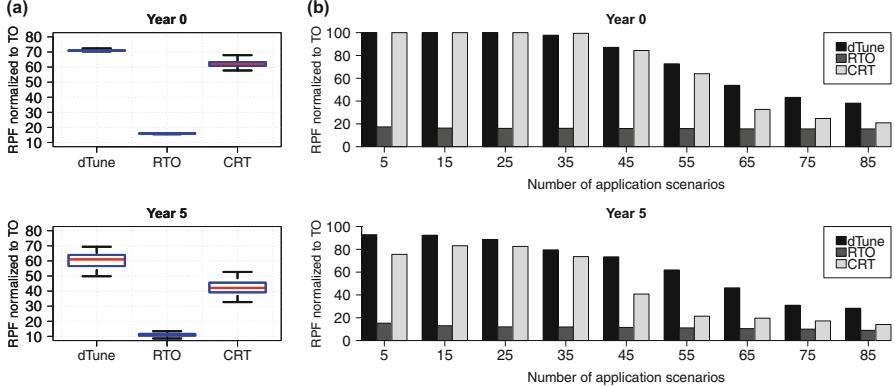


Fig. 5 (a) RPF improvements of *dTune*, *RTO*, and *CRT* normalized to *TO* for different aging years summarizing different chips and workloads. (b) RPF improvements detailing different workloads (adapted from [28])

takes RMT activation decisions based on the available cores and recent error history. For applications with RMT activated, the fastest compiled code version is selected.

After the RMT mode is decided for each application, the **Dependability-Aware Application Version Tuning and Core Assignment** is performed. It starts with an initial decision on the application version for applications where RMT has not been activated, considering their vulnerability and deadline. Then, the core allocation/mapping is performed, which takes the performance variations of individual cores (caused by process variation and aging) into account. It starts with the applications having the highest RTP and intends to allocate cores with similar performance properties to all redundant copies while also considering their distance. Finally, the application versions selected in the earlier step are tuned to improve the RTP further. Since the allocated core is now known, the potential for improving the dependability is evaluated considering the application's deadline.

Figure 5 shows the results of this approach (**dTune**) for different number of applications and different years. For the evaluation, a multi-core system with 10×10 ISA-compatible homogeneous RISC cores is used. These cores differ in their performance characteristics due to aging, where we consider NBTI-induced aging [1], and process variation, where the model of [18] is used. The comparison is done against three approaches: (1) *Chip-Level Redundant Threading (CRT)* which targets maximizing the reliability; (2) *Reliability-Timing Optimizing Technique (RTO)* jointly optimizing functional and timing dependability, but not using RMT; (3) *Timing Optimizing Technique (TO)* targeting to minimize the deadline misses. The evaluation is performed taking *TO* as a reference against which *dtune*, *CRT*, and *RTO* are compared with, where

$$\text{RPF} = 100 \times \left(1 - \frac{\sum_{t \in T} \text{RTP}(t)_Z}{\sum_{t \in T} \text{RTP}(t)_{TO}} \right). \quad (1)$$

Figure 5a shows an overview of the achieved improvements for different chip maps with process variations, scenarios of application mixes and aging years. *dTune* achieves better RPF-results compared to *TO*, *CRT*, and *RTO* for both aging years, as it jointly considers functional and timing dependability as well as the performance variation of the cores. For year 5, a wider spread of RPF-results is observed due to the decrease in processing capabilities of the chips. Figure 5b details the application workload, where it can be observed that *CRT* performs as good as *dTune* for a lower number of applications, but does not deal well with a higher number of applications due to focusing only on minimizing functional dependability.

The solution discussed above can further be enhanced by starting with a preprocessing for application version selection, as demonstrated in [5]. First, the version with the minimal reliability penalty achieving the tolerable miss rate (for applications not being protected by RMT) and the best performance (for applications protected with RMT) are selected. Afterwards, the application-to-core mapping problem is solved for the applications protected with RMT by assigning each of them the lowest-frequency group of cores possible. Then, the applications that are not protected with RMT are mapped to cores by transforming the problem to a minimum weight perfect bipartite matching problem, which is solved by applying the *Hungarian Algorithm* [16]. The decision whether to activate RMT or not is made by iteratively adapting the mode using a heuristic in combination with the application mapping approaches.

Nevertheless, solely adopting *CRT* to maximize the reliability is not good enough, since the utilization of the dedicated cores may be unnecessarily low due to low utilization tasks. If the number of redundant cores is limited, the number of tasks activating RMT is also limited. When the considered multi-core systems have multi-tasking cores rather than single thread-per-core (but homogeneous performance), the same studied problems, i.e., the activation of RMT, mapping of threads to cores, and reliable code version selection, can be addressed more nicely while satisfying the hard real-time constraints. The main idea is to use *Simultaneous Redundant Threading (SRT)* and *CRT* at the same time or even a mixture of them called *Mixed Redundant Threading (MRT)*. There are six redundancy levels characterized as a set of directed acyclic graphs (DAGs) in Fig. 6, where each node (sub-task) represents a sequence of instructions and each edge represents execution dependencies between nodes.

For determining the optimal selection of redundancy levels for all tasks, several dynamic programming algorithms are proposed in [8] to provide coarse- or fine-grained selection approaches while satisfying the feasibility under *Federated Scheduling*. In extensive experiments, the proposed approaches can generally outperform the greedy approach used in *dTune* when the number of available cores is too limited to activate *CRT* for all tasks. Since the fine-grained approach has more flexibility to harden tasks in stage-level, the decrease of the system reliability penalty is at least as good as for the coarse-grained approach. When the resources are more limited, e.g., less number of cores, the benefit of adopting the fine-grained approach is more significant.

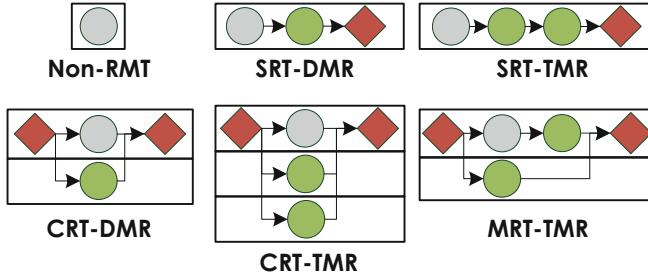


Fig. 6 DAG abstractions of the different redundancy levels, where the gray nodes are original executions and the green nodes are replicas. The red nodes represent the workload due to the necessary steps for forking the original executions and replicas, joining, and comparing the delivered results from DMR/TMR at the end of redundant multithreading. The directed edges represent the dependencies between nodes. Each block represents one core, i.e., the number of cores differs depending on the redundancy level

5 Resilient Design for System Software

Considering the adoption of error detection and recovery mechanisms due to the occurrence of soft errors from time to time, resilient designs for system software can be developed. (1) Execution versions can be determined to handle soft errors without over-provision while satisfying given robustness and timing constraints. (2) Dynamic timing guarantees can be provided without any online adaptation after a fault occurred. (3) Probabilistic analyses on deadline misses for soft real-time system. The detailed designs are presented in the following.

5.1 Adaptive Soft Error Handling

To avoid catastrophic events like unrecoverable system failures, software-based fault-tolerance techniques have the advantages in both the flexibility and application-specific assignment of techniques as well as in the non-requirement for specialized hardware. However, the main expenditure is the significant amount of time due to the additional computation incurred by such methods, e.g., redundant executions and majority voting, by which the designed system may not be feasible due to the overloaded execution demand. *Due to the potential inherent safety margins and noise tolerance, control applications might be able to tolerate a limited number of errors and only degrade its control performance.* Therefore, costly deploying full error detection and correction on each task instance might not be necessary.

To satisfy the minimal requirement of functional correctness for such control applications, (m, k) robustness constraint is proposed, which requires m out of any k consecutive instances to be correct. For each task an individual (m, k) constraint

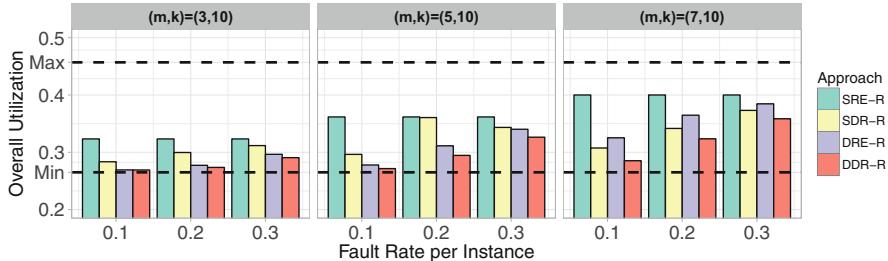


Fig. 7 Overall utilization after applying different compensation approaches on Task Path, where lower is better. Two horizontal and dashed bars represent the maximum (0.457) and the minimum utilization (0.265)

is possible to be given by other means analytically or empirically [35]. Without skipping any instances so likely achieving higher control performance, a static pattern-based approach [4] can be used to comply the reliable executions on the marked instances by following an (m, k) -pattern repeatedly to satisfy the given minimal requirement. To validate the schedulability, the multi-frame task model can then be applied to provide a hard real-time guarantee offline. A run-time adaptive approach [4] can further decide the executing version on the fly by enhancing the static pattern-based approach and monitoring the current tolerance status with sporadic replenishment counters. It is worth noting that the resulting distribution of execution jobs can still follow the (m, k) static patterns even in the worst case. Hence, the schedulability test for the static pattern-based approach can be directly used for the run-time adaptive approach as well.

Figure 7 shows the results for a self-balancing control application under different (m, k) requirements and varying fault rates. When the fault rate increases, the overall utilization of the run-time adaptive approach (DRE and DDR) also rises, since the requirement of reliable executions is increased within the application execution. Furthermore, the static pattern-based approaches (SRE and SDR) are always constant for a fixed (m, k) requirement, as the overall utilization is deterministic by the amount of job partitions. When the fault rate is as low as 10% and the (m, k) requirement is loose as $(3, 10)$, the probability of activating reliable executions is rare, and, hence, the run-time adaptive approach can closely achieve the minimum overall utilization. Overall, the results suggest that the proposed approaches can be used to serve various applications with inherent fault-tolerance depending on their perspectives, thus *avoiding over-provision under robustness and hard real-time constraints*.

5.2 Dynamic Real-Time Guarantees

When soft errors are detected, the execution time of a real-time task can be increased due to potential recovery operations. Such recovery routines may make the system

very vulnerable with respect to meeting hard real-time deadlines. This problem is often addressed by aborting *not so important* tasks to guarantee the response time of the *more important* tasks. However, for most systems such faults occur rarely and the results of *not so important* tasks might still be useful, even if they are a bit late. This implicates to not abort these not so important tasks but keep them running even if faults occur, provided that the more important tasks still meet their hard real-time deadlines. To model this behavior, the idea of *Systems with Dynamic Real-Time Guarantees* [33] is proposed, which determines if the system can provide without any online adaptation after a fault occurred, either full timing guarantees or limited timing guarantees. Please note that, this study is highly linked to the topic of mixed-criticality systems [2]. We can imagine that the system is in the low-criticality mode if *full timing guarantees* are needed, and in the high-criticality mode if only *limited timing guarantees* are provided. However, in most of the related works, such mode changes are assumed to be known, without identifying the mode change. The system only switches from low-criticality to high-criticality mode once, without ever returning to the low-criticality mode. Moreover, the low-criticality tasks are considered to be either ignored, skipped, or run with best efforts as background tasks. Such a model has received criticism as system engineers claim that it does not match their expectations in Esper et al. [11], Ernst and Di Natale [10], and Burns and Davis [2].

Suppose that a task set can be partitioned into two subsets for *more important* and *not so important* tasks, and a fixed priority order is given. To test the schedulability of a preemptive task set with constrained deadlines under a fixed priority assignment, the typical **Time Demand Analysis (TDA)** as an exact test with *pseudo-polynomial run-time* can be directly applied. To determine the schedulability for a *System with Dynamic Real-Time Guarantees*, the following three conditions must hold:

- Full timing guarantees hold, if the given task set can be scheduled according to TDA when all tasks are executed in the normal mode.
- When the system runs with limited timing guarantees, all *more important* tasks will meet their deadlines if they can be proven to be scheduled by TDA while all tasks are executed in the abnormal mode.
- Each *not so important* task has bounded tardiness if the sum of utilization over all tasks in the abnormal mode can be less than or equal to one.

To decide such a fixed priority ordering for a given task set, the **Optimal Priority Assignment (OPA)** can be applied to find a feasible fixed priority assignment, since the above schedulability test is OPA compatible. It is proven that a feasible priority assignment for a *System with Dynamic Real-Time Guarantees* can be found if one exists by using the priority assignment algorithm presented in [33], which has a much better run-time than directly applying OPA.

As faulty-aware system design is desirable in the industrial practice, having an online monitor to reflect the system status is also important. This monitor should trigger warnings if the system can only provide limited timing guarantees, and display the next time the system will return to full timing guarantees. To achieve

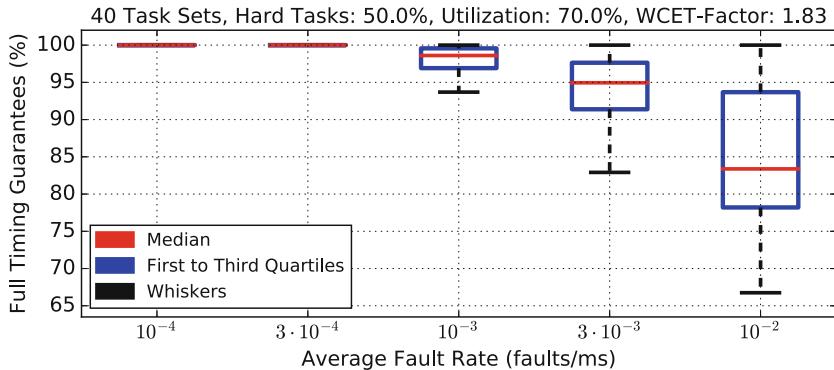


Fig. 8 Percentage of Time where *Full Timing Guarantees* can be given for task sets with utilization 70% in the normal mode under different fault rates. The median of the acceptance rates over 40 task sets is colored in red. The blue box represents the interval around this median that contains the inner 50% of those values while the whiskers display the range of the top/bottom 25% of those values

this, an approximation is needed to detect the change from *full timing guarantees* to *limited timing guarantees*, and for the calculation of an upper bound of the next time instance the system will return to full timing guarantees. To realize the routine of the online monitor, the system software has to ensure that the release pattern is still correct when a task misses its deadline and there is a helper function to keep tracking the number of postponed releases. How to enhance a real-time operating system for the previous two requirements is further discussed in [6].

Figure 8 shows the results with the percentage of time that the system was running with *full timing guarantees*. At a fault rate of 10^{-4} and 3×10^{-4} (faults/ms), the system always provides *full timing guarantees*. When the fault rate is increased, the average of the time where *full timing guarantees* are provided drops. For the worst-case values, the drop is faster but even in this case full timing guarantees are still provided $\approx 92.59\%$ and $\approx 82.91\%$ of the time for fault rates of 10^{-3} and 3×10^{-3} , respectively. This shows that even for the higher fault rates under a difficult setting, the system is still able to provide full timing guarantees for a reasonable percentage of time.

5.3 Probabilistic Deadline-Miss Analyses

When applying software fault-tolerant techniques, one natural assumption is that the *system functions normally most of time*. Therefore, it is meaningful to model the occurrence of different execution of a task by *probabilistic bounds on the worst-case execution time (WCETs)* due to potential recovery routines. This allows the system designer to provide probabilistic arguments, e.g., *Deadline-Miss Probability (DMP)*

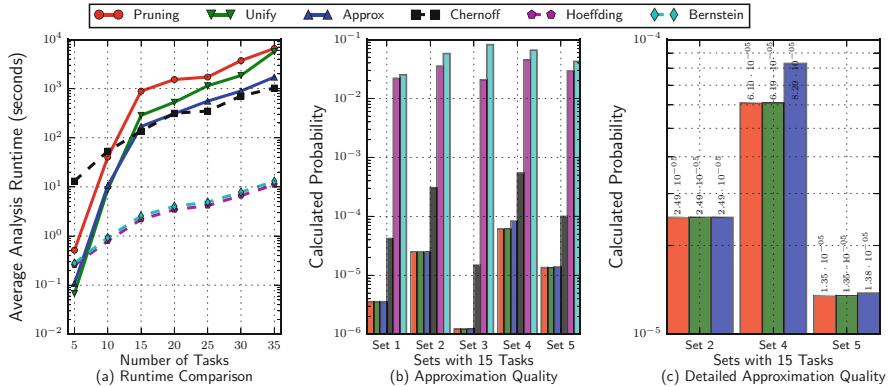


Fig. 9 (a) Average run-time with respect to task set cardinality. (b) Approximation quality for five task sets with Cardinality 15. (c) Detailed approximation quality for the convolution-based approaches

and *Deadline-Miss Rate*, as the statistical quantification to evaluate the proposed analyses scheduling algorithms, etc.

To derive the DMP, statistical approaches, i.e., *Probabilistic response time analysis* and *Deadline-misses probability analysis*, are usually taken into consideration. The state of the art of the probabilistic response time analysis is based on task-level convolution-based approaches [34]. Naturally, convolution-based approaches are computationally expensive to be applied when the number of tasks or jobs is large. Alternatively, *Deadline-Misses probability analysis* [3] is proposed, which can utilize analytical bounds, e.g., *Chernoff bounds* [3, 9], *Hoeffding's and Bernstein's inequalities* [34]. Please note that, the deadline-misses probability analysis is not better than the probabilistic response time analysis in terms of accuracy of the DMP. However, it is essentially much faster and has a better applicability in practice.

Figure 9 shows the results for randomly generated tasks sets with a normal-mode utilization 70%, fault rate 0.025, and for all tasks the execution time of abnormal mode is assumed to be two times of the normal mode. Three approaches based on the task-level convolution-based approaches [34], i.e., *Pruning*, *Unify*, *Approx*, result in similar values, roughly one order of magnitude better than *Chernoff* [3]. Although *Bernstein* [34] and *Hoeffding* [34] are orders of magnitude faster than the other approaches which are compatible with respect to the related run-time, the error of them is large compared to *Chernoff* by several orders of magnitude. The results suggest that, if sufficiently low deadline-miss probability can be guaranteed from analytical bounds, the task-level convolution-based approach then can be considered.

DMP and Deadline-Miss Rate are both important performance indicators to evaluate the extent of requirements compliance for soft real-time systems. However, the aforementioned probabilistic approaches all focus on finding the probability of the first deadline miss, and it is assumed that after a deadline miss the system either

discards the job missing its deadline or reboots itself. Therefore, the probability of one deadline miss directly relates to the deadline-miss rate since all jobs can be considered individually. If this assumption do not hold, the additional workload due to a deadline miss may trigger further deadline misses.

To derive a tight but safe estimation of the deadline-miss rate, an event-driven simulator [7] with a fault injection module can be used, which can gather deadline-miss rates empirically. However, the amount of time needed to perform the simulations is too large. Instead of simulating the targeted task set, an analytical approach [7] can leverage on the above probabilistic approaches that over-approximate the DMP of individual jobs to derive a safe upper bound on the expected deadline-miss rate.

6 Application-Specific Dependability

In this section, we focus on application-specific aspects on dependability improvement with the help of a case study on the *Context Adaptive Variable Length Coding (CAVLC)* used in the H.264 video coding standard [20, 30, 32]. It summarizes *how application-specific knowledge can be leveraged to design a power-efficient fault-tolerance technique for H.264 CAVLC*.

CAVLC is an important part of the coding process and is susceptible to errors due to its context adaptivity, multiple coding tables, and complex structure. It transforms an input with a fixed length to flexible-length code consisting of *codeword/codelength* tuples. The impact of a single error on the subjective video quality is illustrated in Fig. 10a, which shows a significant distortion in a video frame when the header of a macroblock (i.e., a 16×16 pixels block) is affected. Faults during the CAVLC can also propagate to subsequent frames or even lead to encoder/decoder crashes.

Consequently, it is required to address these problems during the CAVLC execution. To reduce the overhead compared to generic solutions, application-

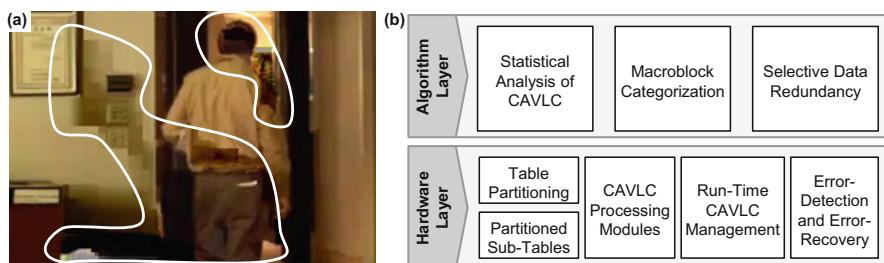


Fig. 10 (a) Example of a corrupted frame showing the effects of a single-bit error. (b) Overview of the contributions for the dependable CAVLC and the corresponding system layers (adapted from [32])

specific knowledge is considered. Specifically, Fig. 10b shows an overview of the dependable CAVLC with contributions on the architecture and algorithm/software layer, which are based on exploiting the video content properties and performing a statistical analysis of CAVLC.

- **Application-Specific Knowledge** is considered by (1) an analysis of error probabilities, (2) distribution of different syntax elements, (3) algorithmic properties, and (4) specifications defined by the standard. It includes an analysis of different macroblock categories (homogeneous/textured, fast/slow motion). The most important observations are that the *total non-zero coefficients have a significant influence on the error probabilities of different syntax elements*. They can be used to detect potential errors at the algorithm level if the macroblock properties are known.
- **Selective Data Redundancy:** Based on the application-specific knowledge obtained by the analysis, selected CAVLC data (e.g., quantized coefficients, coefficient statistics, etc.) can be protected by storing redundant copies and parity data in unused data structures. This is possible, e.g., for the quantized coefficients as the quantization often leads to unused (“0”) entries, where redundant data can be stored in a reflected fashion. *Only the low-frequency coefficients are protected in case the space is insufficient.*
- **Dependable CAVLC Hardware Architecture:** The original and redundant values are loaded by a hardware module, which performs error detection and error recovery. In case of a mismatch, the parity is calculated and compared to the stored one, so that the correct entry can be found. A recovery is even possible if both entries are corrupted by reloading the original block and performing the quantization step again. Additionally, the coding tables used by CAVLC for obtaining the *codeword* and *codelength* need to be protected. For that, the individual tables are split into different sub-tables, where the partitioning decision is based on the distribution of the syntax elements. *Sub-tables not being accessed frequently can then be power-gated for leakage energy savings.* For each sub-table, a block parity-based protection approach is used for error detection, trading-off the additional memory required and the protection offered. Furthermore, entries not being accessed due to the algorithm properties and zero-entries are not stored. Similarly, *the data in tables containing mirrored entries also has to be stored only once*, thereby further reducing the memory requirements and leakage energy.
- **Run-Time Manager:** The dependable CAVLC architecture is controlled by a run-time manager which activates/deactivates the power-gating of the memory parts storing the sub-tables, loads the requested data from the tables, and controls error detection and reloading of data.
- **Dependable CAVLC Processing Flow:** The overall flow starts with a macroblock characterization, which determines the power-gating decision. Then, highly probable values for the syntax elements are predicted, which are used later for the algorithm-guided error detection. Afterwards, the header elements

are loaded by the hardware module performing error detection and error recovery. Finally, the quantized coefficients are coded by CAVLC for each 4×4 block.

This example architecture illustrates how application-specific knowledge can be leveraged to improve the design decisions for enhancing the dependability of the system and its power consumption. It achieves significant improvements in terms of the resulting video quality compared to an unprotected scheme. Moreover, *leakage energy savings of 58% can be achieved by the application-guided fault-tolerance and table partitioning.*

7 Conclusion

Dependability has emerged as an important design constraint in modern computing systems. For a cost-effective implementation, a cross-layer approach is required, which enables each layer to contribute its advantages for dependability enhancement. This chapter presented contributions focusing on the architecture, SW/OS, and application layers. Those include modeling and estimation techniques considering functional correctness and timeliness of applications as well as approaches for generating dependable software (e.g., by dependability-aware software transformations or selective instruction redundancy). Additionally, the run-time system is employed for selecting appropriate dependable application versions and adapting to different workloads and run-time conditions, enabling a tradeoff between performance and dependability. It has furthermore been shown how application-specific characteristics can be used to enhance the dependability of a system, taking the example of a multimedia application.

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500—spp1500.itec.kit.edu).

References

1. Alam, M.A., Kufluoglu, H., Varghese, D., Mahapatra, S.: A comprehensive model for PMOS NBTI degradation: recent progress. *Microelectron. Reliab.* **47**(6), 853–862 (2007). <https://doi.org/10.1016/j.microrel.2006.10.012>
2. Burns, A., Davis, R.: Mixed criticality systems-a review, 7th edn. Tech. rep., University of York (2016)
3. Chen, K., Chen, J.: Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In: 12th IEEE International Symposium on Industrial Embedded Systems, SIES 2017, Toulouse, 14–16 June 2017, pp. 1–8 (2017). <https://doi.org/10.1109/SIES.2017.7993392>

4. Chen, K., Bönnighoff, B., Chen, J., Marwedel, P.: Compensate or ignore? Meeting control robustness requirements through adaptive soft-error handling. In: Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems, LCTES 2016, Santa Barbara, 13–14 June 2016, pp. 82–91 (2016). <https://doi.org/10.1145/2907950.2907952>
5. Chen, K., Chen, J., Kriebel, F., Rehman, S., Shafique, M., Henkel, J.: Task mapping for redundant multithreading in multi-cores with reliability and performance heterogeneity. *IEEE Trans. Comput.* **65**(11), 3441–3455 (2016). <https://doi.org/10.1109/TC.2016.2532862>
6. Chen, K.H., Von Der Brüggen, G., Chen, J.J.: Overrun handling for mixed-criticality support in RTEMS. In: WMC 2016, Proceedings of WMC 2016, Porto (2016). <https://hal.archives-ouvertes.fr/hal-01438843>
7. Chen, K., von der Brüggen, G., Chen, J.: Analysis of deadline miss rates for uniprocessor fixed-priority scheduling. In: 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, 28–31 August 2018, pp. 168–178 (2018). <https://doi.org/10.1109/RTCSA.2018.00028>
8. Chen, K., von der Brüggen, G., Chen, J.: Reliability optimization on multi-core systems with multi-tasking and redundant multi-threading. *IEEE Trans. Comput.* **67**(4), 484–497 (2018). <https://doi.org/10.1109/TC.2017.2769044>
9. Chen, K.H., Ueter, N., von der Brüggen, G., Chen, J.: Efficient computation of deadline-miss probability and potential pitfalls. In: Design, Automation and Test in Europe, DATE 19, Florence, 25–29 March 2019
10. Ernst, R., Di Natale, M.: Mixed criticality systems - a history of misconceptions? *IEEE Des. Test* **33**(5), 65–74 (2016). <https://doi.org/10.1109/MDAT.2016.2594790>
11. Esper, A., Nelissen, G., Nélis, V., Tovar, E.: How realistic is the mixed-criticality real-time system model? In: RTNS, pp. 139–148 (2015)
12. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. CAD Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
13. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: Mibench: a free, commercially representative embedded benchmark suite. In: Proceedings of the Workload Characterization, 2001, WWC-4. 2001 IEEE International Workshop, WWC '01, pp. 3–14. IEEE Computer Society, Washington (2001). <https://doi.org/10.1109/WWC.2001.15>
14. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October 2011, pp. 69–78 (2011). <https://doi.org/10.1145/2039370.2039384>
15. Henkel, J., Bauer, L., Dutt, N.D., Gupta, P., Nassif, S.R., Shafique, M., Tahoori, M.B., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: The 50th Annual Design Automation Conference 2013, DAC '13, Austin, 29 May–07 June 2013, pp. 99:1–99:10 (2013). <https://doi.org/10.1145/2463209.2488857>
16. Kuhn, H.: The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.* **2**, 83–98 (1955). <https://doi.org/10.1002/nav.20053>
17. Oh, N., Shirvani, P.P., McCluskey, E.J.: Error detection by duplicated instructions in superscalar processors. *IEEE Trans. Reliab.* **51**(1), 63–75 (2002). <https://doi.org/10.1109/24.994913>
18. Raghunathan, B., Turakhia, Y., Garg, S., Marculescu, D.: Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: Design, Automation and Test in Europe, DATE 13, Grenoble, 18–22 March 2013, pp. 39–44 (2013). <https://doi.org/10.7873/DATE.2013.023>
19. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Reliable software for unreliable hardware: embedded code generation aiming at reliability. In: Proceedings of the 9th International

- Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWEEK '11 Seventh Embedded Systems Week, Taipei, 9–14 October 2011, pp. 237–246 (2011). <https://doi.org/10.1145/2039370.2039408>
- 20. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Revc: computationally reliable video coding on unreliable hardware platforms: a case study on error-tolerant H.264/AVC CAVLC entropy coding. In: 18th IEEE International Conference on Image Processing, ICIP 2011, Brussels, 11–14 September 2011, pp. 397–400 (2011). <https://doi.org/10.1109/ICIP.2011.6116533>
 - 21. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: RAISE: reliability-aware instruction scheduling for unreliable hardware. In: Proceedings of the 17th Asia and South Pacific Design Automation Conference, ASP-DAC 2012, Sydney, 30 January–2 February 2012, pp. 671–676 (2012). <https://doi.org/10.1109/ASPDAC.2012.6165040>
 - 22. Rehman, S., Shafique, M., Henkel, J.: Instruction scheduling for reliability-aware compilation. In: The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, 3–7 June 2012, pp. 1292–1300 (2012). <https://doi.org/10.1145/2228360.2228601>
 - 23. Rehman, S., Toma, A., Kriebel, F., Shafique, M., Chen, J., Henkel, J.: Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations. In: 19th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2013, Philadelphia, 9–11 April 2013, pp. 273–282 (2013). <https://doi.org/10.1109/RTAS.2013.6531099>
 - 24. Rehman, S., Shafique, M., Aceituno, P.V., Kriebel, F., Chen, J., Henkel, J.: Leveraging variable function resilience for selective software reliability on unreliable hardware. In: Design, Automation and Test in Europe, DATE 13, Grenoble, 18–22 March 2013, pp. 1759–1764 (2013). <https://doi.org/10.7873/DATE.2013.354>
 - 25. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. IEEE Trans. CAD Integr. Circuits Syst. **33**(11), 1597–1610 (2014). <https://doi.org/10.1109/TCAD.2014.2341894>
 - 26. Rehman, S., Kriebel, F., Sun, D., Shafique, M., Henkel, J.: dtune: leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In: The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, 1–5 June 2014, pp. 84:1–84:6 (2014). <https://doi.org/10.1145/2593069.2593127>
 - 27. Rehman, S., Chen, K., Kriebel, F., Toma, A., Shafique, M., Chen, J., Henkel, J.: Cross-layer software dependability on unreliable hardware. IEEE Trans. Comput. **65**(1), 80–94 (2016). <https://doi.org/10.1109/TC.2015.2417554>
 - 28. Rehman, S., Shafique, M., Henkel, J.: Reliable Software for Unreliable Hardware - A Cross Layer Perspective. Springer (2016). <https://doi.org/10.1007/978-3-319-25772-3>
 - 29. Reis, G.A., Chang, J., Vachharajani, N., Rangan, R., August, D.I., Mukherjee, S.S.: Software-controlled fault tolerance. Trans. Archit. Code Optim. **2**(4), 366–396 (2005). <https://doi.org/10.1145/1113841.1113843>
 - 30. Shafique, M., Zatt, B., Rehman, S., Kriebel, F., Henkel, J.: Power-efficient error-resiliency for H.264/AVC context-adaptive variable length coding. In: 2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, 12–16 March 2012, pp. 697–702 (2012). <https://doi.org/10.1109/DATe.2012.6176560>
 - 31. Shafique, M., Rehman, S., Aceituno, P.V., Henkel, J.: Exploiting program-level masking and error propagation for constrained reliability optimization. In: The 50th Annual Design Automation Conference 2013, DAC '13, Austin, 29 May–07 June 2013, pp. 17:1–17:9 (2013). <https://doi.org/10.1145/2463209.2488755>
 - 32. Shafique, M., Rehman, S., Kriebel, F., Khan, M.U.K., Zatt, B., Subramanyan, A., Vizzotto, B.B., Henkel, J.: Application-guided power-efficient fault tolerance for H.264 context adaptive variable length coding. IEEE Trans. Comput. **66**(4), 560–574 (2017). <https://doi.org/10.1109/TC.2016.2616313>
 - 33. von der Bruggen, G., Chen, K., Huang, W., Chen, J.: Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In: 2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, 29 November–2 December 2016, pp. 303–314 (2016). <https://doi.org/10.1109/RTSS.2016.037>

34. von der Brüggen, G., Piatkowski, N., Chen, K., Chen, J., Morik, K.: Efficiently approximating the probability of deadline misses in real-time systems. In: 30th Euromicro Conference on Real-Time Systems, ECRTS 2018, Barcelona, 3–6 July 2018, pp. 6:1–6:22 (2018). <https://doi.org/10.4230/LIPIcs.ECRTS.2018.6>
35. Yayla, M., Chen, K., Chen, J.: Fault tolerance on control applications: empirical investigations of impacts from incorrect calculations. In: 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems, EITEC@CPSWeek 2018, 10 April 2018, Porto, pp. 17–24 (2018). <https://doi.org/10.1109/EITEC.2018.00008>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Fault-Tolerant Computing with Heterogeneous Hardening Modes



Florian Kriebel, Faiq Khalid, Bharath Srinivas Prabakaran, Semeen Rehman, and Muhammad Shafique

1 Introduction

Recent technological advancements in the field of transistor fabrication, such as FinFETs and GAAFETs, have led to significant improvements in the performance of next-generation multi-core processors but at the expense of an increased susceptibility to reliability threats such as soft errors [4, 37], aging [14], and process variations [14]. These threats generate permanent and/or temporary faults that can lead to unexpected system failures and can be disastrous to several safety-critical applications such as automotive, healthcare, aerospace, etc., as well as high-performance computing systems. Therefore, several techniques have been proposed to detect, prevent, and mitigate these reliability threats across the computing stack ranging from the transistor and circuit layer [27, 43] to the software/application layer [2, 42, 44]. Oftentimes, (full-scale) redundancy is employed at the hardware and the software layers, for example, at the *software layer*, by executing multiple redundant thread versions of an application, either spatially or temporally, and at the *hardware layer*, by duplicating or triplicating the pipeline, i.e., Double/Triple Modular Redundancy (DMR/TMR) [28, 29, 32, 46]. However, these reliability techniques exhibit several key limitations, as discussed below:

1. Ensuring temporal redundancy at the software layer, by executing multiple redundant threads of a given application on the same core, would incur a significant performance overhead.

F. Kriebel (✉) · F. Khalid · B. S. Prabakaran · S. Rehman · M. Shafique
Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: florian.kriebel@tuwien.ac.at; faiq.khalid@tuwien.ac.at;
bharath.prabakaran@tuwien.ac.at; semeen.rehman@tuwien.ac.at;
muhammad.shafique@tuwien.ac.at

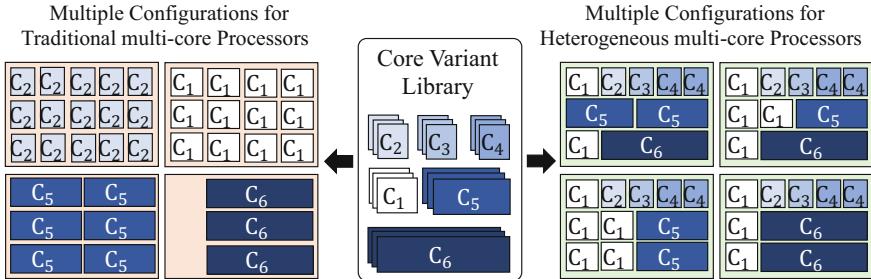


Fig. 1 Different configurations for mitigating dependability threats for traditional (homogeneous) and heterogeneous multi-core systems, respectively

2. Executing multiple redundant threads in multiple cores concurrently, instead of a single core, provides spatial redundancy and nullifies the performance overhead caused by the temporal redundancy. However, due to the activation of multiple cores, this technique incurs a significant power overhead.
3. Similarly, fabricating redundant hardware components to provide full-scale TMR across the pipeline incurs additional area, power, and energy overheads including additional on-chip resources for the data correction and control units.
4. Moreover, these techniques are not adaptive with respect to the dependability requirements of the applications, as well as their inherent error tolerance, during their execution.

To address these limitations, we proposed the **reliability-heterogeneous architectures** in [20, 21, 33, 34]. They offer different types of reliability modes in different cores (i.e., the so-called *reliability-heterogeneous cores*), realized through hardening of different pipeline components using different reliability mechanisms. Hence, such processors provide a foundation for design- and run-time trade-offs in terms of reliability, power/energy, and area. Their motivation arises from the fact that different applications exhibit varying degrees of error tolerance and inherent masking to soft errors due to data and control flow masking. Hence, depending upon the executing applications, their tasks can be mapped to a set of *reliability-heterogeneous cores* to mitigate soft errors, as shown in Fig. 1.

Although this solution significantly reduces the power/energy and performance overheads, it requires a sophisticated *run-time management system* that performs an appropriate code-to-core mapping of the applications, based on their requirements and given power/performance constraints. This requires enabling certain features across the hardware and software layers such as additional control logic, core monitoring units at the hardware layer, and a run-time manager at the software layer. Embedding this chapter's content in the scope of this book and the overall projects [11, 13], the focus of this chapter is limited to the design of such

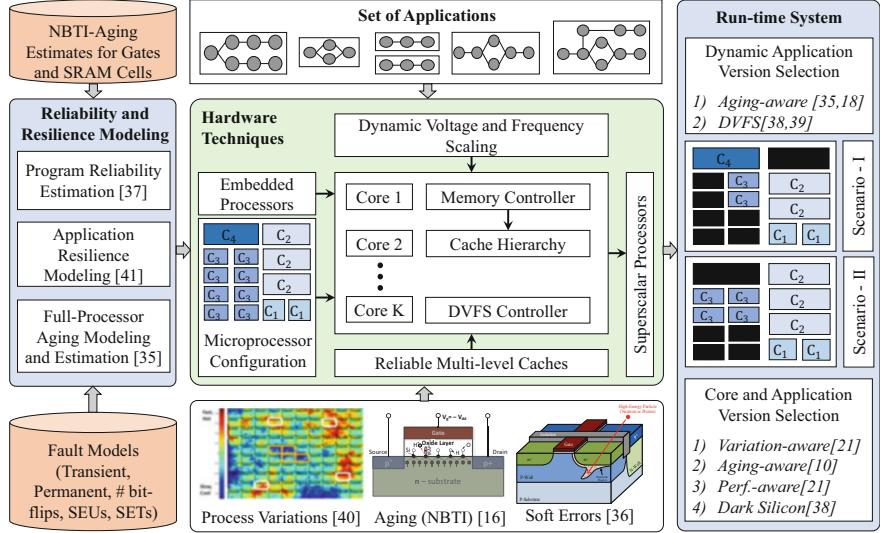


Fig. 2 An overview of dependable computing with heterogeneous hardening modes (adapted from [34]). Image sources: [16, 37]

hardware/software techniques that can enable heterogeneous dependable computing (see Fig. 3).

Typically, the hardware solutions for dependable heterogeneous architectures consist of the following three phases (see Fig. 2):

- 1. Reliability and Resilience Modeling:** First, the effects of different reliability threats (i.e., soft errors, aging, and process variations) on different components of a given multi-core system and different applications are modeled and analyzed based on mathematical analysis, simulation, and/or emulation.
- 2. Hardware Techniques:** Based on the vulnerability analysis of the previous step, multiple reliability-heterogeneous core variants are developed by hardening a combination of the pipeline and/or memory components. Similarly, an analysis of multi-level cache hierarchies has led to the design of multiple heterogeneous reliability cache variants and reliability-aware reconfigurable caches.
- 3. Run-time System:** Afterwards, appropriate task-to-core mapping as well as reliable code version selection are performed, while satisfying the application's reliability requirement and minimizing the power/area overheads. These problems can also be formulated as constrained optimization problems.

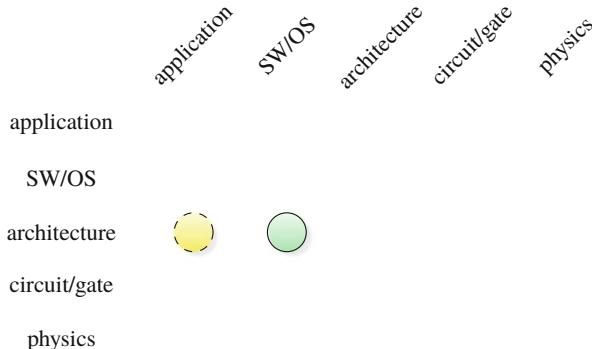


Fig. 3 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

2 Fault-Tolerant Heterogeneous Processors

Reliability threats not only affect the computing cores in the microprocessors, but can also significantly affect the on-chip memory sub-systems, like multi-level caches. This section provides an overview of our techniques for developing reliability-heterogeneous in-order processors and multi-level cache hierarchies. Unlike the traditional homogeneous dependable processors, the development of reliability-heterogeneous processors not only requires design-time efforts to develop multiple variable-reliability processor variants but also requires a run-time management system that can efficiently cater the applications’ requirements (see Fig. 2). Therefore, as illustrated in Fig. 4, developing these hardware techniques can be divided into two phases, namely, design-time and run-time:

1. **Design-Time:** At design-time, first the overall vulnerability of a processor is analyzed. Based on this analysis, we develop hardware techniques that can be used to design reliability-heterogeneous processor cores (see Sect. 2.1). In the next step, these hardened cores are integrated into an architectural-level simulator to evaluate their effectiveness. Similarly, we evaluate the vulnerability of caches, based on which hardware techniques are designed to mitigate the effects of reliability threats in caches (see Sect. 2.2). These reliability-aware caches and multiple reliability-heterogeneous cores are used to design a reliability-heterogeneous processor, as depicted by *Design-Time* in Fig. 4.
2. **Run-time:** To effectively use the reliability-heterogeneous processor, an *adaptive run-time manager for soft error resilience (ASER)* is used to estimate the reliability requirements of the applications (as well as their resilience properties), and to efficiently map their threads to a set of hardened cores while adhering to the user and performance constraints, as depicted by *Run-Time* in Fig. 4.

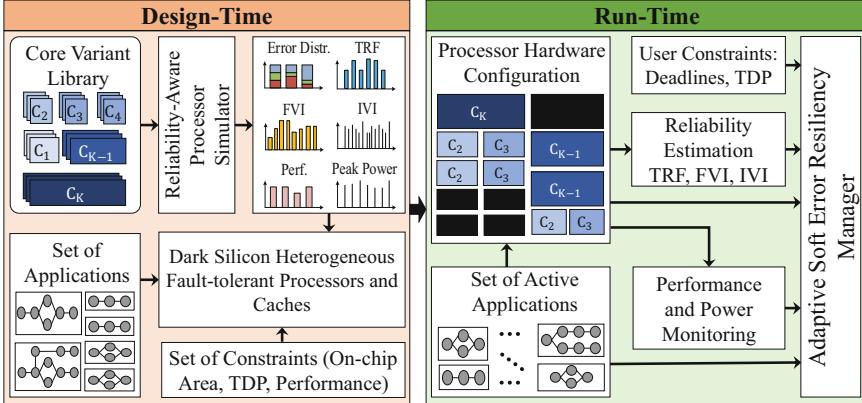


Fig. 4 The design- and run-time methodology to develop dependable heterogeneous processors (adapted from [20, 34])

2.1 Hardening Embedded Processors

To design the hardened cores for reliability-heterogeneous architectures, first, we analyze the vulnerability of these cores to different reliability threats. Based on this vulnerability analysis, instead of enabling full-scale DMR/TMR, we design the micro-architecture of different hardened (in-order) cores. These cores have distinct reliability mechanisms in different pipeline components (ranging from unprotected to fully-protected), but implement the same instruction set architecture (ISA); see the core variant library in Fig. 4. Hence, these cores provide a trade-off between reliability, area, and power/energy consumption. Since not all transistors on a chip can be powered-on at the same time (i.e., the dark silicon problem [7, 31, 41]), we leverage this fact to integrate many different hardened cores to develop a reliability-heterogeneous ISO-ISA processor [20, 21], while adhering to hardware and user-defined constraints (e.g., area, power) considering a target domain (i.e., given a particular set of target applications).

To cater for the application-specific requirements at run-time, an *adaptive run-time manager for soft error resilience (ASER)* determines an efficient application-to-core mapping considering the application's vulnerability and deadline requirements, system performance, thermal design power (TDP), and other user-defined constraints. For example, Fig. 5 depicts the varying reliability improvements of the ASER run-time system approach in comparison with multiple state-of-the-art reliability techniques such as *TRO* (timing dependability optimization aiming at minimizing the deadline misses), *RTO* (optimizing functional as well as timing dependability), *Full-TMR* (activating full TMR), and *AdTMR* (deactivating TMR when the vulnerability lies below a pre-defined threshold). The reliability is measured using the Reliability Profit Function (RPF) which is defined as follows:

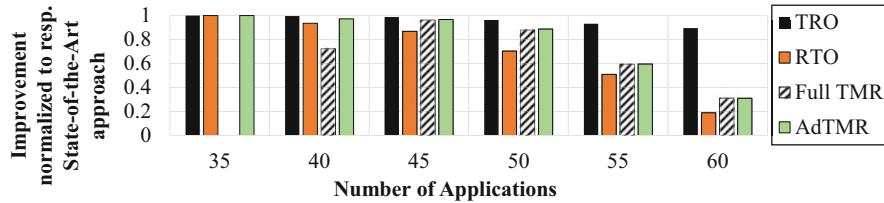


Fig. 5 Reliability Profit Function improvement over different state-of-the-art reliability techniques, i.e., timing reliability optimization (TRO), functional and timing reliability (RTO), TMR, adaptive TMR (adTMR) (adapted from [20])

$$RPF = 1 - \left(\frac{RTP(t)_{ASER}}{RTP(t)_Z} \right) \quad (1)$$

where $\forall t \in T$, T is a set of run-time concurrently executing application tasks ($T = \{T_1, T_2, \dots, T_M\}$), $Z \in \{TRO, RTO, TMR, adTMR\}$, and RTP is the Reliability-Timing Penalty [38]. Note, a higher value of RPF translates to a better reliability. The ASER approach achieves 58–96% overall system reliability improvements when compared to these four state-of-the-art techniques.

2.2 Reliability Techniques for Multi-Level Caches

In any microprocessor, on-chip memories play a significant role to improve the throughput and performance of an application. Moreover, memory elements (such as caches) are even more susceptible to soft errors compared to the computing elements (i.e., logic) as they occupy a significant portion of the total on-chip area [9]. Therefore, for designing dependable multi/many-core processors, different (individual) cache levels as well as the complete cache hierarchy (considering inter-dependency between different cache levels) have to be analyzed and optimized for mitigating reliability threats.

2.2.1 Improving the Reliability of Last-Level Caches

Dynamic reconfiguration of the caches with respect to the running applications has a significant impact on the vulnerability of the on-chip last-level caches, as shown in Fig. 6. It can be observed from the vulnerability analysis of a given cache configuration (see Fig. 6) that due to different access patterns and occupancy of last-level caches for the application, the vulnerability also varies depending on the executing applications. This dynamic change in vulnerability at run-time can be exploited to improve the reliability of the last-level cache. Therefore, dynamic reconfiguration of the last-level cache is exploited to develop a *reliability-aware*

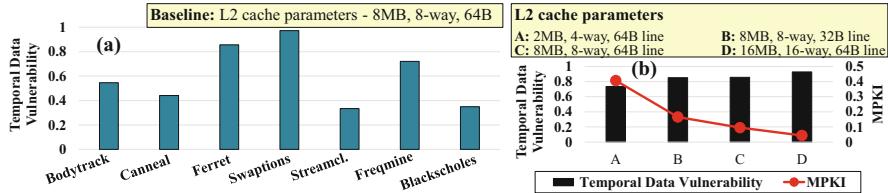


Fig. 6 (a) Vulnerability analysis of different applications from the PARSEC benchmark for the baseline case (L2 cache parameters—8 MB, 8-way, 64 B). (b) Vulnerabilities and cache misses (MKPI) for the *Ferret* application for different cache configurations (adapted from [19])

reconfigurable cache architecture [19, 22]. Towards this, we aim at reducing the vulnerability of concurrently executing applications by employing the following features:

1. A methodology to *quantify the cache vulnerability* with respect to concurrently executing applications.
2. A method for *lightweight online prediction of the application vulnerability online* based on the cache utilization and performance data.
3. A methodology to *dynamically reconfigure the last-level cache* at run-time that targets at minimizing the application vulnerability w.r.t. cache while keeping the performance overhead low, or within a tolerable bound.

This reliability-aware cache reconfiguration [22] can also be applied in conjunction with the error correcting codes (ECCs). For example, Single Error Correcting–Double Error Detecting (SEC-DED) [6] can be combined with the reliability-aware cache reconfiguration [22] to improve reliability in multi-bit error scenarios, or in cases where only some of the cache partitions are ECC-protected due to the area constraints.

2.2.2 Improving the Reliability of the Complete Cache Hierarchy

The application vulnerability towards soft errors is not only dependent on the individual utilization or dynamic reconfiguration of the different individual cache levels (e.g., L1 or L2). Rather, the *vulnerability interdependencies across different cache* levels also have significant impact on the reliability of the system. Therefore, the vulnerability of the concurrently executing applications with respect to the corresponding cache configuration can further be improved by considering these interdependencies across different cache levels. To achieve an efficient design, we first performed an *architectural design space exploration* (DSE), while considering multi-core processors with multiple cache levels executing different multi-threaded applications. Our cache DSE methodology identifies the pareto-optimal configurations with respect to constraints, performance overhead, and targeted vulnerabilities [45]. Afterwards, these configurations are used at run time to

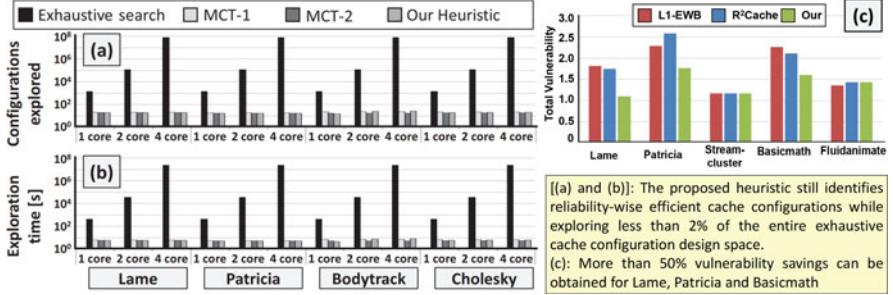


Fig. 7 (a) and (b) Exploration time saving achieved by the proposed approach with respect to exhaustive exploration, multi-level tuning approach (MCT-1) [47] and heuristic (MCT-2) [47]. (c) Vulnerability saving comparison of the proposed approach compared to non-reconfigurable baseline cache with L1 Early WriteBack (EWB) [15] and reliability-aware last-level cache partitioning (R^2 Cache) [22] schemes (adapted from [45])

perform *reliability-aware cache reconfiguration for the complete cache hierarchy*. Figure 7 shows that more than 50% vulnerability saving is achieved by the proposed solution as compared to non-reconfigurable baseline cache with L1 Early WriteBack (EWB) [15] and Reliability-Aware Last-Level Cache Partitioning (R^2 Cache) while exploring less than 2% of the entire exhaustive cache configuration design space.

3 Heterogeneous Reliability Modes of Out-of-Order Superscalar Cores

Embedded processors, although important in a wide range of applications and scenarios, cannot cater the high throughput and performance requirements of personal computers or high-performance computing platforms such as cloud servers or data-centers, which are also constrained in the amount of power that can be consumed. Such high-throughput systems deploy multi-core out-of-order (O3) superscalar processors, such as Intel Core i7 processors in PCs, and Intel Xeon or AMD Opteron processors in servers and data-centers worldwide. An O3 processor executes the instructions of a program *out-of-order*, instead of in-order as is the case in embedded processors (e.g., LEON3), to utilize the instruction cycles that would otherwise be wasted in pipeline stalls. A *superscalar* processor, on the other hand, implements instruction-level parallelism to execute more than one instruction in parallel by dispatching instructions to multiple different execution units embedded in the processor core. Therefore, an O3 superscalar processor offers a significantly higher throughput by combining the advantages of these two individual techniques. However, enabling such high throughput comes at the cost of implementing additional hardware units such as the Re-order Buffer (ROB), which keeps track of the instructions executing out-of-order.

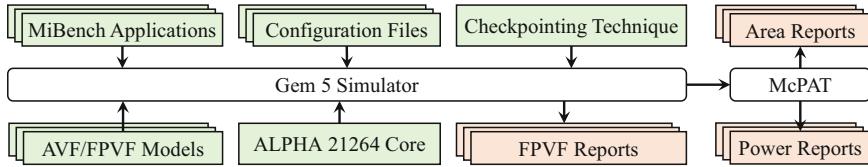


Fig. 8 Experimental tool-flow for vulnerability analysis of out-of-order superscalar *ALPHA* cores

In this section, we analyze the vulnerability of the ALPHA 21264 [17] O3 superscalar processor and design multiple reliability-heterogeneous processor cores from which an optimal configuration can be chosen at run-time based on the applications' reliability requirements.

3.1 Experimental Setup

Figure 8 presents an overview of the tool-flow used to obtain the results. We utilize a modified version of the *gem5 simulator* [5] extended to support the following functionality:

1. Determine the *vulnerable time* of all pipeline components, which in turn is used to compute their *Architectural Vulnerability Factors (AVFs)* [30],
2. *Full support for simulating reliability-heterogeneous cores* obtained by triplicating key pipeline components (instead of implementing full-scale TMR), and
3. *Checkpoint processor state compression* using techniques like DMTCP [3], HBICT [1], and GNU zip [8].

We evaluate our reliability-heterogeneous *ALPHA* 21264 four-issue superscalar processor cores using the *MiBench* application benchmark suite [12].

3.2 Vulnerability Analysis of Out-of-Order Superscalar Processors

The AVF of a component C over a period of N clock-cycles is defined as the probability of a fault that is generated in C to propagate to the final output resulting in an erroneous application output or intermittent termination of the program [30]. It is computed using the following equation:

$$AVF_C = \frac{\sum_{n=0}^{N} VulnerableBits}{TotalBits \times N} \quad (2)$$

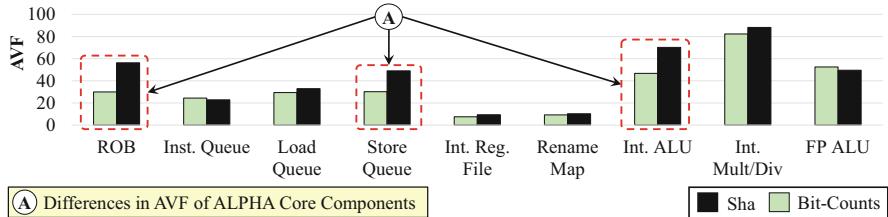


Fig. 9 Differences in AVF of *ALPHA* core components during application execution (*SHA* and *Bit-counts*) (adapted from [33])

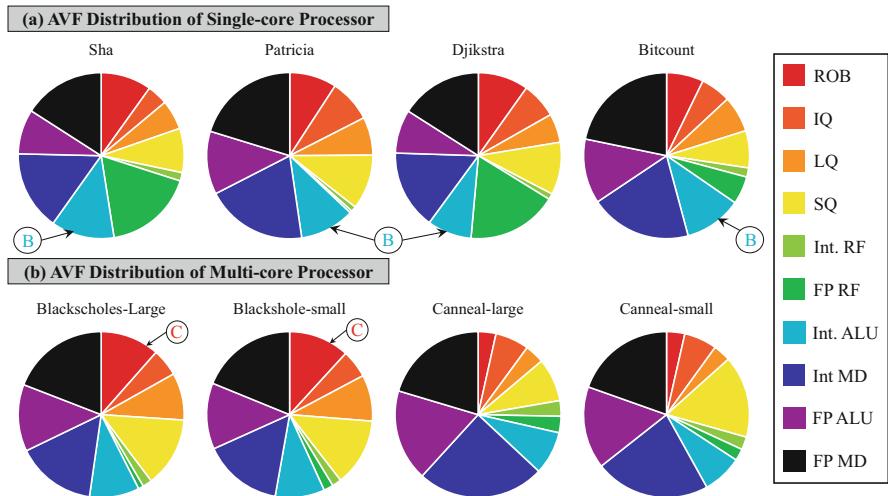


Fig. 10 Vulnerability analysis of *ALPHA* cores in single- and multi-core processors (adapted from [33])

The AVF of each pipeline component is estimated using applications from the *MiBench* and *PARSEC* application benchmark suites for single- and multi-core *ALPHA* 21264 superscalar processors for key pipeline components such as: (1) Re-order Buffer (ROB), (2) Instruction (IQ), (3) Load (LQ), (4) Store Queues (SQ), (5) Integer Register Files (Int. RF), (6) Floating Point Register Files (FP RF), (7) Rename Map (RM), (8) Integer ALUs (Int. ALU), (9) Floating Point ALUs (FP ALU), (10) Integer Multiply/Divide (Int. MD), and (11) Floating Point Multiply/Divide (FP MD). Figures 9 and 10 illustrate the results of the vulnerability analysis experiments for both the single-core and multi-core processors.

We analyze the results obtained from the vulnerability analysis to make the following **key observations**:

1. We have identified three key pipeline components (Integer ALU, Store Queue, and Re-order Buffer) that are more vulnerable during the execution of *SHA*, when compared to *Bit-counts*, as depicted by A in Fig. 9.

2. The AVFs of the individual pipeline components vary for different application workloads. For example, as shown in Fig. 10a the vulnerability of the Integer ALU widely varies for the four application workloads evaluated (labeled *B*).
3. In case of multi-core processors, the size of the input data does not significantly affect the AVF of the pipeline components, as shown by *C* in Fig. 10b.

The AVF of a component varies based on the type and number of instructions present in the application and its properties such as its compute- or memory-intensiveness, instruction-level parallelism, cache hit/miss rate, etc. For example, components like the ROB and the SQ are more vulnerable in *SHA* because of higher levels of instruction-level parallelism and more store instructions.

Therefore, based on this information, we can select certain key pipeline components that can be hardened/triplicated to increase the reliability of the processor for a given application workload. By hardening multiple key pipeline components in different combinations, we design a wide range of reliability-heterogeneous O3 superscalar *ALPHA* cores from which an optimal design configuration can be selected at run-time based on an application's reliability requirement while minimizing the area and/or power overheads.

3.3 Methodology for Hardening Out-of-Order Superscalar Processors

Our methodology for designing reliability-heterogeneous O3 superscalar processors targets two key approaches: (1) Redundancy, and (2) Checkpointing. Redundancy at the hardware layer is ensured by designing a wide range of reliability-heterogeneous processor cores by hardening a combination of the vulnerable pipeline components, depending on the reliability requirements of the target application. The vulnerable components are selected based on the fault-injection experiments and the AVF values of each component for different application workloads. Second, to further enhance processor reliability, we investigate and analyze various compression mechanisms that can be used to efficiently reduce the size of checkpointing data. An overview of our methodology for hardening O3 superscalar processors is presented in Fig. 11. First, we explain how we evaluate the vulnerability of the full processor for a given application workload.

3.3.1 Full-Processor Vulnerability Factor

For evaluating the vulnerability of the full processor for a given application workload, we propose to extend the AVF to estimate what we refer to as the **Full-Processor Vulnerability Factor (FPVF)**. It is defined as the ratio of the total number of vulnerable bits (*VulnerableBits*) in the processor pipeline for the duration they are vulnerable (*VulnerableTime*) to the total number of bits in the processor

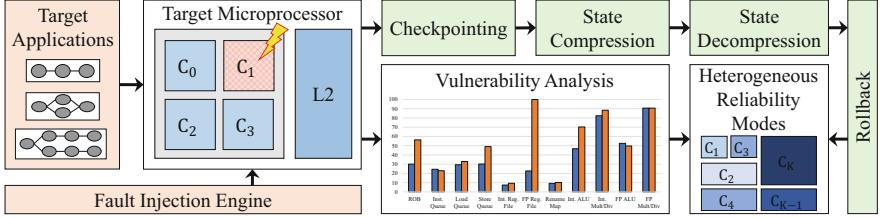


Fig. 11 Methodology for hardening out-of-order superscalar processors (adapted from [33])

pipeline (*TotalBits*) for the total duration of application execution (*TotalTime*). For a given application workload (*W*), we estimate FPVF of our proposed reliability-heterogeneous processors as:

$$FPVF_W = \frac{\sum_{\forall i \in Components} VulnerableBits_i \times VulnerableTime_i}{\sum_{\forall i \in Components} TotalBits_i \times TotalTime_i} \quad (3)$$

3.3.2 Heterogeneous Reliability Modes for ALPHA Cores

Enabling full-scale TMR for all application workloads leads to 200% (or more) area and power overheads, which might not be a feasible option in many real-world systems. Considering the analysis presented in Sect. 3.2, which illustrates that the AVF of the pipeline components varies based on the application workload, *we propose to enable fine-grained TMR at the component-level*. This involves hardening a combination of highly vulnerable pipeline components, instead of the full-processor pipeline to increase processor reliability while reducing the power and area overheads associated with TMR. Hardening involves instantiating three instances of the component with the same set of inputs and a voter circuit that is used to elect the majority output. We propose and analyze 10 different reliability modes (RM) for heterogeneous processors, including the baseline unprotected (U) core. The list of components hardened in these modes are presented in Table 1.

Next, we execute the four *MiBench* application benchmarks on our 10 proposed RMs to estimate the FPVF of each reliability-heterogeneous processor. We also evaluate the area and power overheads incurred by each reliability mode. The results of the experiments are illustrated in Fig. 12. From these results, we make the following **key observations**:

1. Our initial hypothesis, which stated that hardening different combinations of pipeline components (RMs) can reduce the vulnerability to different extents based on the application workload being executed, was correct. We demonstrate this further by considering the applications *SHA* and *Dijkstra*. Typically, the vulnerability of these two applications is similar to each other, except in the cases

Table 1 Heterogeneous reliability modes and corresponding pareto-optimal reliability modes for *MiBench* applications

Reliability mode	Components hardened	Application	Pareto-optimal reliability modes
U	Unprotected	Bit-counts	U, RM4, RM7
RM1	RF	Dijkstra	U, RM4, RM7, RM8
RM2	IQ, RM	Patricia	U, RM4, RM7
RM3	IQ, LQ, SQ	SHA	U, RM1, RM6, RM7, RM8
RM4	IQ, LQ, SQ, RM, ROB	All	U, RM4, RM7, RM8
RM5	RF, IQ, LQ, SQ		
RM6	RF, RM		
RM7	RF, RM, ROB		
RM8	RM, ROB		
RM9	RF, IQ, LQ, SQ, RM		

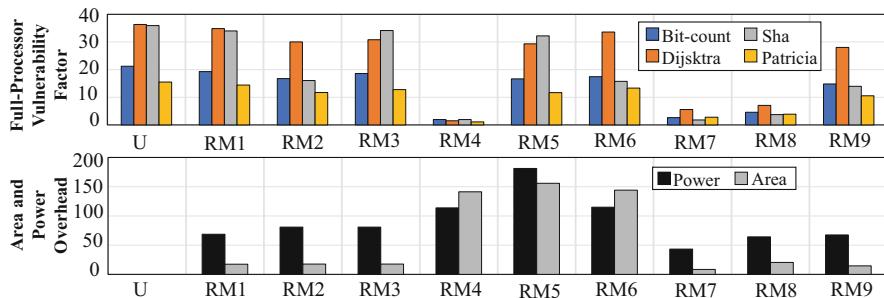


Fig. 12 Full-Processor Vulnerability Factor (FPVF) and power/area trade-off of the proposed heterogeneous reliability modes for different *MiBench* applications (adapted from [33])

of RM2, RM6, and RM9. The full-processor vulnerability of these three RMs has been reduced by more than 50% when executing *SHA* compared to *Dijkstra*.

2. Components such as the Rename Map and Reorder Buffer, when hardened, are highly effective in reducing the FPVF for all four applications. This is illustrated by the reliability modes RM4, RM7, and RM8, which have significantly lower FPVFs compared to their counter-parts. However, these two components occupy a significant percentage of the on-chip resources and hardening them leads to significant area and power overheads as illustrated by Fig. 12. This leads us to infer that hardening specific highly vulnerable pipeline components can significantly reduce the overall processor vulnerability for a wide range of application workloads based on their properties.

Furthermore, based on the data from these experiments, we perform an architectural space exploration that trades-off FPVF, area, and power overheads to extract the pareto-optimal reliability modes. The results of the experiments are illustrated in Fig. 13, where the x -, y -, and z -axes depict the FPVF, area, and power overheads, respectively. From these results, we make the following **key observations**:

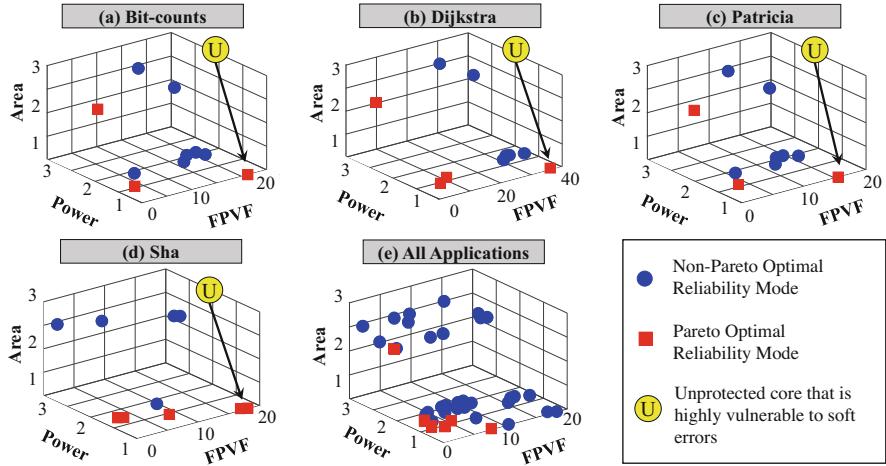


Fig. 13 Architectural space exploration of our heterogeneous reliability modes for *MiBench* applications (adapted from [33])

1. The design labeled *U*, i.e., the unprotected core, is pareto-optimal for all application workloads. This is expected as this reliability mode incurs zero area and power overheads and represents the least reliable processor design.
2. Although RM7 and RM8 significantly reduce the FPVF, due to their differences in power and area overheads, RM7 lies on the pareto-front for all individual application workloads, whereas RM8 is pareto-optimal only for *SHA* and *Dijkstra*. Similarly, RM4 is pareto-optimal for three of the four application workloads.
3. RM4, RM7, and RM8, all lie on the pareto-front when all applications are executed on the cores. This behavior is observed because of the varying levels of vulnerability savings achieved by the RMs when compared to their area and power overheads.
4. RM7 is pareto-optimal for four individual application workloads and reduces the FPVF by 87%, on average, while incurring area and power overheads of 10% and 43%, respectively.

3.3.3 State Compression Techniques

Reliability can also be improved at the software layer by inserting checkpoints in the application code. When an application encounters a checkpoint, the complete processor state, including all intermediate register and cache values, is stored in the main memory. These checkpoint states can be used to re-initialize the processor, which is referred to as rollback, in case a failure is detected and the next sequence of instructions are re-executed.

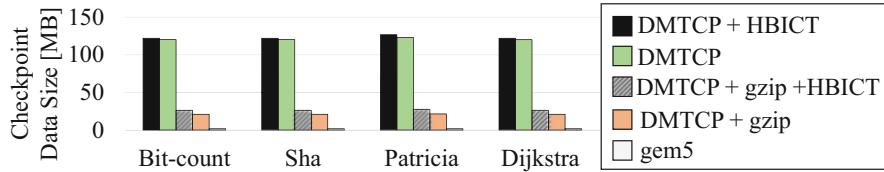


Fig. 14 Effectiveness of state compression techniques in reducing state size (adapted from [33])

The way checkpointing is implemented in *gem5* leads to significant loss in performance in case of frequent checkpoint restoration as the cache and pipeline states are not preserved, which, in turn, leads to a higher number of instructions being executed. *Distributed Multi-Threaded Checkpointing (DMTCP)* is a Linux compatible checkpointing tool that is used to checkpoint Linux processes. The back-end mechanism of DMTCP is accessible to programmers, via Application Programming Interfaces (APIs), to insert checkpoints into their application code. Inside *gem5*, these APIs can be used in combination with its pseudo-instructions to offer the functionality of creating/recovering checkpoint states for the applications being simulated inside *gem5*. Furthermore, the size of data generated by each checkpoint is typically large, especially in the case of O3 superscalar processors with large multi-level cache hierarchies. Therefore, we explore various compression strategies that can be used to efficiently compress and reduce the checkpoint data using techniques like the *Hash-Based Incremental Checkpointing Tool (HBICT)* and *GNU zip (gzip)*. HBICT provides DMTCP support to enable checkpoint compression using an approach called *delta compression*. This kind of compression mechanism preserves only changed fragments of a program's state, thereby considerably reducing the size of checkpoint data. gzip is a file compression technique based on the *DEFLATE algorithm*, which is a combination of lossless data compression techniques such as *LZ77* and *Huffman coding*. gzip can drastically reduce the size of checkpoint data, as illustrated by the results presented in Fig. 14. These techniques and compression algorithms are implemented in *gem5*, in different combinations, to reduce the size of checkpoint data for the four aforementioned *MiBench* applications, by executing them on an unprotected *ALPHA* processor. The effectiveness of different combinations of compression algorithms is illustrated in terms of checkpoint data size in Fig. 14. It can be observed that the combination of DMTCP and gzip is highly successful in reducing the checkpoint size by $\sim 6\times$. On the other hand, a combination of DMTCP, HBICT, and gzip techniques reduces the checkpoint size by $\sim 5.7\times$.

4 Run-Time Systems for Heterogeneous Fault-Tolerance

The techniques discussed in Sects. 2 and 3 also require a run-time manager for incorporating the application vulnerabilities with respect to several reliability threats

such as soft errors, aging, and process variation, as well as considering constraints like dark silicon and required performance (or tolerable performance overhead). Most of the adaptive hardware techniques exploit the application vulnerability to map applications on appropriate cores to reduce their vulnerability. Similarly, this concept can be applied to modify the applications with respect to the available hardened core or caches, which can also be combined with other hardware techniques to further reduce the vulnerabilities of the heterogeneous multi/many-core processors. Therefore, several techniques have been proposed to modify the execution patterns of the application or partitioning the application to develop a run-time system for reliability-heterogeneous multi/many-core processors.

1. **Aging- and Process Variation-Aware Redundant Multithreading [18, 36]:** *dTune* leverages multiple reliable versions of an application and redundant multithreading (RMT) simultaneously for achieving high soft error resilience under aging and process variability [36]. Based on the reliability requirements of the executing applications, *dTune* performs efficient core allocation for RMT while considering the aging state of the processor as well as process variation. It achieves up to 63% improvement in the reliability of a given application. Similarly, another approach [18] utilizes different software versions and RMT to improve the reliability of a system while considering the effects of soft errors and aging on the processor cores, to achieve an improved aging balancing.
2. **Variability-aware reliability-heterogeneous processor [21]:** This work leverages techniques at the hardware and run-time system layers to mitigate the reliability threats. In particular, this work focuses on TMR-based solutions to (partially) harden the cores for developing a many/multi-core reliability-heterogeneous processor. It uses a run-time controller to handle multiple cores with different reliability modes while considering the reliability requirements of the applications. In addition, it also exploits the dark silicon property in multi/many-core processors to offer a wide range of different performance-reliability trade-offs by over-provisioning the processor with reliability-heterogeneous cores.
3. **Aging-aware reliability-heterogeneous processor [10]:** This technique exploits the dark silicon property of the multi/many-core processors to design a run-time approach for balancing the application load to mitigate the reliability threats, i.e., temperature-dependent aging while also considering variability and current age of the cores in order to improve the overall system performance for a given lifetime constraint. The analysis shows that this run-time solution can improve the overall aging of the multi/many-core processor by 6 months to 5 years depending upon the provided design constraints and power overheads. Furthermore, this work also developed a fast aging evaluation methodology based on multi-granularity simulation epochs, as well as lightweight run-time techniques for temperature and aging estimation that can be used for an early estimation of temperature-dependent aging of multi/many-core processors.

There are other techniques which can exploit the functional and timing reliability in real-time systems to improve the application by generating the reliable application

versions or respective thread with different performance and reliability properties [38]. These reliable applications or respective thread can jointly be used with hardware techniques to improve the overall reliability of the multi/many-core heterogeneous processor. Another solution is to exploit the dynamic voltage and frequency scaling to generate the dynamic redundancy and voltage scaling with respect to the effects of process variations, application vulnerability, performance overhead, and design constraints [40]. This technique demonstrates up to 60% power reductions while improving the reliability significantly. Similarly, in addition to redundancy, multiple voltage-frequency levels are introduced while considering the effects of dark silicon in multi/many-core heterogeneous processor [39]. This technique also considers the effects of soft errors and process variations in their reliability management system that provides up to 19% improved reliability under different design constraints [35]. Most of the abovementioned approaches are focused on general purpose microprocessors; however, in application-specific instruction set processors (ASIPs), the hardware hardening and corresponding runtime software assisted recovery techniques can be used to improve the soft error vulnerabilities in ASIP-based multi/many-core systems. For example, dynamic core adaptation and application specificity can be exploited to generate a processor configuration which performs the error (caused by soft error) recovery for a particular application under the given area, power, and performance constraints [24–26]. Moreover, the baseline instruction set of the targeted ASIPs can also be modified or extended to enable the error recovery functionality [23].

5 Conclusion

This chapter discusses the building blocks of computing systems (both embedded and superscalar processors) with different heterogeneous fault-tolerant modes for the memory components like caches as well as for the in-order and out-of-order processor designs. We provide a comprehensive vulnerability analysis of different components, i.e., embedded and superscalar, processors and caches, considering the soft errors and aging issues. We also discuss the methodologies to improve the performance and power of such systems by exploiting these vulnerabilities. In addition, we briefly present that a reliability-aware compiler can be leveraged to comprehend software-level heterogeneous fault-tolerance by generating different reliable versions of the application with respective reliability and performance properties. Further details on reliability-driven compilation can be found in Chap. 5. Towards the end, we also analyze fault-tolerance techniques for application-specific instruction set processors (ASIPs).

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500—spp1500.itec.kit.edu). We would like to thank Arun Subramaniyan, Duo Sun and Segnon Jean Bruno Ahandagbe for their contributions to parts of the works cited in this chapter.

References

1. Agarwal, S., Garg, R., Gupta, M.S., Moreira, J.E.: Adaptive incremental checkpointing for massively parallel systems. In: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 277–286. ACM, New York (2004)
2. Agarwal, M., Paul, B.C., Zhang, M., Mitra, S.: Circuit failure prediction and its application to transistor aging. In: 25th IEEE VLSI Test Symposium (VTS'07), pp. 277–286. IEEE, Piscataway (2007)
3. Ansel, J., Arya, K., Cooperman, G.: DMTCP: transparent checkpointing for cluster computations and the desktop. In: 2009 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12. IEEE, Piscataway (2009)
4. Baumann, R.C.: Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Trans. Device Mater. Reliab.* **5**(3), 305–316 (2005)
5. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. *ACM SIGARCH Comput. Archit. News* **39**(2), 1–7 (2011)
6. Chen, C.L., Hsiao, M.: Error-correcting codes for semiconductor memory applications: a state-of-the-art review. *IBM J. Res. Dev.* **28**(2), 124–134 (1984)
7. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: 2011 38th Annual International Symposium on Computer Architecture (ISCA), pp. 365–376. IEEE, Piscataway (2011)
8. Gailly, J.: Gzip—the data compression program (1993)
9. Geist, A.: Supercomputing's monster in the closet. *IEEE Spectr.* **53**(3), 30–35 (2016)
10. Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., Henkel, J.: Hayat: harnessing dark silicon and variability for aging deceleration and balancing. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, June 7–11, pp. 180:1–180:6 (2015)
11. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. CAD Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
12. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: Mibench: A free, commercially representative embedded benchmark suite. In: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538), pp. 3–14. IEEE, Piscataway (2001)
13. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October, 2011, pp. 69–78 (2011). <https://doi.org/10.1145/2039370.2039384>
14. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: Proceedings of the 50th Annual Design Automation Conference, p. 99. ACM, New York (2013)
15. Jeyapaul, R., Srivastava, A.: Enabling energy efficient reliability in embedded systems through smart cache cleaning. *ACM Trans. Des. Autom. Electron. Syst.* **18**(4), 53 (2013)
16. Kang, K., Gangwal, S., Park, S.P., Roy, K.: NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution? In: Proceedings of the 2008 Asia and South Pacific Design Automation Conference, pp. 726–731. IEEE Computer Society Press, Silver Spring (2008)
17. Kessler, R.E.: The alpha 21264 microprocessor. *IEEE Micro* **19**(2), 24–36 (1999)
18. Kriebel, F., Rehman, S., Shafique, M., Henkel, J.: ageOpt-RMT: compiler-driven variation-aware aging optimization for redundant multithreading. In: Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, June 5–9, 2016, pp. 46:1–46:6 (2016). <https://doi.org/10.1145/2897937.2897980>

19. Kriebel, F., Rehman, S., Subramaniyan, A., Ahandagbe, S.J.B., Shafique, M., Henkel, J.: Reliability-aware adaptations for shared last-level caches in multi-cores. *ACM Trans. Embed. Comput. Syst.* **15**(4), 67:1–67:26 (2016). <https://doi.org/10.1145/2961059>
20. Kriebel, F., Rehman, S., Sun, D., Shafique, M., Henkel, J.: ASER: Adaptive soft error resilience for reliability-heterogeneous processors in the dark silicon era. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
21. Kriebel, F., Shafique, M., Rehman, S., Henkel, J., Garg, S.: Variability and reliability awareness in the age of dark silicon. *IEEE Des. Test* **33**(2), 59–67 (2016)
22. Kriebel, F., Subramaniyan, A., Rehman, S., Ahandagbe, S.J.B., Shafique, M., Henkel, J.: R²cache: reliability-aware reconfigurable last-level cache architecture for multi-cores. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015, Amsterdam, October 4–9, 2015, pp. 1–10 (2015)
23. Li, T., Shafique, M., Ambrose, J.A., Henkel, J., Parameswaran, S.: Fine-grained checkpoint recovery for application-specific instruction-set processors. *IEEE Trans. Comput.* **66**(4), 647–660 (2017)
24. Li, T., Shafique, M., Ambrose, J.A., Rehman, S., Henkel, J., Parameswaran, S.: RASTER: runtime adaptive spatial/temporal error resiliency for embedded processors. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–7. IEEE, Piscataway (2013)
25. Li, T., Shafique, M., Rehman, S., Ambrose, J.A., Henkel, J., Parameswaran, S.: DHASER: dynamic heterogeneous adaptation for soft-error resiliency in ASIP-based multi-core systems. In: 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 646–653. IEEE, Piscataway (2013)
26. Li, T., Shafique, M., Rehman, S., Radhakrishnan, S., Ragel, R., Ambrose, J.A., Henkel, J., Parameswaran, S.: CSER: HW/SW configurable soft-error resiliency for application specific instruction-set processors. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 707–712. EDA Consortium, San Jose (2013)
27. McPherson, J.W.: Reliability challenges for 45 nm and beyond. In: 2006 43rd ACM/IEEE Design Automation Conference, pp. 176–181. IEEE, Piscataway (2006)
28. Mitra, S., Seifert, N., Zhang, M., Shi, Q., Kim, K.S.: Robust system design with built-in soft-error resilience. *Computer* **38**(2), 43–52 (2005)
29. Mukherjee, S.S., Emer, J., Reinhardt, S.K.: The soft error problem: an architectural perspective. In: 11th International Symposium on High-Performance Computer Architecture, pp. 243–247. IEEE, Piscataway (2005)
30. Mukherjee, S.S., Weaver, C., Emer, J., Reinhardt, S.K., Austin, T.: A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36, pp. 29–40. IEEE, Piscataway (2003)
31. Pagani, S., Khdr, H., Munawar, W., Chen, J.J., Shafique, M., Li, M., Henkel, J.: TSP: thermal safe power: efficient power budgeting for many-core systems in dark silicon. In: Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, p. 10. ACM, New York (2014)
32. Portolan, M., Leveugle, R.: A highly flexible hardened RTL processor core based on LEON2. *IEEE Trans. Nucl. Sci.* **53**(4), 2069–2075 (2006)
33. Prabakaran, B.S., Dave, M., Kriebel, F., Rehman, S., Shafique, M.: Architectural-space exploration of heterogeneous reliability and checkpointing modes for out-of-order superscalar processors. *IEEE Access* **7**, 145324–145339 (2019)
34. Rehman, S., Kriebel, F., Prabakaran, B.S., Khalid, F., Shafique, M.: Hardware and software techniques for heterogeneous fault-tolerance. In: 24th IEEE International Symposium on On-Line Testing And Robust System Design, IOLTS 2018, Platja D’Aro, July 2–4, 2018, pp. 115–118 (2018). <https://doi.org/10.1109/IOLTS.2018.8474219>
35. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **33**(11), 1597–1610 (2014)

36. Rehman, S., Kriebel, F., Sun, D., Shafique, M., Henkel, J.: dTune: leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
37. Rehman, S., Shafique, M., Henkel, J.: Reliable Software for Unreliable Hardware: A Cross Layer Perspective. Springer, Berlin (2016)
38. Rehman, S., Toma, A., Kriebel, F., Shafique, M., Chen, J.J., Henkel, J.: Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 273–282. IEEE, Piscataway (2013)
39. Salehi, M., Shafique, M., Kriebel, F., Rehman, S., Tavana, M.K., Ejlali, A., Henkel, J.: dsReliM: power-constrained reliability management in Dark-Silicon many-core chips under process variations. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 75–82. IEEE, Piscataway (2015)
40. Salehi, M., Tavana, M.K., Rehman, S., Kriebel, F., Shafique, M., Ejlali, A., Henkel, J.: DRVS: power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 225–230. IEEE, Piscataway (2015)
41. Shafique, M., Garg, S., Henkel, J., Marculescu, D.: The EDA challenges in the dark silicon era: temperature, reliability, and variability perspectives. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
42. Shafique, M., Rehman, S., Aceituno, P.V., Henkel, J.: Exploiting program-level masking and error propagation for constrained reliability optimization. In: Proceedings of the 50th Annual Design Automation Conference, p. 17. ACM, New York (2013)
43. Shivakumar, P., Kistler, M., Keckler, S.W., Burger, D., Alvisi, L.: Modeling the effect of technology trends on the soft error rate of combinational logic. In: Proceedings International Conference on Dependable Systems and Networks, pp. 389–398. IEEE, Piscataway (2002)
44. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: The case for lifetime reliability-aware microprocessors. In: ACM SIGARCH Computer Architecture News, vol. 32, p. 276. IEEE Computer Society, Silver Spring (2004)
45. Subramanyan, A., Rehman, S., Shafique, M., Kumar, A., Henkel, J.: Soft error-aware architectural exploration for designing reliability adaptive cache hierarchies in multi-cores. In: Design, Automation and Test in Europe Conference and Exhibition, DATE 2017, Lausanne, March 27–31, 2017, pp. 37–42 (2017)
46. Vadlamani, R., Zhao, J., Burleson, W., Tessier, R.: Multicore soft error rate stabilization using adaptive dual modular redundancy. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 27–32. European Design and Automation Association (2010)
47. Wang, W., Mishra, P.: Dynamic reconfiguration of two-level cache hierarchy in real-time embedded systems. *J. Low Power Electron.* **7**(1), 17–28 (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Thermal Management and Communication Virtualization for Reliability Optimization in MPSoCs



Victor M. van Santen, Hussam Amrouch, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf

1 Overview

The VirTherm3D project is part of SPP1500, which has its origins in [10] and [9]. The main cross-layer contributions of VirTherm3D are outlined in Fig. 1. The green circles are our major contributions spanning from the physics to circuit layer and from the architecture to application layer. These contributions include physical modeling of thermal and aging effects considered at the circuit layer as well as communication virtualization at architecture level to support task relocation as part of thermal management at architecture level. Our minor contributions span from the circuit to architecture layer and include reliability-aware logic synthesis as well as studying the impact of reliability with figures of merit such as probability of failure.

2 Impact of Temperature on Reliability

Temperature is at the core of reliability. It has a direct short-term impact on reliability, as the electrical properties of circuits (e.g., delay) are affected by temperature. A higher temperature leads to circuits with higher delays and lower noise margins. Additionally, temperature impacts circuits indirectly as it stimulates or accelerates aging phenomena, which in turn, manifest themselves as degradations in the electrical properties of circuits.

The direct impact of temperature in an SRAM memory cell can be seen in Fig. 2. Increasing the temperature increases the read delay of the memory cell.

V.M. van Santen · H. Amrouch (✉) · T. Wild · J. Henkel · A. Herkersdorf
Karlsruhe Institute of Technology, Technical University of Munich, Munich, Germany
e-mail: victor.santen@kit.edu; amrouch@kit.edu; thomas.wild@tum.de

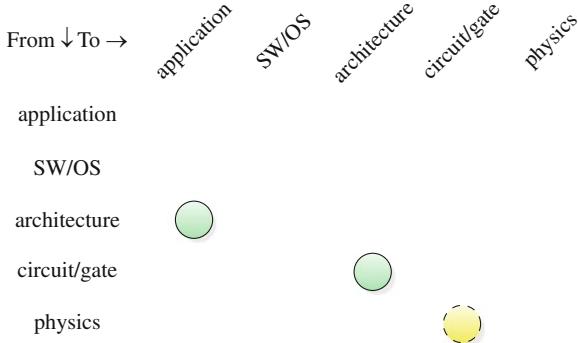


Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

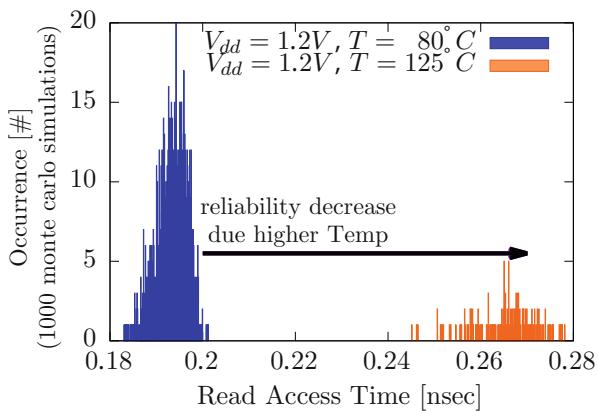


Fig. 2 Shift in SRAM memory cell read delay as a direct impact of temperature. Taken (from [3])

This is because increased temperature degrades performance of transistors (e.g., a reduction in carrier mobility μ), which affects the performance of the memory cell. Therefore, increasing temperature directly worsens circuit performance and thus negatively impacts the reliability of a circuit. If the circuit has a prolonged delay due to the increased temperature, then timing violations might occur. If the circuit has a degraded noise margin, then noise (e.g., voltage drops or radiation-induced current spikes) might corrupt data.

Next to directly altering the circuit properties, temperature also has an indirect impact, which is shown in Fig. 3. Temperature stimulates aging phenomena (e.g., Bias Temperature Instability (BTI)) degrading the performance of transistors (e.g., increasing the threshold voltage V_{th}) over time. Increasing the temperature accelerates the underlying physical processes of aging and thus increases aging-induced degradations.

Because of the two-fold impact of temperature, i.e., by reducing circuit performance directly and indirectly via aging, it is crucial to be considered when

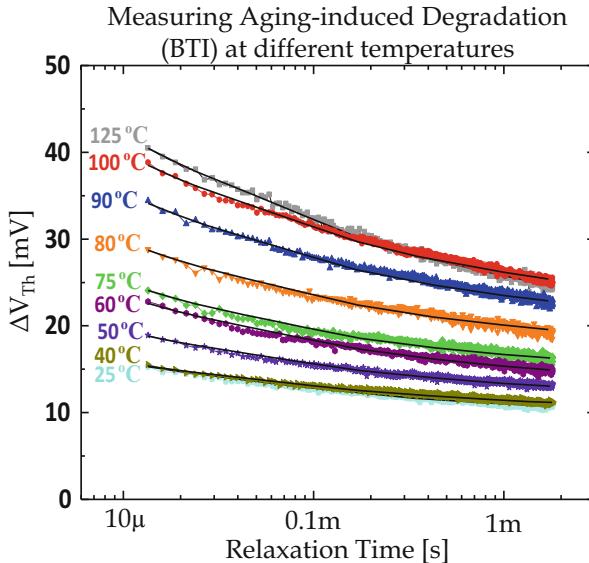


Fig. 3 Indirect impact of temperature stimulating aging. Taken (from [4])

estimating the reliability of a circuit. The temperature at an instant of time (estimated either via measurement or simulation) governs the direct degradation of the circuit, i.e., the short-term direct impact of temperature. Temperature over time governs the long-term indirect impact, as aging depends on the thermal profile (i.e., the thermal fluctuations over a long period). How to estimate temperature correctly both the temperature at an instant as well as the thermal profile is discussed in Sect. 3.

After the temperature is determined via temperature estimation, the impact of temperature on reliability must be evaluated. This is challenging, as the impact of temperature occurs on physical level (e.g., movement of electrical carriers in a semiconductor as well as defects in transistors for aging), while the figures of merit are for entire computing systems (e.g., probability of failure, quality of service). To overcome this challenge, Sect. 4 discusses how to connect the physical to the system level with respect to thermal modeling. To obtain the ultimate impact of the temperature, the figures of merit of a computing system are obtained with our cross-layer (from physical to system level) temperature modeling (see Fig. 4).

Temperature can be controlled. Thermal management techniques reduce temperature by limiting the amount of generated heat or making better use of existing cooling (e.g., distribution of generated heat for easier cooling). Thus, to reduce the deleterious impact of temperature on the figures of merit of systems, temperature must be controlled at system level. For this purpose, Sect. 5 discusses system-level thermal management techniques. These techniques limit temperature below a specified critical temperature to ensure that employed safety margins (e.g., are not violated time slack to tolerate thermally induced delay increases), thus ensuring the reliability of a computing system.

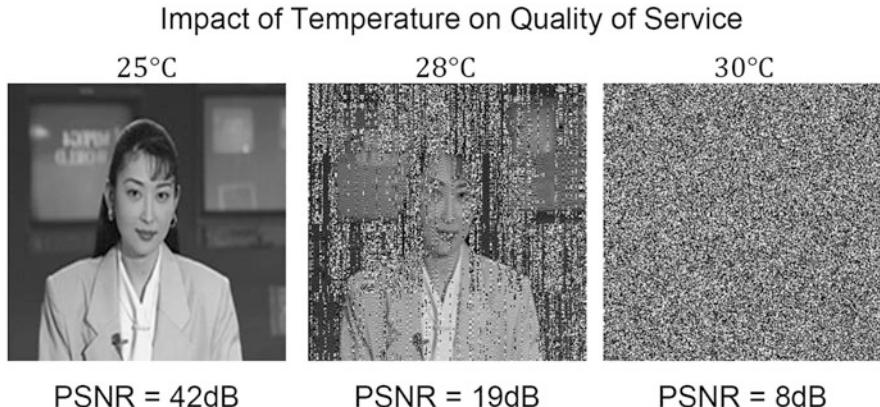


Fig. 4 Image signal-to-noise ratio as a figure of merit for an image processing system. At 25 °C no timing violations occur, while (due to non-existent safety margins) at 28 and 30 °C timing violations degrade PSNR

To support the system management in migrating tasks away from thermal hotspots and thus reducing thermal stress, special virtualization features are proposed to be implemented in the interconnect infrastructure. They allow for a fast transfer of communication relations of tasks to be migrated and thus help to limit downtimes. These mechanisms can then also be applied for generating task replica to dynamically introduce redundancy during system runtime as a response to imminent reliability concerns in parts of the SoC or if reliability requirements of an application change.

As mentioned before, temperature estimation and modeling cross many abstraction layers. The effects of temperature originate from the physical level, where the physical processes related to carriers and defects are altered by temperature. Yet the final impact of temperature has to pass through the transistor level, gate level, circuit level, architecture level all the way to the system level, where the figures of merit of the system can be evaluated. The system designer has to maintain the figures of merit for his end-user, therefore limiting temperature with thermal management techniques and evaluating the impact of temperature on the various abstraction layers. Therefore, Sect. 7 discusses thermal estimation, modeling, and management techniques with a focus on how to cross these abstraction layers and how to connect the physical to the system level. In practice, interdependencies between the low abstraction layers and the management layer do exist. The running workload at the system level increases the temperature of the cores. Hence, the probability of error starts to gradually increase. In such a case the management layer estimates the probability of error based on the information received from the lower layers and then attempts to make the best decision. For instance, it might allow the increase in the probability of error but at the cost of enabling the adaptive modular redundancy (AMR) (details in Sect. 6.3) or maybe migrating the tasks to other cores that are healthier (i.e., exhibit less probability of error).

3 Temperature Estimation via Simulation or Measurement

Accurately estimating the temperature of a computing system is necessary to later evaluate the impact of temperature. Two options exist: (1) Thermal simulation and (2) Thermal measurement. Both options must estimate temperature with respect to time and space. Figure 5 shows a simulated temporal thermal profile of a microprocessor. Temperature fluctuates visibly over time and depends on the applications which are run on the microprocessor.

Figure 9 shows a measured spatial thermal map of a microprocessor. Temperature is spatially unequally distributed across the processor, i.e., certain components of the microprocessors have to tolerate higher temperatures. However, the difference in temperature is limited. This limit stems from thermal conductance across the chip counteracting temperature differences. Thermal conductance is mainly via the chip itself (e.g., wires in metal layers), its packaging (e.g., heat spreaders), and cooling (e.g., heat sink).

3.1 Thermal Simulation

Thermal simulations are a software-based approach to estimate the temperature of a computing system. Thermal simulations consist of three steps: (1) Activity extraction, (2) Power estimation, (3) Temperature estimation. The first step extracts the activity (e.g., transistor switching frequency, cache accesses) of the applications running on the computing system. Different applications result in different temperatures (see Fig. 5). The underlying cause is a unique power profile for each application (and its input data), originating from unique activities per application.

Once activities are extracted, the power profiles based on these activities are estimated. Both steps can be performed on different abstraction layers. On the transistor level, transistor switching consumes power, while on the architecture level

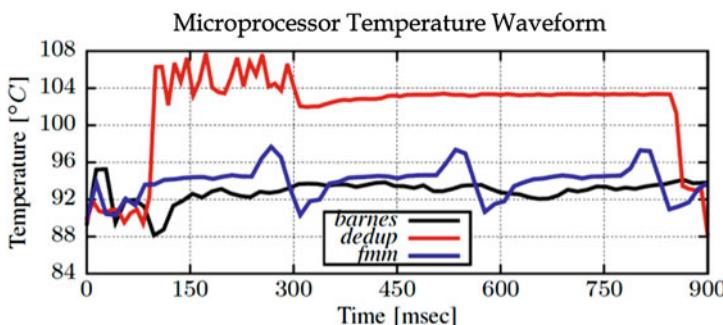


Fig. 5 Simulated thermal profile (temporal) of a microprocessor

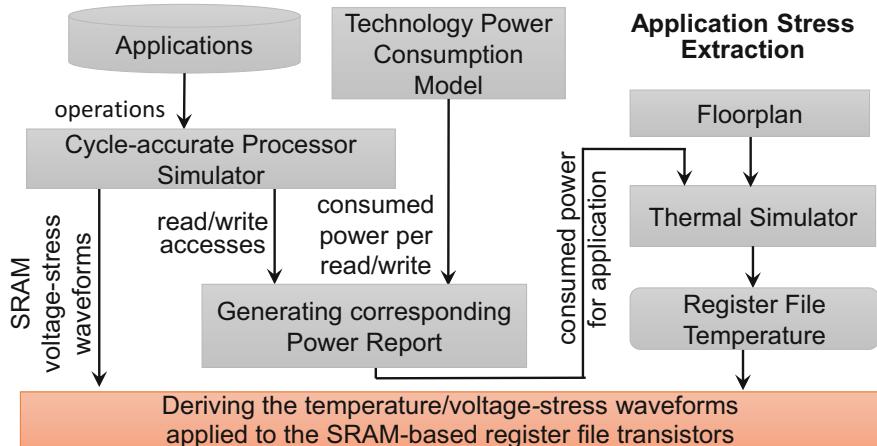


Fig. 6 Flow of a thermal simulation (updated figure from [4]).

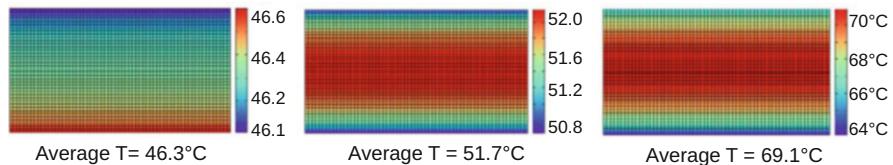


Fig. 7 Thermal map of an SRAM array (granularity: single SRAM cells) under different applications

each cache access consumes a certain amount of power (depending on cache hit or cache miss). Thus activity would be transistor switching/cache accesses and this would result in a very fine-grained power profile (temporally as well as spatially) for the transistor level. At the architecture level, a coarse-grained power profile is obtained with a time granularity per access (potentially hundreds of cycles long) and space granularity is per entire cache block.

With the power profiles known, the amount of generated heat (again spatially and temporally) is known. A thermal simulator then uses a representation of thermal conductances and capacitances with generated heat as an input heat flux and dissipated heat (via cooling) as an output heat flux to determine the temperature over time and across the circuit.

Our work in [4] exemplifies a thermal simulation flow in Fig. 6. In this example, SRAM memory cell accesses are used to estimate transistor switching and thus power profiles for the entire SRAM array. These power profiles are then used with the microprocessor layout (called floorplan) and typical cooling settings in a thermal simulator to get thermal maps in Fig. 7.

The work in [13] models temperature on the system level. Individual processor cores of a many-core computing system are the spatial granularity with seconds as

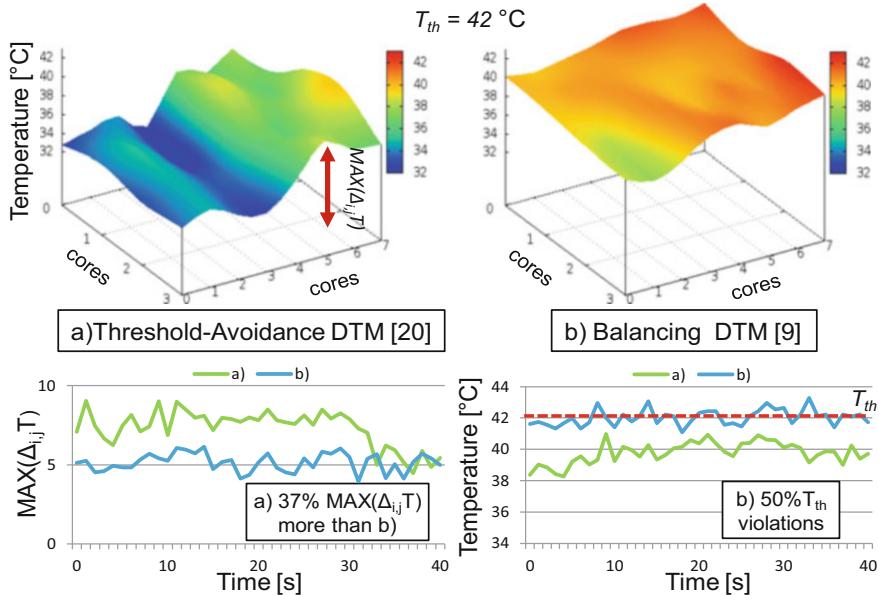


Fig. 8 Thermal estimation at the system level with cores as the spatial granularity (from [13])

the temporal granularity. Abstracted (faster, simpler) models are used to estimate the temperature per processor core, as a transistor level granularity would be unfeasible with respect to computational effort (i.e., simulation time).

While thermal simulations have the advantage of being able to perform thermal estimations without physical access to the system (e.g., during early design phases), they are very slow (hours of simulation per second of operation) and not accurate. Estimating activities and power on fine-grained granularities is an almost impossible task (layout-dependent parasitic resistances and capacitances, billions of transistors, billions of operations per second), while coarse-grained granularities provide just rough estimates of temperature due to the disregard of non-negligible details (e.g., parasitics) at these high abstraction levels (Fig. 8).

3.2 Thermal Measurement

If physical access to actual chips is an option, then thermal measurement is preferable. Observing the actual thermal profiles (temporally) and thermal maps (spatially) intrinsically includes all details (e.g., parasitics, billions of transistors, layout). Thus, a measurement can be more accurate than a simulation. Equally as important, measurements operate in real time (i.e., a second measured is also a second operated) outperforming simulations.

The challenge of thermal measurements is the resolution. The sample frequency of the measurement setup determines the temporal resolution and this is typically

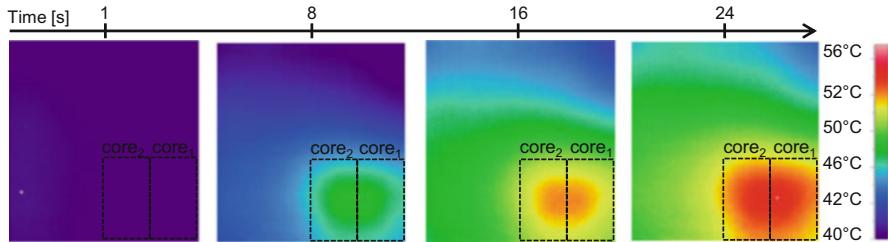


Fig. 9 Measured spatial thermal map of a microprocessor (from [2])

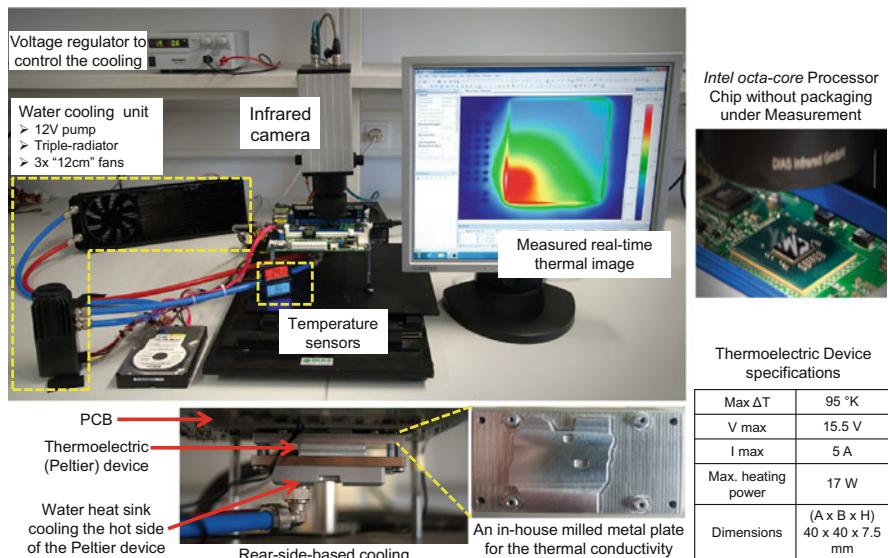


Fig. 10 High fidelity infrared thermal measurement setup (from [2])

in the order of milliseconds, while simulations can provide nano-second granularity (e.g., individual transistor switching). However, since thermal capacitances prevent abrupt changes of temperature as a reaction to abrupt changes in generated heat, sample rates in milliseconds are sufficient. The spatial resolution is equally limited by thermal conductance, which limits the thermal gradient (i.e., difference in temperature between two neighboring component; see Figs. 7 and 9).

The actual obstacle for thermal measurements is accessibility. A chip sits below a heat spreader and cooling, i.e., it is not directly observable. The manufacturers include thermal diodes at a handful of locations (e.g., 1 per core), which measure temperature in-situ, but these diodes are both inaccurate (due to their spatial separation from the actual logic) and spatially very coarse due to their limited number.

Our approach (Fig. 10) [2, 14] is to cool the chip through the PCB from the bottom-side and measure the infrared radiation emitted from the chip directly. Other

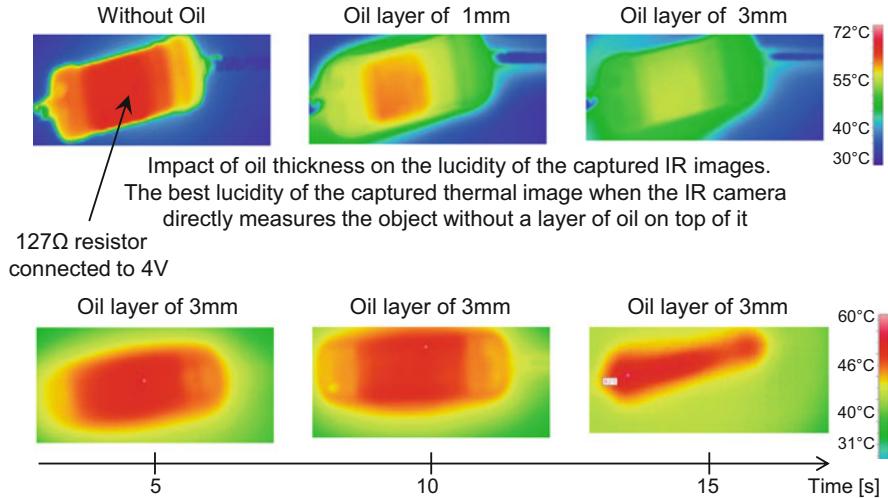


Fig. 11 Limited infrared image lucidity and fidelity due to oil cooling on top of the chip (from [2])

approaches cool the chip with infrared-transparent oils to cool the chip from the top, but this results in heat conductance limiting image fidelity and turbulence in the oil limiting image resolution (see Fig. 11). Our approach does not suffer from these issues and delivers crisp high-resolution infrared images from a camera capable of sampling an image every 20 ms with a spatial resolution of 50 μm . Thus a lucid thermal profile and thermal map are achieved including all implementation details of the chip, as actual hardware is measured.

4 Modeling Impact of Temperature at System Level

Modeling the impact of temperature on a computing system is a challenging task. Estimation of the figures of merit of a computing system can only be performed on the system level, while the effects of temperature are on the physical level. Thus, many abstraction layers have to be crossed while maintaining accuracy and computational feasibility (i.e., keep simulation times at bay). In this section we discuss how we tackle this challenge, starting with the selection of figures of merit, followed by the modeling of the direct impact of temperature and finally aging as the indirect impact of temperature.

4.1 Figures of Merit

The main figures of merit at the system level with respect to reliability are probability of failure P_{fail} and quality of service (e.g., PSNR in image processing). Probability of failure encompasses many failure types like timing violations, data corruption and catastrophic failure of a component (e.g., short-circuit). A full overview of abstraction of failures towards probability of failure is given in the RAP (Resilience Articulation Point) chapter of this book. Typically, vendors or end-users require the system designer to meet specific P_{fail} criteria (e.g., $P_{fail} < 0.01$). Quality of service describes how well a system provides its functionality if a specific amount of errors can be tolerated (e.g., if human perception is involved or for classification problems).

For probability of failure, the individual failure types have to be estimated and quantified without over-estimation due to common failures (as in our work [3], where a circuit with timing violations might also corrupt data). In that work the failure types such as timing violations, data corruption due to voltage noise, and data corruption due to strikes of high-energy particles are covered. These are the main causes of failure in digital logic circuits as a result of temperature changes (e.g., excluding mechanical stress from drops). The probability of failure is spatially and temporarily distributed (see Figs. 12 and 13) and therefore has to be estimated for a given system lifetime (temporally) and for total system failure (combined impact of spatially distributed P_{fail} (e.g., sum of failures or probability that only 1 component out of 3 fail (modular redundancy)).

Quality of service means observing the final output of the computing system and analyzing it. In our work [5, 7] we use the peak signal-to-noise ratio (PSNR) of an output image from an image processing circuit (discrete cosine transformation (DCT) in a JPEG encoder).

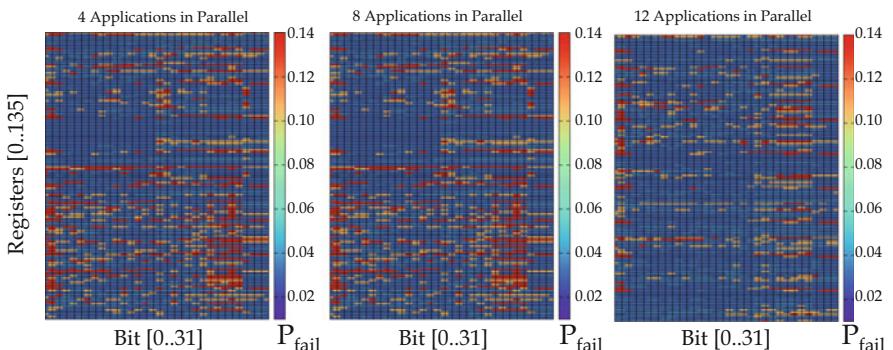


Fig. 12 Spatial distribution of P_{fail} across an SRAM array under two different applications (from [4])

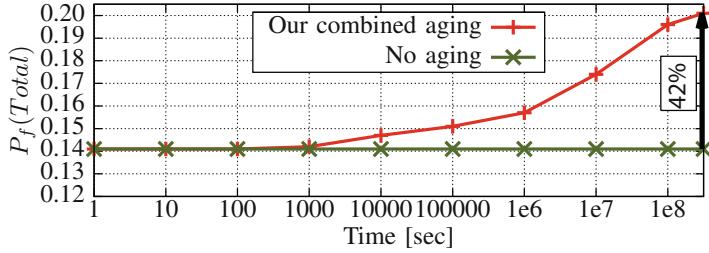


Fig. 13 Temporal distribution of P_{fail} , which increases over time due to aging (from [3])

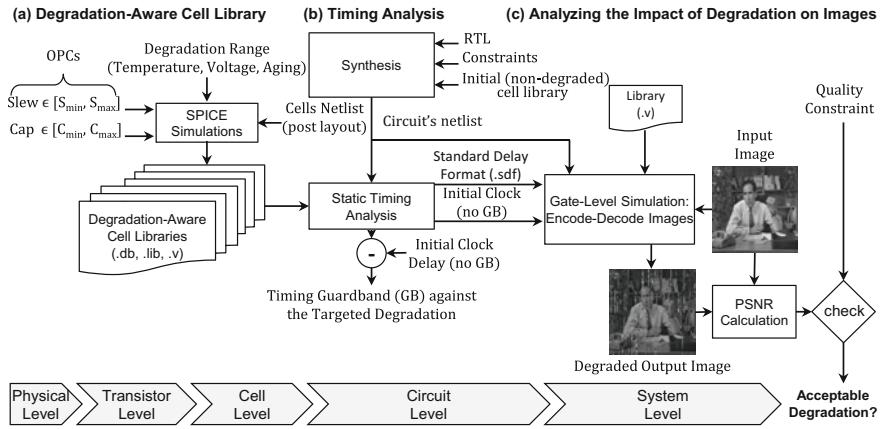


Fig. 14 Flow of the characterization of standard cells and the subsequent use of degradation-aware cell libraries to obtain timing violations (from [12])

4.2 Direct Impact of Temperature

To model the direct impact of temperature, we start at the lowest abstraction layers. Compact transistor models (e.g., BSIM) describe the current flow through the channel of a transistor and the impact of temperature on that current flow. These models are then used in circuit simulators to characterize standard cells (build from transistors) in terms of power consumption and propagation delay [5, 21]. Characterizing the standard cells (see Fig. 14) under different temperatures (e.g., from 25 to 125 °C) captures the impact of temperature on the delay and power consumption of these cells. This information is then gathered in a cell library (a single file containing all delay and power information for these cells) and then circuit and architecture level tools (e.g., static timing analysis tools, gate level simulators) can be used to check individual failure types (e.g., timing violations in static timing analysis) for computing systems (e.g., microprocessors) under various temperatures.

4.3 Aging as Indirect Impact of Temperature

Aging is stimulated by temperature (see Fig. 3) and therefore temperature has an indirect impact on reliability via aging-induced degradations. Aging lowers the resiliency of circuits and systems, thus decreasing reliability (an increase in P_{fail}) as shown in Fig. 15.

For this purpose our work [6, 18, 20] models aging, i.e., Bias Temperature Instability (BTI), Hot-Carrier Injection (HCI), Time-Dependent Dielectric Breakdown (TDDB) and the effects directly linked to aging like Random Telegraph Noise (RTN). All these phenomena are modeled with physics-based models [18, 20], which can accurately describe their temperature dependencies in the actual physical processes (typically capture and emission of carriers in the defects in the gate dielectric of transistors [4, 19]) of these phenomena.

Our work [6, 18] considers the interdependencies between these phenomena (see Fig. 16) and then estimates the degradation of the transistors. Then the transistor modelcards (transistor parameter lists) are adapted to incorporate the estimated degradations and use these degraded transistor parameters in standard cell characterization.

During cell characterization it is important to not abstract, as ignoring the interactions between transistors (counteracting each other when switching) results in underestimations of propagation delay [21] and ignoring the operating conditions

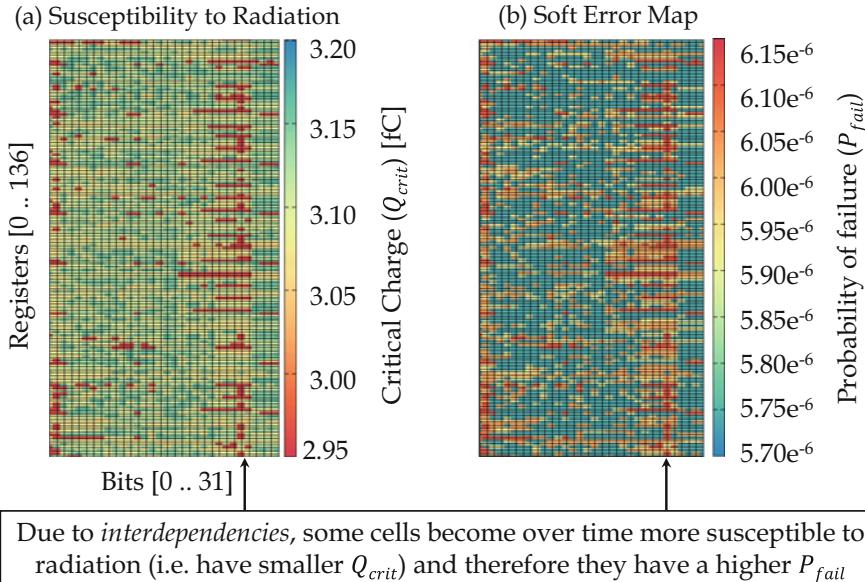


Fig. 15 Link between aging (increasing susceptibility) and P_{fail} (from [6])

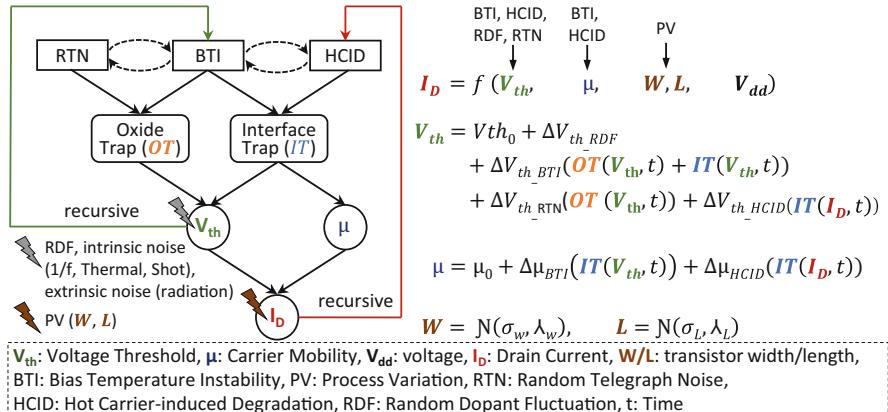


Fig. 16 Interdependencies between the aging phenomena (from [6])

(load capacitance, signal slew) of the cells [5, 20] misrepresents actual cell delay and power consumption.

After all necessary information is gathered, cells are characterized under different temperatures (like in the previous subsection) but not only with altered transistor currents (modeling the direct impact of temperature) but with additionally degraded transistors parameters (modeling the indirect impact of temperature via aging). Thus we combine both the direct and indirect impact into a single standard cell characterization to obtain delay and power information of standard cells under the joint impact of temperature and temperature-stimulated aging.

5 System-Level Management

To limit the peak temperature of a computing system and distribute the temperature evenly, we can employ system-level thermal management techniques. These techniques limit or distribute the amount of generated heat and thus ensure that the temperature stays below a given critical temperature. The two techniques presented in this section are task migration [13] and voltage scaling [17].

5.1 Voltage Scaling

Voltage scaling reduces the supply voltage of a chip or component (e.g., a processor core) to lower the power consumption and thus lower the generated heat. As a first-order approximation, lowering the voltage results in a quadratic reduction of

the consumed (dynamic) power. Therefore lowering the voltage even slightly has a considerable impact on the generated heat and thus exhibited temperature.

Voltage scaling has various side-effects. As the driving strength of transistors is also reduced, when the supply voltage is reduced, voltage scaling always prolongs circuit delays. Hence, voltage scaling has a performance overhead, which has to be minimized, while at the same time the critical temperature should not be exceeded.

Another side-effect is that voltage governs the electric field, which also stimulates aging [17]. When voltage increases, aging-induced degradation increases and when voltage reduces aging recovers (decreasing degradation). In our work in [17] we showed that voltage changes within a micro-second might induce transient timing violations. During such ultra-fast voltage changes, the low resiliency of the circuit (at the lower supply voltage) meets the high degradation of aging (exhibiting from operation at the high voltage). This combination of high degradation with low resiliency leads to timing violations if not accounted for. Continuing operation at the lower voltage recovers aging, thus resolving the issue. However, during the brief moment of high degradation violations occurred.

5.2 Task Migration

Task migration is the process of moving applications from one processor core to another. This allows a hot processing core to cool down, while a colder processor core takes over the computation of the task. Therefore, temperature is more equally distributed across a multi- or many-core computing system.

A flow of our task migration approach is shown in Fig. 17. Sensors in each core measure the current temperature (typically thermal diodes). As soon as the temperature approaches the critical value, then a task is migrated to a different core. The entire challenge is in the question “To which core is the task migrated?” If the core to which the task is migrated is only barely below the critical temperature, then the task is migrated again, which is costly since each migration stalls the processor core for many cycles (caches are filled, data has to be fetched, etc.).

Therefore our work in [13] predicts the thermal profile and makes decisions based on these predictions to optimize the task migration with as little migrations as possible while still ensuring that the critical temperature is not exceeded.

Another objective which has to be managed by our thermal management technique is to minimize thermal cycling. Each time a processor core cools down and heats up again it experiences a thermal cycle. Materials shrink and expand under temperature and thus thermal cycles put stress on bonding wires as well as soldering joints between the chip and the PCB or even interconnects within the chip (when it is partially cooled/heated).

Therefore, our approach is a multi-objective optimization strategy, which minimizes thermal cycles per core, limits temperature below the critical temperature and minimizes the number of task migrations (reducing performance overheads).

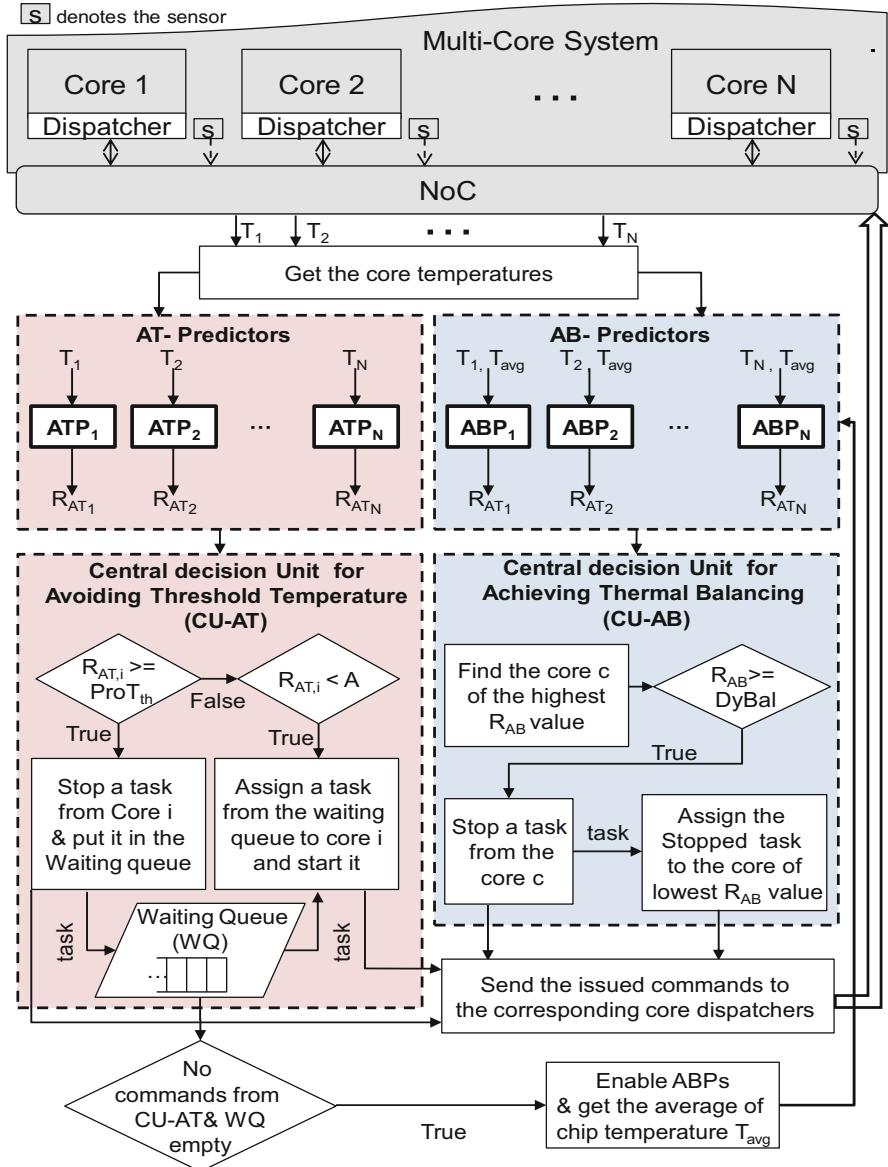


Fig. 17 Flow of our task migration approach to bound temperature in a many-core computing system (from [13])

6 Architecture Support

To support the system-level thermal management in the migration of tasks that communicate with each other the underlying hardware architecture provides specific assistance functions in the communication infrastructure. This encompasses a virtualization layer in the network on chip (NoC) and the application of protection switching mechanisms for a fast switch-over of communication channels. Based on these features an additional redundancy mechanism—called adaptive modular redundancy (AMR)—is introduced, which allows to run tasks temporarily with a second or third replica to either detect or correct errors.

6.1 NoC Virtualization

To support the system management layer in the transparent migration of tasks between processor cores within the MPSoC an interconnect virtualization overlay is introduced, which decouples physical and logical endpoints of communication channels. Any message passing communication among sub-tasks of an application or with the I/O tile is then done via logical communication endpoints. That is, a sending task transmits its data from the logical endpoint on the source side of the channel via the NoC to the logical endpoint at the destination side where the receiving task is executed. Therefore, the application only communicates on the logical layer and does not have to care about the actual physical location of sender and receiver tasks within the MPSoC. This property allows dynamic remapping of a logical to a different physical endpoint within the NoC and thus eases the transparent migration of tasks by the system management. This is shown in Fig. 18, where a communication channel from the MAC interface to task T1 can be transparently

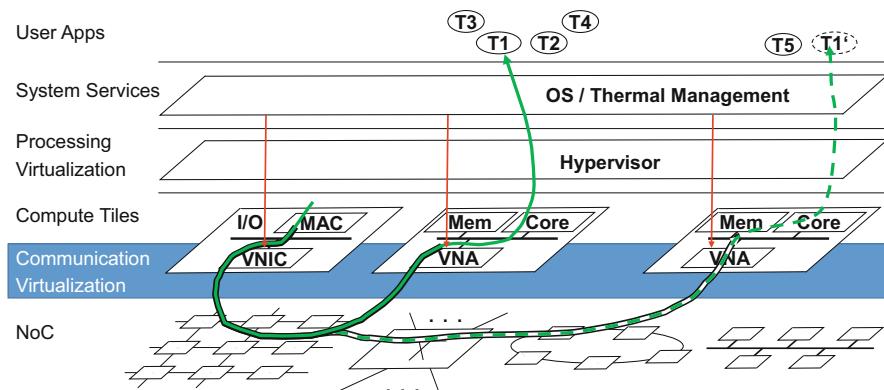


Fig. 18 Communication virtualization layer (from [8])

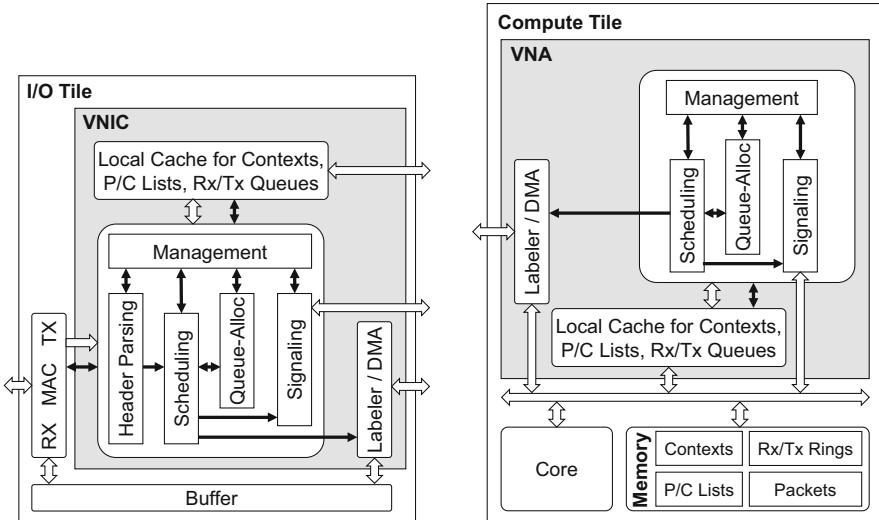


Fig. 19 VNIC and VNA architectures

switched over to a different receiving compute tile/processor core. Depending on the migration target of T1 the incoming data will be sent to the tile executing the new T1' [8]. In Sect. 6.2 specific protocols are described to reduce downtime of tasks during migration.

To implement this helper function, both a virtualized NoC adapter (VNA) and a virtualized network interface controller (VNIC) are introduced that can be reconfigured in terms of logical communication endpoints when a task migration has to be performed [15].

VNA and VNIC target a compromise between high throughput and support for mixed-criticality application scenarios with high priority and best effort communication channels [11]. Both are based on a set of communicating finite state machines (FSMs) dedicated to specific sub-functions to cope with these requirements, as can be seen in Fig. 19. The partitioning into different FSMs enables the parallel processing of concurrent transactions in a pipelined manner.

6.2 Advanced Communication Reconfiguration Using Protection Switching

The common, straight-forward method for task relocation is Stop and Resume: Here, first the incoming channels of the task to be migrated are suspended, then channel state together with the task state are transferred, before the channels and task are resumed at the destination. The key disadvantage is a long downtime. Therefore, an advanced communication reconfiguration using protection switching in NoCs to

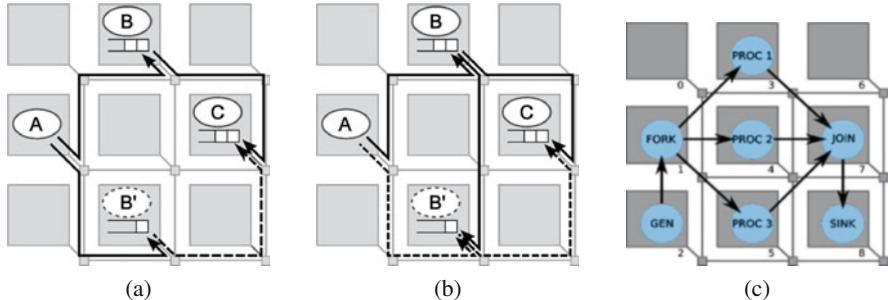


Fig. 20 Variants for communication migration. (a) Dualcast. (b) Forwarding. (c) Example migration scenario

support reliable task migration is proposed [16, 22, 23], which is inspired by protection switching mechanisms from wide area transport networks. Two alternatives to migrate communication relations of a relocated task are *dualcast* and *forwarding* as shown in Fig. 20 for a migration of task B to a different location executing B'. The procedure is to first establish an additional channel to the compute tile where the task is migrated to (location of B'). This can be either done from A being the source of the channel (*dualcast*, Fig. 20a) or from B the original location of the migrated task (*forwarding*, Fig. 20b). Then it has to be ensured that the buffers at the source and destination tiles of the migration are consistent. Finally, a seamless switch-over (task and channels) takes place from the original source to the destination. This shall avoid time-costly buffer copy and channel suspend/resume operations with a focus on low-latency and reliable adaptions in the communication layer.

The different variants have been evaluated for an example migration scenario as depicted in Fig. 20c: In a processing chain consisting of 7 tasks in total, the FORK task, which receives data from a generator task and sends data to three parallel processing tasks, is migrated to tile number 0. Figure 21 shows the latencies of the depicted execution chain during the migration, which starts at $2.5 \cdot 10^6$ cycles assuming FORK is stateless. The results have been measured using an RTL implementation of the MPSoC [16]. In Fig. 21a the situation is captured for a pure software-based implementation of the migration, whereas Fig. 21b shows the situation when all functions related to handling the migration are offloaded from the processor core. In this case task execution is not inhibited by any migration overhead, which corresponds to the situation when the VNA performs the associated functionality in hardware.

As can be seen from Fig. 21b, offloading migration protocols helps to reduce application processing latency significantly for all three variants. The dualcast and forwarding variants enable a nearly unnoticeable migration of the tasks. However, the investigations in [16] show that when migrating tasks with state, the handling of the task migration itself becomes the dominant factor in the migration delays and outweighs the benefits of the advanced switching techniques.

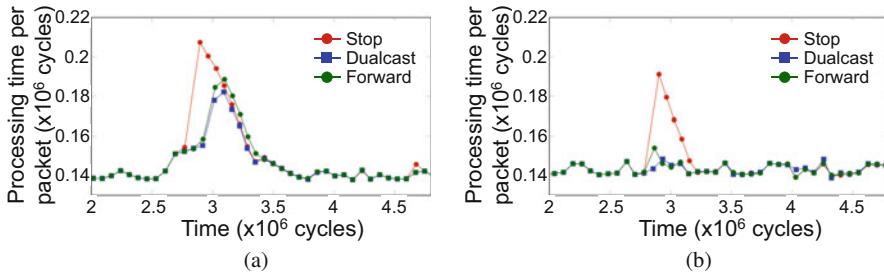


Fig. 21 Results for task migration scenarios (from [16]). **(a)** Relocation without offload. **(b)** Relocation with offload

6.3 Adaptive Modular Redundancy (AMR)

Adaptive modular redundancy (AMR) enables the dynamic establishment of a redundancy mechanism (dual or triple modular redundancy, DMR/TMR) at runtime for tasks that have a degree of criticality that may vary over time or if the operating conditions of the platform have deteriorated so that the probability of errors is too high. DMR will be used if re-execution is affordable, otherwise TMR can be applied, e.g., in case realtime requirements could not be met. AMR functionality builds upon the aforementioned services of the NoC. To establish DMR the dualcast mechanism is used and the newly established task acts as replica instead taking over the processing as in the case of migration. (For TMR two replica are established and triple-cast is applied.) Based on the running task replica, the standard mechanisms for error checking/correction and task re-execution if required are applied.

The decision to execute one or two additional replica of tasks is possibly taken as a consequence of an already impaired system reliability. On the one hand this helps to make these tasks more safe. On the other hand it increases system workload and the associated thermal load, which in turn may further aggravate the dependability issues. Therefore, this measure should be accompanied with an appropriate reliability-aware task mapping including a graceful degradation for low-critical tasks like investigated in [1]. There, the applied scheme is the following: After one of the cores exceeds a first temperature threshold T_1 a graceful degradation phase is entered. This means that tasks of high criticality are preferably assigned to cores in an exclusive manner and low-critical tasks are migrated to a “graceful degradation region” of the system. Thus, potential errors occurring in this region would involve low-critical tasks only. In a next step, if peak temperature is higher than a second threshold T_2 , low-critical tasks are removed also from the graceful degradation region (NCT ejection) and are only resumed if the thermal profile allows for it.

In [1] a simulation-based investigation of this approach has been done using the Sniper simulator, McPAT and Hotspot for a 16-core Intel Xeon X5550 running SPLASH-2 and PARSEC benchmarks. Tasks have been either classified as uncritical

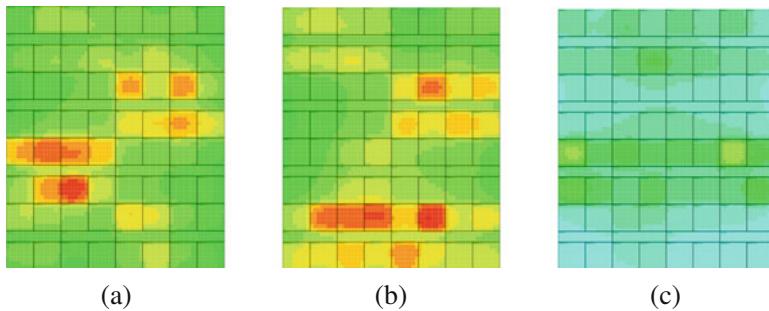


Fig. 22 Thermal profile during different phases (from [1]). The maximum system temperatures are 363 K, 378 K, and 349 K, respectively. **(a)** Initial scheduling. **(b)** Graceful degradation. **(c)** NCT ejection

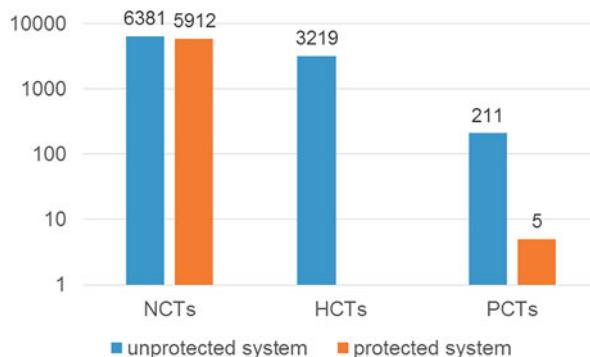


Fig. 23 Number of propagated errors per task criticality (from [1])

(NCT) or high-critical (HCT) with permanently redundant execution. As a third class, potentially critical tasks (PCT) are considered. Such tasks are dynamically replicated if the temperature of the cores they run on exceeds T_1 . In the experiment, financial analysis and computer vision applications from the benchmark sets are treated as high-critical tasks (HCT). The FFT kernel as used in a wide range of applications with different criticality levels is assumed to be PCT.

In a first experiment the thermal profile has been evaluated for normal operation and the two escalating phases. As can be seen from Fig. 22 the initial thermal hotspots are relaxed at the expense of new ones in the graceful degradation region. In turn, when moving to the NCT ejection phase the chips significantly cool down.

In a further investigation, 10,000 bit-flips have been injected randomly into cache memories independent of the criticality of the tasks running on the cores. This has been done both for a system using the mechanisms described above and as a reference for a fully unprotected system.

Figure 23 shows the resulting number of propagated errors for the different task categories. In the protected system, all errors injected into cores running HCTs are

corrected, as expected. For PCTs only those errors manifest themselves in a failure that were injected when the task was not protected due to a too low temperature of the processor core. In general, not all injected errors actually lead to a failure due to masking effects in the architecture or the application memory access pattern. This can be seen for the unprotected system where the overall sum of manifested failures is less than the number of injected errors.

7 Cross-Layer

From Physics to System Level (Fig. 24) In our work, we start from the physics, where degradation effects like aging and temperature do occur. Then we analyze and investigate how these degradations alter the key transistor parameters such as threshold voltage (V_{th}), carrier mobility (μ), sub-threshold slope (SS), and drain current (I_D). Then, we study how such drift in the electrical characteristics of the transistor impacts the resilience of circuits to errors. In practice, the susceptibility to noise effects as well as to timing violations increases. Finally, we develop models for error probability that describe the ultimate impact of these degradations at the system level.

Interaction between the System Level and the Lower Abstraction Levels (Fig. 24) Running workloads at the system level induce different stress patterns for transistors and, more importantly, generate different heat over time. Temperature is one of the key stimuli when it comes to reliability degradations. Increase in temperature accelerates the underlying aging mechanisms in transistors as well as it increases the susceptibility of circuits to noise and timing violations. Such an increase in the susceptibility manifests itself as failures at the system level due to timing violations and data corruption. Therefore, different running workloads result in different probabilities of error that can be later observed at the system level.

Key Role of Management Layer The developed probability of error models helps the management layer to make proper decision. The management layer migrates the running tasks/workload from a core that starts to have a relatively higher probability of error to another “less-aged” core. Also the management layer switches this core from a high-performance mode (where high voltage and high frequency are selected leading to higher core temperatures) to a low-power mode (where low voltage and low frequency are selected leading to lower core temperatures) when it is observed that a core started to have an increase in the probability of error above an acceptable level.

Scenarios of Cross-Layer Management and existing Interdependencies In the following we demonstrate some examples of existing interdependencies between the management layer and the lower abstraction layers.

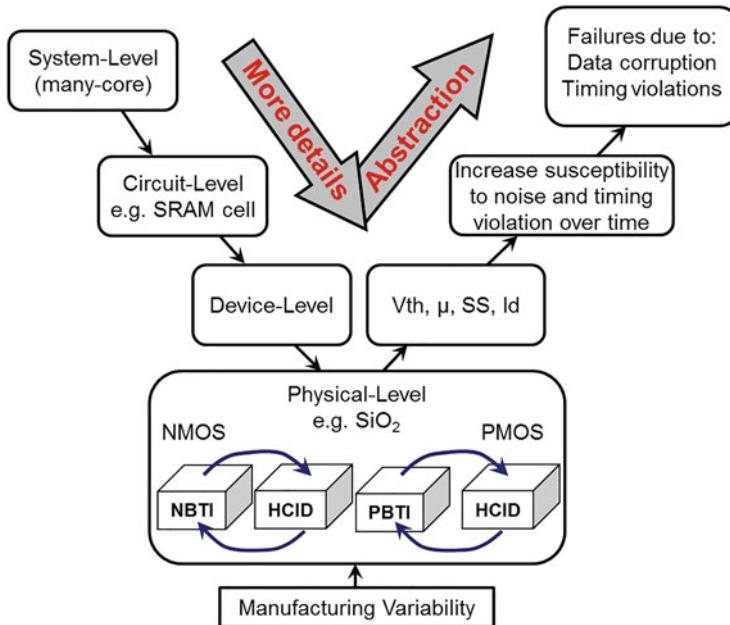


Fig. 24 Overview of how our techniques span various abstraction layers

Scenario-1: Physical and System Layer The temperature of a core increases and therefore the error probability starts to gradually rise. If a given threshold is exceeded and the core has performance margins, the first management decision would be to decrease voltage and frequency and thus limit power dissipation and in consequence counteract the temperature increase of the core.

Scenario-2: Physical, Architecture, and System Layer If there is no headroom on the core, the system management layer can now decide to migrate tasks away from that core, especially if they have high reliability requirements. Targets for migration would especially be colder, less-aged cores with a low probability of errors. With such task migrations, temperature within a system should be balanced, i.e., relieved cores can cool down, while target cores would get warmer. Further, on cores that can cool down again some of the deleterious effects start to heal, leading to a reduction in the probability of errors. In general, by continuously balancing load and as a result also temperature among cores the management layer will take care that error probabilities of cores become similar thus avoiding the situation that one core fails earlier than others. During task migrations the described support functions in the communication infrastructure (circuit layer) can be applied.

Scenario-3: Physical, Architecture, and System Layer If there is no possibility to move critical tasks to a cold core with low error probability, the management layer can employ adaptive modular redundancy (AMR) and replicate such tasks. This allows to counter the more critical operating conditions and increase reliability

by either error detection and task re-execution or by directly correcting errors when otherwise realtime requirements would not be met. However, in these cases the replica tasks will increase the overall workload of the system and thus also contribute thermal stress. In this case, dropping tasks of low criticality is a measure on system level to counter this effect.

In general, the described scenarios always form control loops starting on physical level covering temperature sensors and estimates of error probabilities and aging. They go either up to the circuit level or to the architecture/system level, where countermeasures have to be taken to prevent the system from operating under unreliable working conditions. Therefore, the mechanisms on the different abstraction levels as shown in the previous sections interact with each other and can be composed to enhance reliability in a cross-layer manner.

Further use cases tackling probabilistic fault and error modeling as well as space- and time-dependent error abstraction across different levels of the hardware/software stack of embedded systems IC components are also subject of the chapter “RAP (Resilience Articulation Point) Model.”

8 Conclusion

Reliability modeling and optimization is one of the key challenges in advanced technology. With technology scaling, the susceptibility of transistors to various kinds of degradation effects induced by aging increases. As a matter of fact, temperature is the main stimulus behind aging and therefore controlling and mitigating aging can be done through a proper thermal management. Additionally, temperature itself has also a direct impact on the reliability of any circuit manifesting itself as an increase in the probability of error. In order to sustain reliability, the system level must become aware of the degradation effects occurring at the physical level and how they then propagate to higher abstraction levels all the way up to the system level. Our cross-layer approach provides the system level with accurate estimations of the probability of errors, which allows the management layer to make proper decisions to optimize the reliability. We demonstrated the existing interdependencies between the system level and lower abstraction levels and the necessity of taking them into account via cross-layer thermal management techniques.

References

1. Alouani, I., Wild, T., Herkersdorf, A., Niar, S.: Adaptive reliability for fault tolerant multicore systems. In: 2017 Euromicro Conference on Digital System Design (DSD), pp. 538–542 (2017). <https://doi.org/10.1109/DSD.2017.78>
2. Amrouch, H., Henkel, J.: Lucid infrared thermography of thermally-constrained processors. In: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 347–352. IEEE, Piscataway (2015)

3. Amrouch, H., van Santen, V.M., Ebi, T., Wenzel, V., Henkel, J.: Towards interdependencies of aging mechanisms. In: Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design, pp. 478–485. IEEE, Piscataway (2014)
4. Amrouch, H., Martin-Martinez, J., van Santen, V.M., Moras, M., Rodriguez, R., Nafria, M., Henkel, J.: Connecting the physical and application level towards grasping aging effects. In: 2015 IEEE International Reliability Physics Symposium, p 3D-1. IEEE, Piscataway (2015)
5. Amrouch, H., Khaleghi, B., Gerstlauer, A., Henkel, J.: Towards aging-induced approximations. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2017)
6. Amrouch, H., van Santen, V.M., Henkel, J.: Interdependencies of degradation effects and their impact on computing. *IEEE Des. Test* **34**(3), 59–67 (2017)
7. Boroujerdian, B., Amrouch, H., Henkel, J., Gerstlauer, A.: Trading off temperature guardbands via adaptive approximations. In: 2018 IEEE 36th International Conference on Computer Design (ICCD), pp. 202–209. IEEE, Piscataway (2018)
8. Ebi, T., Rauchfuss, H., Herkersdorf, A., Henkel, J.: Agent-based thermal management using real-time i/o communication relocation for 3d many-cores. In: Ayala, J.L., García-Cámarra, B., Prieto, M., Ruggiero, M., Sicard, G. (eds.) Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation, pp. 112–121. Springer, Berlin (2011)
9. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., et al.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. circuits Syst.* **32**(1), 8–23 (2012)
10. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., et al.: Design and architectures for dependable embedded systems. In: Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 69–78. ACM, New York (2011)
11. Herkersdorf, A., Michel, H.U., Rauchfuss, H., Wild, T.: Multicore enablement for automotive cyber physical systems. *Inf. Technol.* **54**, 280–287 (2012). <https://doi.org/10.1524/itit.2012.0690>
12. Hussam, A., Jörg, H.: Evaluating and mitigating degradation effects in multimedia circuits. In: Proceedings of the 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia '17), pp. 61–67. Association for Computing Machinery, New York (2017). <https://doi.org/10.1145/3139315.3143527>
13. Khdr, H., Ebi, T., Shafique, M., Amrouch, H.: mDTM: multi-objective dynamic thermal management for on-chip systems. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1–6. IEEE, Piscataway (2014)
14. Prakash, A., Amrouch, H., Shafique, M., Mitra, T., Henkel, J.: Improving mobile gaming performance through cooperative CPU-GPU thermal management. In: Proceedings of the 53rd Annual Design Automation Conference, p. 47. ACM, New York (2016)
15. Rauchfuss, H., Wild, T., Herkersdorf, A.: Enhanced reliability in tiled manycore architectures through transparent task relocation. In: ARCS 2012, pp. 1–6 (2012)
16. Rösch, S., Rauchfuss, H., Wallentowitz, S., Wild, T., Herkersdorf, A.: MPSoC application resilience by hardware-assisted communication virtualization. *Microelectron. Reliab.* **61**, 11–16 (2016). <https://doi.org/10.1016/j.microrel.2016.02.009>. <http://www.sciencedirect.com/science/article/pii/S0026271416300282>. SI: ICMAT 2015
17. van Santen, V.M., Amrouch, H., Parihar, N., Mahapatra, S., Henkel, J.: Aging-aware voltage scaling. In: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 576–581. IEEE, Piscataway (2016)
18. van Santen, V.M., Martin-Martinez, J., Amrouch, H., Nafria, M.M., Henkel, J.: Reliability in super-and near-threshold computing: a unified model of RTN, BTI, and PV. *IEEE Trans. Circuits Syst. I Regul. Pap.* **65**(1), 293–306 (2018)

19. van Santen, V.M., Diaz-Fortuny, J., Amrouch, H., Martin-Martinez, J., Rodriguez, R., Castro-Lopez, R., Roca, E., Fernandez, F.V., Henkel, J., Nafria, M.: Weighted time lag plot defect parameter extraction and GPU-based BTI modeling for BTI variability. In: 2018 IEEE International Reliability Physics Symposium (IRPS), pp. P-CR.6-1–P-CR.6-6 (2018). <https://doi.org/10.1109/IRPS.2018.8353659>
20. van Santen, V.M., Amrouch, H., Henkel, J.: Modeling and mitigating time-dependent variability from the physical level to the circuit level. *IEEE Trans. Circuits Syst. I Regul. Pap.* 1–14 (2019). <https://doi.org/10.1109/TCSI.2019.2898006>
21. van Santen, V.M., Amrouch, H., Henkel, J.: New worst-case timing for standard cells under aging effects. *IEEE Trans. Device Mater. Reliab.* **19**(1), 149–158 (2019). <https://doi.org/10.1109/TDMR.2019.2893017>
22. Wallentowitz, S., Rösch, S., Wild, T., Herkersdorf, A., Wenzel, V., Henkel, J.: Dependable task and communication migration in tiled manycore system-on-chip. In: Proceedings of the 2014 Forum on Specification and Design Languages (FDL), vol. 978-2-9530504-9-3, pp. 1–8 (2014). <https://doi.org/10.1109/FDL.2014.7119361>
23. Wallentowitz, S., Tempelmeier, M., Wild, T., Herkersdorf, A.: Network-on-chip protection switching techniques for dependable task migration on an open source MPSoC platform.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Lightweight Software-Defined Error Correction for Memories



Irina Alam, Lara Dolecek, and Puneet Gupta

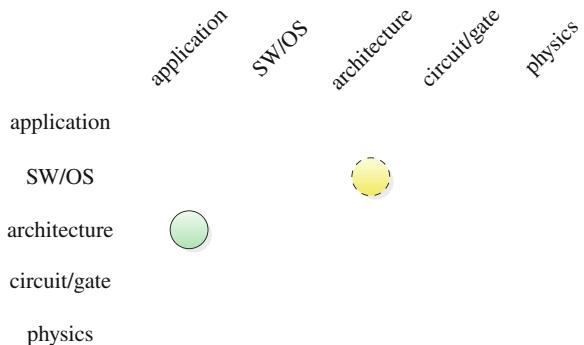
The key observation behind the techniques described in this chapter is that most if not all error correction techniques and codes assume that all words stored in the memory are equally likely and important. This obviously is not true due to architectural or application context. This chapter devises new coding and correction mechanisms which leverage software or architecture “side information” to dramatically reduce the cost of error correction (Fig. 1). The methodology proposed in Sect. 1 is for recovering from detected-but-uncorrectable (DUE) errors in main memories while Sects. 2 and 3 focus on lightweight correction in on-chip caches or embedded memories.

1 Software-Defined Error Correcting Codes (SDECC)

This section focuses on the concept of Software-Defined Error Correcting Codes (SDECC), a general class of techniques spanning hardware, software, and coding theory that improves the overall resilience of systems by enabling heuristic best-effort recovery from detected-but-uncorrectable errors (DUE). The key idea is to add software support to the hardware error correcting code (ECC) so that most memory DUEs can be heuristically recovered based on available *side information* (SI) from the corresponding un-corrupted cache line contents. SDECC does not degrade memory performance or energy in the common cases when either no errors or purely hardware-correctable errors occur. Yet it can significantly improve resilience in the critical case when DUEs actually do occur.

I. Alam · L. Dolecek · P. Gupta (✉)
ECE Department, UCLA, Los Angeles, CA, USA
e-mail: irinal@ucla.edu; dolecek@ee.ucla.edu; puneetg@ucla.edu

Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



Details of the concepts discussed in this section can be found in the works by Gottscho et al. [11, 12].

1.1 SDECC Theory

Important terms and notation introduced here are summarized in Table 1.

A $(t)SC(t+1)SD$ code corrects up to t symbol errors and/or detects up to $(t+1)$ symbol errors. SDECC is based on the fundamental observation that when a $(t+1)$ -symbol DUE occurs in a $(t)SC(t+1)SD$ code, there remains significant information in the received string \mathbf{x} . This information can be used to recover the original message \mathbf{m} with reasonable certainty.

It is not the case that the original message was completely lost, i.e., one need not naïvely choose from all q^k possible messages. If there is a $(t+1)$ DUE, there are exactly

$$N = \binom{n}{t+1} (q-1)^{(t+1)} \quad (1)$$

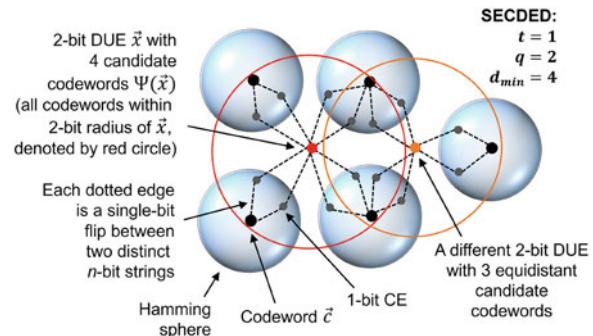
ways that the $(t+1)$ DUE could have corrupted the original codeword, which is less than q^k . Though a $(t)SC(t+1)SD$ code can often detect more than $(t+1)$ errors, a $(t+1)$ error is usually much more likely than higher bit errors. But guessing correctly out of N possibilities is still difficult. *In practice, there are just a handful of possibilities: they are referred to as $(t+1)$ DUE corrupted candidate codewords (or candidate messages).*

Consider Fig. 2, which depicts the relationships between codewords, correctable errors (CEs), DUEs, and candidate codewords for individual DUEs for a Single-bit Error Correcting, Double-bit Error Detecting (SECDED) code. If the hardware ECC decoder registers a DUE, there can be several equidistant candidate codewords at the

Table 1 Important SDECC-specific notation

Term	Description
n	Codeword length in symbols
k	Message length in symbols
r	Parity length in symbols
b	Bits per symbol
q	Symbol alphabet size
t	Max. guaranteed correctable symbols in codeword
$(t)SC(t+1)SD$	(t) -symbol-correcting, $(t+1)$ -symbol-detecting
N	Number of ways to have a DUE
μ	Mean no. of candidate codewords \forall possible DUEs
P_G	Prob. of choosing correct codeword for a given DUE
$\overline{P_G}$	Avg. prob. of choosing correct codeword \forall possible DUEs
d_{min}	Minimum symbol distance of code
linesz	Total cache line size in symbols (message content)
symbol	Logical group of bits
SECDED	Single-bit-error-correcting, double-bit-error-detecting
DECTED	Double-bit-error-correcting, triple-bit-error-detecting
SSCSDSD	Single-symbol-error-correcting, double-symbol-error-detecting
ChipKill-correct	ECC construction and mem. organization that either corrects up to 1 DRAM chip failure or detects 2 chip failures

Fig. 2 Illustration of candidate codewords for 2-bit DUEs in the imaginary 2D-represented Hamming space of a binary SECDED code ($t = 1, q = 2, d_{min} = 4$). The actual Hamming space has n dimensions



q -ary Hamming distance of exactly $(t + 1)$ from the received string \mathbf{x} . Without any *side information* (SI) about message probabilities, under conventional principles, each candidate codeword is assumed to be equally likely. However, in the specific case of DUEs, not all messages are equally likely to occur: this allows to leverage SI about memory contents to help choose the right candidate codeword in the event of a given DUE.

1.1.1 Computing the List of Candidates

The number of candidate codewords for any given $(t + 1)$ DUE \mathbf{e} has a linear upper bound that makes DUE recovery tractable to implement in practice [12]. The candidate codewords for any $(t + 1)$ -symbol DUE received string \mathbf{x} is simply the set of equidistant codewords that are exactly $(t + 1)$ symbols away from \mathbf{x} . This list depends on the error \mathbf{e} and original codeword \mathbf{c} , but only the received string \mathbf{x} is known. Fortunately, there is a simple and intuitive algorithm to find the list of candidate codewords with runtime complexity $O(nq/t)$. The detailed algorithm can be found in [12]. The essential idea is to try every possible single symbol *perturbation* \mathbf{p} on the received string. Each *perturbed string* $\mathbf{y} = \mathbf{x} + \mathbf{p}$ is run through a simple software implementation of the ECC decoder, which only requires knowledge of the parity-check matrix \mathbf{H} ($O(rn\log q)$ bits of storage). Any \mathbf{y} characterized as a CE produces a candidate codeword from the decoder output and added to the list (if not already present in the list).

1.1.2 SDECC Analysis of Existing ECCs

Code constructions exhibit structural properties that affect the number of candidate codewords. In fact, distinct code constructions with the same $[n, k, d_{min}]_q$ parameters can have different values of μ and distributions of the number of candidate codewords. μ depends on the total number of minimum weight non- $\vec{0}$ codewords [12].

The SDECC theory is applied to seven code constructions of interest: SECDED, DECTED, and SSCDSD (ChipKill-Correct) constructions with typical message lengths of 64, and 128 bits. Table 2 lists properties that have been derived for each of them. Most importantly, the final column lists \overline{P}_G —the average (*random baseline*) probability of choosing correct codeword without SI for all possible DUEs. These probabilities are far higher than the naïve approaches of guessing randomly from q^k possible messages or from the N possible ways to have a DUE. Thus, SDECC can handle DUEs in a more optimistic way than conventional ECC approaches.

Table 2 Summary of code properties— \overline{P}_G is most important for SDECC

Class of code	Code params. $[n, k, d_{min}]_q$	Type of code	Class of DUE $(t + 1)$	Avg. # Cand. μ	Prob. Rcov. \overline{P}_G
32-bit SECDED	$[39, 32, 4]_2$	Hsiao [16]	2-bit	12.04	8.50%
32-bit SECDED	$[39, 32, 4]_2$	Davydov [7]	2-bit	9.67	11.70%
64-bit SECDED	$[72, 64, 4]_2$	Hsiao [16]	2-bit	20.73	4.97%
64-bit SECDED	$[72, 64, 4]_2$	Davydov [7]	2-bit	16.62	6.85%
32-bit DECTED	$[45, 32, 6]_2$	—	3-bit	4.12	28.20%
64-bit DECTED	$[79, 64, 6]_2$	—	3-bit	5.40	20.53%
128-bit SSCDSD	$[36, 32, 4]_{16}$	Kaneda [17]	2-sym.	3.38	39.88%

1.2 SDECC Architecture

SDECC consists of both hardware and software components to enable recovery from DUEs in main memory DRAM. A simple hardware/software architecture whose block diagram is depicted in Fig. 3 can be used. Although the software flow includes an instruction recovery policy, it is not presented in this chapter because DUEs on instruction fetches are likely to affect clean pages that can be remedied using a page fault (as shown in the figure).

The key addition to hardware is the *Penalty Box*: a small buffer in the memory controller that can store each codeword from a cache line (shown on the left-hand side of Fig. 3). When a memory DUE occurs, hardware stores information about the error in the Penalty Box and raises an error-reporting interrupt to system software. System software then reads the Penalty Box, derives additional context about the error—and using basic coding theory and knowledge of the ECC implementation—quickly computes a list of all possible *candidate messages*, one of which is guaranteed to match the original information that was corrupted by the DUE. A software-defined data recovery policy heuristically recovers the DUE in a best-effort manner by choosing the most likely remaining candidate based on available *side information* (SI) from the corresponding un-corrupted cache line contents; if confidence is low, the policy instead forces a panic to minimize the risk of accidentally induced mis-corrected errors (MCEs) that result in intolerable non-silent data corruption (NSDC). Finally, system software writes back the *recovery target* message to the Penalty Box, which allows hardware to complete the afflicted memory read operation.

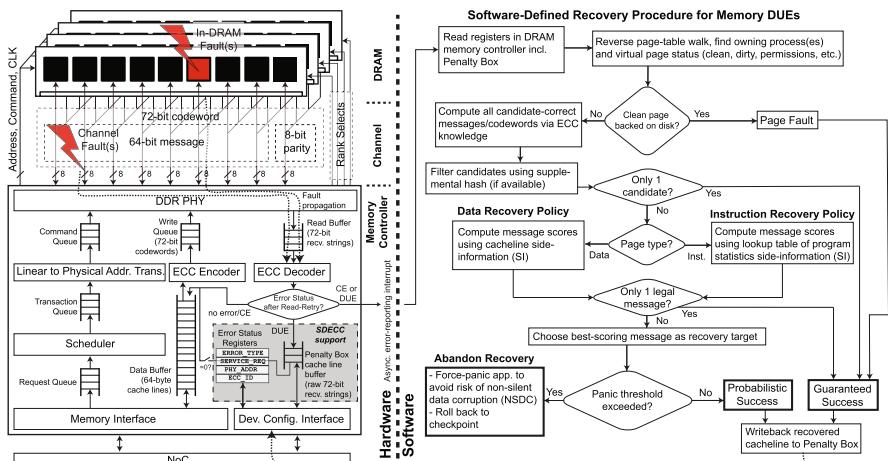


Fig. 3 Block diagram of a general hardware and software implementation of SDECC. The figure depicts a typical DDRx-based main memory subsystem with 64-byte cache lines, x8 DRAM chips, and a $[72, 64, 4]_2$ SECDED ECC code. Hardware support necessary to enable SDECC is shaded in gray. The instruction recovery policy is outside the scope of this work [12]

Overheads The area and power overhead of the essential SDECC hardware support is negligible. The area required per Penalty Box is approximately $736 \mu\text{m}^2$ when synthesized with 15 nm Nangate technology—this is approximately one millionth of the total die area for a 14 nm Intel Broadwell-EP server processor [9]. The SDECC design incurs no latency or bandwidth overheads for the vast majority of memory accesses where no DUEs occur. This is because the Penalty Box and error-reporting interrupt are not on the critical path of memory accesses. When a DUE occurs, the latency of the handler and recovery policy is negligible compared to the expected mean time between DUEs or typical checkpoint interval of several hours.

1.3 Data Recovery Policy

In this section, recovery of DUEs in data (i.e., memory reads due to processor loads) is discussed because they are more vulnerable than DUEs in instructions as mentioned before. Possible recovery policies for instruction memory have been discussed in [11]. There are potentially many sources of SI for recovering DUEs in data. Based on the notion of *data similarity*, a simple but effective data recovery policy called *Entropy-Z* is discussed here that chooses the candidate that minimizes overall cache line Shannon entropy.

1.3.1 Observations on Data Similarity

Entropy is one of the most powerful metrics to measure data similarity. Two general observations can be made about the prevalence of low data entropy in memory.

- **Observation 1.** There are only a few primitive data types supported by hardware (e.g., integers, floating-point, and addresses), which typically come in multiple widths (e.g., byte, halfword, word, or quadword) and are often laid out in regular fashion (e.g., arrays and structs).
- **Observation 2.** In addition to spatial and temporal locality in their memory access patterns, applications have inherent *value locality* in their data, regardless of their hardware representation. For example, an image-processing program is likely to work on regions of pixels that exhibit similar color and brightness, while a natural language processing application will see certain characters and words more often than others.

Similar observations have been made to compress memory [2, 18, 24, 26, 28, 35] and to predict [20] or approximate processor load values [22, 23, 36]. Low byte-granularity intra-cache line entropy is observed throughout the integer and floating-point benchmarks in the SPEC CPU2006 suite. Let $P(X)$ be the normalized relative frequency distribution of a $\text{linesz} \times b$ -bit cache line that has been carved

into equal-sized Z -bit symbols, where each symbol χ_i can take 2^Z possible values.¹ Then the Z -bit-granularity entropy is computed as follows:

$$\text{entropy} = - \sum_{i=1}^{\text{linesz} \times b/Z} P(\chi_i) \log_2 P(\chi_i). \quad (2)$$

The average intra-cacheline byte-level entropy of the SPEC CPU2006 suite was found to be 2.98 bits (roughly half of maximum).

These observations can be leveraged using the data recovery policy *Entropy-Z Policy*. With this policy, SDECC first computes the list of candidate messages using the algorithm described in Sect. 1.1.1 and extracts the cache line side information. Each candidate message is then inserted into appropriate position in the affected cache line and the entropy is computed using Eq. 2. The policy then chooses the candidate message that minimizes overall cache line entropy. The chance that the policy chooses the wrong candidate message is significantly reduced by deliberately forcing a *panic* whenever there is a tie for minimum entropy or if the mean cache line entropy is above a specified threshold `PanicThreshold`. The downside to this approach is that some forced panics will be false positives, i.e., they would have otherwise recovered correctly.

In the rest of the chapter, unless otherwise specified, $Z = 8$ bits, $\text{linesz} \times b = 512$ bits and `PanicThreshold` = 4.5 bits (75% of maximum entropy) are used, which were determined to work well across a range of applications. Additionally, the *Entropy-8* policy performs very well compared to several alternatives.

1.4 Reliability Evaluation

The impact of SDECC is evaluated on system-level reliability through a comprehensive error injection study on memory access traces. The objective is to estimate the fraction of DUEs in memory that can be recovered correctly using the SDECC architecture and policies while ensuring a minimal risk of MCEs.

1.4.1 Methodology

The SPEC CPU2006 benchmarks are compiled against GNU/Linux for the open-source 64-bit RISC-V (RV64G) instruction set v2.0 [34] using the official tools [25]. Each benchmark is executed on top of the RISC-V proxy kernel [32] using the Spike simulator [33] that was modified to produce representative memory access

¹Entropy symbols are not to be confused with the codeword symbols, which can also be a different size.

traces. Each trace consists of randomly sampled 64-byte demand read cache lines, with an average interval between samples of one million accesses.

Each trace is analyzed offline using a MATLAB model of SDECC. For each benchmark and ECC code, 1000 q -ary messages from the trace were chosen randomly and encoded, and were injected with $\min(1000, N)$ randomly sampled $(t + 1)$ -symbol DUEs. N here is the number of ways to have a DUE. For each codeword/error pattern combination, the list of candidate codewords was computed and the data recovery policy was applied. A *successful recovery* occurs when the policy selects a candidate message that matches the original; otherwise, the policy either causes a *forced panic* or recovery fails by accidentally inducing an MCE. Variability in the reported results is negligible over many millions of individual experiments.

Note that the *absolute* error magnitudes for DUEs and SDECC's impact on *overall* reliability should not be compared directly between codes with distinct $[n, k, d_{\min}]_q$ (e.g., a double-bit error for SECDED is very different from a double-chip DUE for ChipKill). Rather, what matters most is the *relative* fraction of DUEs that can be saved using SDECC for a given ECC code.

Entropy-8 is exclusively used as the data recovery policy in all the evaluations. This is because when the raw successful recovery rates of six different policies for three ECCs without including any forced panics were compared, *Entropy-8* performed the best [12]. Few examples of alternate policies include *Entropy-Z* policy variants with $Z = 4$ and $Z = 16$ and *Hamming* which chooses the candidate that minimizes the average binary Hamming distance to the neighboring words in the cacheline. The 8-bit entropy symbol size performs best because its alphabet size ($2^8 = 256$ values) matches well with the number of entropy symbols per cacheline (64) and with the byte-addressable memory organization. For instance, both *Entropy-4* and *Entropy-16* do worse than *Entropy-8* because the entropy symbol size results in too many aliases at the cacheline level and because the larger symbol size is less efficient, respectively.

1.4.2 Recovery Breakdown

SDECC is evaluated next for each ECC using its conventional form, to understand the impact of the recovery policy's (*Entropy-8*) forced panics on the successful recovery rate and the MCE rate. The overall results with forced panics *taken* (main results, gray cell shading) and *not taken* are shown in Table 3.

There are two baseline DUE recovery policies: *conventional* (always panic for every DUE) and *random* (choose a candidate randomly, i.e., P_G). It is observed that when panics are taken the MCE rate drops significantly by a factor of up to $7.3\times$ without significantly reducing the success rate. This indicates that the PanicThreshold mechanism appropriately judges when SDECC is unlikely to correctly recover the original information.

These results also show the impact of code construction on successes, panics, and MCEs. When there are fewer average candidates μ then the chances of successfully

Table 3 Percent Breakdown of SDECC *Entropy-8* Policy (M = MCE, P = forced panic, S = success) [12]

	Panics taken			Panics not taken			Random baseline		
	M	P	S	M	P	S	M	P	S
Conv. baseline	–	100	–	–	–	–	–	–	–
[39, 32, 4] ₂ Hsiao	5.3	25.6	69.1	27.3	–	72.7	91.5	–	8.5
[39, 32, 4] ₂ Davydov	4.5	25.2	70.3	24.0	–	76.0	88.3	–	11.7
[72, 64, 4] ₂ Hsiao	4.7	23.7	71.6	24.7	–	75.3	95.0	–	5.0
[72, 64, 4] ₂ Davydov	4.1	21.9	74.0	22.3	–	77.7	93.2	–	6.9
[45, 32, 6] ₂ DECTED	2.2	20.3	77.5	14.5	–	85.5	71.8	–	28.2
[79, 64, 6] ₂ DECTED	1.5	14.5	84.0	11.0	–	89.0	79.5	–	20.5
[36, 32, 4] ₁₆ SSCDSD	1.5	12.8	85.7	8.5	–	91.5	60.1	–	39.9

recovering are much higher than that of inducing MCEs. The [72, 64, 4]₂ SECDED constructions perform similarly to their [39, 32, 4]₂ variants even though the former have lower baseline $\overline{P_G}$. This is a consequence of the *Entropy-8* policy: larger n combined with lower μ provides the greatest opportunity to differentiate candidates with respect to overall intra-cacheline entropy. For the same n , however, the effect of SECDED construction is more apparent. The Davydov codes recover about 3–4% more frequently than their Hsiao counterparts when panics are not taken (similar to the baseline improvement in $\overline{P_G}$). When panics are taken, however, the differences in construction are less apparent because the policy `PanicThreshold` does not take into account Davydov’s typically lower number of candidates.

The breakdown between successes, panics, and MCEs is examined in more detail. Figure 4 depicts the DUE recovery breakdowns for each ECC construction and SPEC CPU2006 benchmark when forced panics are taken. Figure 4a shows the fraction of DUEs that result in success (black), panics (gray), and MCEs (white). Figure 4b further breaks down the forced panics (gray from Fig. 4a) into a fraction that are *false positive* (light purple, and would have otherwise been correct) and others that are *true positive* (dark blue, and managed to avoid an MCE). Each cluster of seven stacked bars corresponds to the seven ECC constructions.

It can be seen that much lower MCE rates are achieved than the *random* baseline yet also panic much less often than the *conventional* baseline for all benchmarks, as shown in Fig. 4a. This policy performs best on integer benchmarks due to their lower average intra-cacheline entropy. For certain floating-point benchmarks, however, there are many forced panics because they frequently have high data entropy above `PanicThreshold`. A `PanicThreshold` of 4.5 bits for these cases errs on the side of caution as indicated by the false positive panic rate, which can be up to 50%. Without more side information, for high-entropy benchmarks, it would be difficult for any alternative policy to frequently recover the original information with a low MCE rate and few false positive panics.

With almost no hardware overheads, SDECC used with SSCDSD ChipKill can recover correctly from up to 85.7% of double-chip DUEs while eliminating 87.2% of would-be panics; this could improve system availability considerably. However,

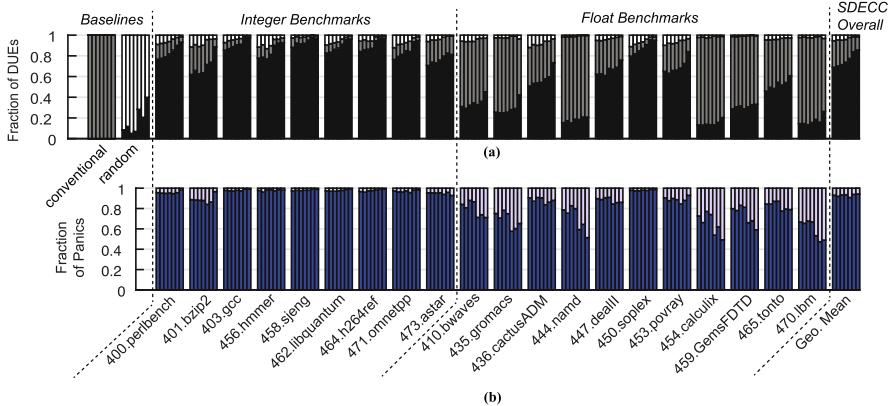


Fig. 4 Detailed breakdown of DUE recovery results when forced panics are taken. Results are shown for all seven ECC constructions, listed left to right within each cluster: [39, 32, 4]₂ Hsiao SECDED—[39, 32, 4]₂ Davydov SECDED—[72, 64, 4]₂ Hsiao SECDED—[72, 64, 4]₂ Davydov SECDED—[45, 32, 6]₂ DECTED—[79, 64, 6]₂ DECTED—[36, 32, 4]₁₆ SSCDSD ChipKill-Correct. **(a)** Recovery breakdown for the *Entropy-8* policy, where each DUE can result in an unsuccessful recovery causing an MCE (white), forced panic (gray), or successful recovery (black). **(b)** Breakdown of forced panics (gray bars in **(a)**). A true positive panic (dark blue) successfully mitigated a MCE, while a false positive panic (light purple) was too conservative and thwarted an otherwise-successful recovery [12]

SDECC with ChipKill introduces a 1% risk of converting a DUE to an MCE. Without further action taken to mitigate MCEs, this small risk may be unacceptable when application correctness is of paramount importance.

2 Software-Defined Error-Localizing Codes (SDELCC): Lightweight Recovery from Soft Faults at Runtime

For embedded memories, it is always challenging to address reliability concerns as additional area, power, and latency overheads of reliability techniques need to be minimized as much as possible. *Software-Defined Error-Localizing Codes* (SDELCC) is a hybrid hardware/software technique that deals with single-bit soft faults at runtime using novel *Ultra-Lightweight Error-Localizing Codes* (UL-ELC) with a software-defined error handler that knows about the UL-ELC construction and implements a heuristic recovery policy. UL-ELC codes are stronger than basic single-error detecting (SED) parity, yet they have lower storage overheads than a single-error-correcting (SEC) Hamming code. Like SED, UL-ELC codes can detect single-bit errors, yet they can additionally *localize* them to a *chunk* of the erroneous codeword. UL-ELC codes can be explicitly designed such that chunks align with meaningful message context, such as the fields of an encoded instruction.

SDELCC then relies on *side information* (SI) about application memory contents to heuristically recover from the single-bit fault. Unlike the general-purpose Software-Defined ECC (SDECC), SDELCC focuses on heuristic error recovery that is suitable for microcontroller-class IoT devices.

Details of the concepts discussed in this section can be found in the work by Gottscho et al. [13].

2.1 Ultra-Lightweight Error-Localizing Codes (UL-ELC)

In today's systems, either basic SED parity is used to detect random single-bit errors or a Hamming SEC code is used to correct them. Unfortunately, Hamming codes are expensive for small embedded memories: they require six bits of parity per memory word size of 32 bits (an 18.75% storage overhead). On the other hand, basic parity only adds one bit per word (3.125% storage overhead), but without assistance by other techniques it cannot correct any errors.

Localizing an error is more useful than simply detecting it. If the error is localized to a *chunk* of length ℓ bits, there are only ℓ *candidate codewords* for which a single-bit error could have produced the received (corrupted) codeword. A naïve way of localizing a single-bit error to a particular chunk is to use a trivial segmented parity code, i.e., assign a dedicated parity bit to each chunk. However, this method is very inefficient because to create C chunks C parity bits are needed: essentially, split up the memory words into smaller pieces.

Instead *Ultra-Lightweight* ELCs (UL-ELCs) is simple and customizable—given r redundant parity bits—it can localize any single-bit error to one of $C = 2^r - 1$ possible chunks. This is because there are $2^r - 1$ distinct non-zero columns that can be used to form the parity-check matrix \mathbf{H} for the UL-ELC (for single-bit errors, the error syndrome is simply one of the columns of \mathbf{H}). To create a UL-ELC code, a distinct non-zero binary column vector of length r bits is assigned to each chunk. Then each column of \mathbf{H} is simply filled in with the corresponding chunk vector. Note that r of the chunks will also contain the associated parity bit within the chunk itself and are called *shared chunks*, and they are precisely the chunks whose columns in \mathbf{H} have a Hamming weight of 1. Since there are r shared chunks, there must be $2^r - r - 1$ *unshared chunks*, which each consist of only data bits. Shared chunks are unavoidable because the parity bits must also be protected against faults, just like the message bits.

An UL-ELC code has a minimum distance of two bits by construction to support detection and localization of single-bit errors. Thus, the set of candidate codewords must also be separated from each other by a Hamming distance of exactly two bits. (A minimum codeword distance of two bits is required for SED, while three bits are needed for SEC, etc.)

For an example of an UL-ELC construction, consider the following $\mathbf{H}_{\text{example}}$ parity-check matrix with nine message bits and $r = 3$ parity bits:

$$\mathbf{H}_{\text{example}} = \begin{matrix} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_6 & S_7 & S_5 & S_6 & S_7 \\ & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & p_1 & p_2 & p_3 \\ \mathbf{H}_{\text{example}} = c_1 & \left[\begin{matrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{matrix} \right] \\ c_2 & \left[\begin{matrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{matrix} \right] \\ c_3 & \left[\begin{matrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{matrix} \right] \end{matrix}$$

where d_i represents the i th data bit, p_j is the j th redundant parity bit, c_k is the k th parity-check equation, and S_l enumerates the distinct error-localizing chunk that a given bit belongs to. Because $r = 3$, there are $N = 7$ chunks. Bits d_1 , d_2 , and d_3 each have the SEC property because no other bits are in their respective chunks. Bits d_4 and d_5 make up an unshared chunk S_4 because no parity bits are included in S_4 . The remaining data bits belong to shared chunks because each of them also includes at least one parity bit. Notice that any data or parity bits that belong to the same chunk S_l have identical columns of \mathbf{H} , e.g., d_7 , d_8 , and p_2 all belong to S_6 and have the column $[0; 1; 0]$.

The two key properties of UL-ELC (that do not apply to generalized ELC codes) are: (1) the length of the data message is independent of r and (2) each chunk can be an arbitrary length. The freedom to choose the length of the code and chunk sizes allows the UL-ELC design to be highly adaptable. Additionally, UL-ELC codes can offer SEC protection on up to $2^r - r - 1$ selected message bits by having the unshared chunks each correspond to a single data bit.

2.2 Recovering SEUs in Instruction Memory

This section focuses on an UL-ELC construction and recovery policy for dealing with single-bit soft faults in instruction memory. The code and policy are jointly crafted to exploit SI about the ISA itself. This SDELC implementation example targets the open-source and free 64-bit RISC-V (RV64G) ISA [34], but the approach is general and could apply to any other fixed-length or variable-length RISC or CISC ISA. Note that although RISC-V is actually a little-endian architecture, for sake of clarity big-endian is used in this example.

The UL-ELC construction for instruction memory has seven chunks that align to the finest-grain boundaries of the different fields in the RISC-V codecs. These codecs, the chunk assignments, and the complete parity-check matrix \mathbf{H} are shown in Table 4. The `opcode`, `rd`, `funct3`, and `rs1` fields are the most commonly used—and potentially the most critical—among the possible instruction encodings, so each of them is assigned a dedicated chunk that is unshared with the parity bits. The fields which vary more among encodings are assigned to the remaining three shared chunks, as shown in the figure. The recovery policy can thus distinguish

Table 4 Proposed 7-Chunk UL-ELC Construction with $r = 3$ for Instruction Memory (RV64G ISA v2.0)

bit \rightarrow	31	27	26	25	24	20	19	15	14	12	11	7	6	0	-1	-3
Type-UJ	imm [20:10:1 11:19:12]									rd				opcode	parity	
Type-U	imm [31:12]									rd				opcode	parity	
Type-I	imm [11:0]					rs1		funct3		rd				opcode	parity	
Type-R	funct7			rs2		rs1		funct3		rd				opcode	parity	
Type-S	imm [11:5]			rs2		rs1		funct3		imm [4:0]				opcode	parity	
Type-SB	imm [12:10:5]			rs2		rs1		funct3		imm [4:1 11]				opcode	parity	
Type-R4	rs3	funct2	rs2	rs1		funct3		rd		rd				opcode	parity	
Chunk	C_1 (shared)	C_2 (shared)	C_3 (shared)		C_4		C_5		C_6		C_7		C_3	C_2	C_1	
Parity-	11111	00	00000	11111	000		11111	000	11111	11111	1111111	1	0	0	0	
Check	00000	11	00000	00000	00000	11111	11111	11111	11111	1111111	1111111	0	1	0	0	
H	00000	00	11111	11111	11111	11111	00000	00000	1111111	1111111	1111111	0	0	0	1	

the impact of an error in different parts of the instruction. For example, when a fault affects shared chunk C_1 , the fault is either in one of the five MSBs of the instruction, or in the last parity bit. Conversely, when a fault is localized to unshared chunk C_7 in Table 4, the UL-ELC decoder can be certain that the `opcode` field has been corrupted.

The instruction recovery policy consists of three steps.

- **Step 1.** A software-implemented instruction decoder is applied to filter out any candidate messages that are illegal instructions. Most bit patterns decode to illegal instructions in three RISC ISAs that were characterized: 92.33% for RISC-V, 72.44% for MIPS, and 66.87% for Alpha. This can be used to dramatically improve the chances of a successful SDELC recovery.
- **Step 2.** Next, the probability of each valid message is estimated using a small pre-computed lookup table that contains the relative frequency that each instruction appears. The relative frequencies of legal instructions in most applications follow power-law distribution [13]. This is used to favor more common instructions.
- **Step 3.** The instruction that is most common according to the SI lookup table is chosen. In the event of a tie, the instruction with the longest leading-pad of 0s or 1s is chosen. This is because in many instructions, the MSBs represent immediate values (as shown in Table 4). These MSBs are usually low-magnitude signed integers or they represent 0-dominant function codes.

If the SI is strong, then there is normally a higher chance of correcting the error by choosing the right candidate.

2.3 Recovering SEUs in Data Memory

In general-purpose embedded applications, data may come in many different types and structures. Because there is no single common data type and layout in memory, evenly spaced UL-ELC constructions can be used and the software trap handler can be granted additional control about how to recover from errors, similar to the general idea from SuperGlue [31].

The SDELC recovery support can be built into the embedded application as a small C library. The application can push and pop custom SDELC error handler functions onto a registration stack. The handlers are defined within the scope of a subroutine and optionally any of its callees and can define specific recovery behaviors depending on the context at the time of error. Applications can also enable and disable recovery at will.

When the application does not disable recovery nor specify a custom behavior, all data memory errors are recovered using a default error handler implemented by the library. The default handler computes the average Hamming distance to nearby data in the same 64-byte chunk of memory (similar to taking the intra-cache line distance in cache-based systems). The candidate with the minimum average

Hamming distance is selected. This policy is based on the observation that spatially local and/or temporally local data tends to also be correlated, i.e., it exhibits *value locality* [20].

The application-defined error handler can specify recovery rules for individual variables within the scope of the registered subroutine. They include globals, heap, and stack-allocated data. This is implemented by taking the runtime address of each variable requiring special handling. For instance, an application may wish critical data structures to never be recovered heuristically; for these, the application can choose to force a crash whenever a soft error impacts their memory addresses. The SDELC library support can increase system reliability, but the programmer is required to spend effort annotating source code for error recovery. This is similar to annotation-based approaches taken by others for various purposes [4, 5, 10, 21, 29, 37].

2.4 SDELC Architecture

The SDELC architecture is illustrated in Fig. 5 for a system with split on-chip instruction and data scratchpad memories (SPMs) (each with its own UL-ELC code) and a single-issue core that has an in-order pipeline.

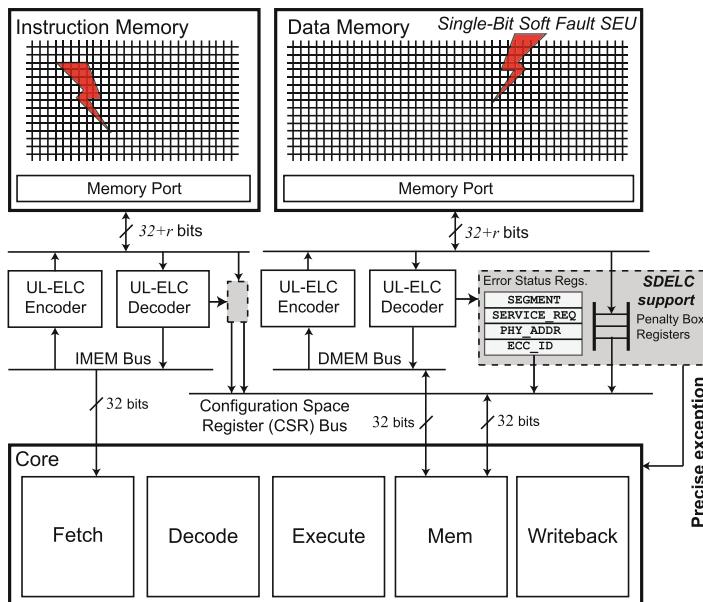


Fig. 5 Architectural support for SDELC on a microcontroller-class embedded system

When a codeword containing a single-bit soft fault is read, the UL-ELC decoder detects and localizes the error to a specific chunk of the codeword and places error information in a *Penalty Box* register (shaded in gray in the figure). A precise exception is then generated, and software traps to a handler that implements the appropriate SDELC recovery policy for instructions or data.

Once the trap handler has decided on a candidate codeword for recovery, it must correctly commit the state in the system such that it appears *as if* there was no memory control flow disruption. For instruction errors, because the error occurred during a fetch, the program counter (`pc`) has not yet advanced. To complete the trap handler, the candidate codeword is written back to instruction memory. If it is not accessible by the load/store unit, one could use hardware debug support such as JTAG. The previously trapped instruction is re-executed after returning from the trap handler, which will then cause the `pc` to advance and re-fetch the instruction that had been corrupted by the soft error. On the other hand, data errors are triggered from the memory pipeline stage by executing a load instruction. The chosen candidate codeword is written back to data memory to scrub the error, the register file is updated appropriately, and `pc` is manually advanced before returning from the trap handler.

2.5 Soft Fault Recovery Using SDELC

To evaluate SDELC, Spike was modified to produce representative memory access traces of 11 benchmarks as they run to completion. Five benchmarks are `blowfish` and `sha` from the MiBench suite [14] as well as `dhryystone`, `matmulti`, and `whetstone`. The remaining six benchmarks were added from the AxBench approximate computing C/C++ suite [37]: `blackscholes`, `fft`, `inversek2j`, `jmeint`, `jpeg`, and `sobel`. Each trace was analyzed offline using a MATLAB model of SDELC. For each workload, 1000 instruction fetches and 1000 data reads were randomly selected from the trace and exhaustively all possible single-bit faults were applied to each of them.

SDELC recovery of the random soft faults was evaluated using three different UL-ELC codes ($r = 1, 2, 3$). Recall that the $r = 1$ code is simply a single parity bit, resulting in 33 candidate codewords. (For basic parity, there are 32 message bits and one parity bit, so there are 33 ways to have had a single-bit error.) For the data memory, the UL-ELC codes were designed with the chunks being equally sized: for $r = 2$, there are either 11 or 12 candidates depending on the fault position (34 bits divided into three chunks), while for $r = 3$ there are always five candidates (35 bits divided into seven chunks). For the instruction memory, chunks are aligned to important field divisions in the RV64G ISA. Chunks for the $r = 2$ UL-ELC construction match the fields of the Type-U instruction codecs (the `opcode` being the unshared chunk). Chunks for the $r = 3$ UL-ELC code align with fields in the Type-R4 codec (as presented in Table 4). A *successful recovery* for SDELC occurs when the policy corrects the error; otherwise, it fails by accidentally mis-correcting.

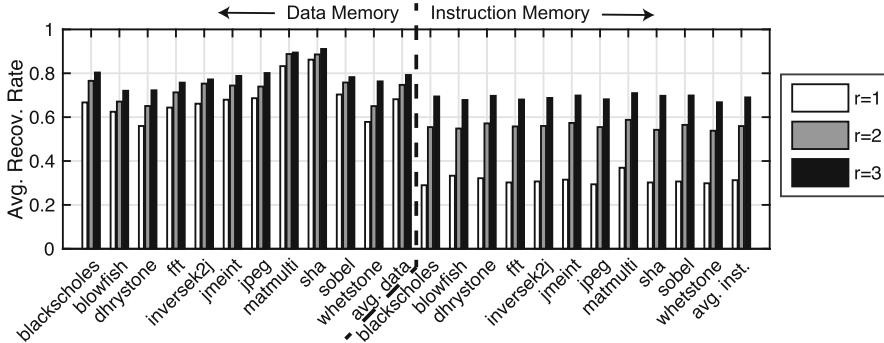


Fig. 6 Average rate of recovery using SDELCA from single-bit soft faults in instruction and data memory. r is the number of parity bits in the UL-ELC construction

2.5.1 Overall Results

The overall SDELCA results are presented in Fig. 6. The recovery rates are relatively consistent over each benchmark, especially for instruction memory faults, providing evidence of the general efficacy of SDELCA. One important distinction between the memory types is the sensitivity to the number r of redundant parity bits per message. For the data memory, the simple $r = 1$ parity yielded surprisingly high rates of recovery using our policy (an average of 68.2%). Setting r to three parity bits increases the average recovery rate to 79.2% thanks to fewer and more localized candidates to choose from. On the other hand, for the instruction memory, the average rate of recovery increased from 31.3% with a single parity bit to 69.0% with three bits.

These results are a significant improvement over a guaranteed system crash as is traditionally done upon error detection using single-bit parity. Moreover, these results are achieved using no more than half the overhead of a Hamming SEC code, which can be a significant cost savings for small IoT devices. Based on these results, using $r = 1$ parity for data seems reasonable, while $r = 3$ UL-ELC constructions can be used to achieve 70% recovery for both memories with minimal overhead.

3 Parity++ : Lightweight Error Correction for Last Level Caches and Embedded Memories

This section focuses on another novel lightweight error correcting code—Parity++: a novel lightweight unequal message protection scheme for last level caches or embedded memories that preferentially provides stronger error protection to certain “special messages.” As the name suggests, this coding scheme requires one extra bit above a simple parity Single-bit Error Detection (SED) code while providing SED

for all messages and Single-bit Error Correction (SEC) for a subset of messages. Thus, it is stronger than just basic SED parity and has much lower parity storage overhead ($3.5\times$ and $4\times$ lower for 32-bit and 64-bit memories, respectively) than a traditional Single-bit Error Correcting, Double-bit Error Detecting (SECDED) code. Error detection circuitry often lies on the critical path and is generally more critical than error correction circuitry as error occurrences are rare even with an increasing soft error rate. This coding scheme has a much simpler error detection circuitry that incurs lower energy and latency costs than the traditional SECDED code. Thus, Parity++ is a lightweight ECC code that is ideal for large capacity last level caches or lightweight embedded memories. Parity++ is also evaluated with a memory speculation procedure [8] that can be generally applied to any ECC protected cache to hide the decoding latency while reading messages when there are no errors.

Details of the concepts discussed in this section can be found in the work by Alam et al. [1] and Schoeny et al. [30].

3.1 Application Characteristics

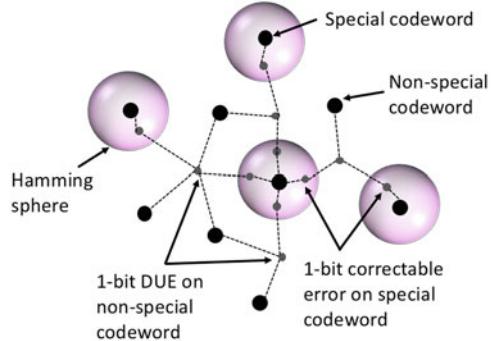
As mentioned in Sects. 1.3 and 2.2, data in applications is generally very structured and instructions mostly follow power-law distribution. This means most instructions in the memory would have the same opcode. Similarly, the data in the memory is usually low-magnitude signed data of a certain data type. However, these values get represented inefficiently, for e.g., 4-byte integer type used to represent values that usually need only 1-byte. Thus, in most cases, the MSBs would be a leading-pad of 0s or 1s. The approach of utilizing these characteristics in applications complements recent research on data compression in cache and main memory systems such as frequent value/pattern compression [3, 35], base-delta-immediate compression [27], and bit-plane compression [19]. However, the main goal here is to provide stronger error protection to these special messages that are chosen based on the knowledge of data patterns in context.

3.2 Parity++ Theory

Parity++ is a type of *unequal message protection* code, in that specific messages are designated a priori to have extra protection against errors as shown in Fig. 7. As in [30], there are two classes of messages, normal and special, and they are mapped to normal and special codewords, respectively. When dealing with the importance or frequency of the underlying data, it is referred to as messages; when discussing error detection/correction capabilities it is referred to as codewords.

Codewords in Parity++ have the following error protection guarantees: normal codewords have single-error detection; special codewords have single-error cor-

Fig. 7 Conceptual illustration of Parity++ for 1-bit error (CE = Correctable Error, DUE = Detected but Uncorrectable Error)



rection. Let us partition the codewords in the code C into two sets, \mathcal{N} and \mathcal{S} , representing the normal and special codewords, respectively. The minimum distance properties necessary for the aforementioned error protection guarantees of Parity++ are as follows:

$$\min_{\mathbf{u}, \mathbf{v} \in \mathcal{N}, \mathbf{u} \neq \mathbf{v}} d_H(\mathbf{u}, \mathbf{v}) \geq 2, \quad (3)$$

$$\min_{\mathbf{u} \in \mathcal{N}, \mathbf{v} \in \mathcal{S}} d_H(\mathbf{u}, \mathbf{v}) \geq 3, \quad (4)$$

$$\min_{\mathbf{u}, \mathbf{v} \in \mathcal{S}, \mathbf{u} \neq \mathbf{v}} d_H(\mathbf{u}, \mathbf{v}) \geq 3. \quad (5)$$

A second defining characteristic of the Parity++ code is that the length of a codeword is only two bits longer than a message, i.e., $n = k + 2$. Thus, Parity++ requires only two bits of redundancy.

For the context of this work, let us assume that Parity++ always has message length k as a power of 2. The overall approach to constructing the code is to create a Hamming subcode of a SED code [15]; when an error is detected, it is decoded to the neighboring special codeword. The overall code has $d_{min} = 2$, but a block in \mathbf{G} , corresponding to the special messages, has $d_{min} \geq 3$. For the sake of notational convenience, let us go through the steps of constructing the (34, 32) Parity++ code (as opposed to the generic $(k + 2, k)$ Parity++ code).

The first step is to create the generating matrix for the Hamming code whose message length is at least as large as the message length in the desired Parity++ code; in this case, the (63, 57) Hamming code is used. Let α be a primitive element of $GF(2^6)$ such that $1+x+x^6 = 0$, then the generator polynomial is simply $g_S(x) = 1 + x + x^6$ (and the generator matrix is constructed using the usual polynomial coding methods). The next step is to shorten this code to (32, 26) by expurgating and puncturing (i.e., deleting) the right and bottom 31 columns and rows. Then add a column of 1s to the end, resulting in a generator matrix, which is denoted as \mathbf{G}_S , for a (33, 26) code with $d_{min} = 4$.

For the next step in the construction of the generating matrix of the (34, 32) Parity++ code, \mathbf{G}_N is added on top of \mathbf{G}_S , where \mathbf{G}_N is the first 6 rows of the generator matrix using the generator polynomial $g_N(x) = 1 + x$, with an appended row of 0s at the end. Note that \mathbf{G}_N is the generator polynomial of a simple parity-check code. By using this polynomial subcode construction, a generator matrix is built with overall $d_{min} = 2$, with the submatrix \mathbf{G}_S having $d_{min} = 4$. At this point, notice that messages that begin with 6 0s only interact with \mathbf{G}_S ; these messages will be the special messages. Note that Conditions 3 and 5 are satisfied; however, Condition 4 is not satisfied. To meet the requirement, a single non-linear parity bit is added that is a NOR of the bits corresponding to \mathbf{G}_N , in this case, the first 6 bits.

The final step is to convert \mathbf{G}_S to systematic form via elementary row operations. Note that these row operations preserve all 3 of the required minimum distance properties of Parity++. As a result, the special codewords (with the exception of the known prefix) are in systematic form. For example, in the (34, 32) Parity++ code, the first 26 bits of a special codeword are simply the 26 bits in the message (not including the leading run of 6 0s).

At the encoding stage of the process, when the message is multiplied by \mathbf{G} , the messages denoted as special must begin with a leading run of $\log_2(k) + 1$ 0's. However, the original messages that are deemed to be special do not have to follow this pattern as one can simply apply a pre-mapping before the encoding step, and a post-mapping after the decoding step.

In the (34, 32) Parity++ code, observe that there are 2^{26} special messages. Generalizing, it is easy to see that for a $(k + 2, k)$ Parity++ code, there are $2^{k - \log_2(k) - 1}$ special messages.

Similar unequal message protection scheme can be used for providing DECTED protection to special messages, while non-special messages get SECDED protection. The code construction has been explained in detail in [30].

3.3 Error Detection and Correction

The received—possibly erroneous—vector \mathbf{y} is divided into two parts, $\bar{\mathbf{c}}$ and η , with $\bar{\mathbf{c}}$ being the first $k+1$ bits of the codeword and η the additional non-linear redundancy bit ($\eta = 0$ for special messages and $\eta = 1$ for normal messages). There are three possible scenarios at the decoder: no (detectable) error, correctable error, or detected but uncorrectable error.

First, due to the Parity++ construction, every valid codeword has even weight. Thus, if $\bar{\mathbf{c}}$ has even weight, then the decoder concludes no error has occurred, i.e., $\bar{\mathbf{c}}$ was the original codeword. Second, if $\bar{\mathbf{c}}$ has odd weight and $\eta = 0$, the decoder attempts to correct the error. Since \mathbf{G}_S is in systematic form, \mathbf{H}_S , its corresponding parity-check matrix can be easily retrieved. The decoder calculates the syndrome $\mathbf{s}_1 = \mathbf{H}_S^T \bar{\mathbf{c}}$. If \mathbf{s}_1 is equal to a column in \mathbf{H}_S , then that corresponding bit in $\bar{\mathbf{c}}$ is flipped. Third, if $\bar{\mathbf{c}}$ has odd weight and either \mathbf{s}_1 does not correspond to any column in \mathbf{H}_S or $\eta = 1$, then the decoder declares a DUE.

The decoding process described above guarantees that any single-bit error in a special codeword will be corrected, and any single-bit error in a normal codeword will be detected (even if the bit in error is η).

Let us take a look at two concrete examples for the (10, 8) Parity++ code. Without any pre-mapping, a special message begins with $\log_2(3) + 1 = 4$ zeros. Let the original message be $\mathbf{m} = (00001011)$, which is encoded to $\mathbf{c} = (1011010110)$. Note that the first 4 bits of \mathbf{c} is the systematic part of the special codeword. After passing through the channel, let the received vector be $\mathbf{y} = (1001010110)$, divided into $\bar{\mathbf{c}} = (1001010110)$ and $\eta = 0$. Since the weight of c is odd and $\eta = 0$, the decoder attempts to correct the error. The syndrome is equal to the 3rd column in \mathbf{H}_S , thus the decoder correctly flips the 3rd bit of $\bar{\mathbf{c}}$.

For the second example, let us begin with $\mathbf{m} = (11010011)$, which is encoded to (0011111101) . After passing through the channel, the received vector is $\mathbf{y} = (0011011101)$. Since the weight of $\bar{\mathbf{c}}$ is odd and $\eta = 1$, the decoder declares a DUE. Note that for both normal and special codewords, if the only bit in error is η itself, then it is implicitly corrected since $\bar{\mathbf{c}}$ has even weight and will be correctly mapped back to \mathbf{m} without any error detection or correction required.

3.4 Architecture

In an ECC protected cache, every time a cache access is initiated, the target block is sent through the ECC decoder/error detection engine. If no error is detected, the cache access is completed and the cache block is sent to the requester. If an error is detected, the block is sent through the ECC correction engine and the corrected block is eventually sent to the requester. Due to the protection mechanism, there is additional error detection/correction latency. Error detection latency is more critical than error correction as occurrence of an error is a rare event when compared to the processor cycle time and does not fall in the critical path. However, a block goes through the detection engine every time a cache access is initiated.

When using Parity++, the flow almost remains the same. Parity++ can detect all single-bit errors but has correction capability for “special messages.” When a single-bit flip occurs on a message, the error detection engine first detects the error and stalls the pipeline. If the non-linear bit says it is a “special message” (non-linear bit is ‘0’), the received message goes through the Parity++ error correction engine which outputs the corrected message. This marks the completion of the cache access. If the non-linear bit says it is a non-special message (non-linear bit is “1”), it is checked if the cache line is clean. If so, the cache line is simply read back from the lower level cache or the memory and the cache access is completed. However, if the cache line is dirty and there are no other copies of that particular cache line, it leads to a crash or a roll back to checkpoint. Note that both Parity++ and SECDED have equal decoding latency of one cycle that is incurred during every read operation from an ECC protected cache. The encoding latency during write operation does not fall in the critical path and hence is not considered in the analyses.

The encoding energy overhead is almost similar for both Parity++ and SECDED. The decoding energy overheads are slightly different. For SECDED, the original message can be retrieved from the received codeword by simply truncating the additional ECC redundant bits. However, all received codewords need to be multiplied with the H-matrix to detect if any errors have occurred. For Parity++, all messages go through the chain of XOR gates for error detection and only the non-systematic non-special messages need to be multiplied with the decoder matrix to retrieve the original message. Since the error detection in Parity++ is much cheaper in terms of energy overhead than SECDED and the non-special messages only constitute about 20–25% of the total messages, the overall read energy in Parity++ turns out to be much lesser than SECDED.

3.5 Experimental Methodology

Parity++ was evaluated over applications from the SPEC 2006 benchmark suite. Two sets of core micro-architectural parameters (provided in Table 5) were chosen to understand the performance benefits in both a lightweight in-order (InO) processor and a larger out-of-order (OoO) core. Performance simulations were run using Gem5 [6], fast forwarding for one billion instructions and executing for two billion instructions.

The first processor was a lightweight single in-order core architecture with a 32kB L1 cache for instruction and 64kB L1 cache for data. Both the instruction and data caches were 4-way associative. The LLC was a unified 1MB 8-way associative L2 cache. The second processor was a dual core out-of-order architecture. The L1 instruction and data caches had the same configuration as the previous processor. The LLC comprises of both L2 and L3 caches. The L2 was a shared 512KB cache while the L3 was a shared 2MB 16-way associative cache. For both the baseline processors it was assumed that the LLCs (L2 for the InO processor and L2 and L3 for the OoO processor) have SECDED ECC protection.

Table 5 Core micro-architectural parameters

	Processor-1	Processor-2
Cores	1 (@ 2 GHz)	2 (@ 2 GHz)
Core type	InO (@ 2 GHz)	OoO (@ 2 GHz)
Cache line size	64B	64B
L1 Cache per core	32 KB I\$, 64 KB D\$	32 KB I\$, 64 kB D\$
L2 Cache	1 MB (unified) 8-way	512 KB (shared, unified) 8-way
L3 Cache	—	2 MB 16-way (shared)
Memory configuration	4 GB of 2133 MHz DDR3	8 GB of 2133 MHz DDR3
Nominal voltage	1 V	1 V

The performance evaluation was done only for cases where there are no errors. Thus, latency due to error detection was taken into consideration but not error correction as correction is rare when compared to the processor cycle time and does not fall in the critical path. In order to compare the performance of the systems with Parity++ against the baseline cases with SECDED ECC protection, the size of the LLCs was increased by $\sim 9\%$ due to the lower storage overhead of Parity++ compared to SECDED. This is the iso-area case since the additional area coming from reduction in redundancy is used to increase the total capacity of the last level caches.

3.6 Results and Discussion

In this section the performance results obtained from the Gem5 simulations (as mentioned in Sect. 3.5) are discussed. Figures 8 and 9 show the comparative results for the two different sets of core micro-architectures across a variety of benchmarks from the SPEC2006 suite when using memory speculation. In both the evaluations, performance of the system with Parity++ was compared against that with SECDED.

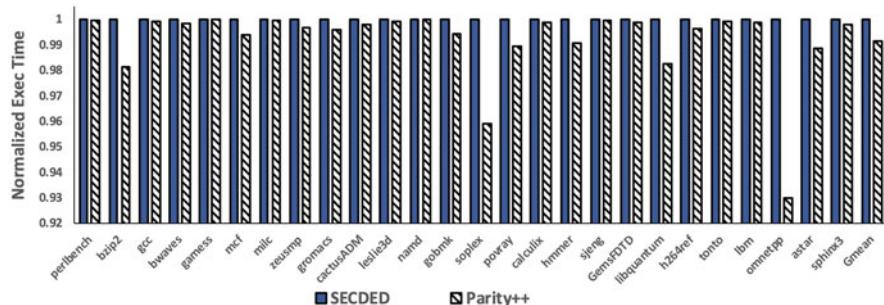


Fig. 8 Comparing normalized execution time of Processor-I with SECDED and Parity++

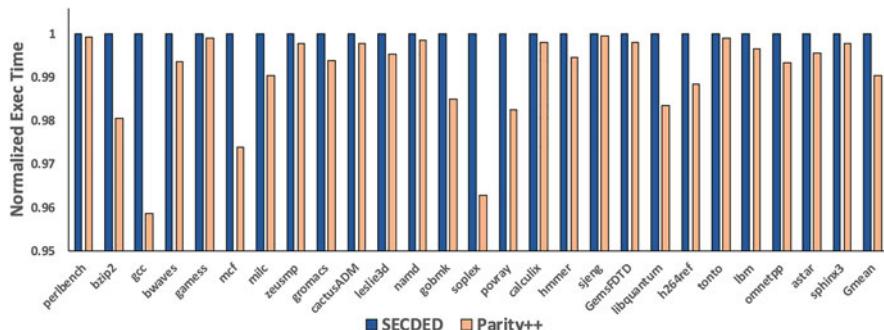


Fig. 9 Comparing normalized execution time of Processor-II with SECDED and Parity++

For both the core configurations, the observations are almost similar. It was considered that both Parity++ and SECDED protected caches have additional cache hit latency of one cycle (due to ECC decoding) for all read operations. The results show that with the exact same hit latency, Parity++ has up to 7% lower execution time than SECDED due to the additional memory capacity. The applications showing higher performance benefits are mostly memory intensive. Hence, additional cache capacity with Parity++ reduces overall cache miss rate. For most of these applications, this performance gap widens as the LLC size increases for Processor-II. The applications showing roughly similar performances on both the systems are the ones which already have a considerably lower LLC miss rate. As a result, increase in LLC capacity due to Parity++ does not lead to a significant improvement in performance.

On the other hand, if the cache capacity is kept constant (iso-capacity), Parity++ helps to save ~5–9% of last level cache area (cache tag area taken into consideration) as compared to SECDED. Since the LLCs constitute more than 30% of the processor chip area, the cache area savings translate to a considerable amount of reduction in the chip size. This additional area benefit can either be utilized to make an overall smaller sized chip or it can be used to pack in more compute tiles to increase the overall performance of the system.

The results also imply that Parity++ can be used in SRAM based scratchpad memories used in embedded systems at the edge of the Internet-of-Things (IoT) where hardware design is driven by the need for low area, cost, and energy consumption. Since Parity++ helps in reducing area (in turn reducing SRAM leakage energy) and also has lower error detection energy [1], it provides a better protection mechanism than SECDED in such devices.

Acknowledgments The authors thank Dr. Mark Gottscho and Dr. Clayton Schoeny who were collaborators on the work presented in this chapter.

References

1. Alam, I., Schoeny, C., Dolecek, L., Gupta, P.: Parity++: lightweight error correction for last level caches. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 114–120 (2018). <https://doi.org/10.1109/DSN-W.2018.00048>
2. Alameldeen, A., Wood, D.: Frequent pattern compression: a significance-based compression scheme for L2 caches. Tech. rep., University of Wisconsin, Madison (2004)
3. Alameldeen, A.R., Wood, D.A.: Frequent pattern compression: a significance-based compression scheme for L2 caches (2004)
4. Bathen, L.A.D., Dutt, N.D.: E-RoC: embedded RAIDs-on-chip for low power distributed dynamically managed reliable memories. In: Design, Automation, and Test in Europe (DATE) (2011)
5. Bathen, L.A.D., Dutt, N.D., Nicolau, A., Gupta, P.: VaMV: variability-aware memory virtualization. In: Design, Automation, and Test in Europe (DATE) (2012)

6. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. *SIGARCH Comput. Archit. News* **39**(2), 1–7 (2011). <http://doi.acm.org/10.1145/2024716.2024718>
7. Davyдов, A., Tombak, L.: An alternative to the Hamming code in the class of SEC-DED codes in semiconductor memory. *IEEE Trans. Inf. Theory* **37**(3), 897–902 (1991)
8. Duwe, H., Jian, X., Kumar, R.: Correction prediction: reducing error correction latency for on-chip memories. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 463–475 (2015). <https://doi.org/10.1109/HPCA.2015.7056055>
9. Gelas, J.D.: The Intel Xeon E5 v4 review: testing broadwell-EP with demanding server workloads (2016). <http://www.anandtech.com/show/10158/the-intel-xeon-e5-v4-review>
10. Gottscho, M., Bathen, L.A.D., Dutt, N., Nicolau, A., Gupta, P.: ViPZonE: hardware power variability-aware memory management for energy savings. *IEEE Trans. Comput.* **64**(5), 1483–1496 (2015)
11. Gottscho, M., Schoeny, C., Dolecek, L., Gupta, P.: Software-defined error-correcting codes. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), pp. 276–282 (2016)
12. Gottscho, M., Schoeny, C., Dolecek, L., Gupta, P.: Software-defined ECC: heuristic recovery from uncorrectable memory errors. Tech. rep., University of California, Los Angeles, Oct 2017
13. Gottscho, M., Alam, I., Schoeny, C., Dolecek, L., Gupta, P.: Low-cost memory fault tolerance for IoT devices. *ACM Trans. Embed. Comput. Syst.* **16**(5s), 128:1–128:25 (2017)
14. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of the IEEE International Workshop on Workload Characterization (IWWC) (2001)
15. Hamming, R.W.: Error detecting and error correcting codes. *Bell Labs Tech. J.* **29**(2), 147–160 (1950)
16. Hsiao, M.Y.: A class of optimal minimum odd-weight-column SEC-DED codes. *IBM J. Res. Dev.* **14**(4), 395–401 (1970)
17. Kaneda, S., Fujiwara, E.: Single byte error correcting – double byte error detecting codes for memory systems. *IEEE Trans. Comput.* **C-31**(7), 596–602 (1982)
18. Kim, J., Sullivan, M., Choukse, E., Erez, M.: Bit-plane compression: transforming data for better compression in many-core architectures. In: Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA) (2016)
19. Kim, J., Sullivan, M., Choukse, E., Erez, M.: Bit-plane compression: transforming data for better compression in many-core architectures. In: Proceedings of the 43rd International Symposium on Computer Architecture, ISCA '16, pp. 329–340. IEEE, Piscataway (2016). <https://doi.org/10.1109/ISCA.2016.37>
20. Lipasti, M.H., Wilkerson, C.B., Shen, J.P.: Value locality and load value prediction. In: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (1996)
21. Liu, S., Pattabiraman, K., Moscibroda, T., Zorn, B.G.: Flikker: saving DRAM refresh-power through critical data partitioning. In: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2011)
22. Miguel, J.S., Badr, M., Jerger, N.E.: Load value approximation. In: Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO) (2014)
23. Miguel, J.S., Albericio, J., Jerger, N.E., Jaleel, A.: The Bunker Cache for spatio-value approximation. In: Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO) (2016)
24. Mittal, S., Vetter, J.: A survey of architectural approaches for data compression in cache and main memory systems. *IEEE Trans. Parallel Distrib. Syst.* **27**(5), 1524–1536 (2015)
25. Nguyen, Q.: RISC-V tools (GNU toolchain, ISA simulator, tests) – git commit 816a252. <https://github.com/riscv/riscv-tools>

26. Pekhimenko, G., Seshadri, V., Mutlu, O., Gibbons, P.B., Kozuch, M.A., Mowry, T.C.: Base-delta-immediate compression: practical data compression for on-chip caches. In: Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT) (2012)
27. Pekhimenko, G., Seshadri, V., Mutlu, O., Gibbons, P.B., Kozuch, M.A., Mowry, T.C.: Base-delta-immediate compression: practical data compression for on-chip caches. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, PACT '12, pp. 377–388. ACM, New York (2012). <http://doi.acm.org/10.1145/2370816.2370870>
28. Pekhimenko, G., Seshadri, V., Kim, Y., Xin, H., Mutlu, O., Gibbons, P.B., Kozuch, M.A., Mowry, T.C.: Linearly compressed pages: a low-complexity, low-latency main memory compression framework. In: Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO) (2013)
29. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: EnerJ: approximate data types for safe and general low-power computation. In: Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI) (2011)
30. Schoeny, C., Sala, F., Gottscho, M., Alam, I., Gupta, P., Dolecek, L.: Context-aware resiliency: unequal message protection for random-access memories. In: Proc. IEEE Information Theory Workshop, Kaohsiung, pp. 166–170, Nov 2017
31. Song, J., Bloom, G., Palmer, G.: SuperGlue: IDL-based, system-level fault tolerance for embedded systems. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2016)
32. Waterman, A.: RISC-V proxy kernel – git commit 85ae17a. <https://github.com/riscv/riscv-pk/commit/85ae17a>
33. Waterman, A., Lee, Y.: Spike, a RISC-V ISA Simulator – git commit 3bfc00e. <https://github.com/riscv/riscv-isa-sim>
34. Waterman, A., Lee, Y., Patterson, D., Asanovic, K.: The RISC-V instruction set manual volume I: user-level ISA version 2.0 (2014). <https://riscv.org>
35. Yang, J., Zhang, Y., Gupta, R.: Frequent value compression in data caches. In: Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO) (2000)
36. Yazdanbakhsh, A., Pekhimenko, G., Thwaites, B., Esmaeilzadeh, H., Mutlu, O., Mowry, T.C.: Mitigating the memory bottleneck with approximate load value prediction. IEEE Des. Test **33**(1), 32–42 (2016)
37. Yazdanbakhsh, A., Mahajan, D., Esmaeilzadeh, H., Lotfi-Kamran, P.: AxBench: a multiplatform benchmark suite for approximate computing. IEEE Des. Test **34**(2), 60–68 (2017)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Resource Management for Improving Overall Reliability of Multi-Processor Systems-on-Chip



Yue Ma, Junlong Zhou, Thidapat Chantem, Robert P. Dick, and X. Sharon Hu

1 Introduction

This section presents the concepts and models associated with soft-error reliability and lifetime reliability, and reviews the related work on these topics.

1.1 Background

Modern multi-processor systems on a chip (MPSoCs) may contain both multicore processors and integrated GPUs, which are especially suitable for real-time embedded applications requiring massively parallel processing capabilities. Since MPSoCs

Y. Ma · X. S. Hu (✉)

Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

e-mail: yma1@nd.edu; shu@nd.edu

J. Zhou

School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

e-mail: jzhou@njust.edu.cn

T. Chantem

Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Arlington, VA, USA

e-mail: tchantem@vt.edu

R. P. Dick

Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

e-mail: dickrp@umich.edu

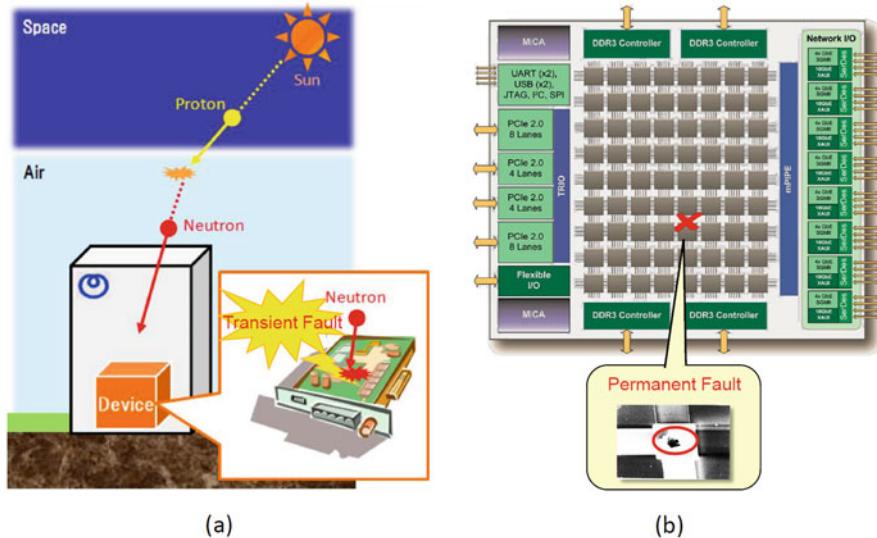


Fig. 1 Illustration of transient and permanent faults

offer good performance and power consumption, they have been widely used in many real-time applications such as consumer electronics, automotive electronics, industrial automation, and avionics [1]. For these applications, the MPSoC needs to satisfy deadline, quality-of-service (e.g., resolution of video playback), and reliability requirements. The reliability requirements include both soft-error reliability (SER), influenced by transient faults, and lifetime reliability (LTR), influenced by permanent faults. This chapter presents approaches to improving SER and/or LTR while satisfying deadline and quality-of-service requirements for real-time embedded systems.

Transient faults are mainly caused by high-energy particle strikes, e.g., resulting from spallation from cosmic rays striking atoms in the upper atmosphere [2] (see Fig. 1a). Transient faults may lead to errors that appear for a short time and then disappear without damaging the device or shortening its lifetime; these are called soft errors. They may prevent tasks from completing successfully. SER is used to quantify the probability that tasks will complete successfully without errors due to transient faults. SER can be increased by using reliability-aware techniques such as replication, rollback recovery, and frequency elevation, which either tolerate transient faults or decrease their rates.

Permanent faults are caused by wear in integrated circuits. An example is illustrated in Fig. 1b. Permanent faults can lead to errors that persist until the faulty hardware is repaired or replaced. Multiple wear-out effects such as electromigration (EM), stress migration (SM), time-dependent dielectric breakdown (TDDB), and thermal cycling (TC) can lead to permanent faults. The rates of these effect depend exponentially on temperature. In addition, thermal cycling depends on the

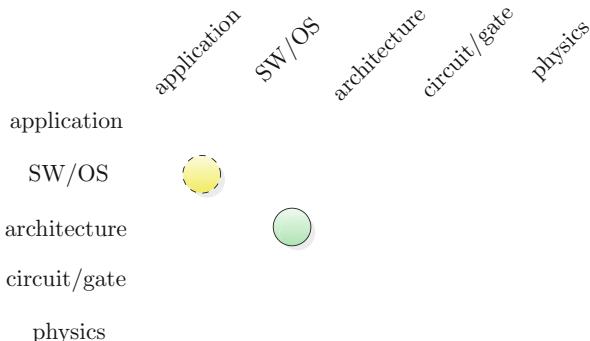


Fig. 2 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

temperature range, maximum temperature, and cycle frequency. To improve LTR, temperature peaks and variation must be limited.

To reduce the cost of repairing/replacing an MPSoC system and maintain some desired level of quality-of-service, improving SER due to transient faults and LTR due to permanent faults become an imperative design concern. In this chapter we present two techniques that optimize SER and LTR separately and show how to make appropriate trade-offs between them for improving overall system reliability. Figure 2 illustrates the abstraction layers representing the main contribution of this chapter.

1.2 Related Work

Considerable research has been done on improving SER. Haque et al. [3] present an energy-efficient task replication method to achieve a high SER target for periodic real-time applications running on a multicore system with minimum energy consumption. Salehi et al. [4] propose a low-overhead checkpointing-based rollback recovery scheme to increase system SER and reduce the number of checkpoints for fault-tolerant real-time systems. Zhou et al. [5] improve system SER by judiciously determining proper replication and speedup of tasks. Zhou and Wei [6] describe a stochastic fault-tolerant task scheduling algorithm that specifically considers uncertainty in task execution caused by transient fault occurrences to increase SER under task deadline constraints. These work increase SER but do not consider permanent faults.

Many studies have focused on increasing LTR. Huang et al. [7] describe an analytical model to derive the LTR of multicore systems and a simulated annealing algorithm to reduce core temperature and temperature variation to improve system LTR. Chantem et al. [8] present a dynamic task assignment and scheduling scheme to maximize system LTR by mitigating core wear due to thermal cycling. Ma et

al. [1] optimize system LTR by establishing an online framework that dynamically controls cores' utilization. Das et al. [9, 10] improve the LTR of network-on-chips (NoCs) and also solve the energy-reliability trade-off problem for multimedia MPSoCs. However, these approaches neglect transient faults.

There is research on handling SER and LTR together. Zhou et al. [5] propose a task frequency and replication selection strategy that balances SER and LTR to maximize system availability. Ma et al. [11] establish an online framework for increasing SER and LTR of real-time systems running on “big–little” type MPSoCs. A genetic algorithm based approach [12] that determines task mappings and frequencies is developed to jointly improve SER and LTR. Aliee et al. [13] adopt mean time to failure (MTTF) as the common metric to evaluate SER and LTR and design a success tree based scheme for reliability analysis for embedded systems. Unlike work [5, 11–13] that ignore the variations in performance, power consumption, and reliability parameters, Gupta et al. [14] explore the possibility of constructing reliable systems to compensate for the variability effects in hardware through software controls. These efforts consider CPU reliability but ignore the reliability effects of GPUs.

1.3 Soft-Error Reliability Model

SER is the probability that no soft errors occur in a particular time interval [5], i.e.,

$$r = e^{-\lambda(f) \times U \times |\Delta t|}, \quad (1)$$

where f is the core frequency, $|\Delta t|$ is the length of the time interval, U is the core's utilization within $|\Delta t|$, and $\lambda(f)$ is the average fault rate depending on f [5]. Specifically, we have

$$\lambda(f) = \lambda_0 \times 10^{\frac{d(f_{\max} - f)}{f_{\max} - f_{\min}}}, \quad (2)$$

where λ_0 is the average fault rate at the maximum core frequency. f_{\min} and f_{\max} are the minimum and maximum core frequency, and d ($d > 0$) is a hardware-specific constant indicating the sensitivity of fault rate to frequency scaling. Reducing frequency leads to an exponential increase in fault rate because frequency is a roughly linear function of supply voltage. As frequency reduces, supply voltage decreases, decreasing the critical charge (i.e., the minimum amount of charge that must be collected by a circuit to change its state) and exponentially increasing fault rate [15].

Since CPU and GPU fabrication processes are similar, the device-level SER model above applies to both. Let r_G and r_i ($i = 1, 2, \dots, m$) represent the SER of the GPU and the i th CPU core, respectively. As the correct operation of an MPSoC system-level depends on the successful execution of GPU and CPU cores, the system-level SER is calculated as the product of reliabilities of all individual cores, i.e.,

$$R = r_G \times \prod_{i=1}^m r_i. \quad (3)$$

1.4 Lifetime Reliability Model

MTTF is commonly used to quantify LTR. We focus on four main failure mechanisms: EM, TDDB, SM, and TC. EM refers to the dislocation of metal atoms caused by momentum imparted by electrical current in wires and vias [16]. TDDB refers to the deterioration of the gate oxide layer [17]. SM is caused by the directionally biased motion of atoms in metal wires due to mechanical stress caused by thermal mismatch between metal and dielectric materials [18]. TC is wear due to thermal stress induced by mismatched coefficients of thermal expansion for adjacent material layers [19].

The system-level MTTF modeling tool introduced by Xiang et al. [20] can be used to estimate LTR when considering the above four failure mechanisms. This tool integrates three levels of models, i.e., device-, component-, and system-level models. At the device level, wear due to the above four mechanisms is modeled. The modeling tool accounts for the effect of using multiple devices in a component upon fault distributions, e.g., the effects of EM are most appropriately modeled using a lognormal distribution at the device level, but with a Weibull distribution for components containing many devices. Based on the device-level reliability models and temporal failure distributions, component-level MTTF is calculated [20]. Then, based on component-level reliability, the system-level MTTF is obtained by Monte Carlo simulation.

2 LTR and SER Optimization

This section introduces two approaches for LTR and SER optimization, and discusses the trade-off between them.

2.1 LTR Optimization

EM, SM, and TDDB wear rates depend exponentially on temperature. However, wear due to thermal cycling depends on the amplitude (i.e., the difference between the proximal peak and valley temperature), period, and maximum temperature of thermal cycles. Figure 3 summarizes some system MTTF data obtained from the system-level LTR modeling tool with default settings [20]. Figure 3a–c depicts the MTTF of an example system as a function of the amplitude, period, and peak

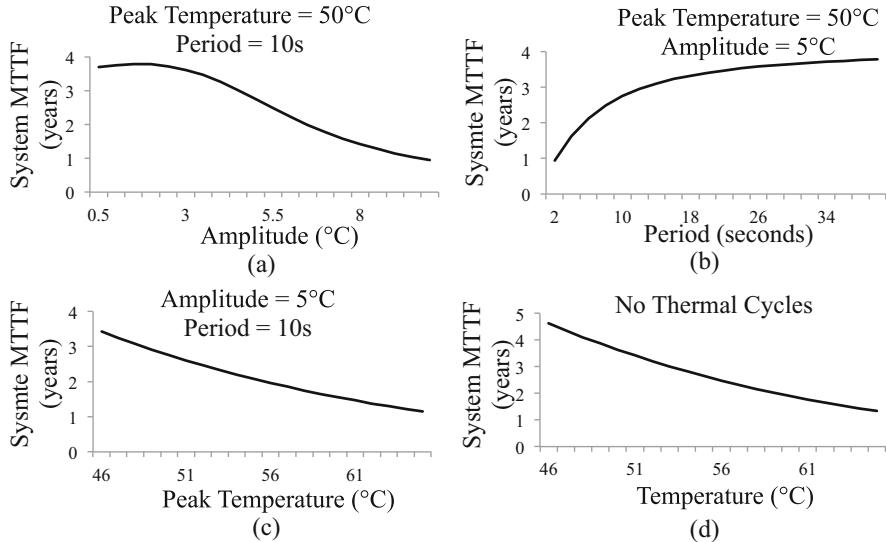


Fig. 3 System MTTF due to: (a) amplitude of thermal cycle; (b) period of thermal cycles; (c) peak temperature of thermal cycles; and (d) temperature without thermal cycles

temperature of thermal cycles, respectively. As a comparison, Fig. 3d shows the system MTTF due to temperature alone without thermal cycles. As can be seen from Fig. 3, system MTTF generally increases for lower temperatures and smaller thermal cycles.

A system's LTR is determined by its operating temperature and thermal cycles. Given that lower frequencies and voltages lead to higher utilization but lower temperatures, one method to improve system MTTF is to control core utilization. For example, we have developed a framework called Reliability-Aware Utilization Control (RUC) [21] to mitigate the effects of both operating temperature and thermal cycling. RUC consists of two controllers. The first controller reduces the peak temperature by periodically reducing core frequencies subject to task deadline requirements. Although frequent changes in core frequency helps to reduce peak temperature, they may increase the frequency of thermal cycling and reduce lifetime reliability. Hence, the second controller minimizes thermal cycling wear by dynamically adjusting the period of the first controller to achieve longer thermal cycles as well as lower peak temperature.

2.2 SER Optimization

Recovery allocation strategies and task execution orders can affect system-level soft-error reliability (as shown in Fig. 4). In this example, there are four tasks that

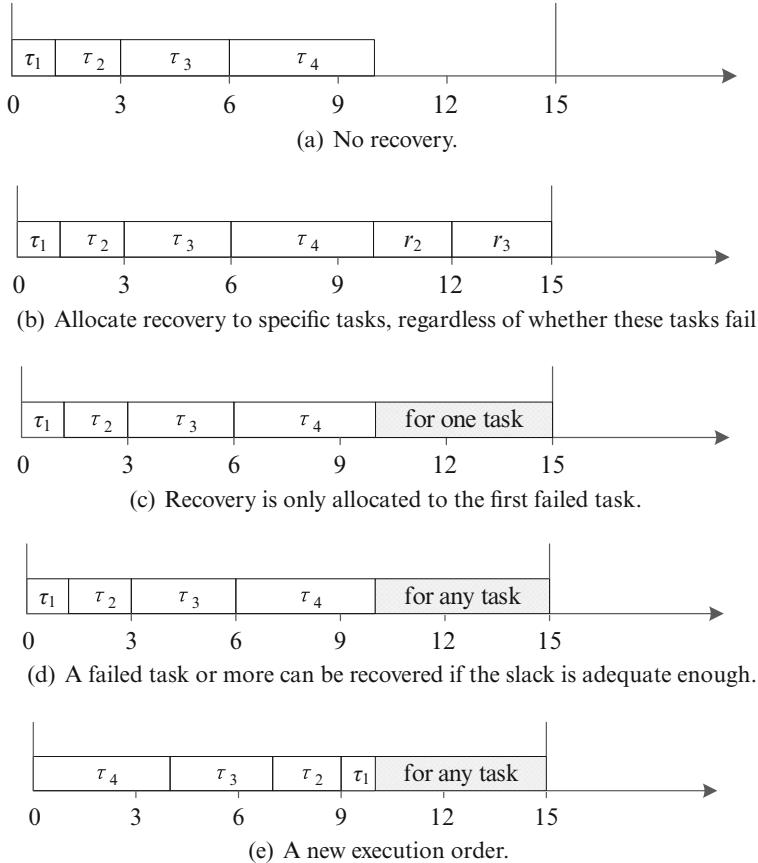


Fig. 4 Motivating examples illustrate different recovery allocation strategies and task execution order affect system-level SER. (a) No recovery. (b) Statically allocate recovery to specific tasks, regardless of whether these tasks fail. (c) Recovery is only allocated to the first failed task. (d) A failed task or more can be recovered if the slack is adequate. (e) A new task execution order

share a common period of 15 s. We further suppose the worst-case execution times of the tasks are 1, 2, 3, and 4 s. All tasks in the set execute at the highest core frequency. As indicated by the reliability model presented in Sect. 1, the SERs of the tasks are 0.904, 0.819, 0.741, and 0.670.

If no recovery is allowed as shown in Fig. 4a, the system-level SER, i.e., the probability that all tasks can complete successfully, is 0.368. Allowing recovery of some tasks increases SER. One method is to allocate recoveries to tasks offline [5]. Figure 4b represents a better solution for maximizing the system-level SER, in which tasks τ_2 and τ_3 have recoveries r_2 and r_3 . In this case, the system-level SER is 0.547. Another approach allocates recovery online [22]. Figure 4c shows a scenario where the first failed task has a recovery [22]. The system-level SER is 0.686, which

is higher than that in Fig. 4b. However, although the slack is dynamically used in Fig. 4c, only one task can be recovered.

In the above online recovery allocation example, a failed task is recovered if the remaining slack is adequate, and tasks consume slack on a first-come, first-served basis (see Fig. 4d). For example, task τ_2 can recover even if τ_1 fails. However, task τ_3 cannot recover if both τ_1 and τ_2 fail because the remaining slack for τ_3 is only 2 s. Task τ_4 can recover only when all tasks succeed or only τ_1 fails. Hence, the probabilities of recovering τ_3 and τ_4 are 0.983 and 0.607, and the system-level SER is 0.716. Now, consider the impact of task scheduling on the system-level SER. Figure 4e represents a new schedule where the task's priority is the inverse of its execution time. In this case, the probabilities of recovering τ_1 , τ_2 , τ_3 , and τ_4 are 0.792, 0.670, 0.670, and 1.000. In contrast with Fig. 4d, the task with the lowest SER, τ_4 , can always be recovered, but the system-level SER is 0.692. Hence, a scheduling algorithm that simply improves the probability of recovery for some specific tasks may not be a good solution.

Based on these observations, we design an SER improvement framework [23] that statically schedules tasks and dynamically allocates recoveries. The framework is composed of a simple and fast scheduling algorithm for special task sets and a powerful scheduling algorithm for general task sets. For more details of the two scheduling algorithms, readers can refer to [23].

3 Trade-Off Between LTR and SER

Certain design decisions (e.g., task mapping and voltage scaling) may increase LTR but decrease SER, and vice versa. In other words, improving overall reliability requires trade-offs between LTR and SER. Recently, several efforts have focused on these trade-offs. Below, we describe two case studies in LTR and SER trade-off: (1) “big–little” type MPSoCs and (2) CPU–GPU integrated MPSoCs.

3.1 “Big–Little” MPSoCs

To address power/energy concerns, various heterogeneous MPSoCs have been introduced. A popular MPSoC architecture often used in power/energy-conscious real-time embedded applications is composed of pairs of high-performance (HP) cores and low-power (LP) cores. Such HP and LP cores present unique performance, power/energy, and reliability trade-offs. Following the terminology introduced by ARM, we refer to this as the “big–little” architecture. Nvidia’s variable symmetric multiprocessing architecture is such an example [24].

Executing tasks on an LP core improves LTR by reducing temperature and improves SER through a higher core frequency. Although the primary goal of “big–little” MPSoCs is to reduce power consumption by executing a light workload on

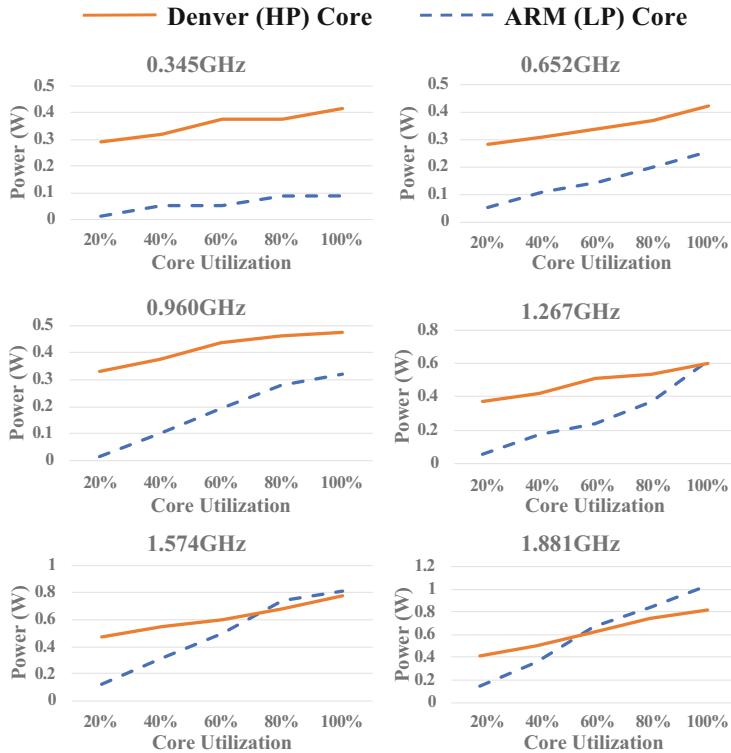


Fig. 5 The measured power consumptions of HP (Denver) core and LP (ARM) core on Nvidia's TX2 chip as functions of utilization and frequency

the LP cores, there are circumstances in which an LP core consumes more power than an HP core. Carefully characterizing the power consumption behavior of HP and LP cores is necessary. For example, the power consumption of the HP core and LP core on Nvidia's TX2¹ is shown in Fig. 5. The LP core consumes less power than the HP core only when the core frequency is low and the workload is light. One possible reason for this phenomenon is that the HP and LP cores have different microarchitectures, as is the case with the TX2. Another possible reason is that the transistors in the HP core and LP core have different threshold voltages. The LP core has low leakage power but requires high voltage to operate at higher frequencies. On the other hand, the HP core can work at high frequency with a low voltage.

The above observations reveal that in order to reduce power consumption of MPSoCs and improve reliability, it is necessary to fully account for the power

¹Note that TX2 is composed of ARM Cortex A57 cores that support multithreading, and Nvidia's Denver cores for high single-thread performance with dynamic code optimization. Denver cores can be treated as HP cores and ARM cores can be treated as LP cores when running single-threaded applications.

features of heterogeneous cores and carefully map tasks to the most appropriate cores. A good guideline is to run tasks having short execution times on LP cores with low frequencies and tasks having long execution times on HP cores with high frequencies. The execution models of HP and LP cores must also be considered. For example, HP and LP cores on Nvidia's TK1 cannot execute at the same time. However, on Nvidia's TX2, HP and LP cores can work simultaneously. Although an HP core and an LP core can execute at different frequencies, all HP cores must share one frequency, as must LP cores. Hence, a strategy to improve reliability should migrate tasks dynamically and consider both the power features and execution models of HP and LP cores. Using this guideline, we have developed frameworks for different hardware platforms to improve soft-error reliability under lifetime reliability, power consumption, and deadline constraints [1, 11].

3.2 CPU–GPU Integrated MPSoCs

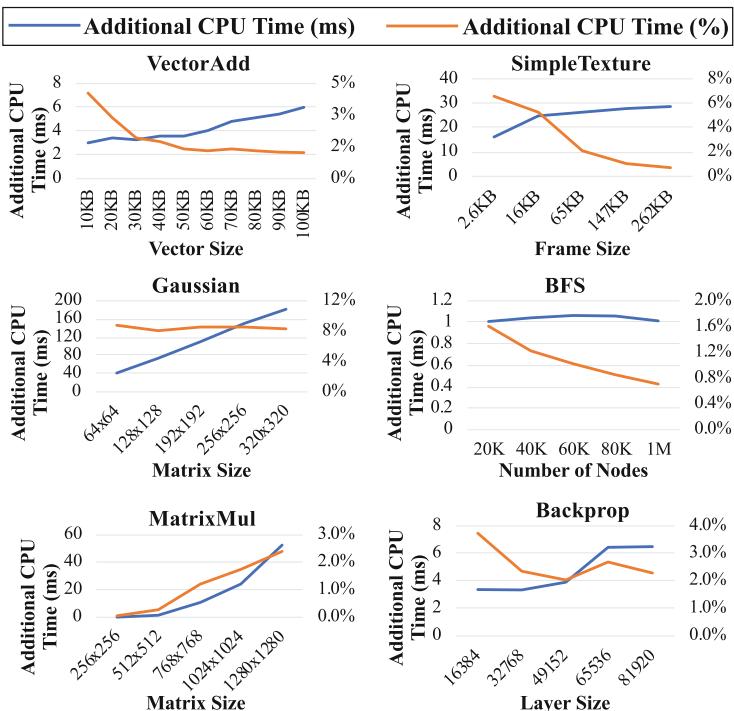
Thanks to the massively parallel computing capability offered by GPUs and the general computing capability of CPUs, MPSoCs with integrated GPUs and CPUs have been widely used in many soft real-time embedded applications, including mobile devices [25] and intelligent video analytics. For many such applications, SER due to transient faults and LTR due to permanent faults are major design concerns. A common reliability improvement objective is to maximize SER under an LTR constraint.

An application task set is used to illustrate how a task's execution time depends on whether it executes on the same core as the operating system. The varying execution times of tasks change the overall workload and operating temperature, influencing LTR and SER. Experiments were performed on Nvidia's TK1 chip (with CUDA 6.5) with default settings to measure task execution times. Six tasks from different benchmark suites were executed (see Table 1). Each task's increase in CPU time resulting from executing on a different core than the operating system is shown in Fig. 6 and the averages of additional GPU times are shown in Table 2. For all tasks, the additional CPU times can be significant and are input dependent. In contrast to the additional CPU time, the additional GPU time is negligible: the additional GPU times of all measured application tasks are less than 1% of the tasks' execution times. This increase can be ignored in most soft real-time applications. Similar phenomena can be observed for other platforms. On Nvidia's TX2 chip, the additional CPU times of application tasks are illustrated in Fig. 7.

The above observations imply that a task's CPU time increases if executed on a different core than the operating system, but its GPU time does not change. Since both LTR and SER increase with a lighter workloads, this observation reveals that we should consider what resources tasks use when assigning them to cores. Generally, the primary core, on which the operating system runs, should be reserved for application tasks that require GPU resources to complete.

Table 1 Application tasks used to measure additional execution times

Name	Description	Source
VectorAdd	Vector addition	CUDA samples [26]
SimpleTexture	Texture use	
MatrixMul	Matrix multiplication	
Gaussian	Gaussian elimination	Rodinia [27]
BFS	Breadth-first search	
Backprop	Back propagation	

**Fig. 6** Measured additional CPU times on TK1 for tasks executing on non-OS CPU cores**Table 2** Observed additional GPU time on TK1

Tasks	Additional GPU time	
	(ms)	(%)
VectorAdd	0.38	0.01
SimpleTexture	0.09	0.00
MatrixMul	0.22	0.11
Gaussian	0.38	0.00
BFS	0.003	0.20
Backprop	0.31	0.68

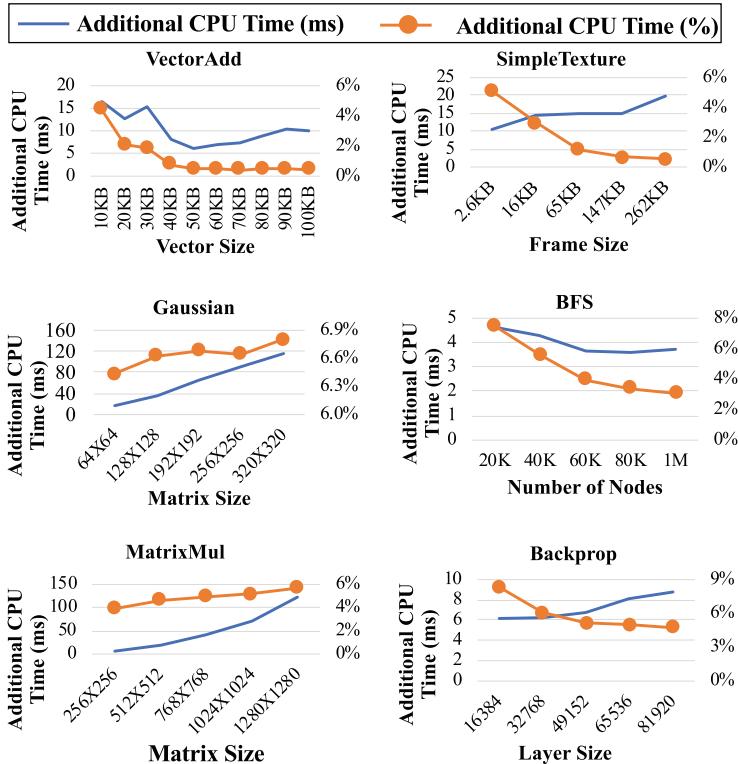


Fig. 7 Measured additional CPU times on TX2 for tasks executing on CPU cores that are different from the core where the operating system runs

4 Conclusion

Real-time embedded system soft-error and lifetime reliabilities are important. Generally, increasing a core's frequency, allocating recoveries and allowing replications improve soft-error reliability, but may increase operating temperature thereby reducing lifetime reliability. MPSoCs used in many applications are heterogeneous and integrate high-performance cores, low-power cores, and even GPUs. System designers should model the task-dependent power consumptions and execution times of the cores available to them, and use these models to solve the SER and LTR trade-off problem.

References

1. Ma, Y., Chantem, T., Dick, R.P., Wang, S., Hu, X.S.: An on-line framework for improving reliability of real-time systems on “big-little” type MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2017)
2. Henkel, J., et al.: Design and architectures for dependable embedded systems. In: Proceedings of the International Conference Hardware/Software Codesign and System Synthesis, October, pp. 69–78 (2011)
3. Haque, M.A., Aydin, H., Zhu, D.: On reliability management of energy-aware real-time systems through task replication. *IEEE Trans. Parallel Distrib. Syst.* **28**(3), 813–825 (2017)
4. Salehi, M., Tavana, M.K., Rehman, S., Shafique, M., Ejlali, A., Henkel, J.: Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems. *IEEE Trans. VLSI Syst.* **24**(7), 2426–2437 (2016)
5. Zhou, J., Hu, X.S., Ma, Y., Wei, T.: Balancing lifetime and soft-error reliability to improve system availability. In: Proceedings of Asia and South Pacific Design Automation Conference, January, pp. 685–690 (2016)
6. Zhou, J., Wei, T.: Stochastic thermal-aware real-time task scheduling with considerations of soft errors. *J. Syst. Softw.* **102**, 123–133 (2015)
7. Huang, L., Yuan, F., Xu, Q.: On task allocation and scheduling for lifetime extension of platform-based MPSoC designs. *IEEE Trans. Parallel Distrib. Syst.* **22**(12), 789–800 (2011)
8. Chantem, T., Xiang, Y., Hu, X.S., Dick, R.P.: Enhancing multicore reliability through wear compensation in online assignment and scheduling. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1373–1378 (2013)
9. Das, A., Kumar, A., Veeravalli, B.: Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems. In: Proceedings of Design, Automation and Test in Europe, March, pp. 689–694 (2013)
10. Das, A., Kumar, A., Veeravalli, B.: Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2014)
11. Ma, Y., Zhou, J., Chantem, T., Dick, R.P., Wang, S., Hu, X.S.: On-line resource management for improving reliability of real-time systems on “big-little” type MPSoCs. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **39**, 88–100 (2018)
12. Das, A., Kumar, A., Veeravalli, B., Bolchini, C., Miele, A.: Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2014)
13. Aliee, H., Gläß, M., Reimann, F., Teich, J.: Automatic success tree-based reliability analysis for the consideration of transient and permanent faults. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1621–1626 (2013)
14. Gupta, P., et al.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **32**(1), 8–23 (2012)
15. Harucha, P., Suensson, C.: Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. Nucl. Sci.* **47**(6), 2586–2594 (2000)
16. Srinivasan, J., Adve, S., Bose, P., Rivers, J.: The impact of technology scaling on lifetime reliability. In: Proceedings of International Conference Dependable Systems and Networks, June, pp. 177–186 (2004)
17. Srinivasan, J., Adve, S., Bose, P., Rivers, J.: Exploiting structural duplication for lifetime reliability enhancement. In: Proceedings of International Symposium on Computer Architecture, June, pp. 520–531 (2005)
18. JEDEC Solid State Technology Association: Failure mechanisms and models for semiconductor devices. Joint Electron Device Engineering Council. Technical Report, August 2003, JEP 122-B
19. Ciappa, M., Carbognani, F., Fichtner, W.: Temperature-aware microarchitecture: modeling and implementation. *IEEE Trans. Device Mater. Reliab.* **3**(4), 191–196 (2003)

20. Xiang, Y., Chantem, T., Dick, R.P., Hu, X.S., Shang, L.: System-level reliability modeling for MPSoCs. In: Proceedings of International Conference Hardware/Software Codesign and System Synthesis, October, pp. 297–306 (2010)
21. Ma, Y., Chantem, T., Dick, R.P., Hu, X.S.: Improving system-level lifetime reliability of multicore soft real-time systems. IEEE Trans. VLSI Syst. **25**(6), 1895–1905 (2017)
22. Zhao, B., Aydin, H., Zhu, D.: Enhanced reliability-aware power management through shared recovery technology. In: Proceedings of International Conference Computer-Aided Design, November, pp. 63–70 (2009)
23. Ma, Y., Chantem, T., Dick, R.P., Hu, X.S.: Improving reliability for real-time systems through dynamic recovery. Proceedings of Design, Automation and Test in Europe, March, pp. 515–520 (2018)
24. Nvidia: Variable SMP (4-plus-1) a multi-core CPU architecture for low power and high performance. <https://www.nvidia.com/content/PDF/tegra> white papers (2018). Online. Accessed Oct 2018
25. Samsung: Samsung Exynos. <https://www.samsung.com/semiconductor/minisite/exynos/> (2018). Online. Accessed Oct 2018
26. Nvidia: CUDA samples. <http://docs.nvidia.com/cuda/cuda-samples/index.html> (2018). Online. Accessed Oct 2018
27. University of Virginia: Rodinia: accelerating compute-intensive applications with accelerators. <https://rodinia.cs.virginia.edu> (2018). Online. Accessed Oct 2018

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part III

Cross-Layer Resilience: Bridging the Gap Between Circuit and Architectural Layer

Daniel Mueller-Gritschneider

Today's design teams, as their forerunners in the past, struggle to master the ever-increasing complexity in chip design driven by new applications such as autonomous driving, complex robotics or embedded machine learning, which demand higher performance under strict power, area, and energy constraints. While Moore's law was providing improvements on all these objectives regularly by moving to the next technology node, a slow-down in scaling is observed nowadays. Yet, design teams came up with intelligent new design principles to provide further performance gains, most famously multi-core and many-core CPUs as well as GPUs combined with complex memory organizations with several level of hierarchy. Additionally, new computing principles moved into the focus of research and industry including near-threshold computing (NTC) for ultra-low-energy operations or the use of runtime-reconfigurable architectures based on field-programmable gate arrays (FPGAs), which flexibly provide specialized compute kernels to boost performance at low power costs.

One major design challenge in these new computing platforms is dependability whenever high system availability is demanded (always on) or, even more strict, in safety-critical applications. Dependable computing requires resilience against a whole range of error sources such as radiation-induced soft errors, aging effects, e.g., caused by Bias Temperature Instability (BTI) or Hot Carrier Injection (HCI), harsh environmental conditions, process variations or supply voltage noise. Depending on the chosen computing principle, new dependability challenges arise, e.g., configuration bits need to be made resilient against errors in FPGAs while NTC-based systems are more sensitive to process variations.

Resilience can be achieved at different layers of the design stack (SW, Compiler, Architecture, Circuit, Device). Traditionally, different parts of the design team look at different layers individually. Hence, resilience can lead to high design overheads

D. Mueller-Gritschneider (✉)
Technical University of Munich, Munich, Germany
e-mail: daniel.mueller@tum.de

as counter-measures on every layer are stacked on top of each other. This leads to the idea of cross-layer resilience. This design method looks for a resilience scheme, in which protection mechanisms work in a cooperative fashion across different layers of the design stack in order to reduce overheads, and, hence, cost. For hardware design, this cross-layer approach can be applied successfully across the architectural and circuit layer. This is demonstrated within the following three chapters.

The chapter titled *Cross-Layer Resilience against Soft Errors: Key Insights* focuses on the analysis and cross-layer protection against soft errors for different system components ranging from embedded processors to SRAM memories and accelerators. While Moore's law is driven mainly by high performance systems, many safety-critical systems are still designed at much older technology nodes to avoid reliability challenges. Yet, due to rising demand of processing power, e.g., for autonomous driving, newer technology nodes become mandatory. This makes it more costly to assure protection against soft errors as their chance of occurrence increases. Soft errors need to be detected and handled in any safety-critical application because they may cause malfunction of the system due to corruption of data or flow of control. Systems deploy protection techniques such as hardening and redundancy at different layers of the system stack (circuit, logic, architecture, OS/schedule, compiler, software, algorithm). Here, cross-layer resilience techniques aim at finding lower cost solutions by providing accurate estimation of soft error resilience combined with a systematic exploration of protection techniques that work collaboratively across the system stack. This chapter provides key insights on applying the cross-layer resilience principle in a lessons-learned fashion.

The chapter *Online Test Strategies and Optimizations for Reliable Reconfigurable Architectures* discusses cross-layer dependability of runtime-reconfigurable architectures based on FPGAs. Such FPGAs are often using the newest technology nodes. Hence, resilience is a major concern as newer nodes experience aging effects earlier and may suffer from higher susceptibility to environmental stress. Device aging can lead to malfunction of the system before its end-of-life, and hence, is a major dependability concern. Incorrect functionality can be detected by executing online built-in self-tests regularly on the device. Two orthogonal online tests are presented in this chapter. These tests can ensure the correctness of the configuration bits of the reconfigurable fabric as well as of the functional parts. Additionally, a design method called module diversification is presented, which enables to recover from faults by providing a self-repair feature. Finally, a design method is presented that implements a stress-aware FPGA placement method. It allows to slow down system degradation due to aging effects and prolongs system lifetime.

The final chapter *Reliability Analysis and Mitigation of Near-Threshold Voltage (NTC) Caches* targets NTC low-energy design and the related reliability concerns. This includes impact of soft error, aging, and process variation while operating at near-threshold voltage with special focus on on-chip caches. The idea is to save energy by scaling supply voltage of the cache blocks. The presented methods guide the designer to optimized NTC cache organizations using a cross-layer reliability analysis approach covering 6T and 8T SRAM cells. Overall, the three chapters bridge the architecture and circuit layer gap while also expanding to adjacent layers of the design stack.

Cross-Layer Resilience Against Soft Errors: Key Insights



Daniel Mueller-Gritschneider, Eric Cheng, Uzair Sharif, Veit Kleeberger,
Pradip Bose, Subhasish Mitra, and Ulf Schlichtmann

1 Introduction

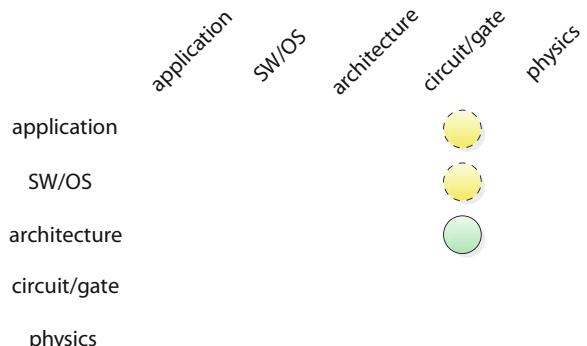
Two tasks need to be solved when designing systems for safety-critical application domains: firstly, the safety of the intended functionality (SoIF) must be guaranteed. SoIF focuses on the ability of the system to sense its environment and act safely. Achieving SoIF becomes a highly challenging task due to the rising complexity of various safety-critical applications such as autonomous driving or close robot–human interaction, which may require complex sensor data processing and interpretation. Secondly, and no less important, the system must also always remain or transit into a safe state given the occurrence of random hardware faults. To achieve this requirement, the system must be capable of detecting as well as handling or correcting possible errors. Safety standards such as ISO26262 for road vehicles define thresholds on detection rates for different automotive safety integration levels (ASIL) depending on the severity of a possible system failure, the controllability by the driver, and the nominal usage time of the system. It is commonly understood that safety-critical systems must be designed from the beginning with the required error protection in mind [39] and that for general-purpose computing systems, error protection is required to achieve dependable computing [19, 21].

U. Sharif · D. Mueller-Gritschneider (✉) · U. Schlichtmann · V. Kleeberger
Infineon Technologies AG, Neubiberg, Germany
e-mail: daniel.mueller@tum.de; uzair.sharif@tum.de; ulf.schlichtmann@tum.de

E. Cheng · S. Mitra
Stanford University, Stanford, CA, USA
e-mail: eccheng@stanford.edu; subh@stanford.edu

P. Bose
IBM, New York, NY, USA
e-mail: pbose@us.ibm.com

Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



This requirement is becoming increasingly challenging as integrated systems—following the continuous trend of Dennard scaling—become more susceptible to fault sources due to smaller transistor dimensions and lower supply voltages. As transistor dimensions scale down the charge stored in memory cells such as SRAM or flip-flops decreases. Soft errors occur due to charge transfers when primary or secondary particles from cosmic radiation hit the silicon [11]. This charge transfer may lead to the corruption of the value stored in the cell. This is referred to as a “soft error” as it does not permanently damage the cell. The vulnerability of cells increases even further with shrinking supply voltage levels or sub-threshold operation. Thus, for the design of safety-critical digital systems, the protection against radiation-induced soft errors is a crucial factor to avoid unacceptable risks to life or property.

This reality motivates methods that aim to increase the resilience of safety-critical systems against radiation-induced soft errors in digital hardware. Common protection techniques against soft errors either harden the memory elements to reduce the probability of soft errors occurring or add redundancy at different layers of the design (circuit, logic, architecture, OS/schedule, compiler, software, algorithm) to detect data corruptions, which can subsequently be handled or corrected by appropriate error handlers or recovery methods. Each protection technique adds overheads and, hence, additional costs. Especially, adding protection techniques on top of each other at all layers—not considering combined effects—may lead to inefficient protection and non-required redundancy. The idea of cross-layer resiliency is to systematically combine protection techniques that work collaboratively across the layers of the system stack. The target is to find more efficient protection schemes with the same soft error resilience at a lower cost than can be reached by ignoring cross-layer effects. For this, cross-layer techniques combine accurate evaluation of the soft error resilience with a broad cross-layer exploration of different combinations of protection techniques. This work demonstrates how to apply the cross-layer resilience principle on custom processors, fixed-hardware processors, accelerators, and SRAM memories with a focus on soft errors. Its main focus spans from application to circuit layer as illustrated in Fig 1. These works lead

to a range of key insights, important for realizing cross-layer soft error resilience for a wide range of system components:

- accurate resilience evaluation is key, e.g., simulation-based fault injection at the flip-flop level is required to accurately evaluate soft errors in logic,
- multi-level/mixed-mode simulation enables very efficient resilience evaluation using fault injection,
- cross-layer resilience exploration must be customized for the component under consideration such as a custom processor, uncore components, third-party processor, accelerator, or SRAM,
- embedded applications such as control algorithms have inherent fault resilience that can be exploited,
- circuit-level techniques are crucial for cost-effective error resilience solutions, and
- existing architecture- and software-level techniques for hardware error resilience are generally expensive or provide too little resilience when implemented using their low-cost variants.

The chapter is structured as follows: first, evaluation methods using fault injection are covered, followed by cross-layer resilience exploration. Finally, experimental results are provided.

2 Evaluation of Soft Error Resilience Using Fault Injection

Fault injection is commonly used to evaluate soft error resilience. Radiation-induced soft errors can be modeled as bit flips [23], which are injected into the system's memory cells such as flip-flops and SRAM cells. There exists a wide range of fault injection methods, which will briefly be discussed in the following.

2.1 Overview on Fault Injection Methods

Hardware-based fault injection injects the fault in a hardware prototype of the system. For example, a radiation beam experiment can be used to provoke faults in an ASIC. This is a very expensive experimental setup, e.g., requiring a radiation source such as used in [1]. The chip hardware can also be synthesized to an FPGA, which is instrumented with additional logic to change bit values in the memory, flip-flops, or combinational paths of the logic to inject a fault using *emulation-based fault injection* [10, 13]. Embedded processors have a debug port to read out their internal states such as architectural registers. These debug ports often also enable the ability to change the internal states. This can be used to inject a fault in the processor using *debug-based fault injection* [15, 41]. Software running on the system can be used to mimic faults in *software-implemented fault injection*, e.g., as

presented in [26, 30, 44]. The compiler can be used to instrument the binary with fault injection code, for *compiler-based fault injection*, e.g., implemented in [18]. *Simulation-based fault injection* injects faults in a simulation model of the system. It is commonly applied to investigate the error resilience of the system and, hence, is the primary focus of this work.

2.2 Simulation-Based Fault Injection

Simulation-based fault injection provides very good properties in terms of parallelism, observability, and early availability during the design. Simulation-based fault injection can be realized at different levels of abstraction. For *gate-level fault injection*, the fault is injected into the gate Netlist of the system obtained after logic synthesis. For *flip-flop-level fault injection*, the fault is injected into the RTL implementation of the system. The fault impact is simulated using logic simulation, e.g., as used in [12, 46]. In *architectural-level fault injection*, the fault is injected either in a micro-architectural simulator or Instruction Set Simulator (ISS). Micro-architectural simulators such as Gem5 [3] simulate all architectural and some micro-architectural states such as pipeline registers of the processor, e.g., as presented in [25], but usually do not accurately model the processor's control logic. An ISS usually only simulates the architectural registers, but not any micro-architectural registers. ISSs are used for fault injection in [14, 24, 35]. In *software-level fault injection*, the fault is directly injected into a variable of the executing program. The software can then be executed to determine the impact of the corrupted variable on the program outputs.

A key insight of previous work was that the evaluation of the soft error resilience of logic circuits such as processor pipelines requires flip-flop-level fault injection, e.g., using the RTL model [9, 38]. Architectural-level and software-level fault injection may not yield accurate results as they do not include all details of the logic implementation as will also be shown in the results in Sect. 4.1. In contrast, soft errors in memories such as SRAM may be investigated at architectural level, which models memory arrays in a bit-accurate fashion.

2.3 Fast Fault Injection for Processor Cores

A good estimation of soft error resilience requires simulating a large amount of fault injection scenarios. This may become computationally infeasible when long-running workloads are evaluated, e.g., for embedded applications. Such long test cases arise in many applications. For example, in order to evaluate the impact of a soft error on a robotic control application, the control behavior needs to simulate several seconds real time, possibly simulating several billion cycles of the digital hardware. An efficient analysis method called ETISS-ML for evaluating the

resilience against soft errors in the logic of a processor sub-system is presented in [37, 38]. A typical processor sub-system of a micro-controller consists of the pipeline, control path, exception unit, timer, and interrupt controller. ETIIS-ML is especially efficient for evaluating the impact of soft errors for long software test cases.

2.3.1 Multi-Level Fault Injection

ETIIS-ML reduces the computational cost of each fault injection run by applying a multi-level simulation approach, which was also applied in other fault injection environments such as [16, 31, 45]. The key idea is to switch abstraction of the processor model during the fault injection run and to minimize the number of cycles simulated at flip-flop level. For this, an ISS is used in addition to the RTL model of the processor at flip-flop level.

The proposed multi-level flow is illustrated in Fig. 2. First the system is booted in ISS mode. This allows to quickly simulate close to the point of the fault injection, at which point, the simulation switches to flip-flop-level. During the RTL warmup phase, instructions are executed to fill the unknown micro-architectural states of the processor sub-system. This is required as the architectural registers are not visible to the ISS simulation. After this RTL warmup, the fault is injected as a bit flip. During the following RTL cool-down phase, the propagation of the fault is tracked. Once the initial impact of the fault propagates out of the processor's micro-architecture or is masked, the simulation can switch back to ISS mode. ETIIS-ML reaches between 40x-100x speedup for embedded applications compared to pure flip-flop-level fault injection while providing the same accuracy [37, 38].

Both the switch from ISS mode to RTL mode as well as the switch from RTL to ISS mode require careful consideration. If a simulation artifact (wrong behavior) is produced by the switching process, it may be wrongly classified as fault impact.

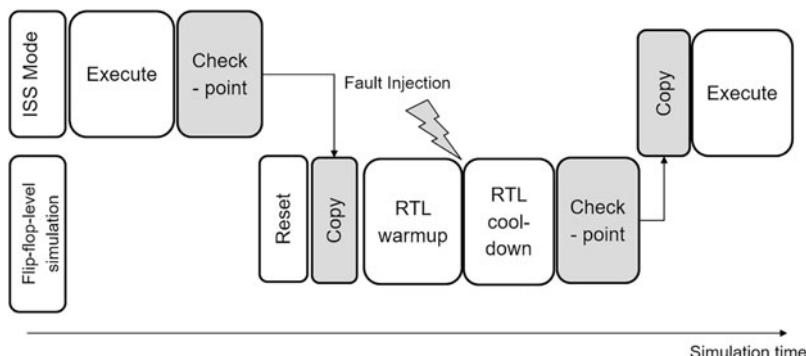


Fig. 2 Multi-level simulation flow of ETIIS-ML

Next, we will detail the state of the art approach used by ETISS-ML to solve these challenges.

2.3.2 Switch from ISS Mode to Flip-Flop-Level Simulation

As shown in Fig. 2, a checkpoint is taken from the ISS to initialize the state in the RTL processor model. This checkpoint only includes the architectural states, the micro-architectural states such as pipeline registers are unknown. In the RTL warmup phase instructions are executed to fill up these micro-architectural states. In order to verify the RTL warmup phase, a (0, 1, X) logic simulation can be applied [37]. All micro-architectural states are initialized to X (unknown), while the values of architectural states are copied from the checkpoint. Additionally, the inputs loaded from external devices such as instruction and data memories as well as peripheral devices are also known from ISS simulation. Naturally, one expects that the micro-architectural states take known values after a certain number of instructions are executed. A key insight here was that this is not the case. Several state machines in the control path and bus interfaces of the processor would start from an unknown state. Hence, all following states remain unknown. One must assume initial states for the RTL state machines, e.g., the reset state. Then one can observe the removal of X values in the RTL model to derive a suitable RTL warmup length for a given processor architecture.

2.3.3 Switch from Flip-Flop-Level Simulation Back to ISS Mode

After the fault has been injected into the RTL model, the flip-flop level simulation is continued during the RTL cool-down phase. When switching back to ISS mode, all micro-architectural states are lost, as only the architectural states are copied over. Hence, one must ensure that one does not lose information about the impact of the fault as this would result in an incorrect estimation. One can take a fixed, very long cool-down phase as proposed in [45]. Yet, this leads to inefficient simulation as many cycles need to be evaluated at flip-flop level. Additionally, one does not gain information as to whether or not the soft error impact is still present in the micro-architectural states. This can be improved by simulating two copies of the RTL model, a faulty processor model and a tracking model [38]. The external state of memories, peripherals, or the environment is not duplicated. The soft error is only injected into the faulty model. In contrast, the tracking model simulates without the error. Writes to the external devices (memories, peripherals) are only committed from the faulty model. Reads from those devices are supplied to both models. Hence, when the soft error is not masked, it may propagate from the faulty model to the architectural state, external memories and devices and, then, be read back to the faulty and tracking model. Whenever both models have the same micro-architectural state, one can be sure that the error either has been masked or has propagated fully to the architectural state or external devices and memories. At this point the simulation

can switch to ISS mode as the architectural state and external devices and memories are also modeled at ISS level. It turns out that some errors never propagate out of the micro-architectural states, e.g., because a configuration is corrupted that is never rewritten by the software. In this case the switch back to ISS mode is not possible as it would cause inaccuracies, e.g., as would be observed with a fixed cool-down length.

2.4 Fast Fault Injection in Uncore Components

In addition to errors impacting processor cores, it is equally important to consider the impact of errors in uncore components, such as cache, memory, and I/O controllers, as well. In SoCs, uncore components are comparable to processor cores in terms of overall chip area and power [33], and can have significant impact on the overall system reliability [8].

Mixed-mode simulation platforms are effective for studying the system-level impact and behavior of soft errors in uncore components as well. As presented in [8], such a platform would achieve a $20,000\times$ speedup over RTL-only injection while ensuring accurate modeling of soft errors. Full-length applications benchmarks can be analyzed by simulating processor cores and uncore components using an instruction-set simulator in an accelerated mode. At the time of injection, the simulation platform would then enter a co-simulation mode, where the target uncore component is simulated using accurate RTL simulation. Once co-simulation is no longer needed (i.e., all states can be mapped back to high-level models), the accelerated mode can resume, allowing application benchmarks to be run to completion.

2.5 Fast Fault Injection for SRAM Memories Using Mixture Importance Sampling

Memories such as on-chip SRAM or caches are already modeled bit-accurately at micro-architectural and instruction-level. Hence, for the evaluation of soft errors in memories, fault injection into faster instruction-level models is possible. Yet, modern SRAMs are very dense such that the probability of multi-bit upsets (MBUs) due to soft errors is not negligible. For MBU fault models, straightforward Monte Carlo simulation requires a large sample size in the range of millions of sample elements to obtain sufficient confidence bounds.

To address this challenge one can apply mixture importance sampling to connect a technology-level fault model with a system-level fault simulation [29]. This propagation of low-level information to the system level is motivated by the Resilience Articulation Point (RAP) approach proposed in [23]. The key idea behind

RAP is that errors in the system should be modeled by probabilistic functions describing MBU's bit flip probabilities including spatial and temporal correlations. Thus, the impact of errors in the system can be evaluated, while maintaining a direct connection to their root causes at the technology level. The sample size to estimate the resilience of the system to soft errors in SRAMs can be massively reduced by guiding the Monte Carlo simulation to important areas. As an illustrative example, we assume that the SRAM is used to realize a data cache with 1-bit parity protection. MBUs that alter an odd number of bits in a cache line are detected by the parity checks and may be corrected by loading the correct value from the next level of memory. MBUs that alter an even number of bits in a cache line remain undetected and may cause silent data corruption. Additionally, MBUs may perturb several neighboring cache lines due to different MBU mechanisms. This can lead to mixed cases of recoverable errors and silent data corruption. For a cache with one bit parity protection, MBUs with even number (2, 4, ...) of bits in one cache line are critical as they may provoke silent data corruption (SDC). The sampling strategy can be biased towards these MBUs by mixture important sampling, which speeds up the resilience evaluation significantly. It is shown that results with high confidence can be obtained with sample sizes in the thousands instead of millions [29]. The resulting fast evaluation enables the efficient exploration of the most efficient cross-layer protection mechanisms for the SRAM memory for an overall optimized reliable system.

3 Cross-Layer Exploration of Soft Error Resilience Techniques

Most safety-critical systems already employ protection techniques against soft errors at different layers. Yet often, possible combinations are not systematically explored and evaluated to identify a low-cost solution. This may result in inefficient redundancy and hardening, e.g., that certain types of faults are detected by multiple techniques at different layers, or certain redundancy is not required, as the circuit is adequately protected (e.g., by circuit-hardening techniques).

In this section several approaches are outlined that focus on cross-layer exploration for finding low-cost soft error protection:

- the CLEAR approach can generate resilience solutions for custom processors with selective hardening in combination with architectural and software-level protection schemes.
- Using a similar approach, on-chip SRAM can be protected with a combination of hardening and error detection codes.
- For third-party processors, hardening and hardware redundancy are not an option. Hence, we show how application resilience can be used in combination with software-level protection to achieve cross-layer resilience.

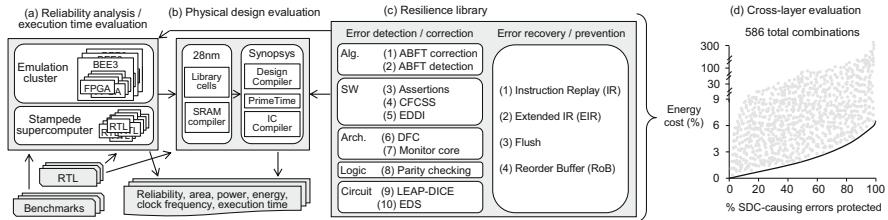


Fig. 3 CLEAR framework: (a) BEE3 emulation cluster/Stampede supercomputer injects over 9 million errors into two diverse processor architectures running 18 full-length application benchmarks. (b) Accurate physical design evaluation accounts for resilience overheads. (c) Comprehensive resilience library consisting of ten error detection/correction techniques + four hardware error recovery techniques. (d) Example illustrating thorough exploration of 586 cross-layer combinations with varying energy costs vs. percentage of SDC-causing errors protected

- Finally, we also discuss how accelerators can be protected with cross-layer resilience techniques.

3.1 CLEAR: Cross-Layer Resilience for Custom Processors

CLEAR (Cross-Layer Exploration for Architecting Resilience) is a first of its kind framework to address the challenge of designing robust digital systems: given a set of resilience techniques at various abstraction layers (circuit, logic, architecture, software, algorithm), how does one protect a given design from radiation-induced soft errors using (perhaps) a combination of these techniques, across multiple abstraction layers, such that overall soft error resilience targets are met at minimal costs (energy, power, execution time, area)?

CLEAR has broad applicability and is effective across a wide range of diverse hardware designs ranging from in-order (InO-core) and out-of-order (OoO-core) processor cores to uncore components such as cache controllers and memory controllers to domain-specific hardware accelerators. CLEAR provides the ability to perform extensive explorations of cross-layer combinations across a rich library of resilience techniques and error sources.

Figure 3 gives an overview of the CLEAR framework. Individual components are described briefly in the following:

3.1.1 Reliability Analysis

While the CLEAR framework provides the ability to analyze the reliability of designs, this component does not comprise the entirety of the framework. The modularity of the CLEAR framework enables one to make use of any number of the accurate fault-injection simulation components described in detail in Sect. 2.2

to perform reliability analysis. The analysis considered in this chapter encompasses both Silent Data Corruption (SDC) and Detected but Uncorrected Errors (DUE).

3.1.2 Execution Time Evaluation

Execution time is measured using FPGA emulation and RTL simulation. Applications are run to completion to accurately capture the execution time of an unprotected design. For resilience techniques at the circuit and logic levels, CLEAR ensures that modifications incorporating such resilience techniques will maintain the same clock speed as the unprotected design. For resilience techniques at the architecture, software, and algorithm levels, the error-free execution time impact is also reported.

3.1.3 Physical Design Evaluation

To accurately capture overheads associated with implementing resilience techniques, it is crucial to have a means for running an entire physical design flow to properly evaluate the resulting designs. To that end, the Synopsys design tools (Design Compiler, IC compiler, PrimeTime, and PrimePower) with a commercial 28nm technology library (with corresponding SRAM compiler) are used to perform synthesis, place-and-route, and power analysis. Synthesis and place-and-route (SP&R) is run for all configurations of the design (before and after adding resilience techniques) to ensure all constraints of the original design (e.g., timing and physical design) are met for the resilient designs as well.

3.1.4 Resilience Library

For processor cores, ten error detection and correction techniques together with four hardware error recovery techniques are carefully chosen for analysis. In the context of soft error resilience, error detection and correction techniques include: Algorithm Based Fault Tolerance (ABFT) correction, ABFT detection, Software assertions, Control Flow Checking by Software Signatures (CFCSS), Error Detection by Duplicated Instructions (EDDI), Data Flow Checking (DFC), Monitor cores, Parity checking, flip-flop hardening using LEAP-DICE, and Error Detection Sequential (EDS). These techniques largely cover the space of existing soft error resilience techniques. The characteristics (e.g., costs, resilience improvement, etc.) of each technique when used as a standalone solution (e.g., an error detection/correction technique by itself or, optionally, in conjunction with a recovery technique) are presented in Table 1. Additionally, four micro-architectural recovery techniques are included: Instruction Replay (IR), Extended IR (EIR), flush, and Reorder Buffer (RoB) recovery. Refer to [7] for an in-depth discussion of specific techniques and their optimizations, including a detailed discussion of Table 1.

Table 1 Individual resilience techniques: costs and improvements as a standalone solution

Layer	Technique	Area cost	Power cost	Energy cost	Exec. time impact	Avg. SDC improve	Avg. DUE improve	False positive	Detection latency	γ
Circuit	LEAP-DICE (no additional recovery needed)	InO OoO	0–9.3% 0–6.5%	0–22.4% 0–9.4%	0–22.4% 0–9.4%	0%	1 × –5000×	1 × –5000×	N/A	1
	EDS (without recovery—unconstrained)	InO OoO	0–10.7% 0–12.2%	0–22.9% 0–11.5%	0–22.9% 0–11.5%	0%	1 × –>100,000×	0.1 × –1×	0%	1 cycle
	EDS (with IR recovery)	InO OoO	0–16.7% 0–12.3%	0–43.9% 0–11.6%	0–43.9% 0–11.6%	0%	1 × –>100,000×	1 × –>100,000×	0%	1 cycle
Logic	Parity (without recovery—unconstrained)	InO OoO	0–10.9% 0–14.1%	0–23.1% 0–13.6%	0–23.1% 0–13.6%	0%	1 × –>100,000×	0.1 × –1×	0%	1 cycle
	Parity (with IR recovery)	InO OoO	0–26.9% 0–14.2%	0–44% 0–13.7%	0–44% 0–13.7%	0%	1 × –>100,000×	1 × –>100,000×	0%	1 cycle
Arch.	DFC (without recovery—unconstrained)	InO OoO	3% 0.2%	1% 0.1%	7.3% 7.2%	6.2% 7.1%	1.2×	0.5×	0%	15 cycles
	DFC (with EIR recovery)	InO OoO	37% 0.4%	33% 0.2%	41.2% 7.3%	6.2% 7.1%	1.2×	1.4×	0%	15 cycles
	Monitor core (with RoB recovery)	OoO	9%	16.3%	16.3%	0%	19×	15×	0%	128 cycles

(continued)

Table 1 (continued)

Layer	Technique	Area cost	Power cost	Energy cost	Exec. time impact	Avg. SDC improve	Avg. DUE improve	False positive	Detection latency	γ
Software	Software assertions for general-purpose processors (without recovery—unconstrained)	InO	0%	0%	15.6%	15.6%	1.5×	0.6×	9.3M cycles	1.16
	CFCSS (without recovery—unconstrained)	InO	0%	0%	40.6%	40.6%	1.5×	0.5×	0%	6.2M cycles
	EDDI (without recovery—unconstrained)	InO	0%	0%	110%	110%	37.8×	0.3×	0%	287K cycles
Alg.	ABFT correction (no additional recovery needed)	InO OoO	0%	0%	1.4%	1.4%	4.3×	1.2×	0%	N/A
	ABFT detection (without recovery—unconstrained)	InO OoO	0%	0%	24%	1–56.9%	3.5×	0.5×	0%	9.6M cycles

3.1.5 Exploration

CLEAR approaches cross-layer exploration using a top-down approach: resilience techniques from upper layers of the resilience stack (e.g., algorithm-level techniques) are applied before incrementally moving to lower layers (e.g., circuit-level techniques). This approach helps generate cost-effective solutions that leverage effective interactions between techniques across layers. In particular, while resilience techniques from the algorithm, software, and architecture layers of the stack generally protect multiple flip-flops, a designer typically has little control over the specific subset of flip-flops that will be protected. Using multiple techniques from these layers can lead to a situation where a given flip-flop may be protected (sometimes unnecessarily) by multiple techniques. Conversely, resilience techniques at the logic and circuit layers offer fine-grained protection since these techniques can be applied selectively to individual flip-flops (i.e., flip-flops not (sufficiently) protected by higher-level techniques).

3.2 *Resilience Exploration for Custom Accelerators*

Domain-specific hardware accelerators will increasingly be integrated into digital systems due to their ability to provide more energy-efficient computation for specific kernels. As a result of their application-specific nature, hardware accelerators have the opportunity to leverage application space constraints when exploring cross-layer resilience (i.e., resilience improvement targets only need to hold over a limited subset of applications). Accelerators also benefit from the ability to create natural checkpoints for recovery by protecting the memory storing the accelerator inputs (e.g., using ECC), allowing for a simple means for re-execution on error detection. Therefore, the cross-layer solutions that provide cost-effective resilience may differ from those of processor cores and warrant further exploration.

3.3 *Cross-Layer Resilience for Exploration for SRAM Memories*

In [28], a cross-layer approach for soft error resilience was applied to SRAM data caches. Again, a systematic exploration requires having a good evaluation of the cost and efficiency of the applied protection mechanisms. In this study, the available protection mechanisms were the following: at circuit level, either (1) the supply voltage could be raised by 10% or (2) the SRAM cells could be hardened by doubling the area. At the architectural level, (3) 1-bit parity could be introduced in the cache lines. The circuit-level hardening techniques require parameterizing the statistical MBU fault model introduced in Sect. 2.5 considering cell area, supply

voltage and temperature. For each configuration, the fault probabilities for MBU patterns need to be evaluated to obtain a good estimate of soft error probabilities. Additionally, the architecture and workload play a key role in the evaluation as not all soft errors are read from the cache. Here again, architectural-level simulation can be used to simulate the workload using fault injection into a bit-accurate cache model.

3.4 Towards Cross-Layer Resiliency for Cyber-Physical Systems (CPS)

In benchmark-type workloads, silent data corruption in a single program output commonly leads to a failure, e.g., an encryption algorithm fails if its encrypted data is corrupted such that it cannot be decrypted. Hence, cross-layer resiliency often targets reducing the rate of silent data corruption.

For cyber-physical systems (CPS), however, many workloads can tolerate deviations from the fault-free outcome, e.g., in an embedded control algorithm, noise, e.g., in sensors, is present and considered in the control design. It will treat silent data corruption as yet another noise source, that can, possibly, be tolerated for minor deviations from the correct value. Another effect is that CPS workloads are commonly scheduled as periodic tasks. Often, the outputs of one instance of a certain task are overwritten by the next instance of a task. Hence, a corruption of the output of a single task has an effect only for a certain duration in time. Subsequent task executions might mitigate the effect of silent data corruption before the system behavior becomes critical. For example for control applications, the sampling rate of the controller is often higher than demanded, such that a single corrupted actuation command will not lead to a failure within one control period. Following sensor readouts will show a deviation from the desired control behavior that is corrected by the controller in subsequent control periods.

In order to consider the inherent resilience of CPS workloads, a full system simulation is required. CPS usually form a closed loop with their environment, e.g., actuation will change the physical system behavior, which determines future sensor readouts. Extensive fault injection for obtaining a good resiliency evaluation is enabled by the fast simulation speed of ETIIS-ML [38], while RTL level fault injection would be prohibitively slow to evaluate system behavior over a long system-level simulation scenario. ETIIS-ML can be integrated into a full-system virtual prototype (VP) that models the system and its physical environment such that error impacts can be classified considering the inherent resilience of CPS workloads. For this, the physical behavior is traced to determine the impact of the error. A major question to be investigated is how this inherent application resilience can be exploited in an efficient way to reduce cost of protection techniques towards cross-layer resilience of CPS.

4 Experimental Results

This section presents results for cross-layer exploration. First, we show results that support our claim that flip-flop level fault injection is required for soft errors in logic. Then we provide the results for cross-layer exploration with CLEAR and ETISSLML for processors. Finally, we show the results for the cross-layer exploration of protection techniques for the data cache of a control system for a self-balancing robot.

4.1 Accuracy of FI at Different Abstraction Levels

For radiation-induced soft errors, flip-flop soft error injection is considered to be highly accurate. Radiation test results confirm that injection of single bit flips into flip-flops closely models soft error behaviors in actual systems [4, 43]. On the other hand, [9] has shown that naïve high-level error injections (e.g., injection of a single-bit error into an architecture register, software-visible register-file, or program variable) can be highly inaccurate.

Accurate fault-injection is crucial for cost-effective application of cross-layer resilience. Inaccurate reliability characterization may lead to over- or underprotection of the system. Overprotection results in wasted cost (e.g., area, power, energy, price) and underprotection may result in unmitigated system failures.

In order to observe the impact of soft errors in the data and control path of a OR1K processor sub-system, the error propagation was tracked to the architectural-visible states in [38] for four test cases. In total 70k fault injection scenarios were run on each test case. The injection points were micro-architectural FFs in the RTL implementation such as pipeline and control path registers, that are not visible at the architectural level. First all soft errors were identified that had no impact on the architectural state since they were either being masked or latent. On average these were 67.51%.

On architectural level, we inject single bit flip fault scenarios as it is unclear what multi-bit fault scenarios could really happen in HW. These scenarios will cover all single bit flip soft errors in an architectural state as well as any soft error in a micro-architectural state that propagates and corrupts just a single bit of an architectural state. In this case it makes no difference whether we inject the single bit flip in the micro-architectural state or architectural state. Yet, the distribution could be different. We now observe the experimental results as given in Table 2: 25.09% of the micro-architectural faults corrupted a single bit in the architectural state for a single cycle. These faults would be covered by fault injection at architectural level. But 7.40% of the soft errors corrupted several bits of the architectural state or lead to several bit flips in subsequent cycles. Injecting single bit soft errors in architectural states at architecture- or software level will not cover these micro-architectural fault

Table 2 Impact of single bit flip in micro-arch FFs on architectural processor state

Test case	Masked or latent [%]	Single bit corruption [%]	Multi-bit corruption [%]
JDCT	66.77	25.68	7.55
AES	66.36	26.13	7.51
IIR	68.88	23.83	7.29
EDGE	68.02	24.73	7.25
Average	67.51	25.09	7.40

Table 3 General-purpose processor core designs studied

	Design	Description	Clk. freq.	Error injections	Instructions per cycle
InO	LEON3 [17]	Simple, in-order (1250 flip-flops)	2.0 GHz	5.9 million	0.4
OoO	IVM [46]	Complex, super-scalar, out-of-order (13,819 flip-flops)	600 MHz	3.5 million	1.3

scenarios. Hence, one needs to look into RTL fault injection to obtain accurate results for these faults.

4.2 Cross-Layer Resilience Exploration with CLEAR

The CLEAR framework is first used to explore a total of 586 cross-layer combinations in the context of general-purpose processor cores. In particular, this extensive exploration consists of over 9 million flip-flop soft error injections into two diverse processor core architectures (Table 3): a simple, in-order SPARC LEON3 core (InO-core) and a complex superscalar out-of-order Alpha IVM core (OoO-core). Evaluation is performed across 18 application benchmarks from the SPECINT2000 [22] and DARPA PERFECT [2] suites.

Several insights resulted from this extensive exploration: accurate flip-flop level injection and layout (i.e., physical design) evaluation reveal many individual techniques provide minimal (less than $1.5 \times$) SDC/DUE improvement (contrary to conclusions reported in the literature that were derived using inaccurate architecture- or software-level injection [20, 36]), have high costs, or both. The consequence of this revelation is that most cross-layer combinations have high cost.

Among the 586 cross-layer combinations explored using CLEAR, a highly promising approach combines selective circuit-level hardening using LEAP-DICE, logic parity, and micro-architectural recovery (flush recovery for InO-cores, reorder

buffer (RoB) recovery for OoO-cores). Thorough error injection using application benchmarks plays a critical role in selecting the flip-flops protected using these techniques.

From Table 4, to achieve a $50\times$ SDC improvement, the combination of LEAP-DICE, logic parity, and micro-architectural recovery provides $1.5\times$ and $1.2\times$ energy savings for the OoO- and InO-cores, respectively, compared to selective circuit hardening using LEAP-DICE. This scenario is shown under “bounded latency recovery.” The relative benefits are consistent across benchmarks and over the range of SDC/DUE improvements.

If recovery hardware is not needed (i.e., there exist no recovery latency constraints and errors can be recovered using an external means once detected), minimal ($<0.2\%$ energy) savings can be achieved when targeting SDC improvement. This scenario is shown under “unconstrained recovery.” However, without recovery hardware, DUEs increase since detected errors are now uncorrectable; thus, no DUE improvement is achievable.

Additional cross-layer combinations spanning circuit, logic, architecture, and software layers are presented in Table 4. In general, most cross-layer combinations are not cost-effective. For general-purpose processors, a cross-layer combination of LEAP-DICE, logic parity, and micro-architectural recovery provides the lowest cost solution for InO- and OoO-cores for all improvements.

4.3 Resilience Exploration for Custom Accelerators

Utilizing a high-level synthesis (HLS) engine from UIUC [5], 12 accelerator designs derived from the PolyBench benchmark suite [42] were evaluated with protection using LEAP-DICE (circuit), logic parity (logic), modulo-3 shadow datapaths (architecture), EDDI (software), and ABFT (algorithm) techniques. Note that, software and algorithm techniques are converted into hardware checkers during high-level synthesis.

Consistent with processor core results, cost-effective resilience solutions for domain-specific hardware accelerators (Table 5) required the use of circuit-level techniques (e.g., a $50\times$ SDC improvement was achieved at less than 6% energy cost using a combination of application-guided selective LEAP-DICE and logic parity). However, even given the application-constrained context of accelerators, software-level (and algorithm-level) resilience techniques were unable to provide additional benefits.

Table 4 Costs vs. SDC (DUE) improvements for various combinations in general-purpose processors

		Bounded latency recovery										Unconstrained recovery										Exec. time impact
		SDC improvement					DUE improvement					SDC improvement					DUE improvement					
		2	5	50	500	Max 2	5	50	500	Max	2	5	50	500	Max	2	5	50	500	Max		
InO	LEAP-DICE + logic parity (+ flush recovery)	A	0.7	1.7	2.5	3	8	0.6	1.5	3.6	4.4	8	0.7	1.6	2.4	2.8	7.6	—	—	—	—	0%
	P	1.9	3.9	6.1	6.7	17.9	1.5	3.4	8.4	10.4	17.9	1.9	3.8	5.9	6.5	17.2	—	—	—	—	—	—
	E	1.9	3.9	6.1	6.7	17.9	1.5	3.4	8.4	10.4	17.9	1.9	3.8	5.9	6.5	17.2	—	—	—	—	—	—
EDS + LEAP-DICE + logic parity (+ flush recovery)	A	0.9	2.3	2.7	3.3	8.4	0.8	2.1	3.8	4.8	8.4	0.9	2.2	2.5	3.2	8.1	—	—	—	—	—	0%
	P	1.9	4.3	6.6	7.2	19.3	1.7	3.8	8.5	11	19.3	1.9	4.2	6.3	7.1	19	—	—	—	—	—	—
	E	1.9	4.3	6.6	7.2	19.3	1.7	3.8	8.5	11	19.3	1.9	4.2	6.3	7.1	19	—	—	—	—	—	—
DFC + LEAP-DICE + logic parity (+ EIR recovery)	A	39.3	41.1	41.5	43.1	45	39.3	39.9	41.9	42.5	45	3.3	5.1	5.6	7.1	10.6	—	—	—	—	—	6.2%
	P	32.4	35.5	38.7	41	50.9	32.5	33.9	38.4	40.7	50.9	1.4	4.8	8.1	10	18.2	—	—	—	—	—	—
	E	44.2	56.7	60.2	62.7	60.3	45.8	48.9	58.3	63	60.3	10.6	13.9	17.4	19.9	25.5	—	—	—	—	—	15.6%
Assertions + LEAP-DICE + logic parity (no recovery)	A	—	—	—	—	—	—	—	—	—	—	0.7	0.9	1	1.1	7.6	—	—	—	—	—	—
	P	—	—	—	—	—	—	—	—	—	—	1.4	1.8	2.2	2.2	17.2	—	—	—	—	—	—
	E	—	—	—	—	—	—	—	—	—	—	17.1	17.5	18	18	24.5	—	—	—	—	—	—
CFCSS + LEAP-DICE + logic parity (no recovery)	A	—	—	—	—	—	—	—	—	—	—	0.3	1	1.4	1.3	7.6	—	—	—	—	—	40.6%
	P	—	—	—	—	—	—	—	—	—	—	0.8	1.8	2.9	3.1	17.2	—	—	—	—	—	—
	E	—	—	—	—	—	—	—	—	—	—	41.5	43	44.6	44.9	146	—	—	—	—	—	—
CFCSS + LEAP-DICE + logic parity (no recovery)	A	—	—	—	—	—	—	—	—	—	—	0.3	1	1.4	1.3	7.6	—	—	—	—	—	40.6%
	P	—	—	—	—	—	—	—	—	—	—	0.8	1.8	2.9	3.1	17.2	—	—	—	—	—	—
	E	—	—	—	—	—	—	—	—	—	—	41.5	43	44.6	44.9	146	—	—	—	—	—	—
EDDI + LEAP-DICE + logic parity (no recovery)	A	—	—	—	—	—	—	—	—	—	—	0	0	0.7	0.9	7.6	—	—	—	—	—	110%
	P	—	—	—	—	—	—	—	—	—	—	0	0	0.6	0.8	17.2	—	—	—	—	—	—
	E	—	—	—	—	—	—	—	—	—	—	110	110	111	111	146	—	—	—	—	—	—

OoO	LEAP-DICE + logic parity (+ RoB recovery)	A	0.06	0.1	1.4	2.2	4.9	0.5	0.7	2.6	3	4.9	0.06	0.1	1.4	2.2	4.9	-	-	-	-	0%
	P	0.1	0.2	2.1	2.4	7	0.1	0.1	2	1.8	7	0.1	0.2	2.1	2.4	7						
E	0.1	0.2	2.1	2.4	7	0.1	0.1	2	1.8	7	0.1	0.2	2.1	2.4	7							
EDS + LEAP-DICE + logic parity (+ RoB recovery)	A	0.07	0.1	1.6	2.2	5.4	0.6	0.8	2.6	3	5.4	0.07	0.1	1.6	2.2	5.4	-	-	-	-	-	0%
	P	0.1	0.2	2.3	2.5	8.1	0.1	0.1	2	1.8	8.1	0.1	0.2	2.3	2.5	8.1						
	E	0.1	0.2	2.3	2.5	8.1	0.1	0.1	2	1.8	8.1	0.1	0.2	2.3	2.5	8.1						
DFC + LEAP-DICE + logic parity (+ EIR recovery)	A	0.2	1	1.8	2	5.3	0.2	0.4	1.7	3.9	5.3	0.1	0.8	1.6	1.8	5.1	-	-	-	-	-	7.1%
	P	1.1	1.4	2	2.8	7.2	0.2	0.2	2.6	3.3	7.2	1	1.3	1.9	2.7	7.1						
	E	21.2	21.5	22.2	23	14.8	20	20.1	22.9	23.6	14.8	10	11.4	12.1	12.9	14.7						
Monitor core + LEAP-DICE + logic parity (+ RoB rec.)	A	9	9	9.8	10.5	13.9	9	9	10.1	11.2	13.9	9	9.8	10.5	13.9	-	-	-	-	-	-	0%
	P	16.3	16.3	20	20.2	23.3	16.3	16.3	20.1	21.5	22.3	16.3	16.3	20	20.2	23.3						
	E	16.3	16.3	20	20.2	23.3	16.3	16.3	20.1	21.5	22.3	16.3	16.3	20	20.2	23.3						

A (area cost %), P (power cost %), E (energy cost %)

Table 5 Costs (area/energy) and improvements for resilience in 12 domain-specific accelerators

Resilience technique(s)	SDC improvement			
	2×	5×	50×	500×
Selective LEAP-DICE	0.9%/3.3%	1.2%/5%	1.7%/7%	2.2%/8.8%
Selective parity checking	1.4%/4.4%	2.2%/6.4%	3.1%/8.7%	3.4%/10.6%
LEAP-DICE + parity	0.6%/2.7%	1%/3.9%	1.3%/5.7%	1.7%/7.4%
Mod-3 + LEAP-DICE + parity	0.7%/3.6%	2.3%/4.7%	2.9%/6.5%	3.3%/8.1%
EDDI + LEAP-DICE + parity	27.6%/33%	27.6%/33.2%	27.6%/33.4%	28.3%/34%
ABFT + LEAP-DICE + parity	11.9%/23.8%	12.2%/24.1%	12.3%/24.2%	12.3%/24.8%

Table 6 Micro-controller (μ C) design studied

	Design	Description	Clk. freq.	Error injections
μ C	OpenRISC [40]	Simple, in-order (no caches), (1440 flip-flops) with timer and interrupt controller	100 MHz	500,000

4.4 Resilience Exploration for Fixed-hardware Micro-Controller

The multi-level simulation was implemented for a fixed-hardware micro-controller (μ C) as shown in Table 6. The RTL implementation uses only the pipeline, programmable interrupt controller, and timer but no caches in order to have a μ C-type processor similar to ARM’s Cortex M family. We study a full system simulation setup based on a SystemC VP, which models an μ C used in a simplified adaptive cruise control (ACC) system. Its goal is to maintain a constant distance between two moving vehicles by controlling the speed of the rear vehicle via the throttle value of the motor (actuator). The processor of the μ C periodically executes a PI control algorithm. The PI control algorithm’s inputs are sensor values measuring the distance to the front vehicle and speed of the rear vehicle. Figure 4 shows the SystemC/TLM model structure of the system with μ C, actuator and sensors. The sensor values are dynamically generated by a physics simulation of the two vehicles based on the commands sent to the actuator. The system boots and then starts execution from time zero. We define a simple safety specification to demonstrate the evaluation. The desired distance between the vehicles is set to 40 m. A fault is classified to cause a system-level failure when the distance leaves the corridor between 20 m and 60 m within a given driving scenario. For this scenario, both vehicles have same speed and a distance of 50 m at time zero.

Figure 5 shows the simulation results for four fault injection (FI) simulations. The green curve shows a soft error that has no influence on the system outputs, which results in the same curve visible in the fault-free run. The blue curve shows the inherent fault tolerance of control algorithms. Even though the actuator output is corrupted by the soft error, the control algorithm is able to recover from the

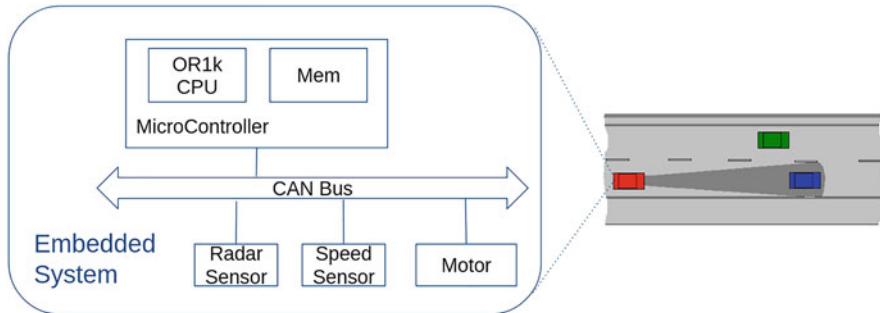


Fig. 4 SystemC VP of control system

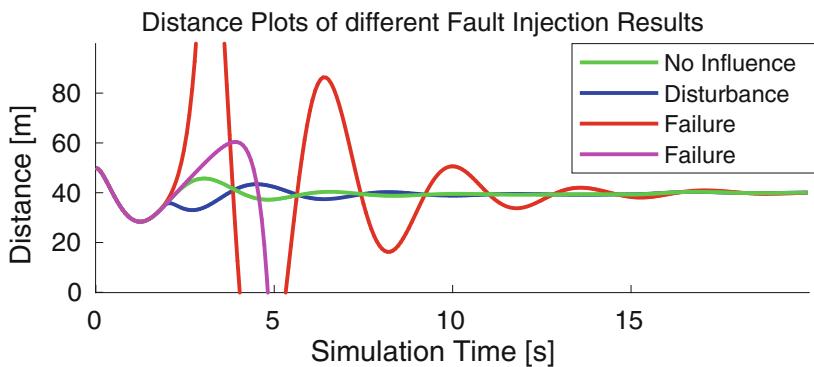


Fig. 5 Distance plotted for different FIs

disturbance. The distance does not leave the specified corridor. Finally, the pink and red curves show faults leading to a system failure.

In order to test cross-layer resiliency, we apply the following error detection and handling mechanisms. We concentrate on methods supported by fixed-hardware μ Cs, for which we would not be able to modify the logic or circuit implementation.

Watchdog Timer (WDT) The control algorithm has to write a value to the actuator every 10 ms. If no actuator write is detected, the system is reset by the WDT.

Task Duplication The control task is executed twice and the results are compared before the actuation.

EDDI EDDI is applied by the compiler to protect the data flow of the control application.

CFCSS CFCSS is applied by the compiler to protect the control flow of the control application.

The compiler can only apply EDDI and CFCSS on the software functions of the PI control task, not on software functions coming from the pre-compiled OR1K C-

Table 7 Comparison of resilience techniques for μ C with watchdog timer (WDT) and external recovery by system reset

Resilience technique(s)	WDT Det. rate	SW Det. rate	SDC rate	Failure rate due to SDC	Exec. time impact
WDT	8.562%	0%	0.674%	0.061%	0%
Task duplic.+WDT	11.429%	1.284%	0.026%	0.002%	146.21%
EDDI+WDT	11.926%	1.706%	0.014%	0.002%	155.86%
CFCSS+WDT	8.929%	2.028%	0.542%	0.047%	0.249%
EDDI+CFCSS+WDT	13.370%	2.169%	0.017%	0.001%	156.857%

libraries. When task duplication, EDDI or CFCSS detect a fault, the SW triggers a reset.

Each method comes with a certain overhead and improvement in SDC rate as shown in Table 7. The column “WDT Det. Rate” shows the percentage of faults detected by the watchdog timer. The column “SW Det Rate” shows the percentage of faults detected by EDDI, CFCSS and the comparison for Task Duplication (depending on which protection is used). The SDC rate shows the percentage of faults that lead to a corrupt actuation value without being detected by a protection technique. Finally, the failure rate due to SDC shows the percentage of SDCs that lead to a failure of the control algorithm. Exec. Time Impact shows the overhead due to software redundancy inserted by the protection mechanisms. A WDT requires additional area, which is usually available on modern μ Cs, hence, this is ignored.

The following conclusions can be derived from the results: overall, the WDT detection rate is very high as it detects most DUEs, that result in incorrect timing of the application. EDDI and task duplication increases the execution time of the control task significantly at the cost of idle time of the processor. Yet, they also lead to significant SDC reduction. EDDI is slightly better, as it works on the intermediate representation (IR) and has a smaller vulnerability window. CFCSS also increases the software detection rate. Upon closer inspection, CFCSS does not lead to a significant reduction in SDC rate for both cases with and without EDDI. The application has a simple control flow, hence, control flow errors are rare. Most of the errors detected by CFCSS are due to errors during execution of the CFCSS check codes themselves. Hence, they would not lead to SDC of the functional code, yet, many errors are reported.

4.5 Resilience Exploration for SRAM Cache of Self-Balancing Robot

The cross-layer exploration was applied to a self-balancing robot system in [28] as shown in Fig. 6. The results are shown in Fig. 7. The figure shows the results for nominal SRAM design (N), increased supply voltage (V), increased area (A) and parity protection (P). The blue bar shows the rate of silent data corruption

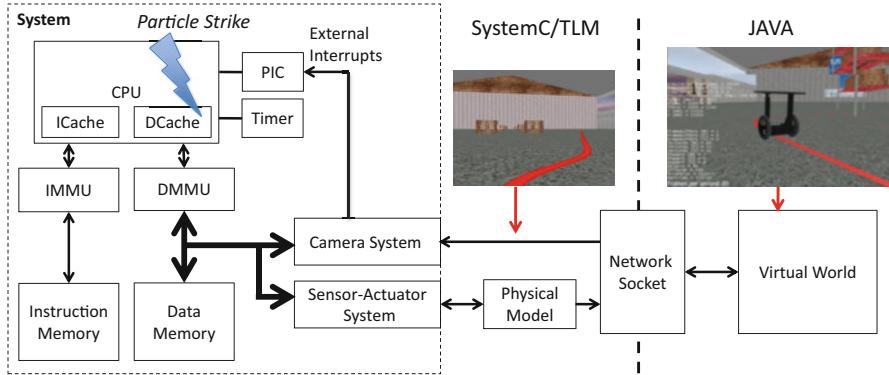


Fig. 6 Full simulation setup for self-balancing robot

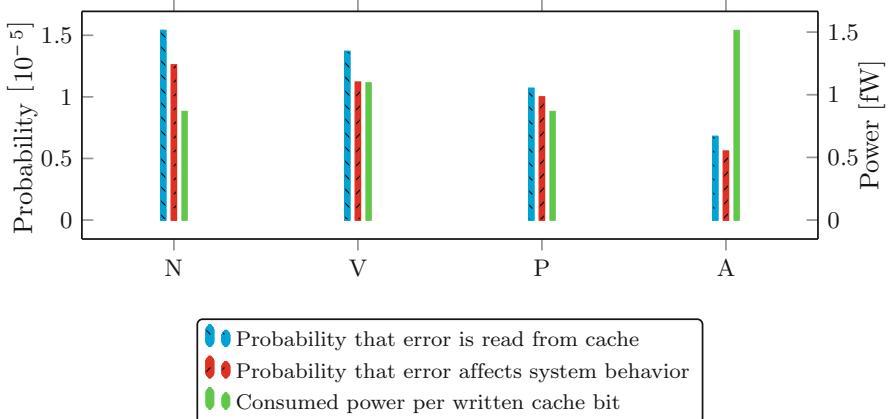


Fig. 7 Resilience exploration for cache of self-balancing robot

caused when a faulty cache line is read. The red bar shows those cases of silent data corruption that significantly affect the system behavior, which we classify as failure. The difference between the blue and red bar denotes the inherent resilience of the system. For hardening the system, increasing the supply voltage (V) decreases the silent data corruption rate (blue) and failure rate (red) but also increases the required power per written cache bit (green). Increasing the area (A) decreases the silent data corruption rate and failure rate more effectively compared to increasing the supply voltage but at the cost of a larger increase in power. In contrast, the parity protection (P) behaves differently to the hardening solutions. While parity also decreases the rate of silent data corruption (blue), we see that those remaining errors that are read from the cache (caused by an even number of upsets in the cache line) relatively often influence the system behavior (red), which is classified as failure. In the case of 1-bit parity protection the system is effectively protected from an odd number of

errors in each cache line. Yet, compared to the nominal case the failure probability of the system is only slightly reduced. The even number of upsets (mostly two bit upsets) are causing more often a failure than the detected single bit upsets. Upsets with three and more bits are not as relevant as they are very rare events. The key insight is that decreasing silent data corruptions thus does not necessarily result in a similar improvement in failure rate when considering the inherent resilience of the CPS application.

5 Conclusions

This chapter covered the fast evaluation of resilience against radiation-induced soft errors with multi-level/mixed-mode fault injection approaches as well as the systematic exploration of protection techniques that collaborate in a cross-layer fashion across the system stack. The methods were shown for case studies on custom processors, accelerators, third-party micro-controllers, and an SRAM-based cache.

Although this chapter has focused on radiation-induced soft errors, our cross-layer methodology and framework are equally effective at protecting against additional error sources such as supply voltage variations, early-life failures, circuit aging, and their combinations. For example, [6] demonstrates that cost-effective protection against supply voltage variation is achieved using Critical Path Monitor (CPM) circuit failure prediction and instruction throttling at 2.5% energy cost for a 64 in-order core design.

For error sources (such as early-life failures and circuit aging) that result from system degradation over longer duration of time (days to years), periodic on-line self-test and diagnostic are particularly effective at generating signatures to observe such degradation [27, 32, 34]. Since many of the resilience techniques considered in this chapter operate independently of the underlying error source, our conclusions regarding these particular techniques are broadly applicable.

Finally, an open question that remains is how to efficiently exploit the inherent resilience of CPS workloads. Full system simulation can help in a fast evaluation, but it remains to be seen in future research how the cost of resilience can be reduced by fully exploiting this potential in a cross-layer fashion.

References

1. Arlat, J., Crouzet, Y., Karlsson, J., Folkesson, P., Fuchs, E., Leber, G.H.: Comparison of physical and software-implemented fault injection techniques. *IEEE Trans. Comput.* **52**(9), 1115–1133 (2003). <https://doi.org/10.1109/TC.2003.1228509>
2. Barker, K., Benson, T., Campbell, D., Ediger, D., Gioiosa, R., Hoisie, A., Kerbyson, D., Manzano, J., Marquez, A., Song, L., Tallent, N., Tumeo, A.: PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual. Pacific Northwest National Laboratory and Georgia Tech Research Institute (2013). <http://hpc.pnnl.gov/projects/PERFECT/>

3. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The GEM5 simulator. *ACM SIGARCH Comput. Archit. News* **39**(2), 1–7 (2011)
4. Bottani, C., Glorieux, M., Daveau, J.M., Gasiot, G., Abouzeid, F., Clerc, S., Naviner, L., Roche, P.: Heavy ions test result on a 65 nm Sparc-V8 radiation-hard microprocessor. In: 2014 IEEE International Reliability Physics Symposium, pp. 5F.5.1–5F.5.6 (2014). <https://doi.org/10.1109/IRPS.2014.6861096>
5. Campbell, K.A., Vissa, P., Pan, D.Z., Chen, D.: High-level synthesis of error detecting cores through low-cost modulo-3 shadow datapaths. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2015). <https://doi.org/10.1145/2744769.2744851>
6. Cheng, E.: Cross-layer resilience to tolerate hardware errors in digital systems. Ph.D Dissertation, Stanford University (2018)
7. Cheng, E., Mirkhani, S., Szafaryn, L.G., Cher, C., Cho, H., Skadron, K., Stan, M.R., Lilja, K., Abraham, J.A., Bose, P., Mitra, S.: Tolerating soft errors in processor cores using clear (cross-layer exploration for architecting resilience). *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**(9), 1839–1852 (2018). <https://doi.org/10.1109/TCAD.2017.2752705>
8. Cho, H., Cheng, E., Shepherd, T., Cher, C.Y., Mitra, S.: System-level effects of soft errors in uncore components. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **36**(9), 1497–1510 (2017). <https://doi.org/10.1109/TCAD.2017.2651824>
9. Cho, H., Mirkhani, S., Cher, C.Y., Abraham, J.A., Mitra, S.: Quantitative evaluation of soft error injection techniques for robust system design. In: Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, pp. 1–10. IEEE, Piscataway (2013)
10. Civera, P., Macchiarulo, L., Rebaudengo, M., Reorda, M.S., Violante, M.: An FPGA-based approach for speeding-up fault injection campaigns on safety-critical circuits. *J. Electron. Testing* **18**(3), 261–271 (2002). <https://doi.org/10.1023/A:1015079004512>
11. Dixit, A., Heald, R., Wood, A.: Trends from ten years of soft error experimentation. In: System Effects of Logic Soft Errors (SELSE) (2009)
12. Ebrahimi, M., Sayed, N., Rashvand, M., Tahoori, M.B.: Fault injection acceleration by architectural importance sampling. In: Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis, CODES '15, pp. 212–219. IEEE Press, Piscataway (2015). <http://dl.acm.org/citation.cfm?id=2830840.2830863>
13. Entrena, L., Garcia-Valderas, M., Fernandez-Cardenal, R., Lindoso, A., Portela, M., Lopez-Ongil, C.: Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. *IEEE Trans. Comput.* **61**(3), 313–322 (2012). <https://doi.org/10.1109/TC.2010.262>
14. Espinosa, J., Hernandez, C., Abella, J., de Andres, D., Ruiz, J.C.: Analysis and RTL correlation of instruction set simulators for automotive microcontroller robustness verification. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2015). <https://doi.org/10.1145/2744769.2744798>
15. Fidalgo, A.V., Alves, G.R., Ferreira, J.M.: Real time fault injection using a modified debugging infrastructure. In: 12th IEEE International On-Line Testing Symposium (IOLTS'06), p. 6 (2006). <https://doi.org/10.1109/IOLTS.2006.53>
16. Foutris, N., Kaliorakis, M., Tselonis, S., Gizopoulos, D.: Versatile architecture-level fault injection framework for reliability evaluation: a first report. In: 2014 IEEE 20th International On-Line Testing Symposium (IOLTS), pp. 140–145. IEEE, Piscataway (2014)
17. Gaisler, A.: LEON3 processor. <http://www.gaisler.com>
18. Georgakoudis, G., Laguna, I., Nikolopoulos, D.S., Schulz, M.: REFINE: realistic fault injection via compiler-based instrumentation for accuracy, portability and speed. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, pp. 29:1–29:14. ACM, New York (2017). <https://doi.org/10.1145/3126908.3126972>
19. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Simunic, T., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**, 8–23 (2013)

20. Hari, S.K.S., Adve, S.V., Naeimi, H.: Low-cost program-level detectors for reducing silent data corruptions. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–12 (2012). <https://doi.org/10.1109/DSN.2012.6263960>
21. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.J.: Design and architectures for dependable embedded systems. In: Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) pp. 69–78 (2011)
22. Henning, J.L.: SPEC CPU2000: measuring CPU performance in the new millennium. Computer **33**(7), 28–35 (2000). <https://doi.org/10.1109/2.869367>
23. Herkersdorf, A., Aliee, H., Engel, M., Gläß, M., Gimmler-Dumont, C., Henkel, J., Kleeberger, V.B., Kochte, M.A., Kühn, J.M., Mueller-Gritschneider, D., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. Microelectron. Reliab. **54**(6), 1066–1074 (2014)
24. Höller, A., Macher, G., Rauter, T., Iber, J., Kreiner, C.: A virtual fault injection framework for reliability-aware software development. In: Dependable Systems and Networks Workshops, pp. 69–74 (2015)
25. Kaliorakis, M., Tselonis, S., Chatzidimitriou, A., Fournis, N., Gizopoulos, D.: Differential fault injection on microarchitectural simulators. In: 2015 IEEE International Symposium on Workload Characterization (IISWC), pp. 172–182. IEEE, Piscataway (2015)
26. Kanawati, G.A., Kanawati, N.A., Abraham, J.A.: FERRARI: a flexible software-based fault and error injection system. IEEE Transactions on Computers **44**(2), 248–260 (1995). <https://doi.org/10.1109/12.364536>
27. Kim, Y.M.: Early-life failures in digital logic circuits. Ph.D Dissertation, Stanford University (2013)
28. Kleeberger, V.B., Gimmler-Dumont, C., Weis, C., Herkersdorf, A., Mueller-Gritschneider, D., Nassif, S.R., Schlichtmann, U., Wehn, N.: A cross-layer technology-based study of how memory errors impact system resilience. IEEE Micro **33**(4), 46–55 (2013). <https://doi.org/10.1109/MM.2013.67>
29. Kleeberger, V.B., Mueller-Gritschneider, D., Schlichtmann, U.: Technology-aware system failure analysis in the presence of soft errors by mixture importance sampling. In: 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS) (2013)
30. Lee, H., Song, Y., Shin, H.: SFIDA: a software implemented fault injection tool for distributed dependable applications. In: Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, vol. 1, pp. 410–415 (2000). <https://doi.org/10.1109/HPC.2000.846589>
31. Li, M.L., Ramachandran, P., et al.: Accurate microarchitecture-level fault modeling for studying hardware faults. In: 2009 IEEE 15th International Symposium on High Performance Computer Architecture (2009)
32. Li, Y.: Online self-test, diagnostics, and self-repair for robust system design. Ph.D Dissertation, Stanford University (2013)
33. Li, Y., Mutlu, O., Gardner, D.S., Mitra, S.: Concurrent autonomous self-test for uncore components in system-on-chips. In: 2010 28th VLSI Test Symposium (VTS), pp. 232–237 (2010). <https://doi.org/10.1109/VTS.2010.5469571>
34. Lorenz, D., Barke, M., Schlichtmann, U.: Monitoring of aging in integrated circuits by identifying possible critical paths. Microelectron. Reliab. **54**(6), 1075–1082 (2014)
35. Maniatakos, M., Karimi, N., Tirumurti, C., Jas, A., Makris, Y.: Instruction-level impact analysis of low-level faults in a modern microprocessor controller. IEEE Trans. Comput. **60**(9), 1260–1273 (2011). <https://doi.org/10.1109/TC.2010.60>
36. Meixner, A., Bauer, M.E., Sorin, D.: Argus: low-cost, comprehensive error detection in simple cores. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), pp. 210–222 (2007). <https://doi.org/10.1109/MICRO.2007.18>

37. Mueller-Gritschneider, D., Dittrich, M., Weinzierl, J., Cheng, E., Mitra, S., Schlichtmann, U.: ETISS-ML: a multi-level instruction set simulator with RTL-level fault injection support for the evaluation of cross-layer resiliency techniques. In: 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 609–612 (2018)
38. Mueller-Gritschneider, D., Sharif, U., Schlichtmann, U.: Performance and accuracy in soft-error resilience evaluation using the multi-level processor simulator ETISS-ML. In: Proceedings of the International Conference on Computer-Aided Design, ICCAD '18, pp. 127:1–127:8. ACM, New York (2018). <https://doi.org/10.1145/3240765.3243490>
39. Oetjens, J.H., Bannow, N., Becker, M., Bringmann, O., Burger, A., Chaari, M., Chakraborty, S., Drechsler, R., Ecker, W., Grüttner, K., Kruse, T., Kuznik, C., Le, H.M., Mauderer, A., Müller, W., Müller-Gritschneider, D., Poppen, F., Post, H., Reiter, S., Rosenstiel, W., Roth, S., Schlichtmann, U., von Schwerin, A., Tabacaru, B.A., Viehl, A.: Safety evaluation of automotive electronics using virtual prototypes: state of the art and research challenges. In: Proceedings of the 51st Annual Design Automation Conference, DAC '14, pp. 113:1–113:6. ACM, New York, NY, USA (2014). <https://doi.org/10.1145/2593069.2602976>
40. OpenRISC 1000 architecture manual v1.1 (2014). <http://opencores.org>
41. Portela-Garcia, M., Lopez-Ongil, C., Valderas, M.G., Entrena, L.: Fault injection in modern microprocessors using on-chip debugging infrastructures. IEEE Trans. Dependable Secure Comput. **8**(2), 308–314 (2011). <https://doi.org/10.1109/TDSC.2010.50>
42. Pouchet, L.N., Yuki, T.: Polybench. <https://sourceforge.net/projects/polybench/>
43. Sanda, P.N., Kellington, J.W., Kudva, P., Kalla, R., McBeth, R.B., Ackaret, J., Lockwood, R., Schumann, J., Jones, C.R.: Soft-error resilience of the IBM POWER6 processor. IBM J. Res. Dev. **52**(3), 275–284 (2008). <https://doi.org/10.1147/rd.523.0275>
44. Stott, D.T., Ries, G., Hsueh, M.C., Iyer, R.K.: Dependability analysis of a high-speed network using software-implemented fault injection and simulated fault injection. IEEE Trans. Comput. **47**(1), 108–119 (1998). <https://doi.org/10.1109/12.656094>
45. Wang, N.J., Mahesri, A., et al.: Examining ace analysis reliability estimates using fault-injection. In: Proceedings of the 34th Annual International Symposium on Computer Architecture (2007)
46. Wang, N.J., Quek, J., Rafacz, T.M., Patel, S.J.: Characterizing the effects of transient faults on a high-performance processor pipeline. In: International Conference on Dependable Systems and Networks, 2004, pp. 61–70 (2004). <https://doi.org/10.1109/DSN.2004.1311877>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third-party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Online Test Strategies and Optimizations for Reliable Reconfigurable Architectures



Lars Bauer, Hongyan Zhang, Michael A. Kochte, Eric Schneider,
Hans-Joachim Wunderlich, and Jörg Henkel

1 Introduction and Motivation

Runtime/reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) are a promising augment to conventional processor architectures such as Central Processing Units (CPUs) and Graphic Processing Units (GPUs). Since the reconfigurable parts are typically manufactured in the latest technology, they may suffer from aging and environmentally induced dependability threats. In this chapter, strategic online test methods for dependable runtime-reconfigurable architectures as well as cross-layer optimizations for high reliability and lifetime are developed. Firstly, two orthogonal online tests are proposed that ensure reliable configuration of the reconfigurable fabric and aid fault detection. Secondly, a novel design method called module diversification is presented that enables self-repair of the system in case of faults caused by degradation effects as well as single-event upsets in the configuration. Thirdly, a novel stress-aware placement method is proposed that aims for slowing down system degradation by aging effects. The combined methods ensure reliable operation across architectural and gate level and allow to prolong the lifetime of dependable runtime-reconfigurable architectures.

The dependable operation of VLSI circuits is not only threatened by test escapes, intermittent or transient errors, but also by emerging hardware defects due to *aging* [11–13]. In nano-scale CMOS circuits, aging is related to *stress* which is defined as the condition under which a circuit structure experiences electrical and physical

L. Bauer (✉) · H. Zhang · J. Henkel

Chair for Embedded Systems, Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: lars.bauer@kit.edu; henkel@kit.edu

M. A. Kochte · E. Schneider · H.-J. Wunderlich

Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Stuttgart, Germany
e-mail: schneiec@iti.uni-stuttgart.de; wu@informatik.uni-stuttgart.de

Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

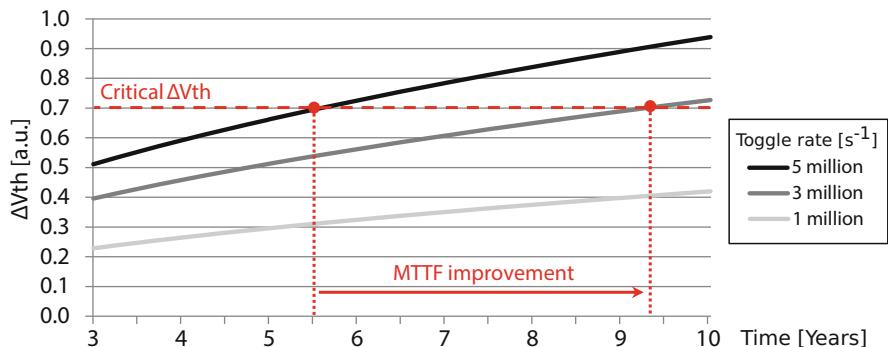
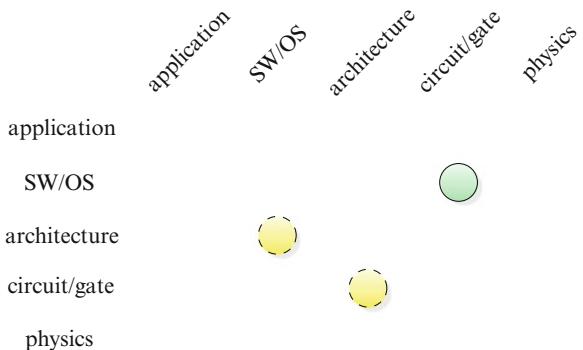


Fig. 2 Threshold voltage increase due to HCl-related stress (based on [22])

degradations. Two types of stress are distinguished: *static stress* and *dynamic stress*. Dynamic stress is typically characterized by the toggle rate of a transistor during which high currents flow between drain and source. A transistor is under static stress when an electric field is exerted across its gate oxide to induce a conducting channel. The stress is characterized by the duty cycle, i.e., the fraction of operation time the transistor is conducting. Dynamic stress leads to aging effects like Hot Carrier Injection (HCl), while static stress can lead to Bias Temperature Instability (BTI). Both are dominating aging mechanisms in nano-CMOS technologies [8, 16] and cause shifts in the threshold voltage ΔV_{th} of a transistor, which ultimately impacts the device performance over time. In this chapter, strategic online test methods for dependable runtime-reconfigurable architectures as well as cross-layer optimizations for high reliability and lifetime are developed (see Fig. 1).

The Mean Time to Failure (MTTF) of a transistor is defined as the time until its threshold voltage exceeds a certain critical value at which the transistor cannot deliver the required performance anymore. As shown in Fig. 2, the MTTF can be greatly increased if the transistor stress and consequently the threshold voltage shift are reduced.

Different aging models exist [2, 8], which indicate that both dynamic and static stress are generally *additive* through accumulation of the degradation effects. As a

result, this additive stress accumulation causes a *monotonic* increase in the transistor degradation over long terms. Although BTI degradation may experience a recovery effect, the recovery requires complex conditions or long relaxation periods [10] and will thus hardly affect the additive property. The monotonic and additive properties allow to consider stress during runtime (e.g., for resource management) with limited computational resources.

1.1 Application Model

In this work, a general application model is considered, as shown in Fig. 3. An application (Fig. 3a) consists of a mixture of normal operations, e.g., memory allocation and data preparation, and one or multiple computationally intensive parts, the so-called *kernels*. A kernel (Fig. 3b) corresponds to an outer loop that iterates through the whole data set and that contains one or multiple inner loops that work on small data parts, specified by the current iteration of the outer loop. For example, in a stencil operation of an image, the outer loop iterates over each output pixel and the inner loop computes the output value based on multiple neighboring input pixel values. Such an inner loop is a good candidate to be implemented as a *Special Instruction* (SI) that is composed of one or multiple *accelerators* of potentially different types. An SI (Fig. 3c) is represented by a data-flow graph (DFG) where each node corresponds to an accelerator and the edges correspond to data-flow between the accelerators [4]. Before the execution of an SI, all required accelerators need to be configured into the reconfigurable fabric, or otherwise the SI has to be emulated in software on the GPP. A sophisticated H.264 video encoder is the main application used for evaluation. The encoder consists of three kernels that require different SIs, implemented by nine types of accelerators [6].

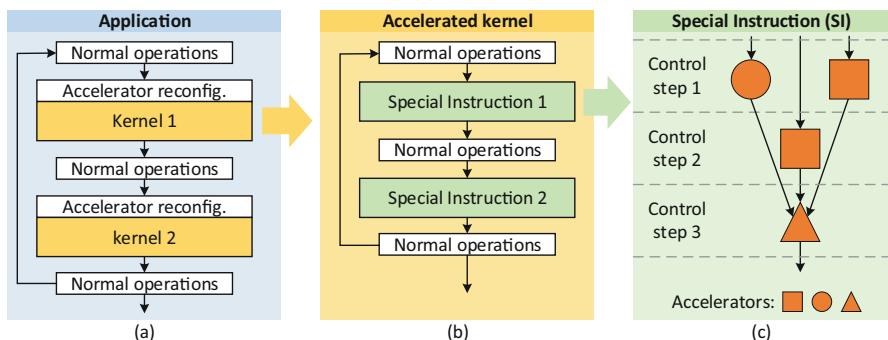


Fig. 3 This generic application model considers applications that consist of one or multiple kernels that may use Special Instructions (SIs) that are implemented by accelerators (based on [23])

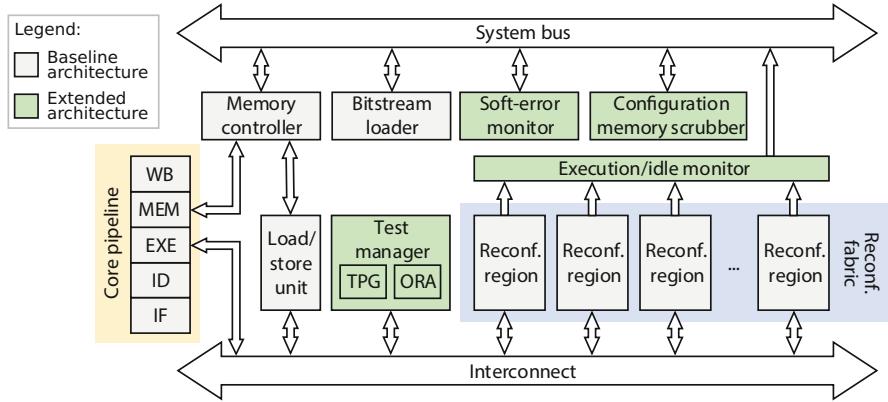


Fig. 4 Target reconfigurable architecture (based on [7])

1.2 Runtime-Reconfigurable Architectures

Runtime reconfiguration enables dynamic hardware customization to adapt to changing application requirements or environmental constraints, which maximizes performance at very low energy consumption. A reconfigurable architecture consists of a general-purpose processor and a *reconfigurable fabric*, partitioned into multiple *reconfigurable regions* (used to implement application-specific accelerators on-demand) that are interconnected via a communication infrastructure.

This chapter presents *Online Test Strategies for Reliable Reconfigurable Architectures* (OTERA), which targets FPGA-based fine-grained reconfigurable architectures as shown in Fig. 4. While transient faults due to single-event upsets are also addressed by OTERA (more details in [7]), this chapter focuses on aging-related challenges. To support dependable operation by online testing, stress balancing, and resource management for reliability and graceful degradation, a reconfigurable baseline architecture is extended by the following components:

- a test manager including a *test-pattern generator* (TPG) and an *output response analyzer* (ORA) to perform structural tests on the reconfigurable fabric and functional tests on the reconfigured accelerators;
- a workload monitor to track when a region is reconfigured and how often the currently configured accelerator is executed, which is used for stress estimation;
- a configuration memory scrubber to detect and correct errors in the configuration memory by periodical read-back and check of the configuration;
- a runtime system for dynamic dependability management by environmental monitoring, online test, reliability management, and aging mitigation.

The architecture is implemented using a LEON processor [9] and a parameterizable number of reconfigurable regions. A SystemC-based cycle-accurate simulator is used to evaluate the architecture and its runtime system. A hardware prototype is

developed on a Xilinx Virtex-5 FPGA and operates at a clock frequency of 100 MHz with a reconfiguration bandwidth of 50 MB/s.

FPGA hardware is composed of a two-dimensional array of reconfigurable primitive logic elements and routing structures that logic functions are mapped to. The two essential components are Configurable Logic Blocks (CLBs) and Programmable Switching Matrices (PSMs). The CLBs are the basic reconfigurable resources for implementing combinatorial and sequential logic functions. The interconnection between the components is configured using the PSMs. The logic function in a reconfigurable region is determined by configuration bits, called its *bitstream*, stored in SRAM-based configuration memory. Modern FPGAs support partial reconfiguration and allow to change the logic function without interrupting the operation in other parts of the chip [19].

An FPGA-based reconfigurable fabric, manufactured in latest technology nodes (e.g., 16 nm for Xilinx' UltraScale+ family), may suffer from degradation due to aging [10, 18]. Due to the increasing susceptibility of ever-shrinking nano-CMOS devices, these effects cannot be ignored anymore [11–13]. The resilience of the reconfigurable fabric is essential to the dependability of reconfigurable architectures, since most of the application's computations are offloaded to the fabric. The dependable operation of a hardware accelerator in the reconfigurable fabric relies on both the structural integrity of the fabric and the accelerator's functional correctness. While structural integrity of the reconfigurable fabric is a prerequisite for functional correctness of accelerators, the latter requires the correct completion of the reconfiguration process and correctness of the configuration data. However, the functionality of accelerators can be impacted during operation, for instance by SEUs that corrupt configuration data [7] as well as degradation of the hardware. To increase the dependability of the reconfigurable architecture, the structural integrity and functional correctness need to be addressed at different layers.

2 Fault Detection Through Strategic Online Testing

As latent defects and aging threaten the structural integrity of nano-CMOS devices, conventional manufacturing and burn-in tests are no longer sufficient to guarantee dependable operation over the whole lifetime. Therefore, *online tests* are required to check the system functionality. This task is particularly challenging for runtime-reconfigurable architectures, since the hardware organization changes during runtime as part of the normal operation [4]. This chapter presents two complementing types of online tests that are scheduled concurrently by the runtime system: *pre-configuration online tests (PRET)* and *post-configuration online tests (PORT)*.

2.1 Generation and Runtime Scheduling of Online Tests

PRET is designed to exhaustively test the underlying hardware structure in the reconfigurable fabric (e.g., logic resources in CLBs) periodically or on-demand. For PRET, an array-based structural test approach is used to generate *test configurations* for the exhaustive test of all logic resources in a reconfigurable region [1, 5]. Additional PRET test configurations are generated to target the application-dependent interconnects [6].

Since errors may also occur during the loading of bitstreams (e.g., due to faults in the configuration logic or transient events like SEUs), the configured function of the targeted region may be wrong or the configuration in other parts of the reconfigurable fabric may be adversely altered. For this reason, PORT is designed to perform at-speed functional tests on accelerators after their instantiation to ensure that they were configured correctly. At runtime, PORT also periodically checks the accelerators for malfunctions due to emergent permanent faults or soft errors in the configuration memory. An Automatic Test Pattern Generation (ATPG) tool is used to generate accelerator-specific test patterns to target the LUTs, combinational functions, and sequential elements in CLBs, as well as interconnects. The stuck-at fault model is used for components for which sufficient structural information is available to derive the faults and for the interconnects. For the remaining components, structural and cell faults are targeted during test generation resulting in a hybrid fault model [6].

Figure 5 shows the proposed online test flow for a reconfigurable fabric with three regions. In the first step (Fig. 5a), the runtime system decides that an accelerator shall be reconfigured into a particular region, which triggers the demand to test the hardware structures in that region before the actual configuration of accelerators (the so-called *on-demand PRET*). To exhaustively test all reconfigurable resources in the region, multiple *test configurations* (TCs) are required. The runtime system can choose to execute PRET incrementally to reduce the delay, applying only a subset of TCs (possibly none) prior to an accelerator reconfiguration. In practice, on-demand PRET-TCs are only scheduled after a certain number of *accelerator configurations* (ACs) have been configured. To reduce the impact on the application performance due to unavailable regions, PRET is only executed at times when the system needs to be reconfigured anyway. The runtime system tracks which TCs were applied to a region in the past and how much time passed since the last exhaustive PRET. Depending on this history, it activates PRET prior to an AC, reconfigures the selected TCs into the region, and uses TPG and ORA of the Test Manager to exercise the region (Fig. 5b).

In addition to on-demand PRETs, the runtime system also schedules *periodic* PRETs to ensure that seldom-reconfigured regions are properly tested. Note that PRET also needs to be executed regularly for regions that the application only reconfigures once and then never again (e.g., if the application only consists of one kernel; see Sect. 1.1). The reason is that PORT—despite its generally high fault coverage (see [6])—cannot always identify all faults. For instance, when an

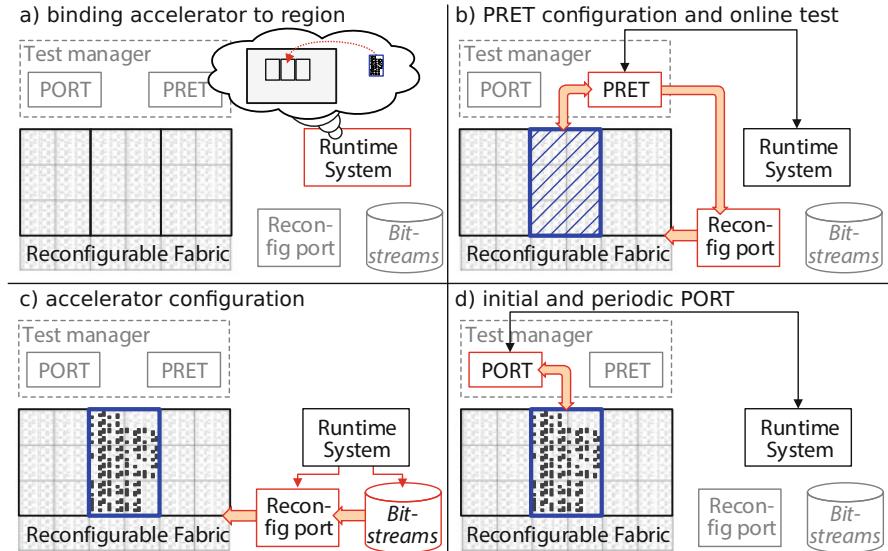


Fig. 5 Test flow with PRET and PORT (based on [6])

accelerator contains internal state, it is not always possible to apply an input value that propagates a possibly faulty value to an observable output. The periodic PRET is implemented using a timer interrupt and a handler that consists of two phases: (1) triggering the reconfiguration of a TC for a particular region and (2) executing PRET after the TC is reconfigured.

If no structural fault is found by PRET, the runtime system reconfigures the desired accelerator into the region (Fig. 5c). Before the accelerator is used by the application, the runtime system triggers an *on-demand PORT* (Fig. 5d) to test whether the reconfiguration process has completed without error. Additionally, accelerators instantiated in other regions are tested as well to check that they were not adversely affected by the reconfiguration. As PORT does not require any reconfiguration of TCs, it operates significantly faster than PRET and is also scheduled periodically during normal operation.

2.2 Online Test Integration

The test manager, TPG, and ORA are integrated into the reconfigurable architecture and coupled to the interconnect for the reconfigurable fabric such that communication channels between the regions and the test manager can be established. PRET and PORT are implemented as dedicated test-SI. In the base architecture, all SIs implicitly configure the interconnect infrastructure for the required data-flow

among accelerators and the system. The test-SIs reuse this mechanism to establish the connections between the test manager and the regions under test.

When the runtime system initiates a test-SI, the test parameters such as the target region or selection of test patterns are sent as the SI input data from the register file of the processor to the test manager. The test manager then generates the patterns by the TPG or sends stored patterns to the regions. While the PRET responses are sent back to the test manager for comparison, the PORT responses are compacted locally in space and time using a 32-bit multiple input signature register (MISR). The MISR is integrated into the interconnect infrastructure such that the outputs and the bus interface of a region are tested as well. After the test, the locally stored signatures are transferred to the test manager and compared with the expected signatures that are specific for each accelerator. At the end of PRET, the pass/fail information is written back to the register file of the processor. On-demand PORT is executed directly after an accelerator configuration to assure that the reconfiguration process completed without error and that the configured accelerator delivers the expected functionality. As PORT tests *all* configured accelerators in one test session, errors in the other accelerators, e.g., due to address decoder faults, are detected as well.

2.3 Experimental Evaluation

The effectiveness of PRET and PORT as well as the impact on the system performance is evaluated for the targeted platform. A test session consists of multiple test configurations (TCs) as shown in Table 1. In total nine TCs are required to test all logic primitives in the CLBs [1], and another nine TCs are required to test the interconnects of the accelerators of the H.264 application [6]. Each TC tests a subset of the logic primitives in the CLBs of a region or a set of interconnects used by the accelerator to be configured (Column 2). Columns 3 and 4 give the area overhead of PRET and the size of the generated partial bitstreams. The total area overhead introduced by PRET for all TCs is 17 CLBs. That is a one-time overhead to implement the test-pattern generator (TPG) and output response analyzer (ORA) for PRET, independent of which reconfigurable region is to be tested, whereas the other numbers in the table are per reconfigurable region. Note that the configuration time with tens of thousands of cycles dominates the actual application of the test patterns (Column 6).

The PRET overhead for the interconnect TCs is not applicable as the deterministic patterns are not generated by a TPG but stored similar to PORT patterns. The responses are compacted in the MISR introduced for PORT. In total 3780 bytes are required to store the test patterns of all interconnect TCs together with their signatures. The interconnect test reaches a fault coverage of up to 100% with the lowest being 98.28% [6].

The application performance loss introduced by PRET depends on the test frequency and number of reconfigurable regions. In this experiment, architectures

Table 1 Test configurations for CLBs and interconnects for reconfigurable regions of 4×20 CLBs (based on [6])

TC	Tested primitives	PRET overhead [CLBs]	Bitstream size [KB]	Freq. [MHz]	Test length [Patterns]
1	LUT conf. as XOR, connected to FF	2	24.0	207	64
2	LUT conf. as XNOR, connected to FF	2	24.0	207	64
3	Carry MUX, interleaved with MUX and latch	1	28.6	168	6
4	Carry MUX, interleaved with MUX and latch	1	26.1	154	6
5	Carry XOR, interleaved with MUX and FF	1	28.0	168	6
6	Carry XOR, interleaved with MUX and FF	1	28.2	154	6
7	Carry-in/-out with multiplexed scan chain	1	27.1	183	6
8	LUT conf. as SR with slice MUX	1	22.9	157	6
9	LUT conf. as RAM with slice output	7	22.3	225	320
10–18	Interconnect and PIPs of 9 accelerators	n.a.	29.6	78.8–191.9	13–123

with 5 and up to 14 reconfigurable regions are considered. The PRET handler is triggered every 1 ms and performs PRET if a region has not been tested for 500 ms. The observed test latencies until a region is completely tested ranged from 3.8 to 8.1 s, i.e., emergent faults do not remain undetected in the system for longer than 1.9 to 4.05 s on average. Table 2 reports the PORT performance impact and test latency. The upper part of the table shows the performance impact for PORT frequencies from 143 to 1000 Hz, i.e., test intervals from 1 to 7 ms. For each PORT frequency, the table shows the minimum and maximum performance loss of ten reconfigurable systems with different number of regions (5–14). The performance overhead due to PORT is very low (between 0.51% and 3.73%) and scales well with higher PORT frequencies. The observed worst case test latency, which corresponds to the longest untested time period of a region, is shown in the lower part of Table 2.

With PRET and PORT both enabled, the system is able to defend the configured accelerators against structural faults induced by aging effects or latent faults and transient events such as radiation [6]. For a PORT frequency of less than 100 Hz, the performance loss was dominated by the configuration frequency. After that point, the PORT frequency dominates the performance loss. The highest observed performance loss of only 4.4% occurs for a PORT frequency of 1000 Hz and a configuration frequency of 41 Hz.

Table 2 Performance loss and worst case test latency under PORT (based on [6])

		PORT application frequency [Hz]						
		143	167	200	250	333	500	1000
Performance loss	min. ^a [%]	0.51	0.59	0.72	0.89	1.20	1.81	3.68
	max. ^a [%]	0.56	0.63	0.75	0.92	1.23	1.85	3.73
Worst case test latency ^b	min. ^a [ms]	7.0	6.0	5.0	4.1	3.3	2.3	1.7
	max. ^a [ms]	7.8	6.8	5.8	4.8	3.8	2.8	1.8

^aSummarizing ten reconfigurable systems with 5–14 regions

^bCorresponds to the longest time period in the whole runtime in which a configured accelerator remains untested

3 Self-Repair by Module Diversification

Using PRET and PORT we can *detect* faults in the reconfigurable fabric. We now present a design method called *module diversification* [21] that generates a set of *diversified* configurations for each module/accelerator to *tolerate* any single-CLB fault and part of multi-CLB faults. The diversified configurations of an accelerator provide all the same functionality, but they vary in their CLB usage. They are reconfigured into the region at runtime without performance degradation. If a faulty CLB is detected, it is isolated from the system (i.e., a configuration is chosen that does not use it) to avoid any errors.

3.1 Diversified Configurations

A module defines the logic functions to be implemented in a region which consists of CLBs that are arranged regularly in a 2-dimensional array in the FPGA fabric. The CLB usage of a configuration is described by a *configuration matrix* as shown in Eq. (1) whose dimensions $X \times Y$ match the width X and height Y of a region in CLBs. If a configuration uses a certain CLB, the corresponding element in the matrix is 1, otherwise 0. For each module, a set $C = \{A_1, \dots, A_w\}$ of configurations matrices with different CLB usage is generated. To be able to tolerate any single-CLB fault, this set of configurations must satisfy the *completeness condition* (Eq. (2)), which ensures that for any CLB in a region at least one diversified configuration A_i exists where the CLB is not used. Given that all diversified configurations implemented in a $X \times Y$ region occupy the same amount $U(< X \cdot Y)$ of CLBs (with at least one free CLB) a minimum number of w_{min} configurations (Eq. (3)) is required for the completeness condition [21].

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (1)$$

$$\forall x, y, 1 \leq x \leq X, 1 \leq y \leq Y : \exists \mathbf{A}_i \in C \text{ with } [\mathbf{A}_i]_{x,y} = 0 \quad (2)$$

$$w_{min} := \lceil \frac{X \cdot Y}{X \cdot Y - U} \rceil \quad (3)$$

Two configurations $\mathbf{A}_i, \mathbf{A}_j \in C$ are said to be *maximally diversified* if their difference in the CLB usage is maximized. The *max diversification condition* [21] states that for every configuration $\mathbf{A}_i \in C$ there exists a maximally diversified configuration $\mathbf{A}_j \in C$ with a common number of CLBs:

$$\forall i, 1 \leq i \leq w_{min} : \exists \mathbf{A}_j \in C, j \neq i \text{ such that}$$

$$\sum_{x,y} ([\mathbf{A}_i]_{xy} \cdot [\mathbf{A}_j]_{xy}) = \begin{cases} 2U - X \cdot Y & \text{if } (U > \frac{1}{2}X \cdot Y) \\ 0 & \text{else.} \end{cases} \quad (4)$$

3.2 Generation Algorithm

Algorithm 1 allows to generate maximally diversified configurations that satisfy the completeness condition [21]. Starting from an initial configuration \mathbf{A}_1 (Line 1) of a module, it incrementally generates diversified versions. A score matrix \mathbf{G} stores

Algorithm 1 Generation of diversified configurations C

```

1.  $C := \{\mathbf{A}_1\}$  //  $\mathbf{A}_1$  is the initial configuration ( $X \times Y$ )
2.  $\mathbf{G} := \mathbf{A}_1$  // Score matrix  $\mathbf{G}$  stores swapping priority of CLBs ( $X \times Y$ )
3.  $\mathbf{A}_{new} := \mathbf{A}_1$ 
4. while  $|C| \neq$  desired number of config.  $\wedge |C| \neq \binom{XY}{U}$  do
5.   zero_elem_list :=  $\{(x, y) \mid [\mathbf{A}_{new}]_{xy} = 0\}$  // unused CLBs
6.   cand_list :=  $\{(x, y) \mid [\mathbf{A}_{new}]_{xy} = 1\}$  // candidate list
7.   sort cand_list in descending order according to the score in  $\mathbf{G}_{xy}$ 
8.   for all  $(x, y)$  in zero_elem_list do
9.     swap_candidates :=  $\{(p, q) \mid (p, q) \in \text{cand\_list} \text{ and } \mathbf{G}_{pq} = \mathbf{G}_{\text{cand\_list}[0]}\}$  // all CLBs with the highest score
10.    farthest_swap_candidate :=  $(p, q) \in \text{swap\_candidates}$  with max. Manhattan distance between  $(x, y)$  and  $(p, q)$ 
11.    swap  $([\mathbf{A}_{new}]_{xy}, [\mathbf{A}_{new}]_{\text{farthest\_swap\_candidate}})$ 
12.    cand_list.pop(farthest_swap_candidate)
13.    if cand_list =  $\emptyset$  then
14.      break
15.    end if
16.  end for
17.  while  $\mathbf{A}_{new} \in C$  do
18.    swap a random zero- with random one-element in  $\mathbf{A}_{new}$ 
19.  end while
20.   $\mathbf{G} := \mathbf{G} + \mathbf{A}_{new}$  // update CLB score
21.   $C := C \cup \{\mathbf{A}_{new}\}$ 
22. end while

```

for each CLB the number of available diversified configurations in C that use the respective CLB resources. The new configuration matrix \mathbf{A}_{new} is initialized by \mathbf{A}_1 and modified in the inner loop (Lines 8–16) by swapping zero- and one-elements. The loop iterates over each element in \mathbf{A}_{new} and swaps all zero-elements with one-elements in an order given by the score matrix (Line 7). If a CLB has a higher score, it is used more often in the diversified configurations. Thus the corresponding one-element in \mathbf{A}_{new} will be swapped first. If CLBs have the same score, the distance-wise farthest one from the current zero-element is swapped first (Lines 9–11) so that the used CLBs are located near each other in the resulting configuration. The first w_{\min} generated configurations correspond to the *minimal* set of configurations [21]. More configurations can be generated to achieve higher reliability or more alternatives during stress balancing (see Sect. 4). Random swapping in Line 18 allows to shuffle CLBs with different stress profiles. The algorithm terminates when either the desired number of configurations or all possible configurations have been generated.

3.3 Experimental Evaluation

To evaluate the reliability improvement and timing costs, the presented method is applied to a set of functional modules from the MCNC benchmark suite [20] and OpenCores.¹ The dimensions of the reconfigurable regions were chosen as 20 CLBs in height (80 CLBs for large modules) and 3–13 CLBs in width, which provides different degrees of CLB redundancy. For each module and region size the minimal set of configurations is generated using the proposed module diversification method. Since the design method applies additional constraints to prohibit certain CLB placements (PROHIBIT commands in Xilinx tools), additional routing effort is introduced that can affect the maximum clock frequency. To assess the impact on the system performance, the maximum frequency of diversified modules was compared to the original configuration. Initially, the clock frequencies of the modules ranged from 122.4 MHz (apex2) to 150.8 MHz (pdc). Experiments show that the timing penalty of the diversified configurations ranges from 0.04% (aes_core) to 9.7% (misex3). While the maximal frequency is given by the slowest configuration of a module, the original implementation also belongs to the configuration set and can be used when full performance is required. Also, if the system frequency is lower than the maximal frequency of the diversified modules, there are no timing penalties at all. Thus, module diversification is a promising approach to obtain fault tolerance without additional area overhead and little to no cost in system performance.

The *reliability* of an entity is the probability that the entity can operate without failure over a time period t . Without any fault-tolerance techniques applied, the overall reliability of a module with U CLBs depends on the reliability $R_{\text{CLB}}(t)$ of each individual CLB (Eq. (5)). With module diversification, the reliability of the

¹<https://www.opencores.org>.

module changes, as shown in Eq. (6). The first term states the probability that all CLBs are fault free. The second term aggregates all possible scenarios of multiple fault occurrences until all CLBs become faulty. The fault coverage $C_f \in [0, 1]$ is the fraction of f -CLB faults which are detected by an online test or concurrent error detection scheme such that reconfiguration with a diversified configuration allows to continue the operation. The fraction of f -CLB faults which can be tolerated with the set of available configurations is denoted by $\alpha_f \in [0, 1]$.

$$R_{\text{No_FT}}(t) = (R_{\text{CLB}}(t))^U \quad (5)$$

$$R_{\text{Div}}(t) = R_{\text{CLB}}(t)^{XY} + \sum_{f=1}^{XY} C_f \alpha_f \underbrace{\binom{XY}{f} (1 - R_{\text{CLB}}(t))^f R_{\text{CLB}}(t)^{(XY-f)}}_{\text{Probability that } f\text{-fold CLB failures can be tolerated}} \quad (6)$$

We use the module `apex4` for the reliability analysis. Without fault-tolerance measures, the module has a very low reliability (≈ 0.91). Figure 6 shows the module reliability for a varying number of configurations and region sizes with CLB reliability $R_{\text{CLB}}(t) = 0.999$ and $C_f = 1.0$. The region size varies from 20×6 to 20×9 CLBs and corresponds to CLB redundancies from 22.4% to 111.8%. Larger region sizes reduce the overall module reliability since they have increased probability of a faulty CLB. By using diversified configurations, the module reliability increases dramatically. As shown, the tolerance of f -CLB faults rises with increasing number of configurations and very high module reliability is achieved (>0.999).

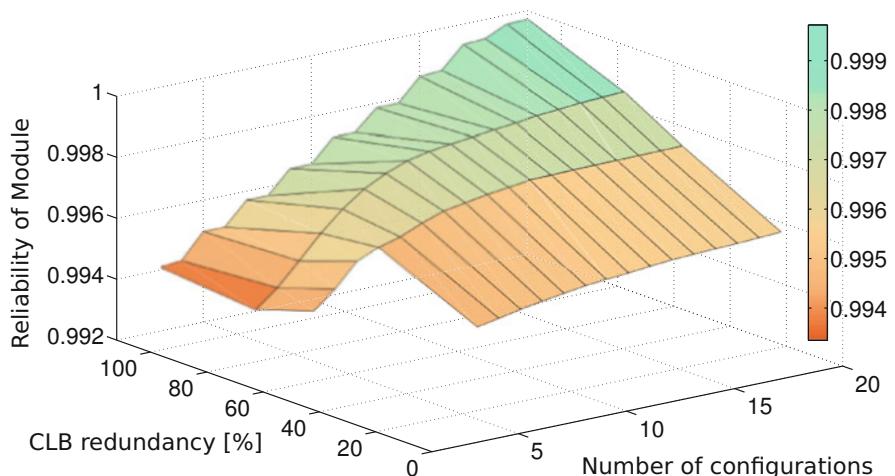


Fig. 6 Module reliability of `apex4` for different ratios of CLB redundancy and number of configurations with CLB reliability 0.999 (based on [21])

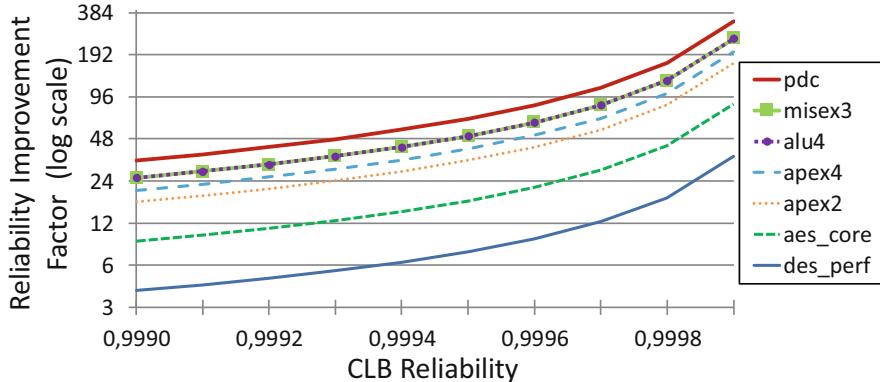


Fig. 7 Reliability improvement factor after module diversification (based on [21])

To estimate the effectiveness of the module diversification, the *reliability improvement factor* (RIF) is used [15]. The RIF is the ratio of the failure probability of the original system and the failure probability of the fault tolerant system using diversified module configurations (Eq. (7)). Figure 7 plots the RIF for the five investigated modules and CLB reliabilities ranging from 0.9990 to 0.9999. As shown, the proposed design method achieves reliability improvement factors of up to 330×.

$$\text{RIF} := \frac{1 - R_{\text{No FT}}}{1 - R_{\text{Div}}} \quad (7)$$

4 Prolonging Lifetime via Stress Balancing

In addition to *reacting* on detected faulty CLBs (e.g., by using diversified modules as in Sect. 3), it is of crucial importance to *proactively* delay the occurrence of permanent faults (or increasing transistor switching delay) by aging mitigation via stress balancing. Different aging mechanisms have been reported for the current generation of CMOS designs, as discussed in Sect. 1. The main causes of these effects are environmental and electrical *stress*. Stress can be induced in different ways, e.g., through the presence of strong electrical fields or high current density [17, 18]. We propose the novel STress-Aware Placement method STRAP that reduces the peak stress by aging mitigation. It combines complex offline optimizations at synthesis time with situation-dependent adaptation at runtime to optimize the intra- and inter-region stress distribution simultaneously. At runtime, STRAP places accelerators to different reconfigurable regions (i.e., it decides to which region they shall be reconfigured) while considering the induced intra- and inter-region stress distribution simultaneously. At synthesis time, STRAP diversifies stress during

place-and-route by preventing overlapping of high stress CLBs from different accelerators, which further improves the intra-region stress distribution at runtime.

4.1 Overview of the Stress-Aware Placement Method STRAP

The MTTF of a system is constrained by the component with the highest stress [17]. In order to prolong the MTTF of a reconfigurable fabric, stress accumulation on individual resources need to be avoided to reduce the peak stress. Figure 8a shows a typical reconfigurable fabric with 8 reconfigurable regions and 4×20 CLBs per region. The figure visualizes the distribution of HCI stress after running an H.264 video encoder. Higher HCI stress corresponds to more toggles per second of a transistor (see Sect. 1). For each CLB, the highest toggle rate of any transistor is identified and plotted in a color-scale from 0 (*low stress*, bright gray) to 20 million toggles per second (*high stress*, dark red). It is noticeable that several CLBs are not used (e.g., most parts of region 5), whereas some CLBs in region 1 contain transistors that are highly stressed. The latter represent *stress hotspots* where high stress accumulates in some of the components in the fabric which have a higher chance to fail much earlier than others, hence reducing the MTTF of the system.

The basic idea of STRAP is to place accelerators such that the *maximal* stress is minimized. Our *method* abstracts stress to the granularity of CLBs, whereas the *evaluation* of our method in Sect. 4.6 considers stress at transistor granularity. If the stress from a stress hotspot can be distributed to less stressed CLBs (like in region 5

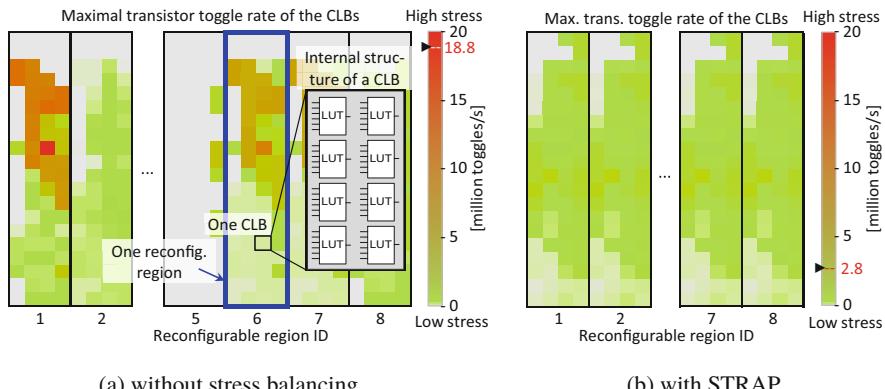


Fig. 8 Transistor stress distribution in a reconf. fabric with eight regions; each region consists of 4×20 CLBs with 8 LUTs each (same setup as for evaluation); the color of a CLB corresponds to the highest toggle rate of any of its transistors; the symbol “filled triangle right” on the scale denotes the maximum stress over all regions (based on [22]). **(a)** Conventional execution without stress balancing. **(b)** Stress-aware placement in STRAP

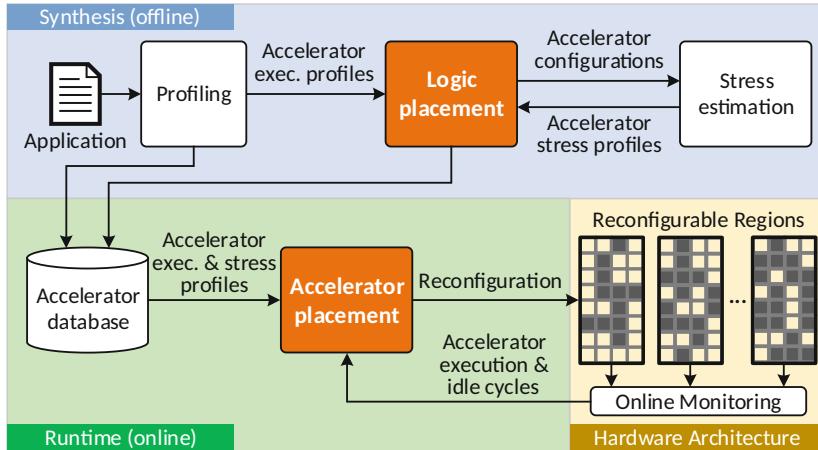


Fig. 9 Overview of the stress-aware placement method (based on [23])

in Fig. 8a), then the maximum stress in the reconfigurable regions is reduced (like in Fig. 8b), leading to increased MTTF.

Figure 9 provides an overview of the stress-aware placement method STRAP, showing the synthesis time techniques, the runtime techniques, and how they interact with the hardware architecture of a reconfigurable system. For logic placement at synthesis time, the challenge is to place-and-route accelerators in a way that supports stress balancing at runtime, but without having runtime information. STRAP first performs an offline application profiling of each application kernel to obtain estimates on (1) how often accelerators will be executed relative to each other and (2) how long each accelerator executes to finish its task. This information is used to steer runtime accelerator placement (Sect. 4.3) and synthesis time logic placement (Sect. 4.4).

Based on the accelerator configuration after place-and-route, the stress estimation process in Fig. 9 analyzes the signal activities in all CLBs used by the accelerator to obtain the information how much stress it induces to a reconfigurable region. Accelerator execution and stress profiles are stored together with the accelerator bitstreams in main memory for runtime decision making.

At runtime, STRAP decides into which reconfigurable region an accelerator shall be reconfigured, whenever the application demands different accelerators. It performs online monitoring of each region to track when the region was reconfigured last and how often the currently reconfigured accelerator was executed. Whenever a region is reconfigured, the execution counter and reconfiguration timestamp are read and reset. Together with the accelerator stress profile created at synthesis time, STRAP then calculates the exact *stress state* for all CLBs of the region. This information is used to decide the runtime accelerator placement.

4.2 Representation of Stress

Stress Granularity In order to handle the transistor stress in an algorithmic way, it needs to be represented compactly to allow an efficient runtime computation for the stress states of regions and the placement decision making. The transistors of a reconfigurable region are stressed by the reconfigured accelerator in a way that is determined by its logic functionality and input signal patterns. As the number of transistors in a region may be huge, the stress experienced by individual transistors is lumped to CLB granularity for the stress-aware placement method. *CLB stress* is defined as the sum of the stress experienced by all transistors in a CLB. With this definition, CLB stress preserves the additive property of transistor stress, i.e., the total stress a CLB experienced from different accelerators is the sum of the induced stress from individual accelerators.

Stress Accumulation With the established stress properties (see Sect. 1), the stress in the reconfigurable fabric can be described in a formal way. The stress state of a reconfigurable region (as it is visualized in Fig. 8) is denoted as matrix \mathbf{S} , where each entry represents the stress experienced by the corresponding CLB in the region. The stress that a particular accelerator induces per clock cycle is obtained from offline stress estimation and called *unit stress*, denoted by a matrix of the same size as \mathbf{S} . In general, the stress increase due to the work done by an accelerator is shown in Eq. (8). Matrices $\mathbf{s}_{\text{exec}}^{\text{unit}}$ and $\mathbf{s}_{\text{idle}}^{\text{unit}}$ denote the unit stress induced by the accelerator during execution or idle time and Sect. 4.6 explains how we use aging models to obtain these values by power/temperature analysis of placed-and-routed accelerators. Scalars τ_{exec} and τ_{idle} denote the number of clock cycles when the accelerator is executing or idle.

$$\mathbf{s} := \tau_{\text{exec}} \mathbf{s}_{\text{exec}}^{\text{unit}} + \tau_{\text{idle}} \mathbf{s}_{\text{idle}}^{\text{unit}} \quad (8)$$

The values for τ_{exec} and τ_{idle} are obtained from offline application profiling to construct the stress matrices (Eq. (8)) for every accelerator. The runtime system uses them to determine how much stress an accelerator would induce to a region *before* actually placing it. It also uses online monitoring (see Sect. 4.1) that provides the actual number of accelerator executions and idle times for each region *after* a computational kernel finished execution. This allows to keep track of the actual stress that a region experienced, which is the starting point for the next placement decision.

4.3 Runtime Accelerator Placement

The reconfigurable fabric consists of N equally sized rectangular regions. During runtime, the application requests to configure $M \leq N$ accelerators to speed up its computational kernels. The runtime system has to decide to which regions the M accelerators shall be configured, by first deciding which $N - M$ regions shall

not be reconfigured, e.g., by using a least recently used replacement policy. The decision to which of the remaining regions an accelerator is placed does not affect the application performance, but it affects the stress applied to the regions.

Each region contains $X \times Y$ CLBs with an (x, y) coordinate. The stress experienced so far by the CLBs in region k is denoted as $[S_k]_{xy}$ and the stress that will be induced by an accelerator j is denoted as $[s_j]_{xy}$ (see Eq. (8)). It depends on how often the accelerator will be executed, as determined by offline profiling (see Sect. 4.1). If an accelerator j is placed into region k , then the accelerator executions increase the stress state of the region to $S'_k = S_k + s_j$. The challenge is to place each accelerator to a region, such that upon completion of the application kernel the maximum CLB stress over the N regions is minimized, i.e., $\max_{k,x,y} [S'_k]_{xy}$ is minimized. It can be easily seen that the strict lower bound of the maximum CLB stress is given by Eq. (9), which is reached if and only if the stress is uniformly distributed over all CLBs. To achieve this at runtime, we propose a heuristic that follows these two rules: (1) maximal utilization of under-stressed CLBs within one region, i.e., the stress shall be evenly distributed among different CLBs within the region (*intra-region distribution*) and (2) avoid placing high stress accelerators into highly stressed regions, i.e. the stress shall be evenly distributed among different regions (*inter-region distribution*). The heuristic uses a profit function (Eq. (10)) for placing accelerator j into region k that considers the stress distribution within one region and across all regions, respectively.

$$\frac{1}{NXY} \left(\sum_k^N \sum_{x,y} [S_k]_{xy} + \sum_j^M \sum_{x,y} [s_j]_{xy} \right) \quad (9)$$

$$\text{Profit}_{jk} = \text{Profit}_{jk}^{intra} + \text{Profit}_{jk}^{inter} \quad (10)$$

To calculate $\text{Profit}_{jk}^{intra}$, the average CLB stress in region k is determined as AvgStress_k and then used to calculate the absolute deviation of the stress of CLB_{xy} in region k from AvgStress_k . The sum over all CLBs in region k denotes the intra-region stress imbalance. It is calculated (1) before placing accelerator j to region k and (2) after hypothetically placing it. The difference of these two values corresponds to the degree of increased stress imbalance if placing accelerator j to region k and is used as $\text{Profit}_{jk}^{intra}$. The idea for $\text{Profit}_{jk}^{inter}$ is very similar. There, the stress of region k is compared with the average stress of all regions before and after hypothetically placing accelerator j to region k [22].

The stress-aware runtime accelerator placement iterates over all required accelerators. In each iteration, it calculates the profits of placing the accelerator into all available regions and then places the accelerator into the region that provides the highest profit. The complexity of this algorithm is $\mathcal{O}(M^2XY)$. If the application does not reconfigure a region for a longer time, then this region would be constantly stressed by one accelerator without stress redistribution. As a solution, the runtime accelerator placement forces that region to be reconfigured after a user-defined time period that should not be too short to prevent increased reconfiguration overhead

and also not too long to avoid stress accumulation. For instance, a time period of 100 million cycles (1 s at 100 MHz) is short enough to avoid aging accumulation and the induced application performance degradation is only 0.21%.

4.4 Synthesis Time Logic Placement

Our runtime accelerator placement uniformly distributes the stress over all reconfigurable regions, compared to the stress-unaware placement. The maximal transistor toggle rate is reduced by more than 73% from 18.8 million toggles/s (see Fig. 8a) down to 5.0. However, when high stress CLBs of different accelerators *overlap* at the same relative (x, y) location, the runtime accelerator placement cannot achieve intra-region stress distribution. STRAP addresses this problem by applying placement constraints at *synthesis time* to diversify (similar to Sect. 3.1) the CLB usage among different accelerators, which reduces the overlapping of high stress CLBs. To minimize the timing impact on accelerators, STRAP only constrains which CLBs shall be used and leaves everything else to the vendor place-and-route algorithm.

The logic placement algorithm (Algorithm 2) diversifies the high stress CLBs of different accelerators to different CLB locations in the regions. First, unconstrained configurations of all accelerators are generated (Lines 1–5). For each accelerator

Algorithm 2 Stress-diversifying logic placement

Input: List of accelerators Acc.

```

1. for  $j := 1$  to  $\text{len}(\text{Acc})$  do
2.   Place-and-route  $\text{Acc}[j]$  without any placement constraints
3.    $s_j := \text{get\_stress}(\text{Acc}[j])$ 
4.    $\text{Acc}[j].\text{max\_freq} := \text{get\_max\_freq}(\text{Acc}[j])$ 
5. end for
6.  $\text{Acc} := \text{sort\_ascending}(\text{Acc}, \text{key}=\text{max\_freq})$ 
7.  $\mathbf{R} := \mathbf{s}_1$ 
8. for  $j := 2$  to  $\text{len}(\text{Acc})$  do
9.    $\text{prohibit\_xy} := \emptyset$ 
10.  for  $x := 1$  to  $\text{Acc}[j].\text{n\_cols}$  do
11.    for  $y := 1$  to  $\text{Acc}[j].\text{n\_rows}$  do
12.      if Condition Eq.(11) is satisfied for  $(x, y)$  then
13.         $\text{prohibit\_xy.add}((x,y))$ 
14.      end if
15.    end for
16.  end for
17.  Place-and-route  $\text{Acc}[j]$  with prohibited CLB locations listed in  $\text{prohibit\_xy}$ 
18.  if Place-and-route failed then
19.     $\text{prohibit\_xy.remove}\left(\arg\min_{xy \in \text{prohibit\_xy}} \left\{ \left[ \hat{\mathbf{R}} + \hat{\mathbf{s}}_j \right]_{xy} \right\} \right)$ 
20.    goto Line 17
21.  end if
22.   $\mathbf{R} := \mathbf{R} + \text{get\_stress}(\text{Acc}[j])$ 
23. end for
```

configuration the CLB stress is estimated (see Sect. 4.2), and the maximal achievable frequency is extracted from the place-and-route log files (Lines 3–4). The generated initial configurations are then sorted in ascending order of their maximal achievable frequencies (Line 6). The fabric typically runs at the frequency of the slowest accelerator f_{min} . In order to minimize the impact on system performance, it is placed and routed without stress-diversifying placement constraints. Its CLB stress distribution is taken as the initial reference distribution (Line 7). As long as the proposed logic placement does not reduce the frequency of an accelerator below f_{min} , there is no performance impact/penalty for the whole system. During the generation of other accelerator configurations, \mathbf{R} keeps track of the sum of the stress distribution of all $j - 1$ previously generated accelerators, i.e., $\mathbf{R} = \sum_{i=1}^{j-1} \mathbf{s}_i$.

The remaining accelerators will be placed-and-routed again in ascending order of their maximal frequencies (Lines 8–23). To avoid that high stress CLBs of the currently placed accelerator $\text{Acc}[j]$ overlap with those in previously placed accelerators $\text{Acc}[1], \dots, \text{Acc}[j-1]$, we prohibit the placement to specific CLB locations for $\text{Acc}[j]$ (Lines 9–17) if Eq. (11) is satisfied, where L_j is the number of used CLBs by the currently place-and-routed accelerator $\text{Acc}[j]$. $\hat{\mathbf{R}}$ and $\hat{\mathbf{s}}_j$ are normalized stress matrices of \mathbf{R} and \mathbf{s}_j . In earlier iterations, the reference distribution is less even, which implies that few CLB locations in the reference distribution have much higher values than the others, and therefore it is less likely that the condition in Eq. (11) is satisfied. In turn, fewer locations are prohibited for placement in earlier iterations, which implies less timing impact on slower accelerators. If place-and-route fails due to too many prohibited CLB locations, the locations xy where the stress overlapping $[\hat{\mathbf{R}} + \hat{\mathbf{s}}_j]_{xy}$ is lowest are removed from `prohibit_xy` (Line 19), and place-and-route is re-executed with the relaxed constraints.

$$\begin{aligned} [\hat{\mathbf{R}}]_{xy} &> \frac{1}{L_j} \sum_{uv} [\hat{\mathbf{s}}_j]_{uv} \\ \text{with } \hat{\mathbf{R}} &= \frac{\mathbf{R}}{\max_{uv} [\mathbf{R}]_{uv}} \text{ and } \hat{\mathbf{s}}_j = \frac{\mathbf{s}_j}{\max_{uv} [\mathbf{s}_j]_{uv}} \end{aligned} \quad (11)$$

With synthesis time stress diversification, high stress CLBs from different accelerators are placed to different CLB locations, and thus better intra-region stress distribution can be achieved during runtime placement. After applying both stress-aware runtime placement and synthesis time stress diversification for dynamic stress, the maximal transistor toggle rate is further reduced by additional 44% from 5.0 million toggles/s down to 2.8 (see Fig. 8b).

4.5 Extended Accelerator Placement with Module Diversification

The module diversification method (see Sect. 3) generates a set of configurations for each accelerator that are diversified in terms of CLB usage. This not only allows to tolerate any single-CLB fault in a region but can also improve the stress distribution with the extra CLB diversity. When faults are detected in the reconfigurable fabric, the placement freedom of accelerators is reduced. The *placement freedom* of an accelerator corresponds to the number of regions for which the accelerator has at least one diversified configuration that can be placed into that region (i.e., that tolerates the permanent faults in that region). Such a region is called a *compatible region*. If the available regions (i.e., those into which no accelerators are placed by the placement algorithm so far) have rather many permanent faults, it can happen that no configuration of the accelerator can be placed into any of them. If an accelerator cannot be placed, then its hardware functionality has to be emulated in software on the processor pipeline, which comes at a significant performance loss.

To avoid such situations, the runtime accelerator placement (see Sect. 4.3) is modified to place the accelerators one after the other in ascending order of their number of compatible regions. If it comes to the situation that some accelerator cannot be placed into the available regions, then the algorithm re-evaluates some of its previous placement decisions (note that the actual reconfigurations are just started after all placements are finally decided). It tries whether it can *swap* one of the already placed accelerators into one of the still available regions such that accelerator can be placed into the region that became free due to swapping. When calculating the placement profit (see Eq. (10)), the algorithm also iterates through all diversified configurations to find out which configuration of the accelerator produces the highest placement profits.

4.6 Experimental Evaluation

For prototyping purposes, we have integrated STRAP into the Xilinx tool-chain and the runtime system of the target reconfigurable architecture. In our evaluation platform, each region consists of 4×20 CLBs with eight 6-input LUTs per CLB. STRAP performs optimizations on CLB granularity. To evaluate the actual stress for each transistor, a transistor-level model of LUTs using NMOS pass transistors for multiplexers is used [22]. To evaluate the threshold voltage shift due to stress, state-of-the-art aging models are employed (detailed equations and used parameters are given in [22]). The resource usage of each accelerator within one region for the H.264 application ranges from 8.8% to 66.3%. Our architectural simulator is used to evaluate the STRAP method for systems that differ in the number of reconfigurable regions and runtime strategies, and to compare it with related work.

Evaluation Flow The placed-and-routed accelerators are fed to Xilinx XPower analyzer to obtain the signal activities and power consumption of logic elements and nets. The power consumption is then aggregated to CLB granularity by summing up the power consumed by LUTs and their fan-in nets in one CLB. The leakage power of a region is proportional to its size. Architectural simulation produces the accelerator execution trace, i.e., the complete execution and idle history of each accelerator in each region. Together with the power profile of each accelerator, we obtain the power trace of each CLB. The power trace and the fabric floorplan of the FPGA² are then fed into Hotspot³ [14] to obtain the temperature trace of each CLB, which will be used to evaluate the threshold voltage shift. The accelerator execution trace and the LUT signal activities of each accelerator are combined to calculate the LUT signal activities for the regions. This is then used to evaluate the stress of individual transistors by using the before-mentioned LUT transistor model.

The number of regions is varied from 5 to 12 and separate evaluation is performed for dynamic and static stress mitigation, since STRAP optimizes either for dynamic or for static stress. The baseline system does not use any stress distribution method. For comparison, two state-of-the-art stress distribution methods [3, 21] were implemented. Zhang et al. [21] use three different configurations for each accelerator and switch between them to migrate stress, whereas Angermeier et al. [3] consider the peak stress of regions to place an accelerator. As proposed for STRAP, Angermeier et al. [3] and Zhang et al. [21] were extended to replace an accelerator if its reconfigurable region has not been reconfigured for 100 million cycles (see Sect. 4.3). This improvement reduces the peak stress of [3, 21] and thus makes the comparison with state-of-the-art more competitive. Regarding temperature variation, a conservative comparison is performed. To calculate the threshold voltage shift for [3, 21], the lowest temperature that was observed for any CLB at any time in the obtained temperature trace is used as the constant temperature for all CLBs, while the highest observed temperature is applied for STRAP. Thus, the threshold voltage shift reported for [3, 21] is a lower limit, whereas the one for STRAP is a conservative upper limit.

Timing Overhead STRAP’s stress-diversifying logic placement at synthesis time may affect the accelerator frequency. The place-and-route tool is given a target frequency of 250 MHz as timing constraint to obtain the maximum operating frequency of each accelerator. On average, the maximum accelerator frequency decreases by 7%. Since accelerators with longer critical path (lower maximum frequency) are imposed with fewer constraints (see Sect. 4.4), their maximum frequencies are less affected. The maximum *system* frequency is however limited by the accelerator with the longest critical path (in our case the PointFilter accelerator, which runs at $f_{min} = 89$ MHz). Therefore, STRAP has no negative timing impact on the system.

²Based on a high-resolution die image acquired from <https://chipworks.com> (now <https://techinsights.com>).

³Smallest possible heat spreader and heat sink with 10 µm thickness, ambient temperature 50 °C.

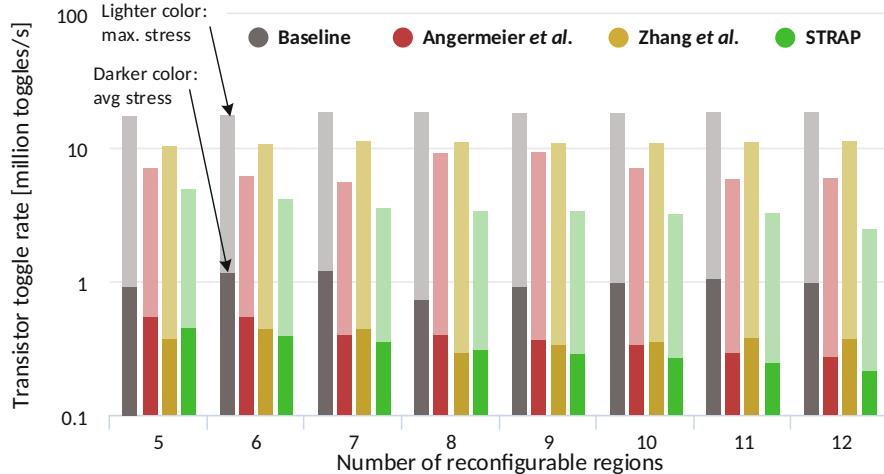


Fig. 10 The dynamic stress in systems with different number of reconfigurable regions when using our STRAP approach compared to the baseline, Angermeier et al. [3] and Zhang et al. [21] (based on [22])

Stress Reduction and MTTF Improvement Figure 10 shows the maximal (lighter color) and average (darker color; arithmetic mean) dynamic transistor stress, measured in million toggles/s, in the whole reconfigurable fabric for systems with different number of regions. It shows that all methods reduce the average stress compared to the baseline because they all distribute the stress to more transistors. While the reduction of the average stress is similar for all three methods, the reduction of the maximal stress (i.e., the critical part for system mean time to failure (MTTF)) differs significantly and requires both runtime and synthesis time optimization. The reason is that Angermeier et al. [3] perform only runtime inter-region stress distribution, while Zhang et al. [21] perform only synthesis time intra-region stress distribution for individual accelerators. In contrast, STRAP performs cross-layer stress-aware placement at runtime and synthesis time, which leads to the highest reduction of maximal stress in all evaluated cases. The reduction of the maximum stress by STRAP is up to 64% and 35% higher than the closest competitors w.r.t. dynamic and static stress, respectively. Table 3 summarizes the stress reduction.

Although during optimization only one type of stress is considered, actually both types of stress are reduced simultaneously. With STRAP targeting the static stress distribution, a reduction of 52% in dynamic and 38% in static stress is observed. When targeting dynamic stress, STRAP delivers 82% reduction in dynamic stress and 21% reduction in static stress. The reason behind the reduction of both stress types is that STRAP implicitly distributes the transistor usage as well, which reduces the individual static and dynamic transistor stress.

Table 3 Reduction of avg./max. stress and MTTF increase of STRAP and state-of-the-art [3, 21] compared to the baseline; averaged over all numbers of reconfigurable regions (based on [22])

Strategy	Reduction of avg. stress [%]		Reduction of max. stress [%]		MTTF improvement [%]	
	Dyn.	Stat.	Dyn.	Stat.	HCI	BTI
Angermeier et al. [3]	60.6	47.4	61.2	0.02	157.7	0.0
Zhang et al. [21]	62.6	49.6	39.9	4.5	66.4	2.3
STRAP	67.9	59.6	80.5	33.1	413.0	13.4

The MTTF improvement due to the stress reduction is calculated by assuming that a device fails when ΔV_{th} of any transistor exceeds 50% of its original value (V_{th0}). The MTTF improvement due to dynamic and static stress reduction is shown in the last two columns in Table 3. With the STRAP method, the MTTF improvement relative to the baseline is 413% and 13% in average for HCI and BTI aging, respectively. Relative to the closest competitors, STRAP achieves up to 177% and 14% MTTF improvement w.r.t. HCI and BTI aging, respectively.

5 Conclusion

The dependable operation of runtime-reconfigurable architectures is threatened by aging. This chapter presented novel methods to ensure reliable reconfiguration, mitigate aging, and tolerate emerging faults in the reconfigurable fabric. The *pre-configuration online tests* (PRET) and *post-configuration online tests* (PORT) check with minor application performance loss, if the reconfigurable fabric is faulty and if the reconfiguration process completed without errors during runtime. The *module diversification* design method generates the minimal number of diversified configurations required to tolerate at least any single CLB-fault in a reconfigurable region. The cross-layer stress-aware placement method STRAP mitigates aging by balancing stress both within a reconfigurable region as well as across all reconfigurable regions in the system. Relative to the closest competitors, STRAP achieves up to 177% and 14% MTTF improvement w.r.t. HCI and BTI aging. This shows that intelligently considering and managing aging threats during runtime can significantly improve the system dependability at limited overheads.

Acknowledgments This work is supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500—<http://spp1500.itec.kit.edu>).

References

1. Abdelfattah, M.S., Bauer, L., Braun, C., Imhof, M.E., Kochte, M.A., Zhang, H., Henkel, J., Wunderlich, H.-J.: Transparent structural online test for reconfigurable systems. In: IEEE International On-Line Testing Symposium (IOLTS), pp. 37–42 (2012)
2. Amrouch, H., van Santen, V.M., Ebi, T., Wenzel, V., Henkel, J.: Towards interdependencies of aging mechanisms. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 478–485 (2014)
3. Angermeier, J., Ziener, D., Glaß, M., Teich, J.: Stress-aware module placement on reconfigurable devices. In: International Conference on Field Programmable Logic and Applications (FPL), pp. 277–281 (2011)
4. Bauer, L., Shafique, M., Henkel, J.: Concepts, architectures, and run-time systems for efficient and adaptive reconfigurable processors. In: NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 80–87 (2011)
5. Bauer, L., Braun, C., Imhof, M.E., Kochte, M.A., Zhang, H., Wunderlich, H.-J., Henkel, J.: OTERA: Online test strategies for reliable reconfigurable architectures. In: NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 38–45 (2012)
6. Bauer, L., Braun, C., Imhof, M.E., Kochte, M.A., Schneider, E., Zhang, H., Henkel, J., Wunderlich, H.-J.: Test strategies for reliable runtime reconfigurable architectures. *IEEE Trans. Comput. (TC)* **62**(8), 1494–1507 (2013)
7. Bauer, L., Zhang, H., Kochte, M.A., Schneider, E., Wunderlich, H.-J., Henkel, J.: Advances in hardware reliability of reconfigurable many-core embedded systems. In: Many-Core Computing: Hardware and Software, pp. 395–416. Institution of Engineering and Technology (IET) (2019)
8. Cao, Y., Velamala, J., Sutaria, K., Chen, M.S.-W., Ahlbin, J., Esqueda, I.S., Bajura, M., Fritze, M.: Cross-layer modeling and simulation of circuit reliability. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. (TCAD)* **33**(1), 8–23 (2014)
9. Gaisler, A.: Homepage of the Leon Processor. Online available: <https://www.gaisler.com/index.php/products/processors/leon3>. Accessed 13 Mar 2019
10. Guo, X., Burleson, W., Stan, M.: Modeling and experimental demonstration of accelerated self-healing techniques. In: IEEE/ACM Design Automation Conference (DAC), pp. 1–6 (2014)
11. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. (TCAD)* **32**(1), 8–23 (2013)
12. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M., Teich, J., Wehn, N., Wunderlich, H.-J.: Design and architectures for dependable embedded systems. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 69–78 (2011)
13. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: IEEE/ACM Design Automation Conference (DAC), pp. 1–10 (2013)
14. Huang, W., Ghosh, S., Velusamy, S., Sankaranarayanan, K., Skadron, K., Stan, M.R.: HotSpot: a compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **14**(5), 501–513 (2006)
15. Lala, P.K.: Self-checking and Fault-Tolerant Digital Design. Morgan Kaufmann, San Francisco (2001)
16. Mahapatra, S.: Fundamentals of Bias Temperature Instability in MOS Transistors: Characterization Methods, Process and Materials Impact, DC and AC Modeling. Springer Series in Advanced Microelectronics, vol. 52. Springer, New Delhi (2015)

17. Srinivasan, S., Krishnan, R., Mangalagiri, P., Xie, Y., Narayanan, V., Irwin, M.J., Sarpatwari, K.: Toward increasing FPGA lifetime. *IEEE Trans. Depend. Sec. Comput. (TDSC)* **5**(2), 115–127 (2008)
18. Stott, E.A., Wong, J.S., Sedcole, P., Cheung, P.Y.: Degradation in FPGAs: measurement and modelling. In: ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), pp. 229–238 (2010)
19. Xilinx: Partial Reconfiguration User Guide, UG702 (v14.1) (2012)
20. Yang, S.: Logic synthesis and optimization benchmarks user guide: version 3.0. MCNC Technical Report, Microelectronics Center of North Carolina (MCNC). <https://ddd.fit.cvut.cz/prj/Benchmarks/>
21. Zhang, H., Bauer, L., Kochte, M.A., Schneider, E., Braun, C., Imhof, M.E., Wunderlich, H.-J., Henkel, J.: Module diversification: fault tolerance and aging mitigation for runtime reconfigurable architectures. In: IEEE International Test Conference (ITC), pp. 1–10 (2013)
22. Zhang, H., Kochte, M.A., Schneider, E., Bauer, L., Wunderlich, H.-J., Henkel, J.: STRAP: stress-aware placement for aging mitigation in runtime reconfigurable architectures. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 38–45 (2015)
23. Zhang, H., Bauer, L., Kochte, M.A., Schneider, E., Wunderlich, H.J., Henkel, J.: Aging resilience and fault tolerance in runtime reconfigurable architectures. *IEEE Trans. Comput. (TC)* **66**(6), 957–970 (2017)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Reliability Analysis and Mitigation of Near-Threshold Voltage (NTC) Caches



Anteneh Gebregiorgis, Rajendra Bishnoi, and Mehdi B. Tahoori

1 Introduction

SRAM based memory elements have been the prominent limiting factor in the near-threshold voltage domain as the supply voltage of SRAM cells does not easily downscale, as it is done for combinational logic. The supply voltage downscaling limitation is due to the significant increase in the failure rate of SRAM cells operating at lower supply voltage values, which in turn severely affects the yield. Various state-of-the-art solutions have been proposed to address this issue. These solutions include variation tolerant SRAM cell design [3, 13, 29] and heterogeneous cache design [31], improve the robustness of cache memories. However, the improvement comes at the cost of increased area and power overheads. Moreover, these approaches mostly ignore the impact of runtime failure mechanisms, such as aging and soft error, on the reliability of memory components. Therefore, design-time reliability failure analysis and mitigation schemes are crucial for the reliable operation of near-threshold caches.

Analyzing failures based on a particular reliability failure mechanism is insufficient for estimating the system-level reliability, as the interdependence among different failure mechanisms has a considerable impact on the overall system reliability. Moreover, the running workload affects the aging and SER of memory components as it determines the SP and AVF of the memory elements. Therefore, performing a combined analysis on the reliability failure mechanisms across different layers of abstraction (as shown in Fig. 1) is crucial, and it helps designers to choose the most reliable components at each abstraction layer, and tackle the reliability challenges of NTC operation.

A. Gebregiorgis · R. Bishnoi · M. B. Tahoori (✉)
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
e-mail: anteneh.gebregiorgis@kit.edu; rajendra.bishnoi@kit.edu; mehdi.tahoori@kit.edu

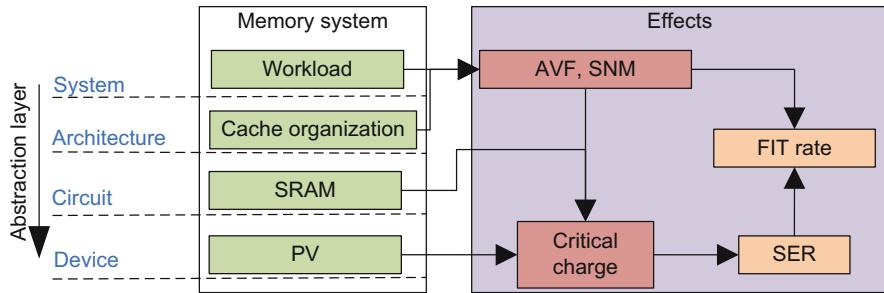


Fig. 1 Cross-layer impact of memory system and workload application on system-level reliability (Failure In-Time (FIT rate)) of NTC memory components, and their interdependence

For this purpose, a comprehensive cross-layer reliability analysis framework addressing the combined effect of aging, process variation, and soft error on the reliability of NTC cache designs is presented in this chapter. Moreover, the chapter presents the advantages and limitations of two different NTC SRAM cell designs (namely, 6T and 8T cells) in terms of reliability (SER and SNM) improvement, area, and energy overheads. The framework presented in this chapter helps to explore the cross-layer impact of different reliability failure mechanisms, and it is useful to study the combined effect of workload and cache organization on the SER and SNM of cache memories. The framework is also helpful to understand how the reliability issues change from super-threshold to the near-threshold voltage domain. Furthermore, it is important for architectural-level design space exploration to find the best cache organization for better reliability and performance trade-offs of NTC caches. Based on the comprehensive analysis using the framework, a memory failure mitigation scheme is developed to improve the energy efficiency of NTC caches.

2 Functional Failure and Reliability Issues of NTC Memory Components

The increase in sensitivity to process variation of NTC circuits affects not only the performance but also functionality. Notably, the mismatch in device strength due to process variation affects the state of positive feedback loop based storage elements (SRAM cells) [3, 10, 14]. The mismatch in the transistors makes SRAM cells to incline for one state over the other, a characteristic that leads to hard functional failure or soft timing failure [17, 20]. The variation-induced functional failure rate of SRAM cells is more pronounced in the nanoscale era as highly miniaturized devices are used to satisfy the density requirements [1]. SRAM cells mainly suffer from three main unreliability sources: (1) aging effects, (2) radiation-induced soft error, and (3) variation-induced functional failures [19]. The SRAM cell susceptibility to these issues increases with supply voltage downscaling.

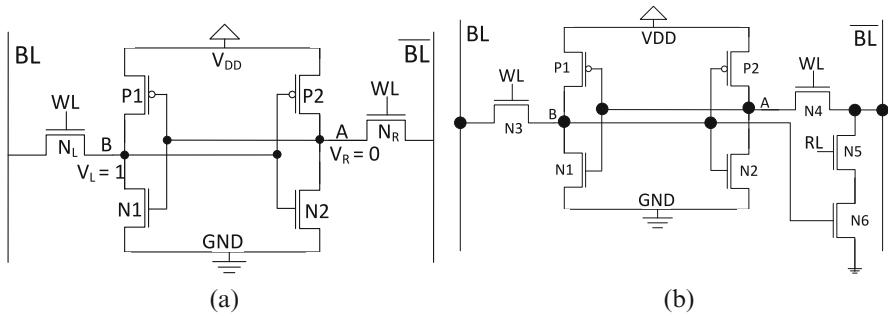


Fig. 2 Schematic diagram of 6T and 8T SRAM cell, where WL = word-line, BL = bit-line and RL = read-line. (a) 6T cell design. (b) 8T cell design

2.1 Aging Effects in SRAM Cells

Accelerated transistor aging is one of the main reliability concerns in CMOS devices. Among various mechanisms, Bias Temperature Instability (BTI) is the primary aging mechanism in nanoscale devices [18]. BTI gradually increases the threshold voltage of a transistor over a long period, which in turn increases the gate delay [18]. BTI-induced threshold voltage shift is a strong function of temperature as it has an exponential dependency. Hence, BTI-induced aging rate is higher at high operating voltage and temperature values. In SRAM cells, BTI reduces the Static Noise Margin (SNM)¹ of an SRAM cell, and makes it more susceptible to failures. BTI-induced SNM degradation is higher when the cell stores the same value for a longer period (e.g., storing “0” at node “A” of the SRAM cell shown in Fig. 2a). Hence, the effect of BTI on an SRAM cell is a strong function of the cell’s Signal Probability (SP).²

2.2 Process Variation in SRAM Cells

Variation in transistor parameters such as channel length, channel width, and threshold voltage results in a mismatch in the strength of the transistors in an SRAM cell, and in extreme cases it makes the cell to fail [15]. The variation-induced memory failure rate increases significantly with supply voltage downscaling, for instance, SRAM cells operating at NTC (0.5 V) have 5× higher failure rate than the cells operating at a nominal voltage [15]. Process variation affects several aspects of SRAM cells, and the main variation-induced SRAM cell failures are:

¹SNM is the minimum amount of DC noise that leads to a loss of the stored value.

²Probability of storing logic “1” in the SRAM cell.

Read Failure Read failure/disturb is a phenomenon where the stored value is distorted during read operation. For example, when reading the value of the cell shown in Fig. 2a, ($V_L = "1"$ and $V_R = "0"$), due to the voltage difference between the access transistor N_R and pull-down transistor N_2 , the voltage at node V_R increases [21, 39]. If this voltage is higher than the trip voltage (V_{trip}) of the left inverter, then the stored value of the cell is changed. Hence, the condition for read failure is expressed as [33]:

$$\text{read failure} = \begin{cases} 1, & \text{if } V_R > V_{trip} \\ 0, & \text{otherwise} \end{cases}$$

where $V_{trip} = V_{P_1} - V_{N_1}$ (here V_{P_1} and V_{N_1} indicate the voltages of the PMOS and NMOS transistors of the left inverter shown in Fig. 2a where P_1 and N_1 are the corresponding PMOS and NMOS transistors of the inverter).

Write Failure Write failure occurs when the cell is not able to write/change its state with the applied write voltage. For example, during a write operation (e.g., writing “0” to the SRAM cell shown in Fig. 2a), the node V_L is discharged through the bit-line BL. Write failure occurs when the node V_L is not reduced to be lower than V_{trip} of the right inverter (V_R) [21, 33]. In the standard 6T SRAM cell, write failure is a challenging issue as the cell cannot be optimized without reducing its read margin [21, 33, 39]. However, this is improved with the help of read/write assist circuitries or differential read/write access as it is done in the 7T, 8T, and 10T SRAM cell designs [3, 8, 10]. In order to illustrate the write failure issue, the write margin behaviors of 6T and 8T NTC SRAM cells are studied and compared in Fig. 3. As shown in the figure, the 6T SRAM cell has a smaller write margin as it

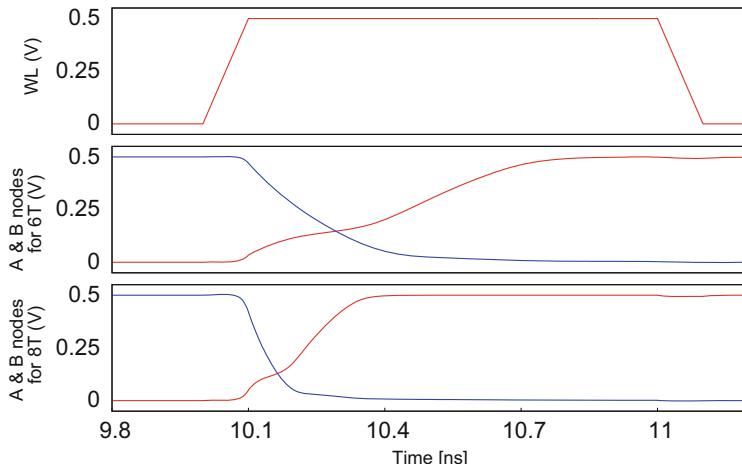


Fig. 3 Write margin (in terms of write latency) comparison of 6T and 8T SRAM cell operating in near-threshold voltage domain (0.5 V)

has longer write latency. On the other hand, the short write latency of the 8T design enables it to have a relatively larger write margin. The improvement in the write margin is because the 8T cell is optimized to improve the write operation without affecting its read operation, as the write and read operations are decoupled.

Hold Failure Hold failure commonly known as metastability issue is a reliability issue that occurs when the SRAM cell is not able to store the value for a longer period [20, 33]. This problem happens during a standby mode if the voltage at nodes V_L or V_R is smaller (smaller SNM value), then the stored value is easily destroyed by a noise voltage due to various sources such as particle strike and leakage current [20, 33].

2.3 Soft Error Rate in SRAM Cells

In SRAM cells, soft error is a transient phenomenon that occurs when charged particles penetrate the cell's cross junction creating an aberrant charge that changes the state of the cell [27]. The primary source of soft errors is related to cosmic ray events such as neutrons and alpha particles. Atmospheric neutrons are one of the higher flux components, and their reaction has a high energy transfer. Thus, neutrons are the most likely cosmic radiations to cause soft errors [16, 19]. Neutrons do not generate electron-hole pairs directly. However, their interaction with the Si-atoms generates secondary particles. These secondary particles produce charges/electron-hole pairs [16]. If the generated charges are larger than the *critical charge*³ of an SRAM cell, then the internal value of the cell is inverted, this phenomenon is commonly referred to as soft error.

Radiation-induced Soft Error Rate (SER) of an SRAM cell increases significantly with decrease in the supply voltage. Previous experiments have shown that the radiation-induced SER increases by 50% for just 20% decrease in the supply voltage [40]. Moreover, the SER of NTC designs is affected by variation and aging-induced SNM degradation.

2.4 Interdependence and Combined Effects

Analyzing failures based on a particular reliability failure mechanism is insufficient for estimating the system-level reliability as the interdependence among different failure mechanisms (such as aging, soft error, and process variation) has a considerable impact on the overall system reliability [4, 19, 20]. Figure 4 shows how the interdependence between different reliability mechanisms (aging, SER, and process

³Minimum amount of charge required to upset the stored value, of an SRAM cell.

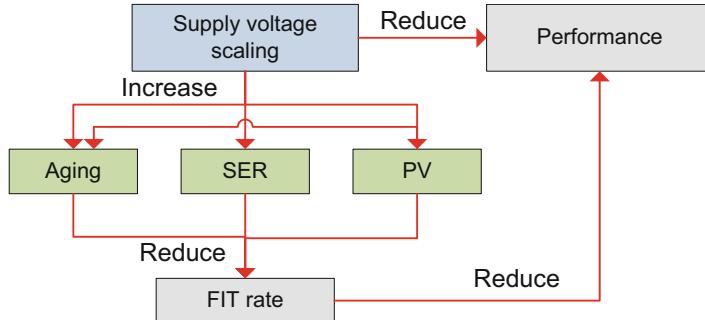


Fig. 4 Interdependence of reliability failure mechanisms and their impact on the system Failure In-Time (FIT) rate in NTC

variation) affects the overall system reliability of memory components in terms of Failure In-Time (FIT rate). As shown in the figure, variation-induced threshold voltage shift increases both aging and SER by reducing the SNM and critical charge of the cell. Similarly, aging-induced SNM degradation increases the sensitivity of SRAM cell to soft errors. The problem is more pronounced when the SRAM cell is operating at NTC domain due to the wide variation extent and higher sensitivity to aging effects [19]. It has been observed that aging has $\approx 5\%$ SNM and critical charge degradation at NTC while process variation-induced SNM degradation reaches as high as 60% [19]. In the super-threshold voltage domain (1.0 V), however, the aging effect increases by $3\times$ to be 15% while variation effect is reduced significantly.

Moreover, the running workload affects the aging rate and SER of memory components, as it determines the signal probability and the Architectural Vulnerability Factor (AVF)⁴ of the memory elements [19]. Therefore, to overcome these reliability challenges and improve the overall system reliability, combined analysis of the reliability failure mechanisms at different levels of abstraction is imperative. Besides, the cross-layer analysis should consider the impact of workload on signal probability as well as architectural vulnerability factor of memory components, and their circuit-level consequences on critical charge and SNM degradation.

2.5 Technology Scaling Effects on SRAM Reliability

Reliability has been an essential issue with the miniaturization of CMOS technology, as different design-time and runtime failures are among the limiting factors of technology scaling [24]. At smaller technology nodes, process variation increases the permanent and transient failures of memory components significantly [11, 15].

⁴AVF is the probability that an error in memory structure propagates to the data path. AVF = vulnerable period/total program execution period.

The authors in [15] show that SRAM cell failure rate increases by more than $2\times$ with downscaling from 90 to 65 nm technology node. Similarly, the authors in [26] demonstrated that technology downscaling increases the radiation-induced soft error rate of SRAM cells significantly.

3 Cross-Layer Reliability Analysis Framework for NTC Caches

The comprehensive cross-layer reliability estimation framework that abstracts the impact of workload, cache organization, and reliability failure mechanisms at different levels of abstraction is illustrated in Fig. 5. The reliability analysis and simulation conducted in this work use the symmetric six-transistor (6T) and 8T SRAM cells shown in Fig. 2a and b. In this work, the device-level critical charge characterization is modeled according to the analytical model presented in [27].

This section presents the cross-layer reliability estimation framework in a top-down manner. The system-level *Failure In-Time* (FIT) rate and SNM extraction are described in Sect. 3.1 followed by the cross-layer SNM and SER estimation in Sect. 3.2.

3.1 System FIT Rate Extraction

The system-level FIT rate of a cache memory is the sum of the FIT rate of each row (cache line). The row FIT rate is calculated as the product of the row-wise SER (extracted based on the circuit-level SER information) and its

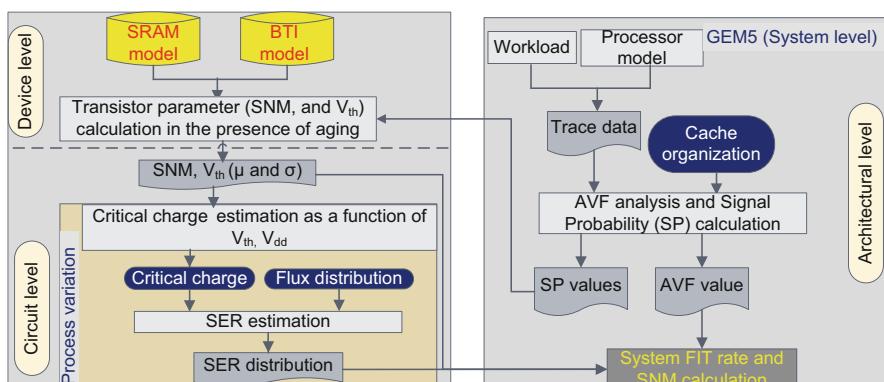


Fig. 5 Holistic cross-layer reliability estimation framework to analyze the impact of aging and process variation effects on soft error rate

Architectural Vulnerability Factor (AVF). Cache AVF is a metrics used to determine the probability that an error in a cache memory propagates to the datapath, and results in a visible error in a program’s final output [38]. Equation (1) shows the system-level FIT rate calculation of cache memories.

$$\text{FIT}_{\text{system}} = \sum_{i=0}^{N-1} \text{AVF}_i \times \text{SER}_i \quad (1)$$

where N is the total number of rows in the cache.

3.1.1 Architecture-Level AVF Analysis

One step of determining the failure rate of memory (cache) due to soft errors is to determine the AVF value of the memory. AVF of a memory array is measured by the ratio of vulnerable periods, time interval in which the memory content is exposed to particle strike, to the total program execution period, and the probability of the erroneous value being propagated [38]. Hence, the vulnerability factor of a memory array is computed based on the liveness analysis commonly known as Architectural Correct Execution (ACE) analysis which is the ratio of ACE (vulnerable) cycles to the total number of operational cycles [42]. Therefore, the AVF value of a memory array with M cells is computed as shown in Eq. (2).

$$\text{AVF}_{\text{array}} = \frac{\sum_{i=0}^{M-1} \text{ACE}_i}{T \times M} \quad (2)$$

where T is the total number of cycles.

3.1.2 Architecture-Level SNM Analysis

Aging-induced SNM degradation of an SRAM cell strongly depends on the Signal Probability (SP) of the cell. Thus, BTI-induced SNM degradation is minimized when the signal probability of the cell is balanced (close to 0.5) [18]. In order to determine the aging-induced SNM degradation, the worst-case SP of the memory row is obtained as the maximum SP distance from 0.5 ($D_i = |\text{SP}_i - 0.5|$) as shown in Eq. (3). Then, the worst-case SP is used by the SNM estimation tool given in Fig. 5 to determine the corresponding aging-induced SNM degradation.

$$\text{SP}_{\text{worst-case}} = \text{MAX}_{i=1}^Z D_i \quad (3)$$

where $D_i = |\text{SP}_i - 0.5|$ and Z is the total number of cells in the memory row.

In order to extract the AVF and SNM of a cache unit, first, it is necessary to extract the trace of the data stored in the cache, read-write accesses, and the duration (number of cycles) of the running workload. Once the information is available, the

reliability analysis tool uses it along with the cache organization to determine the AVF and SP of the cache memory according to Eqs. (2) and (3), and generates the SNM LUT for different signal probability values.

The cache organization (size and associativity) has significant impact on the SER and SNM of the cache, as it determines the hit ratio and the duration data is stored in a cache entry. Hence, different cache size and associativity combinations result in different SER and SNM values for the same workload application. Additionally, SER and SNM are highly dependent on the running workload. In order to explore the impact of cache organization and workload, various organizations and workload applications are investigated.

3.2 Cross-Layer SNM and SER Estimation

3.2.1 SNM Degradation Estimation

Device-Level Aging Analysis

BTI-induced aging degrades the carrier mobility of CMOS transistors, and leads to transistor threshold voltage (V_{th}) shift. In an SRAM cell, the V_{th} shift reduces the noise tolerance margin of the cell, and makes it more susceptible to failures. In the reliability analysis framework, the BTI-induced threshold voltage shift of the transistors in an SRAM cell is evaluated at device-level using a Reaction-Diffusion (RD) model [28]. Then, the device-level V_{th} shift results are used to estimate the corresponding SNM degradation of an SRAM cell at the circuit-level.

Circuit-Level SNM Estimation

The SNM of an SRAM cell is extracted by conducting a circuit-level SPICE simulation. The SPICE simulation uses device-level aging and architecture-level SP results to determine the SNM of the SRAM cell. Finally, the SNM degradation of a particular SP value is obtained according to Eq. (4).

$$\text{DEG}_{\text{SP}} = \frac{\text{SNM}_{\text{SP}} - \text{SNM}_{\text{fresh}}}{\text{SNM}_{\text{fresh}}} \times 100\% \quad (4)$$

where SNM_{SP} is the SNM of the SRAM cell for a particular signal probability value and $\text{SNM}_{\text{fresh}}$ is the SNM of a fresh (new) SRAM cell.

Aging and Process Variation-Induced SNM Degradation Analysis

BTI-induced SNM degradation of an SRAM cell depends not only on the cell signal probability but also on process parameters, such as channel length and

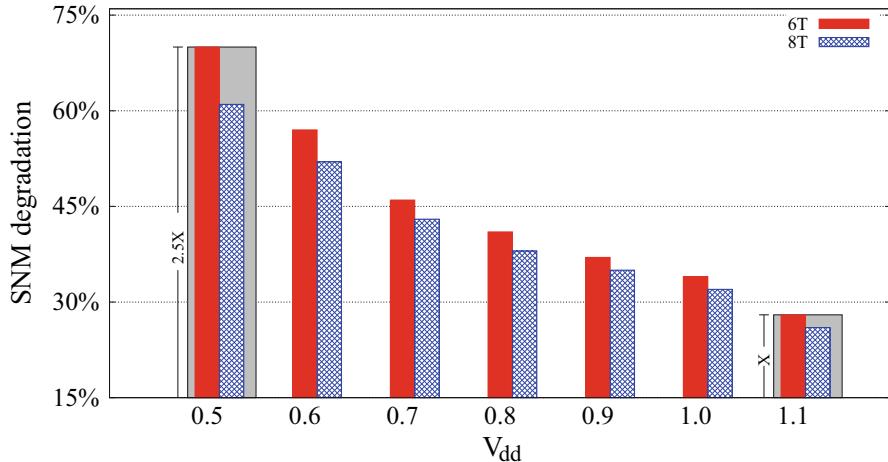


Fig. 6 SNM degradation in the presence of process variation and aging after 3 years of operation, aging+PV-induced SNM degradation at NTC is $2.5\times$ higher than the super-threshold domain

oxide thickness, which are highly affected by manufacturing variabilities. Due to low operating temperature at NTC, aging has relatively less impact on the SNM degradation of near-threshold voltage SRAM cells. However, in combination with variation-induced threshold voltage shift, aging degrades the SNM of SRAM cells significantly.

Figure 6 shows the worst-case aging ($SP = 0.0$) and variation-induced SNM degradation of 6T and 8T SRAM cells after 3 years of operation for wide supply voltage range. The obtained SNM degradation confirms the analytical expectation as the SNM degradation in NTC is $2.5\times$ higher than the degradation in the super-threshold voltage domain (as shown by the gray boxes). While the use of 8T instead of 6T SRAM cells in super-threshold voltage domain has limited improvement in SNM degradation (only 7.7%), it achieves more than 14% reduction in the SNM degradation in the near-threshold voltage domain.

3.2.2 SER Estimation

The SER of an SRAM cell depends on two main factors, the critical charge of the cell and the flux rate of the strike. To determine SRAM cell SER, first, the critical charge of an SRAM cell is obtained from a circuit-level model. Then, the SER value is calculated by combining the critical charge, flux distribution, and the area sensitive to strike.

Device-Level Critical Charge Characterization

The sensitivity of an SRAM cell to radiation-induced soft errors is determined by the critical charge (Q_{critical}) of the cell, as it determines the minimum amount of charge required to alter the state of the cell. The Q_{critical} of an SRAM cell depends on several factors such as supply voltage, threshold voltage, and strength of the transistors of the SRAM cell [9]. The critical charge of an SRAM cell is computed using analytical models or circuit simulators. An analytical model developed in [27] is used to determine the Q_{critical} .

As shown in Fig. 5, the SPICE model of an SRAM cell along with the BTI model is employed to evaluate the impact of BTI on the threshold voltage (V_{th}) of the transistors of an SRAM cell. The BTI analysis uses the SP values of the memory array from higher (architecture-level) analysis to determine the BTI-induced V_{th} shift of the running workload. In this way, the aging effect of the workload is incorporated into the framework. Once the fresh and aged V_{th} values are available, the impact of process variation is incorporated as a normal distribution ($\mu \pm 3\sigma$) of the transistor threshold voltage where μ is the mean V_{th} value and the standard deviation (σ) which is obtained using an industrial standard, measurement based, model (the “Pelgrom model”) given in Eq. (5) [30]. Finally, all these parameters are used by the model given in [27] to extract the Q_{critical} .

$$\sigma \Delta V_{th} = \frac{A_{VT}}{\sqrt{L \times W}} \quad (5)$$

where L and W are the length and width of transistors, and A_{VT} is process specific parameter (the “Pelgrom coefficient”).

Circuit-Level SER Analysis

The circuit-level SER analysis is conducted using the SER extraction module of the framework given in Fig. 5. First, the critical charge of the SRAM cell is extracted using the device-level model [27]. Afterward, the critical charge along with the neutron-induced flux distribution is used to determine the SER of the cell using an experimentally verified empirical model given in Eq. (6) [23]. As shown in Eq. (6), the SER of an SRAM cell has an inverse exponential relation with its critical charge (Q_{critical}). Hence, the higher the Q_{critical} , the lower the SER will be.

$$\text{SER} \propto F A e^{\left(-\frac{Q_{\text{critical}}}{Q_s}\right)} \quad (6)$$

where F is the flux in particles/cm²-s with energy higher than 1 MeV [6]; A is the area sensitive to a strike in cm², and Q_s is the charge collection efficiency.

The main observations from Eq. (6) are:

- The SER of an SRAM cell has an inverse exponential relation to its critical charge. Hence, a small decrease in the Q_{critical} leads to an exponential increase in the cell SER.
- For the same atmospheric neutrons, a small drift in Q_{critical} leads to a significant increase in the SER. Furthermore, transistor up-sizing increases the area which is sensitive to particle strike and hence, higher SER.

SER of 6T and 8T SRAM Cells

In the conventional 6T SRAM cell, the cell must maintain the stored value and it should be stable during read/write accesses. SRAM cell stability is a challenging task when the cell is operating in the near-threshold voltage domain, as the cell mainly suffers from read-disturb. To address this issue, either a read-write assist circuitry should be employed or the pull-down (NMOS) transistors of the SRAM cell should be strengthened by transistor up-sizing [35]. However, the up-sizing also increases the area of the cell that is sensitive to soft errors. Since the read-disturb of the 6T SRAM cell is worst when it operates at lower voltage values, transistor up-sizing cannot adequately mitigate the read-disturb issue which makes the 6T design less desirable for near-threshold voltage operation.

This issue is addressed by using alternative SRAM cell designs (such as 8T [32] and 10T [8] SRAM cells). For example, the read failure issue is solved in the 8T design by decoupling the read and write lines using two additional NMOS access transistors. The decoupling allows to downsize the pull-down NMOS transistors, and reduce the area sensitive to soft errors. Therefore, alternative SRAM designs (e.g., 8T) are recommended for NTC operation, which is verified by studying the reliability and energy efficiency improvement of the 8T SRAM design over the conventional 6T design. The transistor sizing specified in [32] is used for the design of the 6T and 8T SRAM cells used in this study.

Figure 7 shows the fresh and aged SER of the 6T and 8T SRAM designs for different supply voltage values. In the super-threshold voltage domain, (0.9–1.1 V) the 6T and 8T designs have negligible differences in their SER. In NTC, however, the 6T design has higher SER than the 8T design due to the effects of transistor up-sizing which increases the area sensitive to radiation. The combined effect of aging and process variation on 6T and 8T SRAM cells is shown in Fig. 8. Figure 8 shows variation effect has severe impact at NTC, as the SER of the 6T and 8T SRAM cell designs in the near-threshold voltage domain is 4× higher than their SER in the super-threshold voltage domain.

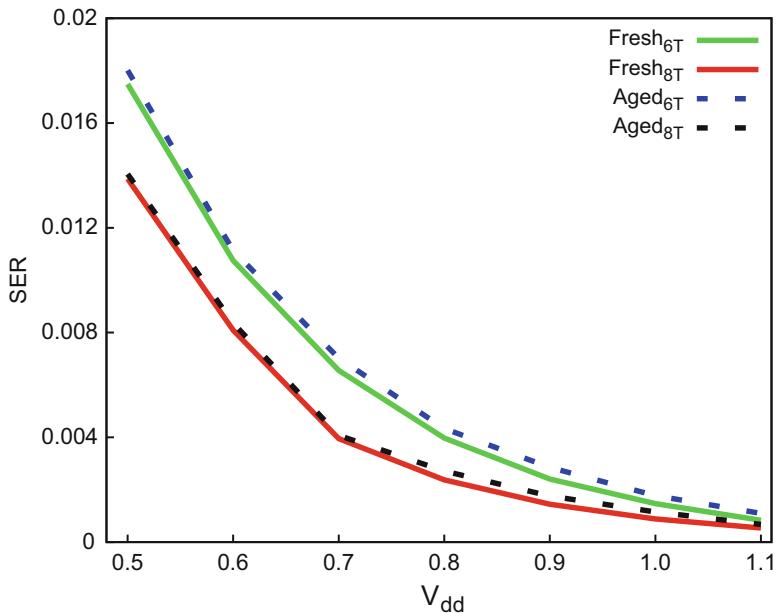


Fig. 7 SER rate of fresh and aged 6T and 8T SRAM cells for various V_{dd} values

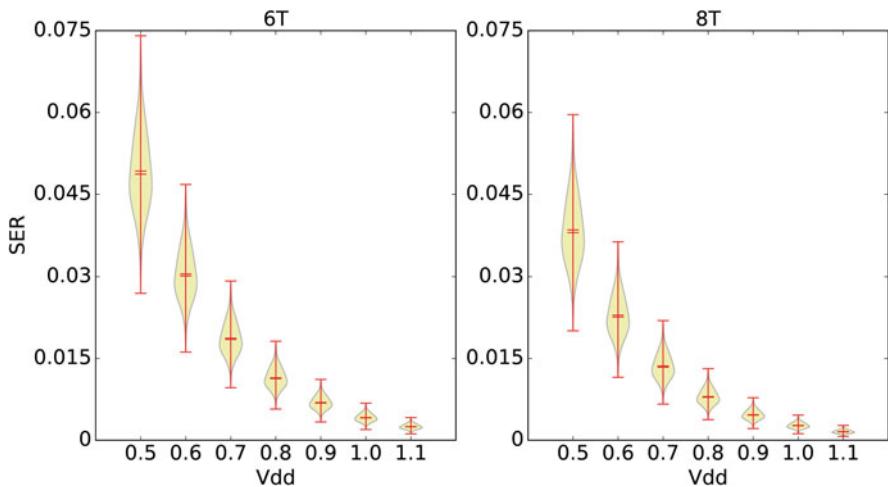


Fig. 8 SER of 6T and 8T SRAM cells in the presence of process variation and aging effects after 3 years of operation

3.3 Experimental Evaluation and Trade-Off Analysis

3.3.1 Experimental Setup

The reliability analysis is conducted using an ALPHA implementation of an embedded in-order core on the Gem5 architectural simulator [7]. Since cache memories are the main focus, various cache sizes (4–16 KB) and wide associativity range from simple directly mapped to 4-way set associative caches are assessed to perform a reliability and performance trade-off analysis. The evaluation is conducted using several workload applications from the SPEC2000 CPU benchmark suite [25]. The workload applications were executed for five million cycles by fast-forwarding to the memory intensive phases. The experimental setup used in this work is presented in Table 1.

The BTI-induced V_{th} shift is extracted by assuming 10% BTI-induced aging after 3 years of operation [37]. First, the 45 nm 6T and 8T SRAM cells are modeled using the PTM model. Afterward, the BTI-induced V_{th} shift LUT and the corresponding SNM degradation for various SP values (0.0–1.0) are obtained using a SPICE simulation. The impact of process variation is considered as a normal distribution of the transistor threshold voltage with a mean ($\mu = V_{th}$, 300 mV) and standard deviation (σ) obtained using the Pelgrom model given in Eq. (5).

To demonstrate the effect of soft error, neutron-induced soft errors are considered as they are the dominant soft error mechanisms at terrestrial altitudes. In order to ensure the proper functionality of both 6T and 8T SRAM cells in the near-threshold voltage domain, their transistors are sized according to the transistor sizing used to model and fabricate near-threshold 6T and 8T SRAM cells specified in [32]. It should be noted that L1 cache is used for illustration purpose only as most embedded

Table 1 Experimental setup, configuration, and evaluated benchmark applications

	Gem5	
Simulation environment	Near-threshold	Super-threshold
<i>Core configuration</i>		
Processor model	Embedded	Embedded
Architecture	Single in-order core	Single in-order core
ISA	ALPHA	ALPHA
Supply voltage	0.5 V	1.1 V
Frequency	100 MHz	1 GHz
Technology node	45 nm PTM	45 nm PTM
<i>Cache configuration</i>		
L1 Cache	Sizes = 4, 8, and 16 KB	Sizes = 4, 8, and 16 KB
	Associativity = 1, 2, and 4 way	Associativity = 1, 2, and 4 way
	Replacement policy = LRU	Replacement policy = LRU
	SRAM cells = 6T and 8T	SRAM cell = 6T
Benchmark	SPEC2000	SPEC2000

NTC processors have limited cache hierarchy. However, the framework is generic, and it is applicable to any cache levels such as L2 and L3.

3.3.2 Workload Effect Analysis

As discussed in Sect. 3.2, BTI-induced SNM degradation of SRAM cell highly depends on the cell's signal probability and the residency time of valid data which varies from one workload application to another. Similarly, the SER of memory components is dependent on the data residency period which is commonly measured using AVF. Hence, for SER analysis, the AVF of different workloads is obtained based on the workload application's data residency period. In order to show the effect of workload variation on SER and SNM degradation, the AVF and signal probabilities of the cache memory are extracted by running different workload applications from the SPEC2000 benchmark suite. Then, the corresponding SNM and SER of the cache memory are obtained using the SER and SNM models presented in Sect. 3.2.

3.3.3 Aging and Variation-Induced SNM Degradation

SNM degradation affects the metastability of SRAM cells. Metastability of SRAM cell determines the stability of the stored value, and it is highly dependent on the worst-case SNM degradation [18]. Therefore, for any workload application, the aging-induced SNM degradation should be evaluated based on the first cell to fail (worst-case SNM degradation).

The impact of workload on the SNM degradation of 6T and 8T based caches across wide supply voltage range is shown in Fig. 9a and b, respectively. For both cases, the SNM degradation increases significantly with supply voltage downscaling. Although the aging rate is slower at lower supply voltage values due to the lower temperature, the wide variation extent in NTC leads to higher aging sensitivity. Hence, in NTC the impact of process variation on SNM is more severe and leads to a significant increase in the aging sensitivity of SRAM cells.

3.3.4 Soft Error Rate Analysis

In order to analyze the impact of workload variation on the soft error rate of cache memories, the architectural vulnerability factor of each workload is extracted and combined with the circuit-level information. Figure 10 shows the contribution of the SPEC2000 workload applications on the SER of the 6T SRAM based cache. As shown in the figure, for all workload applications the SER increases significantly with supply voltage downscaling. For example, the SER of all workload applications increases by five orders of magnitude when the supply voltage is downscaled from the super-threshold voltage (1.1 V) to the near-threshold voltage domain (0.5 V).

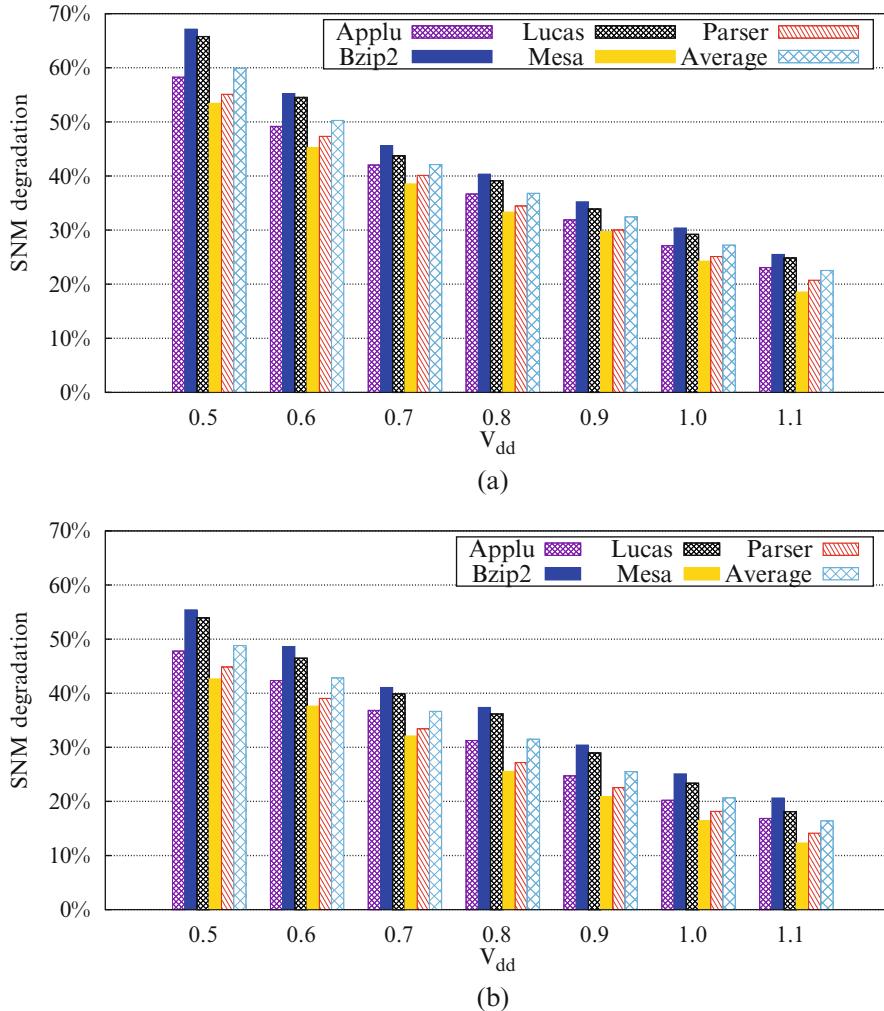


Fig. 9 Workload effects on aging-induced SNM degradation in the presence of process variation for 6T and 8T SRAM cell based cache after 3 years of operation **(a)** 6T SRAM based cache **(b)** 8T SRAM based cache

Additionally, the workload variation has a considerable impact on the soft error rate. For example, the SER of *Bzip2* is almost two orders of magnitude higher than the SER of *Mesa* and *Parser* workload applications. The workload variation impact is observed because *Bzip2* application has higher locality and hit rate which increases the data residency period when compared to the other workload applications. Although the higher hit rate of *Bzip2* leads to a better performance measured in Instructions Per Cycle (IPC), it has a significant impact on the soft error rate of the cache. Hence, it is essential to exploit the workload variation in

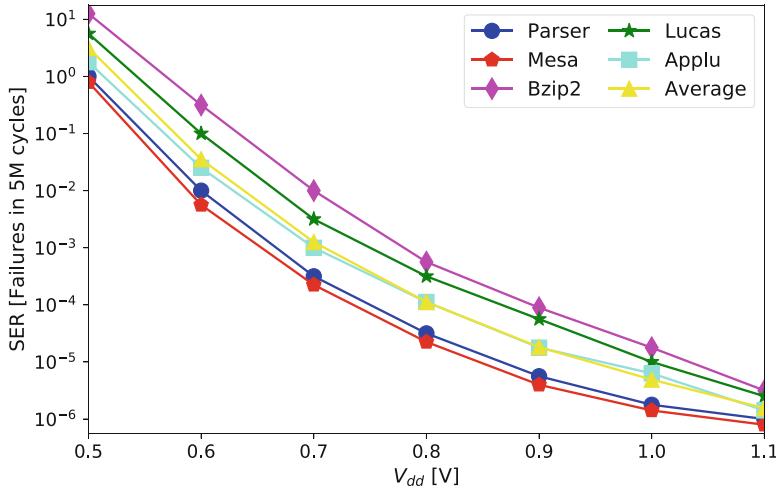


Fig. 10 Workload effect on SER rate of 6T SRAM cell based cache memory for wide supply voltage range

order to downscale the supply voltage of the cache memory in per-application bases for a given target error rate. For a given target FIT rate (e.g., 10^{-2}) the cache has to operate at 0.6 V for *Mesa* and *Parser* workload applications. However, for *Bzip2* the cache has to operate at a higher voltage (0.7 V) for the same target error rate.

3.3.5 Cache Organization Impact on System FIT Rate

Cache organization has a significant impact on the performance of embedded processors [34]. Similarly, the organization has an impact on the reliability of cache units. In NTC, the reliability impact of cache organization is even more pronounced. Hence, a proper cache size and associativity selection should consider both performance and reliability as target metrics. The system failure probability (FIT rate and SNM) of a cache unit is highly dependent on the architectural vulnerability factor and the values stored in the cache as well as their residency time intervals, which is in turn is a strong function of the read-write accesses of the cache. Hence, these parameters are influenced by cache size and associativity.

The performance and reliability impacts of different cache organizations in the near and super-threshold voltage domains are evaluated using the configurations described in Table 1. For near-threshold voltage (0.5 V) the processor core frequency is set to 100 MHz, and the cache latency is set to 1 cycle as gate delay is the dominant factor in the near-threshold voltage domain [12]. In the super-threshold voltage domain, however, the cache latency and interconnect delay have a significant impact on the overall delay. Thus, the cache hit latency is set to 2 cycles for 4 and 8 K cache sizes and 3 cycles for the 16 K cache size [41].

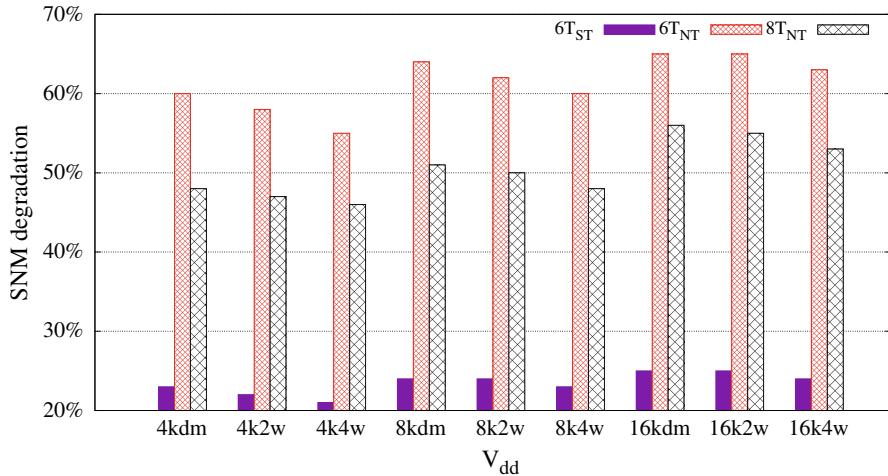


Fig. 11 Impact of cache organization on SNM degradation in near-threshold (NTC) and super-threshold (ST) in the presence of process variation and aging effect after 3 years of operation

3.3.6 Cache Organization and SNM Degradation

Since cache organization determines the data residency period, it has a direct impact on the SNM degradation. Figure 11 illustrates the impact of cache organization on the SNM degradation of near and super-threshold voltage 6T and 8T SRAM cell based memory arrays in the presence of process variation and aging effects after 3 years of operation. The figure shows smaller cache size with higher associativity (4 k-4 w) has less impact on SNM degradation as the data resides in the cache for a smaller duration.

3.3.7 Cache Organization and SER FIT Rate

The cache size and associativity also affect the ACE cycles of cache lines and their failure probabilities. The impact of the cache organization on the FIT rate and performance (IPC) varies along various supply voltage domains. In the super-threshold voltage, an increase in cache size and associativity improves the performance. However, from a FIT rate point of view, an increase in the cache size has a negative impact on FIT rate as it increases the FIT of the cache. Smaller cache sizes, however, have lower performance and better FIT rate. Figure 12 shows the design space of FIT rate and performance (IPC) impact of various cache organizations in the super-threshold voltage domain. In the figure, the FIT rate and performance optimal configuration is (8 k-4 w) as indicated by the blue italic font in Fig. 12.

In the near-threshold voltage domain, the performance is mainly dominated by the delay of the logic unit and the memory failure rate is significantly high. Therefore, it is essential to select a cache organization that gives better reliability

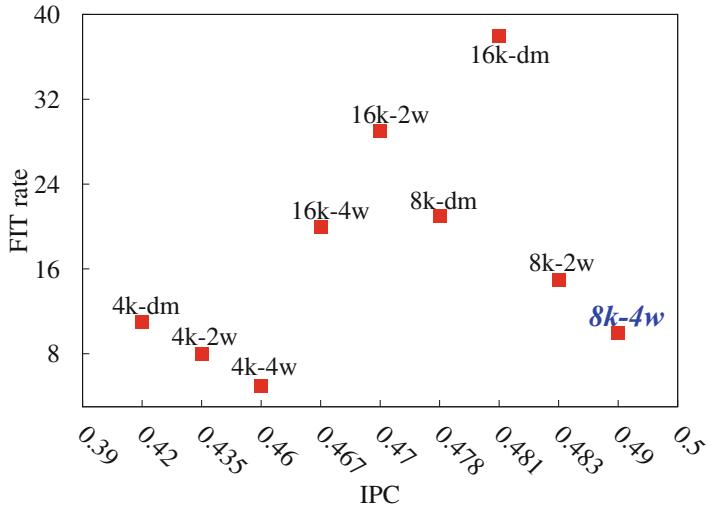


Fig. 12 FIT rate and performance design space of various cache configurations in the super-threshold voltage domain by considering average workload effect (the *blue italic font* indicates optimal configuration)

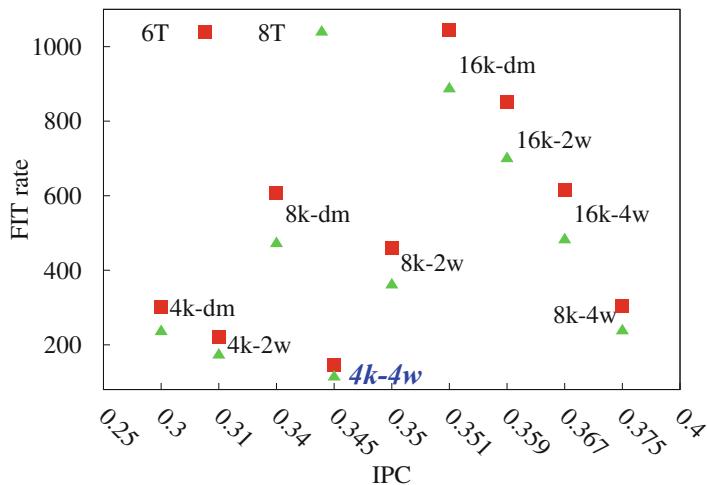


Fig. 13 FIT rate and performance design space of 6T and 8T designs for various cache configurations in the near-threshold voltage domain by considering average workload effect (the *blue italic font* indicates optimal configuration)

(FIT rate and SNM) than performance. Hence, in NTC a smaller cache size with higher associativity gives the best reliability and performance trade-off. Figure 13 shows the design space for the FIT rate and performance trade-off for 6T and 8T designs in NTC.

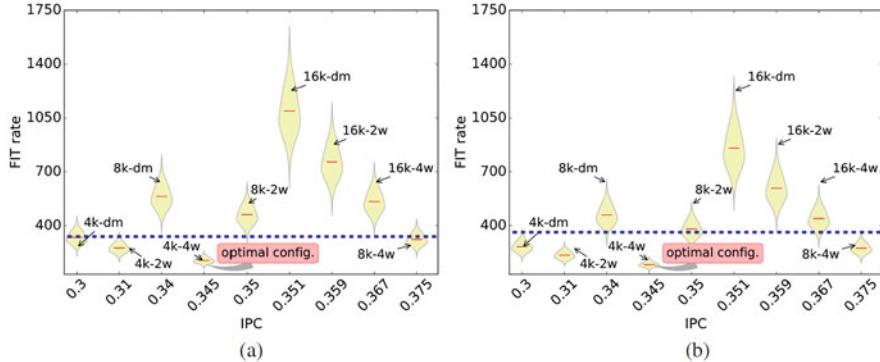


Fig. 14 FIT rate and performance trade-off analysis of near-threshold 6T and 8T caches for various cache configurations and average workload effect in the presence of process variation and aging effects. (a) Near-threshold 6T. (b) Near-threshold 8T

3.3.8 Reliability-Aware Optimal Cache Organization

The experimental results reported in Figs. 11, 12, 13, and 14 show an increase in the cache associativity improves the performance and reliability (both FIT rate and SNM). Hence, in the super-threshold voltage domain, medium cache size (e.g., 8 KB) with higher associativity has a better reliability and performance trade-off. In NTC, however, smaller cache sizes with higher associativity are preferable for two main reasons: (1) The performance is mainly dominated by the processor core, not by the cache units and hence, cache latency is not an important issue. (2) The soft error rate and SNM degradation are higher in NTC than in the super-threshold voltage domain. Hence, the cache size is reduced by half to obtain a better reliability and performance trade-off in NTC.

In the NTC domain, the selection of an optimal cache organization for the 6T SRAM cell based caches is different from the 8T based caches, depending on the FIT rate and performance requirement. For example, for a target tolerable FIT rate of 350 at NTC (as shown by the dotted line in Fig. 14a and b), only 4 KB 4-way associative cache organization is within the acceptable zone for the 6T-based cache. In the 8T-based cache, however, three additional cache organizations (4 K-dm, 4 k-2 w, and 8 k-4 w) are within the acceptable zone. Hence, the 8 k-4 w cache is used in the 8T-based cache to get $\approx 10\%$ performance improvement without violating the reliability constraint.

To implement the suggested cache organizations for a specific supply voltage value (only near-threshold or super-threshold) is straightforward. For caches that are expected to operate in both super and near-threshold voltage domains, the reliability-performance optimum cache organization in the super-threshold voltage (e.g., 4-way 8 KB in this case) is preferable. Then, when switching to the near-threshold voltage domain, some portion of the cache is disabled (power gated) in order to maintain the reliability-performance trade-off at NTC.

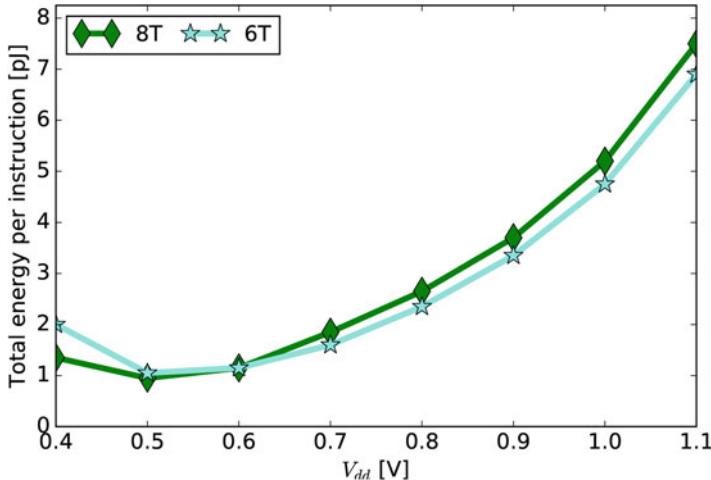


Fig. 15 Energy consumption profile of 6T and 8T based 4 K 4-way cache for wide supply voltage value ranges averaged over the selected workloads from SPEC2000 benchmarks

3.3.9 Overall Energy-Saving Analysis of 6T and 8T Caches

The energy-saving potential of supply voltage downscaling is evaluated by extracting the average energy consumption profile of the 4 K-Byte 4-way set associative cache (i.e., the reliability-performance optimal cache configuration) using 6T and 8T implementations. The energy consumption of the cache memory consists of three different components. These components are peripheries, row and column decoders, and bit-cell array energy consumptions. Since the energy consumption of the periphery and row/column decoder is independent of the bit-cell used, they are assumed to be uniform for both 6T and 8T based caches. Hence, the energy-saving comparison is done based only on the energy consumption of the bit-cell array.

Figure 15 compares the total energy consumption of the 6T and 8T based cache memories for a wide supply voltage range. As shown in the figure, the 8T based cache has slightly higher energy consumption in the super-threshold voltage domain (0.7–1.1 V) than the 6T based cache. The slightly higher energy consumption is because of the additional transistors used for read/write decoupling. However, due to the increase in the failure rate in the near-threshold domain, the 6T based cache consumes more energy than the corresponding 8T based implementation. The energy cost of the higher failure rate is considered as an increase in the read/write latency of the cache. This shows addressing the failures of the 6T cache in NTC results in additional energy cost which makes it less attractive for operating at lower supply voltage values (e.g., below 0.6 V).

3.3.10 Reliability Improvement and Area Overhead Analysis of 8T Based Caches

In a near-threshold voltage SRAM design, the 8T cell improves the soft error rate in the presence of aging and variation effects by up to 25%. Similarly, the SNM is improved by $\approx 15\%$ using 8T SRAM cells in NTC caches. However, it is expected that the 8T SRAM design has 30% area overhead than the 6T design due to the two additional access transistors. In practice, however, the overhead is much less. Since the 6T SRAM has to be up-sized to increase its read stability, the up-sizing increases the cell area of the 6T design to the extent of being larger than the area of 8T design, as experimentally demonstrated in [32].

4 Voltage Scalable Memory Failure Mitigation Scheme

As shown in the analysis presented in Sect. 3, process variation has a significant impact on the failure rate of memory components operating in the near-threshold voltage domain. Hence, addressing variation-induced memory failures plays an essential role in harnessing NTC benefits. One way of mitigating variation-induced memory failures is by determining the voltage downscaling potential of cache memories without surpassing the tolerable/correctable error margins. For this purpose, the operating voltage of caches should be gracefully reduced so that the number of failing bits due to permanent and transient failures remains tolerable.

This section presents a BIST based voltage scalable mitigation technique to determine an error-free supply voltage downscaling potential of caches at runtime. In order to reduce the runtime configuration complexities, the cache organizations such as size, associativity, and block size are determined during design time. In this work, the block size is considered as the smallest unit used to transfer data to and from the cache. Then, a BIST based runtime cache operating voltage downscaling analysis is performed for a given cache organization. To illustrate the impact of block size selection, the voltage downscaling potential of two block sizes is studied.

4.1 Motivation and Idea

Due to the wide variation extent in NTC, different memory cells have different SNM values; as a result, their minimum operating voltages for a proper functionality vary significantly. The cells with smaller SNM values need to operate at a higher supply voltage than the cells with larger SNM values. Therefore, the supply voltage of some cells (cells with smaller SNM value) should be scaled down more conservatively than the cells with larger SNM in order to maintain the overall reliability. This idea is exploited in order to minimize the effect of process variation and determine error tolerant/error-free voltage downscaling potential of near-threshold caches. Since

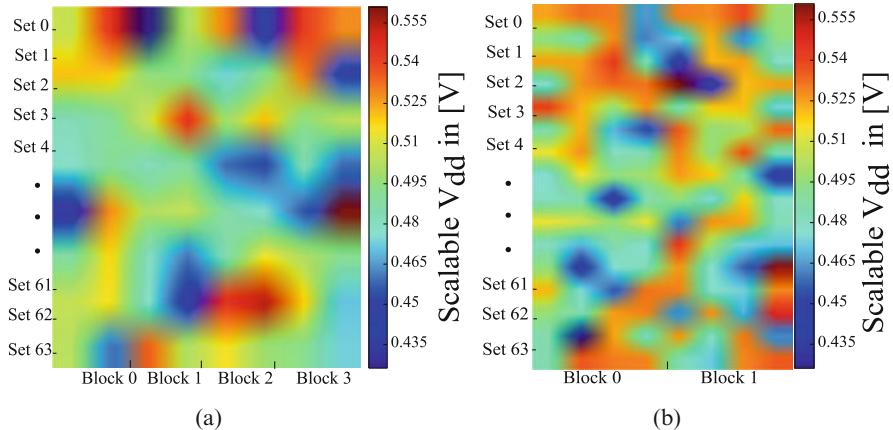


Fig. 16 Error-free minimum operating voltage distribution of 8 MB cache, Set size = 128 Byte (a) block size = 32 Bytes (4 blocks per set) and (b) block size = 64 Bytes (two blocks per set), the cache is modeled as 45 nm node in CACTI

cache memories are divided into several blocks, block size selection has a significant impact on the supply voltage downscaling potential of cache memories. Hence, one needs to analyze the impact of process variation and supply voltage downscaling potential of cache memories in a per block bases.

Cache block size has a substantial impact on the miss rate and miss penalty of caches at the same time. In order to reduce the cache miss rate and its associated penalty, a larger block size is preferable as it improves locality and reduces the miss rate. From a reliability point of view, however, larger block sizes have wide variation extent, and as a result more failing cells in NTC, which makes the entire block fail. These failures force the cache memory to operate at a much higher voltage (i.e., more conservative scaling) leading to a significant reduction in the energy efficiency. However, this is addressed by decreasing the cache block size in order to reduce cache operating voltage as the variation extent is minimal in comparison to larger block sizes.

To exploit this fact, the impact of block size selection on the supply voltage downscaling potential of a near-threshold voltage 8 KB cache is evaluated as shown in Fig. 16. The cache is modeled in CACTI [36] with 128 Byte set size and two different block sizes, and the impact of process variation is modeled using the threshold voltage variation model given in Eq. (5). As shown in the figure, the smaller block size (Fig. 16a) has narrow variation extent, and hence, it has more supply voltage downscaling potential than its larger block size counterpart (Fig. 16b) at design time. During operation time, the supply voltage downscaling potential of the larger block size cache is reduced further due to various runtime factors such as aging-induced SNM degradation and SER. Moreover, smaller block sizes have lower multiple bit failure rates, and hence, simpler ECC schemes are adopted at a minimum cost [2]. Table 2 shows the ECC overhead comparison for 64 and 32 Byte

Table 2 ECC overhead analysis of different block sizes and correction capabilities

ECC schemes	Block size = 64 Byte			Block size = 32 Byte		
	Area overhead	Storage overhead	Latency overhead	Area overhead	Storage overhead	Latency overhead
SECDED	13k gates	11 bits	2 cycle	\approx 4k gates	10 bits	1 cycle
DECTED	>50k gates	21 bits	4 cycles	\approx 10k gates	19 bits	2 cycle
4EC5ED	\approx 60k gates	41 bits	15 cycles	\approx 50k gates	37 bits	9 cycle

block sizes according to [2]. The table shows dividing the cache into smaller blocks has an advantage in terms of ECC overhead. Therefore, appropriate cache block size selection should consider both performance and reliability effects at the same time in order to achieve maximum performance while operating within the tolerable reliability margin. Once the cache block size is determined, the cache supply voltage should be tuned at runtime to incorporate the runtime reliability effects such as aging. For this purpose, a BIST based supply voltage tuning is used, and its concept is discussed in the following subsection.

4.2 Built-In Self-Test (BIST) Based Runtime Operating Voltage Adjustment

Built-In Self-Test (BIST) is a widely used technique to test VLSI system on chip [22]. Since memory components occupy majority of the chip area, BIST plays a significant role in testing large and complex memory arrays easily [5, 22]. In order to determine the runtime supply voltage downscaling potential of caches, it is essential to assume a cache memory is equipped with BIST infrastructure to test the entire memory.

In a conventional BIST, the BIST controller generates the test addresses and test patterns (finite number of read/write operations). Then, the test is performed, and the test result is compared with the expected response to determine the failing cells [5]. In this case, however, since the BIST module has to determine the minimum scalable voltage of each block, the test controller has to be modified in order to iteratively test and generate the minimum scalable voltages of each block. The goal is first to determine the error-free minimum scalable voltage of each cache block with/without error correction hardware. Then, the cache operating voltage is determined based on the block with higher operational voltage as shown in Eq. (7), such that the runtime memory failure is minimized.

$$V_{dd}^{\text{cache}} = \max_{0 \leq i \leq N-1} V_{dd}^{B_i} \quad (7)$$

where N is the total number of cache blocks, and $V_{dd}^{B_i}$ is the runtime minimum scalable voltage of block B_i obtained using the iterative BIST.

Algorithm 1 Runtime cache operating voltage adjustment

```

1: function CACHE-Vdd-SCALING ( $C_s, V_{dd}, B_s, F_m$ ) { $C_s$ =cache size,  $V_{dd}$ = operating voltage,  $B_s$ =block size,
    $F_m$ =tolerable margin of failing bits}
2:    $B_t \leftarrow \frac{C_s}{B_s}$ ; { $B_t$ =total number of cache blocks}
3:   for block i ← 1 to  $B_t$  do
4:      $F_c \leftarrow 0$ ; { $F_c$ =failing cells counter}
5:      $V_{dd}^{new} \leftarrow V_{dd}$ ; { $V_{dd}^{new}$ =voltage used to perform BIST}
6:     while  $F_c \leq F_m$  do
7:       Perform BIST using  $V_{dd}^{new}$ ;
8:        $F_c \leftarrow$  failing cells; {total number of failing cells per block}
9:        $V_{dd}^{new} \leftarrow V_{dd}^{new} - \Delta V_{dd}$ ; {reduce operating voltage by  $\Delta V_{dd}$ }
10:      end while
11:    end for
12:     $V_{dd}^{cache} \leftarrow \max_{1 \leq i \leq B_t} V_{dd_i}^{new}$ ; { $V_{dd_i}^{new}$  new operating voltage of blocki }

```

Algorithm 1 presents the iterative BIST technique used to determine the minimum scalable voltage of cache memory by considering permanent and runtime memory failures. The algorithm takes cache size (C_s), operating voltage (V_{dd}), block size (B_s), and tolerance margin (F_m) as its input. Then, the number of cache blocks is determined by dividing the cache size by the block size (Step 2). Afterward, the minimum scalable voltage of each block is obtained by gradually reducing the operating voltage, and conducting block-level BIST to determine the total number of failing bits at each operating voltage level (Steps 3–10). It should be noted that, the supply voltage is reduced as long as the number of failing bits per block is within the tolerable/correction capability of the adopted error correction scheme. For example, a cache memory equipped with a Single Error Correction Double Error Detection (SECDED) infrastructure tolerates two failing bits per block (hence $F_m = 2$) as SECDED corrects only one bit and detects two erroneous bits at a time. Hence, whenever two failing bits are detected the error-free version is loaded from the lower-level memory which makes SECDED sufficient solution for tolerating two failing bits per block. Finally, the algorithm determines the operating voltage of the cache based on the block with the highest voltage as shown in Step 12.

The overall flow of the cache access control logic along with the BIST infrastructure as well as mapping logic is presented in Fig. 17. The cache controller first decodes the address and identifies the requested block. Then, it determines if the requested block is functional or failing block for the specified operating voltage. If the requested block is functional, then a conventional block access is performed. In case the requested block is a failing one, the error tolerant block mapping scheme is employed to redirect the access request.

Since this approach considers the effect of permanent and transient failure mechanisms, it is orthogonal with different dynamic cache mitigation schemes such as block disabling [1, 43] and strong ECC schemes [2]. For energy-critical systems, block disabling technique is applied in combination with this approach to downscale the cache operating voltage aggressively by disabling the failing blocks at lower operating voltages at the cost of performance reduction (increase in miss rate).

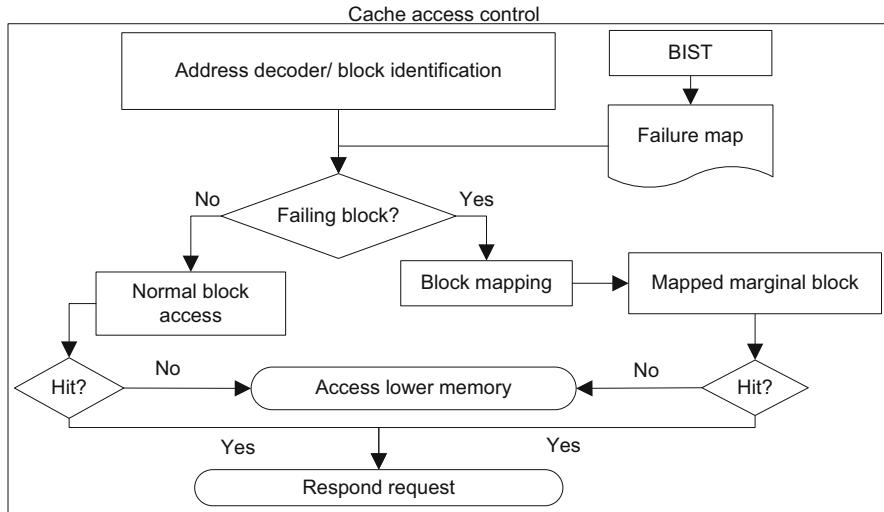


Fig. 17 Cache access control flowchart equipped with BIST and block mapping logic

4.3 Error Tolerant Block Mapping

Once the minimum scalable voltages of the cache blocks are determined, the next task is to disable the failing blocks, and map their read/write accesses to the corresponding non-failing blocks in order to ensure reliable cache operation. Additionally, in order to reduce the vulnerability to runtime failures (such as noise and soft errors), the non-failing blocks are stored in a stack frame sorted by their minimum scalable voltage values. Since the marginal blocks (blocks with less voltage downscaling potential) are more sensitive to runtime failures, they are stored at the top of the stack. Then, access to a disabled block is mapped to the marginal blocks in the stack. The mapping enables to reduce soft error vulnerability of the marginal blocks by reducing their data residency period. Since a stack is a linear data structure in which the insertion and deletion operations are performed at only one end commonly known as “top,” the marginal blocks need to be at the top (upper half) of the stack to ensure their fast replacement.

The mapping process is illustrated in Fig. 18 by using an illustrative example. As shown in the figure, the cache blocks are divided into three categories: (1) red blocks are failing blocks. (2) yellow blocks are marginal blocks (non-failing but with limited supply voltage downscaling potential). (3) blue blocks are robust blocks (i.e., non-failing with higher supply voltage downscaling potential). Hence, the marginal blocks are stored at the top of the stack frame. Then, when a disabled (failing) block is requested (e.g., B5) its access request is mapped to a marginal block at the top of the stack frame (e.g., B4), and the stack pointer is updated to point to the next element in the stack. This process continues until all the disabled blocks are mapped. It should be noted that once a block is mapped, it is removed from the mapping stack

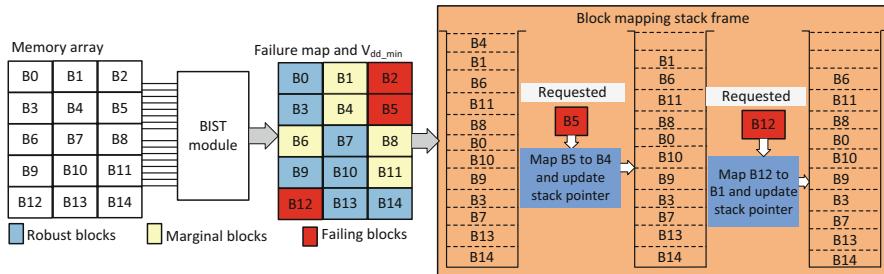


Fig. 18 Error tolerant cache block mapping scheme (mapping failing blocks to marginal blocks)

Table 3 Minimum scalable voltage analysis for different ECC schemes

ECC-Scheme	Minimum scalable voltage in [V]		
	Block size = 16Byte	Block size = 32Byte	Block size = 64Byte
No-ECC	0.50	0.53	0.54
Parity	0.47	0.51	0.53
SECDED	0.43	0.48	0.50

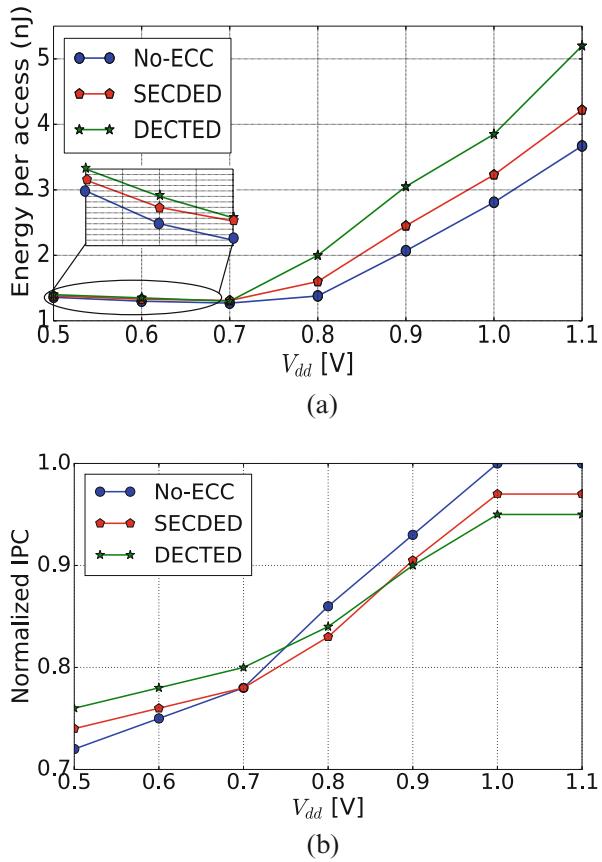
when updating the stack pointer. For example, when block B5 is mapped to block B4, then, block B4 is removed from the stack as shown by the empty slot in Fig. 18.

4.4 Evaluation of Voltage Scalable Mitigation Scheme

4.4.1 Variation-Aware Voltage Scaling Analysis

The supply voltage scalability of three different block sizes (16, 32, and 64 Byte) with different error correction schemes is compared in order to analyze the impact of block size selection on the supply voltage downscaling potential of cache memories with and without error correction schemes. The error-free (correctable error) minimum voltage of three block sizes is studied for 8 KB cache memory without ECC, parity, and Single Error Correction Double Error Detection (SECDED) configurations. Table 3 shows the supply voltage downscaling potential of the studied block sizes. For all ECC schemes (given in Table 3), the cache operating voltage has to be downscaled more conservatively when the block size is larger (64 Bytes). However, larger block sizes help to reduce the cache miss rate that results in a better cache performance. Therefore, for an aggressive supply voltage downscaling, the block size should be selected as small as possible by making performance and energy-saving trade-off analysis.

Fig. 19 Comparison of voltage downscaling in the presence of block disabling and ECC induce overheads for gzip, parser, and mcf applications from SPEC2000 benchmark (a) energy consumption comparison (b) Performance comparison in terms of IPC



4.4.2 Energy and Performance Evaluation of Voltage Scalable Cache Different ECC Schemes

The average energy reduction and performance comparison of voltage scaled cache memory with and without ECC are given in Fig. 19a and b by running selected workloads (*gzip*, *parser*, and *mcf*) from the SPEC2000 benchmark. The energy results in Fig. 19a are extracted from CACTI by considering block disabling, and ECC induced delay and energy overheads. As shown in the figure, supply voltage downscaling improves the energy efficiency significantly. However, the overheads of this scheme, namely ECC energy overhead, block disabling induced cash miss rate, and ECC encoding/decoding delay overhead outweigh the energy gain of supply voltage downscaling when the cache operating voltage is below 0.7 V. Therefore, the energy per access of Double Error Correction Triple Error Detection (DECTED) is higher than SECDED when the supply voltage is scaled down to 0.7 V or below. Similarly, Fig. 19b shows the cache performance (IPC) is reduced significantly with the supply voltage downscaling as more blocks are disabled for reliable operation.

5 Conclusion

Embedded microprocessors, particularly for battery-powered mobile applications, and energy-harvested Internet of Things (IoT) are expected to meet stringent energy budgets. In this regard, operating in the near-threshold voltage domain provides better performance and energy efficiency trade-offs. However, NTC faces various challenges among which increase in functional failure rate of memory components is the dominant issue. This chapter analyzed the combined effect of aging, process variation, and soft error on the reliability of cache memories in super and near-threshold voltage domains. It is observed that the combined effect of process variation and aging has a massive impact on the soft error rate and SNM degradation of NTC memories. Experimental results show process variation and aging-induced SNM degradation is $2.5 \times$ higher in NTC than in the super-threshold voltage domain while SER is $8 \times$ higher. The use of 8T instead of 6T SRAM cells reduces the system-level SNM and SER by 14% and 22%, respectively. Additionally, workload and cache organization have a significant impact on the FIT rate and SNM degradation of memory components. This chapter demonstrated that the reliability and performance optimal cache organization changes when going from the super-threshold voltage to the near-threshold voltage domain.

References

1. Agarwal, A., Paul, B.C., Mahmoodi, H., Datta, A., Roy, K.: A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Trans. Very Large Scale Integr. Syst.* **13**, 27–38 (2005)
2. Alameldeen, A.R., Wagner, I., Chishti, Z., Wu, W., Wilkerson, C., Lu, S.L.: Energy-efficient cache design using variable-strength error-correcting codes. In: ACM SIGARCH Computer Architecture News (2011)
3. Aly, R.E., Faisal, M.I., Bayoumi, M.A.: Novel 7T sram cell for low power cache design. In: Proceedings of the 2005 IEEE International SOC Conference (2005)
4. Amrouch, H., van Santen, V.M., Ebi, T., Wenzel, V., Henkel, J.: Towards interdependencies of aging mechanisms. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (2014)
5. Au, A., Pogiel, A., Rajski, J., Sydow, P., Tyszer, J., Zawada, J.: Quality assurance in memory built-in self-test tools. In: 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (2014)
6. Autran, J.L., Serre, S., Munteanu, D., Martinie, S., Semikh, S., Sauze, S., Uznanski, S., Gasiot, G., Roche, P.: Real-time soft-error testing of 40 nm SRAMS. In: Proceedings of IEEE International Reliability Physics Symposium (IRPS) (2012)
7. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. *ACM SIGARCH Comput. Archit. News* **39**, 1–7 (2011)
8. Calhoun, B.H., Chandrakasan, A.: A 256kb sub-threshold SRAM in 65nm CMOS. In: 2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers (2006)
9. Cazeaux, J.M., Rossi, D., Omana, M., Metra, C., Chatterjee, A.: On transistor level gate sizing for increased robustness to transient faults. In: 11th IEEE International On-Line Testing Symposium, IOLTS (2005)

10. Chang, L., Montoye, R.K., Nakamura, Y., Batson, K.A., Eickemeyer, R.J., Dennard, R.H., Haensch, W., Jamsek, D.: An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches. *IEEE J. Solid-State Circ.* **43**, 956–963 (2008)
11. Chatterjee, I., Narasimham, B., Mahatme, N., Bhava, B., Reed, R., Schrimpf, R., Wang, J., Vedula, N., Bartz, B., Monzel, C.: Impact of technology scaling on SRAM soft error rates. *IEEE Trans. Nuclear Sci.* **61**, 3512–3518 (2014)
12. Chen, H., Manzi, D., Roy, S., Chakraborty, K.: Opportunistic turbo execution in NTC: exploiting the paradigm shift in performance bottlenecks. In: Proceedings of the 52nd Annual Design Automation Conference (2015)
13. Chen, Y.H., Chan, W.M., Wu, W.C., Liao, H.J., Pan, K.H., Liaw, J.J., Chung, T.H., Li, Q., Lin, C.Y., Chiang, M.C., et al.: A 16 nm 128 Mb SRAM in high- κ metal-gate FinFET technology with write-assist circuitry for low-VMIN applications. *IEEE J. Solid-State Circuits* **50**, 170–177 (2015)
14. Dreslinski, R., Wieckowski, M., Blaauw, D.S., Mudge, T.: Near threshold computing: Overcoming performance degradation from aggressive voltage scaling. In: Proceedings of the Workshop Energy-Efficient Design (2009)
15. Dreslinski, R.G., Wieckowski, M., Blaauw, D., Sylvester, D., Mudge, T.: Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. *Proc. IEEE* **98**, 253–266 (2010)
16. Ebrahimi, M., Evans, A., Tahoori, M.B., Seyyedi, R., Costenaro, E., Alexandrescu, D.: Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales. In: Proceedings of the conference on Design, Automation & Test in Europe (2014)
17. Gebregiorgis, A., Tahoori, M.B.: Reliability and performance challenges of ultra-low voltage caches: A trade-off analysis. In: IEEE 24th International Symposium on On-Line Testing and Robust System Design (IOLTS) (2018)
18. Gebregiorgis, A., Ebrahimi, M., Kiamehr, S., Oboril, F., Hamdioui, S., Tahoori, M.B.: Aging mitigation in memory arrays using self-controlled bit-flipping technique. In: 20th Asia and South Pacific Design Automation Conference (ASP-DAC) (2015)
19. Gebregiorgis, A., Kiamehr, S., Oboril, F., Bishnoi, R., Tahoori, M.B.: A cross-layer analysis of soft error, aging and process variation in near threshold computing. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2016)
20. Gebregiorgis, A., Bishnoi, R., Tahoori, M.B.: A comprehensive reliability analysis framework for NTC caches: a system to device approach. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **38**, 439–452 (2018)
21. Grossar, E., Stucchi, M., Maex, K., Dehraene, W.: Read stability and write-ability analysis of SRAM cells for nanometer technologies. *IEEE J. Solid-State Circuits* **41**, 2577–2588 (2006)
22. Hamdioui, S., Al-Ars, Z., Gaydadjiev, G.N., Van de Goor, A.: Generic march element based memory built-in self test. US Patent 8,910,001, 2014
23. Hazucha, P., Svensson, C.: Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. Nuclear Sci.* **47**, 2586–2594 (2000)
24. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., Wehn, N.: Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In: Proceedings of the 50th Annual Design Automation Conference (2013)
25. Henning, J.L.: SPEC CPU2000: Measuring CPU performance in the new millennium. *Computer* **33**, 28–35 (2000)

26. Hubert, G., Artola, L., Regis, D.: Impact of scaling on the soft error sensitivity of bulk, FDSOI and FinFET technologies due to atmospheric radiation. *Integration* **50**, 39–47 (2015)
27. Jahinuzzaman, S.M., Sharifkhani, M., Sachdev, M.: An analytical model for soft error critical charge of nanometric SRAMs. *IEEE Trans. Very Large Scale Integr. Syst.* **17**, 1187–1195 (2009)
28. Jeppson, K.O., Svensson, C.M.: Negative bias stress of MOS devices at high electric fields and degradation of MNOS devices. *J. Appl. Phys.* **48**, 2004–2014 (1977)
29. Jiang, C., Zhang, D., Zhang, S., Wang, H., Zhuang, Z., Yang, F.: A yield-driven near-threshold 8-T SRAM design with transient negative bit-line scheme. In: 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC) (2017)
30. Kuhn, K.J., Giles, M.D., Becher, D., Kolar, P., Kornfeld, A., Kotlyar, R., Ma, S.T., Maheshwari, A., Mudanai, S.: Process technology variation. *IEEE Trans. Electr. Devices* **58**, 2197–2208 (2011)
31. Maric, B., Abella, J., Valero, M.: Adam: An efficient data management mechanism for hybrid high and ultra-low voltage operation caches. In: Proceedings of the Great Lakes Symposium on VLSI (2012)
32. Morita, Y., Fujiwara, H., Noguchi, H., Iguchi, Y., Nii, K., Kawaguchi, H., Yoshimoto, M.: An area-conscious low-voltage-oriented 8T-SRAM design under DVS environment. In: IEEE Symposium on VLSI Circuits (2007)
33. Mukhopadhyay, S., Mahmoodi, H., Roy, K.: Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **24**, 1859–1880 (2005)
34. Olorode, O., Nourani, M.: Improving performance in sub-block caches with optimized replacement policies. *ACM J. Emerg. Technolo. Comput. Syst.* **11**, 1–22 (2015)
35. Seok, M., Chen, G., Hanson, S., Wieckowski, M., Blaauw, D., Sylvester, D.: CAS-FEST 2010: Mitigating variability in near-threshold computing. *IEEE J. Emer. Sel. Topics Circuits Syst.* **1**, 42–49 (2011)
36. Shivakumar, P., Jouppi, N.P.: Cacti 3.0: An integrated cache timing, power, and area model. Technical Report, Compaq Computer Corporation (2001)
37. Siddiqua, T., Gurumurthi, S., Stan, M.R.: Modeling and analyzing NBTI in the presence of process variation. In: 12th International Symposium on Quality Electronic Design (ISQED) (2011)
38. Sridharan, V., Kaeli, D.R.: Using hardware vulnerability factors to enhance AVF analysis. *ACM SIGARCH Comput. Archit. News* **38**, 461–472 (2010)
39. Takeda, K., Hagihara, Y., Aimoto, Y., Nomura, M., Nakazawa, Y., Ishii, T., Kobatake, H.: A read-static-noise-margin-free SRAM cell for low-VDD and high-speed applications. *IEEE J. Solid-State Circuits* **41**, 113–121 (2006)
40. Tonfat, J., Azambuja, J.R., Nazar, G., Rech, P., Frost, C., Kastensmidt, F.L., Carro, L., Reis, R., Benfica, J., Vargas, F., et al.: Analyzing the influence of voltage scaling for soft errors in SRAM-based FPGAs. In: 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS) (2013)
41. Understanding CPU caching and performance (2015). <http://http://arstechnica.com/gadgets/2002/07/caching/2/>

42. Wilkening, M., Sridharan, V., Li, S., Previlon, F., Gurumurthi, S., Kaeli, D.R.: Calculating architectural vulnerability factors for spatial multi-bit transient faults. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (2014)
43. Wilkerson, C., Gao, H., Alameldeen, A.R., Chishti, Z., Khellah, M., Lu, S.L.: Trading off cache capacity for reliability to enable low voltage operation. In: ACM SIGARCH Computer Architecture News (2008)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part IV

Cross-Layer from Physics to Gate- and Circuit- Levels

Mehdi Tahoori

The rapid shrinking of device geometries in the nanometer regime has led to the need for new system-on-chip (SoC) design methodologies at various levels of abstraction. Improvements in chip manufacturing technology and system integration have propelled an astonishing growth of computing systems which are integrated into almost all aspects of our daily lives. However, this trend is facing serious challenges, both at device and system levels. At the device level, as the minimum feature size continues to shrink, a host of vulnerabilities influence the robustness and reliability of computing systems. Some of these factors are caused by the stochastic nature of the nanoscale manufacturing process (e.g., process variability, sub-wavelength lithographic inaccuracies), while other factors appear because of high operating frequencies and intrinsic nanoscale features (e.g., RLC noise, on-chip temperature variation, increased sensitivity to radiation and device aging). Therefore, the reliability of systems on chip is not only limited to the technology parameters and hardware design, it is highly influenced by the runtime environment and executed workload, which can aggravate hardware stress and cause failures.

The chapters presented in this section are looking at the reliability issues from the circuit, logic, and physical design perspective, while linking the issues to the technology from one side and higher abstraction level (architecture, system, and workload) from the other end, therefore they represent a cross-layer angle on these reliability challenges.

The first chapter in this section addressed transistor aging in system bistables, mainly flip-flops. It presents various methods to improve the reliability of gate-level digital circuits by addressing the timing degradation of flip-flops under severe aging and voltage-drop. This is achieved through selective flip-flop optimization. The idea presented in this chapter is to find timing-critical flip-flops under high aging and/or voltage-drop impact, and selectively re-optimize them for operating under such

stress by appropriately sizing their transistors. Therefore, instead of replacing all flip-flops in the netlist with the hardened versioned, only the ones which are subject to extreme stress are replaced by hardened version. This is achieved by a cross-layer analysis which based on the architecture and analyzing the running workload, the flip-flops under severe aging and voltage fluctuation stress are identified. This effectively improves the reliability and lifetime of circuits without imposing much overhead, because these flip-flops constitute a small portion of all flip-flops.

The second chapter in this section focuses on electromigration (EM) which is the major interconnect aging effect in advanced technology nodes. It provides techniques for power grid network sizing while considering electromigration reliability. This chapter starts with an overview power grid network and electromigration fundamentals. The main issue addressed in this chapter is EM immortality and aging effects, used as EM constraints when formulating the optimization problems. When an interconnect line is below a critical stress, the void nucleation cannot occur and hence the metal wires become immortal and will not fail. The chapter first shows that the new Power/ground (P/G) optimization problem, subject to the voltage IR drop and new EM constraints, can still be formulated as an efficient sequence of linear programming (SLP) problem. The new optimization will ensure that none of the wires fail if all the constraints are satisfied. However, requiring all the wires to be EM immortal can be over-constrained. To mitigate this problem, the improvement is to consider the aging effects of interconnect wires in P/G networks.

The third chapter is devoted to various monitoring circuitry for improving cross-layer resiliency. The role of monitor circuits is to establish a bridge between the hardware and other layers by providing information about the devices and the operating environment in runtime. This chapter explores delay-based monitor circuits for design automation with the existing cell-based design methodology. The chapter reviews several design techniques to monitor parameters of threshold voltage, temperature, leakage current, critical delay, and aging. The chapter then demonstrates a reconfigurable architecture to monitor multiple parameters with small area footprint. Finally, an extraction methodology of physical parameters is discussed for model-hardware correlation.

The last chapter discusses yield and aging in scaled technologies. For the robustness of VLSI design methodology and cycles, reliability and yield need to be accurately modeled, systematically optimized, and seamlessly integrated into the existing design flow. This chapter will survey critical aging and yield issues, and then review the state-of-the-art techniques to tackle them, including both modeling and optimization strategies which reside across the Physics and Circuit/Gate layers as part of the overall dependability scheme. The strategies often involve synergistic cross-layer optimization due to the complicated VLSI design procedures nowadays. Novel modeling techniques leveraging machine learning are analyzed along with analytical optimization approaches.

Selective Flip-Flop Optimization for Circuit Reliability



Mohammad Saber Golanbari, Mojtaba Ebrahimi, Saman Kiamehr,
and Mehdi B. Tahoori

1 Introduction, Motivation, and Contributions

VLSI circuits are influenced by several sources of process and runtime variabilities [16]. Among them, supply voltage fluctuation and transistor aging due to BTI are the most important factors [2, 30, 36]. They degrade the performance of VLSI circuits by increasing the delay, and consequently deteriorate lifetime.

The impacts of both voltage-drop and aging are significant on sequential elements such as flip-flops and latches. Due to particular aspects of flip-flops, such as the internal feedback structure, degradation of the transistors of a flip-flop as well as supply voltage fluctuation may lead to serious timing degradation or even functional failure (inability to capture the input independent of timing) [24]. Furthermore, many flip-flops are on the critical paths of a circuit because logic synthesis tools balance the delays of circuit paths to achieve the best performance, area, and power. Therefore, it is necessary to employ design-time mitigation techniques to consider and control such gradual degradation, e.g. by adding appropriate timing margins (*aging and voltage-drop guardband*) [20, 28].

Our analysis shows that in a typical digital design such as a microprocessor, based on the functionality of different components, some flip-flops operate under static or near-static BTI stress, irrespective of the workload. These flip-flops experience large timing degradation because the flip-flop input Signal Probability (SP) is very close to 0.0 or 1.0. Being subject to severe BTI stress, the aforementioned flip-flops degrade faster, imposing a large *aging guardband* to the entire circuit. Flip-flops also experience a large temporally localized voltage-drop, because they are synchronized with the clock edge and supposedly operate at the same time (at

M. S. Golanbari · M. Ebrahimi · S. Kiamehr · M. B. Tahoori (✉)
Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: golanbari@kit.edu; mehdi.tahoori@kit.edu

clock edge), hence, drawing substantial current leading to a significant voltage-drop over Power Delivery Network (PDN) [22]. Moreover, recent studies have shown that the voltage-drop impact gets more severe by technology scaling [2, 21, 38]. Therefore, in a conventional design flow, costly *voltage-drop timing guardband* is considered for reliable circuit operation [22].

In this chapter, we explore methods to improve circuit reliability by addressing the timing degradation of flip-flops under severe aging¹ and voltage-drop, i.e. *selective flip-flop optimization*. The idea is to find timing-critical flip-flops under high aging and/or voltage-drop impact, and selectively re-optimize them for operating under such stress by appropriately sizing their transistors. This effectively improves the reliability and lifetime of circuits without imposing much overhead, because these flip-flops constitute a small portion of all flip-flops.

Simulation results obtained by applying the proposed method to a processor show that the flip-flops optimized with the proposed method exhibit much less delay degradation, while imposing less than 0.1% leakage power overhead to the processor. As a result, the required timing guardband of the processor using the proposed method is significantly less compared to the original processor. Therefore, given a specific clock period, the optimized processor design with the proposed method has 36.9% longer lifetime and better reliability compared to the original processor design.

2 Variability Impact on Flip-Flops

2.1 Flip-Flop Timing

Flip-flop timing metrics such as setup-time (U), hold-time (H), clock-to-q (D_{CQ}), and data-to-q (D_{DQ}) are well discussed in [31, 34]. When the setup-time is large enough, the clock-to-q value is almost constant, but further reduction of the setup-time will increase the clock-to-q value monotonously until a value after which the flip-flop is unable to capture and latch the input [31]. Based on this, the *optimum setup-time* is defined as the setup-time value which causes the clock-to-q value to increase by 10% from its minimum value [32]. Moreover, each flip-flop has two internal paths; one for transferring the input state “zero” to the output i.e. High-to-Low (HL) input transition, and the other for transferring the input state “one” to the output i.e. Low-to-High (LH) input transition. Basically, the timing parameters for these two internal paths can be different [24] as shown in Fig. 1, meaning that there are two sets of timing parameters for internal LH and HL paths of a flip-flop:

$$\{U_{LH}, D_{CQ_{LH}}, D_{DQ_{LH}}\} \quad \text{for LH transition,}$$

¹We consider the impact of Negative Bias Temperature Instability (NBTI) on PMOS transistors, and Positive Bias Temperature Instability (PBTI) on NMOS transistors.

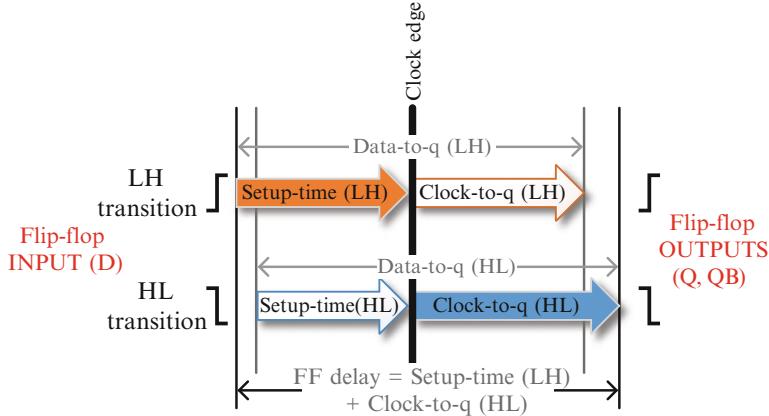


Fig. 1 Different flip-flop timing parameters. The correct functionality is guaranteed by considering the flip-flop delay as illustrated

$$\{U_{HL}, D_{CQ_{HL}}, D_{DQ_{HL}}\} \quad \text{for HL transition.}$$

Flip-flop delay should be defined such that the correct functionality of the flip-flop will be guaranteed, regardless of the transition. Therefore, we define the flip-flop delay as the *summation of the worst setup-time and the worst clock-to-q of both transitions* as shown in Fig. 1.

$$\text{delay} = \max\{U_{LH}, U_{HL}\} + \max\{D_{CQ_{LH}}, D_{CQ_{HL}}\}. \quad (1)$$

This guarantees that in both transitions the input signal is correctly captured and propagated to the flip-flop output.

2.2 Runtime Variation Impacts on Flip-Flops

Several parameters such as supply voltage, workload, and temperature affect the performance of flip-flops in a circuit. Parameters such as temperature and supply voltage affect all the transistors of a flip-flop in the same way, whereas the impact of the input SP is different for the transistors of a flip-flop [23]. This results in an asymmetric aging of transistors according to their stress duty cycles. Therefore, the delay degradation of internal LH and HL paths inside an aged flip-flop depends on the input SP [24].

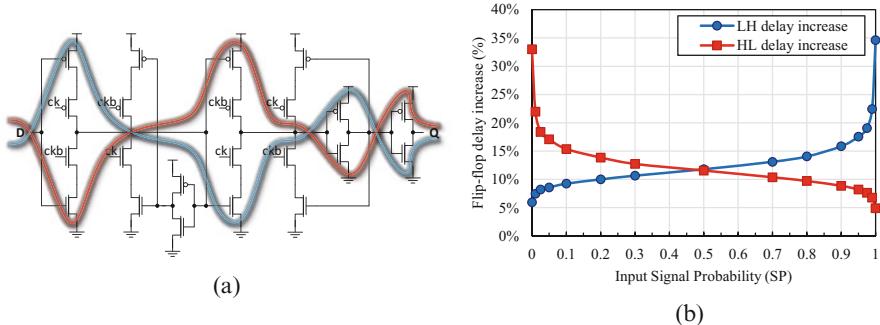


Fig. 2 Separate internal LH (red)/HL (blue) paths of a C2MOS flip-flop [31] (a), and delay of internal LH/HL paths of an aged C2MOS flip-flop after 5 years (optimized for Power Delay Product (PDP) in the fresh state) for different input SPs (b)

In the C2MOS flip-flop² depicted in Fig. 2a, the internal LH and HL paths consist of two separate groups of transistors, which makes the aging of these two paths independent according to the input SP. Figure 2b illustrates the delay of LH and HL transitions of an aged C2MOS flip-flop [31] for different input SPs. When the flip-flop is aged under input $SP = 0.0$ (SP0), the worst delay degradation happens on the flip-flop HL path; however, the delay of the flip-flop LH path is only slightly affected. On the other hand, an aging under input $SP = 1.0$ (SP1) greatly degrades the delay of the flip-flop LH path while slightly affecting the delay of the flip-flop HL path. For moderate aging condition, i.e. $0.1 < SP < 0.9$, the delay degradation of both LH and HL paths is moderate. The reason is that under SP0 and SP1 conditions, *Static BTI* (S-BTI) asymmetrically alters the threshold voltages leading to unbalanced aging of LH and HL paths of the flip-flop as the stress duty cycle of some transistors is 1.0, i.e., always under BTI stress. However, in moderate aging condition, the transistors can partially recover as the stress duty cycle is less than 1.0.

The impact of supply voltage fluctuation on the flip-flops of a circuit depends on the workload variation and dynamic power consumption of the circuit. Therefore, each flip-flop may experience a specific amount of voltage-drop. A voltage-drop causes performance degradation of the flip-flops, which is typically larger than the degradation of simpler combinational gates in the standard cell library. Figure 3 compares the impact of a voltage-drop up to 10% on the delay of an aged flip-flop and an aged inverter. Compared to a no-voltage-drop condition, the delay of the flip-flop increases by 23.6% whereas the delay of the inverter is increased by 15%.

Moreover, the flip-flops of a circuit generally experience higher amount of voltage-drop compared to combinational gates [37]. As a result of temporally

²A C2MOS flip-flop design is a master–slave flip-flop built of two connected C2MOS latches. It is one of the commonly used flip-flops in modern processor designs [31].

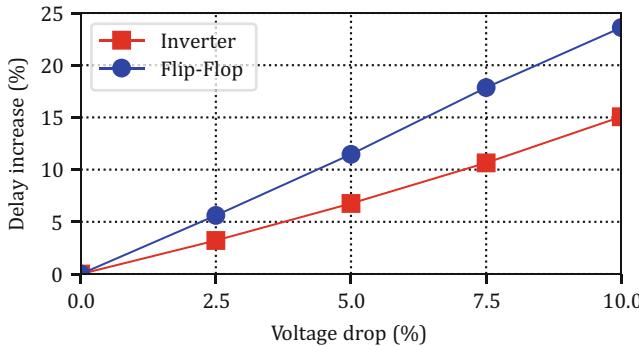
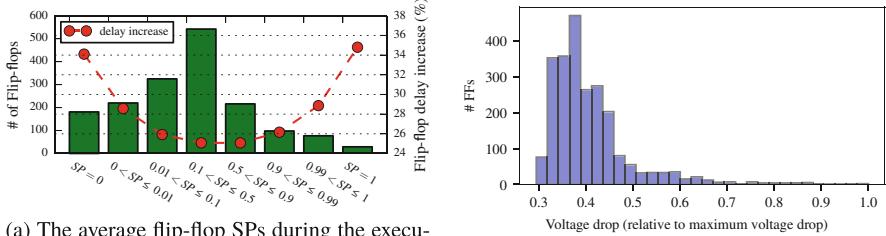


Fig. 3 Comparison between the voltage-drop induced delay degradation of a flip-flop and an inverter, which are aged under same condition (Aging under SP1 for 5 years)

localized switching of flip-flops at the positive (or negative) edge of clock signal, the instantaneous current drawn from PDN at the synchronized clock edge is comparatively high. This leads to high voltage-drop at the clock edge, when the flip-flops are processing their input signals. This peak current consumption is damped over the rest of the clock period, when the combinational cells are active. Therefore, in this work we focus on dealing with the impact of voltage fluctuation on the flip-flops.

Temporal and spatial temperature variations can also affect the circuit performance. The temporal temperature change could be rather high and has been the subject of research since it affects the reliability of the VLSI circuits. It is demonstrated in [17] that the circuit performance can be changed by up to 10% for 110 °C temperature variation. Therefore, in order to meet the reliability constraints, the circuit timing should be adjusted according to the worst temperature corner, which is typically at high temperature. On-chip spatial temperature gradient puts different stress on circuit components across a chip. The amount of on-chip spatial temperature difference (only on cores) based on simulation [3, 7], sensor measurements [33], and thermal camera [3] is reported to be up to $\sim 30^\circ\text{C}$. Since the delay change is approximately 4% for every 40 °C [17, 29], the overall difference between the delay degradation of core flip-flops due to such spatial temperature gradient is expected to be less than 3%, and hence, much smaller compared to voltage-drop variation [11].

The combined impact of voltage-drop and aging significantly degrades the performance of flip-flops. As an example, the delay of a fresh flip-flop optimized with balanced HL/LH delay increases from 98.5 ps to 165.7 ps due to the combined impact of voltage-drop (10%) and S-BTI (5-years under SPO). This is equivalent to 68% delay increase. If such a flip-flop is in a critical path of the circuit, a large timing guardband is required for timing closure considering the reliability constraints. Therefore, it is necessary to find such flip-flops at design-time and optimize them for operating under such conditions.



(a) The average flip-flop SPs during the execution of some MiBench workloads on Leon3, and the corresponding delay degradation in 5 years. In order to be fair in the analysis, the flip-flops which are not exercised by the workloads are excluded from this analysis.

(b) The distribution of voltage-drop on the Leon3 flip-flops. The voltage-drop values are normalized to the maximum voltage-drop across the processor.

Fig. 4 (a) Input signal probabilities and (b) voltage-drop analysis of Leon3 flip-flops executing MiBench Workloads

2.3 Significance of Flip-Flops in Circuit Reliability

In a properly designed circuit, the timing of circuit paths are balanced during the synthesis process. Therefore, many flip-flops are *timing-critical* as they lie on the circuit critical paths. Studies [12, 37] have shown that in VLSI circuits, some flip-flops are under severe static BTI leading to a large timing degradation over time. Furthermore, the impact of voltage-drop on flip-flops could be very high as a result of localized power consumption at a specific time (e.g. positive clock edge) or at a specific location on the circuit layout.

The large impact of S-BTI and voltage-drop on flip-flops has a significant impact on the reliability of a circuit when such flip-flops are timing-critical. In order to investigate the likelihood of having such a scenario in a typical digital design, we use the flow presented in Sect. 4 to extract the voltage-drop and the aging of the Leon3 flip-flops by executing six MiBench workloads [15] namely stringsearch, qsort, basicmath, bitcount, fft, and crc32 on Leon3 processor [10]. In order to be fair, we excluded the flip-flops belonging to the parts which are not exercised by the employed workloads such as interrupt handler, timers, and UART controller. The synthesized netlist of the Leon3 processor has 2352 flip-flops, but the results demonstrated in this section contain only 1686 flip-flops belonging to the parts which are exercised by all employed workloads.

Figure 4a demonstrates the input SP distribution of the aforementioned 1686 flip-flops. The results show that 181 flip-flops always experience input SP0, whereas 29 flip-flops are under input SP1. Our analysis shows that the flip-flops with such behavior typically belong to either the error checking and exception handling registers or higher bits of address registers which are constant due to temporal and spatial locality of the executed instructions. Besides, the SP of a considerable number of flip-flops is very close to either 0.0 or 1.0. Please note that the results reported in Fig. 4a are the average of six employed workloads, and hence, the flip-

flops with $SP = 0$ or $SP = 1$ have such SP across all executed workloads. Similar experiment has been carried out in [18] to study the impact of workload in real systems, which shows that some flip-flops are always under S-BTI across different workloads.

Figure 4b shows the distribution of the maximum voltage-drop impacting the flip-flops of Leon3 processor compared to the peak voltage-drop across all the executed workloads. Please note that it is necessary to consider the maximum voltage-drop over the execution of all workloads, because it eventually impacts the flip-flop characteristics. A significant portion of flip-flops experience on average 41% of the maximum amount of voltage-drop; however, there are flip-flops at the right side tail of the distribution which experience large voltage-drop comparable to the maximum voltage-drop in the circuit.

According to the observations in Fig. 4, there are flip-flops experiencing both S-BTI and high voltage-drop which leads to high-degradation. If such flip-flops are on a critical path of the processor (i.e. timing-critical flip-flops), the degradation of the flip-flops should be reflected in the timing guardband of the circuit. Timing-critical flip-flops can be categorized into different groups based on the impact of voltage-drop and aging as follows:

- low voltage-drop and low aging,
- low voltage-drop but S-BTI aging ($SP0/SP1$)*
- high voltage-drop but typical aging*
- high voltage-drop and S-BTI aging ($SP0/SP1$)*

Therefore, we propose to generate flip-flops specifically optimized for such high-degradation conditions (marked by *) and add them to the standard cell library. Using the proposed flow in Sect. 4, we determine such high-degradation and timing-critical flip-flops and replace them with the optimized versions to improve the timing and reliability of the circuit.

3 Reliability-Aware Flip-Flop Design

In a typical reliability-aware circuit design, one should consider the delay of the elements under variation impacts to ensure the correct functionality of the circuit during the expected lifetime. Therefore, higher delay degradation of timing-critical flip-flops imposes a large timing guardband. In our proposed methodology, we create optimized versions of the flip-flops for different stress conditions based on aging and voltage fluctuations, and use these optimized versions only when a flip-flop is timing-critical and subject to such stress conditions to avoid unnecessary over design. This means that in the cell library, we add the following resilient versions of the flip-flops:

- Aging-resilient flip-flops, optimized for different aging corners ($SP0$ and $SP1$),

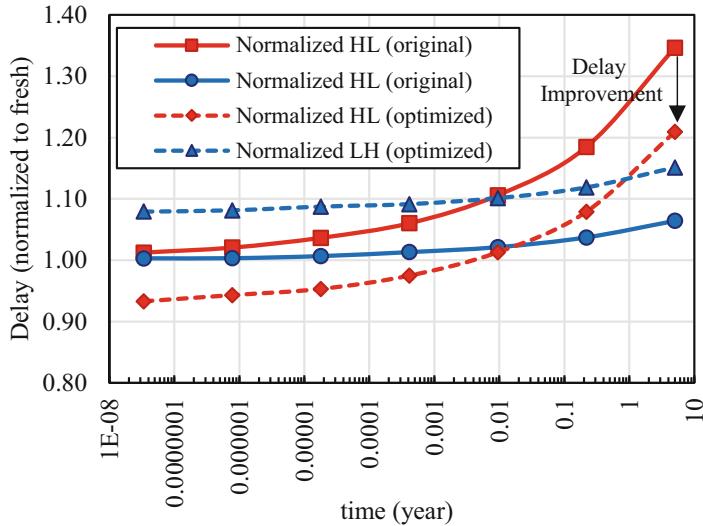


Fig. 5 Delay of a C2MOS flip-flop which is aged under $SP = 0$ over 5 years for LH/HL transitions, compared to the flip-flop optimized for $SP = 0$ showing how the unbalanced aging of internal LH/HL paths worsens the degradation in original flip-flop

- Voltage-drop resilient flip-flops, optimized to have lower performance degradation under voltage fluctuation,
- Aging and voltage-drop resilient flip-flops.

3.1 Aging-Resilient Flip-Flop Design

When the fresh delays of internal paths of a flip-flop (i.e., LH and HL paths) are designed to be similar (depicted as solid lines in Fig. 5), the internal path with higher degradation rate eventually becomes dominant and determines the total delay of the flip-flop. In this case, a significant aging in flip-flop characteristics is observed over time (corresponding to the internal path with higher degradation). On the other hand, if the internal path with higher degradation rate is initially faster (by design) than the internal path with lower degradation rate, the dominant internal path would be the slower one, and hence the higher degradation rate of the faster internal path is masked. Consequently, the overall aging of the flip-flop would be rather small. The delay of the optimized flip-flop, shown in Fig. 5 by dashed lines, exhibits such characteristics. The post-aging delay of the optimized flip-flop would increase by ~ 10 ps, which is much lower than ~ 40 ps increase in the

delay of the original flip-flop. We exploit this method for designing aging-resilient flip-flops.

In order to decrease the overall BTI-induced aging inflicted to a flip-flop, our proposed method balances the delay of internal HL and LH paths of the flip-flop for *post-aging state of the flip-flop*, by resizing the transistors of internal HL and LH paths. In other words, the proposed method increases the fresh delay ($t = 0$) of the flip-flop internal path which has lower degradation rate in order to compensate the overall degradation of the flip-flop after aging. Although the fresh delay of the optimized flip-flop might be slightly larger compared to the fresh delay of the original flip-flop, the overall delay of the optimized flip-flop considering the aging-induced timing margin would be smaller than those of the original flip-flop since the aging rate is much smaller.

Please note that this method reduces the degradation for a given SP, but inevitably worsens the aging at the other corners of SP. For example, if we optimize the flip-flop for SP0, the degradation would be much higher if the optimized flip-flop operates at SP1. Nevertheless, these flip-flops under S-BTI will not operate at other SP corners, because their SP is determined by the circuit structure and functionality. Therefore, we only optimize for the given SP corner. This means that we intentionally sacrifice other corners, which never occur due to the specific functionality of the circuit, to gain a larger improvement.

3.2 Voltage-Drop Resilient Flip-Flop Design

Other than aging, which affects each flip-flop transistor based on the input signal probability, a drop in the supply voltage of the flip-flop slows down all flip-flop transistors in the same way. However, a slight upsizing of specific transistors can compensate the degradation in the flip-flop timing. Therefore, we evaluate the delay of the flip-flop when operating under the impact of voltage-drop, and optimize the flip-flop with the goal of improving the delay. Consequently, the optimized flip-flop would have better timing at the cost of higher power consumption.

3.3 Aging and Voltage-Drop Resilient Flip-Flop Design

The degradation in the flip-flop timing due to both S-BTI and voltage-drop is very large. Such timing degradation may not be effectively compensated by resizing the transistors within a flip-flop area without upsizing the entire flip-flop. Therefore, in addition to targeting for better timing under the impact of the aging and voltage-drop, we allow the optimization algorithm to increase the area of the flip-flop by a small percentage. Please note that an extra Engineering Change Order (ECO) might

be needed to replace the original flip-flop with the optimized version in this case. However, since there exist only a few flip-flops under such degradation it would not be an issue to perform an ECO on placement.

3.4 Problem Formulation for Flip-Flop Resiliency Optimization

The delay of a flip-flop under a specific working condition (including temperature, voltage, and input SP) can be presented as a function of the transistors' widths:

$$\text{delay} = f(W), \quad W = [w_i], \quad (2)$$

where $[w_i]$ is a vector containing the width of flip-flop transistors. Here, *delay* is the delay (Data-to-q) of the flip-flop, according to Eq.(1), under variation impact, which could be S-BTI stress, voltage-drop, or both depending on the optimization approach.

The delay function f is a complicated function of transistors' widths. Our experimental results for flip-flops with different sizing show that f cannot be presented with any general linear function. Therefore, we use Sequential Quadratic Programming (SQP) which is a non-linear programming technique [19]. In SQP, the problem is converted into quadratic sub-problems and solved in order to find a better sizing in each iteration. For this purpose, we follow an iterative approach in order to minimize the delay of Eq.(2). Given an initial sizing, the delay function f is approximated with a quadratic function:

$$f(W) \approx f(W_0) + \nabla f(W_0)^T \cdot (W - W_0) + \frac{1}{2}(W - W_0)^T \cdot H_f(W_0) \cdot (W - W_0), \quad (3)$$

where $\nabla f(W)$ and $H_f(W)$ are the gradient and the Hessian of the delay function f , respectively. Minimizing the quadratic approximation of Eq.(3), with respect to some constraints, which will be discussed later in this section, yields an optimized transistor sizing. Thereafter, the obtained sizing is used as the initial sizing, and a new iteration is launched. This cycle continues until the optimization reaches the required precision, i.e. the difference between the optimized delays of two consecutive iterations becomes smaller than a predefined threshold ϵ_{delay} . Therefore, the solver continues by checking the precision of the resulting delay:

$$|\text{delay}_{i-1} - \text{delay}_i| < \epsilon_{\text{delay}} \quad (4)$$

where delay_i represents the delay of i th iteration.

Another reason to use the quadratic approximation is that the optimum result of a linear problem always lies on the boundaries, while the optimum result of a

Table 1 Flip-flop optimization method summary

Parameters	$W = (w_1, \dots, w_n)$	
Initial guess	W_0 = optimized W for PDP (fresh)	
Constraints	$w_i \geq w_{\min}$ $\sum_1^n w_i \leq (1 + \lambda) \sum W_0$ $\text{power}(W) \leq (1 + \beta) \text{power}(W_0)$	
Target	minimize: delay = $f(W)$	
Constants	w_{\min} Minimum size λ Acceptable excessive area β Acceptable excessive leakage	

quadratic problem can be any point within the boundaries as well as the boundaries themselves. In Sect. 5, we demonstrate that the optimum result does not necessarily lie on the boundaries, and hence a non-linear programming technique is needed to find a better result. Table 1 summarizes the optimization problem.

Several constraints are applied to the optimization problem, relating to transistors size, flip-flop area, and leakage. The first constraint shown in Table 1 limits the minimum size of transistors. The second constraint limits the area of the optimized flip-flop. In case of optimizing for S-BTI or voltage-drop, we consider $\lambda = 0$ to keep the flip-flop area within the area of the original flip-flop which also facilitates keeping the aspect ratio almost equal to the aspect ratio of the original flip-flop. This way, the optimized flip-flop can easily replace the original flip-flop without any layout modifications at the circuit-level. This is achieved by limiting the summation of transistor widths w_i . However, for flip-flops under S-BTI and voltage-drop, we assume a $\lambda > 0$ value to compensate the delay degradation better. The third constraint sets an upper limit for the excessive leakage of the flip-flop by parameter β . This constraint is applied to the optimization problem to limit the leakage power of the optimized flip-flops within an acceptable range. The initial guess of optimization W_0 is the optimum sizing for minimum PDP in the fresh state.

3.5 Reliability-Aware Flip-Flop Optimization Flow

Figure 6 presents our proposed reliability-aware optimization flow. For a given input SP, the SP of all transistors are once calculated using SPICE simulations. Afterwards, based on the extracted SP for transistors and the operating corner of the flip-flop (temperature, supply voltage, etc.), the BTI-induced threshold voltage shifts of all transistors (ΔV_{th}) are obtained. Then, the ΔV_{th} values are back-annotated into the original flip-flop SPICE netlist, and the SPICE netlist of aged flip-flop is generated.

In each SQP iteration, the quadratic sub-problems are created and solved to generate further improved flip-flop sizing. Subsequently, the new sizing is back-annotated into the aged flip-flop netlist extracted before, and a new aged flip-flop with the given sizing is generated. Then, Cadence Virtuoso Liberate [6] is used

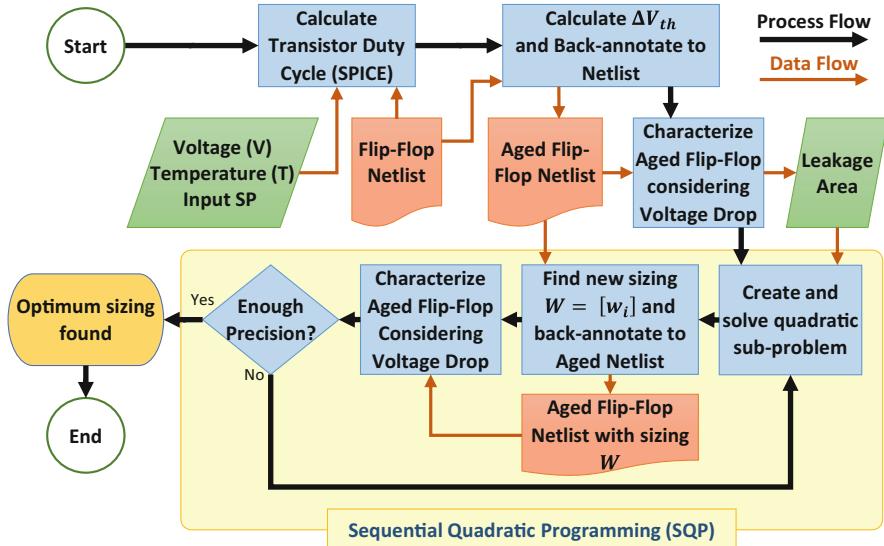


Fig. 6 Overall flow to find the optimum flip-flop sizing for under S-BTI stress and voltage-drop at a specific working corner (voltage, temperature)

to characterize the new flip-flop and extract its delay and power consumption. When the improvement is small enough and the condition in Eq. (4) is met, the SQP method terminates and returns the last sizing as the optimum solution for the problem.

As the process is executed at a specific supply voltage (V_{dd}), it can inherently be used to optimize for a voltage-drop as well, when the given supply voltage includes the impact of the voltage-drop. We can also create voltage-drop resilient version of a flip-flop for typical aging, by considering input SP of 0.5. Therefore, we execute the flow presented in Fig. 6 for these conditions in order to create variation-resilient versions of the flip-flop, assuming a supply voltage of V_{dd} and a maximum voltage-drop of $R\%$:

	Supply voltage (V)	Aging condition
Aging	V_{dd}	S-BTI (SP0, SP1)
Voltage-drop	$(1 - \frac{R}{100})V_{dd}$	Typical aging (SP = 0.5)
Aging and voltage-drop	$(1 - \frac{R}{100})V_{dd}$	S-BTI (SP0, SP1)

After optimization process, it is necessary to re-characterize the flip-flops for different supply voltage ($((1 - \frac{R}{100})V_{dd}$ to V_{dd}) and aging conditions ($0 \leq SP \leq 1$). The characterization results are then used to obtain overall circuit timing under supply voltage fluctuation and aging impacts.

4 Selective Flip-Flop Optimization

This section explains how the optimized flip-flops in Sect. 3 can be employed to improve the reliability of a circuit. The idea is to find the flip-flops affected by S-BTI and/or voltage-drop impacts which are also influential on circuit reliability, i.e. the timing-critical flip-flops, and replace them with the optimized versions. The reason for this *selective flip-flop optimization* is that the reliability-aware flip-flop optimization is costly in terms of leakage overhead per flip-flop. Therefore, flip-flop replacement should be done only for the timing-critical flip-flops which experience S-BTI and/or large voltage-drop to be cost-effective. Since they constitute a small subset of the all flip-flops in the design, the proposed method is able to reduce the overall timing guardband in a cost-effective way.

The overall flow of the proposed selective flip-flop optimization methodology is presented in Fig. 7. The flow uses the results of the Synthesis and Place & Route steps of a VLSI design flow and is composed of (I) *Aging and Voltage-Drop Analysis* and (II) *Selective Flip-flop Replacement* steps. The optimization flow updates the gate-level netlist and the circuit layout to improve the reliability of the circuit under voltage-drop and aging impacts. The outputs of the optimization method can be further used in the rest of the VLSI design flow. Therefore, the proposed method is transparent to the VLSI design flow and can be easily integrated into it.

4.1 Aging and Voltage-Drop Analysis

In this step, the results of Synthesis and Place & Route steps of the VLSI design flow are used to discover the flip-flops which are *aging-critical*, *voltage-drop critical*, and *timing-critical*.

Aging-critical flip-flops are those flip-flops which experience large impact of aging, i.e. flip-flops under S-BTI. To find the aging-critical flip-flops we need to extract the SP of the flip-flops. Therefore, we perform a gate-level simulation running some representative workloads. The representative workloads are pieces of workloads which are typically executed on the circuit. The result of the gate-level simulation is the Voltage Change Dump (VCD) of all nets inside the circuit. Based on this information we can collect SP of all flip-flops and determine the aging-critical flip-flops.

Dynamic power profiles of circuit components can be extracted from the VCD reports. We estimate the dynamic voltage-drop in the circuit based on the power profiles and the layout and packaging of the circuit. This accounts for the resistive and inductive components of the voltage fluctuation. We generate a *voltage-drop map* of the circuit by evaluating the *maximum voltage-drop* of each cell (gates, flip-flops, etc.) over the time and over different workloads. As a result, we find the maximum amount of voltage-drop that each flip-flop experiences over time.

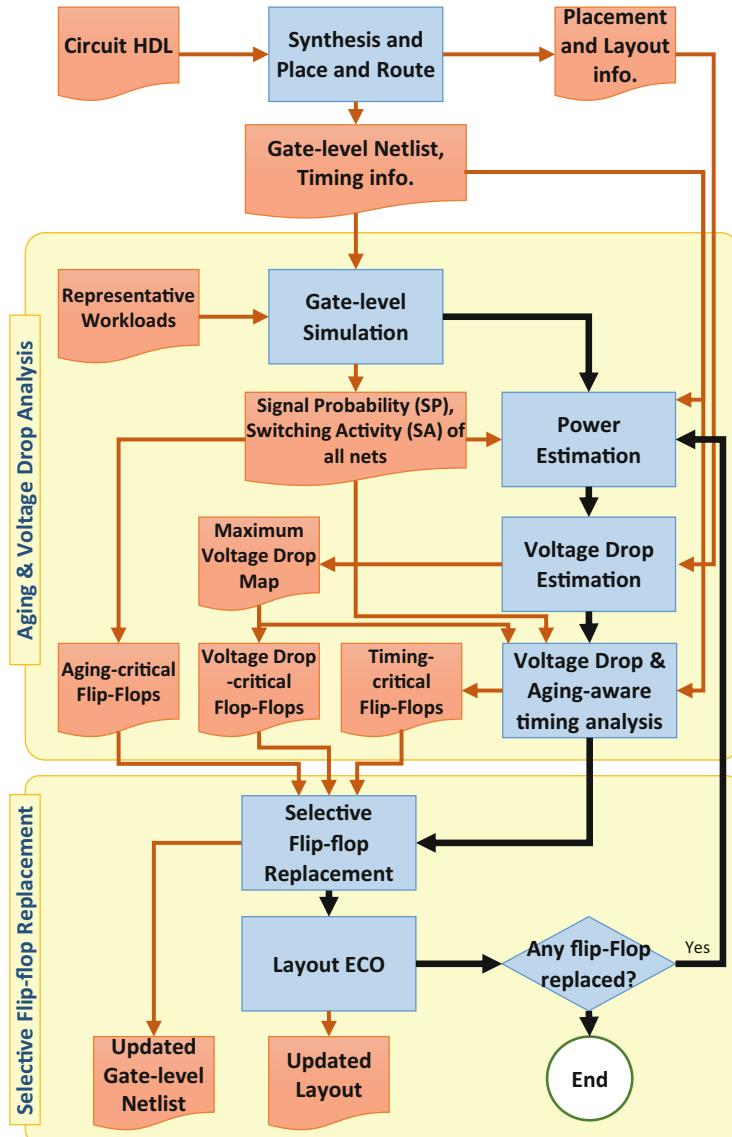


Fig. 7 Circuit optimization flow using the proposed selective flip-flop optimization method

Accordingly, the flip-flops which experience a large amount of voltage-drop are extracted.

Furthermore, the gate-level simulation results are used to perform a *voltage-drop and aging-aware timing analysis* which obtains the delay of circuit paths under variability impacts. We extended the aging-aware timing analysis in [8] by

considering voltage-drop related information. This is done by characterizing the cells at two different supply voltages: the nominal V_{dd} and the supply voltage considering the maximum drop $(1 - \frac{R}{100})V_{dd}$. Then, for each gate/flip-flop in the gate-level netlist, based on the amount of voltage-drop on the gate, we perform a linear interpolation among the standard cell library entries for two supply voltages and find the corresponding timing information. The linear interpolation is a valid method under the assumption of limited change in the supply voltage, as shown in Fig. 3. For a more aggressive voltage fluctuation, it could be necessary to characterize the standard cell libraries for a few intermediate supply voltage values and employ a PCHIP method. Accordingly, we find the timing-critical flip-flops, which are parts of the critical and near-critical paths of the circuit considering the impact of variations.

4.2 Selective Flip-Flop Replacement

In the selective flip-flop replacement step, we replace the flip-flops which are timing-critical, aging-critical, or voltage-drop-critical with their optimized counterparts for such aging and/or voltage-drop conditions. Although a small portion of the flip-flops are replaced during the flip-flop replacement process, the circuit layout, timing, and power properties change since the replaced flip-flops are timing-critical and may have different area and power characteristics. Therefore, the proposed flip-flop replacement is an iterative process which replaces a number of flip-flops with the optimized versions in each iteration. The iterative process continues until no flip-flop needs to be replaced by an optimized version anymore.

In iteration i of the method, we assume that the circuit delay is D^i based on timing analysis results, and d_j^i is the maximum delay of the paths terminating at flip-flop j (including the delay of the flip-flop as well). Therefore, in each iteration:

1. We choose the timing-critical flip-flops with a timing slack value of less than $k\%$ of the circuit delay, i.e. when:

$$d_j^i \geq \left(1 - \frac{k}{100}\right) D^i, \quad j : \text{index of flip-flops.}$$

2. Among these flip-flops, those which are also included in the aging-critical and/or voltage-drop-critical flip-flops, are replaced with the optimized versions.
3. A trial voltage-drop and aging-aware timing analysis is performed and the circuit delay (D^{i++}) is determined considering the replaced flip-flops.
4. We keep the optimized flip-flops only when the corresponding path delay of the flip-flops *before optimization* is larger than a percentage of the evaluated circuit delay (D^{i++}):

$$d_j^i > r \times D^{i++} \implies FF_j \rightarrow FF_{j,opt}. \quad (5)$$

The rest of the updated flip-flops in this iteration are rolled back to the original versions. Please note that we also consider a ratio $r < 1$ into Eq. (5) to compensate for the calculation errors due to simulation.

5. The layout and gate-level netlist of the circuit are updated. The layout is only updated if a cell with larger area is used (particularly applicable to the flip-flops under both aging and voltage-drop as explained in Sect. 3).
6. In case any flip-flop is replaced by an optimized version during this iteration, we need to start a new iteration because the timing and power specification of the circuit are modified. This is done by re-executing the aging and voltage-drop analysis, as explained in Sect. 4.1. The gate-level simulation, which is a time consuming process, does not need to be repeated as its results are not affected by the flip-flop replacement.

The above flow replaces minimum number of flip-flops with the optimized versions and impose minimum amount of overhead to the circuit. In our simulations the flow is terminated within a few iterations, since the changes in the circuit layout, power, and timing are not extensive.

5 Results and Discussions

In this section, we evaluate the efficiency of the proposed selective flip-flop optimization based on simulation results.

5.1 Simulation Setup

We applied the method to several flip-flop topologies, namely C2MOS latch, Dynamic/Static Single Transistor Clocked latch (DSTC/SSTC), and Semi-Dynamic flip-flop (SDFF) [31]. The flip-flops are implemented using 45 nm Bulk CMOS Predictive Technology Model (PTM) transistors [39]. All flip-flops are initially optimized for the minimum PDP in the fresh state (original design). The aging parameters of the model proposed in [4] are tuned so that the post-aging delay of a Fan-Out 4 (FO4) inverter increases by 10% at $SP = 0.5$ over 5 years. For delay and leakage measurements, the output load of flip-flops is set to FO4, and the cells are characterized at room temperature and at different supply voltages, ranging from 80 to 100% of the nominal supply voltage of the technology node.

We used Leon3 processor as a case study for our proposed method. We used Nangate 45 nm open cell library for combinational logic, and aging assumptions are the same as described at the beginning of this section. The processor is synthesized using Synopsys Design Compiler and placement and routing is done using Cadence EDI [5].

We executed various MiBench workloads on the synthesized Leon3 processor and extracted the VCD files. Based on the VCD files, the SP of each node of the synthesized circuit is calculated and the power consumption of the gates and flip-flops is calculated using Synopsys Power Compiler. The voltage-drop map of the processor is also extracted using VoltSpot tool [38], which is able to extract the voltage-drop caused by both resistive and inductive components.

Please note that the proposed technique is not restricted to a specific working condition or flip-flop topology. We proceed with presenting detailed results and analysis for a C2MOS flip-flop. Then, we discuss the results for other types of flip-flops concisely. Afterwards, the dependency of the improvement achieved by the proposed method to the excessive leakage will be investigated. At the end of this section, the impact of using optimized flip-flops on a Leon3 processor lifetime will be demonstrated.

5.2 Detailed Optimization Results of C2MOS Flip-Flop

We apply the proposed optimization flow presented in Sect. 3.5 (see Fig. 6) to C2MOS flip-flop design to create optimized flip-flops for aging and voltage-drop resilience. In order to create the aging-resilient versions of the C2MOS flip-flop, we let the optimizer to consider designs with up to 25% more leakage compared to the original flip-flop by setting the coefficient β in Table 1 to 0.25. At this point, we limit the area of the flip-flop to the area of the original flip-flop, i.e. $\lambda = 0$. Please note that the total overhead of the leakage power for the entire circuit would be negligible since the number of optimized flip-flops in the design would be limited. For example, if according to Sect. 2.3, 12.45% of flip-flops are working under S-BTI, and the leakage overhead of an optimized flip-flop would be less than 25%, the leakage overhead imposed on the flip-flops would be *at most* 3.11% (much less overhead when considering the entire processor design). The aging and voltage-drop resilient version of the C2MOS flip-flop can be created by assuming an extra area up to 20% and more leakage overhead. For this, we assume $\lambda = 0.2$, $\beta = 1$. Using the extra area, the optimizer is able to find a better design for those flip-flops which are timing-critical and are under large impact of aging and voltage-drop. Since these flip-flops are very rare, but have significant impact on the overall processor lifetime and reliability, it is effective to spend more area for large reliability and lifetime gains.

Table 2 compares the characteristics of an original and optimized C2MOS flip-flop (such as setup-time (U), clock-to-q (D_{CQ}), data-to-q (D_{DQ}), delay, and leakage) in three different optimization scenarios:

- Scenario 1 *post-agging PDP*, optimized for PDP in post-agging.
- Scenario 2 The proposed method (optimized for aging), in which the flip-flop is optimized for aging resiliency, by minimizing its delay for post-

Table 2 C2MOS flip-flop characteristics for (1) Original flip-flop (Optimized for PDP in the fresh state), (2) Optimized flip-flop for PDP in post-aging [1], and optimized by the proposed method for (3) only aging, and (4) for aging + vdrop

Parameters ^c	Post-aging PDP (similar to [1])								Proposed method			
	Original (optimized for fresh PDP)				Scenario 1				Optimized for aging		Optimized for aging + vdrop	
	Fresh	Aged	Aged+vdrop ^b	Fresh	Aged	Aged+vdrop	Fresh	Aged	Aged+vdrop	Fresh	Aged	Aged+vdrop
U_{LH} (ps)	20.2	22.6	29.2	25.6	30.0	32.9	23.0	24.6	30.3	30.0	30.0	37.7
$D_{CQ,LH}$ (ps)	78.3	101.8	123.3	91.8	97.6	117.5	83.3	88.8	103.9	72.5	77.5	92.2
$D_{DQ,LH}$ (ps)	98.5	124.4	152.5	117.4	125.6	150.4	106.3	113.4	134.2	102.5	107.6	129.9
U_{HL} (ps)	16.6	30.8	42.4	13.7	28.2	39.6	16.0	30.6	40.6	11.1	24.6	30.6
$D_{CQ,HL}$ (ps)	78.1	100.5	121.0	82.9	97.9	117.2	75.9	88.5	106.4	65.8	75.6	91.7
$D_{DQ,HL}$ (ps)	94.7	131.3	163.4	96.6	126.1	156.8	91.9	119.1	147.0	76.9	100.2	122.3
Delay (ps) (Sect. 2.1)	98.5	132.6	165.7	117.4	126.1	157.1	106.3	119.4	147.0	102.5	107.6	129.9
Leakage (nW)	44.3	30.7	15.5	42.0	27.9	14.1	46.4	31.4	15.8	67.8	46.1	23.1
PDP	4368	4074	2573	4936	3525	2215	4928	3744	2318	6952	4963	3000
Delay degradation, Eq. (6) ^a	–	35%	68%	–	28%	60%	–	21%	49%	–	9.2%	32%
Excessive leakage	–	–	–	–5.2%	–	4.7%	–	–	53%	–	–	–

The results are reported for “fresh”, “aged,” and “aged + vdrop” states and under $SP0^d$ aging

^aThe reference for calculating the delay degradation is 98.5 ps (Original, fresh flip-flop)

^bMeasurements are done under 10% delay degradation assumption (Sect. 5.1) and 10% voltage-drop

^cDynamic power is not reported because it is irrelevant for flip-flops under S-BTI as these flip-flops do not change state frequently

^dOptimization results for SP1 are much better. For example, the delay degradation of the proposed method for aging is only 11% (for SP0 it is 21%)

aging. The acceptable excessive area and leakage are 0% and 25%, respectively ($\beta = 0.25, \lambda = 0$).

- Scenario 3 The proposed method (optimized for aging + vdrop), in which the flip-flop is optimized for aging and voltage-drop resiliency, by minimizing its delay for post-aging and under voltage-drop impact. The acceptable excessive area and leakage are 20 and 100%, respectively ($\beta = 1, \lambda = 0.2$).

The optimization results in Table 2 are reported for “fresh” state (no aging or voltage-drop), for “aged” state (under S-BTI aging SP0 for 5 years), and for “aging + vdrop” state (when the flip-flop is aged under S-BTI for 5 years, and when the supply voltage is dropped by 10%). Setup-time, clock-to-q, and data-to-q values are presented for LH/HL transitions and the delay is calculated according to Sect. 2.1. The delay degradation is the *relative post-aging delay increase of a design compared to the fresh delay of the original design* (marked as bold in the table):

$$\text{delay degradation} = \frac{\text{delay}_{\text{opt.,aged}} - \text{delay}_{\text{orig.,fresh}}}{\text{delay}_{\text{orig.,fresh}}}. \quad (6)$$

Since the optimized flip-flop will replace the corresponding flip-flop in the design, the delay degradation is compared to the fresh delay of the original flip-flop in order to give a better understanding of how close the aged delay of the optimized flip-flop is to the fresh delay of the original design.

Basically, scenario 1 is similar to the methods proposed in many flip-flop optimization methods such as [1, 13] in the sense that they consider a multiplication of energy and delay (e.g., the PDP or the Energy Delay Product (EDP)) as the optimization target. Scenario 1 is able to effectively reduce the PDP by increasing the delay and reducing the leakage, but this may result in an unacceptable timing for S-BTI corners. Table 2 shows that due to not considering the flip-flop delay as the optimization target, the PDP methods cannot find the optimum aging-resilient sizing for S-BTI corners.

As presented, for the original flip-flop, the fresh delay of LH and HL paths is almost identical (see $D_{DQ,LH}$ and $D_{DQ,HL}$), but after aging HL path is much slower than LH path. This leads to 35% delay degradation due to only aging and about 68% when aging and voltage-drop affect the flip-flop. When this flip-flop is optimized for scenario 1, the delay is not reduced well enough because the main concern is PDP not delay. On the other hand, in scenario 2 (proposed method, only for aging), the optimizer alters the sizing to equalize the post-aging delay of the LH/HL paths to achieve the smallest possible post-aging delay with respect to the constraints (119.4 ps). In this case, the post-aging delay is increased by 21% compared to the fresh delay of the original flip-flop. Also the leakage overhead is limited to 4.7%. Since the flip-flop operates in S-BTI zone, the switching rate of the flip-flop is very small. This means that its dynamic power is almost negligible. Therefore, the total power in of flip-flops under S-BTI is determined by the leakage power.

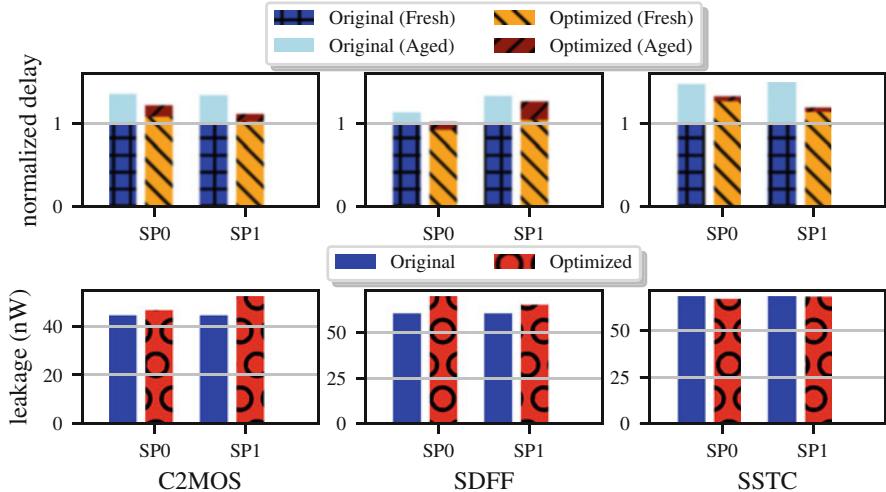


Fig. 8 Performance of the original flip-flop vs. the flip-flop optimized by the proposed method at SP0 and SP1, before and after aging (5 years)

Even though scenario 2's design is much better for flip-flops which are only under the aging impact compared to the original and the state-of-the-art [1] flip-flop designs, the impact of 10% voltage-drop is significant on the delay, i.e. 49% delay degradation. The flip-flop optimization results for scenario 3 show that such flip-flops are more resilient against both aging and voltage-drop impacts. These flip-flops consume about 53% more leakage; however, the delay degradation is only 32% under both aging and voltage-drop. Please note that the number of flip-flops under such condition is very small. Therefore, using flip-flops optimized by scenario 3 has negligible impact on the overall processor power consumption.

5.3 Optimization Results for Other Flip-Flops

Figure 8 provides the optimization results for a set of representative flip-flops. It compares the delay and the leakage of the original and optimized flip-flops, for both fresh and post-aging states. All delay values are normalized to the fresh delay of the corresponding original flip-flops (which are 114.8 ps for C2MOS, 28.5 ps for SDFF, and 71.0 ps for SSTC).

For C2MOS flip-flop, the proposed method reduces the delay degradation in Eq. (6) to 21%, while the delay degradation of the original design is 35% (14% improvement). This flip-flop has a symmetric structure, which means it can have balanced timing for LH/HL transitions (shown in Fig. 2b), while some flip-flop topologies such as SDFF, always have an unbalanced timing for LH/HL transitions due to their internal structure. For example, in an SDFF, the delay of HL transition is

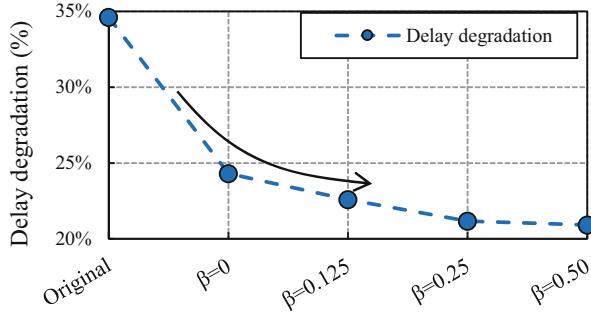


Fig. 9 Delay of C2MOS flip-flops optimized for SP0 aging using extra leakage (scenario 2). Delay degradation saturates as β increases (after $\beta = 0.25$)

always smaller than the LH transition. The reason is that, an intermediate precharged node in this flip-flop should be discharged in LH transition in order to transfer the input ‘‘one’’ to the output, while for the HL transition no such discharging is required. Hence, the slower path is always the LH path. This may worsen the aging if it is coupled with unbalanced aging. For these flip-flops, the optimizer minimizes the delay of the slower path by taking as much area as it can from the faster path, and giving the area to the slower path. For SDFF, this is attained with 15.8% additional leakage at SP0, but it leads to better S-BTI resiliency.

5.4 Delay-Leakage Trade-Off

In order to understand the trade-off between additional leakage and delay, we optimized a C2MOS flip-flop with several excessive leakage amounts ranging from 0 to 50% (i.e. $\beta \in \{0, 0.125, 0.25, 0.5\}$). As shown by Fig. 9, lower delay degradation can be achieved by allowing the optimization method to design flip-flops with higher leakage. However, the improvement saturates as β increases. Hence, providing extra leakage to the optimizer is only beneficial until about 25%, because the improvement in the delay is not significant. Please note that the designed flip-flops with looser leakage constraints, i.e. higher β , do not necessarily have very high leakage. As shown in Table 2, the optimized flip-flop in scenario 2 (only aging) has only 4.7% extra leakage while providing much better resiliency against S-BTI aging compared to the original flip-flop and scenario 1 (state-of-the-art work).

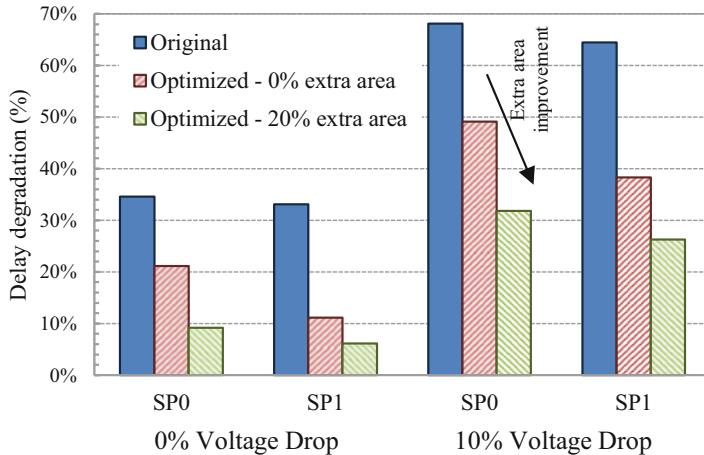


Fig. 10 Comparison of the aging-induced delay degradation under impact of voltage-drop, for original flip-flop, optimized flip-flop with 0% extra area allowance (scenario 2), and optimized flip-flop with 20% extra area allowance (scenario 3). The voltage-drop induced delay increase may be compensated by 20% upsizing of the flip-flop cell during the optimization

5.5 Delay-Area Trade-Off

The impact of a small amount of extra area on the resiliency of the flip-flops against both aging and voltage-drop impacts is studied by changing parameter *excessive area overhead* λ (see Table 1). We run the optimization flow in Sect. 3 for $\lambda \in \{0, 0.2\}$ values and compare the results to the original flip-flop design. Based on the results shown in Fig. 10, the flip-flop designs with no extra area, i.e. scenario 2, exhibit good resiliency against aging; however, under the impact of 10% voltage-drop it has up to 49% delay degradation. Under the impact of voltage-drop, the flip-flop designed with 20% extra area exhibits much better characteristics with maximum 32% delay degradation. This observation confirms that using flip-flops with 20% extra area can be beneficial for the cases when both aging and voltage-drop impacts are severe.

5.6 Circuit-Level Results

The proposed selective flip-flop optimization method presented in Sect. 4 is applied to Leon3 processor with the setup presented in Sect. 5.1 to evaluate the overall impact on the processor timing and reliability. The “original flip-flop” designs are optimized for different output loads for minimum PDP in the fresh state, while the “optimized flip-flop” designs for “aging” and “aging + vdrop” are obtained by applying the proposed method. Therefore, per each original flip-flop design for a

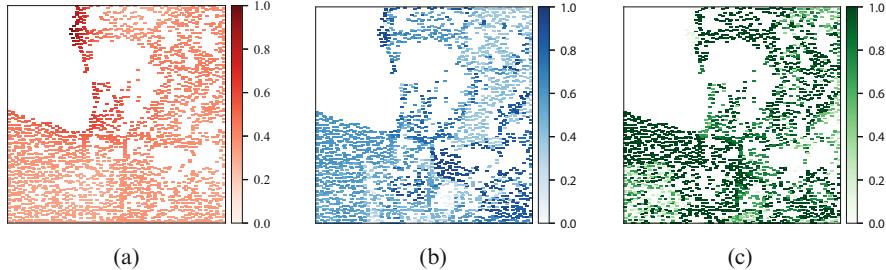


Fig. 11 The layout map of the Leon3 flip-flops during the execution of some MiBench workloads on Leon3, showing relative voltage-drop criticality, timing criticality, and aging-criticality of different flip-flops. Values close to “1” correspond to higher criticality, and values closer to “0” represent the non-critical parts. The top-left part of the processor layout is filled by combinational gates. (a) Relative voltage-drop criticality of flip-flops. (b) Relative timing criticality of flip-flops. (c) Relative aging-criticality of flip-flops

specific output load, there are different optimized designs for S-BTI corners SP0 and SP1 as well as no-vdrop and max-vdrop conditions (according to Sect. 3.5).

The timing of Leon3 processor is evaluated using the “aging and voltage-drop analysis” step of the proposed flow (see Fig. 7). This incorporates using an improved version of an aging-aware timing analysis tool [8] which also considers the impact of supply voltage variation as explained in Sect. 4.1. This timing analysis determines the processor delay under runtime variation impacts.

Figure 11 illustrates the timing of Leon3 flip-flops on the processor layout as well as the calculated impacts of voltage-drop and aging on the processor timing. The presented plots are all normalized to the maximum values (maximum voltage-drop, maximum delay, maximum aging) for better visualization. Therefore, higher values (darker colors) represent a critical situation. Figure 11a presents voltage-drop of the flip-flops extracted using the “aging and voltage-drop analysis” step. The voltage-drop values are normalized to the maximum voltage-drop value extracted during the simulations. As shown, many flip-flops experience at least a moderate voltage-drop during the workload execution. However, the flip-flops on the top-left corner of the layout experience heavy voltage-drop. The timing criticality of the flip-flops is also shown in Fig. 11b. The flip-flops with lower timing slack have values closer to 1.0 in this figure (darker). Interestingly, some of the flip-flops on the top-left corner are also timing-critical. Additionally, the aging-criticality of the flip-flops is presented in Fig. 11c. It is shown that many flip-flops which are under S-BTI are also timing-critical. Most importantly, a few timing-critical flip-flops are affected by both aging and voltage-drop impacts.

Table 3 presents processor delays obtained in fresh state, i.e. no aging or voltage-drop, and when under aging and voltage-drop impacts. We compare the delay of original processor (before applying the proposed method) with the delay of the optimized processors, under runtime variation impacts (aging and voltage-drop) after 7 years. The results are reported for:

Table 3 Processor delay comparison when (1) using only original flip-flops, and (2) using proposed method

	Processor delay in fresh state	Processor delay after 7 years + voltage- drop	Delay degradation	Guardband reduction	Equivalent lifetime improvement
Using original flip-flops	1389.6 ps	1528.2 ps	9.97%	–	–
Proposed (only aging)	1391.3 ps	1494.8 ps	7.44%	33.4 ps	30.8%
Proposed (aging + voltage- drop)	1379.7 ps	1486.7 ps	7.75%	41.5 ps	36.9%

1. “Original processor”: using only original flip-flops,
2. “Optimized processor for aging”: when only the impact of aging is considered during optimization,
3. “Optimized processor for aging and voltage-drop”: when the impacts of aging and voltage-drop are considered during optimization.

The “original processor” is synthesized using the original flip-flops designs in Table 2. Then, we apply the proposed selective flip-flop optimization in two modes: (I) when only aging is considered, and (II) when both aging and voltage-drop are considered. This obtains two versions of the optimized processor, i.e. “Optimized processor for aging” and “Optimized processor for aging and voltage-drop.” In the optimization flow presented in Sect. 4.2, we assume $k = 0.15$. Therefore, all flip-flops with a slack value less than 15% of the processor delay are assumed as timing-critical flip-flops. Additionally, we assume $r = 0.95$, which means up to 5% calculation error guardband in the timing analysis method is acceptable. In fact, r value depends on the accuracy of the timing analysis method. After replacing the critical flip-flops according to the proposed method, the processor delay is obtained again using the “aging and voltage-drop analysis” step.

According to the table, delay of the “original processor” is increased by 9.97% after 7 years. This translates into 138.6 ps timing guardband for 7 years of circuit operation, i.e. $T_{clk} \geq 1528.2$ ps. The “optimized processor for aging” has better delay 1494.8 ps under the impacts of aging and voltage-drop which reduces the required timing guardband by 33.4 ps for 7 years of operation, hence optimizing the performance. Therefore, the degradation rate of this optimized processor is such that it can operate for 9.2 years (30.8% lifetime improvement), if it is used with the timing margins of $T_{clk} = 1528.2$ ps. Finally, the required timing guardband of “Optimized processor for aging and voltage-drop” is further reduced by 41.5 ps compared to the original processor. Therefore, the lifetime of the processor is improved by 36.9% (9.6 years).

The reason for the achieved improvements in Table 3 is explained by Fig. 12. Here, we only plotted the delay of timing-critical flip-flops with a slack smaller

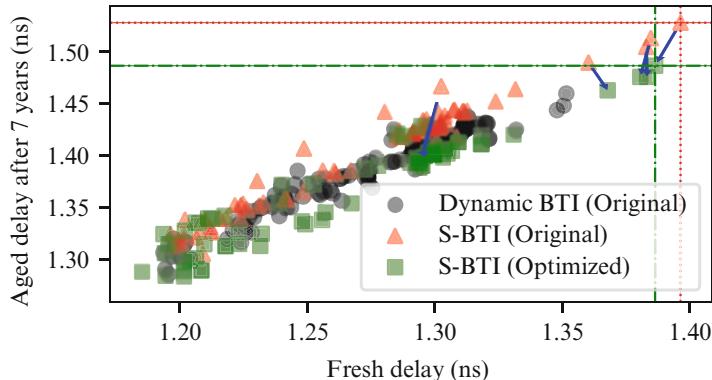


Fig. 12 Fresh delay (no aging, no voltage-drop) vs. increased delay (aged and 10% voltage-drop) of critical paths of Leon3 processor. The proposed selective flip-flop optimization method replaces the original flip-flops under S-BTI (red) with the optimized flip-flops (green) and suppresses the aging and voltage-drop degradation of the most critical paths

than 15% of the processor delay (under aging and voltage-drop impacts). With this assumption, there are 261 timing-critical flip-flops. Among the timing-critical flip-flops, 92 flip-flops are under S-BTI impact (i.e. $0 \leq SP < 0.01$ or $0.99 < SP \leq 1$), 235 flip-flops experience at least 33% relative voltage-drop. After applying the selective flip-flop optimization method, 96 flip-flops are replaced with optimized versions, from which 39 flip-flops are upsized (due to both aging and voltage-drop impact).

As the optimized flip-flops constitute about 4% of all flip-flops in Leon3, the overall leakage overhead with this method is 0.22% according to power analysis results using Synopsys Design Compiler. Moreover, there is virtually no dynamic power overhead because the replaced flip-flops are mostly under S-BTI impact and they rarely switch. The additional area overhead is also very negligible because only 39 flip-flops are replaced by the upsized versions (less than 0.1% area overhead). The ECO process easily fits these flip-flops into the existing layout by slightly moving other cells. Please note that the impact of the voltage-drop and aging on the driving logic paths is much less compared to the flip-flops. Therefore, these paths are degraded at a much lower rate.

6 Comparison with the Related Work

Various methods have been proposed to address the impact of aging and voltage-drop on flip-flops [1, 13, 23, 25]. For example, [1] proposes a method to improve flip-flop reliability for a set of corners with different working conditions such as temperatures and voltages by altering the sizing of transistors. These studies mostly optimize flip-flops for dynamic BTI stress condition, and flip-flops under static BTI

are mostly overlooked. As explained, the traditional optimization techniques such as optimization for the PDP, or EDP cannot effectively address the delay increase of flip-flops under such stress. There are techniques to reduce the overall impact of voltage-drop on VLSI circuits by skewing the clock input of the flip-flops at design-time in order to reduce the peak current at clock edge [9, 35]. However, these methods are not applicable to flip-flops with zero (or close to zero) timing slack on the critical paths. The techniques at high abstraction level by software-guided thread scheduling [27] or by voltage emergency prediction [26] also impose additional overhead at another abstraction level than circuit-level, in order to address a circuit-level problem.

7 Summary

In many cases, NTC circuits are required to operate over a wide voltage range in order to achieve energy efficiency and satisfy performance constraints as needed. Therefore, an NTC circuit may be exposed to reliability issues such as aging and voltage-drop which are significant in the super-threshold region.

In this chapter, we discussed that a non-negligible portion of circuit flip-flops may be under severe aging or large voltage-drop impact, which leads to timing and functional failures. Therefore, these flip-flops need to be treated separately and specific stress-tolerant designs should be used in order to improve the reliability and lifetime. Accordingly, we propose a method to selectively optimize the flip-flops operating under severe aging stress and/or voltage-drop conditions. The proposed optimization flow resizes the flip-flop transistors to obtain the variability-resilient cells. Then, flip-flops which are under the impact of aging and/or voltage-drop are determined using a variation-aware static timing analysis tool, and are replaced by the optimized flip-flops which can withstand aging and voltage-drop impacts much better. Simulation results show that the proposed selective flip-flop optimization method can reduce Leon3 processor timing guardband, and improve the lifetime of the processor by 36.9%, with negligible power and area overhead.

References

1. Abrishami, H., Hatami, S., Pedram, M.: Multi-corner, energy-delay optimized, NBTI-aware flip-flop design. In: International Symposium on Quality Electronic Design (ISQED), pp. 652–659 (2010). <https://doi.org/10.1109/ISQED.2010.5450509>
2. Ajami, A.H., Banerjee, K., Mehrotra, A., Pedram, M.: Analysis of IR-drop scaling with implications for deep submicron P/G network designs. In: International Symposium on Quality Electronic Design (ISQED), pp. 35–40 (2003)
3. Amrouch, H., Ebi, T., Schneider, J., Parameswaran, S., Henkel, J.: Analyzing the thermal hotspots in FPGA-based embedded systems. In: International Conference on Field Programmable Logic and Applications (FPL), pp. 1–4 (2013)

4. Bhardwaj, S., Wang, W., Vattikonda, R., Cao, Y., Vrudhula, S.: Predictive modeling of the NBTI effect for reliable design. In: Custom Integrated Circuits Conference (CICC), pp. 189–192. IEEE, Piscataway (2006)
5. Cadence Encounter Timing System. <http://www.cadence.com>
6. Cadence Virtuoso Liberate Characterization Solution. <http://www.cadence.com/products/cic/liberate/pages/default.aspx>
7. Denney, J., Ramsey, C.: Comparison of finite-difference and spice tools for thermal modeling of the effects of nonuniform power generation in high-power CPUs. *Hewlett-Packard J.* **50**, 37–45 (1998)
8. Ebrahimi, M., Oboril, F., Kiamehr, S., Tahoori, M.B.: Aging-aware Logic Synthesis. In: International Conference on Computer-Aided Design (ICCAD), pp. 61–68 (2013)
9. Fishburn, J.P.: Clock skew optimization. *IEEE Trans. Comput.* **39**(7), 945–951 (1990)
10. Gaisler, A., Göteborg, S.: Leon3 multiprocessing CPU core. Aeroflex Gaisler (2010)
11. Gnad, D.R.E., Oboril, F., Kiamehr, S., Tahoori, M.B.: An experimental evaluation and analysis of transient voltage fluctuations in FPGAs. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **26**(10), 1817–1830 (2018)
12. Golshanbari, M.S., Kiamehr, S., Ebrahimi, M., Tahoori, M.B.: Aging guardband reduction through selective flip-flop optimization. In: IEEE European Test Symposium (ETS) (2015)
13. Golshanbari, M.S., Kiamehr, S., Tahoori, M.B., Nassif, S.: Analysis and optimization of flip-flops under process and runtime variations. In: International Symposium on Quality Electronic Design (ISQED) (2015)
14. Golshanbari, M.S., Kiamehr, S., Ebrahimi, M., Tahoori, M.B.: Selective flip-flop optimization for reliable digital circuit design. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **39**(7), 1484–1497 (2020)
15. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: a free, commercially representative embedded benchmark suite. In: IEEE International Workshop on Workload Characterization, pp. 3–14. IEEE, Piscataway (2001)
16. International Technology Roadmap for Semiconductors (ITRS). <http://www.itrs2.net>
17. Kaul, H., Anders, M.A., Mathew, S.K., Hsu, S.K., Agarwal, A., Krishnamurthy, R.K., Borkar, S.: A 320 mv $56 \mu\text{w}$ 411 GOPS/watt ultra-low voltage motion estimation accelerator in 65 nm CMOS. *IEEE J. Solid State Circuits* **44**(1), 107–114 (2009)
18. Kiamehr, S., Ebrahimi, M., Firouzi, F., Tahoori, M.B.: Extending standard cell library for aging mitigation. *IET Comput. Digit. Tech.* **9**(4), 206–212 (2015)
19. Kraft, D.: A software package for sequential quadratic programming. *Forschungsbericht Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt* **88–28**, 1–33 (1988)
20. Krishnan, A.T., Cano, F., Chancellor, C., Reddy, V., Qi, Z., Jain, P., Carulli, J., Masin, J., Zuhoski, S., Krishnan, S., et al.: Product drift from NBTI: Guardbanding, circuit and statistical effects. In: International Electron Devices Meeting, pp. 4–3 (2010)
21. Mezhiba, A.V., Friedman, E.G.: Scaling trends of on-chip power distribution noise. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **12**(4), 386–394 (2004)
22. Nithin, S., Shannugam, G., Chandrasekar, S.: Dynamic voltage (IR) drop analysis and design closure: Issues and challenges. In: International Symposium on Quality Electronic Design (ISQED), pp. 611–617 (2010)
23. Nunes, C., Butzen, P.F., Reis, A.I., Ribas, R.P.: BTI, HCI and TDDB aging impact in flip-flops. *Microelectron. Reliab.* **53**(9–11), 1355–1359 (2013)
24. Ramakrishnan, K., Wu, X., Vijaykrishnan, N., Xie, Y.: Comparative analysis of NBTI effects on low power and high performance flip-flops. In: International Conference on Computer Design (ICCD), pp. 200–205 (2008)
25. Rao, V.G., Mahmoodi, H.: Analysis of reliability of flip-flops under transistor aging effects in nano-scale CMOS technology. In: International Conference on Computer Design (ICCD), pp. 439–440 (2011)
26. Reddi, V.J., Gupta, M.S., Holloway, G., Wei, G.Y., Smith, M.D., Brooks, D.: Voltage emergency prediction: using signatures to reduce operating margins. In: International Symposium on High-Performance Computer Architecture (HPCA), pp. 18–29 (2009)

27. Reddi, V.J., Kanev, S., Kim, W., Campanoni, S., Smith, M.D., Wei, G.Y., Brooks, D.: Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In: International Symposium on Microarchitecture, pp. 77–88 (2010)
28. Reddy, V., Carulli, J., Krishnan, A., Bosch, W., Burgess, B.: Impact of negative bias temperature instability on product parametric drift. In: International Conference on Test, pp. 148–155 (2004)
29. Sato, T., Ichimiya, J., Ono, N., Hachiya, K., Hashimoto, M.: On-chip thermal gradient analysis and temperature flattening for SoC design. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **88**(12), 3382–3389 (2005)
30. Schlunder, C., Aresu, S., Georgakos, G., Kanert, W., Reisinger, H., Hofmann, K., Gustin, W.: HCI vs. BTI? - Neither one's out. In: IEEE International Reliability Physics Symposium (IRPS), pp. 2F.4.1–2F.4.6 (2012)
31. Stojanovic, V., Oklobdzija, V.G.: Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems. IEEE J. Solid State Circuits **34**(4), 536–548 (1999)
32. Sundareswaran, S.: Statistical characterization for timing sign-off: from silicon to design and back to silicon. Ph.D. Thesis, UT Austin (2009)
33. Tradowsky, C., Cordero, E., Deuser, T., Hübner, M., Becker, J.: Determination of on-chip temperature gradients on reconfigurable hardware. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1–8 (2012)
34. Unger, S.H., et al.: Clocking schemes for high-speed digital systems. IEEE Trans. Comput. **C-35**(10), 880–895 (1986)
35. Vittal, A., Ha, H., Brewer, F., Marek-Sadowska, M.: Clock skew optimization for ground bounce control. In: International Conference on Computer-Aided Design (ICCAD), pp. 395–399 (1996)
36. Wang, W., Yang, S., Bhardwaj, S., Vrudhula, S., Liu, F., Cao, Y.: The impact of NBTI effect on combinational circuit: modeling, simulation, and analysis. IEEE Trans. Very Large Scale Integr. VLSI Syst. **18**(2), 173–183 (2010). <https://doi.org/10.1109/TVLSI.2008.2008810>
37. Wu, J.K., Wu, T.Y., Lu, L.Y., Chen, K.Y.: IR drop reduction via a flip-flop resynthesis technique. In: International Symposium on Quality Electronic Design (ISQED), pp. 78–83 (2008)
38. Zhang, R., Wang, K., Meyer, B.H., Stan, M.R., Skadron, K.: Architecture implications of pads as a scarce resource. In: International Symposium on Computer Architecture (ISCA), pp. 373–384 (2014)
39. Zhao, W., Cao, Y.: New generation of predictive technology model for sub-45nm design exploration. In: International Symposium on Quality Electronic Design (ISQED), pp. 585–590. IEEE Computer Society, Washington (2006)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



EM Lifetime Constrained Optimization for Multi-Segment Power Grid Networks



Han Zhou, Zeyu Sun, Sheriff Sadiqbatcha, and Sheldon X.-D. Tan

1 Introduction

On-chip power supply or power-ground (P/G) networks provide power to the circuit modules in a chip from external power supplies. Since power grid wires experience the largest current flows on a chip, they are more susceptible to long-term reliability issues and functional failures. These reliability issues and failures typically come from metal electromigration (EM), excessive IR drops, and ΔI (Ldi/dt) noise along with recently emerging back end of line time-dependent dielectric breakdown (TDDB) [2, 3, 6].

As technology scales into smaller features with increasing current densities, EM-induced reliability deteriorates, the EM lifetime was projected to be reduced by half for each new technology node by ITRS 2015 [19]. As a result, EM still remains one of the top killers of copper based damascene interconnects for technologies in the sub-10 nm realm. This introduces additional challenges for designing robust power supply networks to satisfy the demanding design requirements.

An important step for power supply synthesis in the typical EDA design flow is sizing the wire width of the power grid stripes, after the topology of the power supply network has been determined, so that the minimum amount of chip area will be used while avoiding potential reliability failures due to electromigration and excessive IR drops. Numerous works have been proposed for the power supply network optimization in the past, primarily based on nonlinear or sequence of linear programming (SLP) methods [8–10, 13, 26, 27, 31].

To satisfy the EM reliability, all the existing methods use the current density of individual wires as the constraint, which is mainly based on the Black's EM

H. Zhou · Z. Sun · S. Sadiqbatcha · S. X.-D. Tan (✉)

Department of Electrical and Computer Engineering, University of California, Riverside, CA,
USA

e-mail: hzhou012@ucr.edu; zsun007@ucr.edu; ssadi003@ucr.edu; stan@ece.ucr.edu

model. However, this constraint is too conservative for modern power grid networks. Furthermore, all existing power supply optimization methods fail to consider the aging effects. With recent advancements in physics-based EM models and numerical analysis techniques such as three-phase EM model [12, 24, 28, 33], it is possible to provide more accurate time to failure (TTF) estimation for multi-segment interconnects.

In this chapter, we present two new P/G network sizing and optimization techniques, which were first introduced in [35, 36]. We will summarize the key contributions and major computing steps from the P/G optimization technique considering the new physics-based EM models. The chapter is organized as follows: Sect. 2 describes the power grid network and its models. Section 3 presents the fundamentals of EM and the voltage-based EM immortality check method for general multi-segment interconnect wires. Section 4 outlines a physics-based three-phase EM model and a fast EM lifetime estimation method. Section 5 introduces the EM immortality constrained P/G network optimization problem and its programming-based solution. Section 6 presents the EM lifetime constrained P/G optimization method, which deals with the EM-induced aging effect. Section 7 summarizes this chapter.

2 Power Grid Modeling

Practical VLSI interconnects (especially the global networks such as power supply and clock networks) have many multi-segment wires as shown in Fig. 1. A multi-segment interconnect wire consists of continuously connected high-conductivity metal within one layer of metallization.

Figure 2 shows a typical mesh-structured P/G network with multi-layer power grids. The modeling assumptions for later optimization are listed as follows. Firstly, because of the concern with the long-term average effects of the current, we focus on the steady state (DC) problem, which means we are only interested in the resistance of the power grid networks. Secondly, the P/G network is composed of an orthogonal mesh of wires and contains multiple segments/branches, which is the typical P/G structure. Lastly, to simplify the problem, the circuits are modeled with shorted vias, which means the via resistance is ignored and vias will not be sized. Figure 3 shows the equivalent circuit of the power grid network in Fig. 2.

As a result, the power grid systems are linear and driven by the DC effective currents [17]. For a power grid network with n nodes,

$$G \times V = I \quad (1)$$



Fig. 1 Example of a multi-segment wire

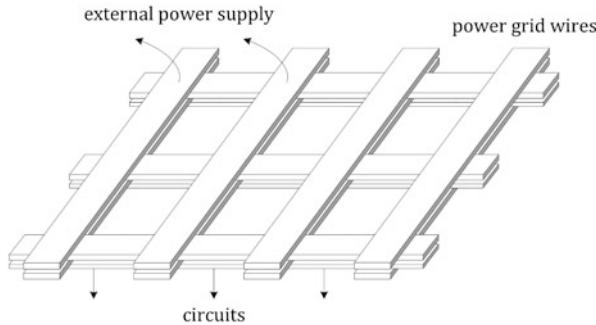


Fig. 2 A small portion of a typical power supply network [22]

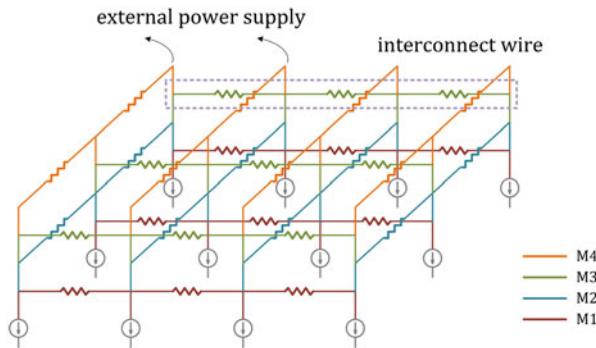


Fig. 3 Equivalent circuit of a small portion of a typical power grid

where G is a $n \times n$ conductance matrix; I is the current source vector; V is the corresponding vector of nodal voltages.

3 Electromigration Fundamentals

3.1 Electromigration Introduction

EM is a physical phenomenon of material migration caused by an electrical field. Wind force, which is produced by current flowing through a conductor, acts in the direction of the current flow and is the primary cause of EM [21]. During the migration process, hydrostatic stress is generated inside the metal wire due to momentum transfer between lattice atoms. Void and hillock formation are caused by conducting electrons at the opposite ends of the wire. The void may lead to early failure or late failure of the wire [1].

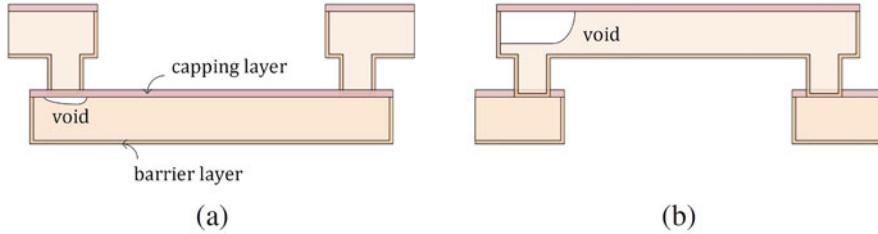


Fig. 4 Side-view of void formation: (a) void in a via-above line (early failure mode); (b) void in a via-below line (later failure mode)

Early failure typically happens in a via-to-via structure as shown in Fig. 4a. When the void forms in a via-above line and reaches critical size [16, 34], which equals the via's diameter, the via will be blocked by the void and thus the connection to the upper layer will also be blocked. This is because the capping layer is fabricated with dielectrics such as Si_3N_4 which will block the current flow. On the contrary, late failure typically happens in a via-below structure as shown in Fig. 4b. Since the barrier layer is fabricated with Ta whose resistivity is much higher than Cu, when the void reaches critical size, current can still go through the barrier layer. Sometimes early failure can happen in a via-below structure and late failure can happen in a via-above structure. Although the void can grow at these positions, the possibility is very low.

When the compressive stress at the anode continues to be built up, hillocks or extrusion may be formed, which will lead to a resistance decrease [30] and can potentially cause short-circuit failure. However, the void nucleation is still the dominant EM failure effect [15].

3.2 Steady State EM-Induced Stress Modeling

Steady state EM-induced stress modeling helps find the immortality information of the interconnect wire quickly as no complex calculations are required. For these kinds of models, stress on the cathode at steady state (σ_{steady}), which is the maximum stress the node experiences, is compared with critical stress (σ_{crit}). If σ_{steady} is lower than σ_{crit} , the wire is considered as immortal. One of the well-known steady state analysis method is *Blech product* [4], but it is only suitable for a single (i.e., one-segment) wire. Recently, a voltage-based EM immortality analysis method for multi-segment interconnect structures has been proposed [23, 24]. In this method, an *EM voltage* (V_E) is calculated as

$$V_E = \frac{1}{2A} \sum_{k \neq g} a_k V_k \quad (2)$$

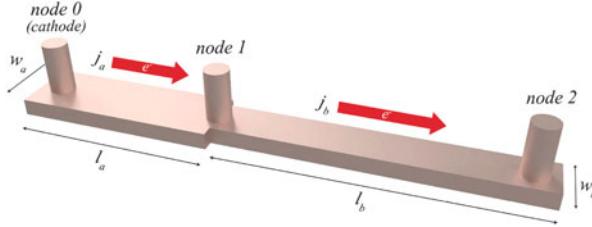


Fig. 5 Interconnect example for EM analysis for straight 3-terminal wire

where V_k is the normal nodal voltage (with respect to cathode node *cat*) at node k , a_k is the total area of branches connected to node k , and A is the total area of the wire. With voltage of node i (V_i), steady state stress at that node (σ_i) can be calculated as $\sigma_i = \beta(V_E - V_i)$, where $\beta = \frac{eZ}{\Omega}$, e is elementary charge, Z is effective charge number, and Ω is the atomic lattice volume. A *critical EM voltage* $V_{crit,EM}$ is defined by

$$V_{crit,EM} = \frac{1}{\beta}(\sigma_{crit} - \sigma_{init}) \quad (3)$$

where σ_{init} is the initial stress. In order to check whether the interconnect wire is immortal, we need to check the following condition

$$V_{crit,EM} > V_E - V_i \quad (4)$$

Note that $V_E - V_i$ is proportional to the stress at cathode node (σ_{cat}).

If this condition is met for all the nodes, EM failure will not happen. Since generally the cathode node has the lowest voltage within an interconnect wire, we may just check the cathode node instead of all the nodes, which means

$$V_{crit,EM} > V_E - V_{cat} \quad (5)$$

where V_{cat} is the voltage at the cathode. Note that inequality (5) can be applied to both power and ground networks.

The method can be illustrated using the following example. Figure 5 shows a 3-terminal wire. In this wire, node 0 is treated as the ground node. Current densities in two segments are j_a and j_b which may not be the same because they will be determined by the rest of the circuit. The *EM voltage* become

$$V_E = \frac{a_0 V_0 + a_1 V_1 + a_2 V_2}{2A} = \frac{a_1 V_1 + a_2 V_2}{2A} \quad (6)$$

where

$$\begin{aligned} V_0 &= 0, & a_0 &= l_a w_a, & \sigma_0 &= \beta V_E \\ V_1 &= j_a l_a \rho, & a_1 &= l_a w_a + l_b w_b, & \sigma_1 &= \beta(V_E - V_1) \\ V_2 &= j_b l_b \rho + j_a l_a \rho, & a_2 &= l_b w_b, & \sigma_2 &= \beta(V_E - V_2) \end{aligned} \quad (7)$$

$$A = \frac{a_0 + a_1 + a_2}{2} \quad (8)$$

We can compare V_E and $V_{crit, EM}$ to see if this wire is immortal.

4 Transient EM-Induced Stress Estimation

In general, the failure process of an interconnect is divided into nucleation phase, incubation phase and growth phase. In the nucleation phase, the stress at the cathode keeps increasing. When it reaches critical stress, a void will be nucleated. The time to reach the critical stress is called nucleation time (t_{nuc}). After the nucleation phase, the void starts to grow (t_{inc}) and eventually leads to wire failure after a period of time (t_{growth}). The TTF or lifetime of the wire can be described as

$$TTF = t_{life} = t_{nuc} + t_{inc} + t_{growth} \quad (9)$$

4.1 Transient EM-Induced Stress Modeling

4.1.1 Nucleation Phase Modeling

It is well-known that the nucleation phase is accurately modeled by Korhonen's equation [20]

$$\frac{\partial \sigma(x, t)}{\partial t} = \frac{\partial}{\partial x} \left[\kappa \left(\frac{\partial \sigma(x, t)}{\partial x} + \Gamma \right) \right] \quad (10)$$

where $\kappa = \frac{D_a B \Omega}{k_B T}$, $D_a = D_0 \exp(-\frac{E_a}{k_B T})$, and $\Gamma = \frac{eZ}{\Omega} \rho_w j$. B is effective bulk elasticity modulus, Ω is atomic lattice volume, k_B is Boltzmann constant, T is temperature, Z is effective charge number, ρ_w is the wire electrical resistivity, x is coordinate along the line, t is time, and j is current density.

Korhonen's equation describes the stress distribution accurately; this PDE-based model is hard to solve directly using numerical methods and has very low efficiency for tree-based EM assessment. Recently a few numerical methods have been proposed such as finite difference methods [5, 11] and analytical expressions based

approaches [7, 32]. In this work, an integral transformation method for straight multi-segment wires [32] is employed. Suppose we have a multi-segment wire, after discretizing Korhonen's equation, the stress can be expressed as

$$\sigma(x, t) = \sum_{m=1}^{\infty} \frac{\psi_m(x)}{N(\lambda_m)} \bar{\sigma}(\lambda_m, t) \quad (11)$$

where the norm of eigenfunctions $N(\lambda_m)$ is

$$N(\lambda_m) = \int_{\chi=0}^L [\psi_m(\chi)]^2 d\chi \quad (12)$$

and the transformed solution of stress $\bar{\sigma}(\lambda_m, t)$ is

$$\begin{aligned} \bar{\sigma}(\lambda_m, t) &= \left(\int_{\chi=0}^L \psi_m(\chi) \cdot \sigma_0(\chi) d\chi \right) e^{-\kappa \lambda_m^2 t} + \frac{1}{\lambda_m^2} (1 - e^{\kappa \lambda_m^2 t}) \\ &\cdot \sum_{k=1}^n \frac{eZ\rho}{\Omega} j_k \cdot \left(\cos \frac{x_{k-1}}{L} m\pi - \cos \frac{x_k}{L} m\pi \right) \end{aligned} \quad (13)$$

Eigenvalues λ_m and eigenfunctions $\psi(x)$ are the solutions of the Sturm–Liouville problem corresponding to the diffusion Eq. (10) and the boundary conditions, which are

$$\lambda_m = \frac{m\pi}{L}, \quad \psi_m(x) = \cos \frac{x}{L} m\pi \quad (14)$$

With Eq. (11), given critical stress σ_{crit} , the nucleation time t_{nuc} can be obtained quickly by using nonlinear equation solving methods such as Newton's method or bisection method.

4.1.2 Incubation Phase Modeling

After the void is nucleated, the incubation phase starts. In this phase, resistance of the interconnect remains almost unchanged since the cross section of the via is not covered by the void and the current can still flow through the copper.

In power grid networks, the interconnect trees are generally multi-segment wires. All segments connected with the void can contribute to the void growth since electron wind at each segment can accelerate or slow down the void growth according to their directions. In this phase, void growth rate v_d is estimated to be [25]

$$v_d = \frac{D_a e Z \rho}{k T W_m} \sum_i j_i W_i \quad (15)$$

where j_i and W_i are the current density and width of the i th segment, respectively. W_m is the width of the main segment where the void is formed.

Then the incubation time t_{inc} can be expressed as

$$t_{\text{inc}} = \frac{\Delta L_{\text{crit}}}{v_d} \quad (16)$$

where ΔL_{crit} is the *critical void length*.

4.1.3 Growth Phase Modeling

After the incubation phase, the void fully covers the via, initiating the growth phase. In this phase the resistance starts increasing. It is important to note that, early failure and late failure have different failure mechanisms.

For early failure, the wire fails once the void covers the via, which means the wire fails at the end of incubation phase and there is no growth phase ($t_{\text{growth}} = 0$). Hence the failure time is the sum of t_{nuc} and t_{inc} .

For late failure, after the void size reaches the critical size, there will be no open circuit because the current can still flow through the barrier layer. In this case, the void growth will lead to resistance increase. When the resistance increases to the critical level, the interconnect wire is considered to be failed. The growth time for late failure is

$$t_{\text{growth}} = \frac{\Delta r(t)}{v_d \left[\frac{\rho_{Ta}}{h_{Ta}(2H+W)} - \frac{\rho_{Cu}}{HW} \right]} \quad (17)$$

where ρ_{Ta} and ρ_{Cu} are the resistivity of tantalum (the barrier liner material) and copper, respectively. W is the line width, H is the copper thickness, and h_{Ta} is the liner layer thickness.

However, the void may saturate before reaching the *critical void length*. The saturation length is expressed in [18] as

$$L_{ss} = L_{\text{line}} \times \left[\frac{\sigma_T}{B} + \frac{eZ\rho j L}{2B\Omega} \right] \quad (18)$$

where L_{ss} is the void saturated length, L_{line} is the total length of the wire, and σ_T is thermal stress. Void growth may stop before the calculated t_{growth} because of the saturated void. If it happens, we treat the wire as immortal or its lifetime is larger than the target lifetime.

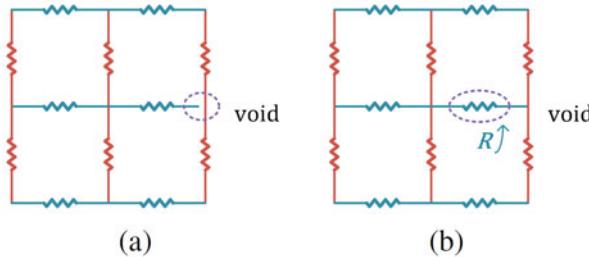


Fig. 6 The electrical impact of different failure mechanisms on the interconnect wires: **(a)** early failure mode; **(b)** late failure mode

4.2 Transient EM Analysis for a Multi-Segment Interconnect Wire

One important aspect of transient EM analysis is calculating the lifetime of a given wire and its electrical conditions. If the increased resistance of the nucleated branch exceeds a threshold, the interconnect tree is marked as failed.

To compute the lifetime t_{life} of a given wire, we need to make sure that the wire is mortal and Eq. (5) is not satisfied. If a target lifetime t_{target} is given, the analysis method will give the resistance change ΔR at the target lifetime.

For those mortal wires, we start with time $t = 1000$ years and use bisection method to find t_{nuc} . The transient hydrostatic stress will be computed by Eq. (11). Once the stress of one segment hits the critical stress, the wire is deemed as nucleated.

Then we need to determine if the wire is void incubation phase immortal. If the saturated void length is less than the critical length, the incubation time (eventually the lifetime) becomes infinite and the resistance remains unchanged.

Otherwise, the failure mode of the wire should be determined by looking at the current direction in the cathode node based on the patterns in Fig. 4.

If the wire is in the early failure mode, then the wire will become an open circuit: the whole interconnect tree will be disconnected from another interconnect wire as shown in Fig. 6a. For the wire in the late failure mode, we have another solution. The wire resistance change will be incurred and the growth time will be computed when the resistance change reaches the threshold as shown in Fig. 6b. If the target lifetime is given, then the wire resistance change ΔR will be computed at the target lifetime.

5 EM Immortality Constrained Optimization for Multi-Segment Interconnects

Study and experimental data show that the current-induced stress developed in the individual segments within an interconnect tree is not independent [14, 29]. In other words, if we just look at the current density for each segment individually, it may appear as if all wire segments are immortal, but the whole interconnect tree could still be mortal. The reason is that the stress in one segment of an interconnect tree depends on other segments [28]. As discussed before, this issue has been resolved by the recently proposed fast EM immortality check method for general multi-segment interconnect wires [23].

In this section, we introduce the EM immortality constrained power grid wire-sizing optimization method considering multi-segment interconnect wires. It can be noticed that the new EM constraint will ensure that all the wires are EM immortal, so we call this method EM immortal power supply optimization.

5.1 Problem Formulation

Let $G = \{N, B\}$ be a P/G network with n nodes $N = \{1, \dots, n\}$ and b branches $B = \{1, \dots, b\}$. Each branch i in B connects two nodes i_1 and i_2 with current flowing from i_1 to i_2 . l_i and w_i are the length and width of branch i , respectively. ρ is the sheet resistivity. The resistance r_i of branch i is

$$r_i = \frac{V_{i_1} - V_{i_2}}{I_i} = \rho \frac{l_i}{w_i} \quad (19)$$

5.1.1 Objective Function

The total routing area of a power grid network in terms of voltages, currents, and lengths of branches can be expressed as follows

$$f(V, I) = \sum_{i \in B} l_i w_i = \sum_{i \in B} \frac{\rho I_i l_i^2}{V_{i_1} - V_{i_2}} \quad (20)$$

We notice that the objective function is linear for branch current variables I and nonlinear for node voltage variables V .

5.1.2 Constraints

The constraints that need to be satisfied for a reliable, working P/G network are shown as follows.

Voltage IR Drop Constraints

In order to ensure proper logic operation, the IR drop from the P/G pads to the nodes should be restricted. For each node, we must specify a threshold voltage

$$V_j > V_{\min} \text{ for power network} \quad (21)$$

where V_j is the nodal voltage and V_{\min} is the minimum required voltage for the power nodes.

Minimum Width Constraints

The widths of the P/G segments are technologically limited to the minimum width allowed for the layer where the segment lies in

$$w_i = \rho \frac{l_i I_i}{V_{i1} - V_{i2}} \geq w_{i,\min} \quad (22)$$

New Electromigration Constraints for Multi-Segment Interconnects

As described before, for a multi-segment interconnect m , the EM constraint should be satisfied

$$V_{crit,EM} > V_{E,m} - V_{cat,m} \quad (23)$$

where $V_{E,m}$ is the *EM voltage* for the m th interconnect tree, which is computed using Eq.(2). $V_{cat,m}$ is the cathode nodal voltage of that tree. Unlike previous methods whose branch currents are monitored and used as constants, in our new method, voltages are used as constraints. Thus, only the cathode node voltage for a whole interconnect tree needs to be monitored and no other complex calculations are required.

We remark that $V_{E,m}$, which is defined in (2), is a function of both nodal voltage and total area of wires. As a result, it is a nonlinear function of the nodal voltage (as the area of a wire segment is a function of both nodal voltage and branch current as defined in the cost function (20)). But if we have the equal width constraints as shown below, then constraint (23) actually becomes a linear function of nodal voltage again. For many practical P/G networks, most wire segments in an interconnect tree indeed have the same width.

Equal Width Constraints

For typical chip layout designs, certain tree branches should have the same width. The constraint is $w_i = w_k$, which can be written as

$$\frac{V_{i1} - V_{i2}}{l_i I_i} = \frac{V_{k1} - V_{k2}}{l_k I_k} \quad (24)$$

Kirchhoff's Current Law (KCL)

For each node j , we have

$$\sum_{k \in B(j)} I_k = 0 \quad (25)$$

where $B(j)$ is the set of branches incident on node j .

5.2 New EM Immortality Constrained P/G Optimization

The power grid optimization aims to minimize objective function (20) subjected to constraints (21)–(25). It will be referred as problem P . Problem P is a constrained nonlinear optimization problem.

5.2.1 Relaxed Two-Step Sequence of Linear Programming Solution

In the aforementioned optimization problem, we notice that the newly added EM constraint (23) is still linear in terms of nodal voltage. As a result, we can follow the relaxed two-phase iterative optimization process [8, 27] and apply the sequence of linear programming technique [27] to solve the relaxed problem. Specifically, we have two phases: the voltage solving phase (P - V phase) and the current solving phase (P - I phase).

P-V Optimization Phase

In this phase, we assume that all branch currents are fixed, then the objective function can be rewritten as

$$f(V) = \sum_{i \in B} \frac{\alpha_i}{V_{i1} - V_{i2}} \quad (26)$$

where $\alpha_i = \rho I_i l_i^2$, subject to constraints (21)–(24). We further restrict the changes of nodal voltages such that their current directions do not change during the optimization process

$$\frac{V_{i1} - V_{i2}}{I_i} \geq 0 \quad (27)$$

Problem P - V is nonlinear; however, it can be converted to a sequence of linear programming problem. By taking the first-order Taylor's expansion of Eq. (26) around the initial solution V^0 , the linearized objective function can be written as

$$g(V) = \sum_{i \in B} \frac{2\alpha_i}{V_{i1}^0 - V_{i2}^0} - \sum_{i \in B} \frac{\alpha_i}{(V_{i1}^0 - V_{i2}^0)^2} (V_{i1} - V_{i2}) \quad (28)$$

Besides, an additional constraint will be added [27]

$$\xi \text{sign}(I_i) \left(V_{i1}^0 - V_{i2}^0 \right) \leq \text{sign}(I_i) (V_{i1} - V_{i2}) \quad (29)$$

where $\xi \in (0, 1)$ is a restriction factor, which will be selected by some trials and experience and $\text{sign}(x)$ is the sign function.

Now, the procedure for solving problem $P\text{-}V$ is transformed to the problem of repeatedly choosing ξ and minimizing $g(V)$ until the optimal solution is found. Theoretically, given $g(V_m) < g(V_{m-1})$, there always exists a ξ such that $f(V_m) < f(V_{m-1})$; however, one-dimensional line search method is a more efficient way to find the solution point. Specifically, given V_m and V_{m-1} , the search direction can be defined as $d_m = V_m - V_{m-1}$. Line search finds an $\alpha \in [0, 1]$ such that

$$f(\alpha d_m + V_{m-1}) < f(V_{m-1}) \quad (30)$$

$\alpha d_m + V_{m-1}$ becomes new V_m for the next iteration.

P-I Optimization Phase

In this phase, we assume that all nodal voltages are fixed, so the objective function becomes

$$f(I) = \sum_{i \in B} \beta_i I_i \quad (31)$$

where $\beta_i = \frac{\rho l_i^2}{V_{i1} - V_{i2}^0}$, subject to constraints (22), (24), and (25). Similarly, we restrict the changes of current directions during the optimization process

$$\frac{I_i}{V_{i1} - V_{i2}} \geq 0 \quad (32)$$

As can be seen, problem $P\text{-}I$ is a linear programming problem.

5.2.2 New EM Immortality Constrained P/G Optimization Algorithm

The new EM immortality constrained P/G optimization starts with an initial feasible solution. We iteratively solve $P\text{-}V$ and $P\text{-}I$. The global minimum of convex problem $P\text{-}V$ will be achieved by performing several linear programming processes iteratively. The entire EM immortality constrained power grid network optimization procedure is summarized as Algorithm 1.

Algorithm 1 New EM immortality constrained P/G wire-sizing algorithm

Input: Spice netlist G_I containing a P/G network.
Output: Optimized P/G network parameters.

```

1: /*Problem Setup*/
2:  $k := 0$ .
3: Compute the initial  $V^k, I^k$  from  $G_I$ .
4: repeat
5:   /*P-V Phase*/
6:   Construct constraints (22), (23), (24), (27) and (29) with  $I^k$ .
7:    $m := 1$ .
8:   Compute  $V_m^k := \arg \min g(V^k)$  subject to constraints (21), (22), (23), (24), (27) and (29).
9:   while  $f(V_m^k) > f(V_{m-1}^k)$  do
10:    Determine the search direction  $d_m := V_{m-1}^k - V_m^k$ .
11:    Choose step size  $\alpha$  for line search.
12:     $V_{m+1}^k := V_m^k + \alpha d_m$ .
13:     $m := m + 1$ .
14: end while
15:  $V^{k+1} := V_m^k$ .
16: /*P-I Phase*/
17: Construct constraints (22), (24) and (32) with  $V^{k+1}$ .
18: Compute  $I^{k+1} := \arg \min f(I^k)$  subject to (22), (24), (25), and (32) constraints.
19:  $k := k + 1$ .
20: until  $|f(V^k, I^k) - f(V^{k-1}, I^{k-1})| < \varepsilon$ 
21: Return  $f(V, I)$ .
  
```

In practice, only a few linear programmings are needed to reach the optimum solution. Thus the time complexity of our method is proportional to the complexity of linear programming.

6 EM Lifetime Constrained Optimization

In the previous sections, we discussed the power grid sizing optimization ensuring none of the interconnect trees fails based on the voltage-based EM immortality check. However, such EM constraint may be too conservative because in reality, some wires can be allowed to have EM failure as long as the power grid network is still functional (its IR drop is still less than the given threshold) at the target lifetime (e.g., 10 years).

6.1 New EM Lifetime Constrained Optimization Flow

In this section, we propose a new EM lifetime constrained P/G wire sizing optimization method in which some segments of multi-segment interconnect wires will be allowed to fail or to age. The impacts of these segments in terms of resistance change or even wire openings will be explicitly considered and modeled. Such

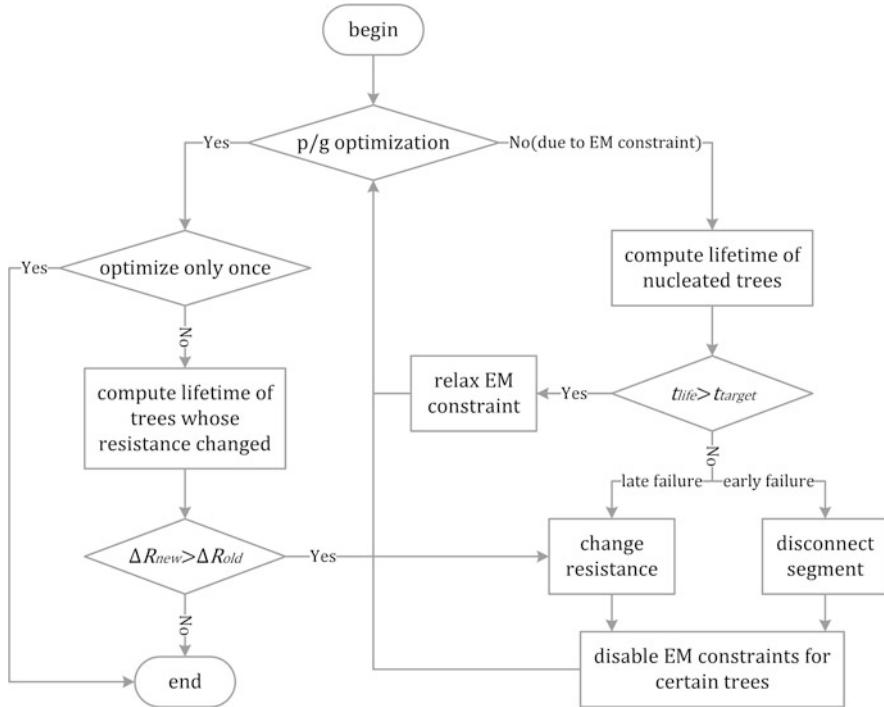


Fig. 7 Flowchart of the EM lifetime constrained P/G optimization process

aging-aware EM optimization essentially takes the EM aging-induced impacts or guard bands into account so that the designed P/G networks can still function nominally during the target lifetime. In this work, we only consider void formation, which is the dominant EM failure effect and will lead to an increase in resistance.

The new optimization flow is shown in Fig. 7. In this new flow, we first check whether a given power supply network can be optimized using Algorithm 1. If the optimization fails due to EM constraint, then the lifetime of all the interconnect trees will be computed based on the EM lifetime estimation method. We have several scenarios to discuss before we perform the optimization again. Let us define $t_{life,m}$ as the lifetime of the m th interconnect tree and t_{target} as the target lifetime.

If $V_{E,m} - V_{cat,m} > V_{crit,EM}$ and $t_{life,m} < t_{target}$

The m th interconnect wire will be marked as a *failed wire*. Then we have the following changes for the wire before the next round of optimization.

If it is an early failure case, the cathode node of the wire segment connected by the failed via will be disconnected, which is called *wire disconnection*. The failure cases will depend on the current directions around the cathode node. Also the disconnection will depend on whether the void growth can eventually reach the critical void size or not as discussed in Sect. 4.1.

If it is a late failure case, the wire segment associated with the cathode node will have a *resistance change*. The specific resistance change for each failed segment will be calculated based on the target lifetime using our EM lifetime estimation method.

If an interconnect tree is marked as failed, then its EM constraint will be disabled as we do not need to consider its immortality anymore.

If $V_{E,m} - V_{cat,m} > V_{crit,EM}$ and $t_{life,m} > t_{target}$

The lifetime of interconnect wire still meets the target lifetime even though it will have void nucleation and resistance change. This also includes the case in which void growth saturates before its size reaches the critical void size. The wire still works since the current can flow through the barrier layer.

The existing $V_{E,m} - V_{cat,m}$ value is used as the new EM constraint (defined as $V_{E,m,next} - V_{cat,m,next}$) for the m th wire only: $V_{E,m} - V_{cat,m} < V_{E,m,next} - V_{cat,m,next}$. This is called *constraint relaxation*. The rational behind it is that we expect the EM status of this wire to become worse during the next optimization so its lifetime will not change too much and still meet the given lifetime after the follow-up optimizations.

After *resistance change*, or *wire disconnection*, or *constraint relaxation*, a new round of SLP programming optimization, which is similar to Algorithm 1, is carried out.

7 Summary

In this chapter, a new P/G network sizing technique is presented, which is based on a voltage-based EM immortality check method for general multi-segment interconnect wires and a physics-based EM assessment technique for fast time to failure analysis. The new P/G optimization problem subject to the voltage IR drop and new EM constraints can still be formulated as an efficient sequence of linear programming problem, and will ensure that none of the wires fails if all the constraints are satisfied. To mitigate the overly conservative nature of the optimization formulation, the EM-induced aging effects on power supply networks for a target lifetime are further considered and an EM lifetime constrained optimization method is demonstrated, which allows some short-lifetime wires to fail and optimizes the rest of the wires. The new methods can effectively reduce the area of the power grid networks while ensuring reliability in terms of immortality or target lifetime, which is not the case for the existing current density constrained P/G optimization methods.

Acknowledgments This chapter is supported in part by NSF grant under No. CCF-1527324, and in part by NSF grant under No. CCF-1816361, and in part by NSF grant under No. OISE-1854276 in part by DARPA grant under No. HR0011-16-2-0009. We also thank Prof. Naehyuck Chang of KAIST for some suggestions to improve presentation of this work.

References

1. Alam, S.M., Gan, C.L., Thompson, C.V., Troxel, D.E.: Reliability computer-aided design tool for full-chip electromigration analysis and comparison with different interconnect metallizations. *Microelectr. J.* **38**(4), 463–473 (2007)
2. Bakoglu, H.B.: Circuits, Interconnections, and Packaging for VLSI. Addison-Wesley, Massachusetts (1990)
3. Black, J.R.: Electromigration-A brief survey and some recent results. *IEEE Trans. Electr. Devices* **16**(4), 338–347 (1969)
4. Blech, I.A. (1976) Electromigration in thin aluminum films on titanium nitride. *J. Appl. Phys.* **47**(4), 1203–1208
5. Chatterjee, S., Sukharev, V., Najm, F.N.: Power grid electromigration checking using physics-based models. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **37**(7), 1317–1330 (2018)
6. Chen, F., Bravo, O., Chanda, K., McLaughlin, P., Sullivan, T., Gill, J., Lloyd, J., Kontra, R., Aitken, T.: A comprehensive study of low-k SiCOH TDDB phenomena and its reliability lifetime model development. In: 2006 IEEE International Reliability Physics Symposium Proceedings, pp. 46–53. IEEE, Piscataway (2006)
7. Chen, H.B., Tan, S.X.D., Huang, X., Kim, T., Sukharev, V.: Analytical modeling and characterization of electromigration effects for multibranch interconnect trees. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **35**(11), 1811–1824 (2016)
8. Chowdhury, S.: Optimum design of reliable IC power networks having general graph topologies. In: Proceedings of the 1989 26th ACM/IEEE Design Automation Conference (DAC), pp. 787–790. IEEE, Piscataway (1989)
9. Chowdhury, S.U., Breuer, M.A.: Minimal area design of power/ground nets having graph topologies. *IEEE Trans. Circuits Syst.* **34**(12), 1441–1451 (1987)
10. Chowdhury, S., Breuer, M.A.: Optimum design of IC power/ground nets subject to reliability constraints. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **7**(7), 787–796 (1988)
11. Cook, C., Sun, Z., Kim, T., Tan, S.X.D.: Finite difference method for electromigration analysis of multi-branch interconnects. In: Proceedings of the 2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), pp. 1–4. IEEE, Piscataway (2016)
12. Cook, C., Sun, Z., Demircan, E., Shroff, M.D., Tan, S.X.D.: Fast electromigration stress evolution analysis for interconnect trees using Krylov subspace method. *IEEE Trans. Very Large Scale Integr. Syst.* **26**(5), 969–980 (2018)
13. Dutta, R., Marek-Sadowska, M.: Automatic sizing of power/ground (P/G) networks in VLSI. In: Proceedings of the 1989 26th ACM/IEEE Design Automation Conference (DAC), pp. 783–786. IEEE, Piscataway (1989)
14. Hau-Riege, S.P., Thompson, C.V.: Experimental characterization and modeling of the reliability of interconnect trees. *J. Appl. Phys.* **89**(1), 601–609 (2001)
15. Hu, C.K., Small, M.B., Ho, P.S.: Electromigration in Al (Cu) two-level structures: effect of Cu and kinetics of damage formation. *J. Appl. Phys.* **74**(2), 969–978 (1993)
16. Hu, C.K., Anaperi, D., Chen, S.T., Gignac, L.M., Herbst, B., Kaldor, S., Krishnan, M., Liniger, E., Rath, D.L., Restaino, D., Rosenberg, R., Rubino, J., Seo, S.C., Simon, A., Smith, S., Tseng, W.T.: Effects of overlayers on electromigration reliability improvement for Cu/low k interconnects. In: 2004 IEEE International Reliability Physics Symposium Proceedings, pp. 222–228. IEEE, Piscataway (2004)
17. Huang, X., Yu, T., Sukharev, V., Tan, S.X.D.: Physics-based electromigration assessment for power grid networks. In: Proceedings of the 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2014)

18. Huang, X., Kteyan, A., Tan, S.X.D., Sukharev, V.: Physics-based electromigration models and full-chip assessment for power grid networks. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.* **35**(11), 1848–1861 (2016)
19. ITRS: International Technology Roadmap for Semiconductors (ITRS) Interconnect, 2015 edition (2015). <http://public.itrs.net>
20. Korhonen, M.A., Bo/rgesen, P., Tu, K.N., Li, C.Y.: Stress evolution due to electromigration in confined metal lines. *J. Appl. Phys.* **73**(8), 3790–3799 (1993)
21. Lienig, J., Thiele, M.: Fundamentals of Electromigration-Aware Integrated Circuit Design. Springer, Berlin (2018)
22. Nassif, S.R.: Power grid analysis benchmarks. In: Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 376–381. IEEE, Piscataway (2008)
23. Sun, Z., Demircan, E., Shroff, M.D., Kim, T., Huang, X., Tan, S.X.D.: Voltage-based electromigration immortality check for general multi-branch interconnects. In: Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–7. IEEE, Piscataway (2016)
24. Sun, Z., Demircan, E., Shroff, M.D., Cook, C., Tan, S.X.D.: Fast electromigration immortality analysis for multisegment copper interconnect wires. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **37**(12), 3137–3150 (2018)
25. Sun, Z., Sadiqbatcha, S., Zhao, H., Tan, S.X.D.: Accelerating electromigration aging for fast failure detection for nanometer ICs. In: Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 623–630. IEEE, Piscataway (2018)
26. Tan, S.X.D., Shi, C.J.R.: Efficient very large scale integration power/ground network sizing based on equivalent circuit modeling. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **22**(3), 277–284 (2003)
27. Tan, S.X.D., Shi, C.J.R., Lee, J.C.: Reliability-constrained area optimization of VLSI power-/ground networks via sequence of linear programmings. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **22**(12), 1678–1684 (2003)
28. Tan, S.X.D., Amrouch, H., Kim, T., Sun, Z., Cook, C., Henkel, J.: Recent advances in EM and BTI induced reliability modeling, analysis and optimization. *Integr. VLSI J.* **60**, 132–152 (2018)
29. Thompson, C.V., Hau-Riege, S.P., Andleigh, V.K.: Modeling and experimental characterization of electromigration in interconnect trees. In: AIP Conference Proceedings, AIP, vol. 491, pp. 62–73 (1999)
30. Verbruggen, A.H., van den Homberg, M.J.C., Jacobs, L.C., Kalkman, A.J., Kraayeveld, J.R., Radelaar, S.: Resistance changes induced by the formation of a single void/hillock during electromigration. In: AIP Conference Proceedings, AIP, vol. 418, pp. 135–146 (1998)
31. Wang, K., Marek-Sadowska, M.: On-chip power-supply network optimization using multigrid-based technique. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **24**(3), 407–417 (2005)
32. Wang, X., Wang, H., He, J., Tan, S.X.D., Cai, Y., Yang, S.: Physics-based electromigration modeling and assessment for multi-segment interconnects in power grid networks. In: Proceedings of the 2017 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1727–1732. IEEE, Piscataway (2017)

33. Wang, X., Yan, Y., He, J., Tan, S.X.D., Cook, C., Yang, S.: Fast physics-based electromigration analysis for multi-branch interconnect trees. In: Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 169–176. IEEE, Piscataway (2017)
34. Zhang, L.: Effects of scaling and grain structure on electromigration reliability of cu interconnects. PhD Thesis, University of Texas at Austin (2010)
35. Zhou, H., Sun, Y., Sun, Z., Zhao, H., Tan, S.X.D.: Electromigration-lifetime constrained power grid optimization considering multi-segment interconnect wires. In: Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 399–404. IEEE, Piscataway (2018)
36. Zhou, H., Sun, Z., Sadiqbatcha, S., Chang, N., Tan, S.X.D.: EM-aware and lifetime-constrained optimization for multisegment power grid networks. *IEEE Trans. Very Large Scale Integr. Syst.* **27**(4), 940–953 (2019)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Monitor Circuits for Cross-Layer Resiliency



Mahfuzul Islam and Hidetoshi Onodera

1 Introduction

The end of supply voltage scaling has pushed circuit designers to find for new solutions to reduce power consumption. One key reason for the stoppage of supply scaling is variability including aging. The International Technology Roadmap for Semiconductors (ITRS) highlights performance variability and reliability management in the next decade as a red brick (i.e., a problem with no known solutions) for the design of computing hardware [1]. Instead of operating under predefined supply voltage and clock frequency, the circuit must adapt itself according to its process conditions, as well as to the dynamic changes of temperature, aging, and workload to harness the full potential of technology scaling. With the resilient operations, a chip's lifetime can be extended, and energy consumption can be reduced.

Due to significant variations in temperature, workload, and aging, dynamic tuning of not only the supply voltage and clock frequency but also the threshold voltages has become a necessity for energy-efficient operation. However, without knowing the device and environmental parameters, tuning of these parameters is not possible. On-chip monitor circuits which provide the information about device and environment come to play an important role. On-chip monitors realize an interface between hardware and software, which then can be utilized for software-controlled optimization. The future LSI (Large Scale Integration) chip will require lots of monitors to track transistor performances, temperature changes, supply voltage droops, and leakage current variations. This chapter describes some design techniques of monitor circuits based on delay cells and then presents a reconfigurable monitor

M. Islam (✉) · H. Onodera
Kyoto University, Kyoto, Japan
e-mail: islam.akmmahfuzul.3w@kyoto-u.ac.jp; onodera.hidetoshi.4x@kyoto-u.ac.jp

architecture to realize different delay characteristics with a small area footprint. An extraction methodology of physical parameters from a set of monitor circuits is presented for model-hardware correlation.

2 Cross-Layer Resiliency

This section describes the benefit of realizing cross-layer resiliency by dynamic tuning of threshold voltage, supply voltage, and clock frequency. Cross-layer resiliency enables energy-efficient operation by eliminating excessive margins. We highlight the importance of run-time sensing of circuit delay, leakage current, switching power, temperature, and threshold voltage to realize minimum energy operation under process, voltage, temperature, activity and temperature variations. Multiple on-chip monitor circuits are required to sense these parameters. Although monitor circuits are not a part of the actual circuit, they are essential components for run-time tuning.

2.1 Parameter Fluctuation and Aging

Variations in physical parameters such as transistor threshold voltage, and temperature have spatial distributions over a chip with both of the random and systematic components. Besides the physical parameter variations, environmental variations also affect circuit performance significantly. Temperature variations of more than 50 °C between different parts within a chip are reported [2]. Increase in temperature degrades circuit performance and increases leakage power. According to ITRS, supply voltage fluctuation is considered to be $\pm 10\%$ of the nominal voltage. Sudden drop of supply voltage may cause critical timing failure causing system malfunctioning. Because of process variation, some chips can be slow and some chips can be fast. Fast chips tend to be leaky causing larger energy consumption. Designers thus face a challenge to meet both of the delay and power constraints, since the circuit needs to operate correctly under all of the variation scenarios.

Device characteristics also degrade over time. Aging causes reliability issues where high temperature accelerates device aging. Device phenomena such as Negative Bias Temperature Instability (NBTI) is reported to cause 10% of delay degradation in digital circuits for a 70-nm process over 10 years [3]. Designing the circuit for the worst possible scenario is energy inefficient as it increases area, power, and cost. A chip may face extreme worst-case scenarios once in several years. The conventional worst-case design methodology, where the operating conditions of a circuit are set such as to meet the worst-case performance, is way too energy inefficient and new design paradigm incorporating on-chip monitor circuits have become indispensable. In the new design paradigm, parameters such as the supply voltage and threshold voltage are tuned in the run-time such that the target delay

and power profile are achieved. As a result, instead of worrying for the worst-case performance, the circuit can now be designed to achieve optimal performances.

2.2 Cross-Layer Resiliency for Energy-Efficient Operation

Figure 1 shows a typical design hierarchy of a system-on-a-chip. First, transistor models for a target process technology node are given to circuit designers. These transistor models contain statistical models to simulate the effects of variations on circuit performance. To guarantee error-free circuit operation, a circuit is tested for extreme cases by using the assumed models. As a result, the circuits tend to be over-designed which result in excessive energy consumption. From a system perspective, the circuits need to operate at different supply voltages and clock frequencies while ensuring correct operations. The selection of adequate clock frequency and supply voltage is performed pessimistically. Design-time optimization is an open-loop operation; thus the operating conditions are set for the worst-cases. The solution obviously is to create a feedback loop into the system which can only be realized by tuning circuit parameters in the run-time. Run-time tuning relaxes the design constraints on the circuit and as a result the circuit become better optimized compared with the one where no run-time tuning is performed.

Figures 2 and 3 show two profiles of energy consumption for an LSI. Figure 2 shows simulated energy and frequency contour plots on the threshold voltage (V_{th}) and the supply voltage (V_{dd}) plane for a model circuit operating at an activity rate of 1%. The model circuit used here is a delay line of 40 inverter cells. A commercial

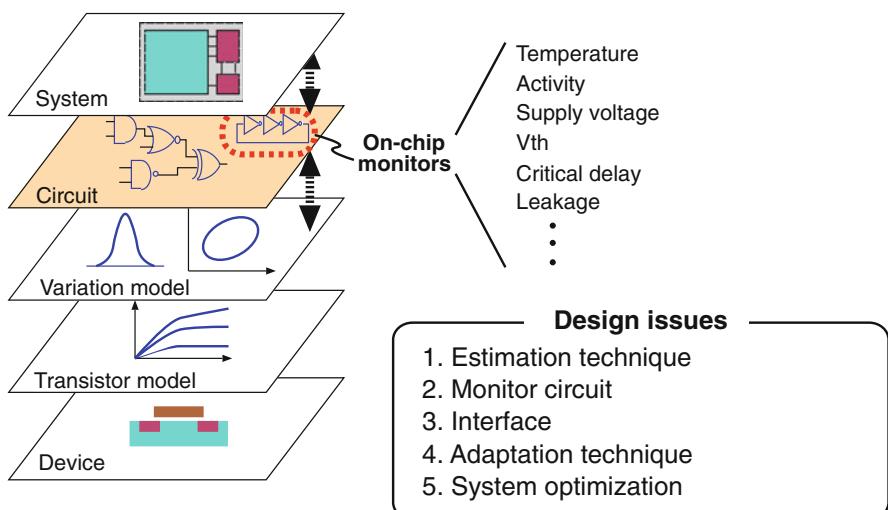


Fig. 1 Cross-layer optimization with the use of monitor circuits

Fig. 2 Energy and frequency contour plot on the V_{th} and V_{dd} plane. Activity rate of 0.01 is assumed

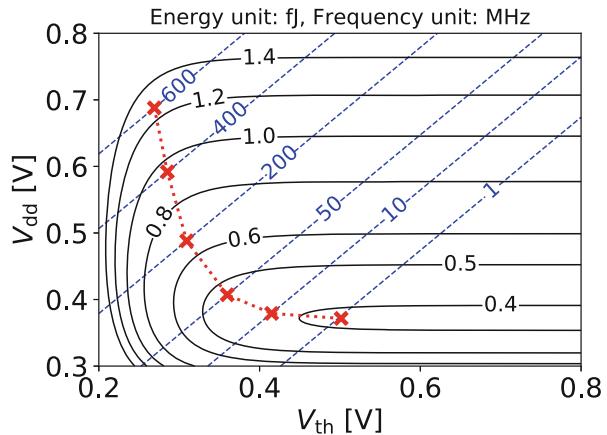
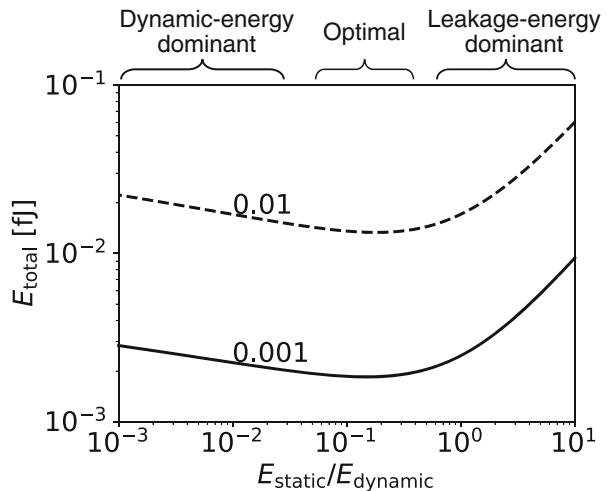


Fig. 3 Total energy per clock cycle against the ratio between static and dynamic energy for a clock frequency of 100 MHz. Having a balanced static and dynamic energy is the key to minimum energy operation



65 nm process is assumed here. Cross points in the plot show the sets of V_{th} and V_{dd} values that give the minimum energy operation for each operating frequency. We observe that the required V_{th} and V_{dd} values, that realize the minimum energy operation, differ significantly with the changes in the clock frequency. Dynamic adaptation of V_{th} and V_{dd} values ensures minimum energy operation for any operating frequency. Figure 3 shows the total energy of the circuit operating at 100 MHz under different combinations of V_{th} and V_{dd} against the ratio of static energy (E_{static}) to dynamic energy ($E_{dynamic}$). We observe that a ratio of 10 to 50% realizes near minimum energy operation. Under the variations of circuit activity, operating frequency and temperature, the energy ratio varies largely. To ensure minimum energy operation, V_{dd} and V_{th} values need to be tuned such that a ratio between 10 and 50% is realized. From the figures, the need for run-time tuning of V_{dd} and V_{th} values are apparent but the problem is how to realize such a mechanism.

Two key mechanisms are required to realize a feedback system. One is the sensing mechanism of the output. The other is to feedback the output to the input of the system. Sensing mechanism is an essential component here. In the case of an LSI, the output parameters are the V_{th} values, circuit delays, temperature, leakage current, and switching current. Sensing these parameters requires multiple on-chip monitor circuits. The monitors provide real-time information of the hardware which can then be used to set the parameters of V_{dd} , V_{th} and clock frequency optimally for reliable operation.

2.3 Role of Monitor Circuits

The past trend of using smaller transistors to achieve higher operating frequency has come to an end [4]. Instead of the clock frequency, system throughput and energy per throughput are the modern specifications for a device. The new era of LSI scaling is a system-on-a-chip (SoC) approach that combines a diverse set of components including adaptive circuits, integrated on-chip monitors, sophisticated power-management techniques, and increased parallelism to build products that are many-core, multi-core, and multi-function [5]. The ability to adapt to the changes in environment and performance will give us the full benefit of technology scaling. Tuning mechanisms and on-chip monitors are needed to realize circuits that have the ability to adapt. The future SoC must have capabilities of post-silicon self-healing, self-configuration, and error correction. Effective use of on-chip monitor circuits will play a major role in continuing the advancement of LSI. Use of on-chip monitors provides us the following advantages:

1. Reduce design margin in each layer of design hierarchy by eliminating pessimism.
2. Tune system parameters based on the actual hardware profile.
3. Provide information for silicon debugging and timing analysis.

To harness the above advantages, the following characteristics of on-chip monitor circuits are preferred:

Digital	Digital in nature realizes robust operation under different supply voltages.
Design automation	Monitor circuits for threshold voltage, temperature, supply voltage, interconnect, activity, and leakage current are required. Thus, design automation is a key factor here for low-cost implementation of the monitors. Cell-based design with delay cells are preferred.
Area efficiency	Area efficiency is an important parameter for fine-grain and distributed implementation of monitor circuits on the chip.

As the target parameters such as the temperature and leakage current are analog values, mechanisms to convert the analog values to digital values are required to interface with the other components of the system. Two design methodologies can be adopted for designing monitor circuit. One methodology performs operations in the analog domain to sense and amplify the effect of the parameter variation and then convert the analog value to a digital value. The other methodology converts the analog value to a digital value as early as possible and then make operations in the digital domain. Incorporating the analog value in the delay of a logic gate realizes the later. Furthermore, the well established cell-based design methodology for automation can be adopted readily for the delay-based implementation of monitor circuits. We therefore explore several delay-based implementations of monitor circuits in this chapter.

3 Delay-Based On-Chip Monitor Design

Delay-based monitor circuits use the mechanisms of converting the target analog value to the delay of a logic gate. The topology of the logic gate thus need to be designed such that the target parameter variation is amplified in the delay. To understand the delay-based monitoring, we first give an overview of the general delay characteristics of logic gates. Then we explore several techniques to tune the delay characteristics such that the monitoring of a target parameter can be realized. Finally, we demonstrate a cell-based design of a reconfigurable monitor circuit that can sense the parameters of nMOSFET and pMOSFET threshold voltages.

3.1 *Delay Characteristics*

Delay-based monitoring is based on the fact that the delay of a logic gate contains information of the transistor drain current I_d . Figure 4 shows four delay paths consisting of different logic gates and interconnects. A delay path of Fig. 4a consists of inverter gates. Delay paths of Fig. 4b and c consist of NAND2 and NOR2 gates. A delay path of Fig. 4d consists of inverter gates with long interconnecting wires. Depending on the topology of the logic gate and the interconnect length, delays of different gates and interconnect show different behavior to process, voltage, and temperature variation. Figure 5 shows the topology of four different logic gates. Figure 5a shows a conventional inverter topology. Figure 5b shows a NAND2 topology where two nMOSFETs are placed in stack. Figure 5c shows a NOR2 topology where two pMOSFETs are placed in stack. Figure 5d shows an inverter topology where two pMOSFETs and two nMOSFETs are placed in stack to mimic the delay behavior of the both of the NAND2 and NOR2 gates.

Under the presence of large within-die random variation, each delay path might behave differently. At a higher supply voltage, a particular path may show the

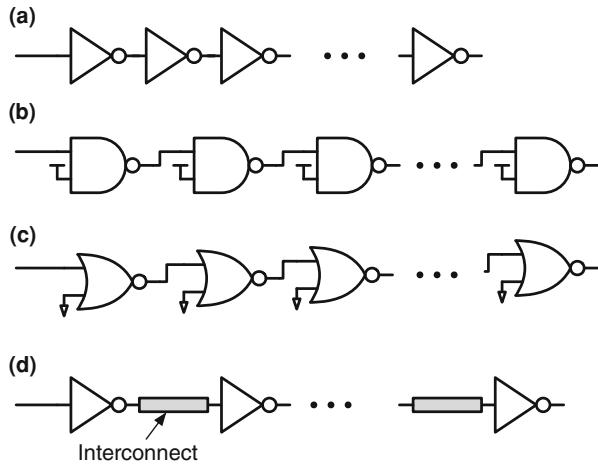


Fig. 4 Delay paths consisting of (a) inverter gates, (b) NAND2 gates, (c) NOR2 gates, and (d) inverter gates with long wires

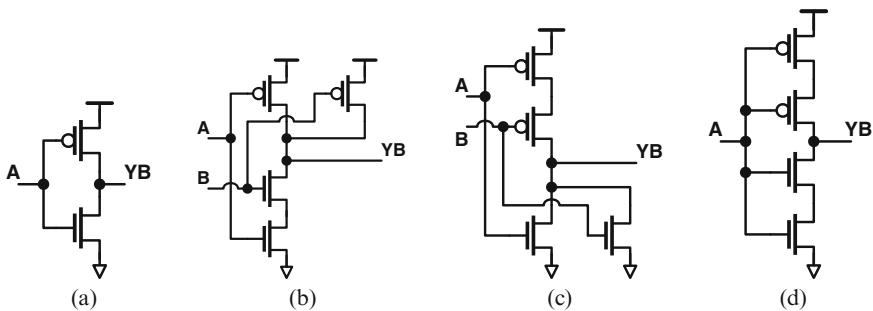
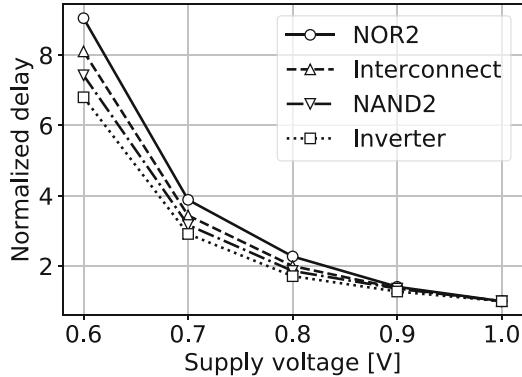


Fig. 5 Topology of different delay cells. (a) Inverter gate. (b) NAND2 gate. (c) NOR2 gate. (d) Universal delay cell

worst-case delay, whereas at a lower supply voltage, a different path may show the worst-case delay. Figure 6 shows the delay change against the change of supply voltage. Topology with a stacked transistor shows higher sensitivity to V_{dd} change than that without a stacked transistor. Topology with a reduced V_{gs} value shows much higher sensitivity to V_{dd} change. The important point is that the delays of different topologies show different sensitivities to process, supply voltage, and temperature changes. Under the presence of within-die variation, the gates of the same logic type also show different delay behavior. Thus, accurate delay estimation of a circuit is challenging. Instead, we can monitor the delay of a representative circuit that gives us a reasonable prediction of the actual delay of the circuit.

Fig. 6 Delay versus supply voltage for different inverter topologies



3.2 Delay Model

A delay model is useful to intuitively understand the different delay characteristics for different topology. The rise and fall delays of an inverter gate can be approximated by the following equations:

$$d_{\text{rise}} = \frac{C_{\text{load}} V_{\text{logic}}}{I_{\text{dp}}}, \quad (1)$$

$$d_{\text{fall}} = \frac{C_{\text{load}} (V_{\text{dd}} - V_{\text{logic}})}{I_{\text{dn}}}. \quad (2)$$

Here, I_{dp} and I_{dn} are the drain currents of pMOSFET and nMOSFET during the ON state, respectively. C_{load} is the load capacitance that consists of the gate capacitance of MOSFETs of the next gate, drain capacitance of pMOSFET and nMOSFET, and interconnect parasitic capacitance. V_{logic} is the logical threshold voltage at which the next gate switches its output value. To model the transistor drain current, EKV model based equation of Eq. 3 is useful to express the drain current that is continuous from weak-inversion to strong-inversion operation: [6, 7].

$$I_{\text{d}} = k \cdot \frac{W}{L} \cdot \ln^{\alpha} \left[1 + \exp \left(\frac{V_{\text{gs}} - (V_{\text{th}} - \gamma V_{\text{bs}} - \lambda V_{\text{ds}})}{\alpha n V_T} \right) \right]. \quad (3)$$

Here, k is a technology-related parameter. γ is the body bias coefficient and λ is the short-channel coefficient. Short-channel effect reduces the threshold voltage when large V_{ds} is applied to the transistor. Thus, large V_{ds} value increases ON current which is beneficial to switching delay, but causes exponential increase in the leakage current.

For the pull-down operation of an inverter gate of Fig. 5a, V_{bs} is zero and V_{ds} changes from V_{dd} to V_{logic} . However, in the case of a NAND2 gate, the values of V_{bs} and V_{ds} differ. The source of the nMOSFET that is connected to the output is

not tied to ground. As a result, V_{bs} becomes negative that causes the V_{th} to increase. Consequently, the V_{ds} value remains within a small value. Smaller V_{ds} value causes less short-channel effect resulting in a higher V_{th} value than a larger V_{ds} value. As a result of negative V_{bs} value and smaller V_{ds} value, the drain current decreases which causes the delay to increase.

3.3 Delay-Based Monitor Circuits

Design of on-chip monitors requires careful choosing of the right topology. Here, we discuss several delay-based design techniques that realize monitoring of different parameters.

3.3.1 Critical Path Monitor

The first and the most important parameter to monitor is the maximum delay of a circuit to ensure that the circuit operates at a certain clock frequency without any timing error. The maximum delay of a circuit is the maximum of delays of all the paths. As a circuit consists of thousands of delay paths, we can choose the following two methods to monitor the maximum delay.

1. Monitor the delays of actual paths, and
2. Monitor the delay of a representative delay path.

The first method, which is in-situ monitoring, requires additional circuitry in the actual delay paths. In the case of in-situ monitors, the Flip-Flops (FF) in a circuit are replaced with special FFs with error detection sequential (EDS) functions. The EDS can either detect whether a timing error has occurred [8, 9] or warn us before the occurrence of actual errors [10–12]. Supply voltage and clock frequency are adapted accordingly based on the EDS signals. The drawback of EDS-based in-situ monitors is that the additional circuits add extra delays, and increase area and power. To reduce the delay and area overhead, we can replace only those FFs where the delays are critical. During the design phase, we can make a list of the potential critical delay paths. However, as shown in Fig. 6, paths show different sensitivity to process, supply and temperature changes. Thus, the number of candidates tend to increase drastically under process, voltage, and temperature variations. Another fundamental drawback to be overcame is that a critical path is not always sensitized. Thus, it is necessary to properly estimate the actual timing slack of the critical path.

The second method requires an additional delay path that is placed near the actual circuit that can track the delay of the actual circuit. This delay path is often called a critical path monitor (CPM). The requirement of such a CPM is that it tracks the maximum delay of the target circuit for all conditions of process, voltage, and temperature variations. CPM is thus a delay path that is synthesized such that it tracks the worst delay of the circuit. However, there is no universal solution on how

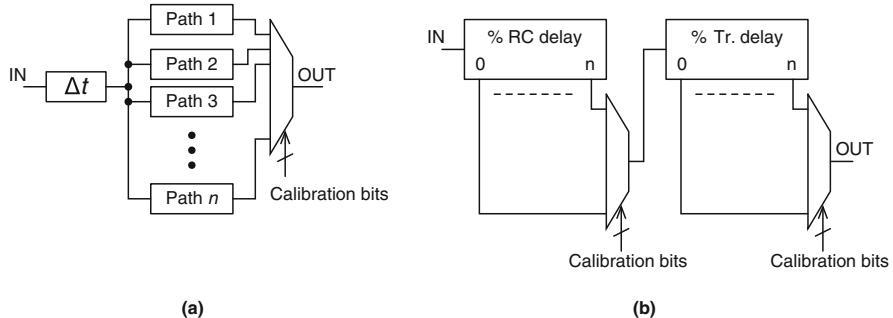


Fig. 7 Synthesis of critical delay path from a combination of series and parallel delay paths. **(a)** Parallel paths. **(b)** Series paths

to design a CPM that meets the above criterion. Two approaches have been proposed on how to synthesize a CPM. One approach is to synthesize a critical path monitor from a list of potential critical paths during the design phase [13–15]. The other approach is to design a reconfigurable delay path consisting of different logic gates and wire lengths, and then configuring the delay path during the test time, such that the delay correlates with the maximum achievable frequency [16–19]. Figure 7 shows a general concept of the synthesis framework of a critical delay path [20]. Several paths such as the paths shown in Fig. 4 are put in parallel. Then the several paths are placed in series. During the calibration process, combinations of parallel and series paths are explored to find a combination that gives the worst delay for all the operating conditions.

Instead of using a reconfigurable delay line, a general purpose delay line consisting of inverter cells with stacked transistors are also proposed so that the path mimics the worst-case delay [21]. Calibration is nonetheless required which can be performed during the design phase and during the test. To encounter the effect of systematic within-die variations, multiple CPMs can be used that are distributed at various places on the chip [15, 21].

3.3.2 Threshold Voltage Monitor

For adaptation of V_{th} values to their optimum values, V_{th} monitors are required. Although there is no universal definition of V_{th} , an arbitrary definition can be used as a reference. For example, the V_{gs} value that gives a fixed I_d value is often used to define the V_{th} value. Conversely, we can track the V_{th} value by observing the change of I_d value if the V_{gs} can be set as a function of V_{th} . Then the delay change resulting from the I_d change can be measured and converted to digital with the use of a reference clock signal. Figures 8 and 9 show two delay cells consisting of inverter gates where either the nMOSFET V_{gs} or the pMOSFET V_{gs} voltage becomes a function of the corresponding V_{th} values (V_{thp} for pMOSFET and V_{thn} for

Fig. 8 V_{thp} -dominant delay cell for V_{thp} monitoring

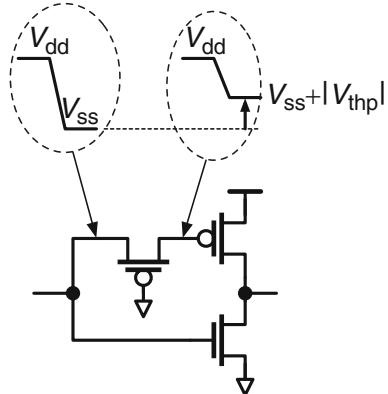
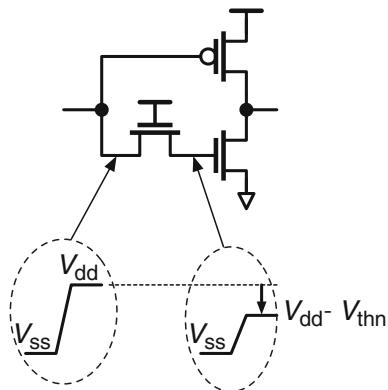


Fig. 9 V_{thn} -dominant delay cell for V_{thn} monitoring



nMOSFET). The V_{th} -sensitive gate-source voltage is realized using pass-transistors as shown in Figs. 8 and 9 [22, 23]. To illustrate the V_{th} monitoring capability of the monitor cells, sensitivity vectors of different inverter topologies are shown in Fig. 10 at nominal supply voltage for a 65 nm bulk process. Here, the sensitivity vector consists of the sensitivity coefficients of the delay to V_{thn} and V_{thp} changes. We observe that the sensitivity coefficients of the pass-transistor inserted cells are multiple times larger than those of conventional inverter, NAND2, and NOR2 cells. An all-digital process variability monitor based on a shared structure of a buffer ring and a ring oscillator is proposed in [24]. The technique utilizes the differences of rise and fall delays of inverter gates because of process variations.

As will be shown next, driving the load with the transistor leakage current also gives us a delay that is exponentially related to V_{th} value change. However, driving the load using leakage current requires careful design because leakage currents through the pull-up and the pull-down paths get involved also. Gate-leakage current is also a factor to degrade the accuracy of such monitors. The topologies of Figs. 8 and 9 give us compact designs that are minimal and fulfill the purpose.

Fig. 10 Sensitivity vectors of different inverter topologies

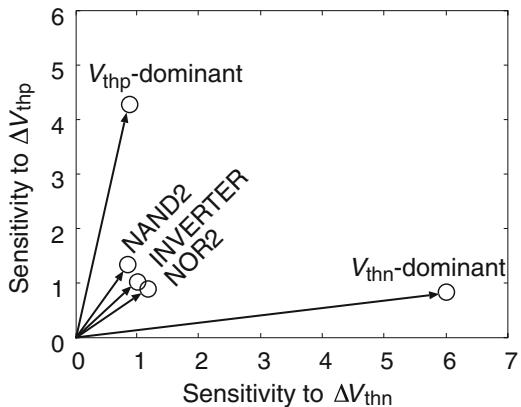
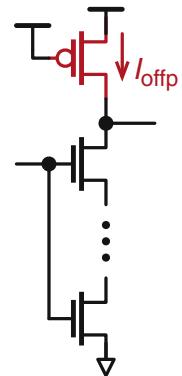


Fig. 11 I_{offp} -dominated delay cell



3.3.3 Aging Monitor

A critical path monitor also acts as an aging monitor. However, the differences in activity rate may cause deviations in the aging between an actual critical path and a monitor path. Therefore, delay paths of different activity rates can be implemented to track aging. Multiple delay lines consisting of inverter gates with different activity would give us precise aging information. Decoupling the NBTI and PBTI effects can be useful for debugging and modeling purposes. In that case, different architectures are proposed for independent NBTI and PBTI monitoring [25].

3.3.4 Sub-threshold Leakage Monitor

Sub-threshold leakage monitor helps us to estimate the leakage current of a circuit. The information can then be used to tune V_{th} , V_{dd} or frequency optimally. Figure 11 shows a delay cell whose rise delay is several orders of magnitude larger than the fall delay. The rise delay is driven by the pMOSFET OFF current, while the fall

Fig. 12 I_{offn} -dominated delay cell

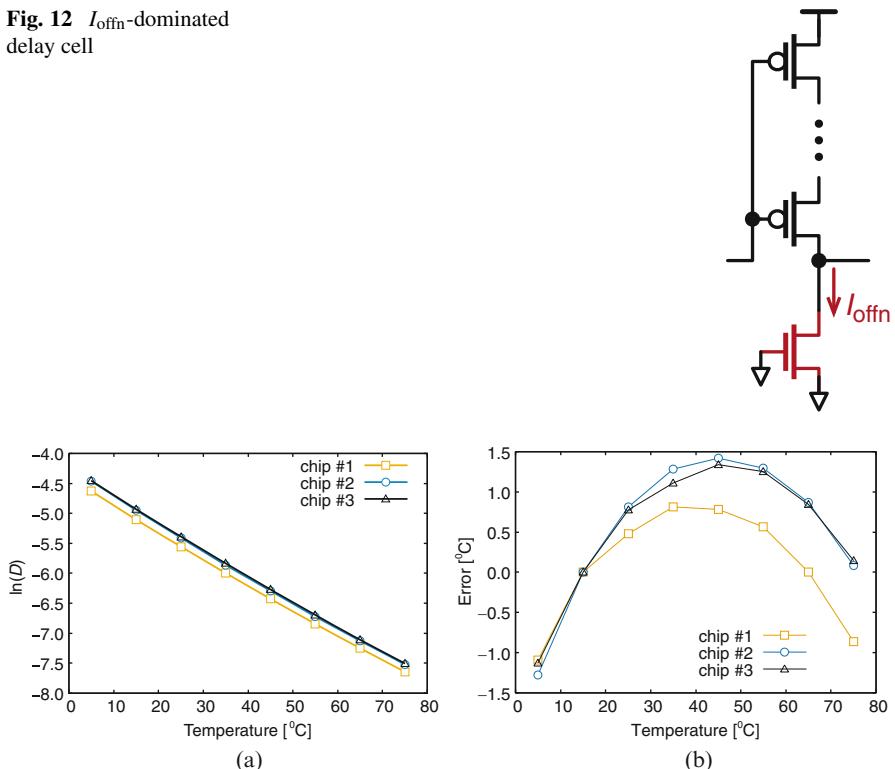


Fig. 13 Temperature monitoring utilizing inverter delay driven by nMOSFET OFF current. (a) Logarithm of oscillation period driven by nMOSFET OFF current against temperature. (b) Monitoring error against temperature after an one-point calibration

delay is driven by the nMOSFET ON current. As a result, the delay of a path consisting of this cell is proportional to the pMOSFET OFF current. Similarly, delay of a path consisting of delay cells of Fig. 12 is proportional to the nMOSFET OFF current. Figure 13a shows the change of measured oscillation period for a delay path consisting of 125 inverter stages against the temperature change. The inverter topology of Fig. 12 is used here. The target process is a 65 nm bulk process. The oscillation period here corresponds to the average OFF current of 125 nMOSFETs. The logarithm of the delay changes linearly with the temperature showing that the monitor tracks the leakage current change correctly.

3.3.5 Temperature Monitor

As leakage current is sensitive to temperature variation, a leakage current monitor can be used for on-chip temperature monitoring. The logarithm of the oscillation period, D , can be expressed by the following equation:

$$\ln(D) = a_T + b_T \cdot T, \quad (4)$$

where T is the absolute temperature, a_T and b_T are temperature coefficients. Figure 13b shows the monitoring error after a one-point calibration for a 65 nm bulk process. Calibration is performed at 15 °C. An error range of −1.3 °C to 1.4 °C is observed. The above error range is small enough for real-time thermal and reliability management.

3.3.6 Supply Voltage Monitor

Supply voltage fluctuation has always been a concern which is getting more severe with the reduction of supply voltage. Supply voltage fluctuation has a static component which results from the power delivery network (PDN) and a dynamic component which is the result of transition from idle to active state of a circuit. As critical path monitors are also sensitive to supply voltage fluctuations and have a high bandwidth, they can also detect dynamic supply voltage fluctuations [17]. In the case of CPMs, the output is the timing information obtained by comparing the path delay and clock period. Thus, the error information does not give whether the error is from temperature or supply voltage for example. However, when combined with other monitors such as temperature and threshold voltage, identification of the causes of timing error becomes possible. The identification of the sources of timing error allows correct optimization and lifetime enhancement. On-chip supply voltage droop monitoring mechanisms have been proposed to evaluate the power delivery network (PDN) [26].

3.3.7 Activity Monitor

Run-time estimation of the static and the dynamic energy can be used to achieve the minimum energy operation as suggested by Fig. 3. As the dynamic energy is proportional to circuit activity rate, we can estimate the dynamic energy by calculating the activity rate of a circuit. A digital dynamic power meter (DDPM) has been used that computes a rolling average of signal activity over a fixed number of clock cycles [27]. The accuracy of the power estimation here depends on careful selection of signals, such that they correspond to the activity of structures that have high power consumption. Instead of monitoring key logic signals, a clock activity adder (CAA) for switching power estimation is also proposed [28]. The approach of the CAA takes advantage of the fact that switching power is highly correlated to register clock activity. Similarly, hardware-event monitors such as memory-access counters and instruction-execution counters can be used for dynamic energy estimation [29]. These monitors depend on counting signal transitions rather than the delay itself.

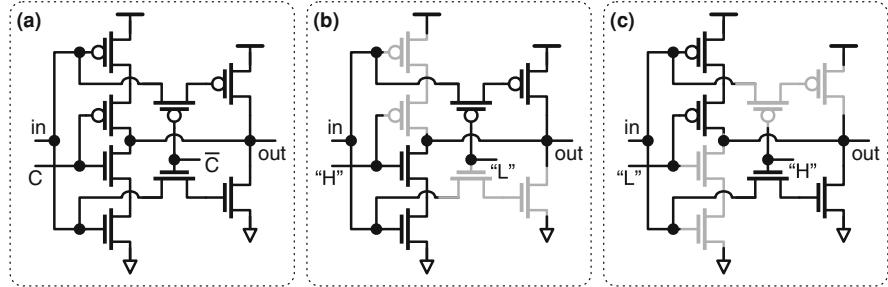


Fig. 14 A reconfigurable inverter cell topology for V_{thp} and V_{thn} monitoring. “C” is a control signal. (a) Reconfigurable topology. (b) V_{thp} -sensitive configuration, and (c) V_{thn} -sensitive configuration

3.4 Reconfigurable Delay Path for Multiple Parameter Monitoring

Delay-based sensing enables us to design a reconfigurable architecture to monitor multiple parameters by configuring the delay path accordingly [30, 31]. For example, we can use the topology of Fig. 14a to monitor both of the V_{thp} and V_{thn} variations. Figure 14b and c shows the two configurations to make the delay V_{thp} - and V_{thn} -sensitive, respectively.

3.5 Cell-Based Design

The use of delay cells provides the advantage of the use of cell-based design flow that enables us to place and distribute the monitors into different parts of the chip. For example, temperature monitors need to be placed at hot-spots where power density is high. Power density maps are generated during the design phase. A cell-based design example in a 65 nm bulk triple-well process for a reconfigurable V_{th} monitoring circuit is shown in Fig. 15. The cells with green highlights in Fig. 15b are the monitor cells of Fig. 15a. The placements of the cells are performed carefully utilizing the “do not touch” and “relative adjacent placement” features of the place and route tool.

3.6 On-Chip Measurement and System Interface

The monitoring circuit needs to be interfaced with system for adaptation and self-tuning. The following three mechanisms can be adopted for on-chip measurement of monitor circuits.

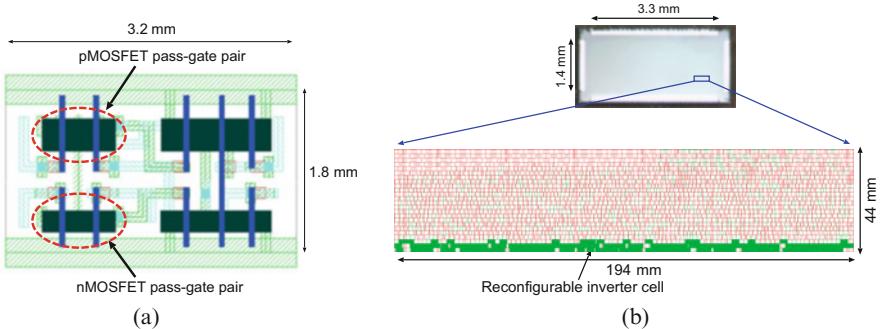


Fig. 15 Delay characteristics for different topology and supply voltage. (a) Cell layout of a reconfigurable V_{th} monitor delay cell. (b) Chip micrograph and layout of a reconfigurable monitor circuit including the controller

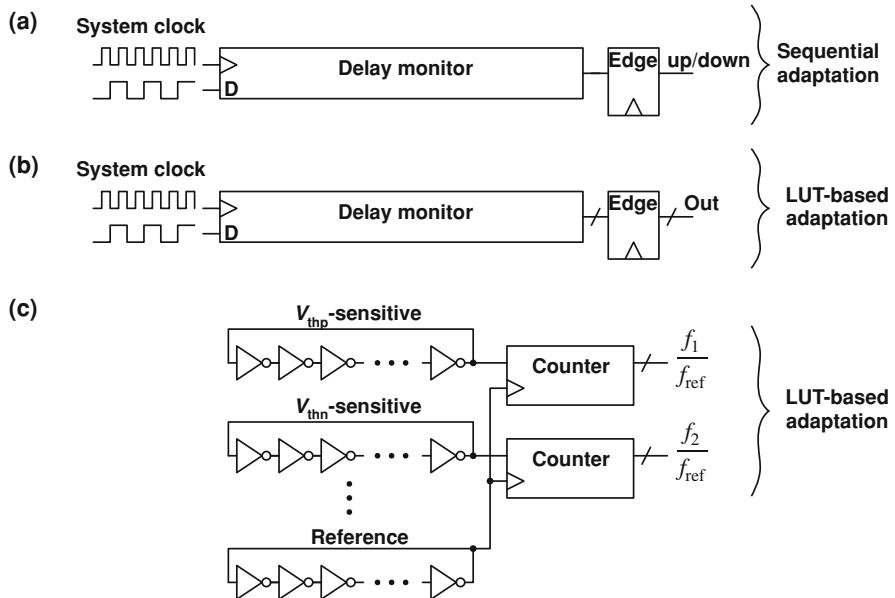


Fig. 16 Three different measurement methods for system interfacing. (a) EDS-based method. (b) Time-to-digital conversion based method, and (c) Frequency ratio based method

1. Edge detection [16, 17, 32].
2. Frequency counting [33].

Edge detection based system can have either a single bit output [16] or multiple bits output [17, 32]. Figure 16 shows three different methods for digitizing the monitored delay. Figure 16a checks whether the delay is smaller or larger than the system clock period [17, 32]. If the delay is smaller, adaptation such as slowing

down the system by reducing the supply voltage can be performed. If the delay is larger, the system will speed up by increasing the supply voltage for example. To ensure that the transitions occur without any timing error, margins are added in the delay path. These margins include within-die random delay effects as well as the response time of the adaptation. A resolution window can also be added to ensure that the adaptation occurs without inducing any timing error.

Figure 16b uses multiple edge detectors to convert the time between the path delay and the clock period to digital codes [16]. The digital codes are then sent to the system controller where a look-up table (LUT) based adaptation can be implemented. Figure 16c shows a measurement method that uses frequency counting [33]. This measurement method is particularly useful for monitoring device parameters of V_{th} , temperature, and so on. Using the system clock for the conversion will require calibration of the monitoring circuit for every supply voltage which will increase the test cost. Instead, we can utilize a locally generated clock using a ring oscillator. The output in this case is the ratio of the monitor frequency and the reference frequency. The measured values of the frequency ratio are then compared with predefined values to monitor how much the monitoring parameter varied from the targeted values. For applications where the clock frequency is fixed, process and temperature sensitive monitors can also be implemented with edge detection mechanisms. An up/down counter based detection circuit to detect the V_{th} deviation from predefined values has been employed for dynamic adaptation of V_{th} values [34].

4 Parameter Extraction for Model-Hardware Correlation

The circuit techniques described in Sect. 3 realize delay characteristics that are sensitive to particular parameter variations. However, they do not give us the value of the parameter variation itself. In this section, we describe a parameter extraction technique that takes the delay values of multiple delay paths and then estimates the variations in each of the parameters. The parameters can be transistor threshold voltage, temperature, gate-length or any device related parameter. We can then utilize the extracted parameters for test strategies and process optimization.

4.1 Parameter Extraction Methodology

In the case of an inverter gate, the gate–source voltage of each transistor goes through different values during a “High” to “Low” and a “Low” to “High” switching events. Thus, it is not straight forward to relate physical device parameters to the delay information. Parameter estimation gets harder when the supply voltage is lowered, such that the delay becomes non-linear to the parameter changes. So, the question is how to correlate the model to each chip to get a good accuracy.

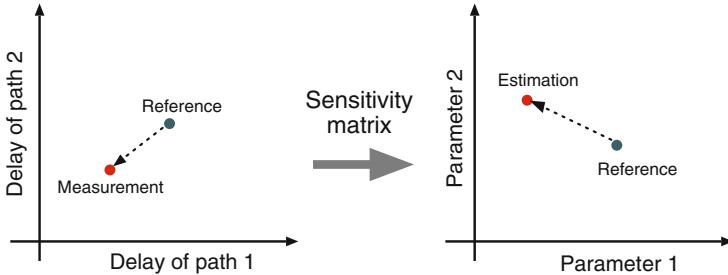


Fig. 17 Estimation of physical parameters from multiple delay paths. Sensitivity coefficient links the physical parameters to delay values

Section 3 demonstrates different types of delay paths to monitor various physical and environmental parameters. The designs are carefully performed to make the delay particularly sensitive to the parameter of interest. The techniques allow us to comparatively track the change of the parameters in the run-time. However, because of the mismatch in the model and hardware, the absolute parameter monitoring contains errors. Calibrations need to be performed to reduce the errors to acceptable ranges. For debugging purposes, we may want to correlate our transistor models to actual transistor characteristics in the chip. To perform model-hardware correlation, key model parameters such as the V_{th} and β may suffice as they are the dominant sources of fluctuations, although other parameters may also be used.

Figure 17 illustrates the concept of parameter extraction from multiple delay values. The left side of the figure plots the delay of a path against the delay of a different path. The round point shows a point which is obtained by circuit simulation. The cross point emulates a measured value from a chip. The difference between the two points here contains process information. Using sensitivity coefficients, we can estimate the amount of deviation in the process parameters and transform the delay space to process space which is shown in the right side of the figure. The key point here is not to use transistor I-V characteristics, rather use the delay characteristics to extract these parameters. For robust extraction of the parameters, we need to design the delay paths, such that the sensitivity matrix has a low condition number [22]. We can then build a system of linear equations using the sensitivity coefficients.

4.2 Measurement Results

To demonstrate the monitoring capability of the V_{thp} -sensitive and V_{thn} -sensitive delay cells of Figs. 8 and 9, measurements of ring oscillators are performed for a 65 nm bulk process. Figure 18 plots the values of V_{thp} and V_{thn} estimated under different body bias conditions for a particular chip. In the figure, the x-axis refers to V_{thp} estimations and the y-axis refers to V_{thn} estimations. Rectangular points are

Fig. 18 Estimation results of V_{thp} and V_{thn} of a chip for different body bias values. Either the pMOSFETs or the nMOSFETs are biased simultaneously

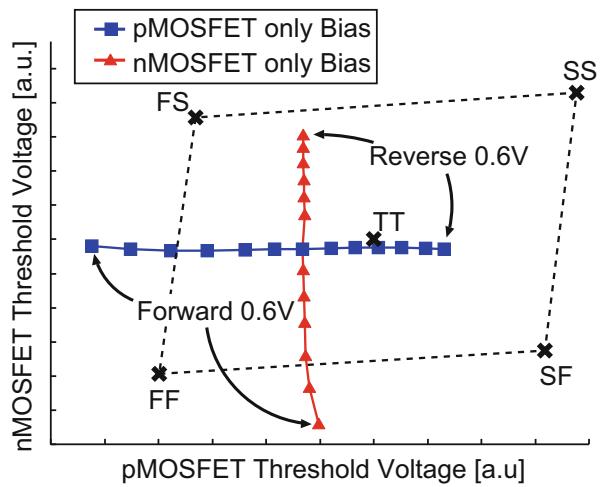
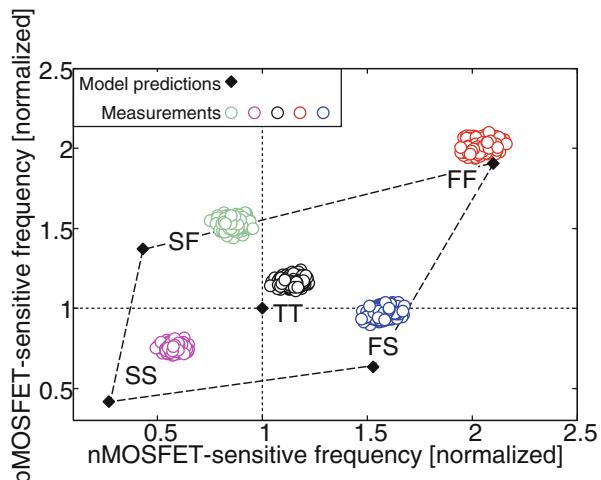


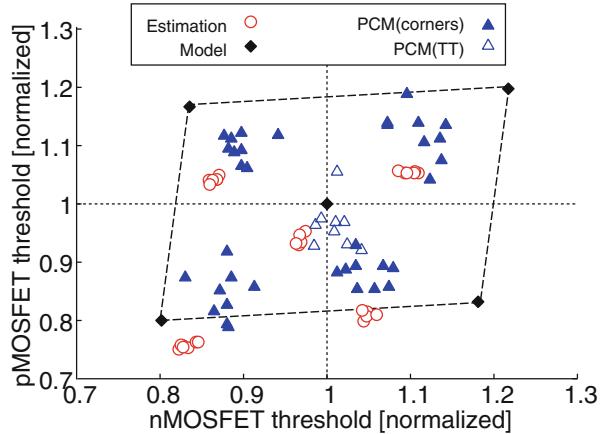
Fig. 19 V_{thp} -sensitive RO (ring oscillator) frequencies against V_{thn} -sensitive RO frequencies



estimated values of V_{thp} and V_{thn} when only pMOSFET is biased. Triangular points refer to estimated values of V_{thp} and V_{thn} when only nMOSFET is biased. When only pMOSFET is biased, the estimated point moves in the horizontal direction referring that only V_{thp} is being changed in the estimation. When only nMOSFET is biased, the estimated point moves in the vertical direction referring that only V_{thn} is being changed in the estimation. Thus, it is demonstrated that any change in the threshold voltage can be detected correctly by the proposed monitor circuits.

Figure 19 shows the measured frequencies of V_{thp} -sensitive and V_{thn} -sensitive ring oscillators from several chips (open circles). The chips have been fabricated targeting either of the five process corners of “TT,” “SS,” “FF,” “FS,” and “SF.” The values are normalized by the values simulated with the transistor models targeted for the “TT” process corner. Frequency values simulated using the other corner models

Fig. 20 V_{th} estimation results for different corner chips



are also plotted in the figure (closed squares). In the figure, process shifts from the “TT” model prediction are observed. Clear deviations are observed for “TT,” “SS,” “SF,” and “FS” corners. The silicon values are higher than the model predictions. With comparison with the models, we can have quick understanding of process shift for each chip. This information allow us to take decisions for silicon debug and test pattern generation. We can now extract the device parameters of V_{thp} , V_{thn} , and β using sensitivity analysis, model-hardware correlation can be obtained that allows us accurately predict the delay performance. Figure 20 plots the estimated V_{thp} and V_{thn} values. V_{th} values provided in the corner models are also plotted in the figure. Furthermore, V_{th} values provided by the Process Control Modules (PCM) that are generally placed in the scribe-lines are also plotted. The estimated values correlate with the PCM data and also show die-to-die variations.

5 Conclusion

In this chapter, we have shown the importance of cross-layer resiliency for energy-efficient and robust operation of circuits. Cross-layer resiliency is performed by tuning the threshold voltage and supply voltage in run-time based on information of process, leakage current, circuit activity, and temperature. Run-time monitoring of these parameters are essential in achieving cross-layer resiliency. To incorporate the monitor circuits into a cell-based design flow, we have discussed delay-based monitoring techniques. Cell-based design of monitor circuits enables to place the monitors inside the circuit. Placing the monitors inside the target circuit realizes better correlations between the monitor behavior and the actual circuit behavior.

We have discussed a general design methodology to synthesize a critical path monitor. There are several methods to monitoring the critical delay having a trade-off relationship between accuracy and implementation cost. Implementation cost

here can be area overhead, test cost, and/or both. The selection of a suitable critical path monitor thus has to be made based on the critical nature of the application.

Besides the critical path monitoring, run-time monitoring of physical parameters of threshold voltage, temperature, and leakage current are essential for energy-efficient operation under parameter fluctuation and aging. Utilizing the relationship between the delay of a logic gate and the physical parameters, several circuit topologies are discussed that amplify the effect a certain parameter. Threshold-dominant inverter topologies and leakage current driven inverters are suitable for temperature, leakage current, and threshold voltage monitoring.

References

1. ITRS, International Technology Roadmap for Semiconductors. Technical Report [Online]. Available: <http://www.itrs.net>
2. Borkar, S., Karnik, T., Narendra, S., Tschanz, J., Keshavarzi, A., De, V.: Parameter variations and impact on circuits and microarchitecture. In: Design Automation Conference, pp. 338–342 (2003)
3. Paul, B.C., Kang, K., Kufluoglu, H., Alam, M.A., Roy, K.: Impact of NBTI on the temporal performance degradation of digital circuits. IEEE Electron Device Lett. **26**(8), 560–562 (2005)
4. Horowitz, M.: 1.1 Computing’s energy problem (and what we can do about it). In: IEEE International Solid-State Circuits Conference, pp. 10–14 (2014)
5. Bohr, M.: The new era of scaling in an SoC world. In: IEEE International Solid State Circuits Conference, pp. 23–28 (2009)
6. Enz, C.C., Krummenacher, F., Vittoz, E.A.: An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications. Analog Integr. Circ. Sig. Process **8**(1), 83–114 (1995)
7. Marr, B., Degnan, B., Hasler, P., Anderson, D.: Scaling energy per operation via an asynchronous pipeline. IEEE Trans. Very Large Scale Integr. VLSI Syst. **21**(1), 147–151 (2013)
8. Ernst, D., Kim, N.S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., Mudge, T., Ave, B., Arbor, A.: Razor: a low-power pipeline based on circuit-level timing speculation. In: IEEE/ACM International Symposium on Microarchitecture, pp. 7–18 (2003)
9. Das, S., Tokunaga, C., Pant, S., Ma, W.-H., Kalaiselvan, S., Lai, K., Bull, D.M., Blaauw, D.T.: RazorII: In situ error detection and correction for PVT and SER tolerance. IEEE J. Solid State Circuits **44**(1), 32–48 (2009)
10. T. Nakura, K. Nose, M. Mizuno, Fine-grain redundant logic using defect-prediction flip-flops. In: IEEE International Solid-State Circuits Conference, pp. 21–23 (2007)
11. Bowman, K.A., Tschanz, J.W., Kim, N.S., Lee, J.C., Wilkerson, C.B., Lu, S.L.L., Karnik, T., De, V.K.: Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. IEEE J. Solid State Circuits **44**(1), 49–63 (2009)
12. Das, B.P., Onodera, H.: Frequency-independent warning detection sequential for dynamic voltage and frequency scaling in ASICs. IEEE Trans. Very Large Scale Integr. VLSI Syst. **22**(12), 2535–2548 (2014)
13. Firouzi, F., Ye, F., Chakrabarty, K., Tahoori, M.B.: Aging-and variation-aware delay monitoring using representative critical path selection. ACM Trans. Des. Autom. Electron. Syst. **20**(3), 39:1–39:23 (2015)
14. Chan, T.-B., Gupta, P., Kahng, A. B., Lai, L.: Synthesis and analysis of design-dependent ring oscillator (DDRO) performance monitors. IEEE Trans. Very Large Scale Integr. VLSI Syst. **22**(10), 2117–2130 (2014)

15. Park, J., Abraham, J.A.: A fast, accurate and simple critical path monitor for improving Energy-Delay product in DVS systems. In: IEEE/ACM International Symposium on Low Power Electronics and Design, pp. 391–396 (2011)
16. Drake, A., Senger, R., Deogun, H., Carpenter, G., Ghiasi, S., Nguyen, T., James, N., Floyd, M., Pokala, V.: A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In: IEEE International Solid-State Circuits Conferencepp. 398–399 (2007)
17. Tschanz, J., Bowman, K., Walstra, S., Agostinelli, M., Karnik, T., De, V.: Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In: IEEE Symposium on VLSI Circuits, pp. 112–113 (2009)
18. Drake, A.J., Floyd, M.S., Willaman, R.L., Hathaway, D.J., Hernandez, J., Soja, C., Tiner, M.D., Carpenter, G.D., Senger, R.M.: Single-cycle, pulse-shaped critical path monitor in the POWER7+ microprocessor. In: Proceedings of the International Symposium on Low Power Electronics and Designpp. 193–198 (2013)
19. Tokunaga, C., Ryan, J.F., Karnik, T., Tschanz, J.W.: Resilient and adaptive circuits for voltage, temperature, and reliability guardband reduction. In: IEEE International Reliability Physics Symposium, pp. 1–5 (2014)
20. Drake, A.J., Senger, R.M., Singh, H., Carpenter, G.D., James, N.K.: Dynamic measurement of critical-path timing. In: IEEE International Conference on Integrated Circuit Design and Technology, pp. 249–252 (2008)
21. Ikenaga, Y., Nomura, M., Suenaga, S., Sonohara, H., Horikoshi, Y., Saito, T., Ohdaira, Y., Nishio, Y., Iwashita, T., Satou, M., Nishida, K., Nose, K., Noguchi, K., Hayashi, Y., Mizuno, M.: A 27% Active-Power-Reduced 40-nm CMOS multimedia SoC with adaptive voltage scaling using distributed universal delay lines. *IEEE J. Solid State Circuits* **47**(4), 832–840 (2012)
22. Islam, A.K.M.M., Tsuchiya, A., Kobayashi, K., Onodera, H.: Variation-sensitive monitor circuits for estimation of global process parameter variation. *IEEE Trans. Semicond. Manuf.* **25**(4), 571–580 (2012)
23. Fujimoto, S., Mahfuzul Islam, A.K., Matsumoto, T., Onodera, H.: Inhomogeneous ring oscillator for within-die variability and RTN characterization. *IEEE Trans. Semicond. Manuf.* **26**(3), 296–305 (2013)
24. Iizuka, T., Asada, K.: All-digital PMOS and NMOS process variability monitor utilizing shared buffer ring and ring oscillator. *IEICE Trans. Electron.* **E95-C**(4), 627–634 (2012)
25. Kim, T.T.H., Lu, P.F., Jenkins, K.A., Kim, C.H.: A ring-oscillator-based reliability monitor for isolated measurement of NBTI and PBTI in high-k/metal gate technology. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **23**(7), 1360–1364 (2015)
26. Nishizawa, S., Onodera, H.: A ring oscillator with calibration circuit for on-chip measurement of static IR-drop. *IEEE Trans. Semicond. Manuf.* **26**(3), 306–313 (2013)
27. Krishnaswamy, V., Brooks, J., Konstadinidis, G., Curtis, M., Pham, H., Turullols, S., Shin, J., Yifan, Y., Zhang, H.: Fine-grained adaptive power management of the SPARC m7 processor. In: IEEE International Solid-State Circuits Conference, pp. 1–3 (2015)
28. Mair, H.T., Gammie, G., Wang, A., Lagerquist, R., Chung, C., Gururaj Rao, S., Kao, P., Rajagopalan, A., Saha, A., Jain, A., et al.: A 20nm 2.5 GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC performance. In: IEEE International Solid-State Circuits Conference, pp. 76–77 (2016)
29. Hokimoto, S., Ishihara, T., Onodera, H.: Minimum energy point tracking using combined dynamic voltage scaling and adaptive body biasing. In: IEEE International System-on-Chip Conference, pp. 1–6 (2016)
30. Islam, A.K.M.M., Onodera, H.: Area-efficient reconfigurable ring oscillator for device and circuit level characterization of static and dynamic variations. *Jpn. J. Appl. Phys.* **53**(4S), 04EE08 (2014)
31. Islam, A.K.M.M., Shiomi, J., Ishihara, T., Onodera, H.: Wide-supply-range all-digital leakage variation sensor for on-chip process and temperature monitoring. *IEEE J. Solid State Circuits* **50**(11), 2475–2490 (2015)

32. Bowman, K., Tschanz, J., Lu, S., Aseron, P., Khellah, M., Raychowdhury, A., Geuskens, B., Tokunaga, C., Wilkerson, C., Karnik, T., et al.: A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE J. Solid State Circuits* **46**(1), 194–208 (2011)
33. Kim, Y., Shin, D., Lee, J., Lee, Y., Yoo, H.J.: A 0.55 V 1.1 mW artificial intelligence processor with on-chip PVT compensation for autonomous mobile robots. *IEEE Trans. Circuits Syst. Regul. Pap.* **65**(2), 567–580 (2018)
34. Mahfuzul, I., Kamae, N., Ishihara, T., Onodera, H.: A built-in self-adjustment scheme with adaptive body bias using P/N-sensitive digital monitor circuits. In: *IEEE Asian Solid State Circuits Conference*pp. 101–104 (2012)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Dealing with Aging and Yield in Scaled Technologies



Wei Ye, Mohamed Baker Alawieh, Che-Lun Hsu, Yibo Lin, and David Z. Pan

1 Introduction

The aging and yield issues arise with aggressive scaling of technologies and increasing design complexity [51, 53]. These issues impact the circuit performance and functionality throughout the product life cycles. The sources of aging and yield concerns lie in different aspects, getting more severe with technology scaling.

Modern VLSI designs have to cope with unreliable components and processes. Device aging and interconnect electromigration effects are likely to cause unexpected performance degradation and even malfunctions at the end of circuit life cycles. Meanwhile, process variations may lead to manufacturing defects and inconsistent device characterization, causing yield issues. Ignoring these effects leads short lifetime of designs and low yield, eventually increases the costs in volume production and maintenance.

Thus, for the robustness of VLSI design methodology and cycles, reliability and yield need to be accurately modeled, systematically optimized, and seamlessly integrated into the existing design flow. This chapter will survey critical aging and yield issues, and then review the state-of-the-art techniques to tackle them, including both modeling and optimization strategies which reside across the Physics and Circuit/Gate layers as part of the overall dependability scheme shown in Fig. 1. The strategies often involve synergistic cross-layer optimization due to the complicated VLSI design procedures nowadays. Novel modeling techniques leveraging machine learning are analyzed along with analytical optimization approaches.

W. Ye · M. B. Alawieh · C.-L. Hsu · D. Z. Pan (✉)

University of Texas at Austin, Austin, TX, US

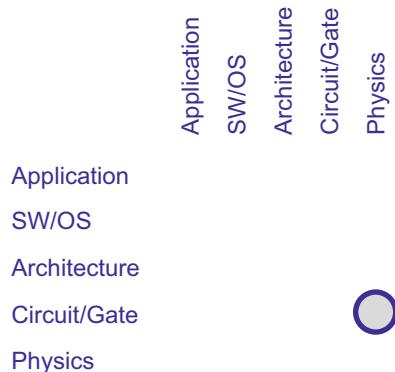
e-mail: weiy@utexas.edu; mohdbaker@utexas.edu; chsu1@utexas.edu; dpan@ece.utexas.edu

Y. Lin

Peking University, Beijing, China

e-mail: yibolin@pku.edu.cn

Fig. 1 This chapter covers reliability and yield issues across the borders between Physics and Circuit/Gate levels



The chapter starts by investigating the device and interconnect reliability issues for modern VLSI designs in Sect. 2. It covers different device aging effects, such as bias temperature instability and hot carrier injection, as well as electromigration effects on power/ground and signal interconnections. The section introduces the modeling techniques along with optimization strategies to increase the design robustness under these effects. Section 3 dives into the state-of-the-art practices in yield issues for both analog and digital circuits. This section examines the impacts of process variations on circuit performance and manufacturing defects followed by effective modeling techniques to capture these issues early in the design flow. In the end, the chapter is concluded with Sect. 4.

2 Reliability Modeling and Optimization

With the continued feature size shrinking, reliability issue becomes increasingly severe. This section covers recent researches on aging modeling and analysis and divide the aging concerns into two sub-categories: aging at the device level and aging at the interconnect level.

2.1 Device Aging

As CMOS technologies continue to shrink, device reliability becomes a major challenge for high performance computing (HPC) and automotive applications which require robust circuit design. This section presents the device reliability modeling and optimization techniques along with mitigation strategies in advanced CMOS technologies.

Device reliability can be divided into time-independent and time-dependent categories. Time-independent reliability issues are caused by manufacturing variations

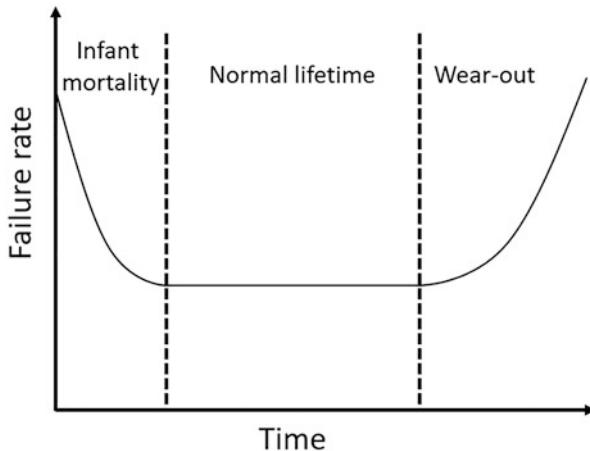


Fig. 2 Bathtub curve that illustrates the device life cycle

or noise such as random telegraph noise (RTN) or soft errors. Time-dependent reliability issues, also known as aging effects, can be illustrated using the bathtub curve in Fig. 2 which has high but decreasing failure rate in early life, low and constant failure rate in normal operation, and increasing high failure rate at the end of life wear-out period. This section focuses on modeling time-dependent reliability issues including *bias temperature instability* (BTI) and *hot carrier injection* (HCI).

BTI is an aging mechanism characterized by an increase in the device threshold voltage and a decrease in its mobility which eventually lead to an increase in the gate delay, and thus performance degradation [9, 58]. The two major factors contributing to the BTI phenomenon are the voltage bias and temperature. The term bias refers to the gate-to-source voltage bias applied to the transistor gate which is mostly a negative bias for PMOS, and a positive bias for NMOS. The theory behind BTI can be jointly explained by the reaction–diffusion (R-D) model and the charge trapping (CT) model [58]. The R-D model describes the degradation process when hole accumulation dissolves Si-H bond (reaction) and hydrogen diffuses away (diffusion), whereas the recovery stage takes place when voltage bias and duty factor stress is not present [25, 26]. The CT model explains the threshold voltage degradation by the trapped charge in the defected gate dielectrics. Early studies focused on BTI mitigation partially due to the fact that BTI dominates aging in early stages; however, HCI is more important at later stages where HCI contributes 40%–80% of device aging after 10 years of deployment [21, 47].

HCI is an aging phenomenon that degrades device drain current and is caused by the accumulation of carriers (electrons or holes) under the lateral electric fields, which can gain enough energy to damage and degrade the device mobility [16]. The traditional theory behind HCI was called lucky electron model, which is a field-based model [12, 57]. However, with the scaling of the supply voltage, the reduced electric field made HCI prediction based on field-based models a challenging task.

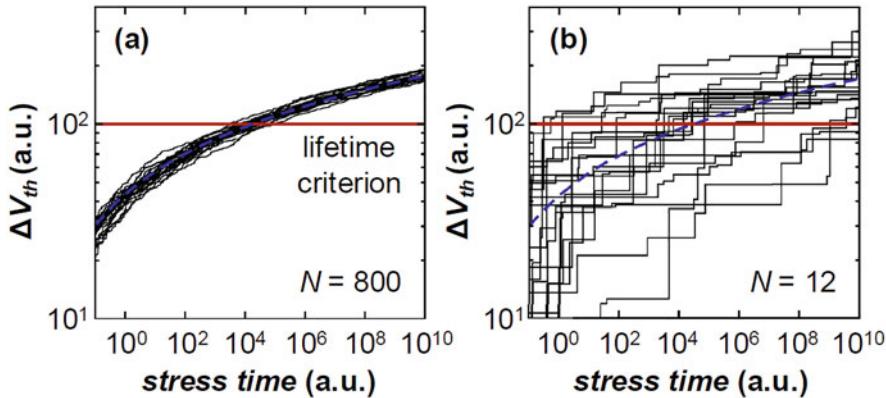


Fig. 3 BTI threshold voltage shift vs time for sub-45nm CMOS. BTI effect has stochastic nature in deep-sub-micro devices. Averaging each sample across large sample size of 800 (a) can achieve a well-defined voltage vs stress time curve while the voltage vs stress time trend in a much small sample size (b) contains larger variation [34]

Recent researches have proposed energy-driven theories to generalize HCI effects when devices are in low supply voltage [27, 52].

Characterizing aging degradation on circuit performance using aging model is a crucial step prior to optimization. Researchers can build deterministic models for BTI and HCI-related aging in old technologies such as 180 nm node. However, Kaczer et al. studied the threshold voltage shift vs time under the BTI effect and found its stochastic nature in deep-sub-micron nodes as shown in Fig. 3 [34]. Lorenz et al. proposed the first gate-level timing analysis considering NBTI and HCI [43]. Huard et al. [32] characterized a digital library gates under NBTI and HCI aging effects. Ren et al. discovered that BTI and HCI-related aging effects have layout dependencies [54]. In [21, 22], Fang et al. proposed frameworks to analyze BTI and HCI impacts on large digital circuits and [59] used ring oscillator-based sensors to estimate HCI/BTI induced circuit aging. Moreover, flip-flop based sensor was introduced in [2] to predict BTI aging circuit failure.

Recent researches not only model the aforementioned aging issues, but also propose design methods and optimizations for more reliable designs. Reliability optimization can be done at architecture level, logic synthesis level, and physical design level. At the architecture level, [48] demonstrated an aging analysis framework that examines NBTI and HCI to predict performance, power, and aging in the early design phase. Firouzi et al. [23] alleviated NBTI effects by using NOP (No operation) assignment and insertion in the MIPS processor. At synthesis level, Kumar et al. introduced standard cell mapping that considers signal probabilities to reduce BTI stress [35]. In [20], both HCI and BTI were considered during logic synthesis stage and put tighter timing constraint on paths with higher aging rate. Chakraborty et al. [13] optimized NBTI-induced clock skew in gated clock tree. Gate sizing [55, 64] and pin-reordering/logic restructuring [68] are also

implemented to minimize BTI effects. At the physical design level, Hsu et al. [29] proposed a layout-dependent aging mitigation framework for critical path timing during standard cell placement stage and [81] introduced aging-aware FPGA placement. Gate replacement techniques were used in [65] to co-optimize circuit aging and leakage.

2.2 Interconnect Electromigration

As IC technologies continue to scale, complex chip functionalities have been made possible by virtue of increasing transistor densities and aggressive scaling of interconnects. Besides, interconnects are getting thinner and running longer. These factors bring along higher current densities in metal wires, a phenomenon that further exacerbates *electromigration* (EM). The failure time from EM is worsened even further by the local temperature increase caused by self-heating of underlying FinFETs.

EM is the gradual displacement of atoms in metal under the influence of an applied electric field and is considered the primary failure mechanism for metal interconnects. After the migration of atoms with electrons in a metal line for a certain period, a void grows on one side, which increases the resistance of the metal line and may eventually lead to open circuits. Hillock is formed on the other side and may cause short circuits. Figure 4 shows the scanning electron microscopy (SEM) images of void and hillock.

2.2.1 Power EM Modeling

An empirical model for the mean time to failure (MTTF) of a metal line subjected to EM is given by Black's equation [11]

$$\text{MTTF} = \frac{A}{J^n} \exp\left(\frac{E_a}{kT}\right), \quad (1)$$

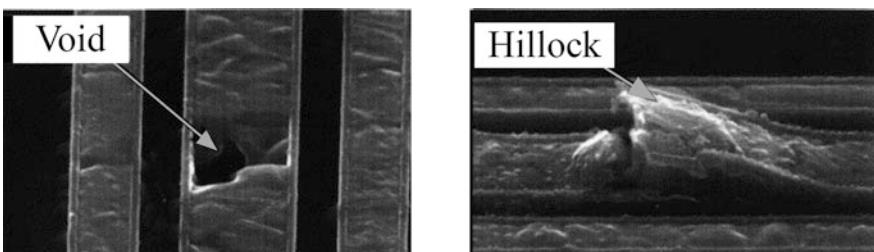


Fig. 4 A void and a hillock generated by electromigration [10]

where A is a constant which comprises the material properties and the geometry of the interconnect, J is the current density, E_a is the activation energy, k is the Boltzmann constant, and T is the temperature. n is the constant exponent of the current density and is usually set to 2. With Black's equation in Eq. (1), the relation between interconnect lifetime and both current and temperature can be readily estimated.

Power grid is one of the interconnect structures most vulnerable to EM due to its high unidirectional currents. Lower-level metal layers of power grids are more susceptible to EM failures due to smaller wire width. Besides, EM violations are most likely to occur around weak power grid connections, which deliver current to high power-consuming regions.

Hsu et al. [30] proposed an average power-based model to evaluate power grid static EM at placement stage. Ye et al. [78] further modified the model by considering the sum of the dynamic and leakage currents for a standard cell at this stage, which is given by:

$$I = \alpha \cdot C \cdot V_{DD} \cdot f + I_{\text{leak}},$$

where α is the cell activity factor, V_{DD} is the supply voltage, and f is the system clock frequency. C is the sum of the load capacitance and the output pin capacitance. Load capacitance further includes downstream gate capacitance and interconnect capacitance. Since nets have not been routed at this stage, half-perimeter wirelength (HPWL) [8] is widely adopted to estimate interconnect capacitance in placement. Power tile is defined as the region between two adjacent VDD (or VSS) power stripes and the adjacent power rails. Figure 5 demonstrates how to calculate the maximum current in the local power rails within a power tile. P_l and P_r are the left and right endpoints of the VDD power rail. d_i^l and d_i^r are the distances from the midpoint of the i -th cell to P_l and P_r , respectively. R_i^l and R_i^r are the wire resistances of the corresponding metal segments, which are proportional to d_i^l and d_i^r . The following equations hold

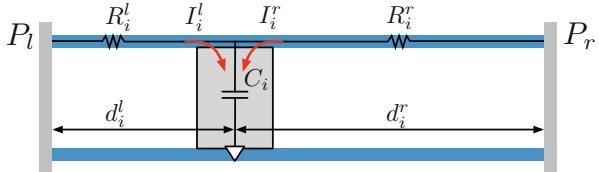
$$I_i^l = d_i^r / (d_i^l + d_i^r) I_i, \quad I_i^r = d_i^l / (d_i^l + d_i^r) I_i.$$

The currents drawn by all the cells in the power tile from P_l and P_r are computed as:

$$I^l = \sum_i I_i^l, \quad I^r = \sum_i I_i^r.$$

Therefore, there exists an EM violation in a particular power tile if $\max\{I^l, I^r\} > I_{\text{limit}}$. In this way, the EM failures in the local power rails can be estimated at the placement stage; thus, enabling an EM-aware placement that can effectively reduce the EM violations.

Fig. 5 The power grid model for current calculation in a power tile [78]



2.2.2 Signal EM Modeling

Previously, electromigration on signal interconnects does not draw great attention. Alternating current (AC) flows inside signal interconnects; when the direction of the current in an interconnect is reversed, the direction of EM diffusion is also reversed. The damage caused by EM can be partially cleared due to this compensation by material backflow. This effect is known as a self-healing, which can significantly extend the lifetime of a wire. Black's equation for AC is given by [40, 63]:

$$\text{MTTF} = \frac{A}{(J_+ - \gamma J_-)^n} \exp\left(\frac{E_a}{kT}\right), \quad (2)$$

where J_+ and J_- are the current densities during positive and negative pulses. γ is the self-healing coefficient which is determined by the duty factor of the current and other factors influencing the scale of self-healing, such as the frequency [39]. Previously, signal electromigration has attracted little attention due to the benefits of healing effect. However, EM failures in signal interconnects are no longer negligible due to higher clock frequencies, large transistor density, and the negative impact of FinFET self-heating at advanced nodes.

In [76], a set of features from the placement is extracted to train a machine learning model for EM detection before routing. Despite the fact that the current profile for the design is not available at the placement stage, multiple features that are highly correlated with the current can be crafted. These features can be divided into net-specific features and neighborhood related features. The net-specific features—including HPWL, the number of net pins, etc.—capture the net attributes. On the other hand, neighborhood related features are used to capture information about possible congestion around net pins.

The pre-routing signal EM hotspot prediction can be reduced to a classification problem [76]. A two-stage detection approach based on logistic regression shown in Fig. 6 is introduced to reduce the number of false alarms. In the first stage, a classification model M1 is trained to predict EM hotspots using all the nets in the training dataset. After the first stage, all nets with NH (Non-hotspot) prediction will be labeled as NH without further processing. For nets labeled H (Hotspot) by M1, a new model, M2, is trained to prune out false alarms. With an accurate classification model to detect signal EM hotspots based on the information available

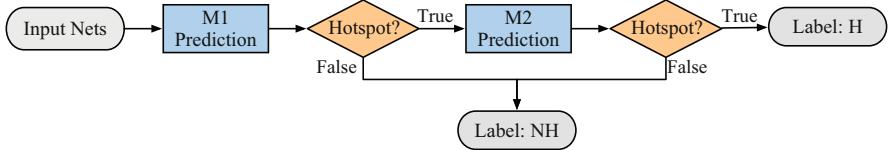


Fig. 6 The flow of the two-stage signal EM hotspot detection approach [76]

at the placement stage, early stage EM handling is enabled, which reduces iterative EM fixing cost.

2.2.3 EM Optimization Flow

Through the preceding EM modeling in Eqs.(1) and (2), EM failures can be detected after the physical design stage, and then be fixed through layout modification. Xie et al. [69] proposed control logics to balance currents in both directions of power rails to mitigate the EM effects. Lienig [38] suggested the exploitation of several EM inhibiting measures, such as bamboo structure, short-length, and reservoir effects. Other studies [14, 33] considered global routing for EM optimization. In [49] de Paris et al. adopted a design strategy using non-default routing (NDR) rules to re-route the wire segments of EM-unsafe signal nets that present high current densities.

Conventionally, EM checking is invoked after the routing stage [36]. Current densities in metal wires are computed and compared with foundry-specified limits to detect EM failures. Next, the failures are fixed with engineering change order (ECO) efforts. EM checking leverages post-routing information to detect violations, which consequently limits the efficiency of addressing techniques. In the routing phase, the locations of standard cells and the corresponding current distribution are already fixed and the traditional fixing approaches such as wire widening and cell resizing are not effective enough to handle the ever-growing number of EM violations [1]. It is of vital importance to incorporate EM detection and fixing techniques into earlier stages of physical design (PD).

Two clear benefits are associated with such early stage EM handling. First, the number of EM violations can be decreased further by using various techniques at different design stages. Second, introducing early stage mitigation techniques can help reduce the resulting overhead when compared to post-routing fixing techniques. Thus, moving the EM detection and resolving steps to earlier stages of the physical design can help in reducing runtime or the number of iterations needed for design closure. In [78], a series of detailed placement techniques was proposed to mitigate power grid EM. Ye et al. [76] proposed a multistage EM mitigation approach at placement and routing phases to address the problematic nets detected by the classification model.

3 Yield Modeling and Optimization

3.1 Performance Modeling

With technologies descending deep into the sub-micron spectrum, process variation manifests itself among the most prominent factors limiting the product yield of analog and mixed-signal (AMS) circuits. Thus, it is indispensable to consider this variation in the design flow of modern ICs [42]. Conventionally, performance modeling has been adopted to capture this variability through analytical models that can be used in various applications such as yield estimation and design optimization [4].

Given a set of samples, the performance model coefficients are conventionally obtained through least-squares regression (LSR). However, LSR can build accurate models only when the number of samples is much greater than the number of unknown coefficients. Thus, given the high dimensionality of the performance models in complex AMS circuit designs, the simulation cost for building accurate models can be exorbitant. Hence, most recent performance modeling techniques incorporate additional information about the model to reduce the number of simulations needed [3, 5, 7].

3.1.1 Sparse Modeling

Although the number of basis functions representing the process variability is large, a few of these basis functions are required to accurately model a specific performance of interest (POI). Hence, the vector of coefficients contains a small number of non-zero values corresponding to important basis functions [37]. This information can be incorporated in the modeling by constraining the number of non-zero coefficients in the final model.

While constraining the number of non-zero coefficients accurately reflects the sparse regression concept, the optimization problem is NP-hard. Besides heuristic approaches that select important basis functions in a greedy manner, Bayesian approaches have been widely applied to address this challenge [37]. In practice, a shrinking prior on the model coefficients is used to push their values close to zero. Examples of this include applying a Gaussian or Laplacian prior which results in Ridge and Lasso regression formulations, respectively. This allows incorporating sparse prior knowledge; however, such approaches do not perform explicit variable selection and they penalize high coefficients values by pushing all coefficients close to zero instead of selectively setting unimportant ones to zero.

On the other hand, a Bayesian spike and slab feature selection technique can be employed to efficiently build accurate performance models [7]. Spike and slab models explicitly partition variables into important and non-important, and then solve for the values of the important variables independently of the feature selection mechanism. A hierarchical Bayesian framework is utilized to determine both the

importance and value of the coefficients simultaneously. At its highest level, the hierarchy dictates that a particular coefficient is sampled from one of the two zero-mean prior Gaussian distributions: a low variance distribution centered around zero, referred to as the **spike**, and a large variance distribution, referred to as the **slab**.

This mixture of priors approach has demonstrated superior results compared to traditional sparse modeling schemes while also providing a feature selection framework that can easily select important features in the model [7].

3.1.2 Semi-Supervised Modeling

Traditionally, performance modeling has been approached from a purely supervised perspective. In other words, performance models were built by using labeled samples obtained through expensive simulations. However, as the complexity of designs increased, obtaining enough samples to build accurate models has become exorbitant. Recently, a new direction, derived from semi-supervised learning, has been explored to take advantage of unlabeled data to further improve the accuracy of performance modeling for AMS designs [3, 5].

In practice, the hierarchical structure of many AMS circuits can be leveraged to incorporate unlabeled data via Bayesian co-learning [5]. In particular, such an approach is composed of three major components. First, the entire circuit of interest is partitioned into multiple blocks based on the netlist hierarchy. Second, circuit-level performance models are built to map the block-level performance metrics to the PoI at the circuit level. Such a mapping is often low-dimensional; thus it can be accurately approximated by using a small number of simulation samples. Third, by combining the aforementioned low-dimensional models and an unlabeled data set, a complex, high-dimensional performance model for the PoI can be built based on semi-supervised learning.

To implement this modeling technique, a Bayesian inference is formulated to integrate the aforementioned three components, along with the prior knowledge on model coefficients, in a unified framework. Experimental results shown in [5] demonstrate that the proposed semi-supervised leaning approach can achieve up to $3.6\times$ speedup when compared to sparse regression-based approaches.

While many AMS circuits exhibit a hierarchical structure, this feature is not always present. Hence, a more general semi-supervised framework which makes no assumption about the AMS circuit structure is desirable [3]. This can be achieved by incorporating a co-learning technique that leverages multiple views of the process variability to efficiently build a performance model. The first is the device level variations such as ΔV_{TH} or Δw_{eff} , while the second view is the underlying set of independent random variables, referred to as process variables. Traditionally, performance modeling targets expressing the PoI as an analytical function of process variables; however, capitalizing on information provided by the device level variability as an alternative view can help efficiently build the performance model for the PoI [3].

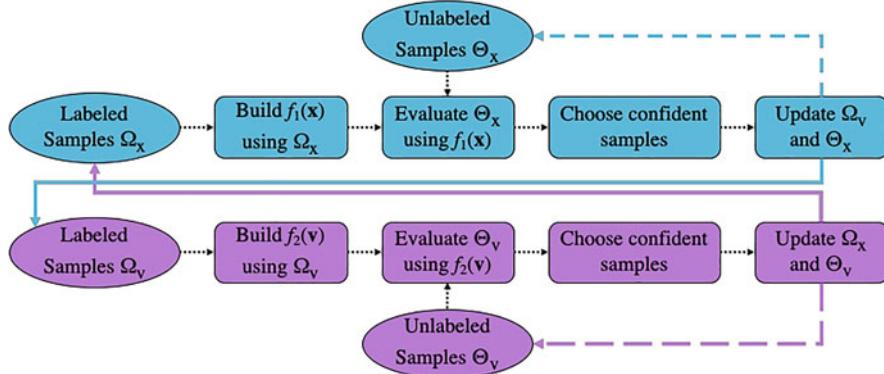


Fig. 7 An iteration of the semi-supervised co-learning modeling framework is illustrated [3]

As shown in Fig. 7, the key idea is to use a small number of labeled samples to build an initial model for each of the views of the data (\mathbf{x} and \mathbf{v}), then attempt to iteratively bootstrap from the initial models using unlabeled data. In other words, initial models can be used to give pseudo-labels for unlabeled data, then the most confident predictions from a particular model are used as pseudo-samples for the other model. In each iteration step, *highly confident* pseudo-samples are fused with the small number of available labeled samples to build a new model. Experimental results demonstrated up to 30% speedup compared to sparse regression-based approaches [3].

3.1.3 Performance Optimization

Besides capturing the major sources of variability in AMS designs, one of the main applications of performance modeling is yield estimation and optimization. In practice, performance optimization can make use of trained models towards optimizing the performance of the design. This is established by first capturing correlations between the performance variability and the device sizes or reconfiguration knobs, then adjusting these parameters to improve the parametric yield [4, 6].

Moreover, with the increase in AMS circuits complexity, increasing nonlinearity stands out as major factor limiting the capabilities of performance modeling and optimization. Hence, performance optimization techniques relying on non-parametric surrogate models and Bayesian optimization frameworks have been recently proposed [31, 83]. These surrogate models are typically Gaussian Processes, and Bayesian optimization is used to find optimal values given a black-box function.

Bayesian Optimization is a sequential sampling based optimization technique for optimizing block-box objective functions. At each step, a set of optimal sampling locations are selected based on a chosen acquisition function. Then, queries of the

objective function to be optimized, e.g. performance of an AMS circuit, which can be costly, are only made at these optimized locations, e.g. via circuit simulations for AMS verification. The new data collected at each step augments the training dataset to retrain a probabilistic surrogate model that approximates the black-box function. Such iterative sampling scheme contributes directly to the accuracy of the surrogate model and guides the iterative global optimization process [31, 83].

3.2 Hotspot Detection

As the feature size of semiconductor transistors continues shrinking, the gap between exploding design demands and semiconductor manufacturability using current mainstream 193 nm lithography is becoming wider. Various designs for manufacturability (DFM) techniques have been proposed; however, due to the complexity of lithography systems and process variation, failures to print specific patterns still happen, which are referred to as lithography hotspots. Examples of two hotspot patterns are shown in Fig. 8.

The hotspot detection problem is to locate the lithography hotspots on a given layout in physical design and verification stages. Conventional simulation-based hotspot detection often relies on accurate yet complicated lithography models and therefore is extremely time-consuming. Efficient and accurate lithography hotspot detection is more desired for layout finishing and design closure in advanced technology nodes.

Pattern matching and machine learning based techniques have been proposed for quick and accurate detection of hotspots. Pattern matching forms a predefined library of hotspot layout patterns, and then compares any new pattern with the patterns in the library [70, 79]. There are some extensions that use fuzzy pattern matching to increase the coverage of the library [41, 66]. However, pattern matching,

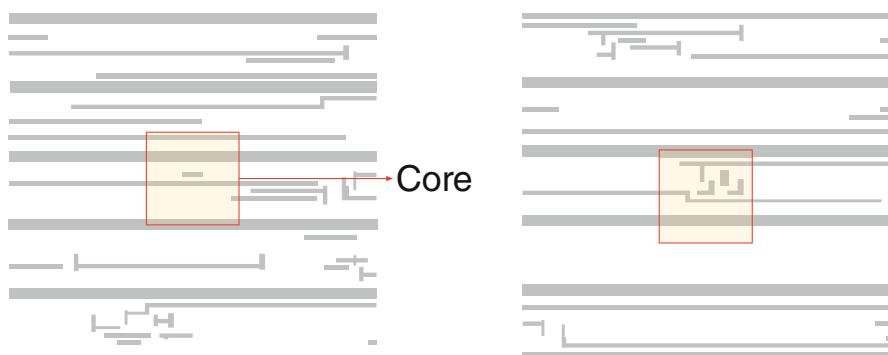


Fig. 8 Example of two hotspot patterns. Core corresponds to the central location where a hotspot appears

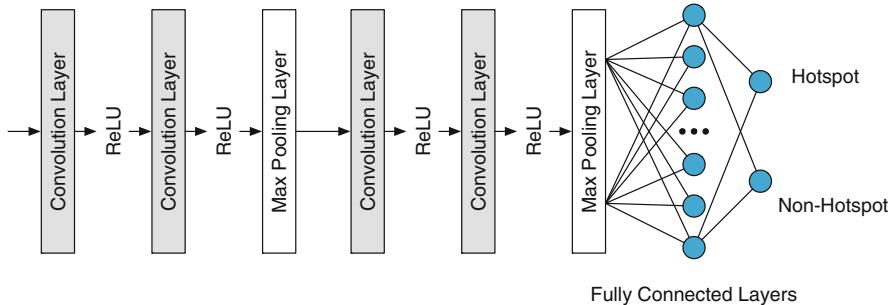


Fig. 9 An example of a neural network for hotspot detection [74]

including fuzzy pattern matching, is insufficient to handle never-before-seen hotspot patterns. Recently, machine learning based approaches have demonstrated good generalization capability to recognize unseen hotspot patterns [17, 18, 45, 50, 80, 82].

3.2.1 Lithography Hotspot Detection with Machine Learning Models

Various machine learning models have been used as hotspot detection kernels with the goal of achieving high accuracy and low false alarms, including support vector machine (SVM) [18, 80], artificial neural network (ANN) [18], and boosting methods [45, 82]. Zhang et al. [82] have also proposed an online learning scheme to verify newly detected hotspots and incrementally update the model. Recently, deep neural networks (DNNs) have been adopted for hotspot detection [46, 60]. DNNs are able to perform automatic feature extraction on the high-dimensional layout during training, which spares the efforts spent on manual feature extraction. Promising empirical results have been observed with DNNs in several papers [46, 60, 73, 74]. Figure 9 shows a typical configuration of the structure of a DNN.

The performance of DNNs usually relies heavily on manual efforts to tune the networks, e.g., the number and types of layers. Matsunawa et al. [46] proposed a DNN structure for hotspot detection that can achieve low false alarms. Yang et al. [74] proposed Discrete Cosine Transform (DCT) based feature representation to reduce the image size for DNNs with a biased learning to improve accuracy and decrease false alarms.

3.2.2 Evaluation of Hotspot Detection Models

One special characteristic of lithography hotspot detection tasks is the imbalance in the layout datasets. Those lithography defects are critical, but their relative number is significantly small across the whole chip. Among various machine learning

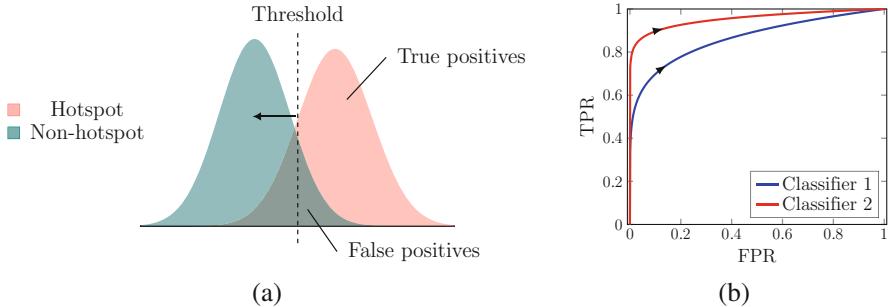


Fig. 10 (a) An overlapping distribution of predicted scores for positive and negative samples and (b) the ROC curves of two example classifiers. As the threshold in (a) moves to the left, both FPR and TPR in (b) go up accordingly [77]

models at hand, the one with a highest true positive rate (TPR) and a lowest false positive rate (FPR) is preferred, but in real-world scenarios, there is always a trade-off between the two metrics. As Fig. 10a demonstrates, if the predicted score implies the belief of the classifier that a sample belongs to the positive class, decreasing the decision threshold (i.e., moving the threshold to the left) will increase both TPRs and FPRs.

The receiver operating characteristic (ROC) curve is considered a robust performance evaluation and model selection metric for imbalanced learning problems. For each setting of the decision threshold of a binary classification model (Fig. 10a), a pair of TPR and FPR values is obtained. By varying the decision threshold over the range $[0, 1]$, the ROC curve plots the relationship between TPR and the FPR (Fig. 10b).

The area under the ROC curve (AUC) is a threshold-independent metric which measures the fraction of times a positive instance is ranked higher than a negative one [62]. The closer the curve is pulled towards the upper left corner, the better is the ability of the classifier to discriminate between the two classes. For example, in Fig. 10b, classifier 2 has a better performance compared to classifier 1. Given that AUC is a robust measure of classification performances especially for imbalanced problems, it is useful to devise algorithms that directly optimize this metric during the training phase.

It has been proven that AUC is equivalent to the Wilcoxon–Mann–Whitney (WMW) statistic test of ranks [28, 44, 67]. However, AUC defined by the WMW metric is a sum of indicator functions which is not differentiable, to which gradient-based optimization methods cannot be applied. In order to make the problem tractable, it is necessary to apply convex relaxation to the AUC by replacing the indicator function with pairwise convex surrogate loss function. There are different forms of surrogate functions: pairwise squared loss [19, 24], pairwise hinge loss [61, 84], pairwise logistic loss [56], and piecewise function given in [71]. Ye et al. [77] compare these surrogate functions and show that those new surrogate loss functions are promising to outperform the cross-entropy loss when applied to the state-of-the-art neural network model for hotspot detection.

3.2.3 Data Efficient Hotspot Detection

Despite the effective machine learning models for hotspot detection, most of them rely on a large amount of data for training, resulting in huge data preparation overhead. Thus, it is necessary to improve the data efficiency during model training, i.e., to achieve high accuracy with as small amount of data as possible.

Chen et al. [15] proposed to leverage the information in unlabeled data during model training, when the amount of labeled data is small. They develop a semi-supervised learning framework, using a multi-task network with two branches to train the classification task for hotspot detection and the other unsupervised clustering task at the same time. The network will label those unlabeled data samples with pseudo-labels in each iteration. The pseudo-labeled data will be selected and added to training with different weights in the next iteration, where the weights here are determined by the clustering branch. The experimental results demonstrate over 3–4% accuracy improvement with 10%–50% amount of labeled training data.

Sometimes, there is additional flexibility to the learning problem where labels for unlabeled data can be queried. This extra capability enables the use of active learning which can actively select the data samples for training a better model. Yang et al [72] propose to iteratively query the actual labels for unlabeled data samples with low classification confidence in each training step and add these samples for training in the next step. The experiments on ICCAD 2016 contest benchmarks show similar accuracy with only 17% of training data samples.

One should note that semi-supervised learning and active learning are two orthogonal approaches to tackle the insufficient of labeled training data. Semi-supervised learning assumes the availability of unlabeled data, while active learning assumes the capability of querying the labels for unlabeled data. They can even be combined to achieve better data efficiency [85].

3.2.4 Trustworthiness of Hotspot Detection Models

Conventionally, hotspot detection approaches have been evaluated by judging upon the detection accuracy and the false alarm rate. While these metrics are indeed important, model trustworthiness is yet another metric that is critical for adopting machine learning based approaches. Addressing this concern requires machine learning models to provide confidence guarantees alongside the label predictions.

In practice, methods for obtaining confidence guarantees when using deep neural network are costly and not yet mature. However, Bayesian-based methods are the typical option when confidence estimation is needed. This can be achieved by adopting a Gaussian Process (GP) based classification that can provide a confidence metric for each predicted instance. With this approach, a label from a trained model is only valid when its confidence level matches a user-defined metric, otherwise, the prediction is marked as untrusted and lithography simulation can be used to further verify the results [75].

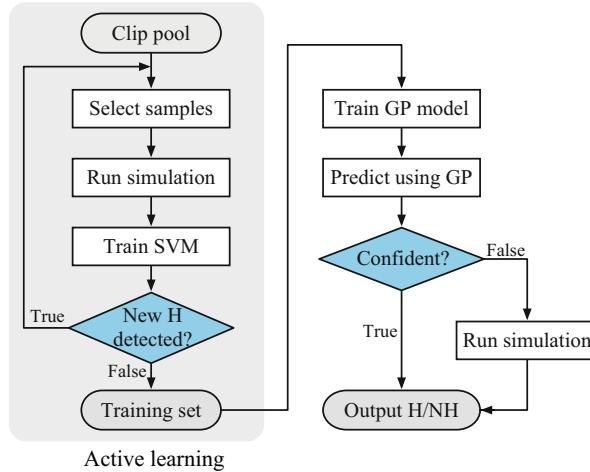


Fig. 11 Overall flow of Litho-GPA including data preparation with active sampling and hotspot detection with Gaussian process [75]

The flow of Litho-GPA, a framework for hotspot detection with Gaussian Process assurance, is illustrated in Fig. 11. In addition to addressing the issue of trust, Litho-GPA adopts active learning to reduce the amount of training data while favoring balance between classes in this dataset.

As a first step, an iterative weak classifier-based sampling scheme is leveraged to prepare a training set containing enough hotspots. Next, a Gaussian Process Regression (GPR) model is trained for the classification task with the selected data samples. This learned model is then used to make predictions with confidence estimation on the testing set. If GPR demonstrated high confidence in the predicted label, the result is trusted; otherwise, the unsure testing samples are verified with lithography simulations.

Experimental results shown in [75] demonstrate Litho-GPA can achieve comparable accuracy to the state-of-the-art deep learning approaches while obtaining on average 28% reduction in false alarms.

4 Conclusion

In this chapter, different important aging and yield issues in modern VLSI design and manufacturing have been discussed. These issues include device aging, interconnect electromigration, process variation, and manufacturing defects are likely to cause severe performance degradation or functionality failure, and thus need to be addressed early in the physical design flow. The chapter has surveyed recent techniques to not only build models for capturing these effects, but also to

develop strategies for optimizing them with the proposed models. These practices demonstrate that synergistic optimization and cross-layer feedback are encouraged to resolve the aforementioned aging and yield issues for robust VLSI design cycles.

References

1. Addressing signal electromigration (EM) in today's complex digital designs. <https://www.eetimes.com/document.asp?docid=1280370>
2. Agarwal, M., Paul, B.C., Zhang, M., Mitra, S.: Circuit failure prediction and its application to transistor aging. In: 25th IEEE VLSI Test Symposium (VTS'07), pp. 277–286 (2007)
3. Alawieh, M.B., Tang, X., Pan, D.: S²PM: semi-supervised learning for efficient performance modeling of analog and mixed signal circuit. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 268–273 (2019)
4. Alawieh, M.B., Wang, F., Kang, R., Li, X., Joshi, R.: Efficient analog circuit optimization using sparse regression and error margining. In: IEEE International Symposium on Quality Electronic Design (ISQED), pp. 410–415 (2016)
5. Alawieh, M.B., Wang, F., Li, X.: Efficient hierarchical performance modeling for analog and mixed-signal circuits via bayesian co-learning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**(12), 1–13 (2018)
6. Alawieh, M.B., Wang, F., Tao, J., Yin, S., Jun, M., Li, X., Mukherjee Tamal Negi, R.N.: Efficient programming of reconfigurable radio frequency (RF) systems. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 772–779 (2017)
7. Alawieh, M.B., Williamson, S.A., Pan, D.Z.: Rethinking sparsity in performance modeling for analog and mixed circuits using spike and slab models In: ACM/IEEE Design Automation Conference (DAC) (2019)
8. Alpert, C.J., Mehta, D.P., Sapatnekar, S.S.: *Handbook of Algorithms for Physical Design Automation*. CRC press, New York (2008)
9. Amrouch, H., Khaleghi, B., Gerstlauer, A., Henkel, J.: Reliability-aware design to suppress aging. In: ACM/IEEE Design Automation Conference (DAC), pp. 1–6 (2016)
10. Arnaud, L., Tartavel, G., Berger, T., Mariolle, D., Gobil, Y., Touet, I.: Microstructure and electromigration in copper damascene lines. *Microelectron. Reliab.* **40**(1), 77–86 (2000)
11. Black, J.R.: Electromigration—a brief survey and some recent results. *IEEE Trans. Electron Devices* **16**(4), 338–347 (1969)
12. C. Hu S. Tam, F.H.P.K.T.C., Terrill, K.W.: Hot-electron-induced MOSFET degradation—model, monitor, and improvement. *IEEE J. Solid-State Circuits* **20**(1), 295–305 (1985)
13. Chakraborty, A., Ganeshan, G., Rajaram, A., Pan, D.Z.: Analysis and optimization of NBTI induced clock skew in gated clock trees. In: IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE), pp. 296–299 (2009)
14. Chen, X., Liao, C., Wei, T., Hu, S.: An interconnect reliability-driven routing technique for electromigration failure avoidance. *IEEE Trans. Dependable Secure Comput.* **9**(5), 770–776 (2012)
15. Chen, Y., Lin, Y., Gai, T., Su, Y., Wei, Y., Pan, D.Z.: Semi-supervised hotspot detection with self-paced multi-task learning. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 420–425 (2019)
16. Chen Ih-Chin, C.J.Y., Chenming, H.: The effect of channel hot-carrier stressing on gate-oxide integrity in MOSFETs. *IEEE Trans. Electron Devices* **35**(12), 2253–2258 (1988)
17. Ding, D., Torres, J.A., Pan, D.Z.: High performance lithography hotspot detection with successively refined pattern identifications and machine learning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **30**(11), 1621–1634 (2011)

18. Ding, D., Yu, B., Ghosh, J., Pan, D.Z.: EPIC: efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 263–270 (2012)
19. Ding, Y., Zhao, P., Hoi, S.C.H., Ong, Y.S.: An adaptive gradient method for online AUC maximization. In: AAAI Conference on Artificial Intelligence, pp. 2568–2574 (2015)
20. Ebrahimi, M., Oboril, F., Kiamehr, S., Tahoori, M.B.: Aging-aware logic synthesis. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 61–68 (2013)
21. Fang, J., Sapatnekar, S.S.: The impact of hot carriers on timing in large circuits. In: 17th Asia and South Pacific Design Automation Conference, pp. 591–596 (2012)
22. Fang, J., Sapatnekar, S.S.: The impact of BTI variations on timing in digital logic circuits. *IEEE Trans. Device Mater. Reliab.* **13**(1), 277–286 (2013)
23. Firouzi, F., Kiamehr, S., Tahoori, M.B.: NBTI mitigation by optimized NOP assignment and insertion. In: 2012 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 218–223 (2012)
24. Gao, W., Jin, R., Zhu, S., Zhou, Z.H.: One-pass AUC optimization. In: International Conference on Machine Learning (ICML), pp. III906–III914 (2013)
25. Grasser, T., Kaczer, B., Goes, W., Reisinger, H., Aichinger, T., Hehenberger, P., Wagner, P., Schanovsky, F., Franco, J., Roussel, P., Nelhiebel, M.: Recent advances in understanding the bias temperature instability. In: 2010 International Electron Devices Meeting, pp. 4.4.1–4.4.4 (2010). doi:[10.1109/IEDM.2010.5703295](https://doi.org/10.1109/IEDM.2010.5703295)
26. Grasser, T., Kaczer, B., Goes, W., Reisinger, H., Aichinger, T., Hehenberger, P., Wagner, P.J., Schanovsky, F., Franco, J., Luque, M.T., et al.: The paradigm shift in understanding the bias temperature instability: from reaction–diffusion to switching oxide traps. *IEEE Trans. Electron Devices* **58**(11), 3652–3666 (2011)
27. Guerin, C., Huard, V., Bravaix, A.: The energy-driven hot-carrier degradation modes of nMOSFETs. *IEEE Trans. Device Mater. Reliab.* **7**(2), 225–235 (2007)
28. Hanley, J.A., McNeil, B.J.: The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* **143**(1), 29–36 (1982)
29. Hsu, C., Guo, S., Lin, Y., Xu, X., Li, M., Wang, R., Huang, R., Pan, D.Z.: Layout-dependent aging mitigation for critical path timing. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 153–158 (2018)
30. Hsu, M.K., Katta, N., Lin, H.Y.H., Lin, K.T.H., Tam, K.H., Wang, K.C.H.: Design and manufacturing process co-optimization in nano-technology. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 574–581 (2014)
31. Hu, H., Li, P., Huang, J.Z.: Enabling high-dimensional Bayesian optimization for efficient failure detection of analog and mixed-signal circuits. In: ACM/IEEE Design Automation Conference (DAC) (2019)
32. Huard, V., Parthasarathy, C., Bravaix, A., Guerin, C., Pion, E.: CMOS device design-in reliability approach in advanced nodes. In: 2009 IEEE International Reliability Physics Symposium, pp. 624–633 (2009)
33. Jiang, I.H.R., Chang, H.Y., Chang, C.L.: WiT: optimal wiring topology for electromigration avoidance. *IEEE Trans. Very Large Scale Integr. Syst.* **20**(4), 581–592 (2012)
34. Kaczer, B., Mahato, S., Valduga de Almeida Camargo, V., Toledoano-Luque, M., Roussel, P.J., Grasser, T., Catthoor, F., Dobrovolsky, P., Zuber, P., Wirth, G., Groeseneken, G.: Atomistic approach to variability of bias-temperature instability in circuit simulations. In: 2011 International Reliability Physics Symposium, pp. XT.3.1–XT.3.5 (2011)
35. Kumar, S.V., Kim, C.H., Sapatnekar, S.S.: NBTI aware synthesis of digital circuits. In: ACM/IEEE Design Automation Conference (DAC), pp. 370–375 (2007)
36. Li, B., Muller, P., Warnock, J., Sigal, L., Badami, D.: A case study of electromigration reliability: from design point to system operations. In: IEEE International Reliability Physics Symposium (IRPS), pp. 2D.1.1–2D.1.6 (2015)
37. Li, X.: Finding deterministic solution from underdetermined equation: large-scale performance modeling by least angle regression. In: ACM/IEEE Design Automation Conference (DAC), pp. 364–369 (2009)

38. Lienig, J.: Electromigration and its impact on physical design in future technologies. In: ACM International Symposium on Physical Design (ISPD), pp. 33–40 (2013)
39. Lienig, J., Thiele, M.: Fundamentals of Electromigration-Aware Integrated Circuit Design. Springer, Berlin (2018)
40. Liew, B.K., Cheung, N.W., Hu, C.: Projecting interconnect electromigration lifetime for arbitrary current waveforms. *IEEE Trans. Electron Devices* **37**(5), 1343–1351 (1990)
41. Lin, S.Y., Chen, J.Y., Li, J.C., Wen, W.Y., Chang, S.C.: A novel fuzzy matching model for lithography hotspot detection. In: ACM/IEEE Design Automation Conference (DAC), pp. 68:1–68:6 (2013)
42. Lin, Y., Alawieh, M.B., Ye, W., Pan, D.: Machine learning for yield learning and optimization. In: IEEE International Test Conference (ITC), pp. 1–10 (2018)
43. Lorenz, D., Georgakos, G., Schlichtmann, U.: Aging analysis of circuit timing considering NBTI and HCI. In: 2009 15th IEEE International On-Line Testing Symposium, pp. 3–8 (2009)
44. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.* **18**(1), 50–60 (1947)
45. Matsunawa, T., Gao, J.R., Yu, B., Pan, D.Z.: A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction. In: Proceedings of SPIE, vol. 9427 (2015)
46. Matsunawa, T., Nojima, S., Kotani, T.: Automatic layout feature extraction for lithography hotspot detection based on deep neural network. In: SPIE Advanced Lithography, vol. 9781 (2016)
47. Nigam, T., Parameshwaran, B., Krause, G.: Accurate product lifetime predictions based on device-level measurements. Proceedings of the 2009 IEEE International Reliability Physics Symposium, pp. 634–639 (2009)
48. Oboril, F., Tahoori, M.B.: ExtraTime: modeling and analysis of wearout due to transistor aging at microarchitecture-level. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–12 (2012)
49. de Paris, L., Posser, G., Reis, R.: Electromigration aware circuits by using special signal non-default routing rules. In: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2795–2798 (2016)
50. Park, J.W., Torres, A., Song, X.: Litho-aware machine learning for hotspot detection. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **37**(7), 1510–1514 (2018)
51. Pecht, M., Radojcic, R., Rao, G.: Guidebook for managing silicon chip reliability. CRC Press, New York (2017)
52. Rauch, S.E., La Rosa, G.: The energy-driven paradigm of nMOSFET hot-carrier effects. *IEEE Trans. Device Mater. Reliab.* **5**(4), 701–705 (2005)
53. Reis, R., Cao, Y., Wirth, G.: Circuit design for reliability. Springer, Berlin (2015)
54. Ren, P., Xu, X., Hao, P., Wang, J., Wang, R., Li, M., Wang, J., Bu, W., Wu, J., Wong, W., Yu, S., et al.: Adding the missing time-dependent layout dependency into device-circuit-layout co-optimization: new findings on the layout dependent aging effects. In: IEEE International Electron Devices Meeting (IEDM) (2015)
55. Roy, S., Pan, D.Z.: Reliability aware gate sizing combating NBTI and oxide breakdown. In: International Conference on VLSI Design, pp. 38–43 (2014)
56. Rudin, C., Schapire, R.E.: Margin-based ranking and an equivalence between AdaBoost and RankBoost. *J. Mach. Learn. Res.* **10**(Oct), 2193–2232 (2009)
57. Tam, S., Ko, P.K., Hu, C.: Lucky-electron model of channel hot-electron injection in MOS-FET's. *IEEE Trans. Electron Devices* **31**(9), 1116–1125 (1984)
58. Sapatnekar, S.S.: What happens when circuits grow old: aging issues in CMOS design. In: 2013 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA), pp. 1–2 (2013)
59. Sengupta, D., Sapatnekar, S.S.: Estimating circuit aging due to BTI and HCI using ring-oscillator-based sensors. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **36**(10), 1688–1701 (2017)

60. Shin, M., Lee, J.H.: Accurate lithography hotspot detection using deep convolutional neural networks. *J. Micro/Nanolithogr. MEMS MOEMS* **15**(4), 043507 (2016)
61. Steck, H.: Hinge rank loss and the area under the ROC curve. In: European Conference on Machine Learning, pp. 347–358. Springer, Berlin (2007)
62. Swets, J.A., Pickett, R.M.: Evaluation of diagnostic systems: methods from signal detection theory. Academic Press, New York (1982)
63. Ting, L., May, J., Hunter, W., McPherson, J.: AC electromigration characterization and modeling of multilayered interconnects. In: IEEE International Reliability Physics Symposium (IRPS), pp. 311–316 (1993)
64. Vattikonda, R., Wang, W., Cao, Y.: Modeling and minimization of PMOS NBTI effect for robust nanometer design. In: ACM/IEEE Design Automation Conference (DAC), pp. 1047–1052 (2006)
65. Wang, Y., Chen, X., Wang, W., Cao, Y., Xie, Y., Yang, H.: Leakage power and circuit aging cooptimization by gate replacement techniques. *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.* **19**(4), 615–628 (2011)
66. Wen, W.Y., Li, J.C., Lin, S.Y., Chen, J.Y., Chang, S.C.: A fuzzy-matching model with grid reduction for lithography hotspot detection. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **33**(11), 1671–1680 (2014)
67. Wilcoxon, F.: Individual comparisons by ranking methods. *Biom. Bullet.* **1**(6), 80–83 (1945)
68. Wu, K.C., Marculescu, D.: Joint logic restructuring and pin reordering against NBTI-induced performance degradation. In: IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE), pp. 75–80 (2009)
69. Xie, J., Narayanan, V., Xie, Y.: Mitigating electromigration of power supply networks using bidirectional current stress. In: ACM Great Lakes Symposium on VLSI (GLSVLSI), pp. 299–302 (2012)
70. Xu, J., Sinha, S., Chiang, C.C.: Accurate detection for process-hotspots with vias and incomplete specification. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 839–846 (2007)
71. Yan, L., Dodier, R.H., Mozer, M., Wolniewicz, R.H.: Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In: International Conference on Machine Learning (ICML), pp. 848–855 (2003)
72. Yang, H., Li, S., Tabery, C., Lin, B., Yu, B.: Bridging the gap between layout pattern sampling and hotspot detection via batch active sampling (2018). arXiv preprint:1807.06446
73. Yang, H., Luo, L., Su, J., Lin, C., Yu, B.: Imbalance aware lithography hotspot detection: a deep learning approach. In: SPIE Advanced Lithography, vol. 10148 (2017)
74. Yang, H., Su, J., Zou, Y., Yu, B., Young, E.F.Y.: Layout hotspot detection with feature tensor generation and deep biased learning. In: ACM/IEEE Design Automation Conference (DAC), pp. 62:1–62:6 (2017)
75. Ye, W., Alawieh, M.B., Li, M., Lin, Y., Pan, D.Z.: Litho-GPA: Gaussian process assurance for lithography hotspot detection. In: IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE) (2019)
76. Ye, W., Alawieh, M.B., Lin, Y., Pan, D.Z.: Tackling signal electromigration with learning-based detection and multistage mitigation. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 167–172 (2019)
77. Ye, W., Lin, Y., Li, M., Liu, Q., Pan, D.Z.: LithoROC: lithography hotspot detection with explicit ROC optimization. In: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), pp. 292–298 (2019)
78. Ye, W., Lin, Y., Xu, X., Li, W., Fu, Y., Sun, Y., Zhan, C., Pan, D.Z.: Placement mitigation techniques for power grid electromigration. In: IEEE International Symposium on Low Power Electronics and Design (ISLPED) (2017)
79. Yu, Y.T., Chan, Y.C., Sinha, S., Jiang, I.H.R., Chiang, C.: Accurate process-hotspot detection using critical design rule extraction. In: ACM/IEEE Design Automation Conference (DAC), pp. 1167–1172 (2012)

80. Yu, Y.T., Lin, G.H., Jiang, I.H.R., Chiang, C.: Machine-learning-based hotspot detection using topological classification and critical feature extraction. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **34**(3), 460–470 (2015)
81. Zhang, H., Kochte, M.A., Schneider, E., Bauer, L., Wunderlich, H., Henkel, J.: Strap: Stress-aware placement for aging mitigation in runtime reconfigurable architectures. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 38–45 (2015)
82. Zhang, H., Zhu, F., Li, H., Young, E.F.Y., Yu, B.: Bilinear lithography hotspot detection. In: ACM International Symposium on Physical Design (ISPD), pp. 7–14 (2017)
83. Zhang, S., Lyu, W., Wang, F., Yan, C., Hu, X., Zeng, X.: An efficient multi-fidelity Bayesian optimization approach for analog circuit synthesis. In: ACM/IEEE Design Automation Conference (DAC) (2019)
84. Zhao, P., Hoi, S.C.H., Jin, R., Yang, T.: Online AUC maximization. In: International Conference on Machine Learning (ICML), pp. 233–240 (2011)
85. Zhu, X., Lafferty, J., Ghahramani, Z.: Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In: ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining, vol. 3 (2003)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Part V

Cross-Layer from Architecture to Application

Michael Glaß

Possibly since the proposal of the *von Neumann architecture*, there exists an invisible barrier between hardware and software in the narrower or architecture and application in the broader sense. Given that powerful compilers have become omnipresent for almost all architectures and languages, application developers are able to treat the architecture layer and respective system components as black boxes—as long as the architecture *overprovides* in the sense that requirements of the application can be *trivially* fulfilled. Two well-known requirements that have questioned this view in the past are performance and energy considerations. The former gave rise to DSPs and GPUs to name two exemplary architectures that address the special needs of complete application domains while at the same time, the applications have to be adapted to efficiently make use of the architectures and avoid performance bottlenecks. The latter—together with real-time requirements—fueled hardware/software co-design in the embedded system domain where individual applications and their requirements are tailored and deployed to typically multiple components featuring different architectures.

In the area of dependable—emphasis on reliable, available, and safe—systems, respective and often stringent requirements have traditionally been addressed by either (a) architectures/components that give guarantees on dependability properties and achieve these by significant conservative margins and guard banding or by (b) system-wide mitigation strategies like dual or triple modular redundancy schemes. In these domains, one can assume dependability to be a prime, if not the prime, design objective while accepting to sacrifice other objectives such as energy, area consumption, or monetary costs.

M. Glaß

Ulm University, Institute of Embedded Systems/Real-Time Systems, Ulm, Germany;
michael.glass@uni-ulm.de

Continuous technology scaling with all its tremendous advantages has at the same time threatened the dependability of CMOS devices: Increasing susceptibility to aging and radiation effects combined with significant variation has reached a point where hiding such effects from the application completely may require excessive and—for many domains and especially embedded systems—uneconomical overheads. At the same time, mentioned system-wide coarse-grained mitigation techniques may be prohibitive due to other important design objectives for embedded systems such as energy budgets, space constraints, and the like. Assuming that errors at architecture layer may have to be exposed to the application, an interesting and key observation is that not all parts of an application are equally prone to errors on component/architecture layer and are equally costly to harden.

In this area of the book at hand, seven chapters present their endeavors to analyze and exploit architectural as well as application properties concurrently to form *cross-layer* approaches that combine architecture and application layer. At the center of this area is the interplay of specific application (domain) properties on the one and the computational and memory elements on the other hand. The area and its chapters are organized as follows:

The following two chapters put focus on the interplay of applications and memory: chapter “Exploiting Memory Resilience for Emerging Technologies: An Energy-Aware Resilience Exemplar for STT-RAM Memories” investigates Spin Transfer Torque Magnetic RAM (STT-RAM) as a technology which may replace SRAM in near-future devices. However, the susceptibility of STT-RAMs to errors especially during write operations depends on the state transition as well as the applied current level. The chapter presents an architectural scheme for STT-RAM cache memories that dynamically profiles the use of each individual bit by the respective application to select cache way and current level; guaranteeing a maximum error rate while minimizing energy consumption. chapter “Design of Efficient, Dependable SoCs Based on a Cross-Layer-Reliability Approach with Emphasis on Wireless Communication as Application and DRAM Memories” investigates inherently error-resilient applications—particularly wireless baseband processing—and proposes to treat hardware errors in a similar fashion as transmission errors over a noisy channel by means of the so-called dynamic resilience actuators. Since wireless baseband applications are not only compute, but can as well be memory intensive, their error resiliency is further exploited by a proposed approximate DRAM technique which trades-off refresh rate and, thus, energy consumption, for reliability according to the robustness of the application.

The next three chapters focus on variation when deploying applications to multi-/many-core systems, covering dependable dynamic resource management, uncertainty-aware reliability analysis, as well as approximate computing: chapter “Power-Aware Fault-Tolerance for Embedded Systems” puts focus on applications to be deployed to multi-/many-core chips under thermal design power constraints. The addressed challenge is to provide an optimal mix of hardware and software hardening techniques for an application such that reliability goals are met without violating power and/or performance constraints. As a remedy, the chapter introduces a power-reliability management technique that combines

design-time analysis and optimization as well as run-time adaptation to account for variation in performance, occurring faults, and power. Chapter “Uncertainty-Aware Compositional System-Level Reliability Analysis” addresses the problem that uncertainties arising from manufacturing tolerances, environmental influences, or application inputs may not be known at design time such that respective worst-case assumptions result in extremely pessimistic reliability analysis results. As a remedy, the chapter presents a compositional and uncertainty-aware reliability analysis approach that explicitly models uncertainties and, thus, exposes not only best-case or worst-case values but probability distributions. These can enhance cost-efficient error mitigation by quantifying the probability of different best-/average-/worst-case situations. Chapter “Hardware/Software Codesign for Energy Efficiency and Robustness: From Error-Tolerant Computing to Approximate Computing” puts focus on the margins that are applied to compensate for the ever-increasing variability. The chapter first presents an approach to drastically reduce margins for GPUs by means of an adaptive compiler that aims at equalizing the lifetime of each processing element. Then, the reduction of margins is pushed so far that errors have to accepted and computations become approximations for which the chapter proposes an automatic FPGA design flow for accelerators that employ such approximate computations.

The last two chapters in this area put their attention to two important application domains in current and future embedded systems: Cyber-physical systems and machine learning. Chapter “Reliable CPS Design for Unreliable Hardware Platforms” investigates cyber-physical systems and especially battery-operated systems where control loops are the key part of their applications. While such control loops traditionally focus on certain metrics such as stability or overshoot, they may affect system components such as the batteries themselves as well as the processing components the applications are deployed to by accelerating their aging. As a remedy, the chapter proposes a design flow that combines an optimization of (a) quality-of-control and battery behavior at design time as well as (b) quality-of-control and processor aging at run-time to satisfy safety requirements. Chapter “Robust Computing for Machine Learning-Based Systems” investigates machine learning approaches as key enablers for various upcoming safety-critical applications from the domain of autonomous systems. Especially in this context, the chapter investigates the susceptibility of these both intrinsically compute and memory-intensive applications to reliability as well as security threats. The chapter discusses techniques to enhance the robustness/resilience of machine learning applications and outlines open research challenges in this domain.

Design of Efficient, Dependable SoCs Based on a Cross-Layer-Reliability Approach with Emphasis on Wireless Communication as Application and DRAM Memories



Christian Weis, Christina Gimmler-Dumont, Matthias Jung,
and Norbert Wehn

1 Introduction

Technology scaling has reached a point at which process and environmental variabilities are no longer negligible, and can no longer be hidden from system designers, as the exact behavior of CMOS devices becomes increasingly less predictable. This will show in the form of static and dynamic variations, time-dependent device degradation and early life failures, sporadic timing errors, radiation-induced soft errors, and lower resilience to varying operating conditions [35]. Already today, conservative margining, guardbanding, and conservative voltage scaling, come at a large cost. Only turning away from conservative worst-case design methodologies for a 100% reliable physical hardware layer will make further downscaling of CMOS technologies a profitable endeavor [7, 32]. This calls for radically new cross-layer-design concepts [14–16] (Fig. 1).

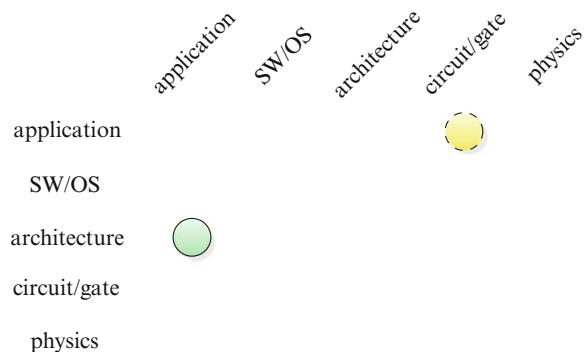
Until today, these problems have mostly been addressed at the lower design levels. At the higher levels, systems are typically designed under the premise of fault-free underlying hardware. Only in extremely critical applications, such as avionics, where the system cost is less important than its dependability, triple modular redundancy (TMR) and similar techniques are employed on a system level. Thus, to no big surprise, the large body of related work focuses on low-level

C. Weis (✉) · N. Wehn
TU Kaiserslautern, Kaiserslautern, Germany
e-mail: weis@eit.uni-kl.de; wehn@eit.uni-kl.de

C. Gimmler-Dumont
European Patent Office, Bruxelles, Belgium
e-mail: cgimmlerdumont@epo.org

M. Jung
Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
e-mail: Matthias.Jung@iese.fraunhofer.de

Fig. 1 Main abstraction layers of embedded systems as used in the SPP1500 Dependable Embedded Systems Priority Program of the German Research Foundation (DFG) and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



techniques to present higher abstraction and design levels with a practically error-free platform built from potentially faulty elements.

To make a platform resilient against transient or permanent faults built-in redundancy or built-in self-recovery techniques have to be employed. They all come at the cost of chip area, power consumption, reduced system throughput, or other implementation related metrics. Lower implementation cost, especially with regard to energy consumption can be obtained when a degradation of hardware reliability to a certain degree is tolerated. In fact, *energy consumption and dependability of integrated circuits can be seen as strongly interrelated problems*: by decreasing the operating voltage, the energy efficiency increases but at the same time the dependability decreases. Thus, energy efficiency and dependability have to be carefully traded off against each other.

An error-resilient architecture that can be seen as practically error-free can be composed of protected components. Applications for these platforms can be implemented in a traditional way, still assuming fault-free operation of the underlying hardware. In addition to this horizontal integration, recent research also evaluates the additional potential of a vertical integration of error resilience on the application level with platforms having a reduced reliability. True *cross-layer optimization* approaches do not only exploit the fact that some important classes of algorithms are inherently error-tolerant, but also adapt applications and hardware architectures jointly to achieve the best possible trade-offs. This chapter focuses on cross-layer optimization for wireless communication systems with emphasis on errors on data path and SRAMs. Furthermore, we consider undependable DRAM subsystems, named *approximate DRAM*.

2 Wireless Baseband Processing

In this section, we present a wireless baseband processing system and a novel cross-layer methodology using resilience actuators (see Sect. 2.1.2) to improve the reliability of this system. Wireless communication systems have an inherent error

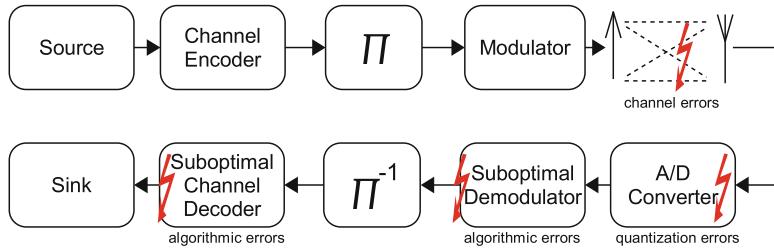


Fig. 2 Wireless communication systems suffer from different sources of errors (indicated by red arrows): channel errors, quantization errors, errors from suboptimal algorithms, and hardware errors. Here, we focus on the receiver side only

resilience. They are designed to recover the originally transmitted data sequence in spite of errors that occur during transmission over a noisy channel. Figure 2 shows a simplified structure of such a system. To achieve a reliable transmission, today's communication systems use advanced forward error correction (FEC) techniques, i.e. the sender adds redundancy to the actual information prior to transmission in the channel encoder. This encoder connects to the modulator via an interleaver (Π). The interleaver is required to break dependencies between neighboring bits while the modulator performs the mapping on symbols (e.g. QAM—quadrature amplitude modulation) that are transmitted over the physical channel. On the receiver side, the noisy signal is converted to the digital domain and fed into the demodulator, which recovers the originally transmitted symbols by exploiting the channel characteristics. After the deinterleaver (Π^{-1}) the channel decoder uses the redundancy to correct transmission errors.

The primary goal is to correct errors from the noisy channel. But implementation efficiency of communication systems in hardware mandates e.g. quantization of data values and the use of suboptimal algorithms, i.e., algorithms that generate results which deviate from the theoretically correct values. Both can be seen as further sources of errors in addition to the noise on channel. In the same way, errors induced by hardware faults can be considered as yet another error source in a communication system. The question is if the hardware errors can be processed in a similar way than the channel and what are the costs.

2.1 Methodology: Error Mitigation Using Dynamic Resilience Actuators

Modeling of hardware errors is crucial for the design of dependable systems. Radiation, thermal effects, aging, or process or parameter variations cause distortions on a physical level which can be modeled by probabilistic bit flips according to the resilience articulation point (RAP) model [16] (see also the chapter “RAP Model—Enabling Cross-Layer Analysis and Optimization for System-on-Chip Resilience” in this book). Depending on its location, a bit flip can have very different effects. An

error in the controller, for example, usually leads to a system malfunction, whereas individual errors in the memories or the data flow are often inherently corrected by a wireless receiver ([12, 31]). Efficiency in terms of area and energy will be achieved by recovering only from those errors which have a significant impact on the system output and by choosing the layer on which the treatment of these error results in the least overhead.

Dynamic approaches for error resilience also have to monitor the current hardware status. This monitoring can be done on different abstraction layers. Examples are error detection sequential (EDS) circuits on microarchitectural layer. EDS circuits are very popular [6]; however, they require pre- and post-silicon calibration. Monitors on higher abstraction layers are application-specific and normally more efficient. For example, [3] proposed to detect timing errors with a small additional hardware block which mimics the critical path under relaxed timing constraints. The result of the mimic hardware is compared to the normally operating unit. Deviations indicate timing errors. For a turbo and convolutional code decoder, the mimic hardware only required 0.7% of the decoder area. In this article we focus on resilience techniques which are employed after hardware errors have been detected, not on the detection methods themselves.

Many state-of-the art publications utilize low-level static resilience techniques to combat the effects of unreliable hardware, e.g., ECC protection of memories, Razor flip flops, or stochastic logic [36]. Static methods have the disadvantage of permanently decreasing the system performance in at least one of the terms of throughput, area, or power, even when no errors occur. In [31] for example, the static protection of a complete LDPC (Low-Density Parity Check) decoder for WiMax/WiFi resulted in an area overhead of 21%.

Dynamic techniques often use available hardware resources or have very low additional costs as we will show in Sect. 2.2.1. However, error detection circuits result in additional costs. When comparing static and dynamic methods, this additional cost has to be taken into account. In general, the choice of the protection method will also depend on the expected hardware error statistics as we will demonstrate in the next paragraph. Eventually, a combination of static and dynamic protection will likely result in the least overhead.

2.1.1 The Dynamic Behavior of Wireless Systems

Modern wireless communication standards, such as LTE (Long Term Evolution) or HSDPA (High Speed Downlink Packet Access) provide mechanisms to monitor and dynamically adapt to changes in the Quality-of-Service (QoS). The QoS in a wireless transmission system is typically defined as the bit or frame error rate with respect to a given signal-to-noise ratio. If the desired QoS cannot be achieved for the current transmission channel, communication parameters like code type, code rate, etc. are adjusted to improve the communications performance (see Fig. 3a). A good example for this dynamic behavior is the hybrid automatic repeat request (H-ARQ), which is used in LTE, HSDPA. These systems typically transmit blocks of data at

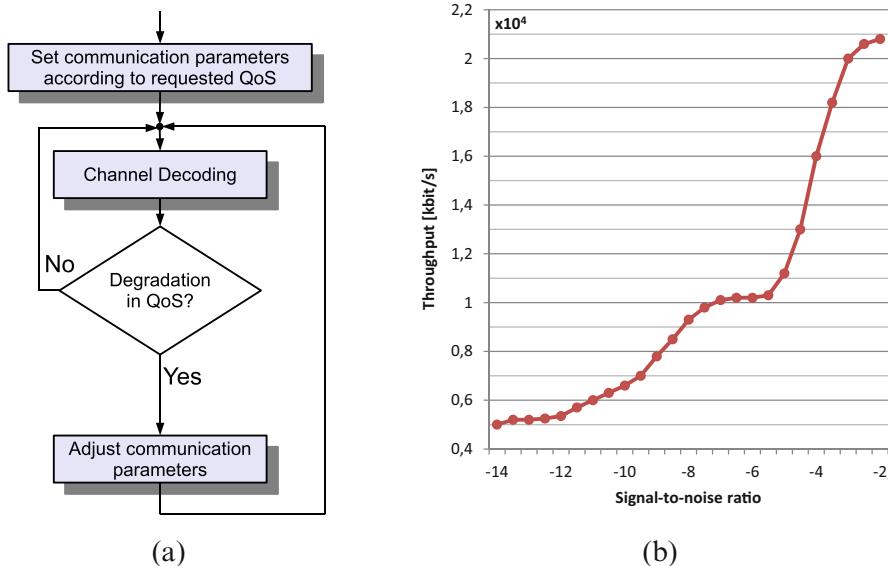


Fig. 3 The standard communication flow dynamically adjusts communication parameters to achieve the required QoS, e.g., the code rate in Hybrid-ARQ systems. (a) Dynamic QoS flow of a modern wireless communication system. (b) In Hybrid-ARQ systems the code rate is dynamically adjusted for each block to ensure error-free transmission

a high data rate and with little error protection, i.e., with a very high code rate. If the decoder fails, the transmission of additional data is requested until the block is correctly decoded. Note that such a retransmission does not contain the same data as before. Instead, different information will be sent every time, which had been punctured on the transmitter side before. The additional information decreases the data rate but at the same time increases the probability that the block can be correctly decoded at the receiver. Figure 3b shows the throughput of a H-ARQ system over different SNR values. For high SNR values, decoding succeeds after the first transmission, i.e., the channel decoder can correct all errors, and a high throughput is obtained. With a decreasing SNR, more and more blocks require additional transmissions and the throughput is lowered. *The system dynamically adapts the code rate and the throughput for each block.*

This example shows how wireless receivers adapt dynamically to changes in the transmission channel, i.e., varying SNR, and correct transmission errors. The question is how this idea can be applied to the case of hardware errors. It has been shown that low rates of hardware errors in a wireless receiver are not visible on the system level. This is due to the fact that for low SNR the channel errors dominate. For high SNR, when the channel error rate is very low, the channel decoder is able to correct the hardware errors. For moderate hardware error rates, some dynamic high-level techniques exist, e.g., increasing the number of decoder iterations to counterbalance the impact of hardware errors. However, for very high error rates

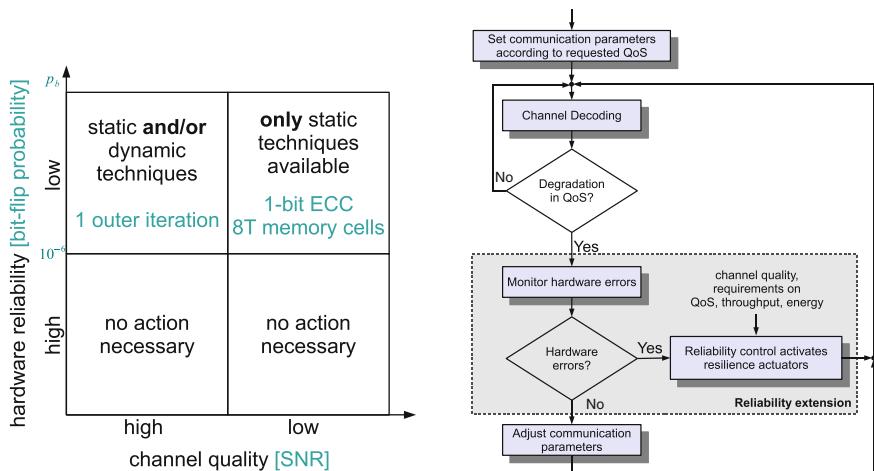
on the hardware level, a purely software-based mitigation is not possible. An increase of reliability can generally be achieved by either static low-level techniques, like e.g., Razor flip flops, triple modular redundancy, or by dynamic high-level techniques, which exploit the flexibility of the receiver, e.g., increase of decoder iterations, or a combination of both. To their advantage, dynamic techniques are mainly algorithmic changes, which can be controlled by software and do not require a more costly change of the underlying hardware.

Consequently, it is possible to use high-level techniques to mitigate hardware errors in wireless communication systems. However, the channel quality changes over the time and channel noise and hardware noise may change independently from each other. In good channel conditions, we can use a part of the error correction capability of the receiver to combat hardware errors if needed. When the channel quality is very poor, all high-level techniques are needed to obtain the required QoS, and hardware errors have to be counterbalanced by static low-complexity methods. This is shown in Fig. 4a. When the hardware reliability is very high, no action has to be taken. High amounts of hardware errors cannot be overcome using dynamic techniques exclusively. A combination of dynamic and static techniques is mandatory. When the channel quality is very poor, only static techniques are available. For medium noise levels, there are potential trade-offs between dynamic and static techniques.

2.1.2 Concept of Dynamic Resilience Actuators

As mentioned before current standards, like HSDPA or LTE, adjust dynamically the QoS at runtime, e.g., higher data throughput rates are specified for higher SNR. This is due to the fact that the computational requirements on the different algorithms decrease with higher SNR in order to enable higher throughput. In future technologies the negotiated QoS may also depend on the reliability of the receiver hardware under given operating conditions. This leads to an entirely new paradigm—adaptive QoS with respect to communication reliability *and* hardware reliability. An illustration of this is the possibility to relax reliability requirements on the underlying hardware instead of providing a higher throughput at high SNR. For example, voltage overscaling can be applied, where the voltage is reduced beyond the point at which fault-free operation of the circuit is guaranteed in order to lower the power consumption of the receiver. In this way, QoS, hardware reliability, and implementation efficiency can be traded off against one another at runtime.

In [3], we presented how this new paradigm can be integrated into the existing QoS flow of wireless communication systems. Figure 4b shows the extended version of the original QoS flow from Fig. 3a. Low rates of hardware errors are implicitly corrected by a wireless receiver. In that case no further action is required. A higher rate of hardware errors results in a degradation of the QoS and, thus, can be detected by the standard QoS flow. The standard QoS flow is already error-resilient by itself, as it dynamically adjusts the communication parameters to obtain a certain QoS. In most cases, however, it will be cheaper in terms of energy to correct a



(a) Depending on the current hardware reliability and channel quality, different static and dynamic techniques (resilience actuators) are available to mitigate the impact of hardware errors. Entries in cyan color quantify the example in Section 2.2.

(b) Extended dynamic QoS flow: The reliability control unit chooses the resilience actuators which result in the least overhead and, thus, in an energy efficient design.

Fig. 4 Our new methodology integrates seamlessly into the existing QoS flow of today's communication systems. The available resilience techniques depend on the current channel quality and hardware reliability. (a) Depending on the current hardware reliability and channel quality, different static and dynamic techniques (resilience actuators) are available to mitigate the impact of hardware errors. Entries in cyan color quantify the example in Sect. 2.2. (b) Extended dynamic QoS flow: The reliability control unit chooses the resilience actuators which result in the least overhead and, thus, in an energy-efficient design

temporary hardware error by the activation of a dynamic protection mechanism than by changing the communication parameters as, e.g., a H-ARQ based correction is very costly with respect to energy consumption.

As already mentioned a degradation of the QoS can be caused by either channel errors or hardware errors. A differentiation of these two error sources is not possible with the existing QoS monitoring system only. Therefore, it is necessary to monitor the reliability status of each hardware component. Single bit flips in the data path for example are often mitigated by the algorithmic error resilience of the receiver. Application-specific detection circuits like the reduced-size ACS (add-compare-select)-unit for turbo decoding proposed in [3] can indicate the status of one component with only a small overhead.

We introduced a reliability control unit which activates one or several *resilience actuators* according to the current monitoring status. A resilience actuator is a dynamic protection mechanism, which can increase the error resilience either on component or on system level. Resilience actuators can be found on hardware level and on software level. So far, we identified four classes of actuators. On the

lowest level, we can change the *hardware operating point*, e.g., the supply voltage or the clock frequency. The trade-off between supply voltage, clock frequency, and power consumption is well studied in the literature. Another possibility is the use of *low-level hardware techniques*, such as the selective protection of critical parts, or setting erroneous likelihood values to zero [31]. Many algorithms have parameters which can be changed at runtime. Advanced channel decoders operate iteratively. The number of iterations is a parameter which can easily be changed for each individual block by the software. For many components, we have a choice of different algorithms, starting from optimal algorithms with a high complexity down to suboptimal algorithms with a very low complexity, which offers a trade-off between QoS and implementation efficiency. The *choice of parameters and algorithms* is another class of actuators [3]. There also exist resilience actuators on *system level*. Adjusting the communication parameters, e.g., by choosing a channel code with a better error correction capability, improves the error resilience, but the effects are not immediate. A faster solution is to shift complexity between different components, when one of the components has a low hardware reliability. It is important to note that resilience actuators are only activated when hardware errors cause a degradation of the QoS.

In general, different actuators or combinations of actuators are suited to deal with different types of hardware errors. Normally, it is preferable to use actuators which do not require changes inside the components or which can be implemented with low complexity. Each actuator offers a different trade-off between hardware reliability, QoS, and implementation performance (throughput, energy). Based on the channel quality and the respective requirements on QoS, throughput, and energy, the reliability control chooses those actuators, which will best fulfill the requirements. Therefore, it is mandatory to characterize each actuator with regard to its influence on communications performance, throughput, area, and energy overhead. Sometimes, the reliability requirements necessitate the use of resilience actuators which have a severe effect, e.g., on the system throughput. In these cases, the reliability control also needs actuators which trade-off throughput and communications performance. The big advantage of this reliability extension is the dynamic protection of the wireless receiver, which is only activated when necessary.

2.2 A Case Study

In the last section, our new methodology was generally introduced. The trade-off between channel quality and hardware resilience and the choice of the resilience actuators are application-specific and cannot be quantified in a general fashion. In this paragraph, we demonstrate our methodology on a concrete example in order to make it more seizedable.

Multiple-antenna or MIMO systems have the potential to increase the data rate of wireless communication systems. They belong to the most advanced systems in 4G and 5G communication standards, and their very high complexity is a challenge for

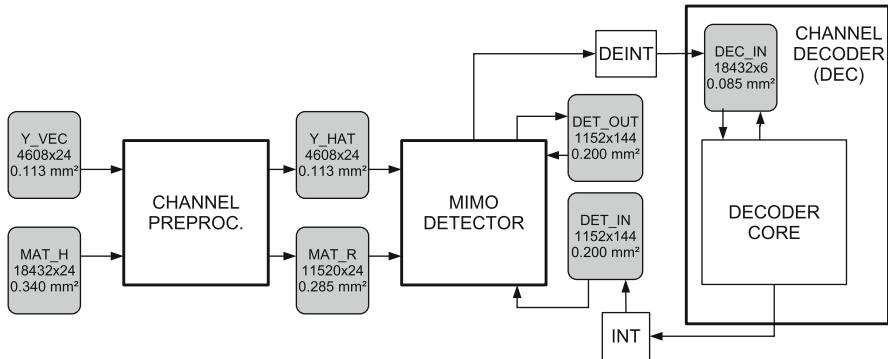


Fig. 5 Generic architecture of an iterative MIMO-BICM receiver including main building blocks and system memories

any hardware implementation. To demonstrate our novel methodology, we chose to apply it to a double-iterative MIMO-BICM (bit-interleaved coded modulation) transmission system.

Aforementioned, a channel code provides redundancy, which allows the correction of transmission errors in the receiver. An interleaver between channel encoder and modulator reduces dependencies between neighboring bits. The modulated symbols are multiplexed to an array of antennas and then transmitted in parallel to increase the data rate. Such a system setup is called MIMO-BICM system. On the receiver side, a MIMO detector decouples the multiple transmission streams, and the channel decoder corrects errors, which have been induced by noise on the communication channel. The most advanced receiver techniques combine the MIMO detector and the channel decoder in an iterative feedback loop to further improve the communications performance of the receiver [17]. These two blocks exchange likelihood values, which reflect their confidence in the results of their computations. The channel decoder can be iterative itself (and often is), which results in a double-iterative receiver structure. The number of iterations is dynamic and depends strongly on the respective system state and QoS requirements.

Multiple-antenna systems are combined with different types of channel codes in the existing standards. WiFi features LDPC codes and convolutional codes, whereas LTE supports only the trellis based convolutional and turbo codes. WiMax supports all three kinds of channel codes. Therefore, we mapped the iterative receiver structure from [11] onto a general architecture framework, which allows us to plug in different MIMO detectors and channel decoders [13]. The generic architecture shown in Fig. 5 connects the main building blocks via several system memories.

We presented in [11] the details of the implementation results for all components of the iterative receiver [13, 34]. All designs were synthesized in a 65 nm low-power bulk CMOS standard cell library. Target frequency after place and route is 300 MHz, which is typical of industrial designs (exception WiMax/WiFi LDPC decoder). The size of the system memories is determined by the largest block length in each

communication standard. For example, LTE turbo codes include up to 18,432 bits. In this case, the system memories require approximately 40% of the total system area. For WiMax/WiFi, the maximum block length is only 2304 bits which results in a much smaller area for the system memories. The power consumption of the memories is not neglectable when compared to the other components [13]. The total power consumption depends heavily on the number of inner and outer iterations.

The system memories add substantially to the die area of such an iterative MIMO-BICM receiver. Memories are very susceptible to hardware errors due to their dense and highly optimized layouts. In [12], we analyzed the impact of hardware errors in the different system memories on the system performance of a MIMO-BICM system. We found out that especially the memories containing complex-valued data, i.e. the channel information and the received vectors, are very sensitive. Figure 6 shows the degradation of the communications performance when errors are injected in the channel information memory. Up to a bit error probability of $p_b = 10^{-6}$ the degradation is negligible for the typical frame error rates (FERs) of a wireless system. Afterwards, the performance decreases gradually with an increasing p_b .

We assume that the memory errors result from supply voltage drops which occur regularly during power state switching. In this context, several resilience actuators exist, which can be applied to different degrees of hardware unreliability in order to mitigate the impact of the hardware errors on the system performance [26]. Table 1 lists them with their influence on area, power consumption, and throughput

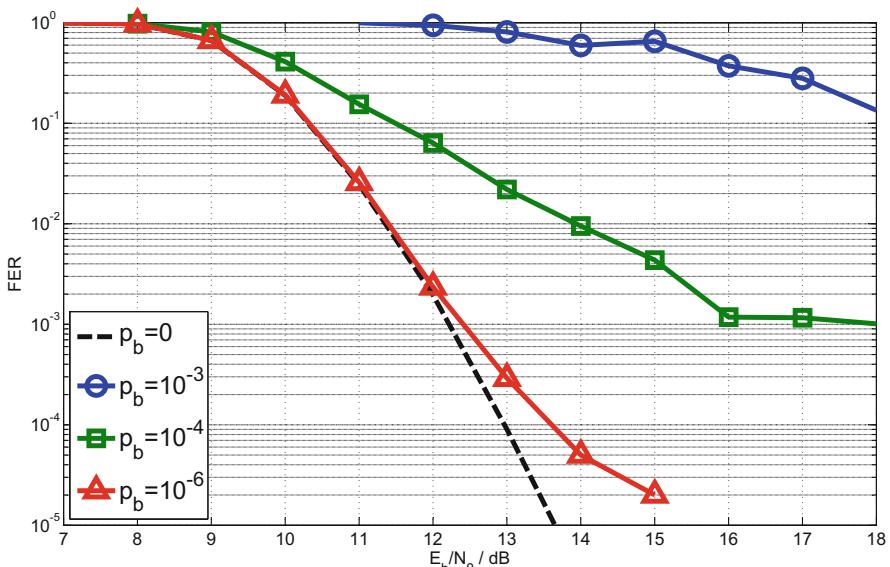


Fig. 6 The system communication performance is gradually decreasing for random bit flips in the channel information memory. p_b depicts the bit error probability

Table 1 Quantitative comparison of resilience actuators for the system memories of an iterative MIMO-BICM receiver

Resilience actuator	Impacts	Tolerated hardware (un-) reliability	Tolerated hardware error probability
Algorithmic error resilience	Area +0%	−200 mV supply voltage	10^{-6}
	Power +0%		
	Throughput −0%		
1 outer iteration	Area +0%	−300 mV supply voltage	$4 \cdot 10^{-5}$
	Power +0%		
	Throughput −75%		
1-bit error correction code	Area +30%	−400 mV supply voltage	$8 \cdot 10^{-4}$
	Power +30%		
	Throughput −0%		
8T memory cells	Area +25%	−500 mV supply voltage	$8 \cdot 10^{-3}$ for the equivalent 6T memory cells
	Power +0%		
	Throughput −0%		

and their error resilience. In Fig. 4a, these actuators are arranged according to our methodology (cyan text). No action has to be taken as long as there is a high hardware reliability, i.e. voltage drops of no more than 200 mV. Within this region, the receiver shows an inherent algorithmic error resilience. For a decreased reliability in which voltage drops up to 300 mV occur, we can react on the highest level by increasing the number of iterations in order to regain communications performance. For transient errors, this leads only to a temporary throughput degradation without loss of communications performance. When errors occur with a high probability $p_b > 5 \cdot 10^{-5}$, high-level resilience actuators cannot provide the necessary resilience. On a lower level, the contents of the memory can be protected by a simple 1-bit error correction code. The resilience can be even further increased on technology level by employing 8-transistor (8T) memory cells instead of 6-transistor (6T) cells resulting in a smaller implementation overhead. 8T memory cells can even tolerate voltage drops of 500 mV. However, the increase in area and power is in both cases permanent.

2.2.1 Resilience Actuators

Error resilience techniques for channel decoding have already been thoroughly investigated (cf. [11]). However, there are potential trade-offs on system level between MIMO detection and channel decoding, which we will discuss in the following. Except for the hardware operating point, we will restrict ourselves to application-specific resilience actuators. Universal, already established, low-level hardware techniques can be applied to any application and result in a constant overhead. Here, we focus on dynamic resilience techniques, which can be switched

on and off as necessary and which do not have a large impact on implementation complexity and energy consumption.

As MIMO detectors have no inherent error correction capability, algorithmic changes inside the detector component cannot improve the error resilience. This has to be done either on system level or by changing the hardware operating point. These actions usually have a negative influence on system throughput and/or communications performance. Therefore, we also introduce algorithmic resilience actuators, enabling a trade-off of throughput and communications performance in order to counterbalance these effects.

- *Hardware operating point:* When timing errors occur, the clock frequency can be reduced or the supply voltage can be increased to make the circuit faster. However, both approaches require additional control circuits and energy. The trade-off between supply voltage and energy is well-understood. The number of bit flips in a memory, for example, strongly depends on the voltage: Increasing the supply voltage decreases the soft error rate. According to [8], the soft error rate drops by about 30% when the operating voltage is increased by 100 mV compared to the nominal voltage. Changing the hardware operating point offers a trade-off between reliability and energy consumption, which is often used for voltage overscaling.
- *Adjustment of detection quality:* Changing the detection quality offers a trade-off between communications performance and throughput but has no direct influence on the error resilience. However, a higher throughput augments the available time budget and, thus, offers a higher potential for error resilience. This resilience actuator uses the available algorithmic flexibility and thus has only a negligible influence on power and area consumption.
- *External LLR (log-likelihood ratio) manipulations:* Instead of accessing the MIMO detector directly, we propose low-complexity techniques, which work only on the LLR-input and -output values of the detector. LLR values have a high robustness against hardware errors. If an LLR value is equal to zero, it contains no information. Thus, the most important information is stored in the sign bit. As long as this sign bit is not compromised, the core information is still correct and the channel decoder can correct the hardware errors. For more details see additionally [11].

Instead of increasing the reliability of components individually, the problem can also be tackled on system level. The double-iterative structure of a MIMO-BICM receiver offers several high-level possibilities to combat the unreliability of its components. We present the most promising techniques in the remainder of this section.

- *Iteration control mechanisms:* An iteration control typically monitors exchanged values in an iterative system and checks stopping conditions to detect the convergence of the processed block. In [12] we analyzed the impact of memory errors on the system behavior of an iterative MIMO system. We observed that errors in any of the memories before the MIMO detector have an increased

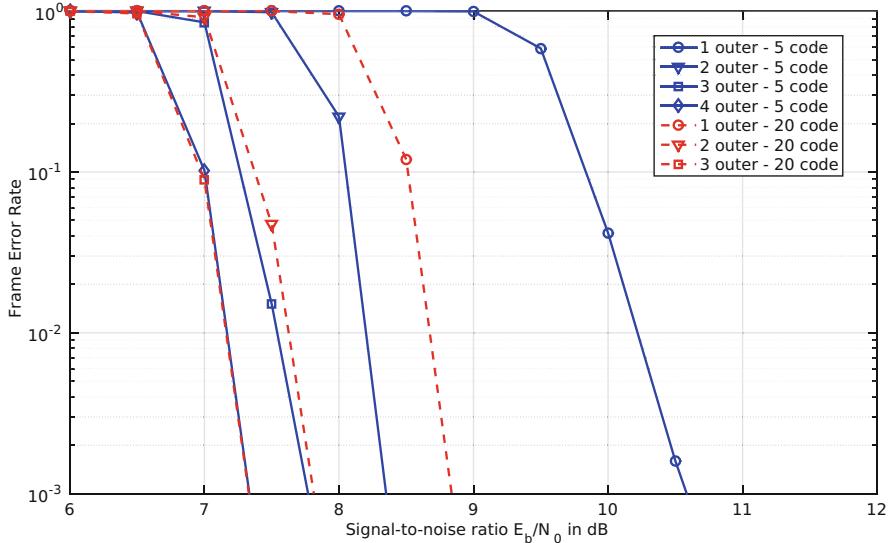


Fig. 7 Example for complexity shifting in a double-iterative MIMO-BICM receiver by varying the number of outer loop and channel decoder iterations

impact on the communications performance if the incorrect values are processed repeatedly during the outer iterations. In [10], it was possible to reduce the number of outer iterations to an average below 2 (from a maximum of 10) without sacrificing communications performance. A further throughput increase is possible by allowing a degradation of communications performance. The additional effort for an iteration control is very low compared to channel decoding [11].

- *Complexity shifting between components:* An example for a global algorithmic adaption is to shift the complexity between system components: When a building block cannot compensate an error locally, the system convergence can still be achieved by increasing the computational effort of other building blocks. Such a shift can be achieved, for instance, between the channel decoder and the MIMO detector, leveraging the outer feedback loop. When the MIMO detector is not able to counterbalance the impact of hardware errors, the number of channel decoder iterations and/or the number of outer loop iterations can be increased in order to maintain the communications performance. Figure 7 shows the frame error rate for a 4×4 antennas, 16-QAM system employing a WiMax-like LDPC code where complexity shifting can be used. We compare the frame error rate for different numbers of decoder iterations and outer iterations. Let us consider the case when the receiver is performing 3 outer iterations and 5 LDPC iterations. When the MIMO detector suffers from hardware errors, e.g. due to a temperature increase, we can temporarily shift more processing to the LDPC decoder by performing only 2 outer iterations and 20 LDPC iterations. The new configuration provides

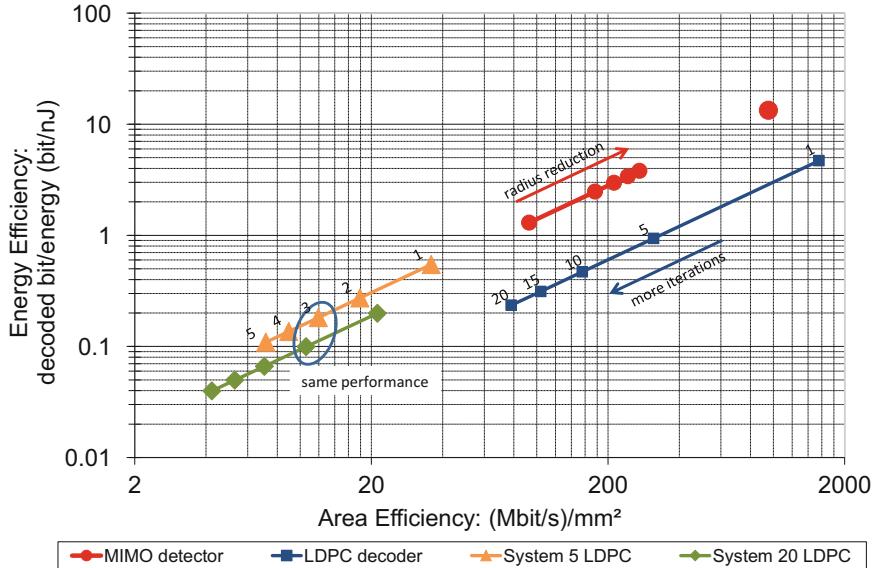


Fig. 8 Implementation efficiency of MIMO detector, WiMax/WiFi LDPC decoder, and two system configurations using the efficiency metrics from [25]

the same communications performance. The question is how such a shift changes the energy efficiency of the MIMO receiver. Figure 8 shows the implementation efficiency of MIMO-BICM receiver and its components. The red curve shows the efficiency of the MIMO detector for different search radii and in a MMSE-SIC configuration (single red point). The blue curve shows the efficiency of an LDPC decoder running with different number of iterations. The LDPC decoder is a flexible decoder which supports all code rates and code lengths from WiMax and WiFi standard. The yellow and the green curve show the system efficiency for different numbers of outer iterations with 5 LDPC iterations (yellow) and 20 LDPC iterations (green), respectively. With the help of this graph, we can quantify the influence of a complexity shift: when changing from 3 outer and 5 LDPC iterations to 2 outer and 20 LDPC iterations, the energy efficiency of the system is reduced by approximately 50% (blue circle). However, the same communications performance is achieved and when the temperature in the MIMO detector decreases, the reliability control can return to the original configuration.

- *Shifting of error correction capability between components:* Typically, MIMO detector and channel decoder are designed and implemented independently of each other. The MIMO transmission scheme provides a large data rate but has no error correction abilities. The error correction capability is solely provided by the channel code to improve the error rate performance of the transmission system. From a system point of view, the MIMO detector does not work on

completely independent data as there are dependencies from the overlaying channel code. However, this diversity cannot be exploited by the detector as the channel interleaver hides the code structure from the detector and because most channel code constraints span over many MIMO detection vectors. Kienle [24] introduced a small block code in each MIMO detection vector in order to generate a small diversity gain in the MIMO detector while simplifying the outer channel code to keep the overall coding rate constant. While this approach targeted the decoding complexity, it can also be used to increase the error resilience of the MIMO detector. Each parity check in one MIMO vector improves the error correction capabilities of the detector. On a system level, the diversity gain can be split between detector and decoder dynamically, thus, allowing the system to react dynamically to changing hardware error rates. The only drawback of this approach is that the diversity separation has to be done on the transmitter side, which causes a higher latency.

3 Approximate DRAM

Some communication systems require large data block sizes that cannot be stored in on-chip memories (SRAMs) anymore. In this case data has to be stored externally in DRAMs. Thus, in the following we shift our focus on DRAMs.

Approximate DRAM is a new concept that adapts the idea of approximate computing to DRAMs [23, 30]. Approximate DRAM exploits this fact by lowering the refresh frequency (reducing vendor guardbands) or even disable the refresh completely and accepting the risk of data errors. The underlying motivation for an Approximate DRAM is the increasing power consumption and performance penalty caused by unavoidable DRAM refresh commands. The authors of [29] and [1] predicted that 40–50% of the power consumption of future DRAM devices will be caused by refresh commands. Moreover, 3D integrated DRAMs like Wide I/O or HMC worsen the scenario with respect to increased cell leakage, due to the much higher temperature. Therefore, the refresh frequency needs to be increased accordingly to avoid retention errors [37].

The characteristic refresh parameters of DRAMs, listed in datasheets, are very pessimistic due to the high process margins added by the vendors to ensure correct functionality under worst-case conditions and most important a high yield [28]. Thus, the DRAM refresh rate recommended by the vendors and JEDEC ($t_{REF} = 64\text{ ms}$) adds a large guardband, as shown in Fig. 9.

As mentioned before many applications like wireless systems have an inherent error resilience that tolerates these errors and therefore, refresh power often can be reduced with a minimal loss of the output quality.

Figure 9 qualitatively shows the retention error behavior over time and the design space for Approximate DRAM. The sphere around the curve represents the process variation, Variable Retention Times (VRT), and Data Pattern Dependencies (DPD). In general, we have two key parameters for Approximate DRAM: The *data lifetime*

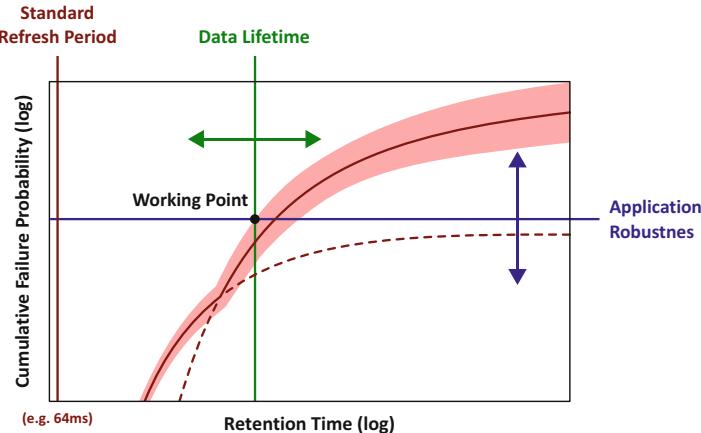


Fig. 9 Qualitative retention error behavior of DRAMs

and the *application robustness*. Both parameters lead to three possibilities in this design space:

- Refresh can be switched off if the data lifetime is smaller than the actual required refresh period.
- Refresh can be turned off if the data lifetime is larger than the required refresh period and the application provides resilience to the resulting number of errors at this working point.
- If the application only provides a maximal robustness the refresh rate is configured according to the resulting working point.

The reliability-energy trade-off for Approximate DRAMs can be explored only by using fast and accurate retention error-aware DRAM models.

In [39] we developed such a model that is usable in full system level simulations. The model was calibrated to the measurement results of DDR3 DRAM devices. A measurement statistic is shown in Fig. 10. Here we measured 40 identical 4 Gbit DDR3 chips from the same vendor. Each single device has been measured ten times at four different temperatures and five retention times, resulting in a total of 8000 measurement points. We plot the retention times versus the normalized and averaged number of errors obtained during each measurement step. The bars mark the minimum and the maximum measured number of errors. We find here a quite prominent variation in the order of 20% (max. number of errors), which shows a large temperature dependency. This needs to be considered as realistic guardband in approximate computing platforms utilizing the Approximate DRAM approach (cf. the sphere in Fig. 9). Additionally, the figure shows a histogram of the absolute number of bit errors (between $1 \cdot 10^6$ and $4 \cdot 10^6$) measured at the data point with 100s retention time and a temperature of 25 °C.

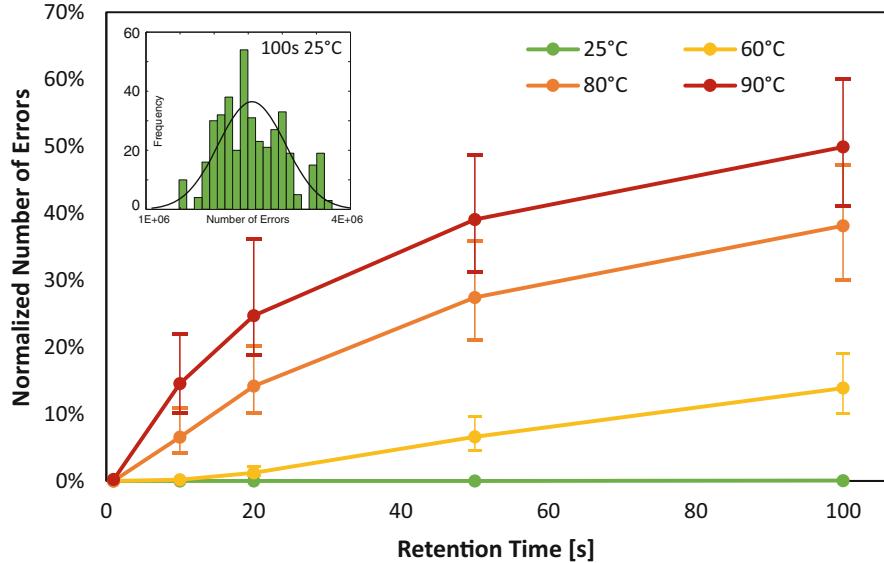


Fig. 10 Retention error measurements of 40 4 Gbit DRAM Devices with different temperatures

Fig. 11 Simulation Framework for Approximate DRAM Explorations and Reliability Management in SoCs

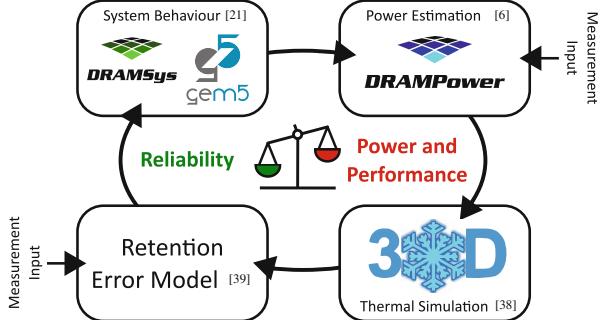


Figure 11 shows our closed-loop simulation flow for investigations on Approximate DRAM. It is based on SystemC Transaction Level Models (TLM) for fast and accurate simulation. This simulation loop uses the modular DRAMSys framework [21] and consists of four key components: DRAM and core models [21], a DRAM power model [5], thermal models [38], and the aforementioned DRAM retention error model. The remaining models are shortly introduced in the following:

- **DRAM and Core Models:** The DRAM model of the framework is based on a DRAM specific TLM protocol called *DRAM-AT* [20]. Due to TLM's modular fashion several types of DRAM and controller configurations can be modeled. For modeling the cores the *gem5* simulator is used [2]. We developed a coupling

between gem5 and SystemC to be able to integrate this powerful research platform in our simulation loop [19].

- **DRAM Power Model:** Since DRAMs contribute significantly to the power consumption of today’s systems [9, 27], there is a need for accurate power modeling. For our framework we use DRAMPower [4, 5], which uses either parameters from datasheets, estimated via DRAMSpec [33] or measurements to model DRAM power.
- **Thermal Model:** 3D packaging of systems like Wide I/O DRAM starts to break down the memory and bandwidth walls. However, this comes at the price of increased power density and less horizontal heat removal capability of the thinned dies. Therefore, we integrated the thermal simulator 3D-ICE [38] in a SystemC wrapper [18] that is included in our closed-loop simulation for Approximate DRAM analysis.

In a detailed case study [22] we used the presented simulation framework (Fig. 11) to investigate the influence of Approximate DRAM on three different applications. We achieved in average a more than 10% decrease of the total energy consumption.

4 Conclusions

Technology scaling is leading to a point where traditional worst-case design is no longer feasible. In this chapter, we presented a new methodology for the design of dependable wireless systems. We combined cross-layer reliability techniques to treat hardware errors with the least possible overhead leading to a high energy efficiency. This methodology enables efficient trade-offs between communications performance, throughput, and energy efficiency. However, the exact trade-off depends on the real application requirements, which was not in the focus of this work. Application-specific resilience actuators together with low-level techniques offer the ability to respond to the changing requirements on reliability and quality-of-service. We illustrated our new methodology on a state-of-the-art generic double-iterative MIMO-BICM receiver which belongs to the most complex systems in modern communication standards.

We identified dynamic resilience actuators on all layers of abstraction. Each actuator offers a trade-off between communications performance, implementation performance (throughput, power), and error resilience. Any actuator which trades off communications performance for throughput, e.g., the sphere radius, can be reused to increase the error resilience, when combined with a reduction of the clock frequency. Throughput and error resilience are, thus, closely related. As we have shown, algorithmic resilience actuators offer a great potential for dynamic trade-offs between communications performance, implementation performance, and error resilience. This work emphasizes the strong mutual dependencies between these three design metrics in a wireless receiver.

When the requirement in communication systems on data block sizes exceeds the capacities of the on-chip memories (SRAMs), external memories, such as DRAMs, have to be used. To reduce their impact on energy and performance we exploited the concept of Approximate DRAM. However, this comes at the cost of reduced reliability. For the exploration of approximate DRAMs we introduced a holistic simulation framework that includes an advanced DRAM retention error model. This model is calibrated to real measurements of recent DRAM devices. Finally, we demonstrated using the holistic simulation platform that the impact of Approximate DRAM on the quality (QoS or QoR) is negligible while saving refresh energy for three selected applications.

References

1. Bhati, I., Chishti, Z., Lu, S.L., Jacob, B.: Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, pp. 235–246. ACM, New York (2015)
2. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. SIGARCH Comput. Archit. News **39**(2), 1–7 (2011). <http://doi.acm.org/10.1145/2024716.2024718>
3. Brehm, C., May, M., Gimmler, C., Wehn, N.: A case study on error resilient architectures for wireless communication. In: Proceedings of the Architecture of Computing Systems, pp. 13–24 (2012)
4. Chandrasekar, K., Akesson, B., Goossens, K.: Improved power modeling of DDR SDRAMs. In: 2011 14th Euromicro Conference on Digital System Design (2011). <http://dx.doi.org/10.1109/DSD.2011.17>
5. Chandrasekar, K., Weis, C., Li, Y., Akesson, B., Naji, O., Jung, M., Wehn, N., Goossens, K.: DRAMPower: Open-source DRAM power & energy estimation tool (2012). <http://www.drampower.info>
6. Das, S., Tokunaga, C., Pant, S., Ma, W.H., Kalaiselvan, S., Lai, K., Bull, D.M., Blaauw, D.T.: RazorII: in situ error detection and correction for PVT and SER tolerance. IEEE J. Solid State Circuits **44**(1), 32–48 (2009). <https://doi.org/10.1109/JSSC.2008.2007145>
7. Designing Chips without Guarantees. IEEE Design & Test of Computers **27**(5), 60–67 (2010). <https://doi.org/10.1109/MDT.2010.105>
8. Dixit, A., Wood, A.: The impact of new technology on soft error rates. In: Proceedings of the IEEE International Reliability Physics Symposium (IRPS) (2011). <https://doi.org/10.1109/IRPS.2011.5784522>
9. Farahini, N., Hemani, A., Lansner, A., Clermidy, F., Svensson, C.: A scalable custom simulation machine for the Bayesian confidence propagation neural network model of the brain. In: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 578–585 (2014). <https://doi.org/10.1109/IRPS.2011.578452210.1109/ASPDAC.2014.6742953>
10. Gimmler, C., Lehnigk-Emden, T., Wehn, N.: Low-complexity iteration control for MIMO-BICM systems. In: Proceedings of the IEEE 21th International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC 2010. Istanbul (2010)
11. Gimmler-Dumont, C., Wehn, N.: A cross-layer reliability design methodology for efficient, dependable wireless receivers. ACM Trans. Embed. Comput. Syst. **13**, 1–29 (2014)
12. Gimmler-Dumont, C., Brehm, C., Wehn, N.: Reliability study on system memories of an iterative MIMO-BICM system. In: Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration 2012 (2012)

13. Gimmller-Dumont, C., Kienle, F., Wu, B., Masera, G.: A system view on iterative MIMO detection: dynamic sphere detection versus fixed effort list detection. *VLSI Design J.* **2012**, Article ID 826350 (2012). <https://doi.org/10.1155/2012/826350>
14. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
15. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Hartig, H., Hedrich, L., et al.: Design and architectures for dependable embedded systems. In: 2011 Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 69–78. IEEE, Piscataway (2011)
16. Herkersdorf, A., Aliee, H., Engel, M., Glab, M., Gimmller-Dumont, C., Henkel, J., Kleeberger, V.B., Kochte, M.A., Kühn, J.M., Mueller-Gritschneider, D., Wehn, N., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectr. Reliab.* **54**(6), 1066–1074 (2014)
17. Hochwald, B., ten Brink, S.: Achieving near-capacity on a multiple-antenna channel. *IEEE Trans. Commun.* **51**(3), 389–399 (2003). <https://doi.org/10.1109/TCOMM.2003.809789>
18. Jung, M.: Icewrapper - a systemC wrapper for 3D-ICE (2015). <http://www.uni-kl.de/3d-dram/tools/icewrapper/>
19. Jung, M., Wehn, N.: Coupling gem5 with systemC TLM 2.0 virtual platforms. In: gem5 User Workshop, International Symposium on Computer Architecture (ISCA). Portland (2015)
20. Jung, M., Weis, C., Wehn, N., Chandrasekar, K.: TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration. In: Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO '13, pp. 5:1–5:6. ACM, New York (2013). <http://doi.acm.org/10.1145/2432516.2432521>
21. Jung, M., Weis, C., Wehn, N.: DRAMSys: A flexible DRAM subsystem design space exploration framework. *IPSJ Trans. Syst. LSI Design Methodol.* **8**, 63–74 (2015)
22. Jung, M., Zulian, E., Mathew, D., Herrmann, M., Brugger, C., Weis, C., Wehn, N.: Omitting refresh - A case study for commodity and wide I/O DRAMs. In: 1st International Symposium on Memory Systems (MEMSYS 2015). Washington (2015)
23. Jung, M., Mathew, D.M., Weis, C., Wehn, N.: Efficient reliability management in SoCs - An approximate DRAM perspective. In: 21st Asia and South Pacific Design Automation Conference (ASP-DAC) (2016)
24. Kienle, F.: Low-Density MIMO Codes. In: Proceedings of the 5th International Symposium on Turbo Codes and Related Topics, pp. 107–112. Lausanne (2008)
25. Kienle, F., Wehn, N., Meyr, H.: On complexity, energy- and implementation-efficiency of channel decoders. *IEEE Trans. Commun.* **59**(12), 3301–3310 (2011). <https://doi.org/10.1109/TCOMM.2011.092011.100157>
26. Kleeberger, V., Gimmller-Dumont, C., Weis, C., Herkersdorf, A., Mueller-Gritschneider, D., Nassif, S., Schlichtmann, U., Wehn, N.: A cross-layer technology-based study of the impact of memory errors on system resilience. *IEEE Micro* **33**(4), 46–55 (2013)
27. Krueger, J., Donofrio, D., Shalf, J., Mohiyuddin, M., Williams, S., Oliker, L., Pfrendt, F.J.: Hardware/software co-design for energy-efficient seismic modeling. In: Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12 (2011)
28. Lee, D., Kim, Y., Pekhimenko, G., Khan, S., Seshadri, V., Chang, K., Mutlu, O.: Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 489–501 (2015). <https://doi.org/10.1109/HPCA.2015.7056057>
29. Liu, J., Jaiyen, B., Veras, R., Mutlu, O.: RAIDR: Retention-aware intelligent DRAM refresh. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12, pp. 1–12. IEEE Computer Society, Washington (2012). <http://dl.acm.org/citation.cfm?id=2337159.2337161>

30. Lucas, J., Alvarez-Mesa, M., Andersch, M., Juurlink, B.: Sparkk: Quality-scalable approximate storage in DRAM. In: The Memory Forum (2014). <http://www.redaktion.tu-berlin.de/fileadmin/fg196/publication/sparkk2014.pdf>
31. May, M., Alles, M., Wehn, N.: A case study in reliability-aware design: A resilient LDPC code decoder. In: Proceedings of the Design, Automation and Test in Europe DATE '08, pp. 456–461. Munich (2008)
32. Mitra, S., Brelsford, K., Kim, Y.M., Lee, H.H.K., Li, Y.: Robust system design to overcome CMOS reliability challenges. IEEE J. Emer. Sel. Topics Circuits Syst. 1(1), 30–41 (2011). <https://doi.org/10.1109/JETCAS.2011.2135630>
33. Naji, O., Weis, C., Jung, M., Wehn, N., Hansson, A.: A high-level DRAM timing, power and area exploration tool. In: Embedded Computer Systems Architectures Modeling and Simulation (SAMOS) (2015)
34. Nazar, G.L., Gimmler, C., Wehn, N.: Implementation comparisons of the QR decomposition for MIMO detection. In: Proceedings of the 23rd Symposium on Integrated Circuits and System Design (SBCCI '10), pp. 210–214. ACM, New York (2010). <https://doi.org/10.1145/1854153.1854204>
35. Nowka, K., Nassif, S., Agarwal, K.: Characterization and design for variability and reliability. In: Proceedings of the IEEE Custom Integrated Circuits Conference CICC 2008, pp. 341–346 (2008). <https://doi.org/10.1109/CICC.2008.4672092>
36. Qian, W., Li, X., Riedel, M., Bazargan, K., Lilja, D.: An architecture for fault-tolerant computation with stochastic logic. IEEE Trans. Comput. 60(1), 93–105 (2011). <https://doi.org/10.1109/TC.2010.202>
37. Sadri, M., Jung, M., Weis, C., Wehn, N., Benini, L.: Energy optimization in 3D MPSoCs with Wide-I/O DRAM using temperature variation aware bank-wise refresh. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–4 (2014). <https://doi.org/10.7873/DATE2014.294>
38. Sridhar, A., Vincenzi, A., Ruggiero, M., Brunschwiler, T., Atienza, D.: 3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling. In: Proceedings of the IEEE International Conference on Computer-Aided Design ICCAD 2010 (2010)
39. Weis, C., Jung, M., Ehses, P., Santos, C., Vivet, P., Goossens, S., Koedam, M., Wehn, N.: Retention time measurements and modelling of bit error rates of WIDE I/O DRAM in MPSoCs. In: Proceedings of the IEEE Conference on Design, Automation & Test in Europe (DATE). European Design and Automation Association (2015)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Uncertainty-Aware Compositional System-Level Reliability Analysis



Hananeh Aliee, Michael Glaß, Faramarz Khosravi, and Jürgen Teich

Acronyms

BDD	binary decision diagram
CRA	compositional reliability analysis
CRN	compositional reliability node
CDF	cumulative distribution function
DSE	design space exploration
EA	evolutionary algorithm
ESL	electronic system level
MOEA	multi-objective evolutionary algorithm
MPSoC	multiprocessor system-on-chip
MTTF	mean-time-to-failure
RAL	reliability abstraction level
RTC	Real-Time Calculus
SER	soft error rate

H. Aliee
Helmholtz Zentrum München, Munich, Germany
e-mail: hananeh.aliee@helmholtz-muenchen.de

M. Glaß (✉)
Ulm University, Ulm, Germany
e-mail: michael.glass@uni-ulm.de

F. Khosravi · J. Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
e-mail: faramarz.khosravi@fau.de; juergen.teich@fau.de

1 Introduction

Continuous technology scaling necessitates to design today’s embedded systems from electronic components with growing inherent unreliability. This unreliability arises from susceptibility to neutron-induced soft errors, negative-bias temperature instability, short-channel effect, gate leakage, etc. Therefore, it is vital to analyze system reliability at design time and employ appropriate reliability-improving techniques if necessary. A variety of reliability analysis techniques have been proposed for both the relatively low levels of abstraction that focus on technology as well as the system level that considers the interplay of hardware and software. But, there exists a gap between the levels where the faults originate, e. g., transistor level, and the system level for which the analysis is required. To close this gap and tame the ever increasing system complexity, *cross-level* analysis methodologies are required. These collect knowledge at lower levels by combining different analysis techniques and provide proper data for the analysis at higher levels of abstraction [24].

Evaluating the reliability of a system at design time, proper reliability-improving techniques can be explored and integrated into the system. However, these techniques typically come with higher monetary costs, latency, energy consumption, etc. This necessitates a multi-objective **DSE!** (**DSE!**) which maximizes reliability without deteriorating other design objectives. Usually, **DSE!** explores and evaluates millions of possible design alternatives (also called implementations) to find the Pareto-optimal ones. Herein, the efficiency of the reliability evaluation and exploration algorithm are the main challenging issues [2]. In [3], we propose an efficient and scalable reliability analysis technique based on Success Trees (STs) which is integrated into a **DSE!** framework to automatically evaluate an implementation’s reliability. Most existing analysis techniques quantify a system’s reliability without giving any hint on what to change to improve it, such that exploration algorithms basically perform random changes, e. g., through genetic operators in case of EA!s (EA!). In [4, 6, 7, 11], we propose to employ the notion of *component importance* to rank components based on their contribution to the system reliability. Later, to improve the reliability of a system with limited budgets, we only need to improve the reliability of highly important components. In [5, 28], we show this guides the **DSE!** towards highly reliable, yet affordable implementations. So far, most existing analysis approaches assume that the reliabilities of components—or their lower bound—are more or less known precisely. Due to shrinking cell geometries, semiconductor devices encounter higher susceptibility to environmental changes and manufacturing tolerances such that a component’s reliability has to be considered *uncertain*. An overview of the most important types of uncertainties for system design is given in Fig. 1.

Effects of unreliability and the associated uncertainty of components can propagate to the system level and become a challenge for system-level design methodologies. Even worse, destructive effects such as extreme temperature can affect several components simultaneously, resulting in correlated uncertainties. Neglecting such correlations can impose an intolerable inaccuracy to reliability analysis.

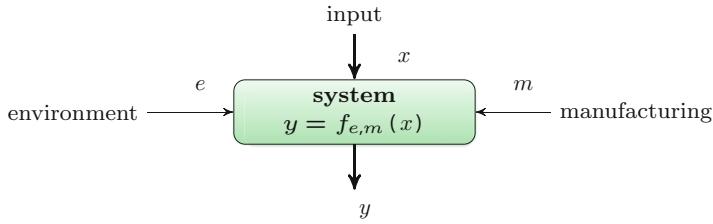


Fig. 1 Uncertainties that influence a system's response and its characteristics: Uncertain environmental influences Δe like cosmic rays may cause soft errors. Manufacturing tolerances Δm may lead to changing system behavior and permanent defects. Finally, uncertainty may also be present in case the inputs to a system may vary (Δx) or might not be known at design time

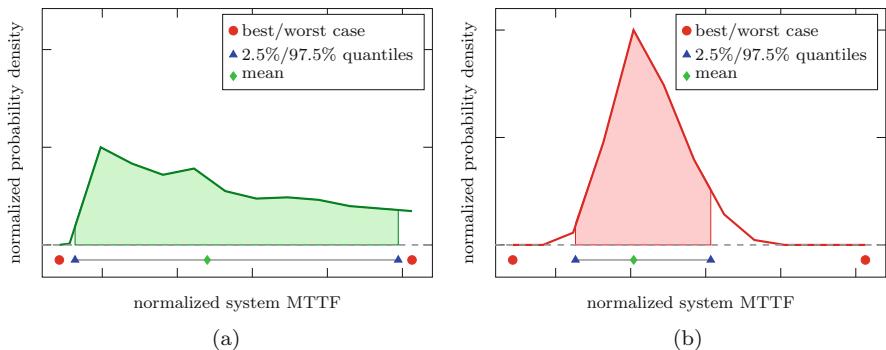
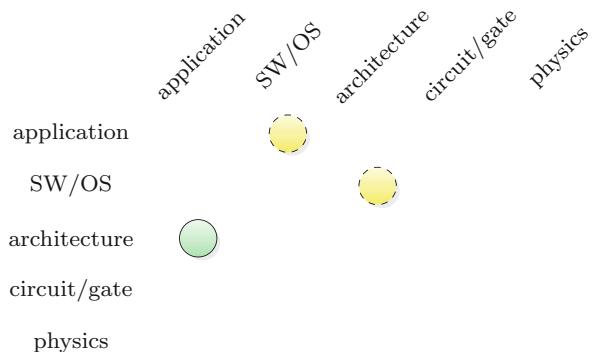


Fig. 2 Impact of uncertainty correlation among the reliability functions of different components on the uncertainty of the system **MTTF!** for an implementation candidate of an H.264 encoder/decoder. **(a)** Correlated component uncertainty. **(b)** Non-correlated component uncertainty

As an example, Fig. 2 depicts the distribution of system **MTTF!** (**MTTF!**) for an H.264 encoder/decoder implementation with and without the consideration of uncertainty correlations. While considering these correlations shows a good match between the simulated cases and the bounds, neglecting them may result in huge deviations from those bounds. This motivates the consideration of uncertainties and especially their correlations in cross-level reliability analysis. In this realm, this chapter introduces a methodology for **CRA!** (**CRA!**) that combines various reliability analysis techniques across different levels of abstraction while being aware of existing uncertainties and their correlations.

Considering uncertainty, system reliability is no longer a single value, but instead represented by a set of samples, upper and lower bound curves, or distribution functions which requires that a **DSE!** can consider implementations with uncertain objectives. Therefore, this chapter focuses on (a) the explicit modeling of uncertainties and their correlations in reliability analysis and (b) the integration of such an analysis into a framework for system-level **DSE!**. The techniques proposed are not tailored to a specific abstraction layer, but can be best classified as combining

Fig. 3 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



architecture and application layers according to the embedded system abstraction layers as depicted in Fig. 3.

The rest of this chapter is organized as follows: Sect. 2 reviews related work and introduces required fundamentals. Section 3 introduces a formal **CRA!** framework and its application using a case study. An explicit modeling of uncertainty in reliability analysis and optimization is given in Sect. 4. Finally, Sect. 5 concludes the chapter.

2 Related Work and Fundamentals

2.1 Reliability Analysis and Optimization

Reliability analysis and optimization are thoroughly studied research topics that are of great importance for nearly every safety-critical system [12], especially embedded systems [36]. However, one can observe that the different areas raise significantly diverse needs for the applied analysis techniques. An overview of well-known reliability analysis techniques can be found in [35].

Up to now, several approaches have been presented for analyzing the reliability of embedded systems at system level which are typically integrated into system-level **DSE!**. In [16], fault-tolerant schedules are synthesized using task re-execution, rollback recovery, and active replication. The authors of [47] try to maximize reliability by selectively introducing redundancy while treating area consumption and latency as constraints. Reliability is introduced as an objective into system-level design in [13]. However, the employed reliability analysis techniques are restricted to series-parallel system structures which render them infeasible for typical embedded systems where processing and communication resources have to be shared. On the other hand, reliability analysis at low levels of abstraction has been studied thoroughly, e. g., transistor level [42] or for prospective switching devices like carbon nanotubes [32].

So far, few systematic approaches have been proposed to upscale the knowledge gathered at low abstraction levels to the system level. The work in [1] proposes a system-level dynamic temperature management scheme to prevent thermal hot spots through a run-time application re-mapping, and to efficiently mitigate aging effects in a many-core architecture. In [23], thermal effects in a Multiprocessor System-on-Chip (MPSoC) on its reliability are propagated into a scheduling and binding optimization at system level. Their analysis is based on a simulation of the MPSoC and given relations between the temperature profile and the resulting reliability. Similar reliability analysis techniques are used in [34] in order to optimize the lifetime of MPSoCs using the so-called *slack allocation*. However, these techniques are able to capture thermal effects only, without investigating the possibility to include and propagate these effects into a more holistic analysis that also takes into account, e. g., soft errors or a complex system structure like a networked embedded system consisting of several interconnected processors or MPSoCs.

2.2 Compositional Approaches to Reliability Analysis

A first attempt to close the gap on accurate power models for reliability analysis between the **ESL!** (**ESL!**) and the gate level is presented in [40]. While the approach sounds promising in modeling thermal effects on the component reliability, it fails to offer a formal framework that allows to integrate different analysis techniques cross level. Herkersdorf et al. [24] [RAP-Chap.] propose a framework for probabilistic fault abstraction and error propagation and show that all physically induced faults manifest in higher abstraction levels as a single or multiple bit flip(s). Similarly, the proposed **CRA!** model aims to propagate the effects of uncertainty and the resulting faults originating from lower levels of abstraction into the system-level analysis by incorporating appropriate reliability analysis techniques for each relevant *error model* at a specific level of abstraction. As a result, the developed concepts become independent of an actual error model since it abstracts from the actual source of unreliability during *upscale*, i. e., the propagation of data from lower levels to higher levels by means of abstraction and data conversion. **CRA!** approaches that consider component-based software are presented in [37]. Although these approaches try to develop a more general compositional analysis scheme, they miss a well-defined mathematical underpinning and do not focus on automatic analysis as needed during **DSE!**.

The use of composition and decomposition in well-defined formal models that allow abstraction to avoid state space explosion has been addressed in, e.g., [25]. An especially interesting and formally sound approach can be found in [9]. In this chapter, we develop a formal approach, inspired by techniques from the verification area, for **CRA!**. A particular challenge will be the consideration and explicit modeling of uncertainties in the formal model where there is no similar technique or need in the area of verification given.

2.3 Uncertainty Considerations

There exist intense studies on the effect of uncertainties on the system reliability for general engineering problems, cf. [38], as well as circuits and microarchitectures, cf. [27]. However, few studies focus on the cross-level reliability analysis of embedded systems in the presence of uncertainty arising from manufacturing tolerances, etc.

Uncertainty-Aware Analysis One example is a cross-level adaptive reliability prediction technique proposed in [17] that derives information from different levels of abstraction and allows to consider the simultaneous effects of process, voltage, temperature and aging variations, and soft errors on a processor. The authors of [18] propose a cross-level framework to analyze the combined impact of aging and process variation on the **SER!** (**SER!**) and static noise margin of memory arrays in near threshold voltage regimes. This framework enables to explore workload, as instruction per second, and cache configuration, as cache size and associativity, in order to minimize **SER!** and its variations for 6T and 8T SRAM cells. Contrary to all mentioned approaches, this chapter explicitly treats each effect of uncertainty during reliability analysis of a system. Proposed is an analysis technique that obtains the range of reliability that is achievable for a system given its configuration and the uncertainties of its components.

Uncertainty-Aware Optimization Optimization problems may be affected by various sources of uncertainty including perturbation of decision variables as well as effects of noise and approximation on objective functions [26]. In this work, uncertainty is explicitly modeled as variations in component failure rates and costs. The uncertainty propagates through reliability analysis and cost evaluation at system level and renders design objectives to be uncertain as well. To make correct decisions when comparing and discriminating implementations during **DSE!**, the employed optimization algorithm needs to take the uncertainty of the design objectives into account as well. The work in [44] proposes a mathematical approach to calculate the probability of an implementation dominating another, given all uncertain objectives follow either uniform or any discrete distributions. However, extending this approach to consider diversely distributed uncertain objectives requires solving difficult integrals demanding a huge computational effort. To this end, approximate simulation-based approaches, e. g., in [30], provide trade-offs between execution time and accuracy of calculating this probability. In [33], it is proposed to compare uncertain objectives with respect to their lower and upper bounds. However, this approach fails to distinguish largely overlapping intervals with even significantly different distributions. A lot of work has been proposed for problems with continuous search spaces and linear objective functions, see e. g., [15]. However, typical embedded system design problems have discrete search spaces, non-linear and often not differentiable objective functions, and have to cope with stringent constraints. Thus, these optimization techniques cannot be applied without further investigation and modification. In [39], an approach based on an

uncertainty-aware **MOEA!** (**MOEA!**) that targets reliability as one design objective is presented. The approach takes into account the uncertainty of the reliability of each system component and tries to maximize the robustness of the system. This chapter presents a novel uncertainty-aware multi-objective optimization approach applicable for **DSE!** of reliable systems at system level, see Sect. 4.

2.4 System-Level Design Fundamentals

This chapter targets the system-level design of embedded MPSoCs, typically specified by an application graph, a resource graph, and a set of possible task-to-resource mappings. The application graph includes a set of tasks to be executed and specifies the data and control flow among them. The resource graph consists of hardware resources, namely, processors and accelerators connected by communication infrastructures such as buses or networks-on-a-chip. The mappings specify which tasks can be executed on which resources. Figure 4 shows an example specification with three tasks $t_i, i \in [0 \dots 2]$, five resources $r_j, j \in [0 \dots 4]$, and eight mappings $m_{i,j}$ from t_i to r_j .

Implementation candidates are derived via system-level synthesis [10] performing the steps: (a) *Resource allocation* selects a subset of resources that are part of the implementation. (b) *Task binding* associates at least one instance of each task to an allocated resource by activating the respective task-to-resource mapping. (c) *Scheduling* determines a feasible start time for each task instance. An implementation is *feasible* if and only if all constraints regarding, e. g., communication, timing, or utilization are fulfilled. Figure 4 highlights a possible feasible implementation

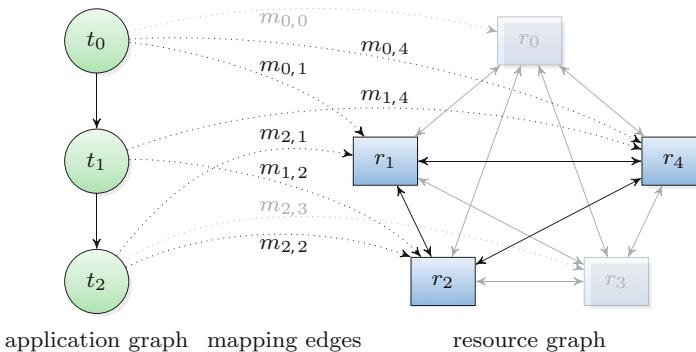


Fig. 4 A specification comprising (a) an application graph where edges indicate data dependencies of tasks, (b) a resource graph with edges representing dedicated communication between resources, and (c) a set of task-to-resource mappings which model possible execution of tasks on resources. A possible implementation candidate obtained by system-level synthesis is depicted with non-allocated resources and non-active bindings being grayed out

with non-allocated resources and non-activated mappings being grayed out. More details of the underlying system synthesis and **DSE!** in the context of reliability analysis and optimization can be found in [22, 43].

3 Compositional Reliability Analysis (CRA)

This section introduces models and methods for **CRA!** as proposed in [21]. Figure 5 shows a schematic view of **CRA!** and its required mechanisms. To realize a cross-level analysis, it encapsulates existing reliability analysis techniques in **CRN!**s (**CRN!**s) at multiple **RAL!**s (**RAL!**s). It tames analysis complexity within a certain **RAL!** using composition and decomposition and connects different **RAL!**s through adapters. Each **CRN!** applies an analysis step $\mathcal{Y}(t) = X(S)$ at a specific **RAL!** where X is a concrete analysis technique and S is a (sub)system. A **CRN!** derives a specific measure \mathcal{Y} over time t . A **RAL!** in **CRA!** may combine several (design) abstraction levels where the same errors and, especially, their causes are significant.

Adjacent **RAL!**s are connected by the concept of *adapters* that have to perform three tasks: (a) *refinement* provides the data required for the analysis in the lower **RAL!**, (b) *data conversion* transforms the output measures from the lower **RAL!** to the input required at the higher **RAL!**, and (c) *abstraction* during both refinement and data conversion tames analysis complexity. A concrete example of **CRA!** describing a temperature-reliability adapter for MPSoCs is presented in Sect. 3.1.

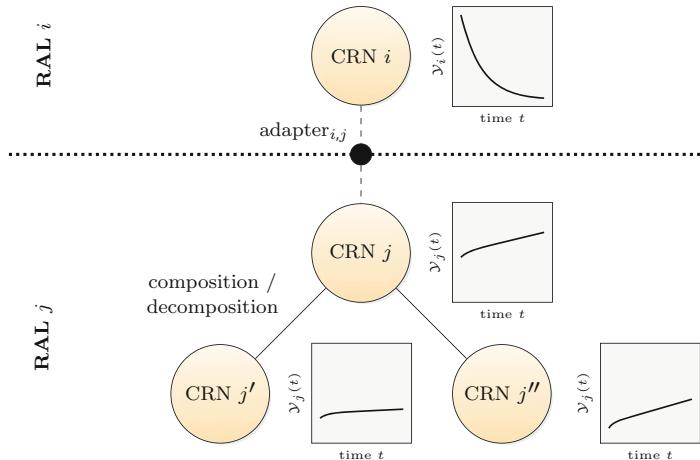


Fig. 5 A schematic view of **CRA!**

Another important aspect of **CRA!** concerns the feasibility of composition and decomposition with respect to reliability analysis. While, of course, composition and decomposition should reduce the complexity of the analysis, errors caused by approximation or abstraction should be bounded. For example, a rule to bound the approximation error of a decomposition D of a given system S into n subsystems S_1, \dots, S_n is as follows:

$$D(S) = \{S_1, \dots, S_n\} \text{ is feasible, if } \exists \epsilon : |X(S) - (X(S_1) \circ \dots \circ X(S_n))| \leq \epsilon$$

with \circ being an analysis-dependent operator, e. g., multiplication, and ϵ being the maximum approximation error. A special focus of these investigations is the proper handling of decomposed nodes that influence each other. Nowadays, hardly any subsystem of an embedded system is truly independent of all other subsystems. Thus, this rule should be extended as follows to consider both the truly independent individual properties of the decomposed nodes and their dependencies during composition C :

$$\begin{aligned} D(S) = \{S_1, \dots, S_n\} \text{ is feasible, if } \exists \epsilon : \\ |X(S) - C(X(S_1) \circ \dots \circ X(S_n), P(\{S_1, \dots, S_n\}))| \leq \epsilon. \end{aligned} \quad (1)$$

In this case, the composition C not only takes into account the parts of the subsystem that can be analyzed independently, but also performs a corrective postprocessing P to take into account their interactions.

Similarly, we have developed rules for the connection of different **RAL!**s. The task of an adapter is to convert the measure \mathcal{Y} used at the lower **RAL!** into the measure \mathcal{Y}' used at the higher **RAL!**s, for example, $\mathcal{Y}' = A(\mathcal{Y})$. Especially because of the models and methods needed for converting from one **RAL!** to another, a thorough analysis of the function A needs to be carried out. In most cases, this function will not provide an exact result, but will require an abstraction such as by the determination of tight upper and lower bounds. Thus, the developed rules will define requirements for the functions in the adapter used for abstraction and data conversion.

3.1 CRA Case Study and Uncertainty Investigations

In [21], a concrete application of **CRA!** to realize a temperature-aware redundant task mapping approach is presented. In the following, a brief summary of the case study is given with focus being put on the aspect of uncertainty introduced due to the application of composition and decomposition. This further motivates the need to develop techniques to explicitly model and consider uncertainty during analysis and optimization as is presented in Sect. 4.

3.1.1 CRA Case Study

In the context of system-level design and especially the design of reliable embedded systems as introduced in Sect. 2, deploying redundant (software) tasks can be considered a rather cost-efficient technique to enhance system reliability. However, the resulting additional workload may lead to increased temperature and, thus, a reliability degradation of the (hardware) components executing the tasks. The case study in [21] combines three different techniques on three **RAL!**s: At the highest **RAL!**, a reliability analysis based on **BDD!** (**BDD!**), see, e.g.,[20], computes the system reliability of a complete 8-core MPSoC and requires the reliability function of each component (core) in the system. To determine the latter, an intermediate **RAL!** uses the behavioral analysis approach **RTC!** (**RTC!**) [45] to derive the upper bound for the workload of each core over time. This workload is passed to the lowest **RAL!** where this information is used to carry out a temperature simulation based on HotSpot [41] to deliver a temperature profile of each core. Using these temperature profiles and assuming electromigration as a fault model, [21] proposes an adapter that—based on the works in [14, 46]—delivers a temperature-aware reliability function for each core back to the highest **RAL!** in order to complete the system analysis.

3.1.2 Uncertainty Investigations

As given in Eq. 1, composition/decomposition may result in an imprecision ϵ_o of an output measure $o \in O$. In [19], we present techniques for formal decomposition and composition for **CRN!**s that describe the system via Boolean formulas, typically used by **BDD!**s, Fault Trees, etc. Here, functional correlations between components are fully captured in the Boolean formulas, and we propose an exact composition/decomposition scheme on the basis of early quantification. However, correlations are typically non-functional, with heat dissipation between adjacent cores being a prominent one. Consider again the case study described before and Fig. 6: Not decomposing the system into individual cores results in a temperature simulation of all cores at the lowest level, implicitly including the effect of heat dissipation in-between cores, see Fig. 6 (top-left). A naive decomposition could decompose the system into independent cores such that the workload of each core is determined and a reliability function would be gathered by per-core temperature simulations on the lowest level, see Fig. 6 (middle-left). This, however, would completely neglect the effect of heat dissipation between cores. As a third option, [21] investigates a *corrective postprocessing* within the adapter between the lower levels where the workload of cores and the temperature simulation are analyzed independently, while a simple model that considers the distance and steady-state temperature of each core is used to approximate the respective heat flow, see Fig. 6 (bottom-left).

The imprecision resulting from the three discussed decomposition variants is given in Fig. 6 (right), derived from ≈ 8000 different system implementations

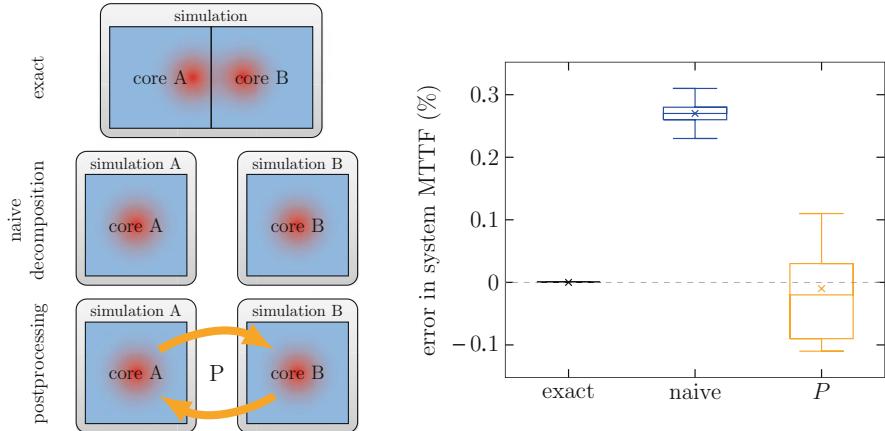


Fig. 6 A simulation of two cores captures heat dissipation (exact, top) while a decomposition (naive, middle) is unable to capture heat dissipation between cores. A corrective postprocessing (P , bottom) enables to reduce analysis complexity while providing a basic notion of heat dissipation. The resulting imprecisions in percentage on system-wide **MTTF!** are depicted on the right

analyzed as part of a **DSE!**: While no decomposition is treated as an exact base value—with respect to heat dissipation being considered and not the overall exactness of the simulation—the naive decomposition constantly overestimates the system-wide **MTTF!** by $\approx 26\%$. On the other hand, the corrective postprocessing delivers results with a rather good match in terms of the median and average error, but also shows that the correction may come at errors of up to $\approx 10\%$. At the same time, compared to the complete simulation, the decomposition including corrective postprocessing achieves a $\approx 2 \times$ average speed-up. These results further motivate the need for analysis and optimization techniques—as presented in the next section—that can explicitly model uncertainty such as the shown imprecision.

4 Uncertainty in Reliability Analysis and Optimization

To design and optimize systems for reliability, existing uncertainties in their environment and internal states, see Fig. 1, must be explicitly integrated into reliability analysis techniques. Implicit uncertainty modeling hides the effects of controllable and non-controllable uncertainties, e. g., into a single reliability function, and fails to distinguish between them. On the other hand, explicit modeling determines the range of achievable reliability of a component or subsystem, e. g., using upper and lower bound functions.

We introduce two solutions for uncertainty modeling: (a) using upper and lower bounding curves for the achievable reliability and (b) abstracting various uncertainties into a finite set of typical use cases and providing a system reliability

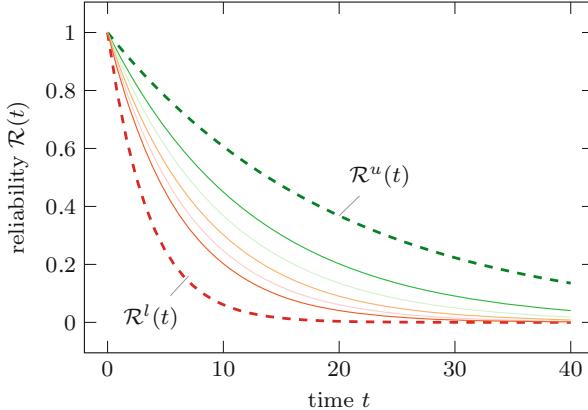


Fig. 7 Reliability functions $\mathcal{R}(t)$ that result from uncertainties in the internal heat dissipation of a silicon system that arise from, e.g., changing task binding on a processing unit. Shown is a range that is determined by an upper $\mathcal{R}^u(t)$ and a lower bound $\mathcal{R}^l(t)$ reliability function and the reliability functions for 5 use cases, e.g., 5 favored task schedules that show a distribution within the range

function for each case, see Fig. 7. While the former offers a range and abstracts from the distributions in between bounds, the latter variant explicitly determines important cases in that range, but of course, comes with an increased complexity. This section covers both approaches and assumes that uncertainty obtained from lower abstraction levels is available at higher levels as known distributions or sampled data.

As introduced earlier, incorporating reliability-increasing techniques into a system at design time may deteriorate other design objectives. Due to the explicit modeling of uncertainties, a multi-objective uncertainty-aware optimization becomes necessary. Given that the system reliability is no more a single value, optimization algorithms must be able to handle uncertain objectives given as probability distributions, a set of samples or upper and lower bound curves, and allow for a quantitative comparison of different designs.

4.1 Uncertainty-Aware Reliability Analysis

The uncertainty-aware reliability analysis technique introduced in the following is originally proposed in [29]. It models the reliability of a component r with uncertain characteristics \mathcal{U}_r using reliability functions $\mathcal{R}_r(t)$ that are distributed within given lower and upper bound reliability functions, i.e., $\mathcal{U}_r = [\mathcal{R}_r^l(t), \mathcal{R}_r^u(t)]$. A *sampler* is used to take \mathcal{U}_r as input and deliver a sampled reliability function $\mathcal{R}_r^s(t)$ with $\mathcal{R}_r^l(t) \leq \mathcal{R}_r^s(t) \leq \mathcal{R}_r^u(t)$. It ensures that the sampled reliability functions follow the intended distribution within the given bounds, and enables

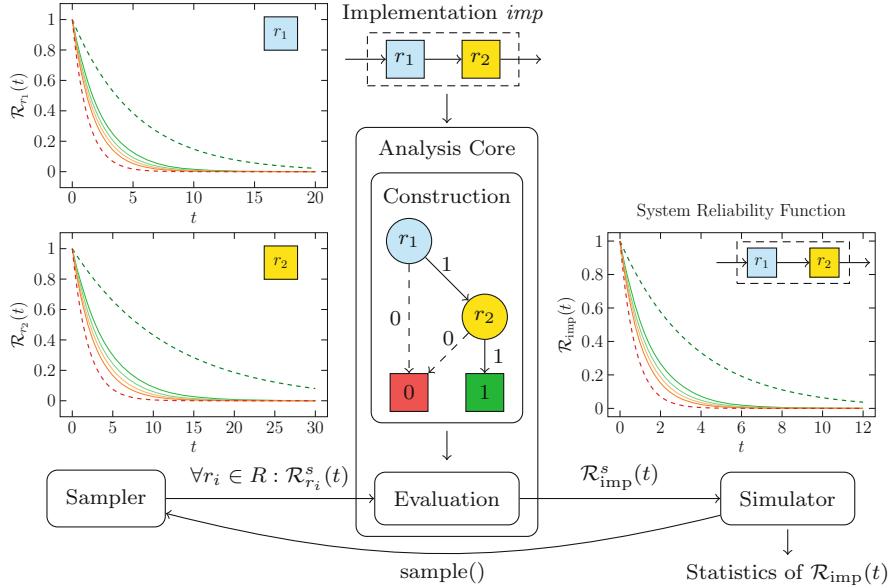


Fig. 8 An overview of the proposed uncertainty-aware reliability analysis

the consideration of arbitrary distributions, in particular well-known discrete and continuous distributions.

In practice, component reliability is typically derived from measurements that are fitted to closed-form exponential ($\mathcal{R}_r(t) = e^{-\lambda_r \cdot t}$) and Weibull ($\mathcal{R}_r(t) = e^{-\lambda_r \cdot t^{b_r}}$) reliability functions with λ being the component's failure rate. The uncertainty model \mathcal{U}_r includes a set of uncertain parameters P_r , distributed within the bounds $[P_r^l, P_r^u]$. The sampler takes a sample from each parameter $p_r \in P_r$ and constructs a sample reliability function. For example, for an exponential distribution with bounds $[\lambda_r^l, \lambda_r^u] = [0.0095, 0.0099]$, a sample reliability function $\mathcal{R}_r^s(t) = e^{-0.0098t}$ can be generated.

The overall flow of the analysis approach is shown in Fig. 8 and includes the following steps: (a) The sampler samples a reliability function $\mathcal{R}_r^s(t)$ from the uncertainty distribution of each component r , (b) an analysis core uses these samples and calculates a sample reliability function for the given system implementation $\mathcal{R}_{imp}^s(t)$, and (c) a statistical *simulator* collects a set of sampled reliability functions $\Phi_{imp} = \bigcup_{s=1}^n \{\mathcal{R}_{imp}^s(t)\}$ and constructs the uncertainty distribution of the system reliability function.

The analysis core can be realized by any existing technique that requires a reliability function of each component and calculates the system reliability function. In the concrete case, Fig. 8 shows a formal technique based on **BDD!**s that models the reliability of a system implementation with two components in series. The statistical simulator determines the number of required samples n to later obtain

desired statistics like mean and quantiles from Φ_{imp} with a guaranteed confidence level.¹ As an example, for each sample $\mathcal{R}_{\text{imp}}^s(t)$ in Φ_{imp} , the **MTTF!** can be calculated using the integration below:

$$\text{MTTF!}_s = \int_0^\infty t \cdot f^s(t) dt \quad \text{where} \quad f^s(t) = -\frac{d\mathcal{R}_{\text{imp}}^s(t)}{dt}. \quad (2)$$

Using sample **MTTF!**s, design objectives such as the best-, worst-, and average-case **MTTF!** can be derived.

4.1.1 Uncertainty Correlation

To model any existing correlation between uncertain parameters of system components, we investigate whether they are exposed to common uncertainty sources, and are, thus, subject to correlative variations. Take temperature as an example: Components that are fabricated in the same package may be exposed to the same temperature, which means their reliability characteristics can be considered in a *correlation group*, whereas components in different packages might be considered independent. Assuming that the uncertainty sources and the correlation groups are given, we introduce models for obtaining correlated samples from the uncertainty distribution of component reliability functions in [29, 31]: To sample from an uncertain parameter p , we check if it is a member of any correlation group or not. If p is a member of G , we first generate a random probability g for the group G at the beginning of each implementation evaluation step and then calculate a sample from p using the inverse **CDF!** (**CDF!**) of the probability distribution of p at point g . Otherwise, a sample is taken independently from the distribution of p . Note that since the uncertain parameters in a correlation group might be differently distributed, returning the same quantile g from their distributions does not necessarily yield the same value, see Fig. 9. Thus, through sampling, the uncertain parameters in G vary together, and their variations are independent of those of the parameters outside G .

4.2 Uncertainty-Aware Multi-Objective Optimization

Finally, to enable the optimization of system implementations with multiple uncertain objectives, we propose an uncertainty-aware framework in [29]. It extends a state-of-the-art **DSE!** [43] and employs a **MOEA!** as the optimization core. These techniques introduce *dominance criteria* to compare different implementations and select which one to store in an *archive* and vary for the next iteration.

¹Efficient sampling techniques [8] can be used to reduce the number of required samples.

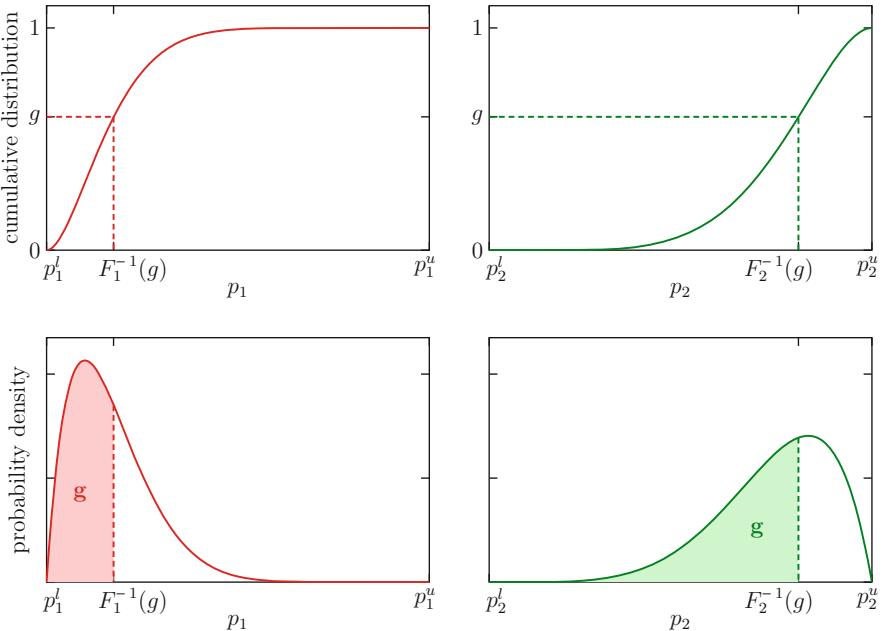


Fig. 9 Generating samples for correlated uncertain parameters p_1 and p_2

To maximize m objectives O_1, \dots, O_m , each being a single value, the dominance of two implementations A and B is defined as follows:

$$A > B \iff \forall i \in [1, v] : O_A(i) \geq O_B(i) \wedge \exists j \in [1, v] : O_A(j) > O_B(j) \quad (3)$$

Here, $A > B$ means “ A dominates B ” and $O_A(j) > O_B(j)$ means “ A is better than B in the j -th objective.” Since this dominance criterion compares each of the m objectives independently, we refer to $O(i)$ as O for brevity.

The proposed uncertainty-aware optimization compares uncertain objectives using the following three-stage algorithm: (a) If the intervals of O , specified by the lower bound O^l and upper bound O^u , of two implementations A and B do not overlap, one is trivially better ($>$) than the other. (b) If the intervals overlap, we check if one objective is significantly better with respect to an *average criterion*, e.g., mean, mode, or median. (c) If the average criterion does not find a preference, a *spread criterion* compares objectives based on their deviation, e.g., standard deviation, variance, or quantile intervals, and judges whether one is considerably better. In case none of the three stages determines that one objective is better, the objectives are considered equal. The flow of this comparison operator is illustrated in Fig. 10.

To find if one uncertain objective has significantly better average O^{avg} or deviation O^{dev} compared to the other, we use two configurable threshold values

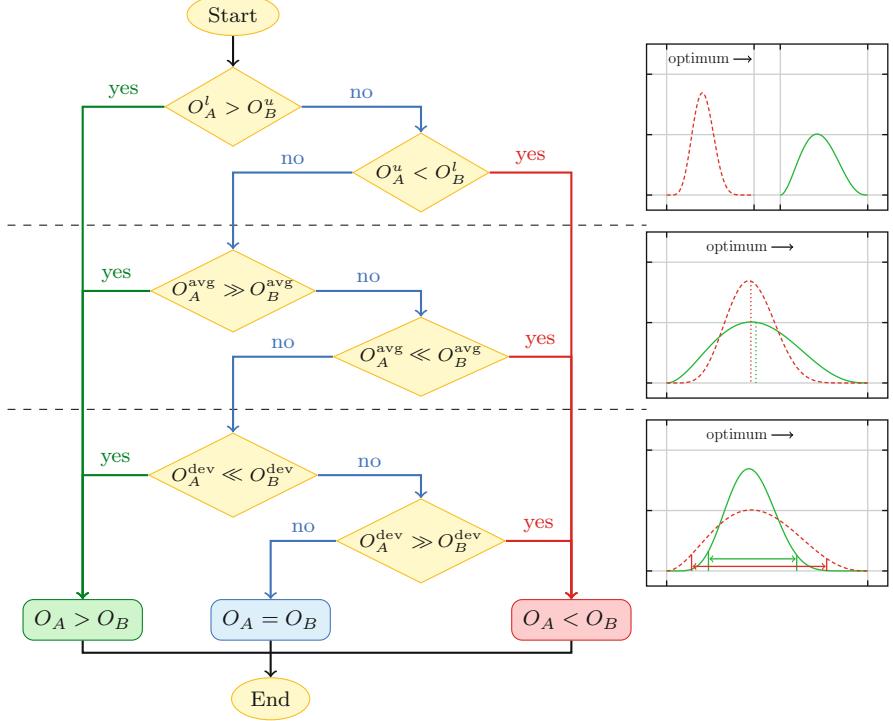


Fig. 10 The flow of the proposed three-stage comparison operator

ε^{avg} and ε^{dev} , respectively. For the average criterion, a configurable threshold value ε^{avg} determines if the difference of the considered average-case objective values is significant with respect to the given objective bounds. This enables to control the sensitivity of the second stage of the comparison:

$$\frac{O_A^{\text{avg}} - O_B^{\text{avg}}}{\max(O_A^u, O_B^u) - \min(O_A^l, O_B^l)} \geq \varepsilon^{\text{avg}}. \quad (4)$$

Here, $\varepsilon^{\text{avg}} = 0$ always prefers the objective with better average case, while $\varepsilon^{\text{avg}} = 1$ renders the average criterion ineffective since the left-hand side of Eq. (4) is always less than one. Thus, the scope of ε^{avg} must be carefully selected based on the objective's criticality to guarantee a required precision.

The spread criterion prefers the objective value with smaller deviation and uses a threshold value ε^{dev} to control the sensitivity of the comparison, i. e.,

$$\frac{O_B^{\text{dev}} - O_A^{\text{dev}}}{O_A^{\text{dev}} + O_B^{\text{dev}}} \geq \varepsilon^{\text{dev}} \Rightarrow O_A > O_B. \quad (5)$$

Given $\varepsilon^{\text{dev}} = 0$, any small difference between O_A^{dev} and O_B^{dev} is reckoned, which can lead to crowding in the solution archive. It causes solution A which is indeed weakly dominated by another solution B to be regarded as non-dominated because one of its uncertain objectives has a slightly better deviation than the corresponding objective of B . On the other hand, the spread criterion becomes ineffective if $\varepsilon^{\text{dev}} = 1$ and any significant difference between deviations of two uncertain objectives would be overlooked. Therefore, the value of ε^{dev} must be carefully selected.

Note that the statistics of an uncertain objective O required in the proposed comparison operator are calculated using samples from its distribution. Given a set of n samples for O , its variance can be calculated as follows:

$$O^{\sigma^2} = \frac{1}{n} \sum_{i=1}^n (O^j - O^\mu)^2 \quad \text{where} \quad O^\mu = \frac{1}{n} \sum_{i=1}^n O^j, \quad (6)$$

with O^μ denoting the mean of the distribution of O . Moreover, to find the q^{th} quantile of this distribution, we use the *inverse empirical distribution function* which traverses the samples in the ascending order and returns the very first sample after the $q\%$ smallest samples.

Figure 11 shows the resulting Pareto fronts for optimizing MTTF! and cost of an H.264 specification using the proposed comparison operator vs. a common uncertainty-oblivious approach that compares instances of uncertain objectives with respect to their mean values. The specification incorporates 15 resources, 66 tasks, and 275 mappings. The proposed operator uses mean and 95% quantile interval as the average and spread criteria, respectively. Depicted are the mean values,

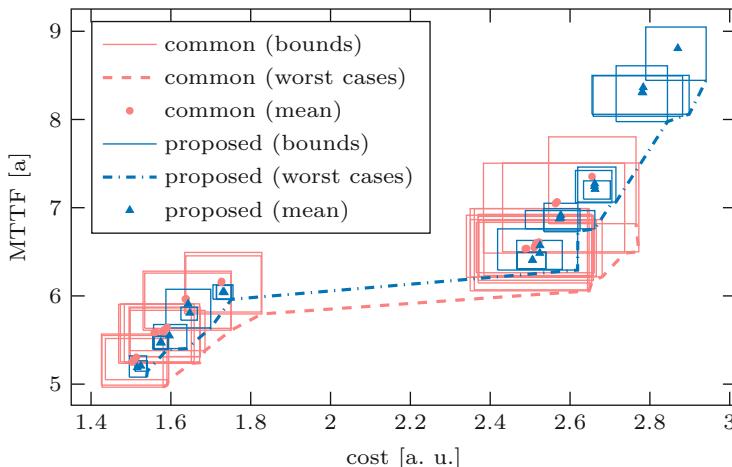


Fig. 11 Pareto fronts when optimizing MTTF! and cost of an H.264 encoder/decoder using a common uncertainty-oblivious approach vs. the proposed comparison operator

boxes enclosing the uncertainty distributions, and lines connecting the worst cases. The results show that the proposed comparison operator enables the **DSE!** to find implementation candidates of smaller uncertainty, and yet comparable quality in the average case.

5 Conclusion

Progressive shrinkage in electronic devices has brought them vulnerabilities to manufacturing tolerances as well as environmental and operational changes. The induced uncertainty in component reliability might propagate to system level, which necessitates uncertainty-aware cross-level reliability analysis. This chapter presents a cross-level reliability analysis methodology that enables handling the ever increasing analysis complexity of embedded systems under the impact of different uncertainties. It combines various reliability analysis techniques across different abstraction levels by introducing mechanisms for (a) the composition and decomposition of the system during analysis and (b) converting analysis data over abstraction levels through adapters. It also provides an explicit modeling of uncertainties and their correlations. The proposed methodology is incorporated in an automatic reliability analysis tool that enables the evaluation of reliability-increasing techniques within a **DSE!** framework. The **DSE!** employs meta-heuristic optimization algorithms and is capable of comparing system implementation candidates with objectives regarded as probability distributions.

Acknowledgments This work is supported in part by the German Research Foundation (DFG) as associated project CRAU (GL 819/1-2 & TE 163/16) of the priority program Dependable Embedded Systems (SPP 1500).

References

1. Al Faruque, M., Jahn, J., Ebi, T., Henkel, J.: Runtime thermal management using software agents for multi-and many-core architectures. *IEEE Design Test Comput.* **27**(6), 58–68 (2010)
2. Aliee, H.: Reliability analysis and optimization of embedded systems using stochastic logic and importance measures. Doctoral Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) (2017)
3. Aliee, H., Glaß, M., Reimann, F., Teich, J.: Automatic success tree-based reliability analysis for the consideration of transient and permanent faults. In: *Design, Automation & Test in Europe (DATE)*, pp. 1621–1626 (2013)
4. Aliee, H., Glaß, M., Khosravi, F., Teich, J.: An efficient technique for computing importance measures in automatic design of dependable embedded systems. In: *Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 34:1–34:10 (2014)
5. Aliee, H., Borgonovo, E., Glaß, M., Teich, J.: Importance measures in time-dependent reliability analysis and system design. In: *European Safety and Reliability Conference (ESREL)* (2015)

6. Aliee, H., Borgonovo, E., Glaß, M., Teich, J.: On the Boolean extension of the Birnbaum importance to non-coherent systems. *Reliab. Eng. Syst. Safe.* **160**, 191–200 (2016)
7. Aliee, H., Banaiyianmofrad, A., Glaß, M., Teich, J., Dutt, N.: Redundancy-aware design space exploration for memory reliability in many-cores. In: *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMW)* (2017)
8. Aliee, H., Khosravi, F., Teich, J.: Efficient treatment of uncertainty in system reliability analysis using importance measures. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2019)
9. Bensalem, S., Bozga, M., Sifakis, J., Nguyen, T.: Compositional verification for component-based systems and application. In: *Automated Technology for Verification and Analysis (ATVA)*, pp. 64–79 (2008)
10. Bickle, T., Teich, J., Thiele, L.: System-level synthesis using evolutionary algorithms. *Des. Autom. Embed. Syst.* **3**(1), 23–58 (1998)
11. Borgonovo, E., Aliee, H., Glaß, M., Teich, J.: A new time-independent reliability importance measure. *Eur. J. Oper. Res.* **254**(2), 427–442 (2016)
12. Bowen, J., Stavridou, V.: Safety-critical systems, formal methods and standards. *Softw. Eng. J.* **8**, 189–189 (1993)
13. Coit, D.W., Smith, A.E.: Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Trans. Reliab.* **45**(1), 254–260 (1996)
14. Council, J.E.D.E.: Failure Mechanisms and Models for Semiconductor Devices. JEDEC Publication JEP122-B (2003)
15. Deb, K., Gupta, H.: Searching for robust Pareto-optimal solutions in multi-objective optimization. In: *Evolutionary Multi-Criterion Optimization (EMO)*, pp. 150–164 (2005)
16. Eles, P., Izosimov, V., Pop, P., Peng, Z.: Synthesis of fault-tolerant embedded systems. In: *Design, Automation & Test in Europe (DATE)*, pp. 1117–1122 (2008)
17. Farahani, B., Safari, S.: A cross-layer approach to online adaptive reliability prediction of transient faults. In: *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pp. 215–220 (2015)
18. Gebregiorgis, A., Kiamehr, S., Oboril, F., Bishnoi, R., Tahoori, M.B.: A cross-layer analysis of soft error, aging and process variation in near threshold computing. In: *Design, Automation & Test in Europe (DATE)*, pp. 205–210 (2016)
19. Glaß, M., Lukasiewycz, M., Haubelt, C., Teich, J.: Towards scalable system-level reliability analysis. In: *Design Automation Conference (DAC)*, pp. 234–239 (2010)
20. Glaß, M., Lukasiewycz, M., Reimann, F., Haubelt, C., Teich, J.: Symbolic system level reliability analysis. In: *International Conference on Computer-Aided Design (ICCAD)*, pp. 185–189 (2010)
21. Glaß, M., Yu, H., Reimann, F., Teich, J.: Cross-level compositional reliability analysis for embedded systems. In: *International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, pp. 111–124 (2012)
22. Glaß, M., Teich, J., Lukasiewycz, M., Reimann, F.: Hybrid Optimization Techniques for System-Level Design Space Exploration, pp. 1–31. Springer, Dordrecht (2017)
23. Gu, Z., Zhu, C., Shang, L., Dick, R.: Application-specific MPSoC reliability optimization. *IEEE Trans. Very Large Scale Integr. Syst.* **16**(5), 603 (2008)
24. Herkersdorf, A., Aliee, H., Engel, M., Glaß, M., Gimmerl-Dumont, C., Henkel, J., Kleeberger, V., Kochte, M., Kühn, J., Müller-Gritschneider, D., Nassif, S., Rauchfuss, H., Rosenstiel, W., Schlichtmann, U., Shafique, M., Tahoori, M., Teich, J., Wehn, N., Weis, C., Wunderlich, H.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectr. Reliab.* **54**(6–7), 1066–1074 (2014)
25. Hooman, J.: Specification and Compositional Verification of Real-Time Systems. Springer, Berlin (1991)
26. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. *IEEE Trans. Evol. Comput.* **9**(3), 303–317 (2005)
27. Jung, S., Lee, J., Kim, J.: Variability-aware, discrete optimization for analog circuits. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **33**(8), 1117–1130 (2014)

28. Khosravi, F., Reimann, F., Glaß, M., Teich, J.: Multi-objective local-search optimization using reliability importance measuring. In: Design Automation Conference (DAC), pp. 1–6 (2014)
29. Khosravi, F., Müller, M., Glaß, M., Teich, J.: Uncertainty-aware reliability analysis and optimization. In: Design, Automation & Test in Europe (DATE), pp. 97–102 (2015)
30. Khosravi, F., Borst, M., Teich, J.: Probabilistic dominance in robust multi-objective optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1597–1604 (2018)
31. Khosravi, F., Müller, M., Glaß, M., Teich, J.: Simulation-based uncertainty correlation modeling in reliability analysis. The Institution of Mechanical Engineers, Part O J. Risk Reliab. **232**, 725–737 (2018)
32. Liang, S., Zhang, Z., Pei, T., Li, R., Li, Y., Peng, L.: Reliability tests and improvements for Sc-contacted n-type carbon nanotube transistors. Nano Res. **6**(7), 535–545 (2013)
33. Limbourg, P.: Multi-objective optimization of problems with epistemic uncertainty. In: Evolutionary Multi-Criterion Optimization (EMO), pp. 413–427 (2005)
34. Meyer, B., Hartman, A., Thomas, D.: Cost-effective slack allocation for lifetime improvement in NoC-based MPSoCs. In: Design, Automation & Test in Europe (DATE), pp. 1596–1601 (2010)
35. Misra, K.B.: Reliability Analysis and Prediction: A Methodology Oriented Treatment, vol. 15. Elsevier, Amsterdam (2012)
36. Narayanan, V., Xie, Y.: Reliability concerns in embedded system designs. Computer **39**, 118–120 (2006)
37. Pham, T.T., Defago, X., Huynh, Q.T.: Reliability prediction for component-based software systems: dealing with concurrent and propagating errors. Sci. Comput. Program. **97**, 426–457 (2015)
38. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms—a comprehensive survey. Swarm Evol. Comput. **33**, 18–45 (2017)
39. Salazar, A.D., Rocco, S.C.: Solving advanced multi-objective robust designs by means of multiple objective evolutionary algorithms (MOEA): a reliability application. Reliab. Eng. Syst. Safe. **92**(6), 697–706 (2007)
40. Sander, B., Schnerr, J., Bringmann, O.: ESL power analysis of embedded processors for temperature and reliability estimations. In: Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 239–248 (2009)
41. Skadron, K., Stan, M.R., Huang, W., Velusamy, S., Sankaranarayanan, K., Tarjan, D.: Temperature-aware microarchitecture. In: 30th Annual International Symposium on Computer Architecture (ISCA), pp. 2–13 (2003)
42. Stathis, J.: Reliability limits for the gate insulator in CMOS technology. IBM J. Res. Dev. **46**(2–3), 265–286 (2002)
43. Streichert, T., Glaß, M., Haubelt, C., Teich, J.: Design space exploration of reliable networked embedded systems. J. Syst. Architect. **53**(10), 751–763 (2007)
44. Teich, J.: Pareto-front exploration with uncertain objectives. In: Evolutionary Multi-Criterion Optimization (EMO), pp. 314–328 (2001)
45. Wandeler, E., Thiele, L.: Real-Time Calculus (RTC) Toolbox (2006). <http://www.mpa.ethz.ch/Rtctoolbox>
46. Xiang, Y., Chantem, T., Dick, R.P., Hu, X.S., Shang, L.: System-level reliability modeling for MPSoCs. In: Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 297–306 (2010)
47. Xie, Y., Li, L., Kandemir, M., Vijaykrishnan, N., Irwin, M.: Reliability-aware co-synthesis for embedded systems. J. VLSI Signal Proce. **49**(1), 87–99 (2007)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Robust Computing for Machine Learning-Based Systems



Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Mohammad Taghi Teimoori, Florian Kriebel, Jeff (Jun) Zhang, Kang Liu, Semeen Rehman, Theocharis Theocharides, Alessandro Artusi, Siddharth Garg, and Muhammad Shafique

1 Introduction

Machine learning (ML) has emerged as the principal tool for performing complex tasks which are impractical (if not impossible) to code by humans. ML techniques provide machines the capability to learn from experience and thereby learn to perform complex tasks without much (if any) human intervention. Over the past decades, many ML algorithms have been proposed. However, Deep Learning (DL), using Deep Neural Networks (DNNs), has shown state-of-the-art accuracy, even surpassing human-level accuracy in some cases, for many applications [31]. These applications include, but are not limited to, object detection and localization, speech recognition, language translation, and video processing [31].

The state-of-the-art performance of the DL-based methods has also led to the use of DNNs in complex safety-critical applications, for example, autonomous driving [11] and smart healthcare [10]. DNNs are intrinsically computationally

M. A. Hanif (✉) · F. Khalid · R. V. W. Putra · M. T. Teimoori · F. Kriebel · S. Rehman

M. Shafique

Technische Universität Wien (TU Wien), Vienna, Austria

e-mail: muhammad.hanif@tuwien.ac.at; faiq.khalid@tuwien.ac.at; rachmad.putra@tuwien.ac.at; florian.kriebel@tuwien.ac.at; seemeen.rehman@tuwien.ac.at; muhammad.shafique@tuwien.ac.at

J. Zhang · K. Liu · S. Garg

New York University, New York, NY, USA

e-mail: jeffjunzhang@nyu.edu; kang.liu@nyu.edu; sg175@nyu.edu

T. Theocharides

University of Cyprus, Nicosia, Cyprus

e-mail: ttheocharides@ucy.ac.cy

A. Artusi

University of Cyprus, Nicosia, Cyprus

MRG DeepCamera RISE, Nicosia, Cyprus

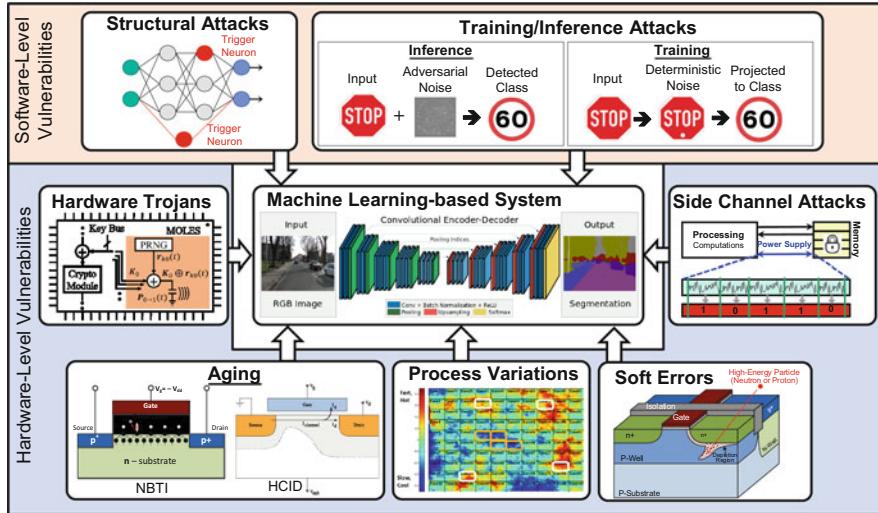
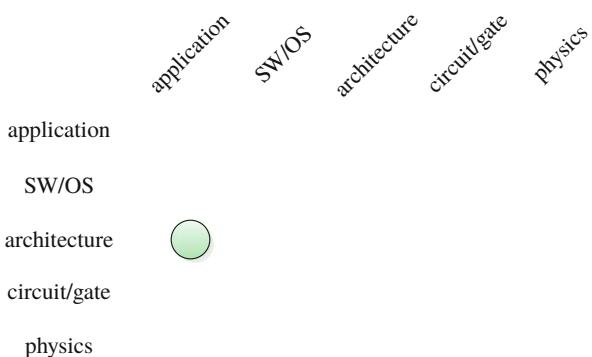


Fig. 1 Overview of different reliability and security vulnerabilities to machine learning-based systems. (Picture sources: [47, 49])

Fig. 2 Main abstraction layers of embedded systems and this chapter's major (green, solid) contributions



intensive and also require high memory resources [53]. Current research mainly focuses on the development of less computationally intensive and resource-efficient DNNs that can offer high accuracy, and energy and performance efficient DNN accelerators for ML-based applications [1, 18, 23, 29, 34, 36, 37, 44, 53]. However, when considered for safety-critical applications, the robustness of these DNN-based systems to different reliability and security vulnerabilities also becomes one of the foremost objectives. An overview of different types of vulnerabilities in ML-based systems is shown in Fig. 1, which are discussed from the architectural- and application-layer perspective in this chapter. Figure 2 shows the abstraction layers in the context of the SPP 1500 covered in this chapter.

Reliability Threats: In hardware design, reliability is the ability of the hardware to perform as intended for a specified duration, i.e., the lifetime of the hardware. There are a number of hardware related vulnerabilities that can disrupt the functionality of a digital system in its lifetime.

1. **Soft Errors** are transient faults caused by high energy particle strikes. These faults surface at hardware-layer as bit-flips and can propagate to the application layer resulting in incorrect output.
2. **Aging** is the gradual degradation of the hardware due to different physical phenomena like Hot carrier Injection (HCI), Negative-Bias Temperature Instability (NBTI), and Electromigration (EM). It leads to timing errors and eventually can also lead to permanent faults [56].
3. **Process variations** are the imperfections caused by the variations in the fabrication process of the chips. This can lead to variations in the timing and leakage power characteristics within a chip as well as across different chips [45].

Apart from the above-listed vulnerabilities, environmental conditions can also affect the reliability of a system. Such factors include temperature, altitude, high electric fields, etc.

A number of techniques have been proposed for improving the resilience of the systems against the reliability threats. However, most of these mitigation techniques are based on redundancy, for example, DMR: dual modular redundancy [58] and TMR: triple modular redundancy [35]. The redundancy based approaches, although considered to be very effective for other application domains [19], are highly inefficient for DNN-based systems because of the compute intensive nature of the DNNs [48], and may incur significant area, power/energy, and performance overheads. *Hence, a completely new set of resource-efficient reliability mechanisms is required for robust machine learning systems.* A list of techniques proposed for improving the reliability of DNN-based systems, which are later discussed in the following sections of the chapter, are mentioned in Fig. 3.

Security Threats: In system design, security is defined as the property of a system to ensure the confidentiality, integrity, and availability of the hardware and the data while performing the assigned tasks. There are several security vulnerabilities that can be exploited to perform security attacks.

1. **Data Manipulation:** The input data or data during inter-/intra-module communication in a system can be manipulated to perform several security attacks. For example, in DNNs, the training dataset and the inference data can be manipulated to perform misclassification or confidence reduction attacks [17, 24, 26, 27, 43, 51].
2. **Denial-of-Service:** A tiny piece of code/hardware or flooding the communication channels can be used to trigger the malfunctioning or failure of the system. For example, in DNNs, adding an extra neuron/set of neurons [17] or introducing the kill switch in DNN-based hardware can lead to system failure or malfunctioning, i.e., misclassification.

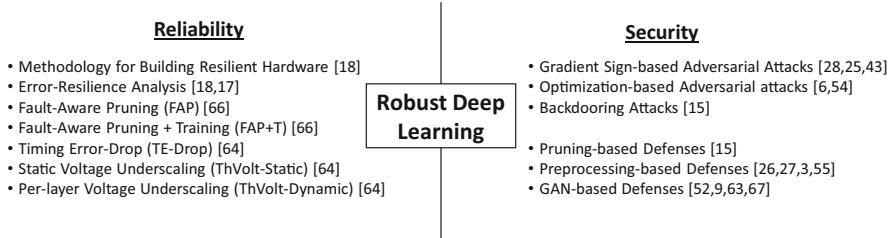


Fig. 3 Overview of the works discussed in this chapter for addressing reliability and security vulnerabilities of deep learning-based systems

3. **Data/IP Stealing:** The side-channel information (in hardware, power, timing, and loopholes or behavior leaking properties of the algorithms) can be exploited to steal the confidential information. For example, in DNNs, the gradient information can be used to steal trained model [50, 57, 60].

Several countermeasures have been developed to address these threats, but most of these defenses are either based on obfuscation or run-time monitoring [3, 22]. These techniques are very effective for traditional systems, however, DNN-based systems require different approaches because of their unique security vulnerabilities, i.e., training/inference data manipulation. Some of the techniques proposed for addressing the security of DNN-based systems are listed in Fig. 3 and are later discussed in the chapter.

In the following sections, we discuss:

1. A brief overview of DNNs and the hardware accelerators used for efficiently processing these networks.
2. In Sect. 3, we present our methodology for building reliable systems and discuss techniques for mitigating permanent and timing errors.
3. The security vulnerabilities in different types of DNNs are discussed in Sect. 4.
4. Open challenges and further research opportunities for building robust systems for ML-based safety-critical applications

2 Preliminaries

2.1 Deep Neural Networks

A neural network can be described as a network of interconnected neurons. *Neurons* are the fundamental computational units in a neural network where each neuron performs a weighted sum of inputs (dot-product operation), using the inputs and the weights associated with each input connection of the neuron. Each output is then (optionally) passed through an *activation function* which introduces non-linearity and thereby allows the network to learn complex classification boundaries.

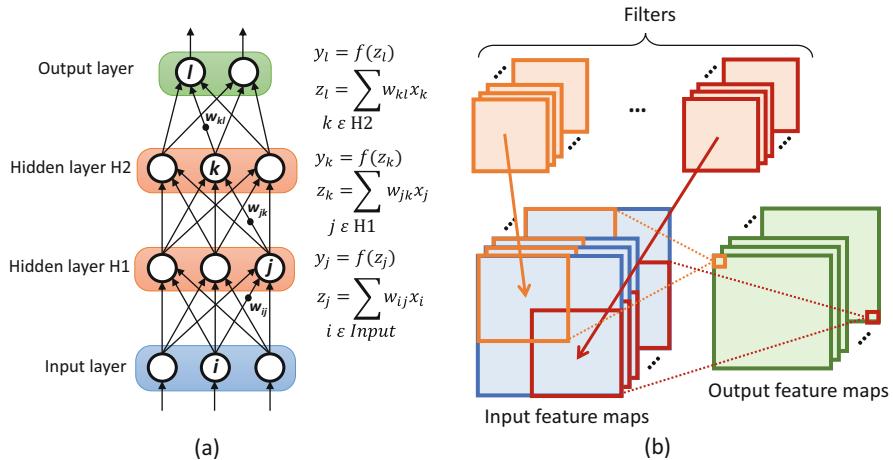


Fig. 4 Illustration of (a) a multi-layer perceptron and (b) a convolutional layer

In neural networks, neurons are arranged in the form of *layers*. There are several types of NNs, for instance, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Multi-Layer Perceptrons (MLPs) [31]. Although the techniques discussed in the following sections are not limited to a specific type of NNs, in this chapter, we mainly focus on feed-forward neural networks (i.e., CNNs and MLPs) because of their widespread use in many artificial intelligence applications.

An MLP is a type of NN that is composed of multiple fully-connected layers. In a fully-connected layer, each neuron is connected to all the neurons in the neighboring layers. An example illustration of a three layer MLP is shown in Fig. 4a.

A CNN is a type of NN that is composed of several convolutional layers and the fully-connected layers. An example illustration of a *convolutional layer* is shown in Fig. 4b. The layer is composed of multiple filters which are convolved with the input feature maps to generate the output feature maps. The depth of the filters and the input feature maps is the same. Each filter results in one *output feature map* and, therefore, the number of output feature maps is equal to the number of filters in a convolutional layer. These input and output feature maps are also referred to as *activation maps*. A detailed description of CNNs can be found in [53].

2.2 Hardware Accelerators for Deep Neural Networks

To enable the use of DNNs in energy-/power-constraint scenarios as well as in high performance applications, several different hardware architectures for DNN acceleration have been proposed. While all the accelerators provide some unique

features and support some specific dataflows in a more efficient manner, *systolic array-based designs are considered among the promising ones* [18, 23, 37, 61].

A systolic array is a homogeneous network of processing elements (PEs), which are tightly coupled together. Each PE in the network receives data from its nearest neighbors, performs some function, and passes on the result and data to the neighboring PE/s. The systolic array-based architectures alleviate the memory bottleneck issue by *locally reusing the data*, without the need of expensive memory read and write operations. Moreover, the systolic arrays are intrinsically efficient at performing matrix multiplications, which is the core operation of neural networks. Therefore, many accelerators use these arrays at their core for accelerating the neural networks [18, 23, 37, 61]. The *Tensor Processing Unit (TPU)*, a DNN accelerator that is currently in use in the datacenters of Google, is a systolic array-based architecture that uses an array of 256×256 multiply-and-accumulate (MAC) units. The TPU provides $15 \times - 30 \times$ faster execution, and $30 \times - 80 \times$ more efficient (in terms of performance/Watt) performance than the K80 GPU and the Haswell CPU [23].

Figure 5 illustrates a design overview of an exemplar DNN accelerator which is based on the TPU architecture. The design is used as the basic architecture in the following section. The architecture is composed of a systolic array of MAC units, similar to that in the TPU. Prior to the computations, the weights are pre-loaded in the PEs from the weight memory in a manner that the weights from the same filter/neuron are loaded in the same column of the array. During processing, the

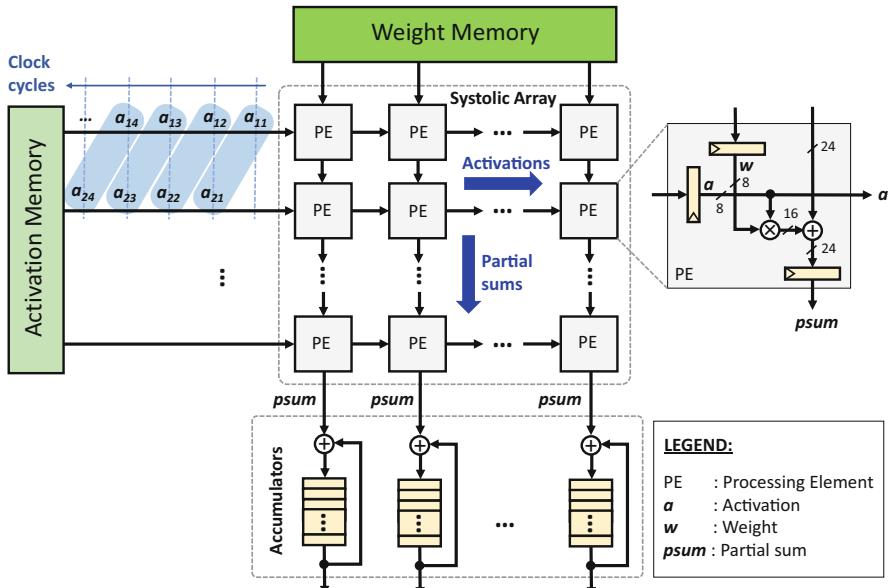


Fig. 5 A systolic array-based DNN accelerator architecture (adapted from [65])

weights are held *stationary* inside the PEs and the activations are streamed in from the activation memory. At each clock cycle, the activations are passed-on from left to right while the partial sums are moved downstream. The activations across rows are aligned such that the activations corresponding to a particular output reaches a particular PE at the same instance when its partial sum reaches that PE. In case, the size of a filter/neuron is larger than the number of rows in the array, each output computation related to the filter/neuron is divided into multiple portions and the accumulators at the bottom are used for temporarily holding the partial sums while rest of the corresponding partial sums are computed by the array. A more detailed explanation of the architecture can be found in [65].

3 Reliable Deep Learning

In this section, we present our methodology for building reliable hardware for DNN-based applications. We also highlight a few case studies, targeting different types of reliability threats, for building reliable yet efficient hardware for DNN-based applications.

3.1 Our Methodology for Designing Reliable DNN Systems

Figure 6 presents our design flow for developing reliable hardware for DNN-based applications [17]. The methodology is composed of two parts: (1) Design-time steps; and (2) Run-time steps.

The **design-time** steps focus on proposing a hardware architecture which is capable of mitigating different types of reliability faults that arise due to process variations and aging, as well as aggressive voltage scaling (i.e., permanent faults

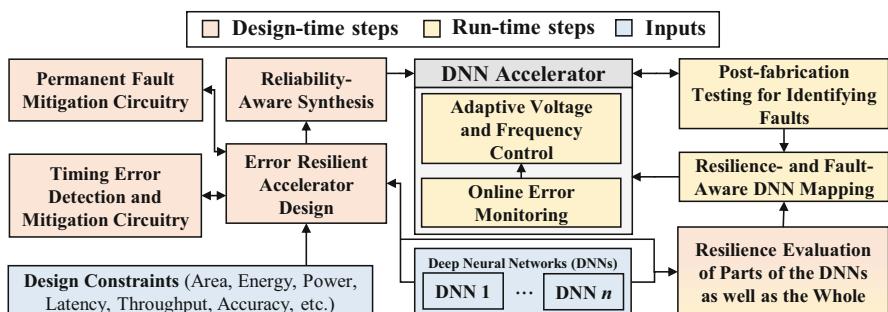


Fig. 6 Our methodology for designing reliable hardware for DNN-based applications (adapted from [17])

and timing errors). Provided a set of design constraints, representative DNN models, and resilience of the DNNs to different types of reliability threats and errors, a baseline hardware architecture is designed. We then reinforce it with different architectural enhancements for mitigating permanent faults (see Sect. 3.3) and handling timing errors (see Sect. 3.4). *The architectural enhancements are performed in a manner that they do not significantly affect the resource efficiency of the baseline architecture.* Once the architecture is finalized, the hardware is synthesized using reliability-aware synthesis techniques, for example, by using standard cells to selectively harden vulnerable nodes in the hardware [33], to harden the more vulnerable parts of the hardware design.

The **run-time** steps focus on proposing *mapping policies* for mapping DNN computations to the synthesized hardware. The mapping policies are decided based on the fault maps generated using post-fabrication and testing, and the error resilience of the DNNs. *Techniques like error injection can be used for the resilience analysis* [16, 46]. *Fault-aware training of DNNs* can also be used for designing/modifying network architecture/parameters (see Sect. 3.3). Moreover, adaptive voltage scaling can be employed for trading off reliability with energy efficiency based on the error resilience of the DNNs. If required, software-level redundancy can also be employed to further improve the reliability by performing the computations related to critical neurons/filters multiple times.

3.2 Resilience of DNNs to Reliability Threats

Neural Networks are assumed to be inherently error resilient [12]. However, different types of errors can have different impact on the output of a DNN. This section presents the accuracy analysis of DNNs in the presence of different types of reliability faults.

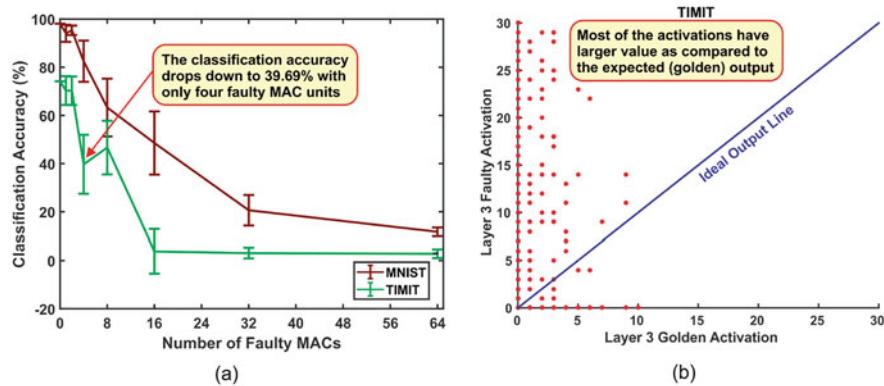
3.2.1 Resilience of DNNs to Permanent Faults

This section highlights the resilience of DNNs to permanent faults by empirically analyzing the effects of *stuck-at* permanent faults in the TPU-based accelerator (presented in Fig. 5) on the classification accuracy of different DNNs. The datasets (i.e., *MNIST* and *TIMIT*) and the corresponding network architectures used for this analysis are listed in Table 1. To study the resilience, the TPU with a systolic array of 256×256 MAC units is synthesized using 45 nm OSU PDK to generate a gate-level netlist and then stuck-at faults are inserted at internal nodes in the netlist. For this analysis, faults only in the data-path were considered as the faults in the memory components can be mitigated using Error Correction Codes (ECC) and faults in control-path can lead to undesirable results.

Figure 7a shows the impact of using a faulty TPU for two different classification tasks, i.e., *image classification using the MNIST dataset* and *speech recognition*

Table 1 Datasets and the corresponding 8-bit DNNs used for evaluation (adapted from [65])

Dataset	Network architecture	Accuracy(%)
MNIST [30]	Fully-connected (L1–L4): $784 \times 256 \times 256 \times 256 \times 10$	98.15
TIMIT [4]	Fully-connected (L1–L4): $1845 \times 2000 \times 2000 \times 2000 \times 183$	73.91
ImageNet [7]	Convolutional (L1–L2): $(224, 224, 3) \times (27, 27, 64) \times (13, 13, 192)$ Convolutional (L3–L5): $(13, 13, 384) \times (13, 13, 256) \times (6, 6, 256)$ Fully-connected (L6–L8): $4096 \times 4096 \times 1000$	76.33 (Top-5)

**Fig. 7** Impact of stuck-at-faults in the baseline TPU-based architecture on DNN applications. (a) Classification accuracy drop due to stuck-at-fault MACs. (b) Impact of TPU stuck-at-faults on DNN applications (adapted from [66])

using the TIMIT dataset. It can be seen in the figure that the classification accuracy of both the tasks decreases significantly with the increase in the number of faulty PEs in the hardware. For example, the classification accuracy for the TIMIT dataset drops from 74.13 to 39.69% when only four (out of 256×256) MAC units are faulty and is almost 0% when the number of faulty MACs increases to 16 or more.

The reason for the significant drop in accuracy can be understood by comparing the golden (fault-free) output of the neurons of a particular layer with the outputs computed by the faulty TPU. Figure 7b shows that the computed output of the final layer of the network used for the TIMIT dataset in most of the cases has higher activation value as compared to the expected. This is mainly because of the fact that *stuck-at faults, in some of the cases, affect the higher order bits of the MACs output*. This highlights the need for permanent fault mitigation in the hardware to increase the yield as hardware with permanent faults cannot be used for ML-based applications, specifically for the safety-critical applications.

3.2.2 Resilience of DNNs to Timing Faults

Timing failures in high performance nanometer technology-based digital circuits are a major reliability concern and are caused by various mechanisms, e.g., power supply disturbance, crosstalk, process variations, as well as aging. Moreover, the operating conditions, which play a vital role in defining the performance and energy efficiency of the hardware, also have a significant impact on the frequency of the timing errors. Although it is assumed that the critical paths, which are more vulnerable to timing errors, are rarely exercised, the timing errors can significantly affect the functionality of an application. Here, we highlight this for DNN-based applications by analyzing the energy-quality trade-off achieved using voltage underscaling. We show the analysis for two widely accepted types of timing error mitigation techniques: (1) *timing error detection and recovery* (TED) [9]; and (2) *timing error propagation* (TEP) [41, 62]. The TED makes use of additional components (e.g., using Razor flip-flops [9]) for detecting timing errors, and recovers by reliably re-executing the function in case of errors. On the other hand, TEP allows errors to propagate through to the application layer in the hope that the application is error resilient.

For this analysis, the TPU-based hardware architecture discussed in Sect. 2.2 is considered. The architecture is assumed to be composed of a 256×256 MAC array. The terms *Local Timing Error* and *Global Timing Error* are used to characterize the resilience. The local timing error is used to denote the error in a single MAC unit. The global timing error defines the error in the complete systolic array. Figure 8b shows the impact on the classification accuracy for the *MNIST* dataset with voltage underscaling when the timing errors are allowed to propagate through to the application layer. It can be seen from the figure that as soon as the timing errors start occurring, i.e., below the voltage underscaling ratio of $r = 0.9$ (as shown in Fig. 8b), the classification accuracy of the DNN for TEP drops sharply.

As mentioned above, the TED-based approaches work on the principle of error detection and recovery. The recovery phase in TED defines its limitation for huge systolic array-based systems as, for synchronization of the data flow, the complete systolic array has to be stalled to recover the error in a single PE. This limitation of the TED-based approach can be highlighted using Fig. 8a which shows the impact of voltage underscaling on the overall energy consumption of the TPU-based hardware architecture for generating accurate outputs. It can be noted from the figure that *the overall energy consumption for a recovery based technique starts increasing as soon as errors start appearing*, which is the case for even the most naive type of error recovery mechanism, i.e., single cycle recovery.

3.2.3 Resilience of DNNs to Memory Faults

To illustrate the importance of memory faults, we presented an analysis in [17] where we injected random faults at bit-level in the weight memory (i.e., the memory storing the network parameters) and studied the impact of those faults on the

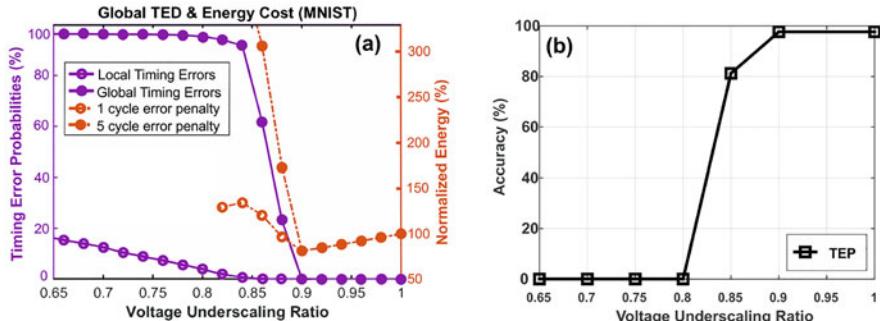


Fig. 8 (a) Timing error probabilities versus voltage underscaling ratio, and the corresponding energy cost for global TED. (b) DNN accuracy on the MNIST versus voltage underscaling for TEP. (Adapted from [65])

accuracy of a DNN. The analysis concluded that, for the higher significance bits of the weights, the accuracy of the DNNs drop sharply with the increase in error rate. We also studied the impact of different types of bit-flips, i.e., from 0 to 1 bit-flips and from 1 to 0 bit-flips, and found that the 0 to 1 bit-flips result in erroneous output while the 1 to 0 bit-flips do not impact the accuracy much. This is inline with the concept of dropout [20] and dropconnect [59] in the sense that in case of 1 to 0 bit-flips the erroneous output is leaned towards 0 value, whereas in case of 0 to 1 bit-flips the error can increase significantly if the bit-flip occurs in any of the higher significance bits. This analysis was performed on the AlexNet network using the ImageNet dataset. Similar, fault injection methods, e.g., [16] and [46], can also be used for analyzing the resilience of DNNs, as a whole as well as of individual layers/neurons of the networks.

3.3 Permanent Fault Mitigation

To mitigate permanent faults in the computing units of the hardware, two different methods have been proposed: (1) Fault-Aware Pruning (FAP); and (2) Fault-Aware Pruning + Training (FAP+T).

The **Fault-Aware Pruning (FAP)** works on the principle of pruning the weights (i.e., setting them to zero) that have to be mapped on faulty MAC units. *The principle is inline with the concepts of dropout [20] and dropconnect [59] which are commonly used for regularization and avoiding over-fitting.* For this work, the TPU architecture shown in Fig. 5 with static mapping policy is assumed. The static mapping policy means that each weight is mapped to a specific PE while multiple weights can be mapped to the same PE at different time instances. Moreover, it is also assumed that post-fabrication tests are performed on each TPU chip to extract the fault map which indicates the faulty PEs.

Fig. 9 Systolic array-based architecture for permanent fault mitigation (adapted from [66])

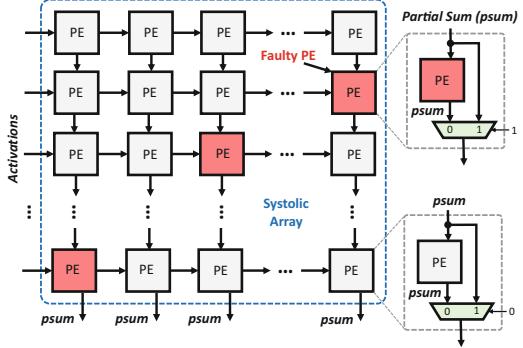


Figure 9 shows an implementation that can be used to realize the concept where a bypass path is provided for each MAC unit [66]. The bypass path enables to skip the contribution of a specific partial sum in case the specific PE is faulty, which is equivalent to setting the weight to zero. The area overhead of the modified design is only around 9% [66].

The **Fault-Aware Pruning + Training (FAP+T)** technique starts with the FAP approach, however, it additionally retrains the unpruned weights while forcing the pruned weights to zero to optimize the network parameters. One drawback of this approach is that the fault map of each chip can be different which means that a network has to be retained for each chip based on its own fault map.

Figure 10 shows the impact on the classification accuracy versus the percentage of faulty MAC units for three different classification problems mentioned in Table 1. The results show that both the techniques show significant resilience to the permanent faults. Moreover, the FAP+T technique outperforms FAP because of the involved optimization of the network parameters and allows the DNN-based system to run with negligible accuracy loss even when 50% of its MAC units are faulty. However, in cases where FAP+T is impractical FAP can also provide reasonable accuracy, specifically in cases where the number of faulty units is less.

3.4 Timing Fault Mitigation

As mentioned in Sect. 3.2.2, the conventional TED approaches have significant overheads when used for DNN accelerators. Here, we discuss the new architectural innovations proposed in Thundervolt [65] for mitigating timing errors in DNN accelerators in a performance efficient manner.

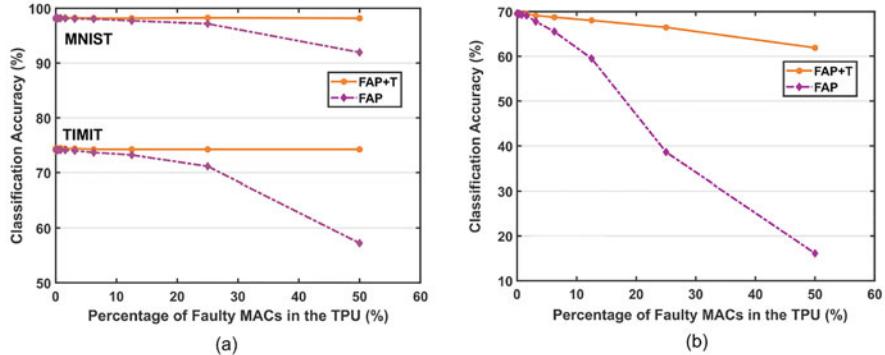
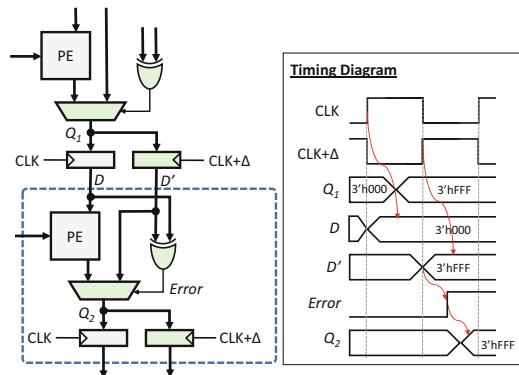


Fig. 10 Classification accuracy versus percentage of faulty MACs using FAP and FAP+T for the networks used corresponding to (a) MNIST and TIMIT; and (b) ImageNet datasets (adapted from [66])

Fig. 11 A block-level diagram illustrating the architectural modifications for TE-Drop and the impact of timing errors on the computation of a neuron (adapted from [65])



3.4.1 TE-Drop

Thundervolt [65] proposed a novel technique to deal with timing errors in a systolic array-based DNN accelerator, i.e., TE-Drop. *TE-Drop utilizes the Razor flip-flops to detect timing errors, however, it does not re-execute erroneous MAC operations.* Similar to the FAP techniques, TE-Drop also works on the principle that the contribution of each individual MAC output to the output of a neuron in DNNs is small. Hence, a few MAC operations can be ignored without significantly affecting the overall accuracy of the network. In case of a timing error, TE-Drop allows the MAC unit to sample the correctly computed output to an alternate register operating on a delayed clock. The succeeding PE is then bypassed and the correctly computed output is provided instead. The architectural modifications required to realize the concept are shown in Fig. 11.

Figure 11 illustrates the functionality of the TE-Drop with the help of a timing diagram. Here, it is assumed that the shadow clock is delayed by 50% of the clock

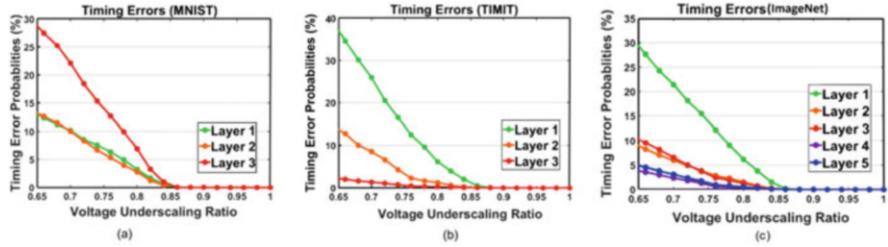


Fig. 12 Timing error probabilities for each layer of the networks used corresponding to (a) MNIST, (b) TIMIT, and (c) ImageNet datasets (adapted from [65])

period. It is assumed that the clock frequency is defined such that the error signal and correct partial sum from the erroneous MAC become available after this much duration. Note that the error signal is obtained by OR-ing the bitwise XOR of all the individual Razor flip-flop at the output of the MAC unit.

3.4.2 Per-Layer Voltage Underscaling

In most of the accelerators, it is assumed that the layers of a DNN are executed in a serial fashion (i.e., one after the other), where processing of each layer can take thousands of clock cycles, depending on the size of the layer. Figure 12 shows the timing error rate versus voltage underscaling ratio plots for each individual layer of three DNN architectures mentioned in Table 1. It can be seen from the figures that the error rate varies significantly across layers. Based on this observation, a per-layer voltage underscaling scheme was proposed in Thundervolt [65] that distributes the total timing error budget equally among the layers of a network to ensure that the more sensitive layers should not consume a significant part of the budget and limits the achievable efficiency gains.

Figure 13 compares two versions of Thundervolt:

1. **ThVolt-Static** where each voltage underscaling ratio is kept the same throughout a DNN execution.
2. **ThVolt-Dynamic** that utilizes per-layer voltage underscaling based on the sensitivity of each layer.

For the baseline, the results of the TEP scheme are also shown. The plot for ThVolt-Static is obtained by sweeping voltage underscaling ratios, and that of ThVolt-Dynamic is obtained by sweeping the total timing error budget. The figures show that for each case Thundervolt outperforms TEP scheme, and for complex tasks (e.g., image classification on the ImageNet dataset) the **ThVolt-Dynamic** outperforms the **ThVolt-Static** approach.

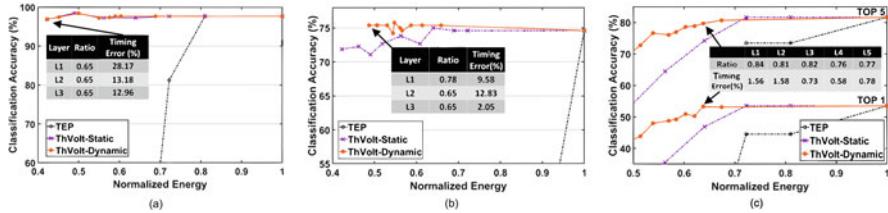


Fig. 13 Accuracy versus energy trade-off using Thundervolt [65] on validation data. (a) MNIST. (b) TIMIT (c) ImageNet (adapted from [65])

4 Secure Deep Learning

In this section, we present different security attacks on DNNs and potential countermeasures.

4.1 Security Attacks on DNNs

Several security attacks have been proposed by exploiting the security vulnerabilities, especially data dependency and unpredicted behavior of intermediate layers of DNN-algorithms during training as well as inference. However, *adversarial and backdooring attacks are some of the most effective and popular attacks for DNNs*. Therefore, in the following subsections, we analyze the state-of-the-art adversarial attacks and proposed backdoor attacks.

4.1.1 Adversarial Perturbation Attacks

It can be defined as the crafted imperceptible noise to perform *targeted* or *untargeted misclassification* in a DNN-based system. In these attacks, an attacker's objective can be summarized as follows: given an image x with a classification label $y = \text{classifier}(x)$, where classifier is the function of the neural network. The attacker aims to find an image x' whose classification label is y' , such that $y' = \text{classifier}(x') \neq y$, and $\|x' - x\| \leq \delta$, where δ is an upper bound of the distortion from x to x' . For example, some input adversarial attacks are shown in Fig. 14.

Several attacks have proposed to exploit the adversarial vulnerabilities in DNN-based systems. However, based on the attack methodology, these attacks can broadly be categorized into Gradient Sign Methods and Optimization-based approaches.

1. **Gradient Sign Methods:** These attacks exploit the derivatives and backpropagation algorithm to generate the attack images with imperceptible crafted noise. The main goal of these attacks is to minimize the prediction probability of the true label so as to mislead the network to output a different label (can be targeted or untargeted) other than the ground truth. Some of the most commonly proposed

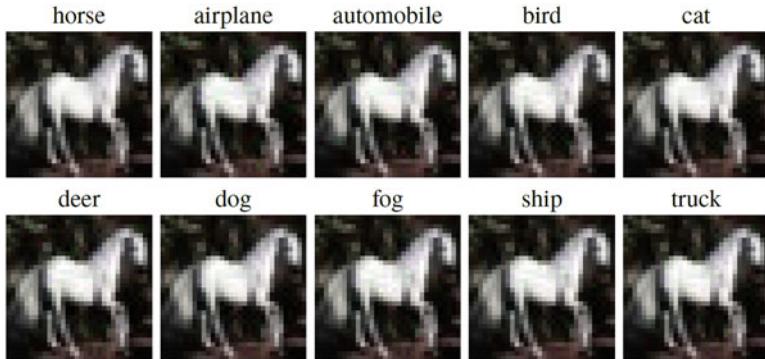


Fig. 14 Clean and adversarial images with different prediction labels, where the clean image of a horse and its adversarial images remain extremely similar, however, their prediction labels are quite distinct and each targets a totally different class

attacks are Fast Gradient Sign (FGS), Iterative Fast Gradient Sign (IFGS), and Jacobian-based saliency map attack (JSMA) methods [42]. Based on the similar principle, there are following attacks which do not require training data and also have less convergence time (in terms of queries):

- *TrISec*: This attack exploits the backpropagation algorithm to identify the small change (attack noise) in input pixels with respect to misclassification at the output, while ensuring the imperceptibility [25].
- *RED-Attack*: Most of the state-of-the-art attacks require a large number of queries to generate an imperceptible attack. However, in resource-constraint scenarios, these attacks may fail, therefore, we proposed a methodology that generates an attack image with imperceptible noise while requiring a very less number of queries [26].

2. Optimization-based Approaches: Unlike the gradient-based approaches, these attacks redefine the loss function (i.e., the cost function used for optimization) by adding extra constraints with respect to targeted or untargeted misclassification, and then propose different optimization algorithms to generate adversarial images. For example, Limited Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [54] and Carlini and Wagner (CW) [5] attacks use the box-constrained L-BFGS algorithm with single and multi-objective optimization, respectively.

Other types of neural networks, i.e., Capsule Networks and Spiking Neural Networks, are emerging as an alternative because of their robustness to affine transformations and potential for offering higher energy efficiency, respectively. However, recent works showed that these networks are also vulnerable to adversarial attacks [38, 39].

4.1.2 Backdoor Attacks

Due to expensive and computationally intensive training, the training (or fine-tuning) of DNNs is usually outsourced which opens the new frontiers of security threats, e.g., *backdoored neural networks (BadNets [14])*. These threats arise due to the involvement of untrusted third party service providers that can insert backdoors by training the ML models on compromised training data, or by altering the DNN structure. The untrusted third party also ensures the required accuracy of the backdoored model on most validation and testing inputs, but cause targeted misclassification or confidence reduction based on *backdoor trigger*. For example, in case of autonomous driving use case, an attacker can introduce the backdoor in a street sign detector while ensuring the required accuracy for classifying street signs in most of the cases, however, it can perform either targeted or untargeted misclassification, i.e., classifies stop signs with a particular sticker as speed limit signs or any other sign different from stop sign. This kind of misclassification can lead to catastrophic effects, e.g., in case of misclassification of a stop sign, autonomous vehicle does not stop at the intersection which can result in an accident.

4.2 Defences Against Security Attacks on DNNs

Several countermeasures have been proposed to defend against the adversarial attacks, i.e., *DNN masking*, *gradient masking*, *training for known adversarial attacks*, and *pre-processing of the CNN inputs* [6]. For examples, Fig. 15 shows

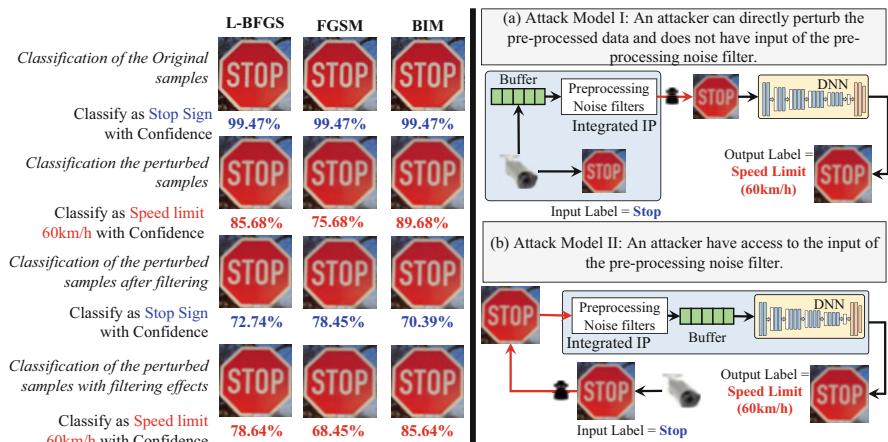


Fig. 15 Impact of the pre-processing filtering on the state-of-the-art adversarial attacks with different attack models with and without the access of filters. **(a)** Attack model I: an attacker can directly perturb the pre-processed data and does not have input of the pre-processing noise filter. **(b)** Attack model II: an attacker have access to the input of the pre-processing noise filter (adapted from [24, 27])

that *low-pass pre-processing filters that can nullify the adversarial attacks if they are not known to the attacker* [24, 27]. Therefore, based on this analysis, we have proposed to utilize the pre-processing quantization to improve the perceptibility of the attack noise [2]. Similarly, Sobel-filters can also be used to decrease the attack strength [55].

However, these defences are not applicable to backdoor-based attacks because the backdoor attacks intrude the networks and are activated through a specific trigger. Therefore, to address these attacks, we propose to use *pruning* as a natural defense because it eliminates the neurons that are dormant on clean inputs, consequently disabling backdoor behavior [14]. Although these defenses are effective, most of them provide defense against known adversarial and backdoor attacks. Therefore, one of the most important problems in designing secure machine learning systems is *the ability to define threats, and model them sufficiently so that any learning system can be trained to be able to identify such threats*.

4.2.1 Generative Adversarial Networks

To address the above-mentioned challenge, Generative Adversarial Networks (GANs) have emerged as one of the prime solutions because of their ability to generate the model by learning to mimic actual models [13]. In particular, GANs is a framework to estimate generative models where simultaneously two models are trained, *generator* (G) and *discriminator* (D) (see Fig. 16). This is achieved through an adversarial process where the two models are competing with each other for achieving two opposite goals. Simply speaking, D is trying to distinguish real images from fake images and G is trying to create images as close as possible to real images so as D will not be able to distinguish them, as illustrated in Fig. 16. When dealing with inference scenarios, the challenge is to provide a training set which includes attack-generated data patterns labeled of course correctly as attacks. For example, an autonomous system may rely on visual information to orient and steer itself or to undertake significant decisions. However, white or patterned noise can be maliciously inserted into a camera feed that may fool the system, and thus results in potentially catastrophic scenarios. The problem with modeling these types of attacks is that the attack models are hard to mathematically formulate, and thus

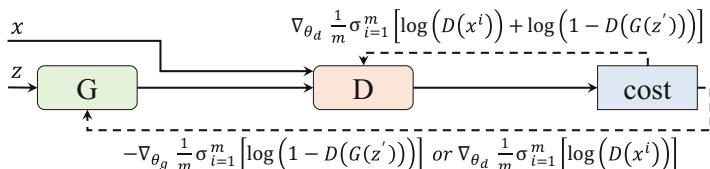


Fig. 16 GANs framework: an illustration of how G and D are trained, adapted from [21]

hard, if not impossible, to replicate and therefore, train the system to recognize them as attacks. Hence, GANs provide us with this capability, as we can utilize the G model to generate-and-evaluate threat models and train the D model to differentiate between what we consider an attack or not. However, GAN-based threat comes with the following challenges [21, 40]:

1. **Collapsing:** In this case G produces only a single sample or set of similar samples, regardless the type of input given to it.
2. **Convergence:** Since G and D models are competing towards achieving two opposite goals, this may make the model parameters to oscillate, destabilizing the training process.
3. **Gradient Vanish:** If one of the two models becomes more powerful than the other, the learning signal is becoming useless, making the system incapable to learn.
4. **Over-Fitting:** This is typically due to the unbalance optimization of G and the D models, e.g., if too much time is spent on minimizing G , then D will most likely collapse to a few states.
5. **Sensitive:** It is characterized by being highly sensitive to the selection of the hyperparameters, i.e., learning rate, momentum, etc.; making the training process much more tedious.

4.2.2 Case Study: Noisy Visual Data, and How GANs Can be Used to Remove Noise and Provide Robustness

To illustrate how a GAN-based framework can be used to define threats and subsequently to provide robustness in a DNN-based system, we use computer vision as an example because security threats in computer vision applications may arise from either physical attacks, cyber attacks or a combination of both. We use the hazing in images to model such threats. To remove this threat, we use the GANs because of their capability in preserving fine details in images and producing results that look perceptually convincing [28]. The goal is to translate the input image with haze, into a haze-free output image. In this case, the noise distribution z is the noisy image, and it is given as input to the GANs, i.e., haze input image. Afterwards, a new sample image F is generated by G . D will receive as input the generated image F and the ground truth haze-free image, to be trained to distinguish between real and artificially generated images (see Fig. 17).

This approach has recently been used for haze removal [8, 52, 63]. These methods mainly differ from each other, based on the utilized deep learning structure for G and D , i.e., using three types of G to solve the optimization of the haze removal problem [63], using the concept of cycle GAN introduced in [67]. Also they may differ for the type of loss function used for the training process, where the overall objective functions is constrained to preserve certain features or priors. However, they provide a solution that, in most of the cases, is capable to improve the quality performances of the state-of-the-art haze removal methods for single image, so as making this quite a promising area.

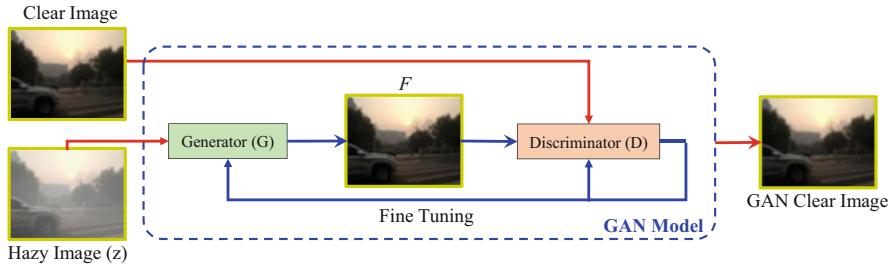


Fig. 17 Example of GANs used for removing haze noise from a single image. The haze image is input to G that generate an output image F and D receives as input both the generated image F and the free-haze image. Input images taken from [32]

5 Open Research Challenges

Machine learning has paved its way to a majority of the fields that involve data processing. However, regardless of all the work which has been carried out in interpreting the neural networks and making the ML-based systems reliable, there are still quite some challenges which are to be addressed before ML algorithms (specifically, DNNs) can be widely accepted for complex safety-critical applications. Following is a list of a few of the main challenges in this direction.

- **Error-Resilience Evaluation Frameworks:** One approach towards this for timing error estimation is proposed in [64]. However, more sophisticated frameworks are required to study the impact of multiple types of reliability threats and their interdependence in a time efficient manner.
- **Methodologies for Designing Robust and Resource-Efficient DNNs:** Retraining a DNN in the presence of hardware-induced faults [15] can improve their resilience. However, there is a need to investigate the types of DNN architectures which are inherently resilient to most (if not all) of the reliability threats. Furthermore, there is a need to investigate frameworks to develop robust ML systems by synergistically investigating reliability and security vulnerabilities.
- **Reliable and Resource-Efficient Hardware Architectures:** With all the security and reliability challenges highlighted in the chapter, there is a dire need to re-think the way current DNN hardware is designed, such that the vulnerabilities that cannot be addressed at the software-level have to be addressed through a robust DNN hardware.
- **Interpretability of Deep Neural Networks:** Developing interpretable DNNs is a challenge, however, it has to be addressed in order to better understand the functionality of the DNNs. This will help us in improving the learning capabilities of the DNNs, as well as in uncovering their true vulnerabilities and thereby will help in developing more efficient and robust network architectures.
- **Practicality of the Attacks:** With the ongoing pace of the research in ML, new methods and types of network architectures are surfacing, e.g., CapsuleNets.

Also, the focus of the community is shifting more towards semi-/un-supervised learning methods as they overcome the need for large labeled datasets. Therefore, there is a dire need to align the focus with the current trends in the ML community. Also, the attacks should be designed considering the constraints of the real systems, i.e., without making unrealistic assumptions about the number of queries and the energy/power resources available to generate an attack. An early work in this direction by our group can be found at [26].

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500—spp1500.itec.kit.edu) and in parts by the National Science Foundation under Grant 1801495.

References

1. Ahmad, H., Tanvir, M., Abdullah, M., Javed, M.U., Hafiz, R., Shafique, M.: Systimator: a design space exploration methodology for systolic array based CNNs acceleration on the FPGA-based edge nodes (2018). arXiv:1901.04986
2. Ali, H., Tariq, H., Hanif, M.A., Khalid, F., Rehman, S., Ahmed, R., Shafique, M.: QuSecNets: quantization-based defense mechanism for securing deep neural network against adversarial attacks (2018). arXiv:1811.01437
3. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples (2018). arXiv:1802.00420
4. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 2654–2662. Curran Associates, New York (2014). <http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf>
5. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks (2016). arXiv:1608.04644
6. Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: a survey (2018). arXiv:1810.00069
7. Deng, J., Dong, W., Socher, R., Li, L.: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). <https://doi.org/10.1109/CVPR.2009.5206848>
8. Engin, D., Genç, A., Ekenel, H.K.: Cycle-Dehaze: enhanced cycleGAN for single image dehazing. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, June 18–22, 2018, pp. 825–833 (2018). <https://doi.org/10.1109/CVPRW.2018.00127>. http://openaccess.thecvf.com/content_cvpr_2018_workshops/w13/html/Engin_Cycle-Dehaze_Enhanced_CycleGAN_CVPR_2018_paper.html
9. Ernst, D., Das, S., Lee, S., Blaauw, D., Austin, T., Mudge, T., Kim, N.S., Flautner, K.: Razor: circuit-level correction of timing errors for low-power operation. IEEE Micro **24**(6), 10–20 (2004)
10. Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., Dean, J.: A guide to deep learning in healthcare. Nat. Med. **25**(1), 24 (2019)
11. Fink, M., Liu, Y., Engstle, A., Schneider, S.A.: Deep learning-based multi-scale multi-object detection and classification for autonomous driving. In: Fahrerassistenzsysteme 2018, pp. 233–242. Springer, Berlin (2019)

12. Gebregiorgis, A., Kiamehr, S., Tahoori, M.B.: Error propagation aware timing relaxation for approximate near threshold computing. In: Proceedings of the 54th Annual Design Automation Conference 2017, p. 77. ACM, New York (2017)
13. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pp. 2672–2680. MIT Press, Cambridge (2014). <http://dl.acm.org/citation.cfm?id=2969033.2969125>
14. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: identifying vulnerabilities in the machine learning model supply chain (2017). arXiv:1708.06733
15. Hacene, G.B., Leduc-Primeau, F., Soussia, A.B., Gripon, V., Gagnon, F.: Training modern deep neural networks for memory-fault robustness. In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE, Piscataway (2019)
16. Hanif, M.A., Hafiz, R., Shafique, M.: Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In: 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 913–916. IEEE, Piscataway (2018)
17. Hanif, M.A., Khalid, F., Putra, R.V.W., Rehman, S., Shafique, M.: Robust machine learning systems: reliability and security for deep neural networks. In: 2018 IEEE 24th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp. 257–260. IEEE, Piscataway (2018)
18. Hanif, M.A., Putra, R.V.W., Tanvir, M., Hafiz, R., Rehman, S., Shafique, M.: MPNA: a massively-parallel neural array accelerator with dataflow optimization for convolutional neural networks (2018). arXiv:1810.12910
19. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: Proceedings of the 50th Annual Design Automation Conference, p. 99. ACM, New York (2013)
20. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors (2012). arXiv:1207.0580
21. Hui, J.: Gan why it is so hard to train generative adversarial networks! Elsevier, Amsterdam (2018). https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
22. Jia, J., Gong, N.Z.: Attriguard: a practical defense against attribute inference attacks via adversarial machine learning. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 513–529 (2018)
23. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1–12. IEEE, Piscataway (2017)
24. Khalid, F., Hanif, M.A., Rehman, S., Qadir, J., Shafique, M.: Fademl: understanding the impact of pre-processing noise filtering on adversarial machine learning (2018). arXiv:1811.01444
25. Khalid, F., Hanif, M.A., Rehman, S., Shafique, M.: ISA4ML: training data-unaware imperceptible security attacks on machine learning modules of autonomous vehicles (2018). arXiv:1811.01031
26. Khalid, F., Ali, H., Hanif, M.A., Rehman, S., Ahmed, R., Shafique, M.: Red-attack: resource efficient decision based attack for machine learning (2019). arXiv:1901.10258
27. Khalid, F., Hanif, M.A., Rehman, S., Qadir, J., Shafique, M.: FAdeML: understanding the impact of pre-processing noise filtering on adversarial machine learning. In: Design, Automation and Test in Europe. IEEE, Piscataway (2019)
28. Kupyn, O., Budzan, V., Mykhailych, M., Mishkin, D., Matas, J.: Deblurgan: blind motion deblurring using conditional adversarial networks. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
29. Kwon, H., Samajdar, A., Krishna, T.: MAERI: enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 461–475. ACM, New York (2018)

30. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). <https://doi.org/10.1109/5.726791>
31. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
32. Li, B., Ren, W., Fu, D., Tao, D., Feng, D., Zeng, W., Wang, Z.: Benchmarking single-image dehazing and beyond. IEEE Trans. Image Process. **28**(1), 492–505 (2019)
33. Limbrick, D.B., Mahatme, N.N., Robinson, W.H., Bhuvan, B.L.: Reliability-aware synthesis of combinational logic with minimal performance penalty. IEEE Trans. Nuclear Sci. **60**(4), 2776–2781 (2013)
34. Lu, W., Yan, G., Li, J., Gong, S., Han, Y., Li, X.: FlexFlow: a flexible dataflow accelerator architecture for convolutional neural networks. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 553–564. IEEE, Piscataway (2017)
35. Lyons, R.E., Vanderkulk, W.: The use of triple-modular redundancy to improve computer reliability. IBM J. Res. Development **6**(2), 200–209 (1962)
36. Marchisio, A., Shafique, M.: Capstore: energy-efficient design and management of the on-chip memory for CapsuleNet inference accelerators (2019). arXiv:1902.01151
37. Marchisio, A., Hanif, M.A., Shafique, M.: CapsAcc: an efficient hardware accelerator for CapsuleNets with data reuse (2018). arXiv:1811.08932
38. Marchisio, A., Nanfa, G., Khalid, F., Hanif, M.A., Martina, M., Shafique, M.: Capsattacks: robust and imperceptible adversarial attacks on capsule networks (2019). arXiv:1901.09878
39. Marchisio, A., Nanfa, G., Khalid, F., Hanif, M.A., Martina, M., Shafique, M.: SNN under attack: are spiking deep belief networks vulnerable to adversarial examples? (2019). arXiv:1902.01147
40. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J.: Unrolled generative adversarial networks. In: Proceedings of the International Conference on Learning Representations (ICLR’17) (2017)
41. Nakhaee, F., Kamal, M., Afzali-Kusha, A., Pedram, M., Fakhraie, S.M. Dorost, H.: Lifetime improvement by exploiting aggressive voltage scaling during runtime of error-resilient applications. Integr. VLSI J. **61**, 29–38 (2018)
42. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE, Piscataway (2016)
43. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 506–519. ACM, New York (2017)
44. Putra, R.V.W., Hanif, M.A., Shafique, M.: ROMANet: fine-grained reuse-driven data organization and off-chip memory access management for deep neural network accelerators (2019). arXiv:1902.10222
45. Raghu Nathan, B., Turakhia, Y., Garg, S., Marculescu, D.: Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: 2013 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 39–44. IEEE, Piscataway (2013)
46. Reagen, B., Gupta, U., Pentecost, L., Whatmough, P., Lee, S.K., Mulholland, N., Brooks, D., Wei, G.Y.: Ares: a framework for quantifying the resilience of deep neural networks. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2018)
47. Rehman, S., Shafique, M., Henkel, J.: Reliable Software for Unreliable Hardware: A Cross Layer Perspective. Springer, Berlin (2016)
48. Schorn, C., Guntoro, A., Ascheid, G.: Efficient on-line error detection and mitigation for deep neural network accelerators. In: International Conference on Computer Safety, Reliability, and Security, pp. 205–219. Springer, Berlin (2018)
49. Shafique, M., Garg, S., Henkel, J., Marculescu, D.: The EDA challenges in the dark silicon era: temperature, reliability, and variability perspectives. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
50. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE, Piscataway (2017)

51. Suciu, O., Marginean, R., Kaya, Y., Daume III, H., Dumitras, T.: When does machine learning {FAIL}? Generalized transferability for evasion and poisoning attacks. In: 27th {USENIX} Security Symposium ({USENIX} Security'18), pp. 1299–1316 (2018)
52. Swami, K., Das, S.K.: Candy: conditional adversarial networks based fully end-to-end system for single image haze removal (2018). <https://arxiv.org/abs/1801.02892v2>
53. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017)
54. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). arXiv:1312.6199
55. Tariq, H., Ali, H., Hanif, M.A., Khalid, F., Rehman, S., Ahmed, R., Shafique, M.: SSCNets: a selective sobel convolution-based technique to enhance the robustness of deep neural networks against security attacks (2018). arXiv:1811.01443
56. Tiwari, A., Torrellas, J.: Facelift: hiding and slowing down aging in multicores. In: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, pp. 129–140. IEEE Computer Society, Washington (2008)
57. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: 25th {USENIX} Security Symposium ({USENIX} Security'16), pp. 601–618 (2016)
58. Vadlamani, R., Zhao, J., Burleson, W., Tessier, R.: Multicore soft error rate stabilization using adaptive dual modular redundancy. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 27–32. European Design and Automation Association, Leuven (2010)
59. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International Conference on Machine Learning, pp. 1058–1066 (2013)
60. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 36–52. IEEE, Piscataway (2018)
61. Wei, X., Yu, C.H., Zhang, P., Chen, Y., Wang, Y., Hu, H., Liang, Y., Cong, J.: Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In: Proceedings of the 54th Annual Design Automation Conference 2017, p. 29. ACM, New York (2017)
62. Whatmough, P.N., Das, S., Bull, D.M., Darwazeh, I.: Circuit-level timing error tolerance for low-power DSP filters and transforms. IEEE Trans. Very Large Scale Integration Syst. **21**(6), 989–999 (2013)
63. Yang, X., Xu, Z., Luo, J.: Towards perceptual image dehazing by physics-based disentanglement and adversarial training. In: Thirty-Second AAAI Conference on Artificial Intelligence (AAAI) (2018)
64. Zhang, J.J., Garg, S.: Fate: fast and accurate timing error prediction framework for low power DNN accelerator design. In: Proceedings of the International Conference on Computer-Aided Design, p. 24. ACM, New York (2018)
65. Zhang, J., Rangineni, K., Ghodsi, Z., Garg, S.: ThUnderVolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep neural network accelerators (2018). arXiv:1802.03806
66. Zhang, J.J., Gu, T., Basu, K., Garg, S.: Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In: 2018 IEEE 36th VLSI Test Symposium (VTS), pp. 1–6. IEEE, Piscataway (2018)
67. Zhu, J., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2242–2251 (2017). <https://doi.org/10.1109/ICCV.2017.244>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Exploiting Memory Resilience for Emerging Technologies: An Energy-Aware Resilience Exemplar for STT-RAM Memories



**Amir Mahdi Hosseini Monazzah, Amir M. Rahmani, Antonio Miele,
and Nikil Dutt**

1 Introduction

In the recent years, the aggressive progress in technology scaling has allowed to integrate a larger number of processing cores in the same chip thus leading to the fabrication of multicore and manycore devices. To efficiently exploit such processing power, it is imperative to proportionally increase the performance and bandwidth of the on-chip cache memory sub-system.

Unfortunately, the commonly-used static RAM (SRAM) technology imposes fundamental limitations for the quest of high memory performance in the next generation computing systems. Indeed, the low density of SRAM cells forces to dedicate approximately 60% of the area of today's chips to the cache memories [7, 19]. Moreover, SRAM memories present a considerably high leakage power consumption becoming a considerable issue with the continuous technology scaling

A. M. H. Monazzah (✉)
Iran University of Science and Technology (IUST), Tehran, Iran

Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
e-mail: monazzah@iust.ac.ir

A. M. Rahmani
University of California, Irvine (UCI), Irvine, CA, USA
e-mail: a.rahmani@uci.edu

Institute of Computer Technology, TU Wien, Austria
e-mail: antonio.miele@polimi.it

A. Miele
Politecnico di Milano, Milano, Italy
e-mail: dutt@uci.edu

N. Dutt
University of California, Irvine (UCI), Irvine, CA, USA

beyond 40 nm leading to the leakage power to contribute up to 80% of the overall energy consumption of the cache memories [7].

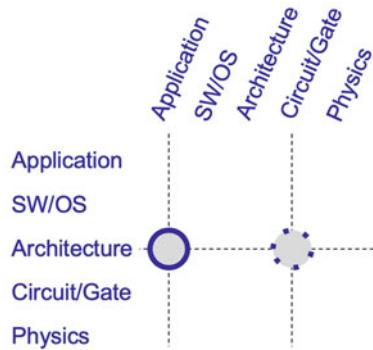
For these reasons, a recent trend is to consider Non-Volatile Memory (NVM) technologies, such as Phase Change Memory (PCM), Resistive RAM (ReRAM), Ferroelectric RAM (FeRAM), and Spin Transfer Torque Magnetic RAM (STT-MRAM or STT-RAM) as alternative solutions to SRAMs in multicore and many-core chips [23]. Among them, various studies [2, 3, 11, 17, 20] have observed that STT-RAM represents the most promising technology for on-chip cache memories. In particular, STT-RAM allows to increase the memory size, thanks to its higher density, and outperforms the SRAM counterpart in terms of energy consumption. In fact, since STT-RAM is a non-volatile technology, its leakage power consumption is negligible.

Unfortunately, STT-RAM technology suffers from a different set of reliability issues that has to be carefully addressed in order to realize its deployment in commercial products. STT-RAMs present a high susceptibility to failure both during write/read operations and in idle status [2, 23]. More precisely, the memory cell may suffer from *retention errors*, which are caused by thermal noises that lead to an unintentional bit flip of the value stored in an idle cell. Moreover, during read operations, it may occur that the cell content incorrectly flips leading to the so-called *read disturbance*, or the returned value has an undistinguished state, dubbed as *false read*. Finally, also the write operation may suffer from *write errors*, which are caused by thermal fluctuations in the magnetization process that lead to storing a wrong value in the cell w.r.t. the one in the input. Among these reliability threats, write errors impose the most challenging issue [2, 7].

A circuit-level strategy to eliminate write errors is to increase the current (voltage) applied during the write operation [23]. However, it mainly leads to higher energy consumption, and secondarily to a higher probability of permanent failures of the device. Indeed, higher current implies an increase in the temperature and in turn to an increase in the probability of junction barrier breakdown [7]. An alternative design strategy is the use of Error Correction Codes (ECCs) to harden the architecture of the cache memory [1]. However, if it is applied in a naive way, it may result in a significant area overhead. For instance, when Bose–Chaudhuri–Hocquenghem (BCH) 7 ECC is employed, such overhead can be as high as 15% of the data block area [2]. Such additional area causes a significant energy overhead. Given such rationale, several studies [2, 3, 6, 7, 17, 18, 20, 21, 24] have individually investigated these techniques and proposed strategies to improve their effectiveness. Given this background, we believe that there is an opportunity for a larger improvement of these hardening schemes by holistically amalgamating these two techniques. Moreover, an opportunistic integration and tuning of these two techniques can also lead to a considerable improvement in energy consumption of the cache memory architecture while at the same time guaranteeing the error rate threshold.

This chapter proposes FlexRel, a reliability improvement technique which utilizes the STT-RAM write current as an actuation knob and multi-level ECC

Fig. 1 Positioning of the proposed approach in the overall cross-layer vision of the book



protection scheme to conduct an optimal trade-off between reliability and energy consumption in STT-RAM cache memories. Targeting an overall block write error rate threshold that should be guaranteed in applications running on a platform, FlexRel proposes a cache way partitioning scheme that utilizes different combinations of write currents and ECC protection codes in each partition to satisfy that threshold. Then, during the run-time of applications, the FlexRel controller redirects the more vulnerable blocks to more robust partitions to keep the write error rate below the write error threshold. Within the overall cross-layer vision of the book, the main contribution of this chapter can be primarily classified as an architecture-to-application cross-layer approach, also using gate/circuit-level actuation, as illustrated in Fig. 1. In fact, this approach exploits application-level profiled information as an input to an architecture-level memory hardening technique based on ECC while using current tuning at circuit-level, with the final goal of optimizing energy consumption and application reliability.

We evaluate FlexRel using gem5 simulator [4] running SPEC CPU2006 [9] workloads. We compare the efficiency of FlexRel against an optimized uniform protection (OUP) scheme from reliability, energy, area, and performance perspectives. The simulation results show that, while FlexRel meets the write error rate threshold, it outperforms OUP scheme in terms of energy and area by up to 13.2 and 7.9%, respectively. Furthermore, The restriction of write traffics to specific partitions in FlexRel incurs only a 1.7% performance overhead to the system, on average.

The rest of the chapter is organized as follows. Section 2 introduces the necessary background presenting the basic architecture of a STT-RAM cell and the energy/error rate issues of this technology. Section 3 briefly surveys the previous approaches for hardening STT-RAMs highlighting the adopted strategies and differentiating them from the proposed approach. Section 4 is the core of this chapter and presents our proposed FlexRel approach, consisting of an enhanced memory architecture capable of trading off reliability and energy consumption. The proposed solution has been experimentally validated and results are discussed in Sect. 5. Finally, Sect. 6 draws conclusions.

2 STT-RAMs and Their Energy-Reliability Challenges

This section introduces the basics of STT-RAM technology, its architecture and how read/write operations are performed. The second part of the section focuses on energy vs. write error issues in this type of memory. This discussion represents the preliminaries for the proposed energy-aware error-tolerant scheme for STT-RAM.

2.1 Basic Architecture of STT-RAM

Figure 2 shows the basic cell structure of a STT-RAM, called 1 Transistor 1 Magnitude Tunnel Junction (MTJ), shortly 1T-1J. The cell is constructed from an MTJ element and an access NMOS transistor. MTJ itself includes three layers which are a MgO-based barrier (called *tunneling oxide barrier*), a ferromagnetic layer with fixed magnetic field direction (called *reference layer*), and a ferromagnetic layer with free magnetic field direction (called *free layer*). The MgO-based barrier layer is sandwiched between two ferromagnetic layers. STT-RAM works based on the relative magnetic field direction of the free layer and the reference layer (parallel or anti-parallel states). As shown in the figure, the parallel state will represent a logic value "0" while the anti-parallel one a logic value "1." The different relative ferromagnetic field directions lead to different resistances in MTJ, i.e. R_{High} and R_{Low} (R_H and R_L) [5]. In the following, we explore the read and write operation mechanisms in a STT-RAM cell.

The read operation in a STT-RAM cell is initiated by setting the word line (WL in the figure) to turn on the access NMOS transistor. Then, a small read current I_R (or read voltage, V_R) is applied to the MTJ from the source line (SL in the figure) through an access transistor [30]. By applying I_R to MTJ, a current (or voltage) is

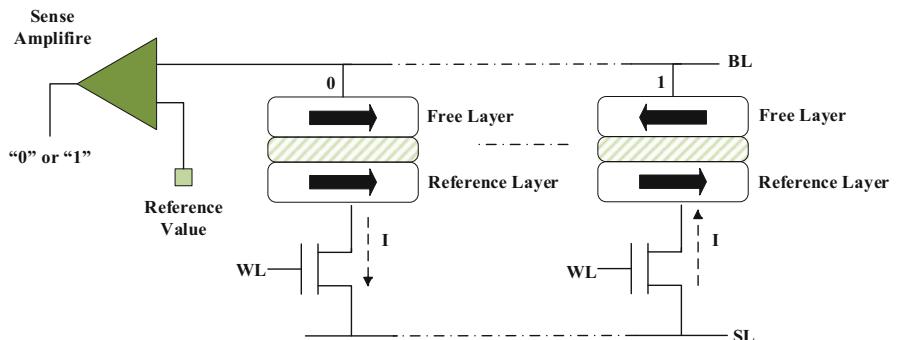


Fig. 2 A typical STT-RAM cell structure: on the left side, a parallel magnetic field direction represents a logic value 0, while on the right side, an anti-parallel magnetic field direction a logic value "1"

sensed on the bit line (BL in the figure). Based on the sensed values of current or voltage in the BL during the read operations, the resistance of MTJ is determined as high (R_H) or low (R_L). The STT-RAM cell value is determined by calculating the Tunneling Magneto Resistance (TMR), a ratio parameter defined as

$$\text{TMR} = \frac{(R_H - R_L)}{R_L} \quad (1)$$

The parallel (anti-parallel) state of MTJ leads to low (high) resistance of MTJ, calculated by means of the following equations:

$$V_{BL_L} = I_R \times (R_L + R_{NMOS}) \quad (2)$$

$$V_{BL_H} = I_R \times (R_H + R_{NMOS}) \quad (3)$$

If the sense amplifier which calculates the voltage (current) of bit line decides the sensed voltage is the same as V_{BL_L} , the MTJ value is logically “0,” otherwise, if the sensed voltage is the same as V_{BL_H} , MTJ value is logically “1.”

To perform a write operation and modify the value stored in MTJ, we need to change the magnetic field direction of MTJ free layer. By changing the magnetic field direction of the free layer, the resistance of MTJ will change [5, 10]. To this end, again the access transistor should be turned on by setting the word line. Then, a write current is applied from the source line to the baseline or vice versa. The direction of applied write current determines the magnetic field direction of the free layer. By applying the write pulse to the MTJ, when the amount of spin polarized current exceeds a threshold value, the magnetic field direction of the free layer flips.

2.2 Error Rate vs. Energy Consumption Trade-Off

One of the main issues in STT-RAM is its stochastic switching nature caused by the effects of thermal fluctuations. Among the side-effects of stochastic switching, write failure is the most important reliability challenge [2, 7]. More precisely, write failure occurs during the write operation and its effect is that the value stored in the MTJ will be different from the one provided as data input. From a physical point of view, it happens according to the stochastic behavior of STT-RAM cell when the magnetic field direction of the free layer could not change during the pre-determined write pulse width [12, 27]. There are many parameters that contribute to switching the MTJ state during the write operations, e.g., MTJ switching current, process variations, thermal fluctuations, and switching pulse width. According to [16], the write failure probability can be calculated using the following equation:

$$P_{wf}(t_w) = \exp\left(-t_w \cdot \frac{2\mu_B p (I_w - I_{C_0})}{(c + \ln\left(\frac{\pi^2 \Delta}{4}\right)) \cdot (em(1 + p^2))}\right) \quad (4)$$

where Δ is the thermal stability factor, I_{C_0} is the critical MTJ switching current at 0 °K, c is the Euler constant, e is the magnitude of electron charge, m denotes the magnetic momentum of the free layer, p is the tunneling spin polarization, μ_B is the Bohr magneton, I_w is the write current, and t_w is the write pulse width.

While STT-RAM technology is a promising candidate to resolve the static energy challenge of SRAM technology in on-chip memories, from the dynamic energy consumption perspective, it imposes considerable energy consumption for a reliable write operation due to its stochastic switching feature; the higher the I_w , the lower the write error rate of STT-RAM will be. For example, we used NVSim [8] to experimentally compare two alternative implementations, in SRAM and STT-RAM technologies, with the same 32 KB cache architecture with 64 Byte word lines implemented in 45 nm. Our results show that from the leakage power point of view, the SRAM cache imposes 41.896 mW power consumption, while STT-RAM cache only charges 9.066 mW power consumption to the design. On the other hand, each read operation in SRAM and STT-RAM implementations uses 11.421 and 82.493 pJ dynamic energy consumption, respectively. Finally, each write operation in SRAM and STT-RAM technologies enforces 5.712 and 534.375 pJ dynamic energy consumption to the design, respectively. As a conclusion, I_w in Eq. 4 is the main contributor for dynamic energy consumption in STT-RAM. Accordingly, I_w is one of the effective circuit-level knobs available to control the reliability-energy trade-off during a write operation. Generally, a lower I_w decreases the write energy, but it also amplifies the probability of write failure.

To systematically analyze this aspect, we performed a quantitative evaluation of the write error rate of STT-RAM at different write current amplitudes by using the STT-RAM SPICE model introduced in [14]. In particular, we characterized a STT-RAM cell by using parameters reported in Table 1 and ran several Monte Carlo simulations. Figure 3 depicts the write error rate of the STT-RAM cell when the write current is varied and the cell is flipped from 0 → 1 (in red) or vice versa (in blue). As shown in Fig. 3, we retrieved the trend lines of error rate patterns in both directions to generate the STT-RAM write error rate formulas. These formulas are useful to estimate the write error rates of STT-RAM at any write current. We therefore conclude that I_w is an effective circuit-level knob available to control the

Table 1 STT-RAM HSPICE model configurations

Parameter	Value ($\mu \pm 3\sigma$)
MTJ length	32 nm
MTJ width	96 nm
MTJ thickness	2.44 nm
Relative initial angle	$0 \pm 35^\circ / 180 \pm 35^\circ$
Transistors technology size	32 ± 1 nm

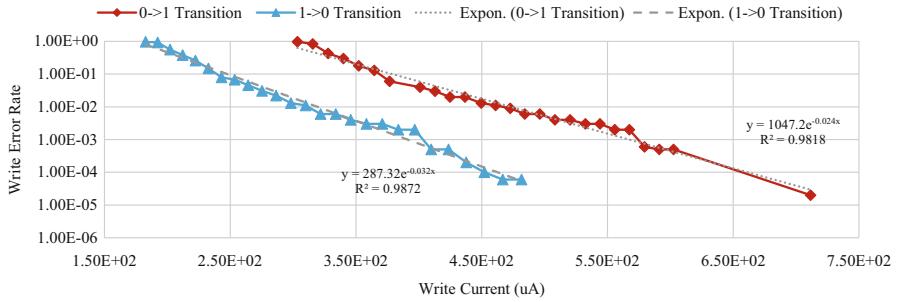


Fig. 3 STT-RAM cell write error rate vs. write current in different MTJ state transitions (write pulse width fixed at 10 ns)

quality-energy trade-off during a write operation. Moreover, it can be noted that the write error rate is asymmetric and $0 \rightarrow 1$ is the most critical transition. This is due to the fact that the initial MTJ state affects the total energy required to change its resistance [29]. For $0 \rightarrow 1$ bit transition, to change the MTJ state from parallel (low resistance) to anti-parallel (high resistance), more energy (power \cdot time) needs to be spent compared to the amount of energy needed for the transition in the opposite direction ($1 \rightarrow 0$) [28]. As a consequence, the MTJ state transition is asymmetric from the error rate vs. energy consumption perspective.

Finally, it is worth mentioning that unlike the provided knob in SRAM technology (memory bank voltage scaling) that is coarse-grained and affects a large portion of data in memory, STT-RAM exploits I_w which is fine-grained and can be tuned for granularity of a data block in memory. As we show in the following, this feature offered by STT-RAM provides a unique opportunity for flexible adjustment of the energy-reliability knob.

3 Related Work on STT-RAM Reliability

In recent years, several studies have addressed reliability issues of STT-RAM. In [23], the authors present a survey of the preliminary approaches addressing faults in various non-volatile memory technologies with the focus on both permanent and transient faults. Regarding permanent faults, the basic strategies are to (1) increase the current during write operations and (2) augment the architecture by using Error Correction Codes (ECCs). Indeed these strategies are the ones later used by many subsequent approaches.

In [22], the authors propose a strategy, Verify and Correct (VnC), which consists of reading each value immediately after the write operation in the STT-RAM cache to assess its correctness. Since read delay is negligible, such an approach may lead to performance degradation in case of high failure rate. The approach is later enhanced by combining VnC with a limited ECC to reduce the need of rewriting upon

failures. Ahn et al. [1] propose a scheme where ECC is shared among several cache blocks thus reducing its hardware cost. Cheshmikhani et al. [6] and Azad et al. [3] present ECC schemes with an optimized interleaved bit selection mechanism to minimize codewords' vulnerability variations. Finally, the authors of [25] introduce adaptiveness in the hardening scheme; where two different levels of ECC can be selectively chosen for each incoming block in the STT-RAM cache. The level of protection is selected based on the vulnerability of the incoming block which is calculated by enumerating the number of $0 \rightarrow 1$ bit transitions. The idea of using several ECC scheme in an adaptive way is further explored in [2] by defining an STT-RAM cache architecture, called A²PT using several ECC levels and integrating a specific hardware module which selects the replacement candidate in order to minimize the Hamming distance between the stored block the newly incoming one.

A different strategy is proposed in [11] where the classical Least-Recently-Used (LRU) cache replacement policy is substituted with a new algorithm which performs a Least-Error-Rate (LER) replacement. To reduce the probability of write errors in STT-RAM, this algorithm tries to write the incoming block in a location which imposes the least number of $0 \rightarrow 1$ bit transitions among the victim block candidates. In [7], the authors observe that the stochastic switching in write operations is mainly caused by the device heating. Therefore, to reduce the write errors, they propose to replace the LRU policy with a thermal-aware counterpart, which tries to write the incoming block in a location which imposes the least temperature increase among the victim block candidates.

As discussed in Sect. 2, acting on the current applied during the write operation sensibly affects the correctness of the stored value; moreover, $0 \rightarrow 1$ is the most susceptible bit transition. For these reasons, Kim et al. [13] propose two different circuit design techniques applied at each single bit-line to balance out the asymmetric write current and optimize the memory design in terms of write-power and reliability. In a similar manner, Monazzah et al. [17] exploit the tuning of the write current to explicitly trade memory reliability for energy saving in the context of approximate computing. The approach considers software applications capable of tolerating a certain degree of errors in the results, such as image processing applications. Therefore, for each write operation, the current to be applied is dynamically selected based on the reliability requirement annotated in the application source code as well as the Hamming distance between the block to be written and the candidate to be replaced. In such a way, the energy consumption is minimized under a predefined number of errors that can be ignored in the application output.

The main contribution of our approach presented in this chapter is to holistically integrate ECC deployment and write current tuning based on our prior works presented in [2] and [17]. The main property of our approach is its self-adaptiveness to dynamically tune the system operating point to the characteristics of the running applications to optimize the energy consumption of the system while keeping the observed error rate under control.

4 FlexRel: An Energy-Aware Reliability Improvement Approach for STT-RAM Caches

In this section, we present our proposed approach called FlexRel. As a preliminary discussion, first we explore the conventional ECC deficiency in tolerating the write failures of STT-RAM caches. Then, we observe how different data patterns lead to different write error rates in cache blocks. At the end, we discuss in detail the FlexRel approach for the STT-RAM caches which utilizes the STT-RAM write current actuation knob and multi-level ECC protection scheme to conduct an optimum trade-off between reliability and energy consumption.

4.1 The Effects of Write Patterns on ECC Protection Level

As mentioned in previous sections, the write error rate in the STT-RAM cache depends on the bit differences between the contents of data that was previously stored in the cache block and the contents of the new incoming block. Indeed, during a write operation while failure may occur in the bit locations that should be toggled, for the other bit locations that will not experience any toggle, we do not observe any write error. In Sect. 2.2, we observed that in STT-RAM the write error rates of $0 \rightarrow 1$ bit transitions is higher than $1 \rightarrow 0$ bit transitions by about two orders of magnitude for the same current amplitude in both directions. Accordingly, FlexRel will mainly focus on $0 \rightarrow 1$ bit transitions since they represent the main contributor to write error rate in STT-RAM.

Generally, the total number of $0 \rightarrow 1$ bit transitions in a STT-RAM cache block is proportional to the Hamming Weight (HW) of the new incoming block, that is the total number of 1 in the bit representation of each block [2, 26]. On the other hand, the maximum number of $0 \rightarrow 1$ bit transitions in a cache block write operation happens when all of the bit locations storing value “1” in the new incoming block should store on bit locations that previously contained “0.” Considering this fact, for a STT-RAM cache that is protected with an ECC code with t -bit error correction capability, we can estimate the Block Error Rate (BER) of a STT-RAM cache write operation according to Eq. 5 [26]:

$$\text{BER}(w, t) \approx 1 - \sum_{i=0}^t C_w^i P_{ER\ 0 \rightarrow 1}^i (1 - P_{ER\ 0 \rightarrow 1})^{w-i} \quad (5)$$

where, $P_{ER\ 0 \rightarrow 1}$ is the bit failure rate in $0 \rightarrow 1$ switching, t the error correction capability of ECC, w the HW of the incoming data, and C_w^i the combination of HW taken i at a time.

We conducted an experimental evaluation of the BER of STT-RAM cache write operations for the incoming blocks when varying HW. In particular, we configured

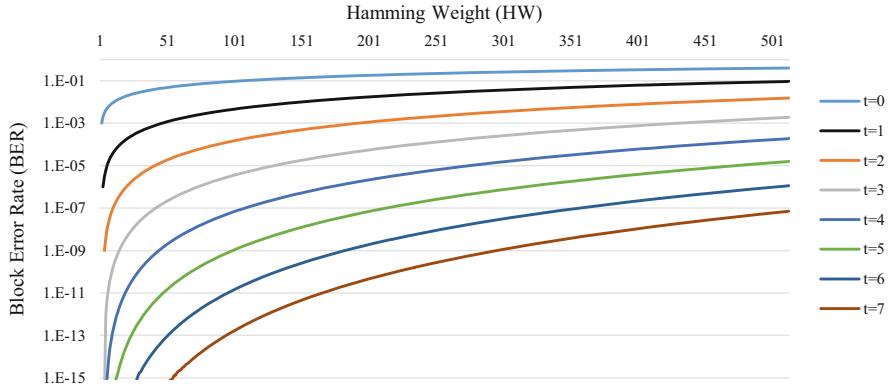


Fig. 4 Block Error Rate (BER) in different ECC schemes for various Hamming Weight (HW) data

a 512-bit STT-RAM cache block, with $0 \rightarrow 1$ write error rate, i.e., $P_{ER\ 0 \rightarrow 1}$ of 10^{-3} as reported in [1, 22, 26]. Figure 4 depicts the results of the experiments which verify that the BER of the cache blocks with different ECC protection scheme are considerably affected by the HW of the incoming data. For example, if we consider ECC with protection capability of t bit errors in a cache block, for HW of 25 we observed the BER of 10^{-15} , while for HW of 50 and 100 this protection scheme delivers BER of 10^{-13} and 10^{-11} , respectively, which is considerably different.

The results shown in Fig. 4 are calculated based on Bose, Chaudhuri, and Hocquenghem (BCH) coding scheme [15] which is a well-known scheme in protecting memory architectures. In BCH code, ECC converts to k -bit data and $(n - k)$ ECC check bits. The complexity of the peripherals that is required to protect the k -bit data is also increased with the increase in the ECC protection capability. Generally, for protecting a $k = 512$ bit cache line using BCH code with $(t + 1)$ bit error correction capability, we require $(10t + 1)$ check bits [2]. The results demonstrated in Fig. 4 is calculated for various coding schemes: SECDED (Single Error Correction-Double Error Detection), DEC-TED (Double Error Correction-Triple Error Detection), 3EC4ED, 4EC5ED, 5EC6ED, 6EC7ED, and 7ED8EC codes with $t = 1$, $t = 2$, $t = 3$, $t = 4$, $t = 5$, $t = 6$, and $t = 7$ error(s) correction capabilities, respectively.

With the emergence of ECC protection scheme in the cache memories to protect the data, conventionally, all the cache blocks are protected with the same ECC protection capability (t). This conventional ECC protection scheme is called Uniform, i.e., all of the blocks in the cache utilize the same ECC protection level. However, as we can see in Fig. 4, the different HWs in write requests lead to various BER for each cache block during the execution time. Accordingly, in the Uniform protection scheme the highest HW needs to be considered to select an ECC protection level that satisfies the write error rate threshold. As an example, considering Fig. 4, the ECC protection level $t = 6$ should be selected to satisfy the

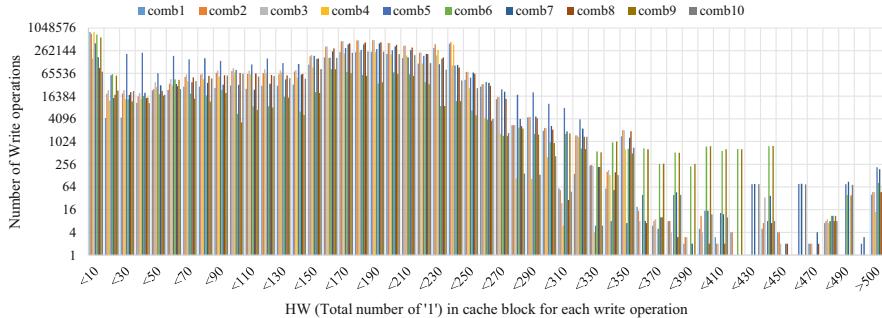


Fig. 5 The distribution of write operations based on Hamming Weight (HW) across a 4-MB 16-way shared L2 cache

write error rate threshold of 10^{-6} in the worst case where all of the bit locations in the target cache block experience toggle for the new incoming write request. However, satisfying a write request that all of its bit locations experience $0 \rightarrow 1$ bit transition is a very rare event. Therefore, using ECC with $t = 6$ correction capability is usually more than enough.

Accordingly, we investigated the distribution of HW across a 4-MByte 16-way set associative shared L2 cache during the execution of workloads. To this end, we ran combinations of benchmarks that are selected from SPEC CPU2006 [9] benchmark suite.¹ Figure 5 depicts the distribution of write requests' HWs across the shared L2 cache. Figure 5 illustrates that most of the write requests had less than 350 HW, while for the uniform full-protection ECC schemes we need to consider the worst-case HW (512 in 512 bit cache line size) for reliable write operations leading to significant under-utilization of resources.

One of the main concerns in utilizing uniform full-protection ECC configuration for the caches is the amount of energy consumption that is imposed to the system. Indeed, as illustrated in Figs. 4 and 5, while utilizing the full-protection configuration guarantees the reliable write operations, it imposes high energy consumption for most of the time that the write operations contain lower number of value 1 than the considered ECC-related threshold. For this reason, FlexRel exploits a non-uniform multi-protection level ECCs scheme to save energy. In addition, it improves the hardening scheme by deploying different write current levels introducing different STT-RAM write error rates.

¹The details of simulator configurations and workload combinations will be mentioned later in Sect. 5, in particular in Tables 3 and 4.

4.2 FlexRel Organization

In FlexRel, we divide a cache memory into several protection level zones. Unlike the previous studies that benefit only from different ECC codes to conduct multi-level protection scheme (e.g. [2, 3, 25]), in FlexRel we consider a combination of write current level and ECC protection level to satisfy a pre-determined write error rate threshold at each zone. The main strategy that is considered in FlexRel for cache partitioning is to assign the lowest possible write current for the zones that face high amounts of write operations to alleviate STT-RAM high write energy consumption and instead apply stronger ECC protection codes in these zones to satisfy the write error threshold. On the other hand, for the zones that experience low amounts of write operations we consider high write current with weaker ECC protection codes to alleviate the static energy consumption of ECC parts of the cache ways. For the sake of better intuition, in the following we explain the FlexRel approach considering a STT-RAM L2 cache memory architecture being 16-way set-associative and having a 64 Byte (512 bit) cache line as our case study example, while in general, FlexRel approach is applicable to all associative STT-RAM caches with any configuration and at any memory abstraction level.

The first step in designing a FlexRel-equipped cache is to classify the write requests of the cache based on the HW (which shows the vulnerability of write requests) and the portion of write requests. The partitioning in FlexRel applies at way granularity. Here, as an example, we consider four protection levels in our case study FlexRel-equipped cache. The straightforward approach to assign the cache ways to one of the four protection levels is the uniform assignment (in case of 16-way set associative cache it implies to assign four ways to each protection level). Considering the efficiency challenge that was mentioned for the uniform ECC protection technique, this straightforward assignment may face a considerable waste of resources. Therefore, our approach to enhance FlexRel has been to consider once again the write request patterns depicted in Fig. 5 to configure the portion of cache ways in each protection level. For this decision, we need to consider the cumulative amount of write requests in each protection level to provide enough space for them and keep the system performance as high as possible.

Thus, we partition the 16-way FlexRel-equipped cache to four protection levels as depicted in Fig. 6. According to the figure, we assign half of the cache ways to protection level 2 ($101 \leq HW \leq 250$) which should serve the most amount of write requests. Furthermore, protection level 1 ($HW \leq 100$) which should serve the second most amount of write requests benefits from a quarter of cache ways, while each of protection levels 3 and 4 only utilizes two ways to serve their low-intensive write requests. Figure 7 depicts the proposed FlexRel scheme for our case study example which includes four zones regarding protection levels.

After the way partitioning of FlexRel-equipped cache is completed, we should determine combinations of STT-RAM write current and ECC code to deliver a reliable write operation in each zone. To select these combinations, first we should consider a write error rate threshold to meet during the write operations in FlexRel-

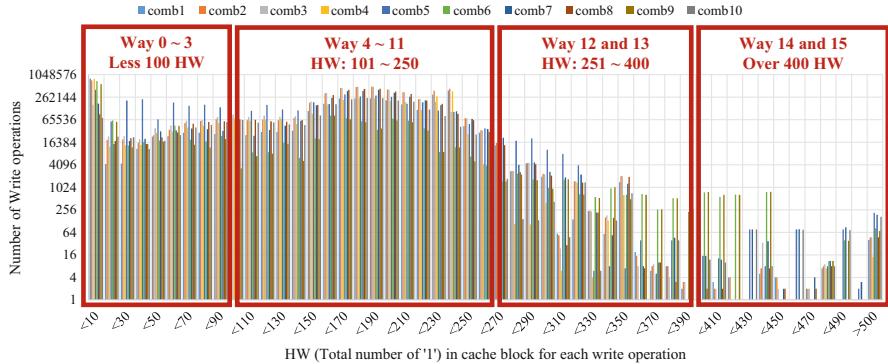


Fig. 6 Way partitioning across a 4-MB 16-way FlexRel-equipped shared L2 cache

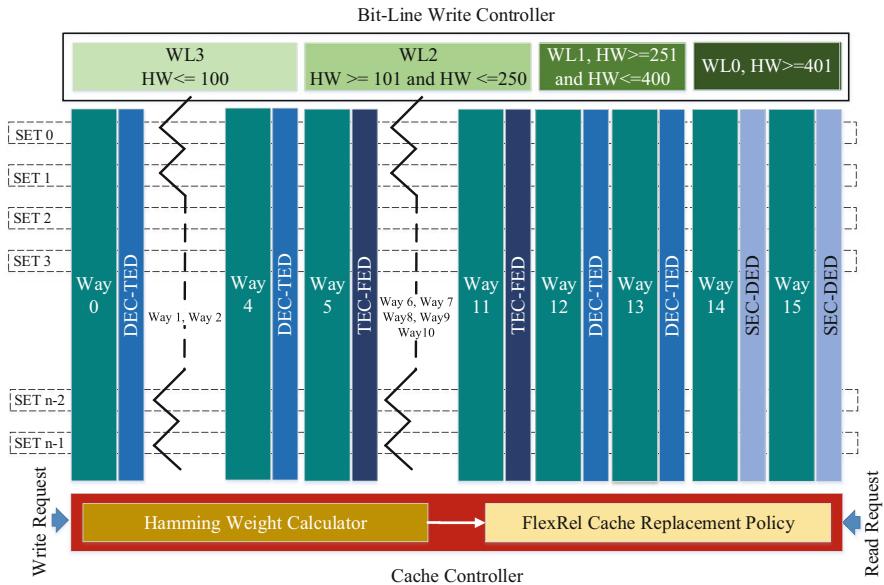


Fig. 7 A 16-way set associative FlexRel-equipped cache

equipped cache. To guarantee this threshold, FlexRel can either increase the write voltage and decrease the protection level of ECCs or vice versa. As an example, we consider 10^{-8} as write error rate threshold that should be met in all protection levels considering the write requests' HWs.

Consequently, Table 2 depicts a transducer map considered for FlexRel. As shown in the table, based on the configuration of write currents and ECC protection capabilities different dynamic energies and static powers are consumed in FlexRel-equipped cache ways. The last row of the table shows the amount of dynamic energy and static power of a uniform full-protection scheme with the same write error rate

Table 2 Transducer map for a 256 KB way embedded in a 4 MB 16-way set-associative FlexRel-equipped cache

Protection level	Write current	0 → 1 Error rate	ECC protection level (t)	Cache 0 → 1 block error rate	Way dynamic energy	Way static energy	Read	Write
				Lowest HW	Highest HW			
1	706.2 μA	3×10^{-5}	2	None	4.4×10^{-9}	67.3 pJ	4.79 nJ	50.6 mW
2	666.9 μA	8×10^{-5}	3	1.5×10^{-10}	6.4×10^{-9}	68.5 pJ	4.61 nJ	53.1 mW
3	778.2 μA	5×10^{-6}	2	3.2×10^{-10}	1.3×10^{-9}	67.5 pJ	5.27 nJ	50.6 mW
4	935.0 μA	1×10^{-7}	1	7.9×10^{-10}	1.3×10^{-9}	66.6 pJ	6.19 nJ	48.5 mW
OUP	586.0 μA	6×10^{-4}	7	None	1.42×10^{-9}	73.3 pJ	4.37 nJ	60.6 mW

The energy consumption and Error rates depicted in this table are retrieved with the aid of NVSim [8] and HSPICE Monte Carlo® simulations

threshold mentioned for FlexRel-equipped cache. It is worthy of mentioning that in each protection level, FlexRel should provide the required facilities so that the highest HW write request in that level meets write error rate thresholds. The write currents at different protection levels in Table 2 reveal the mentioned strategy in FlexRel way partitioning. For example, comparing protection level 2 (with high amounts of write operations) with protection level 4 (with low amounts of write operations), for level 2 we considered lower write current ($666.9 \mu\text{A}$) with stronger ECC code ($t = 3$) to alleviate the high write energy consumption of STT-RAMs. On the other hand, for level 4, we consider higher write current ($935.0 \mu\text{A}$) with weaker ECC code ($t = 1$) to alleviate the ECC static energy consumption.

Now that the data storage architecture of FlexRel is explored, the final element that we should consider in the architecture of FlexRel-enable cache is to design a mechanism to redirect the write request to their corresponding ways based on their HW during the execution. As depicted in Fig. 7, we developed a new replacement policy to perform this redirection.

Considering our case study example, Algorithm 1 depicts the traffic controller and replacement policy defined for FlexRel-equipped cache. This algorithm can be easily modified to apply to any other FlexRel-equipped cache with different configurations than the case study example. The FlexRel controller is responsible for calculating the HW of the incoming write request (Line 1). Then, if the write request is hit in the cache (Lines 3–10), FlexRel controller will verify the possibility of writing the new request to the hit block. To this end, FlexRel controller checks the HW of the new incoming request with the HW boundaries of the hit block's way protection level (Line 4). If the new incoming block satisfies the HW boundaries of the hit block's way protection level, FlexRel controller will satisfy the write request (Lines 5–7). Otherwise, the hit block will become invalid, and a cache miss signal will be triggered for the new incoming write request (Lines 7–10).

On the other hand, if the new incoming write request is missed in the cache (Line 11–24), based on the HW of this request that was calculated previously (Line 1), the FlexRel replacement policy should select the appropriate protection level for this request and evict a block from the protection level's assigned ways. It should be noted that FlexRel replacement policy uses LRU replacement policy in each protection level to evict the blocks (Lines 13, 16, 19, and 22).

5 Experimental Results

To explore the effectiveness of FlexRel in saving the energy consumption while meeting the reliability constraints we conducted a set of simulations. To this end, we used gem5 [4] simulating a quad-core ARM processor. The frequency of this processor is set to 1 GHz. The details of simulation configurations are summarized in Table 3. We extracted the dynamic and leakage power of STT-RAM cache ways from NVSim [8] with the aid of HSPICE. SPEC CPU2006 benchmark suites [9]

Algorithm 1: FlexRel controller and replacement policy for 16-way set associative cache

```

input : New incoming write request (WR)
output : The target block in cache for satisfying request
1 HW = calculate_HW(WR);
2 blk = Hit(WR);
   /* Check the availability of the requested block. */
3 if blk then
   /* The requested blk is found in the cache (hit). */
4   satisfy_request = check_way_boundary(blk→way,HW);
5   if satisfy_request then
6     | return(blk);
7   else
8     | Invalidate(blk);
9     | blk = null;
10    | /* After generating a miss signal for this request,
11       |   FlexRel replacement policy will decide about the new
12       |   location of this block. */
13  end
14 else
   /* The requested blk is not found in the cache (miss). */
15 if HW ≤ 100 then
16   | blk = LRU(way0 ~ way3);
17   | return(blk);
18 else if 101 ≤ HW ≤ 250 then
19   | blk = LRU(way4 ~ way11);
20   | return(blk);
21 else if 251 ≤ HW ≤ 400 then
22   | blk = LRU(way12 and way13);
23   | return(blk);
24 else
25   | blk = LRU(way14 and way15);
26 end
  
```

were used as the workloads in this study. Table 4 depicts the combination of benchmarks in each workload.

It should be noted that, for the sake of improving the accuracy of the experiments, all of the simulation results were retrieved after skipping the L2 cache warm-up phase. During the experiments, we implemented and compared the following schemes:

- **Optimized Uniform Protection**—In this scheme, L2 cache ways were protected with uniform 7EC8ED BCH code with low write current mentioned in the last row of Table 2 (OUP). This uniform protection satisfied the considered block write error rate considered in this study (i.e., 10^{-8}).

Table 3 Experimental setup for gem5 simulations

Parameter	Value
ISA	ARMv7-A
No. of cores	4
L1 \$ size, assoc.	32 KB, 4
L2 \$ size, assoc.	4 MB, 16
Cache configuration	L1 (Private) L2 (Shared, FlexRel-enabled)
Cache block size	64B
Cache warm-up instructions	100 million
No. simulated instructions	100 million

Table 4 Workload combinations

Combination	Core 0	Core 1	Core 2	Core 3
Comb1	perlbench	bzip2	mcf	soplex
Comb2	perlbench	bzip2	omnetpp	xalancbmk
Comb3	perlbench	mcf	omnetpp	xalancbmk
Comb4	bzip2	mcf	soplex	xalancbmk
Comb5	gcc	bwaves	mcf	cactusADM
Comb6	namd	dealII	soplex	calculix
Comb7	perlbench	gcc	mcf	namd
Comb8	perlbench	namd	soplex	xalancbmk
Comb9	bwaves	dealII	namd	calculix
Comb10	gcc	bwaves	soplex	xalancbmk

- **FlexRel**—In this scheme, L2 cache ways were protected with variable strength ECCs and write currents mentioned in Table 2, according to the discussed cache structure in FlexRel.

Figure 8 depicts the normalized energy consumption of FlexRel-equipped shared L2 cache compared with the optimized uniform scheme. We evaluated the efficiency of FlexRel in terms of energy consumption from three perspectives, i.e., dynamic energy consumption, static energy consumption, and overall energy consumption. According to Fig. 8, FlexRel increased the dynamic energy consumption of ways by up to 19% in *comb1* which intensively used the protection levels that consume high write energy consumption. It is worth noting that since the optimized uniform scheme utilizes the least write current, the FlexRel scheme will impose more dynamic energy. On the other hand, since FlexRel utilizes low protection ECCs in comparison with optimized uniform ECC scheme, it significantly improves the static energy consumption in almost all of the combinations. Indeed, while the leakage power of the ways in the FlexRel-equipped cache was significantly lower than the optimized uniform scheme, the high-performance overhead experienced in *comb2* (see Fig. 9) led to the same static energy consumption in both schemes, and further increased the overall energy consumption by 3%.

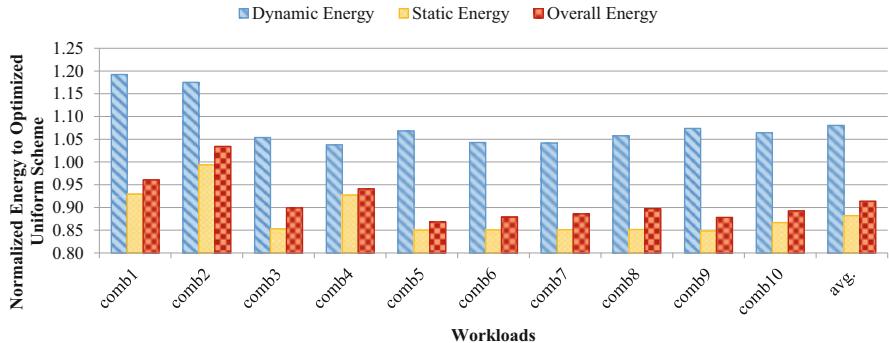


Fig. 8 Energy consumption of the ways of 4 MB shared FlexRel-equipped L2 cache normalized to optimized uniform scheme

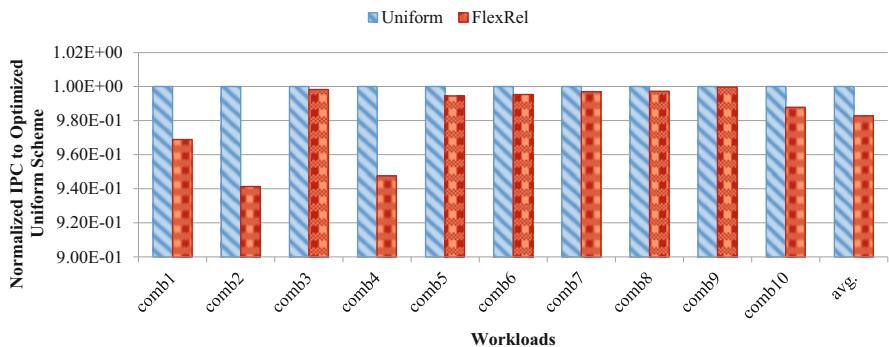


Fig. 9 The Instruction per Clock (IPC) of a system includes a 4 MB shared FlexRel-equipped L2 cache normalized to the IPC of a system that includes optimized uniform scheme in its L2 cache

In general, since static energy consumption is the main contributor in the energy consumption of the cache ways, FlexRel achieves a considerable improvement in the overall energy consumption (static + dynamic) of ways in the L2 cache. On average, while FlexRel increases the dynamic energy consumption of the ways by 8%, it saves the static energy consumption and overall energy consumption of L2 cache ways by 12% and 9%, respectively. The calculated amount of saved static power in each FlexRel-equipped cache's way is about 70.6 mW. With this amount of power, we will be able to supply the static power of more than seven 32 KB L1 caches (like the ones considered in this study) each consuming 8.9 mW static power.

Figure 9 shows the Instruction per Clock (IPC) of a system using a 4 MB shared FlexRel-equipped L2 cache normalized to the IPC of a system that incorporates the optimized uniform scheme in its L2 cache. Since FlexRel modifies the replacement policy of the L2 cache to redirect the write requests to their corresponding protection level's ways it may impose some performance penalty to the system in the situations when (1) the target protection level's ways face intensive write requests from

Table 5 The amount of way area saving in FlexRel at different protection levels compared with optimized uniform scheme

Protection Level	ECC protection level (t)	ECC check bits	% of area improvement
1	2	21	8.5
2	3	31	6.6
3	2	21	8.5
4	1	11	10.2
% of average area improvement	7.9		

different addresses or (2) we should evict a hit block because the total number of “1”s in this block forces FlexRel to change its location.

According to Fig. 9, in the worst case scenario, i.e., *comb2* that the miss penalty in FlexRel approach is considerable, the IPC of the system is degraded by 6%. This result illustrates the importance of cache partitioning in the performance of the system. In other words, while the considered partitioning provides reasonable performance for most of the workloads, for *comb1*, *comb2*, and *comb4*, the considered partitioning utilized by Algorithm 1 led to high-performance overheads for these workloads. Previously we mentioned that these performance overheads even affect the energy efficiency of FlexRel for *comb1*, *comb2*, and *comb4* workloads. On average, FlexRel decreases the IPC of the system by a negligible 1.7%.

Finally, w.r.t. area, ECC check bits assigned at each protection level are the main contributor. Accordingly, Table 5 reports ECC check-bits, and the area saving at different protection levels of FlexRel compared with the optimized uniform protection scheme. In general, considering a 16-way set associative L2 cache, the flexible scheme provided by FlexRel could save the occupied ways’ area by about 7.9%, on average.

6 Conclusions and Future Work

In this chapter, we proposed FlexRel, an energy-aware reliability improvement architectural scheme for STT-RAM cache memories. FlexRel is an architecture-to-application cross-layer approach that considers a memory architecture provided with Error Correction Codes (ECCs) and a custom current regulator for the various cache ways and conducts a trade-off between reliability and energy consumption. The FlexRel cache controller dynamically profiles the number of $0 \rightarrow 1$ bit transitions of each write operation and, based on this critical parameter it selects the most-suitable cache way and current level to deliver the necessary reliability level (in terms of occurred write errors) while minimizing the energy consumption.

The results of evaluating FlexRel show that, while the scheme satisfies the reliability requirements, it delivers up to 13.2% energy saving and up to 10.2% cache ways’ area saving, compared with the most efficient uniform protection scheme. The

performance overhead imposed by FlexRel to the system due to the modifications of cache ways' access traffics is 1.7%, on average.

As future work, we will further improve and refine the FlexRel in two aspects. First, we will focus on the proposed replacement policy to improve the performance of the system for workloads that face significant block evictions due to HW boundary violations. To minimize the performance overhead, we will attempt to devise a dynamic cache partitioning scheme capable of changing the configuration of the FlexRel-equipped cache when considerable performance degradation is observed.

References

1. Ahn, J., Choi, K.: Selectively protecting error-correcting code for area-efficient and reliable STT-RAM caches. In: Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 285–290 (2013)
2. Azad, Z., Farbeh, H., Monazzah, A.M.H., Miremadi, S.G.: An efficient protection technique for last level STT-RAM caches in multi-core processors. *IEEE Trans. Parallel Distr. Syst.* **28**(6), 1564–1577 (2017)
3. Azad, Z., Farbeh, H., Monazzah, A.M.H.: ORIENT: Organized interleaved ECCs for new STT-MRAM caches. In: Proceedings of the Design, Automation Test in Europe Conference and Exhibition (DATE), pp. 1187–1190 (2018)
4. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. *ACM SIGARCH Comput. Archit. News* **39**(2), 1–7 (2011)
5. Chen, Y., Li, H., Wang, X., Zhu, W., Xu, W., Zhang, T.: A nondestructive self-reference scheme for spin-transfer torque random access memory (STT-RAM). In: Proceedings of the Design, Automation Test in Europe Conference and Exhibition (DATE), pages 148–153, 2010.
6. Cheshmikhani, E., Farbeh, H., Asadi, H.: ROBIN: Incremental oblique interleaved ECC for reliability improvement in STT-MRAM caches. In: Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 173–178 (2019)
7. Cheshmikhani, E., Farbeh, H., Miremadi, S.G., Asadi, H.: TA-LRW: A replacement policy for error rate reduction in STT-MRAM caches. *IEEE Trans. Comput.* **68**(3), 455–470 (2019)
8. Dong, X., Xu, C., Xie, Y., Jouppi, N.: NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **31**(7), 994–1007 (2012)
9. Henning, J.L.: SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Comput. Archit. News* **34**(4), 1–17 (2006)
10. Hosomi, M., Yamagishi, H., Yamamoto, T., Bessho, K., Higo, Y., Yamane, K., Yamada, H., Shoji, M., Hachino, H., Fukumoto, C., Nagao, H., Kano, H.: A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In: Proceedings of the IEEE International Electron Devices Meeting (IEDM), pp. 459–462 (2005)
11. Monazzah, A.M.H., Farbeh, H., Miremadi, S.G.: LER: Least-error-rate replacement algorithm for emerging STT-RAM caches. *IEEE Trans. Device Mat. Rel.* **16**(2), 220–226 (2016)
12. Jin, Y., Shihab, M., Jung, M.: Area power and latency considerations of STT-MRAM to substitute for main memory. In: Proceedings of the Symposium on Computer Architecture (ISCA) (2014)

13. Kim, Y., Gupta, S., Park, S., Panagopoulos, G., Roy, K.: Write-optimized reliable design of STT MRAM. In: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), ISLPED '12, pp. 3–8 (2012)
14. Kim, J., Chen, A., Behin-Aein, B., Kumar, S., Wang, J., Kim, C.: A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies. In: Proceedings of the Custom Integrated Circuits Conference (CICC), pp. 1–4 (2015)
15. Lin, S., Costello, D.: Error Control Coding: Fundamentals and Applications. Prentice Hall, Upper Saddle River (1983)
16. Marins de Castro, M., Sousa, R.C., Bandiera, S., Ducruet, C., Chavent, A., Auffret, S., Papusoi, C., Prejbeanu, I.L., Portemont, C., Vila, L., et al.: Precessional spin-transfer switching in a magnetic tunnel junction with a synthetic antiferromagnetic perpendicular polarizer. *J. Appl. Phys.* **111**(7), 07C912 (2012)
17. Monazzah, A.M.H., Shoushtari, M., Miremadi, S.G., Rahmani, A.M., Dutt, N.: QuARK: Quality-configurable approximate STT-MRAM cache by fine-grained tuning of reliability-energy knobs. In: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), pp. 1–6 (2017)
18. Oboril, F., Shirvani, A., Tahoori, M.: Fault tolerant approximate computing using emerging non-volatile spintronic memories. In: Proceedings of the VLSI Test Symposium (VTS), pp. 1–1 (2016)
19. Rahmani, A.M., Liljeberg, P., Hemani, A., Jantsch, A., Tenhunen, H.: The Dark Side of Silicon, 1st edn. Springer, Berlin (2016)
20. Ranjan, A., Venkataramani, S., Fong, X., Roy, K., Raghunathan, A.: Approximate storage for energy efficient spintronic memories. In: Proceedings of the Design Automation Conference (DAC), pp. 1–6 (2015)
21. Shoushtari, M., Rahmani, A.M., Dutt, N.: Quality-configurable memory hierarchy through approximation: Special session. In: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), pp. 1–2 (2017)
22. Sun, H., Liu, C., Zheng, N., Min, T., Zhang, T.: Design techniques to improve the device write margin for MRAM-based cache memory. In: Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), pp. 97–102 (2011)
23. Swami, S., Mohanram, K.: Reliable nonvolatile memories: techniques and measures. *IEEE Design Test* **34**(3), 31–41 (2017)
24. Teimoori, M.T., Hanif, M.A., Ejlali, A., Shafique, M.: Adam: adaptive approximation management for the non-volatile memory hierarchies. In: Proceedings of the Design, Automation Test in Europe Conference and Exhibition (DATE), pp. 785–790 (2018)
25. Wang, X., Mao, M., Eken, E., Wen, W., Li, H., Chen, Y.: Sliding basket: An adaptive ECC scheme for runtime write failure suppression of STT-RAM cache. In: Proceedings of the Design, Automation Test in Europe Conference and Exhibition (DATE), pp. 762–767 (2016)
26. Wen, W., Mao, M., Zhu, X., Kang, S., Wang, D., Chen, Y.: CD-ECC: Content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD), pp. 1–8. IEEE, New York (2013)
27. Xu, W., Sun, H., Wang, X., Chen, Y., Zhang, T.: Design of Last-level on-chip cache using spin-torque transfer RAM (STT RAM). *IEEE Trans. Very Large Scale Integr. Syst.* **19**(3), 483–493 (2011)
28. Zhang, Y., Wang, X., Li, Y., Jones, A.K., Chen, Y.: Asymmetry of MTJ switching and its implication to STT-RAM designs. In: Proceedings of the Design, Automation Test in Europe Conference and Exhibition (DATE), pp. 1313–1318 (2012)
29. Zhang, Y., Zhang, L., Chen, Y.: MLC STT-RAM design considering probabilistic and asymmetric MTJ switching. In: Proceedings on International Symposium on Circuits and Systems (ISCAS), pp. 113–116 (2013)
30. Zhang, Y., Li, Y., Sun, Z., Li, H., Chen, Y., Jones, A.K.: Read performance: The newest barrier in scaled STT-RAM. *IEEE Trans. Very Large Scale Integr. Syst.* **23**(6), 1170–1174 (2015)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Hardware/Software Codesign for Energy Efficiency and Robustness: From Error-Tolerant Computing to Approximate Computing



Abbas Rahimi and Rajesh K. Gupta

1 Introduction

Let us step back and first look at an ideal hardware where the entire software stack can be executed. In reality, however, the hardware underneath of computing is being challenged as CMOS scaling continues to nanometer dimensions [14, 41]. The hardware experiences different sources of variability over time, or across different parts (see Fig. 1a). These variations include: manufacturing process variability that causes static variations in critical dimension, channel length, and threshold voltage of devices [6]; temporal aging/wear out variability that causes slow degradation in devices [22]; and finally, dynamic variability in ambient condition that is caused by fluctuations in operating temperature and supply voltage [8, 25]. The way that designers typically combat with these sources of variability is to consider worst-case design by imposing a large margin in hardware to ensure the correct execution of the software stack. This conservative margin leads to a loss of energy efficiency. Further, the ever-increasing amount of variability [15] limits how far we can drive down the energy per operation (i.e., voltage scaling). This means that we cannot reduce the energy as we used to.

What if we reduce the excessive margin to enable better energy scaling? The direct manifestation of reducing margin is a timing error as shown in Fig. 1b. A timing error means capturing an invalid value to a storage element like a flip-flop or a memory cell, so the result of computation might become wrong. Instead of blindly dealing with variability and its resulting timing errors, we propose to expose them to

A. Rahimi (✉)
ETH Zurich, Zurich, Switzerland
e-mail: abbas@iis.ee.ethz.ch

R. K. Gupta
University of California San Diego, La Jolla, CA, USA
e-mail: gupta@cs.ucsd.edu

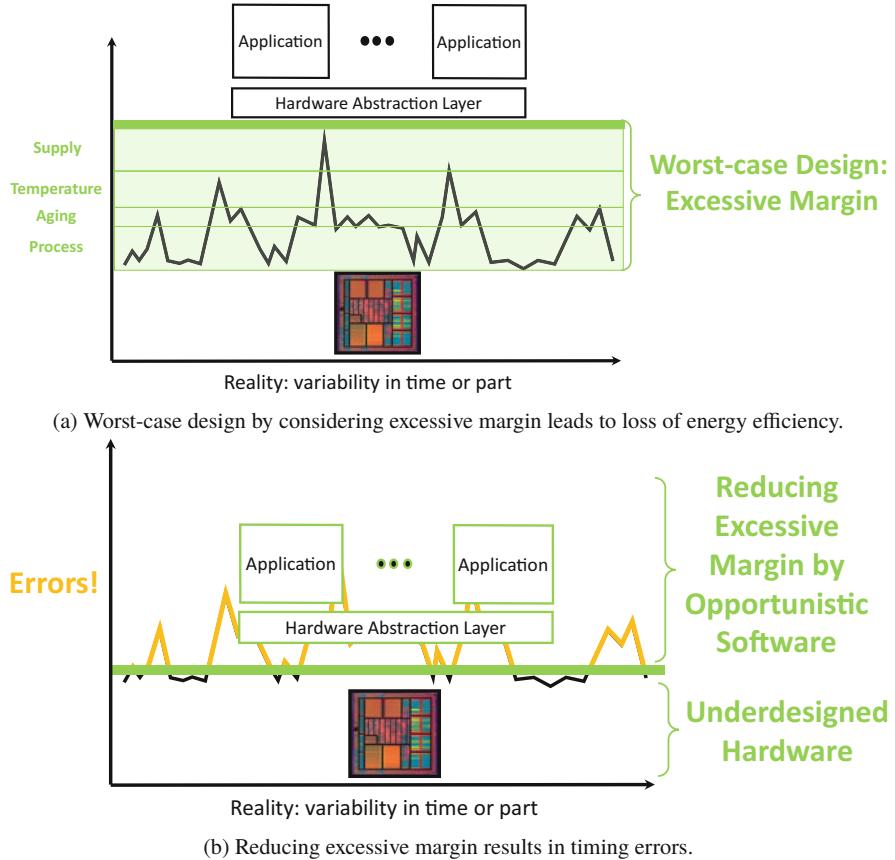
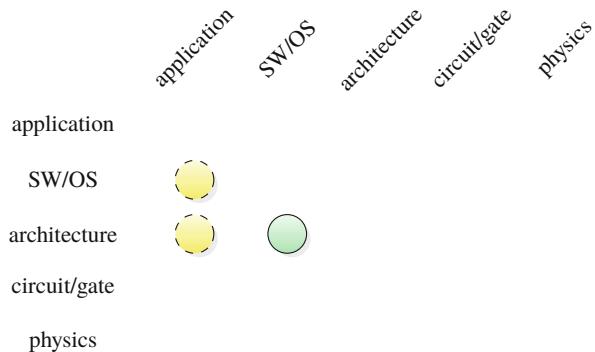


Fig. 1 Overdesigned (i.e., worst-case) hardware vs. underdesigned hardware and their interactions with software stack. **(a)** Worst-case design by considering excessive margin leads to loss of energy efficiency. **(b)** Reducing excessive margin results in timing errors

the higher levels in the stack where their side effects can be mitigated [27]. Essentially, we develop an opportunistic software layer that can operate with reduced margins and sense variability in *underdesigned* hardware—instead of overdesigned hardware with positive margins. The software layer accordingly performs adaptation by means of metadata mechanisms that reflect the state of hardware and variability, then the software can perform introspection and adaptation. The main continuations of this chapter lie on the application, software, and architectural layers as illustrated in Fig. 2.

Fig. 2 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



1.1 Clockwise Y-Chart: From Positive Margin, to Zero Margin, to Negative Margin

In the following, we discuss about the possible approaches to reduce margin and handle timing errors. The tree possible approaches are conceptualized in a Y-chart shown in Fig. 3.

This first approach is to predict and prevent the errors by keeping a positive margin. Hence, we try to reduce the excessive margin but it is still positive to ensure the error-free execution of software. In this direction, our work spans defining and measuring the notion of error tolerance, from instruction set architecture (ISA) to procedures, and to parallel programs. These measures essentially capture the likelihood of errors and associated cost of error correction at different levels. We first characterize the manifestations of variability in ISA [28] that is the finest granularity to represent a processor functionality. Then, we characterize a sequence of instructions where the timing errors can be eliminated [31]. Going higher in the software stack, we schedule different procedure calls in multi-core architecture [29] and finally a large number of kernels on massively parallel cores [32] such that there are no timing errors. At the boundary of hardware/software, we focus on adaptive compilation methods to reduce the side effects of aging and increase lifetime for massively parallel integrated architectures like GPUs [30].

What is the next approach? The next approach is about detecting and correction errors by reducing the margin to zero (i.e., operating at the edge of errors). Basically, we reduce the margin to zero such that the errors can occur. Hence, we first need to detect the errors by means of circuit sensors [7, 11, 46] and then take actions to correct them. Our focus was instead on reducing the cost of error correction in software. In this direction, we focus on variability-aware runtime environment to cover various embedded parallel workloads in OpenMP including tasks [34, 38], parallel sections, and loops [37].

Finally, the third approach is about accepting errors—i.e., approximate computing—by pushing the margin to negative. This means that the errors and approximations are becoming acceptable as long as the outcomes have a well-

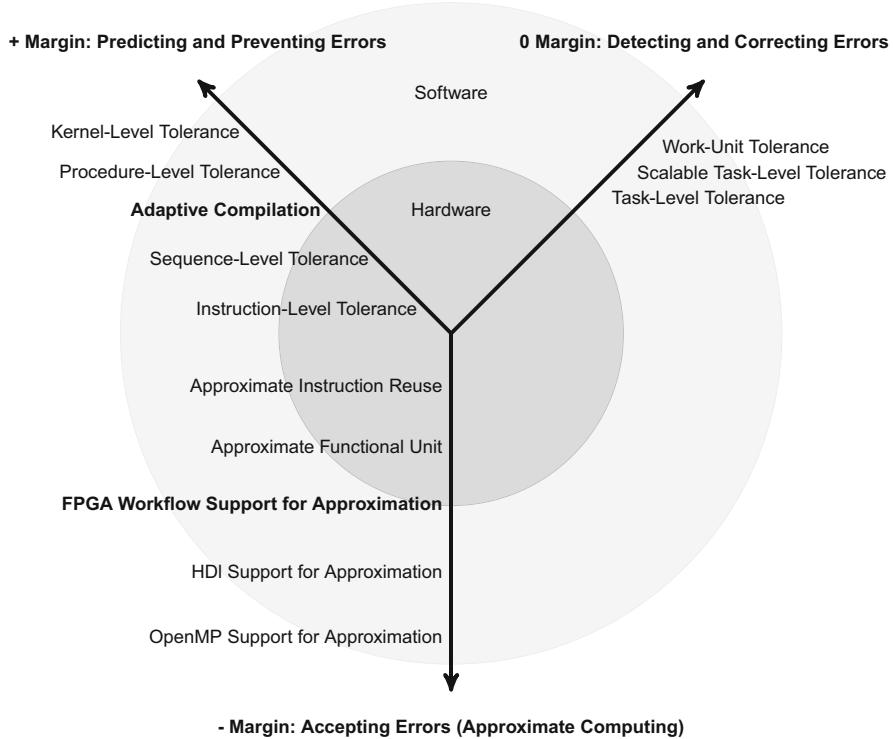


Fig. 3 Taxonomy of error tolerance in a clockwise Y-Chart: from positive margin, to zero margin, to finally negative margin

defined statistical behavior. In this approach, fatal errors can be avoided at the cost of benign approximation that can in fact allow for improving throughput and energy efficiency. Toward this goal, we enable approximate computing in instructions [33, 36, 39, 40], functional units [16–19], runtime execution environments [35], and ultimately hardware description languages [47], and high-level synthesis [23].

By looking at the Y-chart clockwise, we go from preventing errors, to correcting errors, and finally to accepting errors. This move also changes the margin from positive, to zero, and eventually to negative, leading to higher energy efficiency. In the rest of this chapter, we only focus on two methods describing how cooperative hardware/software techniques can improve energy efficiency and robustness in the presence of variability. These two methods, highlighted with bold in the Y-chart, cover examples from approaches with positive margin (i.e., predicting and preventing errors in Sect. 2) and negative margin (i.e., accepting errors in Sect. 3). Interested readers can refer to [42] for reading more about approaches with zero margin (i.e., detecting and correcting errors).

2 Positive Margin Example: Mitigating Aging in GPUs by Adaptive Compilation

In this section, we demonstrate a prime example of software that can respond to hardware variations due to aging. The goal is to reduce the excessive margin due to device aging in a setting where margin is still positive to guarantee correct execution. The idea is to combine hardware sensing circuits and adaptive software (an aging-aware compiler) to manage the workload stress. One major aging mechanism is negative bias temperature instability (NBTI) that adversely affects the reliability of a processing element by introducing new delay-induced faults. However, the effect of these delay variations is not uniformly spread across processing elements within a chip: some are affected more—hence less reliable—than others. We propose an NBTI-aware compiler-directed very long instruction word (VLIW) assignment that uniformly distributes the stress of instructions among available processing elements, with the aim of minimizing aging without any performance penalty [30]. The compiler matches the measured aging degradation with the distribution of instructions to equalize the expected lifetime of each processing element.

The rest of this chapter is organized as follows: Section 2.1 covers an overview of NBTI-induced performance degradation. Section 2.2 describes a GPU architecture and its workload distribution used in this study. Finally, our adaptive compiler is presented in Sect. 2.3.

2.1 NBTI Degradation

Among various aging mechanisms in hardware, the generation of interface traps under NBTI in PMOS transistors has become a critical issue in determining the lifetime of CMOS devices [10]. NBTI manifests itself as an increase in the PMOS transistor voltage threshold (V_{th}) that causes delay-induced failures (see Fig. 4 (left)). NBTI is best captured by the reaction–diffusion model [26]. This model describes NBTI in two stress and recovery phases. NBTI occurs due to the generation of the traps at the Si–SiO₂ interface when the PMOS transistor is negatively biased (i.e., during the stress phase). As a result, V_{th} of the transistor increases which in turn slows down the device. Removing stress from the PMOS transistor can eliminate some of the traps which partially recover the V_{th} shift. This is also known as the recovery phase. The work in [5] derived a long-term cycle-to-cycle model of NBTI. NBTI effects can be significant: its impact on circuit delay is about 15% on a 65 nm technology node and it gets worse in sub-65 nm nodes [4]. Further, NBTI-induced performance degradation is typically non-uniform which is a major concern for many-core GPUs, e.g., with up to 320 five-way VLIW processing elements [1].

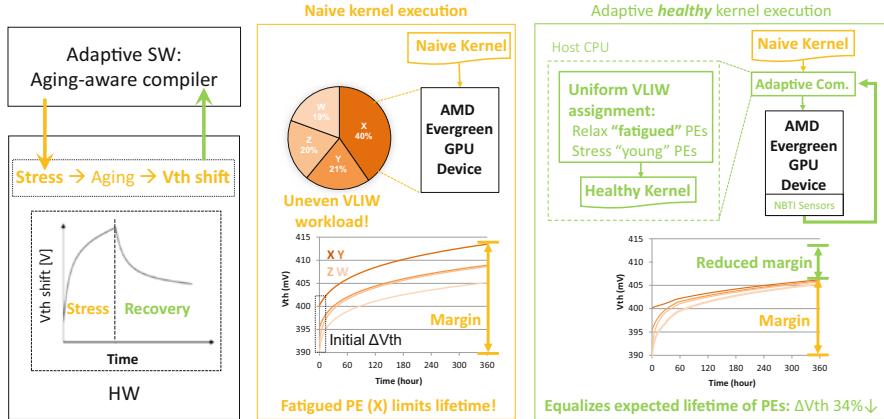


Fig. 4 Adaptive compiler to mitigating aging in GPUs: (left) sensing NBTI degradation; (middle) naive kernel execution and its impact on degradation of processing elements; (right) adaptive compiler and healthy kernel execution

2.2 GPU Architecture and Workload

We focus on the evergreen family of AMD GPUs (a.k.a. Radeon HD 5000 series), designed to target not only graphics applications but also general-purpose data-intensive applications. The Radeon HD 5870 GPU compute device consists of 20 compute units (CUs), a global front-end ultra-thread dispatcher, and a crossbar to connect the global memory to the L1-caches. Every CU contains a set of 16 stream cores. Finally, each stream core contains five processing elements (PEs), labeled X, Y, Z, W, and T constituting a VLIW processor to execute machine instructions in a vector-like fashion. The five-way VLIW processor capable of issuing up to five floating point scalar operations from a single VLIW consists primarily of five slots ($\text{slot}_X, \text{slot}_Y, \text{slot}_Z, \text{slot}_W, \text{slot}_T$). Each slot is related to its corresponding PE. Four PEs (X, Y, Z, W) can perform up to four single-precision operations separately and perform two double-precision operations together, while the remaining one (T) has a special function unit for transcendental operations. In each clock cycle, VLIW slots supply a bundle of data-independent instructions to be assigned to the related PEs for simultaneous execution. In an n -way VLIW processor, up to n data-independent instructions, available on n slots, can be assigned to the corresponding PEs and be executed simultaneously. Typically, this is not done in practice because the compiler may fail to find sufficient instruction-level parallelism to generate complete VLIW instructions. On average, if m out of n slots are filled during an execution, we call the achieved packing ratio is m/n . The actual performance of a program running on a VLIW processor largely depends on the packing ratio.

2.2.1 GPU Workload Distribution

Here, we analyze the workload distribution on the Radeon HD GPUs at architecture level, where there are many PEs to carry out computations. As it is mentioned earlier, the NBBI-induced degradation strongly depends on the resource utilization, which depends on the execution characteristics of the workload. Thus, it is essential to analyze how often the PEs are exercised during the execution of the workload. To this end, we first monitor the utilization of various CUs (inter-CU) and then the utilization of PEs within a CU (intra-CU).

To examine the inter-CU workload variation, the total number of executed instructions by each CU is collected during a kernel execution. We observe that the CUs execute almost equal number of instructions, and there is a negligible workload variation among them. We have configured six compute devices with different number of CUs, {2, 4,..., 64}, to finely examine the effect of the workload variation on a variety of GPU architectures.¹ For instance, during DCT kernel execution, the workload variation between CUs ranges from 0% to 0.26% depending on the number of physical CUs on the compute device. Execution of a large number of different kernels confirms that the inter-CU workload variation is less than 3%, when running on the device with 20 CUs (i.e., HD 5870). This nearly uniform inter-CU workload distribution is accomplished by load balancing and uniform resource arbitration algorithms of dispatcher in the GPU architecture.

Next, we examine the workload distribution among the PEs. Figure 4 (middle) shows the percentage of the executed instructions by various PEs during execution of kernels. We only consider four PEs (PE_X , PE_Y , PE_Z , PE_W) which are identical in their functions [1]; they differ only in the vector elements to which they write their result at the end of the VLIW. As shown, the instructions are not uniformly distributed among PEs. For instance, the PE_X executes 40% of ALU instructions, while the PE_W executes only 19% of the instructions. This non-uniform workload variation causes non-uniform aging among PEs. In other words, some PEs are exhausted more than other and thus have shorter lifetime as shown in Fig. 4 (middle). Unfortunately, this non-uniformity happens within all CUs since their workload is highly correlated together. Therefore, no PE throughout the entire compute device is immune from this unbalanced utilization.

The root cause of non-uniform aging among PEs is the frequent and non-uniform execution of VLIW slots. In other words, higher utilization of PE_X implies that slot_X of VLIW is occupied more frequently than the other slots. This substantiates that the compiler does not uniformly assign the independent instructions to various VLIW slots, mainly because the compiler only employs optimization for increasing the packing ratio through finding more parallelism to fully pack the VLIW slots. The VLIW processors are designed to give the compiler tight control over program

¹The latest Radeon HD 5000 series, HD 5970, has 40 CUs featuring 4.3 billion transistors in 40 nm technology.

execution; however, the flexibility afforded by such compilers, for instance, to tune the order of instructions packing, can be used towards reliability improvement.

2.3 Adaptive Aging-Aware Compiler

The key idea of an aging-aware compilation is to assign independent instructions uniformly to all slots: idling a fatigued PE and reassigning its instructions to a young PE through swapping the corresponding slots during the VLIW bundle code generation. This basically exposes the inherent idleness in VLIW slots and guides its distribution that does matter for aging. Thus, the job of dynamic binary optimizer, for k -independent instructions, is to find k -young slots, representing k -young PEs, among all available n slots, and then assign instructions to those slots. Therefore, the generated code is a “healthy” code that balances workload distribution through various slots maximizing the lifetime of all PEs (see Fig. 4 (right)). Here, we briefly describe how these statistics can be obtained from silicon, and how the compiler can predict and thus control the non-uniform aging. The adaptation flow includes four steps: (1) aging sensor readout; (2) kernel disassembly, static code analysis, and calibration of predictions; (3) uniform slot assignment; (4) healthy code generation. We explain them in the following.

The compiler first needs to access the current aging data (ΔV_{th}) of PEs to be able to adapt the code accordingly. The ΔV_{th} is caused by the temporal degradation due to NBTI and/or the intrinsic process variation, thus PEs even during the early life of a chip might have different aging. Employing the compact per-PE NBTI sensors [44, 45] which provide ΔV_{th} measurement with 3σ accuracy of 1.23 mV for a wide range of temperature enables large scale data-collection across all PEs. The performance degradation of every PE can be reliably reported by a per-PE NBTI sensor, thanks to the small overhead of these sensors. The sensors support digital frequency outputs that are accessed through memory-mapped I/O by the compiler in arbitrary epochs of the post-silicon measurement. After sensor readouts, the compiler estimates the degradation of PEs using the NBTI models. In addition to the current aging data, the compiler needs to have an estimate regarding the impact of future workload stress on the various PEs. Hence, a just-in-time disassembler disassembles a naive kernel binary to a device-dependent assembly code in which the assignment of instructions to the various slots (corresponding PEs) are explicitly defined and are thus observable by the compiler. Then, a static code analysis technique is applied that estimates the percentage of instructions that will be carried out on every PE in a static sense. It extracts the future stress profile, and thus the utilization of various PEs using the device-dependent assembly code. If the predicted stress of a PE is overestimated or underestimated, mainly due to the static analysis of the branch conditions of the kernel’s assembly code, a linear calibration module fits the predicted stress to the observed stress, in the next adaptation period.

Thus far, we have described how the compiler evaluates the current performance degradation (aging) of every PE and their future performance degradation due to

the naive kernel execution. Then, the compiler uses this information to perform code transformations with the goal of improving reliability, without any penalty in the throughput of code execution (maintaining the same parallelism). To minimize stress, the compiler sorts the predicted performance degradation of the slots increasingly and the aging of the slots decreasingly and then applies a permutation to assign fewer/more instructions to higher/lower stressed slots. This algorithm is applied for every adaptation period. As a result of the slot reallocation, the minimum/maximum number of instructions is assigned to the highest/lowest stressed slot for the future kernel execution. This reduces ΔV_{th} shifts by 34%, thus uniforming the lifetime of PEs and allowing for reducing the positive margin as shown in Fig. 4 (right).

Execution of all examined kernels shows that the average packing ratio is 0.3 which means there is a large fraction of empty slots in which PEs can be relaxed during kernels execution. This low packing ratio is mainly due to the limitation of instruction-level parallelism. The proposed adaptive compilation approach superposes on top of all optimization performed by a naive compiler and does not incur any performance penalty since it only reallocates the VLIW slots (slips the scheduled instructions from one slot to another) within the same scheduling and order determined by the naive compiler. In other words, our compiler guarantees the iso-throughput execution of the healthy kernel. It also runs fully in parallel with GPU on a host CPU, thus there will be no penalty for GPU kernel execution if dynamic compilation of one kernel can be overlapped with the execution of another kernel. You can refer to [49] for further details.

3 Negative Margin Example: Enabling Approximate Computing in FPGA Workflow

Modern applications including graphics, multimedia, web search, and data analytics exhibit significant degrees of tolerance to imprecise computation. This amenability to approximation provides an opportunity to reduce the excessive margin, namely to negative, by accepting errors that trade the quality of the results for higher efficiency. Approximate computing is a promising avenue that leverages such tolerance of applications to errors [12, 13, 20, 21, 24, 47]. However, there is a lack of techniques that exploits this opportunity in FPGAs.

In [23], we aim to bridge the gap between approximation and the FPGA acceleration through an automated design workflow. Exploiting this opportunity is particularly important for FPGA accelerators that are inherently subject to many resource constraints. To better utilize the FPGA resources, we devise an automated design workflow for FPGAs [23] that leverages imprecise computation to increase data-level parallelism and achieve higher computational throughput. The core of our workflow is a source-to-source compiler that takes in an input kernel and applies a novel optimization technique that selectively reduces the precision of the kernel data and operations. By selectively reducing the precision of the data and

operation (analogous to setting margin to negative), the required area to synthesize the kernel on the FPGA decreases allowing to integrate a larger number of operations and *parallel* kernels in the fixed area of the FPGA (i.e., improving energy efficiency per unit of area). The larger number of integrated kernels provides more hardware context to better exploit data-level parallelism in the target applications. To effectively explore the possible design space of approximate kernels, we exploit a genetic algorithm to find a subset of safe-to-approximate operations and data elements and then tune their precision levels until the desired output quality is achieved. Our method exploits a fully software technique and does not require any changes to the underlying FPGA hardware. We evaluate it on a diverse set of data-intensive OpenCL benchmarks from the AMD accelerated parallel processing (APP) SDK v2.9 [3]. We later describe OpenCL execution model and its mapping on FPGA in Section 3.1. The synthesis result on a Stratix V Altera FPGA shows that our approximation workflow yields $1.4 \times - 3.0 \times$ higher throughput with less than 1% quality loss (see Sect. 3.2).

3.1 OpenCL Execution Model and Mapping on FPGAs

Altera and Xilinx recently offer high-level acceleration frameworks for OpenCL [2, 43], hence we target acceleration of data-intensive computational OpenCL applications. The challenge is however devising a workflow that can be plugged into the existing toolsets and can automatically identify the opportunities for approximation while keeping the quality loss reasonably low. OpenCL is a platform-independent framework for writing programs that execute across a heterogeneous system consisting of multiple compute devices including CPUs or accelerators such as GPUs, DSPs, and FPGAs. OpenCL uses a subset of ISO C99 with added extensions for supporting data and task-based parallel programming models. The programming model in OpenCL comprises of one or more device kernel codes in tandem with the host code. The host code typically runs on a CPU and launches kernels on other compute devices like the GPUs, DSPs, and/or FPGAs through API calls. The instance of an OpenCL kernel is called a work-item. These kernels execute on compute devices that are a set of compute units (CUs), each comprising of multiple PEs having ALUs. The work-items execute on a single PE and exercise the ALU.

The Altera OpenCL SDK [2] allows programmers to use high-level OpenCL kernels, written for GPUs, to generate an FPGA design with higher performance per Watt [9]. In this work, an OpenCL kernel is first compiled and then synthesized as a special dedicated hardware for mapping on an FPGA. FPGAs can further improve the performance benefits by creating multiple copies of the kernel pipelines (synthesized version of an OpenCL kernel). For instance, this replication process can make n copies of the kernel pipeline. As the kernel pipelines can be executed independently from one another, the performance would scale linearly with the number of copies created owing to the data-level parallelism model supported by OpenCL.

In the following, we describe how our method can reduce the amount of resources for a kernel pipeline to save area and exploit remaining area resources to boost performance by replication. Our method systematically reduces the precision of data and operations in OpenCL kernels to shrink the resources used per kernel pipeline by transforming complex kernels to simple kernels that produce *approximate* results.

3.2 Source-to-Source Compiler

We provide a source-to-source compiler to generate approximate kernels from OpenCL kernels with exact specification as shown in Fig. 5. This transformation automatically detects and simplifies parts of the kernel code that can be executed with reduced precision while preserving the desired quality-of-result. To achieve this goal, our compiler takes in as inputs, an *exact* OpenCL kernel, a set of input test cases, and a metric for measuring the quality-of-result target. The compiler investigates the exact kernel code and detects data elements, i.e., OpenCL kernel variables, that provide possible opportunities for increased performance in exchange for accuracy. It then automatically generates a population of approximate kernels by

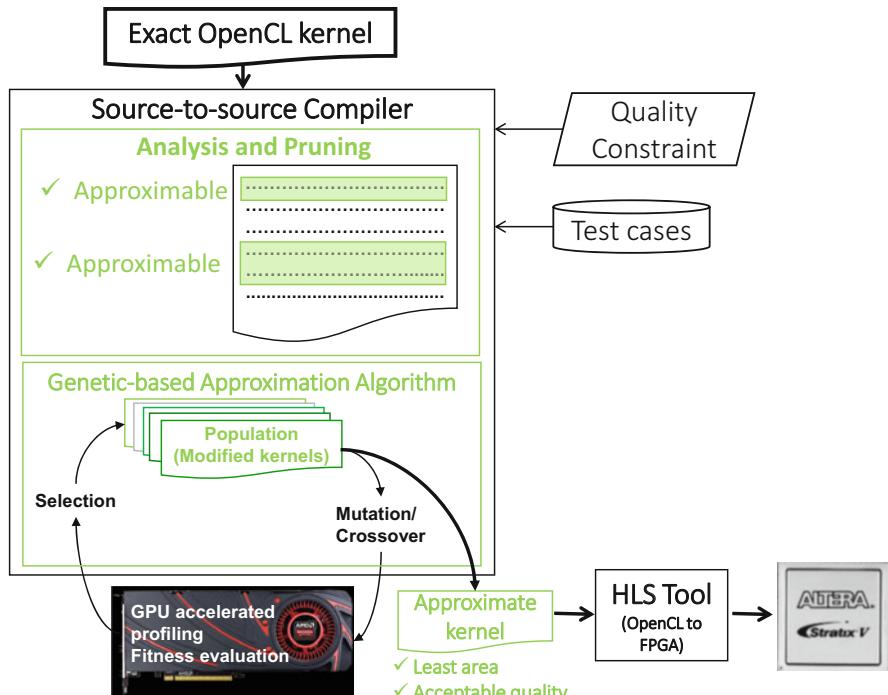


Fig. 5 FPGA approximation design workflow

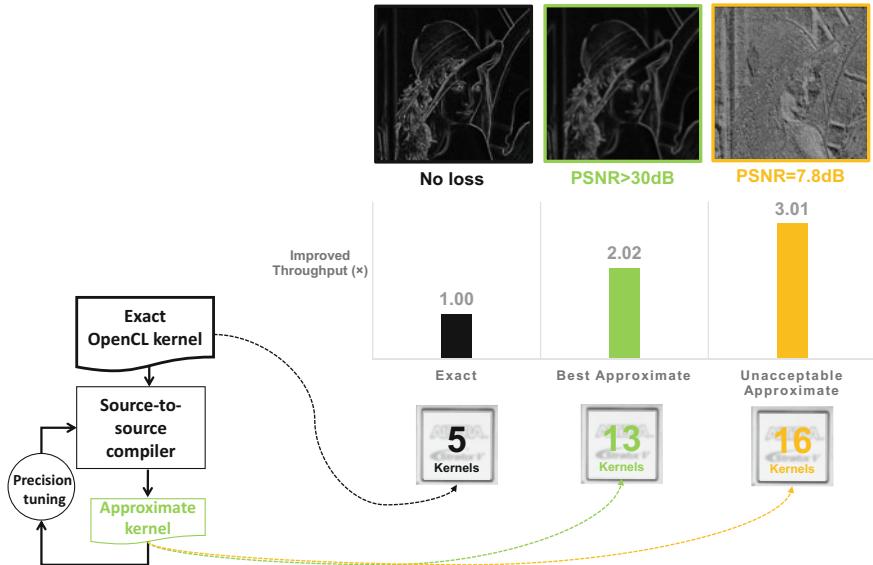


Fig. 6 Example of mapping exact and approximate kernels of Sobel filter on FPGA

means of a genetic algorithm. It can select the approximate kernels that produce acceptable results with the help of GPU profiling. These approximate kernels provide improved performance benefits by reducing the area when implemented on the FPGAs. The compiler finally outputs an optimized approximate kernel with the least area whose output quality satisfies the quality-of-result target.

The compiler uses the precision of the operations and data to tune performance as a trade-off against precision. The transformation investigates a set of kernels where in each version, some of these potential variables are replaced with a less accurate variable. To avoid a huge design space exploration, we devise an algorithm that first detects those variables that are amenable to approximation and then applies a genetic algorithm to approximate the kernel. We discuss the details of our algorithm in [23].

Figure 6 shows an example of Sobel filter kernel that is optimized by our compiler. Naive mapping of exact Sobel kernel allows us to map five instances of the kernel on the FPGA. However, by using the approximate version of kernel, we can map 13 instances of kernel on the same FPGA roughly improving throughput by 2 \times , thanks to the data-level parallel execution of the kernel, while meeting the quality constraint. We set the quality loss target to a maximum of 0.7% for image processing applications (which is equivalent to PSNR of a minimum 30 dB) and 1% for other applications which is conservatively aligned with other work on quality trade-offs [20, 21, 24, 48]. Benchmarking five kernels from OpenCL AMD APP SDK v2.9 shows that our compiler integrates a larger number of parallel kernels on the same FPGA fabric that leads to 1.4 \times –3.0 \times higher throughput on

a modern Altera FPGA with less than 1% loss of quality. This is a prime example of accepting *disciplined* errors, in the context of approximate computing, for improved throughput. The approach can be generalized to any controllable error caused by various sources. Further details are provided in [23].

4 Conclusion

Microelectronic variability is a phenomenon at the intersection of microelectronic scaling, semiconductor manufacturing, and how electronic systems are designed and deployed. To address this variability, designers resort to margins. We show how such excessive margins can be reduced, and their effects can be mitigated, by a synergy between hardware and software leading to efficient and robust microelectronic circuit and systems.

We first explore approaches to reduce the margin and enable better than worst-case design while avoiding the errors. We demonstrate its effectiveness on GPUs where the effect of variations is not uniformly spread across over thousands processing elements. Hence, we devise an adaptive compiler that equalizes the expected lifetime of each processing element by regenerating an aging-aware healthy kernel. Such new kernel guides its workload distribution that does matter for the aging, hence effectively responding to the specific health state of GPUs.

Next, we focus on approaches that significantly reduce the margins by accepting errors and exploiting approximation opportunities in computation. We explore purely software transformation methods to unleash untapped capabilities of the contemporary fabrics for exploiting approximate computing. Exploiting this opportunity is particularly important for FPGA accelerators that are inherently subject to many resource constraints. To better utilize the FPGA resources, we develop an automated design workflow for FPGA accelerators that leverages approximate computation to increase data-level parallelism and achieve higher computational throughput.

References

1. Advanced Micro Devices, Inc: AMD Evergreen Family Instruction Set Architecture
2. Altera sdk for opencl: <http://www.altera.com/products/software/opencl/opencl-index.html>
3. Amd app sdk v2.9: <http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/>
4. Bernstein, K., Frank, D., Gattiker, A., Haensch, W., Ji, B., Nassif, S., Nowak, E., Pearson, D., Rohrer, N.: High-performance cmos variability in the 65-nm regime and beyond. *IBM J. Res. Dev.* **50**(4.5), 433–449 (2006). <https://doi.org/10.1147/rd.504.0433>
5. Bhardwaj, S., Wang, W., Vattikonda, R., Cao, Y., Vrudhula, S.: Predictive modeling of the nbt effect for reliable design. In: Custom Integrated Circuits Conference, 2006, CICC '06, pp. 189–192. IEEE (2006). <https://doi.org/10.1109/CICC.2006.320885>

6. Bowman, K., Duvall, S., Meindl, J.: Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution. In: Solid-State Circuits Conference, 2001. Digest of Technical Papers, ISSCC, 2001, pp. 278–279. IEEE International (2001). <https://doi.org/10.1109/ISSCC.2001.912637>
7. Bowman, K., Tschanz, J., Kim, N.S., Lee, J., Wilkerson, C., Lu, S., Karnik, T., De, V.: Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE J. Solid State Circuits* **44**(1), 49–63 (2009). <https://doi.org/10.1109/JSSC.2008.2007148>
8. Bowman, K., Tokunaga, C., Tschanz, J., Raychowdhury, A., Khellah, M., Geuskens, B., Lu, S.L., Aseron, P., Karnik, T., De, V.: Dynamic variation monitor for measuring the impact of voltage droops on microprocessor clock frequency. In: Custom Integrated Circuits Conference (CICC), 2010, pp. 1–4. IEEE (2010). <https://doi.org/10.1109/CICC.2010.5617415>
9. Chen, D., Singh, D.: Invited paper: Using openc1 to evaluate the efficiency of cpus, gpus and fpgas for information filtering. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012, pp. 5–12. <https://doi.org/10.1109/FPL.2012.6339171>
10. Chen, G., Li, M.F., Ang, C., Zheng, J., Kwong, D.L.: Dynamic nbt1 of p-mos transistors and its impact on mosfet scaling. *IEEE Electron Device Lett.* **23**(12), 734–736 (2002). <https://doi.org/10.1109/LED.2002.805750>
11. Drake, A., Senger, R., Deogun, H., Carpenter, G., Ghiasi, S., Nguyen, T., James, N., Floyd, M., Pokala, V.: A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In: Solid-State Circuits Conference, 2007, ISSCC 2007. Digest of Technical Papers, pp. 398–399. IEEE International (2007). <https://doi.org/10.1109/ISSCC.2007.373462>
12. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Architecture support for disciplined approximate programming. In: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVII, pp. 301–312. ACM, New York, NY, USA (2012). <https://doi.org/10.1145/2150976.2151008>. <http://doi.acm.org/10.1145/2150976.2151008>
13. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-45, pp. 449–460. IEEE Computer Society, Washington, DC, USA (2012). <https://doi.org/10.1109/MICRO.2012.48>
14. Ghosh, S., Roy, K.: Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proc. IEEE* **98**(10), 1718–1751 (2010). <https://doi.org/10.1109/JPROC.2010.2057230>
15. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
16. Jiao, X., Rahimi, A., Narayanaswamy, B., Fatemi, H., de Gyvez, J.P., Gupta, R.K.: Supervised learning based model for predicting variability-induced timing errors. In: 2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS), pp. 1–4 (2015). <https://doi.org/10.1109/NEWCAS.2015.7182029>
17. Jiao, X., Jiang, Y., Rahimi, A., Gupta, R.K.: Wild: A workload-based learning model to predict dynamic delay of functional units. In: 2016 IEEE 34th International Conference on Computer Design (ICCD), pp. 185–192 (2016). <https://doi.org/10.1109/ICCD.2016.7753279>
18. Jiao, X., Jiang, Y., Rahimi, A., Gupta, R.K.: Slot: A supervised learning model to predict dynamic timing errors of functional units. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2012 (2012)
19. Jiao, X., Rahimi, A., Jiang, Y., Wang, J., Fatemi, H., de Gyvez, J.P., Gupta, R.K.: Clim: A cross-level workload-aware timing error prediction model for functional units. *IEEE Trans. Comput.* **67**(6), 771–783 (2018). <https://doi.org/10.1109/TC.2017.2783333>
20. Kahng, A., Kang, S.: Accuracy-configurable adder for approximate arithmetic designs. In: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, pp. 820–825 (2012)

21. Kulkarni, P., Gupta, P., Ercegovac, M.: Trading accuracy for power with an underdesigned multiplier architecture. In: 2011 24th International Conference on VLSI Design (VLSI Design), pp. 346–351 (2011). <https://doi.org/10.1109/VLSID.2011.51>
22. Li, X., Qin, J., Bernstein, J.: Compact modeling of mosfet wearout mechanisms for circuit-reliability simulation. *IEEE Trans. Device Mater. Reliab.* **8**(1), 98–121 (2008). <https://doi.org/10.1109/TDMR.2008.915629>
23. Lotfi, A., Rahimi, A., Yazdanbakhsh, A., Esmaeilzadeh, H., Gupta, R.K.: Grater: An approximation workflow for exploiting data-level parallelism in fpga acceleration. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1279–1284 (2016)
24. Moreau, T., Wyse, M., Nelson, J., Sampson, A., Esmaeilzadeh, H., Ceze, L., Oskin, M.: Snnap: Approximate computing on programmable socs via neural acceleration. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 603–614 (2015). <https://doi.org/10.1109/HPCA.2015.7056066>
25. Murali, S., Mutapcic, A., Atienza, D., Gupta, R., Boyd, S., Benini, L., De Micheli, G.: Temperature control of high-performance multi-core platforms using convex optimization. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE ’08, pp. 110–115. ACM, New York, NY, USA (2008). <https://doi.org/10.1145/1403375.1403405>. <http://doi.acm.org/10.1145/1403375.1403405>
26. Ogawa, S., Shiono, N.: Generalized diffusion-reaction model for the low-field charge-buildup instability at the si-sio₂ interface. *Phys. Rev.* **51**(7), 4218–4230 (1995)
27. Rahimi, A.: From variability-tolerance to approximate computing in parallel computing architectures. Ph.D. thesis, University of California San Diego, <https://escholarship.org/uc/item/1c68g008> (2015)
28. Rahimi, A., Benini, L., Gupta, R.K.: Analysis of instruction-level vulnerability to dynamic voltage and temperature variations. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 1102–1105 (2012). <https://doi.org/10.1109/DAT.2012.6176659>
29. Rahimi, A., Benini, L., Gupta, R.K.: Procedure hopping: A low overhead solution to mitigate variability in shared-l1 processor clusters. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED ’12, pp. 415–420. ACM, New York, NY, USA (2012). <https://doi.org/10.1145/2333660.2333754>. <http://doi.acm.org/10.1145/2333660.2333754>
30. Rahimi, A., Benini, L., Gupta, R.K.: Aging-aware compiler-directed vliw assignment for gpgpu architectures. In: Proceedings of the 50th Annual Design Automation Conference, DAC ’13, pp. 16:1–16:6. ACM, New York, NY, USA (2013). <https://doi.org/10.1145/2463209.2488754>. <http://doi.acm.org/10.1145/2463209.2488754>
31. Rahimi, A., Benini, L., Gupta, R.K.: Application-adaptive guardbanding to mitigate static and dynamic variability. *IEEE Trans. Comput.* (2013). <https://doi.org/10.1109/TC.2013.72>
32. Rahimi, A., Benini, L., Gupta, R.K.: Hierarchically focused guardbanding: An adaptive approach to mitigate pvt variations and aging. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pp. 1695–1700 (2013). <https://doi.org/10.7873/DAT.2013.342>
33. Rahimi, A., Benini, L., Gupta, R.K.: Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. *IEEE Trans. Circuits Syst. II Express Briefs* **60**(12), 847–851 (2013). <https://doi.org/10.1109/TCSII.2013.2281934>
34. Rahimi, A., Marongiu, A., Burgio, P., Gupta, R.K., Benini, L.: Variation-tolerant openmp tasking on tightly-coupled processor clusters. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pp. 541–546 (2013). <https://doi.org/10.7873/DAT.2013.121>
35. Rahimi, A., Marongiu, A., Gupta, R.K., Benini, L.: A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters. In: 2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–10 (2013). <https://doi.org/10.1109/CODES-ISSS.2013.6659022>

36. Rahimi, A., Benini, L., Gupta, R.K.: Temporal memoization for energy-efficient timing error recovery in gpgpus. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–6 (2014). <https://doi.org/10.7873/DATE2014.113>
37. Rahimi, A., Cesarini, D., Marongiu, A., Gupta, R.K., Benini, L.: Improving resilience to timing errors by exposing variability effects to software in tightly-coupled processor clusters. IEEE J. Emerging Sel. Top. Circuits Syst. **4**(2), 216–229 (2014). <https://doi.org/10.1109/JETCAS.2014.2315883>
38. Rahimi, A., Cesarini, D., Marongiu, A., Gupta, R.K., Benini, L.: Task scheduling strategies to mitigate hardware variability in embedded shared memory clusters. In: Proceedings of the 52Nd Annual Design Automation Conference, DAC '15, pp. 152:1–152:6. ACM, New York, NY, USA (2015). <https://doi.org/10.1145/2744769.2744915>. <http://doi.acm.org/10.1145/2744769.2744915>
39. Rahimi, A., Ghofrani, A., Cheng, K.T., Benini, L., Gupta, R.K.: Approximate associative memristive memory for energy-efficient gpus. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE '15, pp. 1497–1502 (2015). <http://dl.acm.org/citation.cfm?id=2757012.2757158>
40. Rahimi, A., Benini, L., Gupta, R.K.: Circa-gpus: Increasing instruction reuse through inexact computing in gp-gpus. IEEE Des. Test **33**(6), 85–92 (2016). <https://doi.org/10.1109/MDAT.2015.2497334>
41. Rahimi, A., Benini, L., Gupta, R.K.: Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software. Proc. IEEE **104**(7), 1410–1448 (2016). <https://doi.org/10.1109/JPROC.2016.2518864>
42. Rahimi, A., Benini, L., Gupta, R.K.: From Variability Tolerance to Approximate Computing in Parallel Integrated Architectures and Accelerators. Springer International Publishing (2017)
43. SDAccel: <http://www.xilinx.com/products/design-tools/sdx/sdaccel.html> (2015)
44. Singh, P., Karl, E., Sylvester, D., Blaauw, D.: Dynamic nbti management using a 45 nm multi-degradation sensor. IEEE Trans. Circuits Syst. I Regular Papers **58**(9), 2026–2037 (2011). <https://doi.org/10.1109/TCSI.2011.2163894>
45. Singh, P., Karl, E., Blaauw, D., Sylvester, D.: Compact degradation sensors for monitoring nbti and oxide degradation. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(9), 1645–1655 (2012). <https://doi.org/10.1109/TVLSI.2011.2161784>
46. Tschanz, J., Bowman, K., Walstra, S., Agostinelli, M., Karnik, T., De, V.: Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In: 2009 Symposium on VLSI Circuits, pp. 112–113 (2009)
47. Yazdanbakhsh, A., Mahajan, D., Thwaites, B., Park, J., Nagendrakumar, A., Sethuraman, S., Ramkrishnan, K., Ravindran, N., Jariwala, R., Rahimi, A., Esmaeilzadeh, H., Bazargan, K.: Axilog: Language support for approximate hardware design. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 812–817 (2015). <https://doi.org/10.7873/DATE.2015.0513>
48. Yazdanbakhsh, A., Park, J., Sharma, H., Lotfi-Kamran, P., Esmaeilzadeh, H.: Neural acceleration for gpu throughput processors. In: Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, pp. 482–493. ACM, New York, NY, USA (2015). <https://doi.org/10.1145/2830772.2830810>. <http://doi.acm.org/10.1145/2830772.2830810>
49. Yuan, F., Xu, Q.: Intimefix: A low-cost and scalable technique for in-situ timing error masking in logic circuits. In: Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE, pp. 1–6 (2013)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Reliable CPS Design for Unreliable Hardware Platforms



Wanli Chang, Swaminathan Narayanaswamy, Alma Pröbstl,
and Samarjit Chakraborty

1 Introduction

Battery-operated cyber-physical systems (CPS) increasingly exist in households, factories, and the public area. For instance, zero local emission, independence from fossil fuels, and potential improvement of energy conversion efficiency have made electric vehicles (EVs) an alternative of conventional vehicles with internal combustion engines (ICEs). Design of the underlying embedded control loops such as electric motor control, braking control, stabilization, and battery management plays a crucial role in EVs and other types of battery-operated devices. Conventionally, these control loops are evaluated by a number of quality-of-control (QoC) indices. One common QoC metric is settling time. In order to ensure performance and reliability, the design also needs to take into account a number of issues on the hardware implementation platforms, such as battery behaviour and semiconductor aging. Battery is the key component influencing the device performance, when being the main power source. As the integrated circuit fabrication technology has progressed, processors become more and more susceptible to aging. In order to ensure correct functioning, the processor operating frequency has to be reduced, which could potentially worsen QoC and compromise reliability. The focus of this

W. Chang (✉)
University of York, York, YO10 5DD, UK
e-mail: wanli.chang@york.ac.uk

S. Narayanaswamy · A. Pröbstl
TU Munich, München, Germany
e-mail: swaminathan.narayanaswamy@tum.de; alma.proebst@tum.de

S. Chakraborty
UNC Chapel Hill, Chapel Hill, NC, United States
e-mail: samarjit@cs.unc.edu

chapter is on a design framework towards reliability of CPS considering unreliable hardware platforms.

A battery pack with large capacity is needed to offer longer usage. However, with larger capacity, the battery weight also increases leading to higher energy consumption. Moreover, the capacity is often restricted by the space that can be allocated to the battery pack. One potential solution to the above problem is to design the controller in such a way that the energy consumption of the control task can be minimized.

All off-the-shelf battery packs are labelled with a nominal capacity. However, due to the rate capacity effect, the full charge capacity (FCC) of a battery pack, which is defined to be the amount of electric charges that can be delivered from the battery after it is fully charged, actually varies with different discharging current profiles. Generally speaking, larger discharging current tends to reduce the FCC. For most common lithium-ion batteries in the market, the capacity could potentially get significantly compromised if the rate capacity effect is not properly considered in the control systems design. In this chapter, we discuss an optimization framework considering QoC as one design objective and battery usage as the other. We quantify the battery usage by the total duration that the battery can be used to continuously run the control task after one full charge. In order to maximize the battery usage, the energy consumption of the control task should be small and the battery FCC should be increased by generating a battery-friendly discharging current profile. The battery aging effect can also be incorporated. That is, the battery behaviour in the long run is another optimization dimension.

The other important design aspect is processor aging. As a processor ages, the switching time of its transistors increases, resulting in longer path delays. On-chip monitors could be used to measure the delay of the critical path. It always has to be guaranteed that the signal transmission can be completed along any path within one clock cycle. Therefore, the processor operating frequency is reduced based on the new critical path delay. On the other hand, a shorter sampling period can potentially provide a better QoC. Therefore, with a smaller processor operating frequency, the sampling period increases and QoC gets deteriorated, which is dangerous and thus highly unwanted for safety-critical applications such as electric motor control in EVs. To deal with the above situation, we can re-optimize the controller with the longer sampling period, which results from processor aging.

The entire design flow towards CPS reliability considering unreliable hardware platforms is divided into two phases. In Phase I, before the processor ages, an optimization framework is used with QoC and battery behaviour considered as design objectives. With heuristic methods implemented, this battery-aware controller design gives a Pareto front of well-distributed and non-dominated solutions. The trade-off between these objectives is explored. In Phase II, after the processor ages, QoC is found to get degraded if the controller design does not change. The same optimization framework is used with slight modification. The processor aging effect is mitigated in the way that there is a minimal change of QoC with all safety requirements satisfied.

The remainder of this chapter is organized as follows: Sect. 2 gives an overview of the background on embedded control systems design, battery rate capacity effect

and aging, as well as processor aging. In Sect. 3, we present the reliable CPS design framework, and finally, Sect. 4 concludes the chapter.

2 Background

2.1 Control Systems

In this subsection, we first describe the feedback control application considered in this chapter and several background concepts. Then, we present the system modelling of the electric motor control application in EVs.

2.1.1 Basic Concepts

Plant Dynamics A control scheme is responsible for controlling a plant or dynamical system. In this chapter, we consider linear time-invariant (LTI) single-input single-output (SISO) systems,

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}\mathbf{x}(t),\end{aligned}\tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^m$ is the state vector, $\dot{\mathbf{x}}(t)$ is the derivative of $\mathbf{x}(t)$ with respect to time, $y(t)$ is the output, and $u(t)$ is the control input applied to the system. The number of dimensions for the system is m . Constant matrices \mathbf{A} , \mathbf{B} , and \mathbf{C} are of appropriate dimensions with respect to m . In a state-feedback control algorithm, the control input $u(t)$ is computed utilizing the plant states $\mathbf{x}(t)$ as feedback signals. The computed $u(t)$ is then applied to the plant, which is expected to achieve the desired behaviour.

Discretized Dynamics In most applications, the controller is implemented in a digital fashion on a computer. This implies that the plant states must be sampled when measured by the sensors. Assuming the sampling period to be a constant, the continuous-time system in (1) can be transformed into the following discrete-time system:

$$\begin{aligned}\mathbf{x}[k+1] &= \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d u[k], \\ y[k] &= \mathbf{C}_d\mathbf{x}[k],\end{aligned}\tag{2}$$

where sampling instants are $\{t_k \mid k \in \mathbb{N}\}$, the sampling period is $t_{k+1} - t_k = h$, and

$$\mathbf{A}_d = e^{\mathbf{A}h}, \quad \mathbf{B}_d = \int_0^h (e^{\mathbf{A}t} dt) \mathbf{B}, \quad \mathbf{C}_d = \mathbf{C}.\tag{3}$$

It is noted that $\mathbf{x}[k]$ and $y[k]$ are the values of $\mathbf{x}(t)$ and $y(t)$ at $t = t_k$. The initial condition is denoted as $\mathbf{x}[0]$. The control input $u[k]$ is applied to the plant from t_k to t_{k+1} .

Feedback Controller One common goal of a control task is to make $y[k] \rightarrow r$ as soon as possible, where r is the reference for $y[k]$ to track. Towards this and other application-specific objectives, we design $u[k]$ utilizing the states $\mathbf{x}[k]$. This is then a state-feedback controller with a general structure as follows:

$$u[k] = \mathbf{K}\mathbf{x}[k] + Fr, \quad (4)$$

where \mathbf{K} is the feedback gain vector and F is the feedforward gain.

Closed-Loop System With the feedback controller as shown in (4), the system closed-loop dynamics from (2) becomes

$$\mathbf{x}[k+1] = (\mathbf{A}_d + \mathbf{B}_d\mathbf{K})\mathbf{x}[k] + \mathbf{B}_dFr = \mathbf{A}_{cl}\mathbf{x}[k] + \mathbf{B}_dFr, \quad (5)$$

where \mathbf{A}_{cl} is the closed-loop system matrix. Different locations of the closed-loop system poles, i.e., eigenvalues of \mathbf{A}_{cl} , result in different system behaviours. In the pole placement, poles are placed (eigenvalues are set) to fulfil various high-level goals, such as optimization of QoC and other application-specific objectives, and constraints satisfaction. In order to ensure system stabilization, all the poles must be less than unity. In this chapter, we restrict the poles in the real non-negative plane—which is common in most of the real-life design problems.

Once the poles are decided, the feedback gain vector \mathbf{K} can be computed with Ackermann's formula. The static feedforward gain F used to make $y[k]$ track the reference r can be computed by

$$F = \frac{1}{(\mathbf{C}_d(\mathbf{I} - \mathbf{A}_{cl})^{-1}\mathbf{B}_d)}, \quad (6)$$

where \mathbf{I} is an n -dimensional identity matrix [2].

QoC We use settling time as the metric to quantify the QoC. The time it takes for the system output $y[k]$ to reach and stay in a closed region around the reference value r (e.g., $0.98r - 1.02r$) is the settling time and denoted as t_s . Shorter t_s indicates better QoC. When the controller poles are given, and the feedback and feedforward gains are computed accordingly, the output behaviour can be simulated, and the settling time can be derived.

Constraints There are often hard physical constraints that have to be respected in the embedded control systems, as part of the safety requirements [6, 8]. For instance, the input signal $u[k]$ could be constrained by an upper limit U_{\max} and a lower limit U_{\min} . Similarly, the plant states could be constrained by a region. With the given controller poles and the corresponding gains, both the plant states and the control input throughout the entire control task can be simulated. Therefore, the constraints satisfaction can be evaluated.

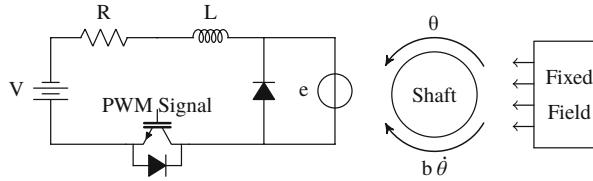
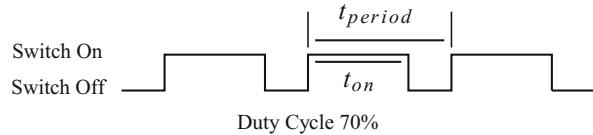


Fig. 1 A diagram of a DC motor with the armature circuit powered up by a battery pack

Fig. 2 Pulse-width modulation control signals in the armature circuit to adjust the DC voltage applied to the rotor



2.1.2 Electric Motor Control

Electric motor control is a key function in EVs. As shown in Fig. 1, we consider a DC motor running in the *speed control* mode. The controller is supposed to operate the motor at various speeds according to the driver input and environmental conditions. The DC voltage provided by the battery pack is V . The resistance and inductance in the armature circuit are R and L , respectively. The back electromotive force (EMF) from the motor is e . The insulated gate bipolar transistor (IGBT) works as a switch controlled by pulse-width modulation (PWM) signals at the gate. When the switch is on, V is applied to the armature circuit. When the switch is off, the diode flows out the remaining current in the motor and thus the applied voltage is equivalent to zero. Periodic PWM signals are shown in Fig. 2, where the duty cycle c is calculated as

$$c = \frac{t_{on}}{t_{period}}, \quad (7)$$

and the effective voltage applied in the armature circuit is

$$V_{eff} = cV. \quad (8)$$

We can clearly see that V_{eff} is adjustable between 0 and V by controlling the PWM signals.

In general, the torque T generated by a DC motor is proportional to the armature current i and the strength of the magnetic field. We assume the magnetic field to be constant and thus the torque is calculated as

$$T = K_t i, \quad (9)$$

where K_t is the motor torque constant. We denote the angular position of the motor to be θ . The angular velocity and acceleration are then $\dot{\theta}$ and $\ddot{\theta}$, respectively. The back EMF is proportional to the angular velocity of the shaft by a constant factor K_e as follows:

$$e = K_e \dot{\theta}. \quad (10)$$

A viscous friction model is assumed and the friction torque is proportional to the shaft angular velocity $\dot{\theta}$ by a factor of b . Now we can derive the following governing equations based on Newton's second law and Kirchhoff's law:

$$\begin{aligned} J\ddot{\theta} + b\dot{\theta} &= K_t i, \\ L \frac{di}{dt} + Ri &= V_{eff} - K_e \dot{\theta}, \end{aligned} \quad (11)$$

where J is the moment of inertia of the motor. It is noted that in the steady state (i.e., $\ddot{\theta} = 0$),

$$\dot{\theta} = \frac{K_t i}{b}. \quad (12)$$

The state-space system modelling as in (1) becomes

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} &= \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} V_{eff}, \\ y &= [1 \ 0] \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix}. \end{aligned} \quad (13)$$

The states are the angular velocity of the motor $\dot{\theta}$, constrained in $[0, \dot{\theta}_{max}]$, and the armature current i , constrained in $[0, i_{max}]$. The control input is the effective voltage V_{eff} , constrained in $[0, V]$ as discussed above. The system output is $\dot{\theta}$. The control goal is to make $\dot{\theta}$ track r .

2.2 Battery

Batteries are increasingly used as power source for many applications nowadays ranging from low-power applications such as portable electronics, wearable devices to high-power applications such as EVs and stationary electrical energy storage (EES) systems for smart grid applications [3]. Lithium-ion battery chemistry has been dominating the market for most low-power and high-power applications mainly due to their high energy and power densities compared to other rechargeable battery chemistry. While the terminal voltage and nominal capacity of a single

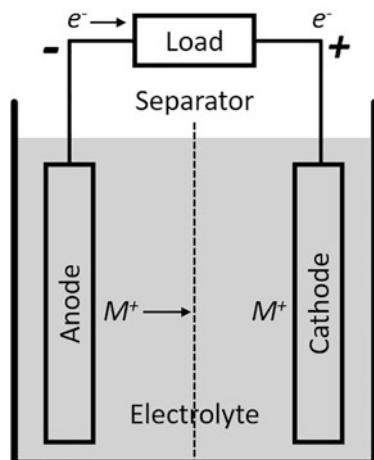
lithium-ion cell are limited for achieving high operating voltages and high capacities required for EVs, multiple individual lithium-ion cells are combined in series or parallel to form a high-power battery pack.

Major concerns affecting the widespread adoption of EVs include range anxiety and battery degradation that will result in an early replacement of their power source. For instance, battery packs in EVs have to be replaced when their state-of-health (SoH), a ratio of capacity at present to the capacity when the battery was new, falls below 70%. In addition to the long-term aging, battery packs are also subject to capacity degradation within individual charging–discharging cycles. This is mainly due to the *rate capacity* effect, which states that discharging a battery with a higher current will reduce the overall capacity of the pack that can be used in this cycle. Therefore, while designing control applications that use battery as a power source, the capacity degradation at single charging–discharging cycles and long-term battery aging have to be considered for maximizing the battery usage and its lifetime.

2.2.1 Battery Basics

Batteries are electrochemical storage devices, meaning their chemical reaction is coupled with an electron transfer. They perform a reversible chemical reaction, which allows them to store electrical energy (*charging*) and release the stored electrical energy by performing the opposite reaction (*discharging*). The basic unit of a rechargeable battery is an electrochemical cell, which consists of a positive electrode *cathode*, a negative electrode *anode*, and an electrolyte to favour the movement of the charge carriers between the two electrodes inside the cell as shown in Fig. 3.

Fig. 3 Electrochemical cell. Shuttle ions (M^+) are oxidized at the anode and move towards the cathode releasing an electron (e^-) to the outer circuit powering the load [13]



During the discharging process, shuttle ions (M^+) are oxidized at the anode side and release electrons (e^-), which travel through the outer circuit to power the load. The oxidized shuttle ions move through the electrolyte to the cathode inside the cell and are reduced by the incoming electrons from the outer circuit. This process is represented by the following equations:



The opposite reaction takes place during charging, facilitating storage of electrical energy in the form of chemical reactions.

In the ideal case, one would assume that while discharging the voltage of the electrochemical cell as seen by the load stays constant throughout the discharging process and suddenly drops to zero when the battery capacity is empty. Moreover, the capacity of the battery stays constant irrespective of the amplitude of the discharge current. However, in reality, the battery exhibits several non-linear effects and as a result the battery voltage instead of remaining constant slowly decreases with time while discharging. Furthermore, the usable capacity of a battery significantly depends on the rate of the load current. Discharging a battery with a higher current will result in a reduced effective capacity obtained from the cell.

2.2.2 Rate Capacity Effect

The FCC of a battery pack is reduced when a battery is discharged with a higher discharge current [5]. This can be seen from Fig. 4 where discharging a cell with a higher current reaches the lower threshold voltage faster than discharging with a lower current. This effect is termed as *rate capacity* or *rate* effect. The fundamental concept behind the rate capacity effect can be explained in terms of overpotential as in [13]. Whenever a current is drawn from a battery, the voltage of that battery will drop depending upon the magnitude of the discharging current. For a battery to obtain maximum energy output, the cell voltage V_T should follow the discharge profile of the equilibrium voltage V_0 , which is defined as the cell voltage at the chemical equilibrium at a given state of charge and temperature. However, the cell voltage deviates with the discharging current and this deviation is termed as overpotential η , which can be expressed as

$$\eta = V_0 - V_T. \quad (16)$$

This overpotential is mainly divided into three parts as ohmic, activation, and concentration overpotentials. At higher states of charge, the cell voltage is predominantly dominated by ohmic overpotentials, which behaves like a resistive drop to the cell voltage and as the cell discharges to a lower state of charge, the activation and

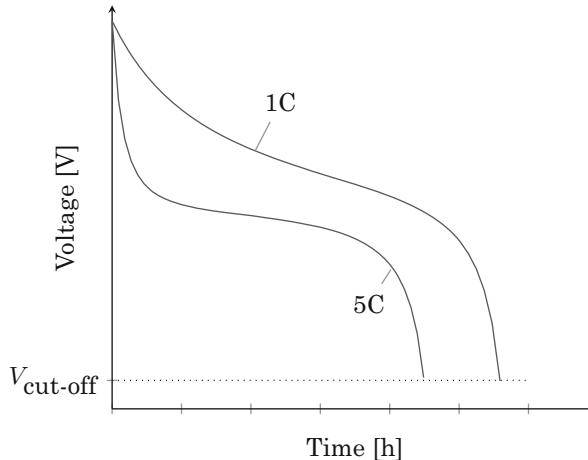


Fig. 4 Higher discharge current results in a reduced usable capacity of the cell due to rate capacity effect. The discharge rate is given by C rating, where 1C means the cell is discharged completely within 1 h [9]

concentration overpotentials dominate the ohmic drop. This reduction in cell voltage and the capacity due to the overpotentials of the cell is termed as rate capacity effect.

In battery terminology, the *C-rate* is often used to define the charge or discharge current of a battery. 1C corresponds to the current necessary to charge or discharge the battery completely in 1 h, whereas a 2C discharge will deplete the battery in half hour. The rate capacity effect is modelled by using Peukert's law [9] as

$$L = \frac{a}{I^b}, \quad (17)$$

where L is the battery lifetime, I is the discharge current, a and b are constants obtained from experiments. In ideal case a would be the battery capacity and b would be equal to 1, whereas in reality a is close to the battery's capacity and b is greater than one. While this model holds good for predicting battery capacity for constant continuous load, it does not work well with variable or interrupted loads. An extended version of Peukert's law was proposed in [15] as

$$L_t = \frac{a}{\left(\frac{\sum_{k=1}^n I_k(t'_{k+1}-t'_k)}{L_t} \right)^b}, \quad (18)$$

where $t'_1 = 0$ is the starting time stamp and $L_t = t'_{n+1}$ is the total duration that the battery can be used and divided into n slots.

2.2.3 Battery Aging

In addition to the single cycle capacity loss due to rate capacity effect, which can be rectified by reducing the discharging current at subsequent cycles, battery aging is a long-term process where the battery cell cannot hold the same amount of charge as it was new. Battery aging can be classified into calendar aging and cycling aging, where the former refers to the loss of capacity due to storing at high states of charge and high temperatures and the latter refers to the loss of lithium-ions due to the charge/discharge process. The main factors for battery cycling aging are depth of discharge (DoD), average state of charge, state-of-charge swing, temperature, and the rate of the discharge current [18] as

$$Q_{loss} = f(t, T, DoD, Rate), \quad (19)$$

where t is the cycling time. Without the DoD, which does not significantly affect the cycling capability of lithium-ion cells, the capacity loss can be modelled with the following equation:

$$Q_{loss} = B \cdot \exp\left(\frac{-E_a}{RT}\right) (A_h)^z, \quad (20)$$

where R is the gas constant, T is the temperature, A_h is the ampere-hour throughput of the cell, z is the power law factor, and B is a constant obtained by experimental data. With multiple experimental analysis for different discharge rates performed in [18], the value of z was approximated to 0.55 and the constant B was calculated for each C-rate. Figure 5 shows that with a higher discharge current the battery capacity drops significantly and will reach their end-of-life faster than discharging at a lower discharge current.

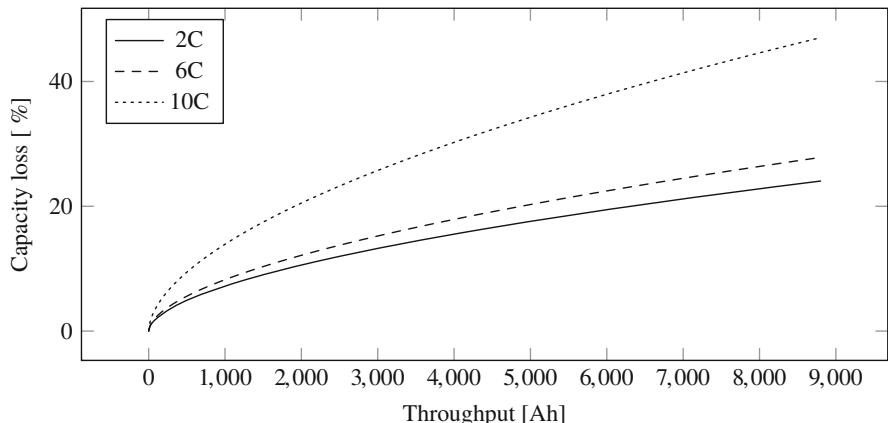


Fig. 5 Capacity loss due to higher discharge rates [5]

The capacity loss with discharging can be approximately modelled by the following equation as proposed in [18]:

$$Q_{loss} = B \cdot \exp \left[\frac{-31700 + 370.3 \cdot C_{rate}}{RT} \right] (A_h)^{0.55}. \quad (21)$$

2.3 Processor Aging

Processors are known to age over time and stress resulting in reduced operating speed. This is problematic in the sense that lower processor speed negatively impacts the performance of applications running on it.

2.3.1 Aging Mechanisms

The main transistor aging mechanisms are hot carrier injection and negative bias temperature instability [1, 17]. Hot carrier injection results in changes in the threshold voltage of the semiconductor. Similarly in the case of negative bias temperature instability, the threshold voltage of MOSFETs is increased. These aging-induced voltage changes result in longer transistor switching time. And as a consequence, the operation of the transistor becomes less reliable, which is of course highly undesirable. Such increased switching time lowers the performance of a processor and of applications running on the processor. Applications then potentially violate performance requirements and produce faulty calculation results, which in most cases is not acceptable.

2.3.2 Countermeasures

As a countermeasure to increased path delays, chips typically would run at very conservative clock rates, also called guard bands or safety margins. Such guard bands include enough margin to achieve the same clock rate throughout the whole intended lifetime of a processor. Intuitively, we see that this pessimistic approach results in a huge waste of resources or energy as the processor could generally achieve much higher speeds [11]. The problem even becomes more severe as we see a trend to decreasing transistor sizes which increases operational variations.

However an increase in the supply voltage could compensate for aging circuits [16]. By that, the delays could be kept constant and the operating frequency could stay the same throughout the intended lifetime of the processor. An adaptive control circuit accommodates for the currently required voltage settings. The downside of this approach is quadratically increased dynamic power consumption of the processor [14] and additional constraints such as maximum input current, cooling requirements, and temperature-dependent reliability problems [17].

Another measure to be taken is to decrease the operating frequency of the processor to compensate for critical path delays [1]. Other than increasing the supply voltage, decreasing the operating frequency actually lowers the power consumption. However, the processor becomes slower, which results in degraded control performance [4] and schedulability issues [12]. Nevertheless, dynamic operating frequency adjustments are a promising approach due to not negatively impacting the overall energy consumption while maintaining high usage of resources.

If aging could be measured on-chip in real-time, the operating voltage and frequency could be adjusted to always provide the maximum speed possible. This however means that control applications—that were designed for a higher speed, which becomes infeasible at some point in time—need to be readjusted to the changes. Such on-chip aging monitors have been developed for the delay of critical paths. Paths that potentially become critical in the future need to be identified first and then their degradation needs to be watched [19]. The path timing monitors typically work on replications of the paths that have statically been identified as the critical ones to not interfere with real functions. The information gathered from the replicated paths is then used to decide if the operating frequency needs to be changed and the according new frequency can be determined from the monitored delays. Processors that implement such critical path monitors are also called autonomic frequency scaling processors.

From the application designers' perspective, the processors lose speed over time and this needs to be considered when designing applications that will run on those processors. Lower processor operating frequency results in longer worst-case execution time of programs. However, in control applications that require high sampling frequencies this worst-case execution time may become the bottleneck for reliable control output. As already outlined above, safety margins between worst-case execution time and sampling period are possible but costly as the processor would run much below its capabilities throughout most of its lifetime. Hence, making full use of the available resources, here the processor speed, is of high interest in cost-sensitive domains.

2.3.3 Aging Estimation

A simple model of critical path delays uses temperature, supply voltage, and stress time, i.e., time during which the processor is active [12]. Let us use this model to consider the use case of processor aging in an electrical taxi. Assuming that the electrical taxi is in use for two-thirds of a day with drivers taking shifts, i.e., for 16 h, we can now estimate the decrease of the processor speed used in the car. We find that after 2 years of taxi use, the on-time of the processor is approximately 1.33 years. As a consequence, the processor speed has degraded in the worst case by roughly 7%. The degradation then continues. After four and then 10 years, the corresponding duration of the processor being switched on amounts to 2.66 and 6.66 years, respectively. These on-times relate to critical path delays of roughly 9% and 12%. As a vehicle usually incorporates many real-time and safety-critical tasks on

multiple processors and at the same time the automotive domain can be considered to be very cost-sensitive, such delays need to be considered in the design stage.

2.3.4 Related Work from the Software Perspective

Multiple works have proposed techniques to design software for aged processors or to reduce the aging process. Processors that have slowed down due to aging have higher execution delays of tasks, which is particularly problematic in the context of hard real-time systems and safety-critical applications [12]. As a result, the schedulability analysis for such safety-critical systems needs to take the estimated worst-case execution delays into consideration and the traditional problem formulation needs to be extended by system lifetime constraints. All scheduled tasks with worst-case delays need to meet their respective deadlines at all times even with severe aging-induced slow-down of the processor speed.

Mitigation of aging can be done in multi-core systems. Such systems often use redundant multi-threading to reduce soft errors. The aging variation among cores is due to varying workloads. Such unbalanced aging states are highly undesired as the system lifetime is constrained by the weakest component, i.e., the slowest core. As a remedy, the mapping of tasks should consider the current aging status of the respective cores. The proposed system [10] maps tasks in a way that aging variations are mitigated and aging of already slower cores is reduced.

3 Reliable CPS Design Framework

We formulate the reliable CPS design on unreliable hardware platforms to be an optimization problem with two objectives— t_s to quantify QoC and L_t to quantify the battery usage. We aim to minimize t_s and maximize L_t . Usually an optimization technique takes objectives either to minimize or maximize but not both. Therefore, we minimize $f_1 = t_s$ and $f_2 = -L_t$. It is noted that L_t is only related to the single-cycle behaviour with the battery rate capacity effect. Other objectives with respect to battery aging can also be defined, such as the total duration that a battery can run the control task after some time like 1 year, i.e., the L_t in 1 year, or when the capacity drops below the threshold like 70%.

The constraints are on the plant states and control input. Additional constraints on the objectives can be imposed depending on the requirements. For example, t_s can be set as shorter than or equal to 20 s. The decision variables are the poles that are less than unity on the real non-negative plane. Clearly, the decision space is continuous. Given a set of decision variables, the objectives and constraints can be evaluated as explained in Sect. 2.

There are generally two goals to pursue in solving bi-objective or multi-objective optimization problems. First, the final solution set (i.e., the obtained Pareto front) only consists of non-dominated points. By convention, Point A is said to dominate

Point B, if Point A is better than or equal to Point B in all objectives and better than Point B in at least one objective. Second, the final solution set has a good distribution in terms of objective values. This gives designers better options under different circumstances.

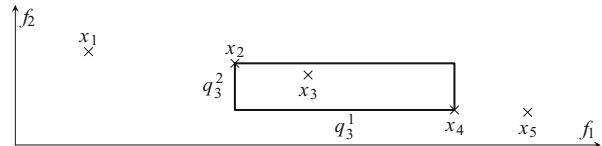
It is challenging to solve the formulated non-convex optimization problem with a continuous design space. Stochastic population-based heuristics such as the non-dominated sorting genetic algorithm (NSGA) can be used. In NSGA, an initial population is first generated and serves as parents. Offspring are then produced with crossover and mutation. The crossover function tries to keep the good genes of parents, which in this context means that the offspring are close to parents in the decision space.¹ The mutation function aims to better explore the decision space. Elitism is implemented for environmental selection, so that the next generation is selected among both the parents and offspring. This not only speeds up convergence but also ensures that good solutions will not be lost once they are found. There are two termination conditions whether the population has converged and whether the maximum allowed number of generations has been reached.

In selection, all the parents and offspring are sorted and ranked by domination. For each point, the number of points that dominate it (i.e., dominating points) is its rank. The new generation is filled in a way that points with lower ranks have priorities. This sorting feature values dominance more than the differences in individual objectives.

Among all the non-dominated points obtained by the above NSGA-based optimization, some may be very close to others in both the objectives. Therefore, it is not necessary to keep all of them. We need to choose a few points to form a well-distributed final solution set. First of all, we define the crowding distance below. As illustrated in Fig. 6, assuming that there are two objectives $\{f_1, f_2\}$ and n solution points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ ² ordered by the value of either objective, for each point $\mathbf{x}_i, i \in \{1, 2, \dots, n\}$, that is not at the end of this point sequence, the crowding distance of \mathbf{x}_i in terms of the objective $f_k, k \in \{1, 2\}$ can be calculated as

$$q_i^k = |f_k(\mathbf{x}_{i+1}) - f_k(\mathbf{x}_{i-1})|, \quad (22)$$

Fig. 6 Illustration of the crowding distance calculation



¹This may not be the case in general. Offspring can be quite different from the parents.

²These are just general notations to explain the method. In this chapter, the decision variables are the poles as discussed earlier.

Algorithm 1 Removal of less representative solution points according to the crowding distance ranking

Input: $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, $n_d, \{\rho_1, \rho_2\}, \{f_1, f_2\}$
Output: $S_d = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_d}\}$

- 26 **for** $j \in \{1, 2, \dots, n - n_d\}$ **do**
- 27 **for** $i \in \{1, 2, \dots, n - j + 1\}$ **do**
- 28 calculate q_i^1 and q_i^2 for the element \mathbf{x}_i as in (22)
- 29 **for** $k \in \{1, 2\}$ **do**
- 30 sort S based on q_i^k from maximum to minimum **for** $i \in \{1, 2, \dots, n - j + 1\}$ **do**
- 31 assign the position index (1 for maximum to $n - j + 1$ for minimum) of \mathbf{x}_i in S to
 r_i^k
- 32 **for** $i \in \{1, 2, \dots, n - j + 1\}$ **do**
- 33 calculate r_i^0 as in (23)
- 34 Remove the element with the maximum r_i^0 from S
- 35 $S_d = S$

where \mathbf{x}_{i+1} and \mathbf{x}_{i-1} are the two closest points to \mathbf{x}_i on each side, respectively. Since we deal with a set of Pareto points that are non-dominated, \mathbf{x}_{i+1} and \mathbf{x}_{i-1} are closest to \mathbf{x}_i in terms of both objectives. Both the end points of the point sequence are assumed to have infinite crowding distance calculation.

The algorithm removing the less representative points to achieve a good distribution is shown in Algorithm 1. The desired number of Pareto points is denoted as n_d . First, for each point, we calculate the two crowding distances corresponding to the two objectives. (Lines 2–4) Two ranks r_i^1 and r_i^2 are assigned to it based on the comparison in crowding distances with other points. (Lines 5–10) If the point \mathbf{x}_i has the maximum crowding distance in terms of f_1 among all the n points, then $r_i^1 = 1$. If \mathbf{x}_i has the minimum crowding distance, then $r_i^1 = n$. The overall rank of \mathbf{x}_i is

$$r_i^0 = \rho_1 r_i^1 + \rho_2 r_i^2, \quad (23)$$

where ρ_1 and ρ_2 are importance factors of the two objectives, respectively. (Lines 11–13) These values depend on the application and

$$\rho_1 + \rho_2 = 1. \quad (24)$$

For example, if in an application, only the distribution in terms of f_1 is important, we may set ρ_1 to be 1 and ρ_2 to be 0. In this case, r_i^0 is equal to r_i^1 and all the points are ranked according to their crowding distances in terms of f_1 . After each point \mathbf{x}_i has an overall rank r_i^0 , the point with the largest r_i^0 is removed from the solution set. (Line 14) The entire process starting from crowding distance calculation is iterated until the desired number of points n_d is reached. Both the end points of the point sequence are always kept in the set (due to the infinite crowding distances)

Fig. 7 An example trade-off between QoC and battery usage

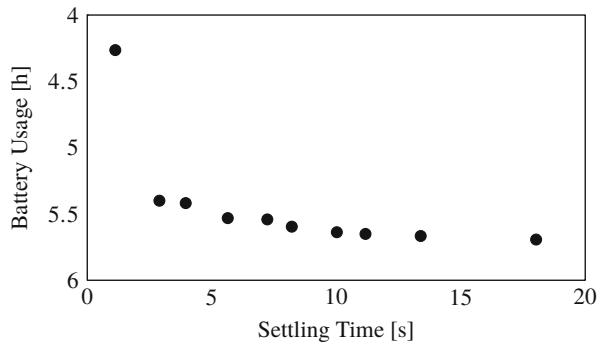


Table 1 The ten design options: the original values and the aged values

Design option	t_s	L_t	Aged t_s	Aged %	Aged L_t	Aged %
1	1.1257 s	4.2643 h	1.2369 s	9.88%	4.4601 h	4.59%
2	2.8972 s	5.4000 h	3.1859 s	9.96%	5.2717 h	-2.38%
3	3.9540 s	5.4182 h	4.3483 s	9.97%	5.2742 h	-2.66%
4	5.6430 s	5.5315 h	6.2062 s	9.98%	5.3111 h	-3.98%
5	7.2314 s	5.5412 h	7.9534 s	9.99%	5.5423 h	0.02%
6	8.2142 s	5.5959 h	9.0345 s	9.99%	5.5141 h	-1.46%
7	10.0238 s	5.6377 h	11.0251 s	9.99%	5.5639 h	-1.31%
8	11.1746 s	5.6506 h	12.2910 s	9.99%	5.5627 h	-1.56%
9	13.3920 s	5.6663 h	14.7302 s	9.99%	5.4613 h	-3.62%
10	18.0271 s	5.6930 h	19.8287 s	9.99%	5.5239 h	-2.97%

The percentage with the aged values is computed based on the original values

to maintain the coverage of the solution set. It is noted that Algorithm 1 takes two objectives into account and can be trivially extended for more objectives.

An example trade-off between QoC and battery usage with the electric motor control presented earlier in this chapter is illustrated in Fig. 7. As the processor ages, there is a decrease in the processor operating frequency and an increase in the sampling period. Taking the number 10% as an example, the change of both objectives is reported in Table 1. In 8 out of the 10 design options shown in Fig. 7, the aged points are dominated by the original points. That is, the settling time is increased (with a positive percentage) and the battery usage is decreased (with a negative percentage). The average deterioration in the control performance is 9.97%. The average deterioration in the battery usage is 1.53%. It is noted that for design option 2, the constraints on the plant states and control input, as discussed earlier in this chapter, are not satisfied anymore.

The processor aging effect can be mitigated by re-optimizing the controller poles with the prolonged sampling period, using the design framework earlier in this chapter. After obtaining the Pareto front, there can be different ways to reach the final solution set. For instance, Algorithm 1 can be deployed again. Alternatively, for each design option, we can keep the point that is closest to the original point

Table 2 The ten design options: the recovered values with re-optimization

Design option	Recovered t_s	Recovered L_t
1	1.1475 s	-7.23%
2	2.8389 s	-10.89%
3	3.9264 s	-9.70%
4	5.9551 s	-4.05%
5	7.5233 s	-5.41%
6	8.1094 s	-10.24%
7	10.8225 s	-1.84%
8	10.8225 s	-11.95%
9	13.7615 s	-6.58%
10	13.7615 s	-30.60%

The percentage is computed based on the aged values

in the settling time. The latter is executed in this case. The recovered results after re-optimization are shown in Table 2. In 9 out of the 10 design options, the recovered points dominate the aged points. That is, the settling time is decreased (with a negative percentage) and the battery usage is increased (with a positive percentage). The average improvement in the control performance is 9.85%. The average improvement in the battery usage is 1.32%. It should be noted that the design options 7 and 8 have the same recovered point. So do the design options 9 and 10. For all the design options, the constraints on the plant states and control input are guaranteed to be satisfied.

4 Concluding Remarks

In this chapter, we have discussed a design optimization framework for CPS. We consider unreliable hardware platforms with respect to processor aging and battery aging and rate capacity effect. The trade-off between the QoC and battery usage is explored. Furthermore, when the processor ages, both the QoC and battery usage get deteriorated, and safety requirements may be violated. The processor aging effect can be mitigated by re-optimizing the controller with the prolonged sampling period, using the same design framework. The change of QoC is minimal and the safety requirements are guaranteed to be met—leading to reliable CPS design. Besides the processor and battery, there are other hardware components that can be unreliable and should be investigated, e.g., the memory and communication systems [7].

References

1. Bowman, K., Tschanz, J., Wilkerson, C., Lu, S.L., Karnik, T., De, V., Borkar, S.: Circuit techniques for dynamic variation tolerance. In: 2009 46th ACM/IEEE Design Automation Conference, pp. 4–7. IEEE, New York (2009)
2. Chang, W., Chakraborty, S.: Resource-aware automotive control systems design: a cyber-physical systems approach. *Found. Trends Electron. Des. Autom.* **10**(4), 249–369 (2016)
3. Chang, W., Lukasiewycz, M., Steinhorst, S., Chakraborty, S.: Dimensioning and configuration of ees systems for electric vehicles with boundary-conditioned adaptive scalarization. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2013)
4. Chang, W., Pröbstl, A., Goswami, D., Zamani, M., Chakraborty, S.: Battery-and aging-aware embedded control systems for electric vehicles. In: 2014 IEEE Real-Time Systems Symposium, pp. 238–248. IEEE, New York (2014)
5. Chang, W., Proebstl, A., Goswami, D., Zamani, M., Chakraborty, S.: Reliable CPS design for mitigating semiconductor and battery aging in electric vehicles. In: 2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 37–42 (2015)
6. Chang, W., Roy, D., Zhang, L., Chakraborty, S.: Model-based design of resource-efficient automotive control software. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2016)
7. Chang, W., Goswami, D., Chakraborty, S., Ju, L., Xue, C., Andalam, S.: Memory-aware embedded control systems design. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **36**(4), 586–599 (2017)
8. Chang, W., Goswami, D., Chakraborty, S., Hamann, A.: OS-aware automotive controller design using non-uniform sampling. *ACM Trans. Cyber-Phys. Syst.* **2**(4), 26 (2018)
9. Jongerden, M., Haverkort, B.: Battery Modeling. No. TR-CTIT-08-01 in CTIT Technical Report Series, Design and Analysis of Communication Systems (DACS) (2008)
10. Knebel, F., Rehman, S., Shafique, M., Henkel, J.: Ageopt-rmt: compiler-driven variation-aware aging optimization for redundant multithreading. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2016). doi:[10.1145/2897937.2897980](https://doi.org/10.1145/2897937.2897980)
11. Lefurgy, C.R., Drake, A.J., Floyd, M.S., Allen-Ware, M.S., Brock, B., Tierno, J.A., Carter, J.B.: Active management of timing guardband to save energy in power7. In: Proceedings of the 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1–11. IEEE, New York (2011)
12. Masrur, A., Kindt, P., Becker, M., Chakraborty, S., Kleeberger, V., Barke, M., Schlichtmann, U.: Schedulability analysis for processors with aging-aware autonomic frequency scaling. In: Proceedings of the 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 11–20. IEEE, New York (2012)
13. Narayanaswamy, S., Schlueter, S., Steinhorst, S., Lukasiewycz, M., Chakraborty, S., Hostler, H.E.: On battery recovery effect in wireless sensor nodes. *ACM Trans. Des. Autom. Electron. Syst.* **21**(4), 60:1–60:28 (2016)
14. Park, J., Abraham, J.A.: A fast, accurate and simple critical path monitor for improving energy-delay product in dvs systems. In: Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design, pp. 391–396. IEEE, New York (2011)
15. Rakhmatov, D.N., Vrudhula, S.B.K.: An analytical high-level battery model for use in energy management of portable electronic systems. In: IEEE/ACM International Conference on Computer Aided Design (ICCAD 2001). IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281), pp. 488–493 (2001)
16. Stojanovic, V., Markovic, D., Nikolic, B., Horowitz, M.A., Brodersen, R.W.: Energy-delay tradeoffs in combinational logic using gate sizing and supply voltage optimization. In: Proceedings of the 28th European Solid-State Circuits Conference, pp. 211–214. IEEE, New York (2002)

17. Tschanz, J., Kim, N.S., Dighe, S., Howard, J., Ruhl, G., Vangal, S., Narendra, S., Hoskote, Y., Wilson, H., Lam, C., et al.: Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In: 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, pp. 292–604. IEEE, New York
18. Wang, J., Liu, P., Hicks-Garner, J., Sherman, E., Soukiazian, S., Verbrugge, M., Tataria, H., Musser, J., Finomore, P.: Cycle-life model for graphite-lifepo4 cells. *J. Power Sour.* **196**(8), 3942–3948 (2011)
19. Wang, S., Chen, J., Tehranipoor, M.: 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), pp. 736–741. ACM, New York (2012). doi:10.1145/2429384.2429543. <http://doi.acm.org/10.1145/2429384.2429543>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Power-Aware Fault-Tolerance for Embedded Systems



Mohammad Salehi, Florian Kriebel, Semeen Rehman,
and Muhammad Shafique

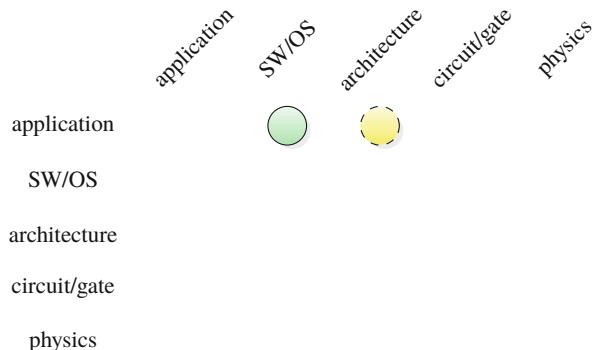
1 Introduction

High core integration in multi-/many-core chips can facilitate reliability management through exploiting different hardening modes considering variants of redundant multithreading (RMT) [17]. However, in such large-scale chips the maximum number of cores that can simultaneously operate is constrained by the thermal design power (TDP, i.e., the maximum amount of power a chip is expected to dissipate and the nominal value for the cooling system to be designed around). Under a given TDP budget either less cores can be powered-on at the full performance level (the power-gated cores are referred to as dark silicon) or relatively more cores can be powered-on at a lower performance level (referred to as “dim” or “gray” silicon) [16]. In case TDP is exceeded, the elevated on-chip temperatures beyond the cooling capacity aggravate reliability threats like temperature-dependent transient faults and aging [1, 5, 6], unless the chip is throttled down which may lead to performance degradation. Reliability management under TDP constraints becomes even more challenging in the presence of manufacturing process variations that result in chip-to-chip or core-to-core variations in the maximum operating frequency and leakage power. This chapter presents a system-level power-reliability management technique for dark silicon multi-/many-core processors that jointly accounts for transient faults, process variations, and the TDP constraint.

M. Salehi (✉)
Guilan University, Rasht, Iran
e-mail: mohammad.salehi@guilan.ac.ir

F. Kriebel · S. Rehman · M. Shafique
Technische Universität Wien (TU Wien), Wien, Austria
e-mail: florian.kriebel@tuwien.ac.at; semeen.rehman@tuwien.ac.at;
muhammad.shafique@tuwien.ac.at

Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions



In this chapter, at first, the system modeling including the power consumption and reliability models, as well as the reliability techniques are presented. Then, the power, reliability, and performance tradeoffs at the software and hardware levels as well as for different hardening modes are studied. After that, the power-reliability management technique is presented. It jointly considers multiple hardening modes at the software and hardware levels, each offering distinct power, reliability, and performance properties. At the software level, it leverages multiple reliable code versions that vary in terms of their reliability, performance, and power properties. At the hardware level it exploits different protection features and different RMT modes, subjected to the manufacturing process variations and different operating conditions (like changing the voltage-frequency levels). Finally, a framework for the system-level optimization is introduced. It considers different power-reliability-performance management problems for many-core processors depending upon the target system and user constraints (i.e., power, reliability, and performance constraints).

The main contributions of this chapter in the scope of this book lie on the application, SW/OS, and architectural layers as illustrated in Fig. 1.

2 System Models

2.1 Power Consumption Model

Power consumption in digital systems consists of static power (e.g., due to sub-threshold leakage) and dynamic power (mainly dissipated due to the circuit switching activities). The power consumption, when the system is operating under a supply voltage and a corresponding maximum allowable frequency (we call it a voltage and frequency (V-f) level), can be written as Eq. 1. For the systems that support the dynamic voltage and frequency scaling (DVFS), different V-f levels are specified at the design time, while at run time, a V-f level is selected considering the

system workload [10]. The static power P_{Static} increases exponentially when the threshold voltage (V_{th}) decreases and is proportional to the supply voltage (V). The dynamic power $P_{Dynamic}$ is proportional to the circuit switching activity (α), load capacitance (C_L), operating frequency (f), and the square of the supply voltage (V) [11, 12, 14, 15].

$$P(V, f) = P_{Static} + P_{Dynamic} = I_0 e^{\frac{-V_{th}}{\eta V_T}} V + \alpha C_L V^2 f \quad (1)$$

2.2 Fault and Reliability Models

In this chapter, transient faults which appear randomly in the underlying hardware and then disappear after certain time are considered. Examples of transient faults are single- and multiple-bit upsets due to energetic radiation particle strikes, circuit metastability, signals cross-talk, voltage noises due to electromagnetic interference (EMI), etc. [1, 5]. Transient faults in the hardware level may manifest themselves as bit-flips in the memory or combinational logic, i.e., the so-called soft errors. These errors may propagate to the software level (e.g., as silent data corruption, crash, halt, and wrong register values) and may finally result in a software failure [5].

These transient faults occur randomly and are typically modeled as a Poisson process with rate λ . The fault rate increases exponentially with a decrease in supply voltage V , as Eq. 2 [11, 12, 15].

$$\lambda(V) = \lambda_0 10^{\frac{V_{max}-V}{\Delta}} \quad (2)$$

In Eq. 2, λ_0 is the raw fault rate at the maximum voltage V_{max} (i.e., the minimum value for fault rate λ) and the parameter Δ determines the amount of increase in fault rate with one step decrease in voltage.

The *software's vulnerability* to soft errors due to hardware-level transient faults at the instruction-level can be quantified by the function vulnerability index (FVI) model [8, 9]. This model projects the error probability for an application software considering vulnerabilities of different instructions (modeled using instruction vulnerability index (IVI)) when executing through different hardware units (e.g., different pipeline stages) in a core. The IVI refers to the probability of an instruction's result being erroneous. It accounts for *temporal vulnerabilities* of different instructions (i.e., different instructions have different execution latency, instruction dependency, and intervals of the operand values) as well as *spatial vulnerabilities* (i.e., different hardware components occupy different chip area and perform different operations) [8, 9]. Knowing the hardware-level fault rate (λ) and the software vulnerability to soft errors (FVI), the software failure rate can be projected as $\lambda(V) \cdot FVI$. Accordingly, the functional reliability FR of an application execution that is defined as the probability of failure-free execution of the application can be written as [11, 12, 15]:

$$FR(FVI, c, V, f) = e^{-\lambda(V) \cdot FVI \cdot \frac{c}{f}} \quad (3)$$

In Eq. 3, $\frac{c}{f}$ is the application execution time under the operating frequency f and c is the number of clock cycles that are required by the core to finish the application execution.

Besides the *functional reliability* of an application, in many systems (e.g., real-time embedded systems), it is also required that the application execution has to finish before a deadline, referred to as *timing reliability* (i.e., probability of meeting deadlines). To jointly consider the functional reliability FR and timing reliability TR , the functional-timing reliability model of Eq. 4 can be employed. In this model, the parameter $0 \leq \beta \leq 1$ specifies the priority for functional and timing reliability. For example, for the systems with tight timing constraints, lower values for β are considered, and for the systems with severe constraints of timing and reliability (e.g., hard real-time systems), $\beta = 0.5$ can be considered to represent the same priority for functional and timing reliability.

$$R = \beta FR + (1 - \beta) TR \quad (4)$$

The reliability for a single application execution given by Eq. 4 may not satisfy the reliability constraint of the target system. In the following section, we study reliability techniques and hardening modes that can be used for soft error mitigation and reliability improvement in many-core processors.

2.3 Reliability Techniques

One prominent technique for tolerating transient faults in many-core processors is *process level redundancy (PLR)*, where multiple identical copies of an application task are executed on different cores and the application finishes successfully if at least one of the task executions finishes successfully (i.e., the application execution fails only if all the task executions fail). Therefore, the *total application reliability* is defined as the probability of at least one application task being executed successfully. Suppose that n identical copies of an application task are executed on n different cores and possible faults can be detected with the probability of μ_{FD} . The total reliability of the application can be calculated as:

$$R_{total}(R_1, R_2, \dots, R_n) = \mu_{FD} \left(1 - \prod_{i=1}^n (1 - R_i) \right) \quad (5)$$

In Eq. 5, R_i is the reliability of the i -th task copy execution (given by Eq. 4). Here, it is assumed that (1) there is no spatial correlation between fault occurrences in different cores and (2) parallel task executions on different cores are not dependent from the viewpoint of fault propagation, i.e., a fault occurrence in a core does not

affect the operation of the other cores. This assumption is also considered for the other reliability techniques in this chapter in which we have parallel task executions on different cores.

2.3.1 Software Error Detection with Re-execution (SEDR)

In this technique, each application task is executed on a single core and a software error detection mechanism (e.g., software-based control flow checking and acceptance tests) is used for error detection. Here, if an error occurs during the task execution, the task is re-executed once again on the same core for error recovery.

Therefore, the reliability of this technique can be calculated by Eq. 6, where μ_{SFD} is the error detection coverage of the software error detection mechanism (i.e., the probability of detecting existing errors).

$$R_{SEDR}(R) = \mu_{SFD}(R + (1 - R)R) = \mu_{SFD}(2R - R^2) \quad (6)$$

Here, it is assumed that there is no temporal correlation between the fault occurrences in consecutive executions of the same task, i.e., a fault occurrence during an execution of a task does not affect the next execution of the same task. This assumption is also considered for the other reliability techniques in this chapter in which we have consecutive task executions on the same core.

2.3.2 Dual Modular Redundancy (DMR) with Re-execution (DMRR)

Software-based error detection in the SEDR technique may not provide a high error detection coverage and also it may not be useful for some applications, e.g., it may entail incurring extra delay that may not be acceptable for hard real-time systems. One practical and powerful error detection mechanism is the comparison of the output result. In this mechanism, two identical copies of each application task are executed on different cores in parallel and the output results of the task are compared for error detection (i.e., DMR is applied at the individual core level). If the comparison task finds that the results are in agreement, the result is assumed to be correct. The implicit assumption here is that it is highly unlikely that both task executions experience the identical errors and they produce identical erroneous results. If the results are different, an error has occurred during the task execution, and the task is re-executed on another core for error recovery. Let R_{cmp} be the reliability of the result comparison process. Assuming that the two cores are identical, each with a reliability R , the reliability of the DMRR technique can be calculated by Eq. 7.

$$R_{DMRR}(R, R_{cmp}) = R_{cmp} \left(R^2 + 2(1 - R)R^2 \right) = R_{cmp}(3R^2 - 2R^3) \quad (7)$$

2.3.3 Triple Modular Redundancy (TMR)

N-Modular redundancy (NMR) that is an $M - of - N$ system with N (an odd number) and $M = (N + 1)/2$ can be applied at the individual core level, where N copies of each task are executed on N different cores in parallel and the results of at least M of them are required to be identical for proper operation. Thus, the task execution fails when the majority voting task finds that fewer than M results are identical [13]. This is similar to the redundant multithreading (RMT) approach if considering architecture-level redundancy management or process level redundancy (PLR) approach if considering operating system-level redundancy management. Here, it is considered that TMR ($N = 3$) is applied at the individual core level, i.e., three copies of each task are executed in parallel on three different cores, and majority voting is performed on the results for error masking. Let R_{vot} be the reliability of the majority voting task. The reliability of TMR can be calculated by Eq. 8.

$$R_{TMR}(R, R_{vot}) = R_{vot} \left(R^3 + 3R^2(1 - R) \right) = R_{vot} \left(3R^2 - 2R^3 \right) \quad (8)$$

3 Power–Reliability–Performance Tradeoffs

3.1 Tradeoffs at the Hardware Level

Due to technology process variations, the maximum operating frequency and the leakage power consumption vary for different cores in a single chip [3]. Figure 2a illustrates that the core-to-core frequency and leakage power variations in an Intel’s 80-core test chip are up to 38 and 47%, respectively [7]. Therefore, regardless of which application is executed, different processing cores present different performance and power consumption.

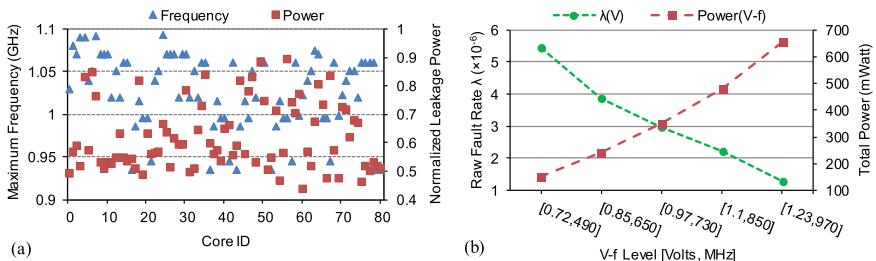


Fig. 2 (a) Core-to-core variations in maximum operating frequency and leakage power and (b) hardware-level fault rate and power variations at different V-f levels. Adapted from [15]

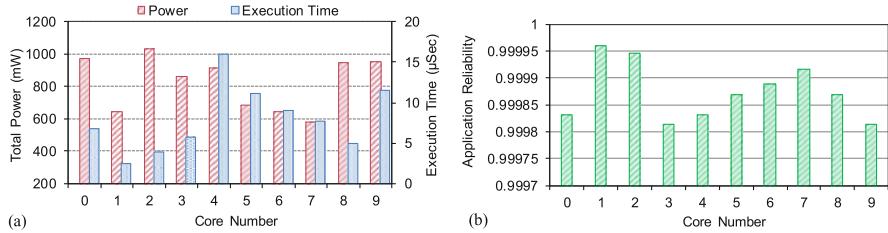


Fig. 3 Core-to-core variations in power, execution time, and reliability when executing the same application. Adapted from [15]

One effective way to reduce power consumption is to decrease the operating V-f level through the DVFS technique. However, based on Eq. 2, decreasing the V-f level increases the hardware-level fault rate. Figure 2b shows that how the total power consumption (Eq. 1) and the hardware-level fault rate (given by Eq. 2) for a given core vary at different V-f levels. For the processor cores in our experiments, it is considered that the V-f level can have five different values as shown in Fig. 2b, i.e., the minimum V-f level is [0.72 V, 490 MHz] and the maximum V-f level is [1.23 V, 970 GHz], see details in Sect. 5.

Due to the core-to-core variations in the frequency and leakage power, when a given application is executed on different cores but under the same supply voltage, it presents different power consumption, performance (execution time), and reliability. Figure 3 shows core power consumption, application reliability, and execution time for the discrete cosine transform (DCT) application when it is executed on different cores but under the same supply voltage. Figure 3a illustrates that due to core-to-core variations in operating frequency, executing a given application on different cores presents different power consumption and performance properties.

According to Eq. 3, the application reliability depends upon the hardware-level fault rate, software vulnerability, and application execution time. Figure 3b illustrates that when a given application is executed on different cores, it provides different reliability levels. This is because, in this case, software vulnerability (*FVI*) and hardware-level fault rate (λ) remain the same in Eq. 3 (the same application is executed under the same voltage level) but due to core-to-core variations in operating frequency, the application execution time ($\frac{C}{f}$) varies when executed on different cores.

The analyses in Figs. 2 and 3 illustrate that the diversities in power and performance of different cores in a chip when executing the same application along with exploiting different V-f levels can be utilized for efficient reliability management at hardware level.

3.2 Tradeoffs at the Software Level

Since different applications execute different instructions on different operand values, they present different power, performance, and reliability properties even when executed on the same core and under the same V-f level. Figure 4 shows the power consumption, execution time, software vulnerability, and reliability for different applications when executed on the same core and under the same V-f level. Different applications exhibit different circuit switching activity and also require different clock cycles to complete, and hence, they exhibit distinct power consumption and execution time properties even when executed on the same core; see Fig. 4a.

Also, since different instructions present different vulnerabilities to soft errors (e.g., single event upsets), as shown in Fig. 4b, different applications exhibit distinct software vulnerabilities. Figure 4b shows that different applications, even when executed under the same V-f level (i.e., under the same hardware-level fault rate) and on the same core, exhibit different system-wide reliability. This is because, based on Eq. 3, the application reliability also depends upon its software vulnerability and execution time.

The above analysis shows that different applications exhibit different power, performance, and reliability levels when executed on different cores, thus enabling power-reliability-performance tradeoffs at software level.

3.3 Tradeoffs for Hardening Modes

3.3.1 Tradeoffs for Reliability Techniques

Reliability techniques usually employ *different types of redundancy* (e.g., hardware, software, and time redundancy) and *different redundancy levels* (e.g., dual or triple modular redundancy). Therefore, they offer different reliability, performance, and power properties. Also, two different reliability techniques may provide the same error tolerance capability but at different performance and power cost. For example,

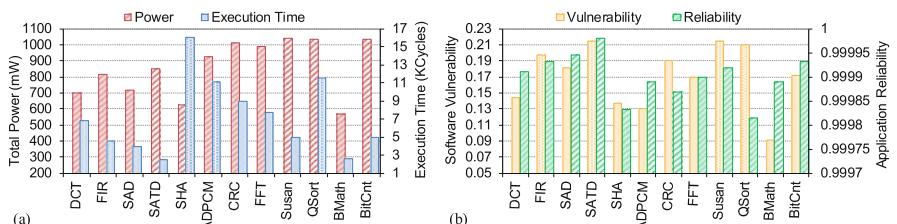


Fig. 4 Application-to-application variations in power, execution time, software vulnerability, and application reliability when executed by the same core. Adapted from [15]

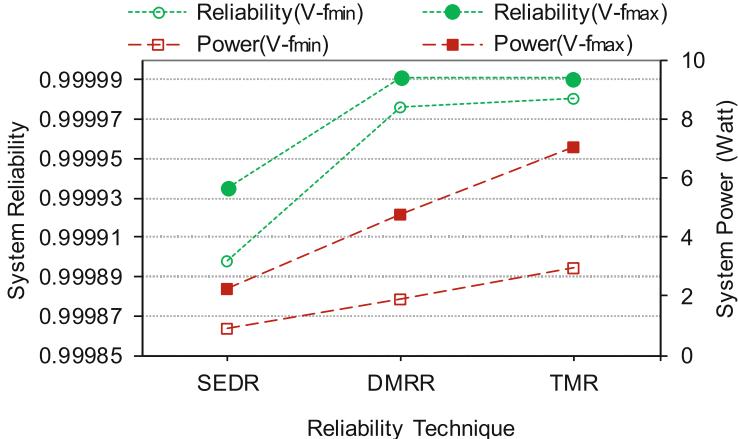


Fig. 5 System-wide reliability and power for different reliability techniques when performed at the minimum and maximum V-f levels ($V-f_{\min}$ and $V-f_{\max}$, respectively). Adapted from [12]

both the DMRR and TMR techniques can tolerate one single task failure; however, when an error occurs, DMRR requires more time to re-execute the task for error recovery, which incurs a performance overhead. Nevertheless, DMRR may consume less power and energy when compared to TMR. This is because when no error occurs, which could be a case for most of the time, DMRR does not require re-executing the task, while TMR always executes the third copy.

Figure 5 shows system reliability and power consumption when the reliability techniques in Sect. 2.3 are employed. To illustrate the effects of scaling the operating V-f level on the system reliability and power consumption, the reliability techniques are executed under the minimum and maximum V-f levels (i.e., $V-f_{\min}$ and $V-f_{\max}$). Also, in this figure, reliability techniques with different redundancy levels are considered (i.e., SEDR with a low redundancy level and TMR with a high redundancy level). Figure 5 illustrates that increasing the redundancy level (from SEDR to TMR) and V-f level (from $V-f_{\min}$ to $V-f_{\max}$) improves reliability but at the cost of increased power consumption.

The experiment in Fig. 5 shows that different reliability techniques when operating in different V-f levels exhibit distinct power, performance, and reliability properties, enabling power-reliability-performance tradeoffs that can be employed for power-reliability management.

3.3.2 Tradeoffs for Software Hardening

To further expand the power-reliability-performance optimization space, a reliability-aware compiler can be used to generate multiple reliable compiled code versions for a given application task through reliability-driven software

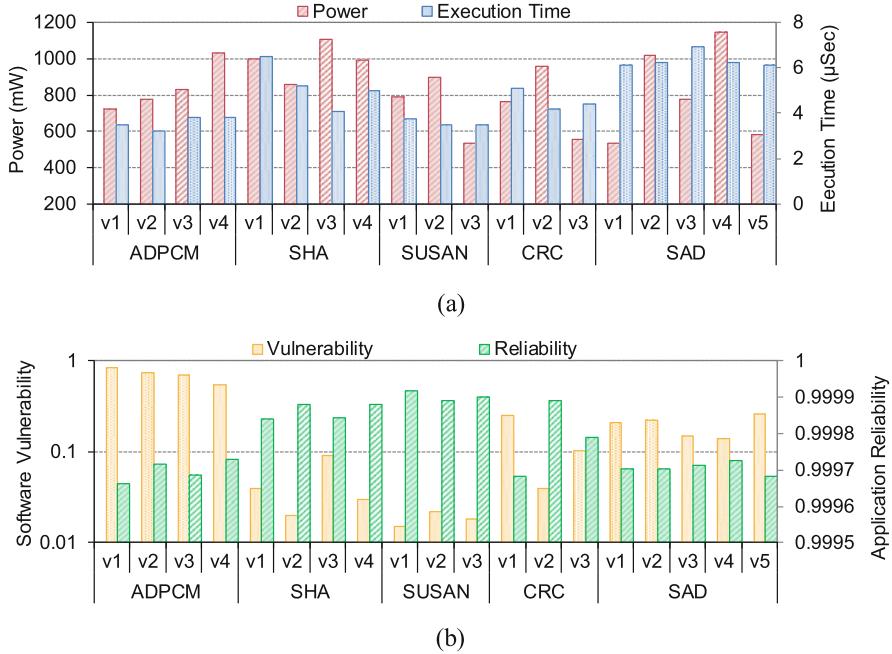


Fig. 6 Different compiled code versions of each application have different: (a) Power and performance (execution time); and (b) Software vulnerability (in log scale) and application reliability. Adapted from [15]

code transformations (see more details in [8, 9]). Different code versions of the same task present dissimilar power, reliability, and performance properties while implementing the same functionality. For instance, Fig. 6 shows power, execution time (in terms of clock cycles), software vulnerability, and overall reliability (given by Eq. 3) of different compiled code versions for five applications.

The reliability and execution time of an application task also vary with the operating V-f level of the underlying core. Figure 7 shows the reliability and execution time of three code versions for the *ADPCM* application under different V-f levels. Figure 7a illustrates that how different code versions of the *ADPCM* application when executed under different V-f levels can be used to achieve a given reliability requirement for the application (R_{req} in this figure). For instance, to meet the reliability requirement $R_{req} \geq 0.999$, shown by the dotted horizontal line in Fig. 7a, the operating V-f level for the code versions *cv1* and *cv2* should be at least [0.97 V, 730 MHz], whereas the V-f level for the code version *cv3* can be [0.85 V, 650 MHz]. Assume that the application execution has a deadline constraint to finish within 5ms, as shown by the dotted horizontal line in Fig. 7b. In this case, the operating V-f level for the code version *cv2* should be at least of [0.85 V, 650 MHz], for *cv1* should be at least [0.97 V, 730 MHz], and for *cv3* should be at least [1.1 V,

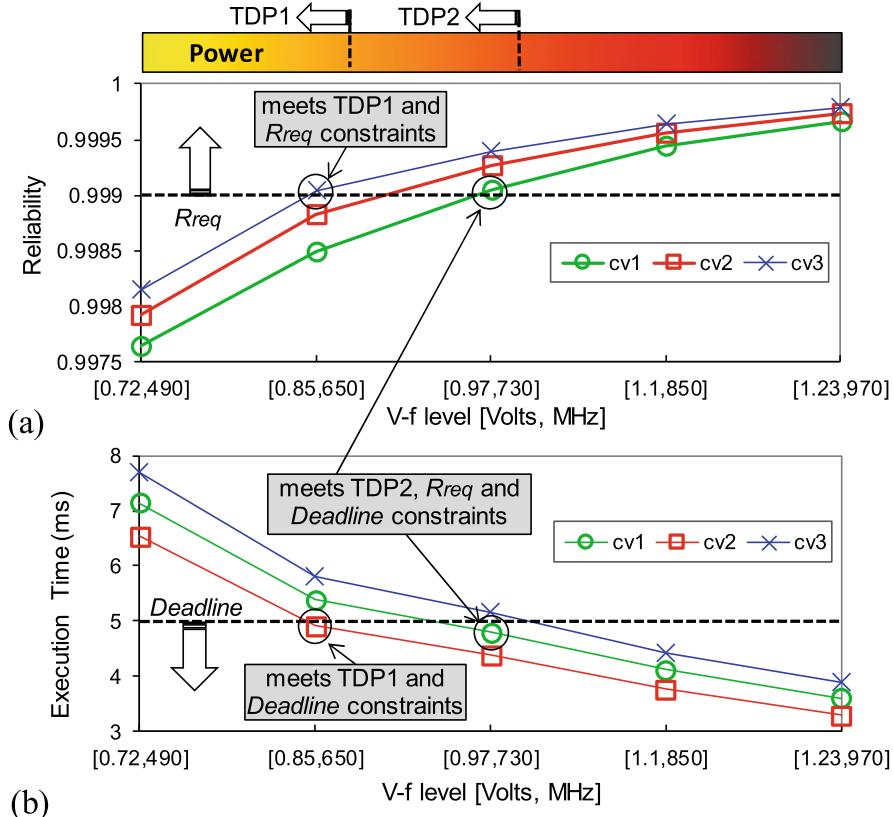


Fig. 7 Reliability and execution time of three compiled code versions for the ADPCM application under different voltage–frequency levels. Adapted from [11]

850 MHz]. Now assume that the underlying core has a TDP constraint that requires its operating V-f level should be at most [0.85 V, 650 MHz] (the TDP1 constraint in Fig. 7). Under the TDP1 constraint, to meet the reliability constraint we should select the code version *cv3*; see Fig. 7a. However, under the given TDP1 constraint, if the deadline constraint has to be met, we would select the code version *cv2*; see Fig. 7b. As another example assume that the core has the TDP2 constraint (i.e., the operating V-f level for the core should be at most [0.97 V, 730 MHz]). In this case, the code version *cv1* is the best choice since it can satisfy both the reliability constraints of $R_{req} \geq 0.999$ and the deadline constraints within 5 ms while meeting the power constraint TDP2.

4 Power–Reliability–Performance Management

From Sects. 2 and 3, the following key observations can be derived that lay the foundation of designing an efficient system for power–reliability–performance Management.

1. Executing tasks at a higher V-f level provides lower execution time and fault rate, resulting in higher system-wide reliability. However, the task power consumption at a high V-f level may be beyond the chip power constraint.
2. An effective way to decrease the power consumption is to lower the operating V-f level, e.g., through DVFS. However, lowering the V-f level leads to an increased execution time of the task that may result in a performance degradation and a missed deadline.
3. Different compiled code versions for an application task exhibit different vulnerability and execution time properties when executed on the same core.
4. Different compiled code versions for each application task when executed by different reliability techniques on different cores with frequency variations and supporting different V-f levels present distinct power, reliability, and performance properties.

In short, the variations in vulnerability and execution time of different compiled versions for each task along with the variations in reliability, power, and performance when using different reliability techniques and V-f levels can be exploited for power–reliability–performance optimization.

The previous works, dynamic redundancy and voltage scaling (DRVS) [12] and dark silicon reliability management (dsReliM) [11], consider the above-mentioned variations at hardware and software levels for run-time power–reliability management. DRVS exploits run-time reliability technique (task-level redundancy) with V-f selection for each application task to minimize system power consumption under reliability and timing (deadline) constraints. dsReliM leverages multiple pre-compiled code versions for each application task with V-f selection at run time to maximize reliability under timing and power constraints. However, such techniques that solely use task-level redundancy or pre-compiled code versions may impose the following restrictions on power–reliability management. Although task-level redundancy can substantially increase reliability, it may increase chip power consumption beyond its power constraint. Also, this technique can only be used if sufficient cores are available for task-level redundancy. In this case, it may be useful to leverage reliable compiled codes to improve reliability, since no extra cores are required to execute different code versions for each application task. Although compile-time software hardening can decrease power consumption, it may increase the execution time of the tasks beyond their timing constraint. Therefore, power–reliability management requires joint considerations of reliability, performance, and power properties of hardening techniques at both software and hardware levels, which is the primary consideration of this chapter.

4.1 Problem Definition

System reliability and total power consumption, V-f level and code version assignments, and task-to-core mapping are represented by different matrices with $n \times m \times c \times v$ elements. Here, n is the number of ready tasks, m is the number of available code versions for each task, c is the number of free cores, and v is the number of available V-f levels for each core. The matrices are:

- $R \in \mathbb{R}^{n \times m \times c \times v}$: A matrix to represent the system reliability. In this matrix, each element $R_{i,j,k,l}$ represents the reliability of the task i when the code version j of the task is executed by the core k under the V-f level l .
- $P \in \mathbb{R}^{n \times m \times c \times v}$: A matrix to represent the system total power consumption. In this matrix, each element $P_{i,j,k,l}$ represents the power consumption for the task i when the code version j of the task is executed by the core k under the V-f level l .
- $X \in \{0, 1\}^{n \times m \times c \times v}$: A matrix to represent the code version and V-f level assignments and task-to-core mapping. Code version j for the task i is mapped to the core k and is executed under the V-f level l if and only if $X_{i,j,k,l} = 1$.

Considering power, reliability, and performance as a design object or a design constraint, the potential goals of a power-aware reliable system design can be:

1. Maximize system reliability while keeping total power consumption under a given power constraint (e.g., TDP) and meeting tasks timing requirements (e.g., tasks deadlines) OR
2. Minimize power consumption while satisfying the system reliability and timing requirements.

The power-reliability-performance management problems can be formulated as a constrained 0-1 integer linear program (ILP). In the following, the problem is formulated where reliability is the design objective, while power and performance are the design constraints. That is,

- **Optimization Goal:** Maximizing the system reliability that is defined by the correct execution of all the application tasks.

$$\text{maximize } \prod_{i,j,k,l} X_{i,j,k,l} R_{i,j,k,l} \quad (9)$$

This is a 0-1 assignment problem, and hence, we have

$$X_{i,j,k,l} \in \{0, 1\} \quad (10)$$

- **Chip Power Constraint:** Total power consumption of the chip, i.e., the sum of power consumption of all cores should be less than the chip power constraint (i.e., chip-level TDP).

$$\sum_{i,j,k,l} X_{i,j,k,l} P_{i,j,k,l} \leq P_{TDP,chip} \quad (11)$$

- **Cores Power Constraint:** Power consumption of each core should be less than the core power constraint (i.e., core-level TDP).

$$X_{i,j,k,l} P_{i,j,k,l} \leq P_{TDP,k} \quad (12)$$

- **Tasks Timing Constraint:** The execution time $\frac{w_{i,j}}{f_{k,l}}$ of a task i when the code version j of the task (with $w_{i,j}$ clock cycles) is executed on the core k at the V-f level l should satisfy the task timing constraint (defined by the deadline d_i).

$$X_{i,j,k,l} \frac{w_{i,j}}{f_{k,l}} \leq d_i \quad (13)$$

- **Code Version Constraint:** The code version does not change during a task execution, i.e., for each execution of a task only one code version can be used.

$$\forall i, k, l \sum_j X_{i,j,k,l} = 1 \quad (14)$$

- **V-f Levels Assignment Constraint:** The V-f level does not change during a task execution, i.e., during a task execution the underlying core can only perform under a single V-f level.

$$\forall i, j, k \sum_l X_{i,j,k,l} = 1 \quad (15)$$

4.2 Proposed Solution

The *power-aware fault-tolerance (PAFT)* technique in this chapter jointly accounts for soft errors, process variations, user defined reliability constraint, and processor power constraint (i.e., TDP). At design time, considering the inherent software-level variations in the execution time of the applications, power, and vulnerability, the PAFT technique selects suitable code versions from multiple compiled codes for each application task (Sect. 4.2.1). At run time, considering the hardware-level variations in performance, fault rate, and power, the PAFT approach selects the hardware/software hardening mode (i.e., reliability technique and code version for each task) and performs task mapping and V-f level allocation (Sect. 4.2.2).

4.2.1 Design-Time Code Selection

As discussed in Sect. 3.3, different compiled code versions for an application task and also different reliability techniques (with different redundancy levels) exhibit different reliability, performance, and power properties. Therefore, for each application task, a tradeoff can be made between two cases:

1. Exploiting a code version with higher reliability and a reliability technique with lower redundancy level (e.g., SEDR) to achieve both high reliability and low power consumption.
2. Exploiting a code version with higher performance and a reliability technique with a higher redundancy level (e.g., TMR) to achieve both high performance and high reliability.

To enable the above tradeoff at run time, we leverage the design-time generated multiple code versions for each task using a reliability-aware compiler (see details in [8, 9]). Then, two types of code versions are chosen as follows (as shown in Fig. 8):

1. **Reliability-Driven Code Selection:** At run time, the reliability of executing a code version of a task on a single core (in the SEDR mode) may be high enough to satisfy the system reliability requirement. In this case, for the task, there is no need to employ a reliability technique with a higher redundancy level. However, the execution time of all the code versions with high reliability, even when executed on a high-performance core and at the maximum V-f level may not be low enough to meet the task deadline constraint. Also, the power consumption of a code version with high reliability may be higher than the processor power budget. Therefore, at design time, for each application task, a set of code versions with high reliability, low execution time, and low power consumption is selected. To do this, first we find the reliability-wise best code versions. Then, from the highly reliable code versions, the performance-wise best code versions (i.e., the code versions with the lowest execution time) are selected. Finally, from the selected code versions, the code versions that provide the lowest power consumption are chosen.
2. **Performance-Driven Code Selection:** The reliability of executing a single task on a single core (in SEDR mode) may not be high enough to satisfy the task reliability requirement. In this case, the task can be executed under a reliability technique with a higher redundancy level (e.g., in the DMRR or TMR mode) to improve its reliability. Here, a code version with a high performance is executed under a high redundancy level to make a balance between timing and functional reliability. Therefore, the performance-wise best code versions with high reliability and low power consumption are chosen.

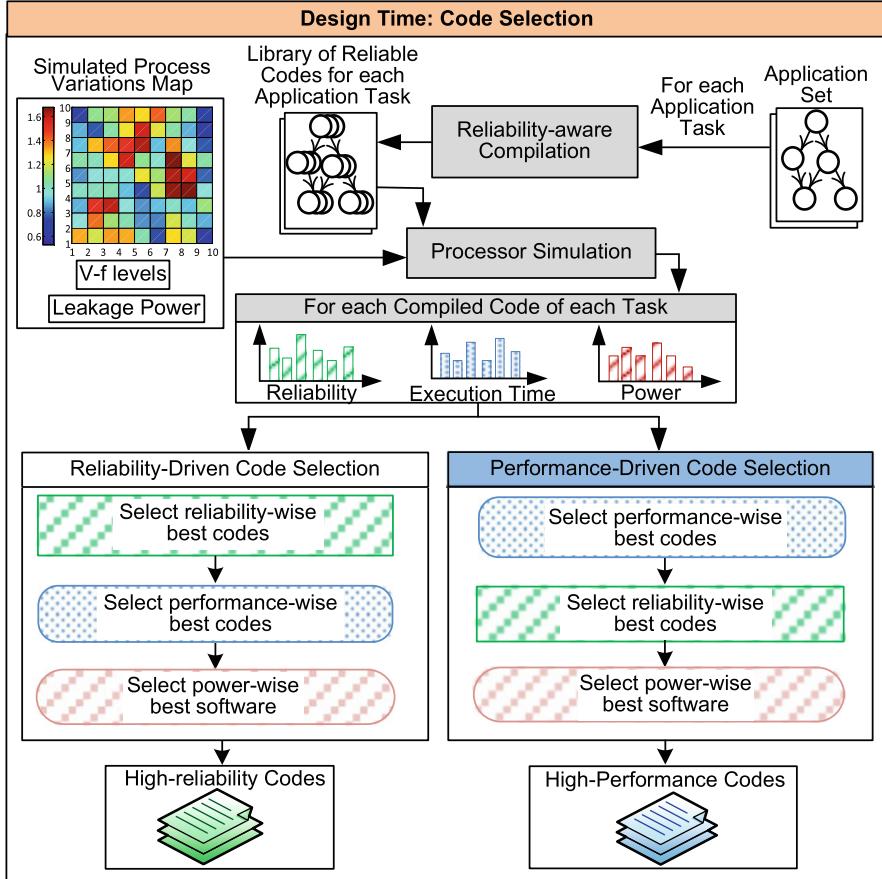


Fig. 8 Overview of the design-time part of the power-aware fault-tolerance (PAFT) technique

4.2.2 Run-Time Hardening Mode and V-f Level Selection and Task-to-Core Mapping

The run-time part of the PAFT technique is shown in Fig. 9. It chooses the hardening mode (reliable code version and reliability technique), operating V-f levels and set of cores to implement the reliability techniques, such that the design objectives are achieved while satisfying the design constraints (e.g., maximized system reliability under timing and power constraints). The problem can be effectively solved by the use of existing ILP solvers (formulated in Sect. 4.1). Since 0-1 ILP problems belong to the class of NP-complete problems, ILP solvers generally exploit branch-and-bound mechanisms to find the optimal solution which leads to an exponential increase in their run-time complexity. Therefore, ILP solvers cannot be used in online scenarios where the parameters that are required for decision-making are

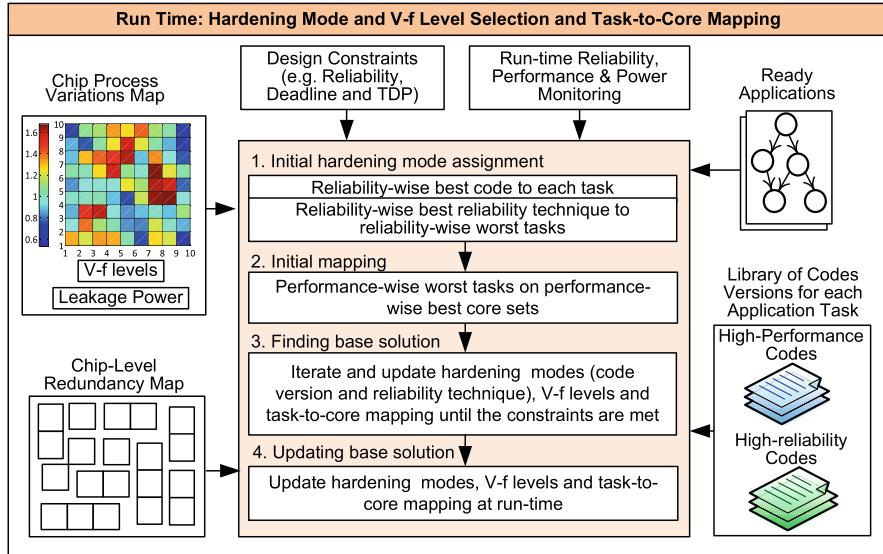


Fig. 9 Overview of the run-time part of the power-aware fault-tolerance (PAFT) technique

determined at run time (e.g., ready tasks and free cores). In the case of this problem, the complexity of ILP solvers increases at run time with the number of ready tasks, code versions for each task, reliability techniques, free cores, and V-f levels. Therefore, for this problem, a heuristic is developed, which at first aggressively chooses the hardening mode, operating V-f levels and set of cores in such a way that the highest possible reliability and performance are obtained. Afterwards, it iterates and updates the hardening mode, operating V-f levels and task-to-core mapping until the design constraints (e.g., reliability, deadline, and power constraints) are satisfied. To do this, the run-time part of the PAFT technique gives the chip processor variation map, chip-level redundancy map, design constraints, and library of selected code versions for each application task as input and performs the following four key steps to each ready application:

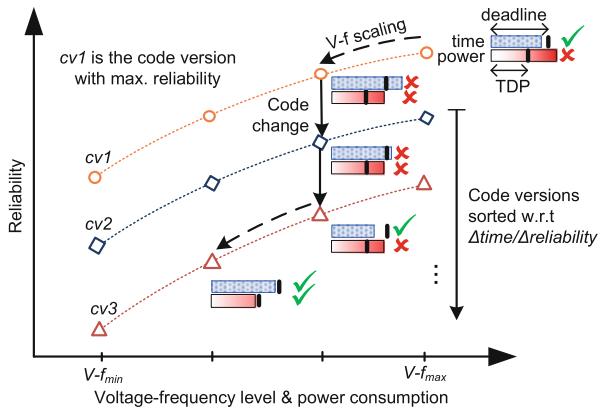
- 1. Initial Hardening Mode Assignment:** First, a reliability-wise best code version is assigned to each task to achieve the highest possible functional reliability for each task. Then, beginning from the task with the lowest reliability, the reliability-wise best technique is assigned to the reliability-wise worst tasks. Therefore, the reliability of the tasks with the lowest reliability is improved, resulting in an improvement in the overall system reliability (the overall system reliability is less than or equal to the reliability of the task with the lowest reliability).
- 2. Initial Mapping:** In this step, starting from the task with the highest execution time (lowest performance), the performance-wise worst tasks are mapped on the performance-wise best cores (cores with the highest operating frequency).

3. **Finding the Base Solution:** Until now, the highest possible reliability and performance have been achieved for the tasks which may be higher than the system performance and reliability requirements. Also, this may lead to an increased chip power consumption beyond its power constraint. In this step, starting from the task with the highest power consumption, the redundancy and V-f level assigned to the task are increased to reduce power consumption until the point that the chip power constraint is satisfied. Here, reducing the operating V-f level may lead to a task deadline miss. In this case, the task code is replaced with a code version with less execution time.
4. **Updating the Base Solution:** At run time, missed deadlines can be considered for performance monitoring and the number of encountered errors can be considered for making the reliability decision. Also, the power information can be acquired from a proxy power monitor. By the use of the run-time reliability, performance, and power monitoring information the system considers the following cases for power-reliability-performance management:
 - (a) When the execution of a task finishes, the free cores are employed to improve the system reliability and performance through updating reliability techniques and task-to-core mapping.
 - (b) When an error occurs, after tolerating the error by the use of a recovery mechanism, the redundancy and V-f levels assigned to the tasks are increased to compensate the reliability degradation and also to provide high reliability against possible consequent errors.
 - (c) When chip power consumption approaches its power constraint, redundancy and V-f levels are decreased to reduce power consumption.

Since estimating the tasks reliability, power, and performance properties is time-consuming, the decisions in steps (a)–(c) are made at run time based on the reliability, power, and performance values that are obtained through the design-time measurements and simulations.

Figure 10 shows how the run-time part of the system works on a ready task. In this figure, for simplicity of the explanations, it is assumed that the reliability technique (e.g., SEDR, DMR, and TMR) and the underlying cores for the task execution are already determined and now the code version and V-f level should be chosen under timing (deadline) and power (TDP) constraints. Suppose that, for the task, the design-time code selection part has selected different pre-compiled code versions (cv_1, cv_2, cv_3, \dots) with different reliability and execution time properties. At run time, to achieve maximum reliability without considering the deadline and power constraints, the code version with the highest reliability (i.e., cv_1 in Fig. 10) is selected to be executed at the maximum V-f level ($V-f_{max}$). In Fig. 10, without loss of generality, it is assumed that the execution time of cv_1 at $V-f_{max}$ is less than its deadline but its power consumption at $V-f_{max}$ exceeds the chip power (TDP) constraint. In this case, the V-f level is scaled down until the power consumption decreases below the TDP constraint. Suppose the case where reducing the V-f level to a lower level increases the task execution time beyond the task deadline (the task timing constraint is missed). In this case, the code is changed to a version with less

Fig. 10 Code version and V-f level assignment for reliability management under timing (deadline) and power (TDP) constraints



execution time (the code version that can meet the deadline). Here, among the code versions that can meet the deadline, the one is selected that provides the maximum execution time reduction and the minimum reliability loss (i.e., the code version with the maximum $\Delta\text{time}/\Delta\text{reliability}$). However, if there is no code version that can meet the deadline, to achieve the minimum performance degradation, the one with the minimum execution time is selected. After selecting the suitable code version, the V-f level is scaled down and if needed, the code version is updated until the TDP constraint is met (as shown in Fig. 10).

5 Experimental Setup and Results

Figure 11 shows the experimental setup and evaluation framework. Experiments were conducted by the use of a system-level many-core simulator developed in the C/C++ language. Accurate power and performance (execution time) parameters of applications and underlying hardware were provided for the simulator through processor synthesis, logic simulation, and power estimation. To do this, the Synopsys Design Compiler and a TSMC 45 nm technology file were used to synthesize a VHDL implementation of a LEON3 processor core [2]. Different benchmark applications of *MiBench* [4] (listed in Fig. 4) were used, and multiple compiled code versions for each application task were generated by the use of a reliability-aware compiler of [8, 9]. ModelSim was used for logic simulation to acquire execution time (clock cycles) and activity factors for each compiled code versions of each application. Power estimation was done using the Synopsys Power Compiler with the process synthesis and logic simulation outputs.

As another input for the simulator, the process variation maps were generated through SPICE simulations. The frequency and leakage power variations were modeled through simulating a 13-stage ring oscillator containing *F*O4 inverters based on two-input NAND gates (like in [11, 12, 15]). To implement DVFS for

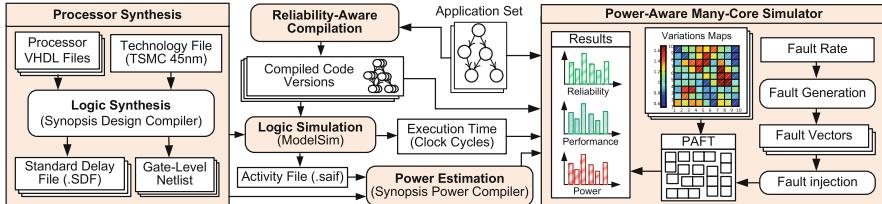


Fig. 11 The experimental setup and simulation flow

the processor cores, it is considered that the voltage level can change from 0.72 to 1.23 V, with 0.13 V steps, and the corresponding frequency to the minimum and maximum voltage levels is 490 MHz and 970 GHz, respectively.

For reliability evaluations, multiple fault vectors were generated by a Poisson process where the transient fault rate at different V-f levels was modeled based on Eq. 2 with $\lambda_0 = 10^{-4}$ and $\Delta = 1V$. Also, for the system, two types of reliability requirements were considered: (1) *functional reliability (FR)* where only correct output of the tasks is required and the tasks have no deadline and (2) *functional-timing reliability (FTR)* where both correct output of the tasks and meeting tasks deadlines are required. For FTR, for each task, we considered a deadline between its execution time and $1.5 \times$ its execution time. Considering the stochastic behavior of transient faults, multiple combinations of benchmark applications were executed for 100,000 times (as a Monte Carlo simulation) and reported the average results.

To model chip power budget, different TDP constraints were considered for each chip between 40 and 100% of its maximum power consumption when all cores perform at their maximum V-f level. This determines a wide range of TDP from a high TDP constraint where up to 40% of the cores can perform at their highest V-f level (i.e., at least 60% dark silicon) to no TDP constraint where all of the cores can perform at their highest V-f level (i.e., 0% dark silicon). Also, two types of system workload were considered in the experiments: (1) *high workload*, when the number of ready tasks is more than 50% of available cores and (2) *low workload*, when the number of ready tasks is less than 50% of the number of available cores.

To evaluate the accuracy and run-time efficiency of the *power-aware fault-tolerance (PAFT)* technique in finding solutions at run time, it was compared with the following techniques:

- dsReliM [11]: which uses compile-time software hardening (different reliable code versions) with run-time code version and V-f level selection.
- DRVS [12]: which exploits run-time task-level redundancy through the SEDR, DMRR, and TMR modes (explained in Sect. 3.3) with V-f level selection.
- ILP Solver: which exploits both compile-time software hardening and run-time task-level redundancy with code version and V-f selection. It searches for the

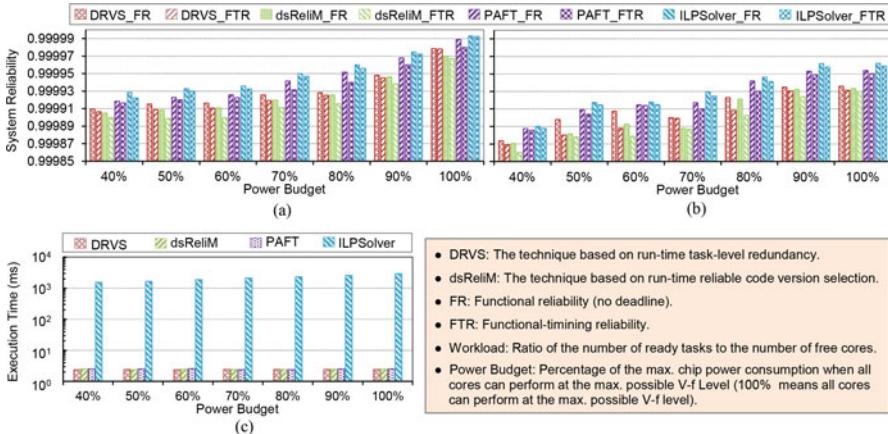


Fig. 12 Reliability and execution time for different power-reliability management techniques. (a) Reliability under low workload. (b) Reliability under high workload. (c) Execution time. Adapted from [15]

optimum solution through ILP solving. For this system, Gurobi¹ was used as a well-known ILP solving tool.

The results of the reliability and execution time efficiency evaluations are shown in Fig. 12. From this figure, the following observations can be made:

- All four techniques achieve higher reliability when there is no timing constraint for the tasks (denoted by FR in Fig. 12) compared to the case when the tasks have a timing constraint (denoted by FTR in Fig. 12). This is because when tasks have no deadline, a higher task-level redundancy (more task copies) can be considered for the execution of the tasks, resulting in a higher reliability. In addition, highly reliable code versions even with a high execution time can be executed. However, when there are timing constraints only the code versions that can satisfy the timing constraints can be selected. For the same reasons, similar results are obtained for higher system workloads (see Fig. 12b).
- All four techniques provide higher reliability when the chip power budget increases from 40 to 100% of the maximum chip power consumption. This is because more cores can be powered-on and higher redundancy levels can be leveraged for more task executions. In addition, highly reliable code versions even with higher power consumption can be executed.
- From the viewpoint of accuracy, reliability levels provided by PAFT deviate far less than one order of magnitude from the optimum reliability provided by ILP Solver, while the execution time of PAFT is up to 1680× less than the execution

¹<http://www.gurobi.com/>.

time of ILP Solver for an 8×8 cores chip (Fig. 12c shows the average execution time for chips with 4×4 , 6×6 , and 8×8 cores).

- As Fig. 12c shows, the execution time for PAFT is up to 3% higher than the execution time of DRVS and dsReliM, while it achieves at least one order of magnitude more reliability. This is because PAFT leverages both task-level redundancy and code version and V-f selection to discover better tradeoffs between reliability, power, and performance.

6 Conclusion

This chapter presents a power-aware fault-tolerance technique (PAFT) that jointly accounts for transient faults, process variations, and the TDP constraint in multi-/many-core chips. It synergistically exploits different reliability techniques, software hardening modes, and V-f levels at run time for power-reliability management. The problem was modeled as a constrained 0–1 integer linear program (ILP), and a computationally lightweight yet efficient heuristic-based technique for solving the problem was proposed. Results have shown that compared to an ILP solver tool, PAFT deviates far less than one order of magnitude in terms of reliability efficiency while seeding up the reliability management decision time by a factor of up to 1680. PAFT also provides at least one order of magnitude reliability improvement under different TDP constraints when compared to the systems that use either hardware reliability techniques or software hardening modes while increasing the execution time less than 3%.

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500—spp1500.itec.kit.edu).

References

1. Brooks, D.M., Dick, R.P., Joseph, R., Shang, L.: Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro* **27**(3), 49–62 (2007). <https://doi.org/10.1109/MM.2007.58>
2. Gaisler, C.: LEON3 processor. <http://www.gaisler.com/index.php/products/processors/leon3>. Accessed 07 June 2019
3. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. CAD Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
4. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of the Workload Characterization, IEEE International Workshop, WWC’01, pp. 3–14. IEEE Computer Society, Washington (2001). <https://doi.org/10.1109/WWC.2001.15>

5. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, Part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October, 2011, pp. 69–78 (2011). <https://doi.org/10.1145/2039370.2039384>
6. Henkel, J., Bauer, L., Zhang, H., Rehman, S., Shafique, M.: Multi-layer dependability: from microarchitecture to application level. In: The 51st Annual Design Automation Conference 2014, DAC'14, San Francisco, June 1–5, 2014, pp. 47:1–47:6 (2014). <https://doi.org/10.1145/2593069.2596683>
7. Rangan, K.K., Powell, M.D., Wei, G., Brooks, D.M.: Achieving uniform performance and maximizing throughput in the presence of heterogeneity. In: 17th International Conference on High-Performance Computer Architecture (HPCA-17 2011), February 12–16 2011, San Antonio, pp. 3–14 (2011). <https://doi.org/10.1109/HPCA.2011.5749712>
8. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Reliable software for unreliable hardware: embedded code generation aiming at reliability. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October, 2011, pp. 237–246 (2011). <https://doi.org/10.1145/2039370.2039408>
9. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. *IEEE Trans. CAD Integr. Circuits Syst.* **33**(11), 1597–1610 (2014). <https://doi.org/10.1109/TCAD.2014.2341894>
10. Salehi, M., Ejlali, A.: A hardware platform for evaluating low-energy multiprocessor embedded systems based on COTS devices. *IEEE Trans. Ind. Electron.* **62**(2), 1262–1269 (2015). <https://doi.org/10.1109/TIE.2014.2352215>
11. Salehi, M., Shafique, M., Kriebel, F., Rehman, S., Tavana, M.K., Ejlali, A., Henkel, J.: dsRelim: power-constrained reliability management in dark-silicon many-core chips under process variations. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015, Amsterdam, October 4–9, 2015, pp. 75–82 (2015). <https://doi.org/10.1109/CODES+ISSS.2015.7331370>
12. Salehi, M., Tavana, M.K., Rehman, S., Kriebel, F., Shafique, M., Ejlali, A., Henkel, J.: DRVS: power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In: IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2015, Rome, July 22–24, 2015, pp. 225–230 (2015). <https://doi.org/10.1109/ISLPED.2015.7273518>
13. Salehi, M., Ejlali, A., Al-Hashimi, B.M.: Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. *IEEE Trans. Parallel Distrib. Syst.* **27**(5), 1497–1510 (2016). <https://doi.org/10.1109/TPDS.2015.2444402>
14. Salehi, M., Tavana, M.K., Rehman, S., Shafique, M., Ejlali, A., Henkel, J.: Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems. *IEEE Trans. VLSI Syst.* **24**(7), 2426–2437 (2016). <https://doi.org/10.1109/TVLSI.2015.2512839>
15. Salehi, M., Ejlali, A., Shafique, M.: Run-time adaptive power-aware reliability management for manycores. *IEEE Des. Test* **35**(5), 36–44 (2018). <https://doi.org/10.1109/MDAT.2017.2775738>
16. Shafique, M., Garg, S., Henkel, J., Marculescu, D.: The EDA challenges in the dark silicon era: temperature, reliability, and variability perspectives. In: The 51st Annual Design Automation Conference 2014, DAC'14, San Francisco, June 1–5, 2014, pp. 185:1–185:6 (2014). <https://doi.org/10.1145/2593069.2593229>
17. Shye, A., Moseley, T., Reddi, V.J., Blomstedt, J., Connors, D.A.: Using process-level redundancy to exploit multiple cores for transient fault tolerance. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25–28 June 2007, Edinburgh, pp. 297–306 (2007). <https://doi.org/10.1109/DSN.2007.98>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Our Perspectives



Jian-Jia Chen and Joerg Henkel

Research and development in the last decades have led to a silicon process that has been expected to become inherently undependable in the near future when migrating towards new technologies. The special priority program (SPP) 1500 funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) in 2010–2016 and the Variability Expedition funded by the National Science Foundation (NSF) in 2010–2015 made a joint effort to explore design challenges of *Power Consumption, Reliability, Interference, and Manufacturability* under such a design requirement.

The exploration started with a vision to go beyond simply developing fault-tolerant systems that monitor the device at run-time and react to error detection. Instead, the design should consider error as a design constraint and develop methodologies to achieve *resilience* at the presence of errors. Under such a design principle, error is inevitable and the error rate should be a tradeoff against performance.

This book summarizes the achievements of the SPP 1500 partners, the Variability Expedition partners, and their collaborators. After telling the successful stories in the previous chapters, this chapter provides a summary of our perspectives of the exploration and a short outlook of future.

One important perspective to achieve resilience at the presence of faults is to *quantitatively* define resilience and errors and use the resilience in a *cross-layer* manner. Specifically, the RAP model summarized in chapter “RAP Model—Enabling Cross-Layer Analysis and Optimization for System-on-Chip Resilience” provides a milestone to help annotate how variability related to physical faults can be expressed

J.-J. Chen
TU Dortmund, Dortmund, Germany
e-mail: jian-jia.chen@tu-dortmund.de

J. Henkel (✉)
Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: henkel@kit.edu

at higher abstraction levels. RAP is a result of several working group meetings and collaborative efforts among SPP 1500 partners. It has been also used as a demonstrator in several projects. We believe that RAP is an initial step towards good abstractions that can be used to model the faulty hardware and its impact on the software. It may be possible that the probabilistic information encoded in RAP is not precise enough for further optimizations. We envision that a more flexible and more precise model to correctly quantify the resilience will be needed in the future. It may be a set of models that can be configured depending on the required accuracy level.

One possible way to analyze system-level resilience in a modularized manner is to enable compositional reliability analysis. The support of composition and decomposition is important for modularized analysis and can be used to model uncertainties in both functional and non-functional properties. The modularity provided in chapter “EM Lifetime Constrained Optimization for Multi-Segment Power Grid Networks” extends the existing compositional performance analysis (CPA) (or real-time calculus) to handle reliability. We believe that there is a great potential to utilize the concept in the system design. However, to achieve composition and decomposition, rules to bound the approximation errors of composition and/or decomposition would be needed. The automatic design of efficient and effective rules is essential for compositional reliability analysis.

In many of the results of the SPP 1500 and Variability Expedition partners, cross-layer and interactive optimization has been explored. Unlike the classic multi-layered approach, in which each layer *passively* takes the input from the higher/lower layers, the cross-layer approach applies *active* optimization routines across multiple layers. Since the system-level resilience cannot be optimized unless all the layers are optimized, such a cross-layer approach has been used as interfaces between different layers. An overview of the (coarse-grained) layers and their interactions can be found in Fig. 10 in chapter “RAP Model—Enabling Cross-Layer Analysis and Optimization for System-on-Chip Resilience”.

To validate the research results, *fault injection* through instrumentation, emulation, and simulation has been developed and used. Fault injection is an important routine that should be deployed before fault detection. The computational effort of fault injection can become a bottleneck. Proper models and tools for fault injections are important contributions of the research partners. For example, the FPGA fault injection tool in chapter “Dependability Aspects in Configurable Embedded Operating Systems” can be used to emulate the entire SoC with specific faults. Different fault injection scenarios can be found in chapter “Lightweight Software-Defined Error Correction for Memories”. Despite its importance, to the best of our knowledge, there is no integrated tool that can be used for benchmarking the quality and (intended) consequence of fault injection. Although there have been several attempts to provide an integrated tool from the partners in SPP1500 for different types of fault injection, the diverse scenarios in the cross-layer settings made the integration very difficult. We envision that fault injection tools that can be configured and applied for different layers and scenarios can be developed in the near future so that cross-layer design and optimization can be further modularized and deployed.

When the design considers error as a design constraint, the system has to be *adaptive* to react (or even be proactive) according to the faults and errors to achieve the targeted resilience. Adaptive methods in physical, micro-architecture, architecture (ISA), compiler, and operating systems are explored and discussed. It has been demonstrated in several research results that adaptivity should be applied across layers. For example, the error semantics in chapter “Soft Error Handling for Embedded Systems using Compiler-OS Interaction” in the software development process provides the information in the compilation needed for the operating systems to be adaptive according to faults. Moreover, the annotation of multiple execution versions in chapter “Cross-Layer Dependability: From Architecture to Software and Operating System” provides a means to the run-time system to execute different versions according to the reliability condition. Furthermore, the dependability aspects can be further configured in operating systems as demonstrated in chapter “ASTEROID and the Replica-Aware Co-scheduling for Mixed-Criticality”. We strongly believe that adaptivity is a key insight. However, the reported achievement is based on ad hoc treatments for well-defined scenarios. It will be very practical and impactful to explore automatic adaptivity so that suggestions of proper means can be provided to the designers for achieving high resilience.

The adaptive handling of errors and faults naturally makes the timing behavior dynamic over time. When there is no fault, an embedded system functions correctly with respect to the specified timing. However, when there are faults, the embedded system may not function correctly anymore since some jobs may be aborted or may miss their deadlines. Therefore, it is of importance to explore both functional and timing correctness. If all jobs have to meet their deadlines, the hardware may have to be over-dimensioned. If some jobs can be allowed to miss their deadlines when faults are present, the system designer just has to ensure that all the desired timing behavior can be verified offline. Such dynamic timing requirements can be modeled as mixed criticality. When the system does not suffer from any fault, it is at the low-criticality mode. When the system suffers from some faults, it is promoted to the high-critical mode. Such a treatment has been presented in chapter “Dependability Aspects in Configurable Embedded Operating Systems”. An alternative is to explore the probability (or miss rate) of deadline misses, presented in chapter “Cross-Layer Dependability: From Architecture to Software and Operating System”. Although the above treatments are successful, they are not originally from the resilience perspectives. It remains open whether the timing requirements to achieve system resilience should be treated as the first-class design objective. More specifically, although dynamic timing behavior and requirements are considered, they are not directly related to resilience. Moreover, the tradeoffs of the timing requirement and system resilience in the presence of faults are still in the infant stage and require more research efforts to reach a conclusion.

We believe that the successful stories in the previous chapters and the perspectives presented in this chapter provide cornerstones for the design of dependable systems on unreliable hardware. Based on the foundation established by the partners, designs which consider faults/errors as a design constraint will be continued in different directions, including physical, micro-architecture, architecture (ISA), compiler, and operating systems layers, and, *most importantly*, in a **cross-layer** manner.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Index

A

Accelerator
ACs, 282
costs and improvements for resilience, 265, 268
diversified configurations, 286–287
DNN, 483–485, 490, 491
placed-and-routed, 298
placement freedom, 297
PORT tests, 284
runtime accelerator placement, 293–295
runtime reconfiguration, 280
SIs, 279
STRAP, 291–292, 298
test-SIs, 283
Accelerator configurations (ACs), 282
Adaptive aging-aware compiler, 534–535
Adaptive compilation methods, GPUs
aging-aware compilation, 534–535
GPU workload distribution, 533–534
mitigating aging, 531, 532
NBTI degradation, 531–432
Adaptive compiler, 531, 532
Adaptive cross-layer techniques, *see*
 Dynamically Reconfigurable
 Processors (DRPs)
Adaptive cruise control (ACC) system, 268
Adaptive design, 591
Adaptive modular redundancy (AMR), 184, 196, 199–201
Adaptive reliability, 462
Adaptive run-time manager for soft error
 resilience (ASER), 164, 165
Adversarial attacks, 482, 493–496

Aging

adaptive aging-aware compiler, 534–535
aging-induced SNM degradation, 308, 310
device aging, 311, 410–413
dynamic stress, 278
HCI and BTI, 300
interconnect electromigration (*see*
 Electromigration (EM))
mitigation, 280, 290, 300
monitor, 396, 556
NBTI degradation, 531
physics and circuit/gate levels, 409, 410
in SRAM cells, 305
static stress, 278
stress balancing, 290–291
and voltage-drop resilient flip-flop design,
 345–436
and yield (*see* Yield modeling and
 optimization)
Algorithm Based Fault Tolerance (ABFT), 258
Application-level optimization, 23–24
Application programming interface (API), 105, 106, 175
Application-specific error handling, 47
Application-specific instruction set processors
 (ASIPs), 177
Application-Specific Integrated Circuit
 (ASIC), 59
Application-Specific Knowledge, 156
Approximate computing, 3, 19, 51–53, 432,
 512, 530, 535, 539
Approximate DRAM
 application robustness, 450
 characteristic refresh parameters, 449
 closed-loop simulation flow, 451

Approximate DRAM (*cont.*)

- concept, 449
- data lifetime, 449–450
- DDR3 chips, 450
- DRAM and core models, 451–452
- DRAM power model, 452
- DRAM subsystems, 436
- power consumption and performance penalty, 449
- qualitative retention error behavior, 450
- reliability-energy trade-off, 450
- retention error measurements, 40 4Gbit DRAM devices, 451
- thermal models, 452
- 3D integrated, 449

Architectural-level fault injection, 252

Architecture support

- AMR, 199–201
 - network on chip (NoC), 196
 - NoC virtualization, 196–197
 - protection switching, 197–199
- Aspect-Oriented Programming (AOP), 86**
- ASTEROID Project**
- architecture, 58, 59
 - characterization, 76
 - complex applications, 57
 - error detection, 58
 - error model, 65
 - error propagation, 57
 - fork-join tasks, 77
 - independent tasks, 77–78
 - MiBench applications, 75–76
 - mixed-critical applications, 58
 - off-chip real-time communication, 59
 - offsets, 66–67
 - OS-assisted replication, 59
 - soft errors, 57, 58
 - sufficient independence, 57
 - synthetic workload, 78–81
 - system model, 63
 - task model, 63–65

Asymmetric write error, 511, 512

Asynchronous DRAM refresh (ADR), 104

Automatic repeat request (ARQ), 59

Automatic test pattern generation (ATPG) tool, 282

Automotive safety integration levels (ASIL), 249

Avionics, 435

B

Backdoor attacks, 493, 495, 496

Backdoored neural networks (BadNets), 495

Batteries

- basic unit, 551
- capacity, 552
- discharging process, 552
- electrochemical cell, 552
- EVs, 551
- lithium-ion battery, 550

Battery aging, 554

- ampere-hour throughput, 554
- calendar and cycling, 554
- capacity loss, 555

Biased voltage scaling (BIVOS), 51

Bias temperature instability (BTI), 182, 192, 247, 278, 305, 386, 411–413

Binary decision diagrams (BDD), 466, 469

Bit flip error manifestation, 2, 4, 7, 16

Built-in self-test (BIST)

- cache access control flowchart, 328
- conventional, 326
- iterative BIST technique, 327
- VLSI system on chip, 326
- voltage scalable mitigation technique, 324

C

Cache memories, 309, 325, 329, 506, 507, 514, 516

Cell-based design, 336

Checkpointing, 37, 74, 171, 175

Chip-Level Redundant Threading (CRT), 148

Circuit aging, 272, 412

Circuit-level results

- aging, 358
- Leon3 flip-flops, 359
- processor, 359, 360

Clockwise Y-chart, 529–530

C2MOS flip-flop

- characteristics, 353
- LH and HL paths, 355
- optimization, 353
- performance, 356
- proposed method, 355
- sense, 355

CMOS technologies, 2, 311, 410

Coarse-grained reconfigurable architectures (CGRA), 123, 124

Commercial off-the-shelf (COTS), 86

Communication reconfiguration, 197–199

Communication systems

- implementation efficiency, 437
- wireless, 436–437

Communication virtualization, *see*

- Multiprocessor systems on chip (MPSoCs)

- Compiler-based fault injection, 252
Compiler-OS interaction
 approximate computing approaches, 51–53
 class of applications, 44
 control-based systems, 48–50
 error handling, 44, 47
 error impacts, 48
 evaluation, 44
 fault tolerance, 33
 FEHLER system, 35–37, 42–43
 flexible error handling, 47
 generic error correction, 46
 injection rates, 45
 microbenchmarks, 43
 quality assessment tool, 45
 reliable memory, 46, 47
 selective error correction approach, 43
 semantic annotations, 35–37
 semantics of errors, 33–35
 static analyses, 39–42
 timing behavior, 37–39
Compositional analysis, 461–462
Compositional Performance Analysis (CPA), 59
Compositional reliability analysis (CRA)
 approaches, 461
 BDD, 466, 469
 corrective postprocessing, 465–467
 models and methods, 464
 and RAL, 464–466
 temperature-reliability adapter for
 MPSoCs, 465–467
Configurable Logic Blocks (CLBs)
 combinatorial and sequential logic
 functions, 281
 configuration matrix, 286
 max diversification condition, 287
 module reliability, 289
 RIF, 290
 TC tests, 284
 test configurations, 285
Constrained-optimization, 590
Context Adaptive Variable Length Coding
 (CAVLC), 155
Continuous technology scaling, 432, 458, 505
Control-based systems, 48–50
Controller design, 546
Control performance, 48, 560, 561
Critical errors, 35
Critical Instant Event Model, 72
Critical path monitor, 393
Cross-layer
 ASTEROID approach, 30
 automotive industry, 29
 DanceOS approach, 30
 design, 590
 embedded real-time systems, 29
 embedded software, 29
 error semantics, 29
 fault-tolerance hardening process, 30
 non-annotated variables, 30
 non-functional properties, 29
 optimization, 436
Cross-Layer Early Reliability Evaluation
 for the Computing Continuum
 (CLRECO) project, 6
Cross-Layer Exploration for Architecting
 Resilience (CLEAR)
 abstraction layers, 257
 costs vs. SDC (DUE) improvements,
 265–267
 error injection, 265
 execution time evaluation, 258
 exploration, 261
 framework, 257
 general-purpose processor cores, 264
 hardware designs, 257
 physical design evaluation, 258
 reliability analysis, 257–258
 resilience library, 258–260
Cross-layer resilience
 abstraction layers, 250
 aging, 386
 applications, 247
 architectural level, 263
 complex memory organizations, 247
 cost-effective application, 263
 CPS, 262
 custom accelerators, 261, 265, 268
 design method, 248
 device aging, 248
 dynamic adaptation, 388
 dynamic tuning, 386
 energy consumption, 387
 energy per clock cycle, 388
 fault injection (*see* Fault injection)
 fixed-hardware micro-controller, 268–270
 hardware design, 248
 low-cost soft error protection, 256
 memory cells, 250
 micro-architectural faults, 263, 264
 monitor circuits, 387, 389–390
 Moore’s law, 247, 248
 on-chip caches, 248
 physical parameters, 386
 protection techniques, 250
 radiation test, 263
 random hardware faults, 249

- Cross-layer resilience (*cont.*)
- requirement, 250
 - safety-critical applications, 247, 249
 - safety-critical digital systems, 250, 256
 - safety standards, 249
 - self-balancing robot system, 270–272
 - sensing mechanism, 389
 - soft errors, 248, 250
 - SoiF, 249
 - SRAM memories, 261–262
 - system components, 248, 251
 - worst-case design methodology, 386–387
- Cross-level analysis, 458, 459, 462, 474
- Cyber-physical systems (CPS), 262
- algorithm, 559
 - battery pack, 546
 - closed-loop dynamics, 548
 - constraints, 548, 557
 - crowding distance calculation, 558, 559
 - design flow, 546
 - discrete-time system, 547
 - FCC, 546
 - feedback control application, 547
 - feedback controller, 548
 - hardware platforms, 557
 - in households, 545
 - non-convex optimization, 558
 - non-dominated points, 558
 - optimization problems, 557–558
 - plant dynamics, 547
 - processor aging, 546
 - QoC metric, 545
 - reliability, 546
- D**
- Dark silicon reliability management (dsReliM), 576
- Data memory, 220–221
- Data recovery policy, 212–213
- Data similarity, 212–213
- Deadline-Miss Probability (DMP), 142, 153–155
- Deadline-Miss Rate, 142
- Debug-based fault injection, 251
- Deep learning (DL)
- Litho-GPA, 424
 - reliability and security vulnerabilities, 482
 - reliable (*see* Reliable deep learning)
 - robustness, 481
 - secure (*see* Secure deep learning)
- Deep neural network (DNNs)
- accelerator, 483–484
 - autonomous driving, 479
 - CNN, 483
- convolutional layer, 483
- DL-based methods, 479
- hardware accelerators, 483–485
- multi-layer perceptron, 483
- neural network, 482
- resilience
- to memory faults, 488–489
 - to permanent faults, 486–487
 - to timing faults, 488
- robustness, 497
- security attacks
- adversarial perturbation attacks, 493–494
 - backdoor attacks, 495
 - smart healthcare, 479
 - systolic array, 484, 486, 488
- Defense, 482, 496
- Delay-area trade-off, 358
- Delay-based monitor circuits
- aging monitor, 396
 - characteristics, 390
 - critical path monitor, 393, 394
 - gate-leakage current, 395
 - logic gates, 390
 - pMOSFET OFF current, 397
 - sub-threshold leakage monitor, 396
 - temperature monitoring, 397–398
 - threshold voltage monitor, 394–395
 - topology, 390
- Delay-based on-chip monitor design
- delay model, 392
 - parameters, 390
 - topology, 391
- Delay-based sensing
- cell-based design, 399–400
 - edge detection based system, 400
 - monitoring circuit, 400
 - system interfacing, 400
- Delay-leakage trade-off, 357–358
- Delay monitor, 400
- Dependability
- instruction scheduling, 144
 - reconfigurable architecture, 280, 281
 - selective instruction redundancy, 144–145
 - software transformations, 142–144
 - system software
 - functional and timing, 145–147
 - multi-core systems, 147–150
- Dependability aspects, *see* Embedded operating systems
- Dependable CAVLC Hardware Architecture, 156
- Dependable CAVLC processing flow, 156–157

- Dependable embedded systems, *see* Resilience
articulation point (RAP) model
- Dependable Embedded Systems Priority
Program, 436
- Dependable systems on unreliable hardware,
437
- Design constraint, 591
- Design optimization, 561
- Design space exploration (DSE), 167, 304,
458–464, 467, 474, 538
- Design-time code selection
overview, 580
performance, 580
reliability, 579
- Detected-but-uncorrectable errors (DUE), 207,
258
- Device-level state, 131–133
- Device under test (DUT), 20
- Digital dynamic power meter (DDPM), 398
- Directed acyclic graphs (DAGs), 149
- Direct impact of temperature, 191
- Discrete cosine transformation (DCT), 190
- Distributed Multi-Threaded Checkpointing
(DMTCP), 175
- DNV memory
API functions, 105, 106
atomic blocks, 105, 108
crashes, 106
data layout, 108, 109
data validation, 108
durability on demand, 106
ECC, 107
error detection, 107
error vector, 108
evaluation, 109–111
power failure, 106
PSU, 106
random distributed seven-bit error, 111
reliable transactions, 107
SECDED Hamming code, 107
software transactions, 111
TxStaging section, 108
- Double/Triple Modular Redundancy
(DMR/TMR), 161
- Dual modular redundancy (DMR), 58, 147,
569
- Duplication with comparison (DWC), 20, 122,
126
- Dynamically reconfigurable processors
(DRPs), 20–21
abstraction layers, 121–122
CPU pipelines, 126–127
cross-layer reliability approaches, 121,
133–135
- device-level state, 131–133
dynamic remapping, 129–130
dynamic testing, 127–129
hardware architects, 121
PE clusters, 125–126
reliability threats, 123
semiconductors, 121
software layers, 122, 123
spectrum, 122
testing reliability, 130–131
- Dynamic compensation, 50
- Dynamic random access memories (DRAMs)
ADR, 104
approximate (*see* Approximate DRAM)
DDR3-1600 DRAM, 76
SDECC, 211
- Dynamic redundancy and voltage scaling
(DRV), 576
- Dynamic remapping, 129–130
- Dynamic resilience actuators, 432, 440–442
- Dynamic stress, 278, 299
- Dynamic testing
banking transactions, 20
characteristics, 23
DRPs, 20–21
DUT, 20
DWC, 20
feasibility region, 22
probability, 20
TMR, 20
- Dynamic timing requirements, 592
- E**
- EarlyWriteBack (EWB), 168
- Electric motor control
DC voltage, 549
state-space system modelling, 550
- Electromigration (EM)
checking, 416
failure effect, 368
failure time, 413
hydrostatic stress, 367
immortality check method
constraints, 375, 376
objective function, 374
- multi-segment wire, 366
- optimization flow, 416
- P/G optimization
linear programming technique, 376
restriction factor, 377
Taylor's expansion, 376
technique, 366
wire-sizing algorithm, 378

- Electromigration (EM) (*cont.*)
 power EM modeling, 413–415
 power grid wires, 365
 power supply synthesis, 365
 reliability, 365
 SEM image, 413
 single EM modeling, 415–416
 steady state, 368
 straight 3-terminal wire, 369
 two-stage signal EM hotspot detection, 416
 void formation, 368
 voltage, 369
- Electronic system level (ESL), 461
- Embedded Configurable operating system (eCos), 96
- Embedded field programmable gate array (eFPGA), 21
- Embedded operating systems
 abstraction layers, 87
 AspectC++, 96, 97
 compile-time introspection, 97–98
 control flow, 98–99
 cross-layer techniques, 119
 data access, 98–99
 design constraints, 117
 eCos-kernel data structures, 99
 EDM/ERM variants, 99
 evaluation, 93–95, 100
 fault-avoiding operating system, 89–90
 fault-detecting operating system, 91–93
 fault injection (FI) experiment, 99
 generic object-protection mechanisms, 100–101
 Hamming code, 96
 hard-and software stack, 118
 hardware designs, 85
 lifetime reliability, 118–119
 memory errors, 118
 MPSoCs, 118
 non-functional properties, 119
 object-oriented programming, 96
 online health monitoring, 118
 OS kernel data structures, 96
 protection effectiveness and overhead, 101–103
 reliability, 95
 soft-error reliability, 118–119
 software-based redundancy techniques, 86
 software-based soft-error detection and correction, 87–88
 software-stack hardening, 86
 state consistency, 88
 supply voltages, 117
 system components, 117
- system-software stacks, 96
- Embedded systems, *see* Power-aware fault-tolerance
 abstraction layers, 140
 adaptive soft error handling, 150–151
 application-specific dependability, 155–157
 dependability (*see* Dependability)
 dependability-driven adaptive run-time system, 140
 dependability modeling, 141–142
 DMP, 153–155
 dynamic real-time guarantees, 151–153
 estimation approaches, 139, 141–142
 multiple system layers, 139
 offline and online optimizations, 140
 real-time systems, 139
 reliability and resilience modeling, 139
 soft error mitigation techniques, 139
- EM lifetime constrained optimization
 interconnect, 380
 P/G wire sizing, 378
 wire disconnection, 379
- Emulation-based fault injection, 251
- Energy consumption, 323, 436, 488, 509–511, 519, 521, 522
- Error correcting codes (ECCs), 167, 207
- Error detection and recovery mechanisms (EDMs/ERMs), 87
- Error detection sequential (EDS) circuits, 438
- Error Propagation Index (EPI), 15
- Error rate, 509–511
- Error resilience
 ASER, 164, 165
 DNNs, 486
 resilience actuators, 441, 442, 445
 soft error resilience (*see* Soft error resilience)
- Error-resilient architecture, 436
- Error tolerance, 118, 162, 529, 530, 572
- Experimental setup and evaluation framework, 583–584
- Extrapolated absolute failure count (EAFC), 101
- F**
- Failure Mode and Effects Analyses (FMEAs), 59
- FAME Runtime Environment (FAMERE), 42–43
- Fault abstraction, 25
- Fault and reliability models
 functional reliability, 568
 simulation setup, 552

- soft errors, 567
software's vulnerability, 567
temporal vulnerabilities, 567
timing, 568
transient faults, 567
- Fault-aware Microvisor Environment (FAME), 37
- Fault-aware pruning (FAP), 489–490
- Fault-aware pruning + training (FAP+T) technique, 490
- Fault injection, 590
- flip-flop-level simulation, 254–255
 - ISS mode, 254–255
 - methods, 251–252
 - multi-level, 253–254
 - simulation-based, 252
 - SRAM memories, 255–256
 - uncore components, 255
- Fault-tolerance, 30, 43, 288, 289
- Fault trees, 466
- Federated Scheduling, 149
- Field programmable gate array (FPGA), 123, 247
- accelerators, 535
 - chip hardware, 251
 - CLBs, 281
 - DRPs, 20–21, 123
 - embedded (eFPGA), 21
 - execution time, 258
 - mapping, 131
 - OpenCL execution model and mapping, 536–537
 - PSMs, 281
 - reconfigurable fabric, 280–281
 - source-to-source compiler, 537–539
- Figures of merit, 190–191
- Finite state machines (FSMs), 197
- Flexible Error Handling Module, 34, 125
- Flip-flop-level fault injection, 252
- Flip-flop optimization
- C2MOS, 340
 - degradation conditions, 343
 - designed circuit, 342
 - digital design, 337
 - LH and HL, 338, 356
 - optimization results, 356
 - parameters, 339
 - processor design, 338
 - runtime variation, 339
 - S-BTI and voltage-drop, 342–343
 - setup-time, 338
 - simulation, 338
 - temporal and spatial temperature, 341
 - VLSI circuits, 337
- voltage-drop, 340
- Flip-flop resiliency optimization
- boundaries, 346
 - delay function, 346
 - flow, 347
 - transistors, 347
 - voltage, 348
- Forward error correction (FEC) techniques, 437
- Frequency scaling, 177, 236
- Full charge capacity (FCC), 546
- Full-processor vulnerability factor (FPVF), 171, 172
- Full timing guarantees, 153
- Fully Depleted Silicon on Insulator (FDSOI) processes, 122
- Functional correctness, 2, 15, 16, 139, 141, 150, 157, 281
- Functional units (FU), 124
- Function Resilience model, 15, 141
- Function Vulnerability Index (FVI), 14
- G**
- Generative Adversarial Networks (GANs), 496–498
- Generic Object Protection (GOP), 96, *See also* Embedded operating systems
- Graceful degradation, 122, 123, 199, 200, 280
- H**
- Hardening embedded processors, 165, 166
- Hardware-based fault injection, 251
- Hardware variations, 531
- Hash-Based Incremental Checkpointing Tool (HBICT), 175
- Heterogeneity
- abstraction layers, 163, 164
 - fault-tolerant
 - cache hierarchy, 167–168
 - design-and run-time methodology, 164, 165
 - hardening embedded processors, 165, 166
 - last-level caches, 166–167
 - microprocessors, 164
 - hardening modes, 163
 - hardware and software layers, 162
 - memory components, 177
 - reliability-heterogeneous architectures, 162
 - reliability techniques, 161, 162
 - reliability threats, 161
 - run-time management system, 162

Heterogeneity (*cont.*)
 run-time systems
 adaptive hardware techniques, 176
 ASIPs, 177
 multi/many-core processors, 176
 techniques, 176
 safety-critical applications, 161
 software layer, 161
 superscalar processors (*see* Superscalar processors)
 transistor fabrication, 161
 Heterogeneous reliability modes, 172–174
 High-critical tasks (HCT), 200
 High-level synthesis (HLS), 265, 537
 High Speed Downlink Packet Access (HSDPA), 438, 440
 Hot carrier injection (HCI), 118, 122–123, 134, 192, 247, 278, 411–412
 Hungarian Algorithm, 149
 Hybrid automatic repeat request (H-ARQ), 438, 439
 Hybrid RMT-Tuning technique, 147

I

Instruction Error Masking Index (IMI), 15, 141
 Instruction Error Propagation Index (EPI), 141
 Instruction memory, 218–220
 Instruction set architecture (ISA), 165
 Instruction Set Simulation (ISS), 23
 Instruction Set Simulator (ISS), 252
 Instruction vulnerability
 application softwares, 12, 13
 hardware components, 12
 hardware layer, 13
 microarchitectural components, 14
 microarchitectural layer, 15
 modeling approaches, 16
 processor components, 14
 real-time systems, 15
 software layer models, 13, 15
 temporal and spatial vulnerabilities, 14, 15
 types of errors, 12
 Instruction Vulnerability Index (IVI), 13–14, 141
 Integrated circuits, 436
 Interconnect
 failure mode, 373
 transient EM analysis, 373
 Interference, 567, 589
 Intermediate representation (IR), 270
 International Organization for Standardization, 2
 Inter-process communication (IPC) level, 86

J

Join-Point Template Library (JPTL), 98

K

Kernels, 279
 Korhonen’s equation, 370

L

Leakage monitor, 396
 LEON processor, 280
 Lifetime reliability (LTR) model
 device-level, 237
 levels of models, 237
 metal and dielectric materials, 237
 metal atoms, 237
 optimization, 237–238
 Weibull distribution, 237
 Lithography, 420–424
 Long Term Evolution (LTE), 438, 440, 443, 444
 Look-up table (LUT), 401

M

Machine learning (ML)
 DNNs, 479–480 (*see also* Deep neural network (DNNs))
 effective, 423
 embedded systems, 480
 lithography hotspot detection, 421–422
 model trustworthiness, 423
 and pattern matching, 420
 reliability threats, 481
 robustness, 481
 security threats, 481–482
 techniques, 479
 Max diversification condition, 287
 Mean time to failure (MTTF), 5, 236
 LTR, 237
 STRAP, 291–292
 stress accumulation, 291
 system-level modeling tool, 237
 threshold voltage, HCI-related stress, 278
 transistor, definition, 278

Memories

DRAMs (*see* Dynamic random access memories (DRAMs))
 ECC protection, 438
 iterative MIMO-BICM receiver, 443
 MIMO-BICM system, 444
 power consumption, 444

- resilience actuators, 445
SRAMs (*see* Static random-access memories (SRAMs))
6T memory cells, 445
8T memory cells, 445
Meta heuristic, 474
Micro-architectural simulators, 252
Microelectronic variability, 539
Microprocessors, 164, 166, 177, 185, 186, 188, 222, 331, 337
MIMO system
 complexity shifting, 447–448
 detection quality, adjustment, 446
 dynamic resilience techniques, 445–446
 error correction capability, 448–449
 external LLR manipulations, 446
 generic architecture, iterative MIMO-BICM receiver, 443
 hardware operating point, 446
 iteration control, 446–447
 MIMO-BICM system, 443, 444
 multiple-antenna systems, 442, 443
 resilience actuators, quantitative comparison, 445
 system memories, 444
 wireless communication systems, 442–443
Mis-corrected errors (MCEs), 211
Mixed-criticality
 co-scheduling approach, 60
 critical components, 61
 error recovery strategies, 61
 gang scheduling, 62
 implicit synchronization, 61
 levels of protection, 61
 mapping replicated tasks, 61–62
 operating system service, 61
 pipeline fingerprinting, 61
 set of mechanisms, 61
 software execution, 60
 SPP, 60
 TDM-based scheduling, 62
Mixed Redundant Threading (MRT), 149
Model chip power budget, 584
Model-hardware correlation
 circuit techniques, 401
 frequencies, 403
 linear equations, 402
 monitoring capability, 402
 parameter extraction methodology, 401–402
pMOSFET, 403
Modeling Function Resilience, 15
Modeling Reliability under Variation (MoRV) project, 6
Modern FPGAs, 281
Modern wireless communication standards, 438
Module diversification
 diversified configurations, 286–287
 experimental evaluation, 288–290
 generation algorithm, 287–288
 RIF, 290
Monte Carlo simulations, 128
MPSoC temperature
 abstraction layers, 184
 electrical properties, 181
 indirect impact, 182, 183
 management layer, 184
 mechanisms, 184
 physical level, 183
 signal-to-noise ratio, 183, 184
 SRAM memory cell, 181, 182
 system management, 184
 thermal management techniques, 183
 transistors, 182
 two-fold impact, 182
Multi-bit upsets (MBU), 7, 9, 255
Multi-cores, 145
Multi-objective optimization strategy, 194
Multiple-antenna systems, 442, 443
Multiple input signature register (MISR), 284
Multi-processor systems on a chip (MPSoCs)
 abstraction layers, 235
 cross-layer, 181, 201–203
 energy-efficient task replication method, 235
 integrated GPUs, 233
 LTR, 235–237
 multicore processors, 233
 permanent faults, 234
 physical modeling, 181
 real-time applications, 234
 reliability-aware techniques, 234
 reliability requirements, 234
 SER, 236–237
 system level
 aging, 192–193
 direct impact of temperature, 191
 figures of merit, 190–191
 temperature (*see* MPSoC temperature)
 thermal cycling, 234, 235
 thermal measurement, 187–189
 thermal simulations, 185–187
 transient faults, 234
Multi-threading, 557

N

- Nanometer CMOS variability, 2
 Nano-scale CMOS circuits, 277
 Near threshold computing (NTC)
 aging and variation-induced SNM degradation, 317
 cache designs, 304
 cache organization
 and SER FIT rate, 320–321
 and SNM degradation, 320
 on system FIT rate, 319
 cross-layer reliability estimation framework
 SER estimation, 312–315
 SNM degradation estimation, 311–312
 system-level FIT rate, 309–311
 experimental setup, 316–317
 interdependence and combined effects, 307–308
 overall energy-saving analysis, 6T and 8T caches, 323
 reliability-aware optimal cache organization, 322
 soft error rate analysis, 317–319
 SRAM cells
 aging effects, 305
 process variation, 305–307
 soft error rate, 307
 system-level reliability, 304
 technology scaling, SRAM reliability, 308–309
 voltage scalable memory failure
 BIST (*see* Built-in self-test (BIST))
 block size selection, 325
 cache block size, 325
 energy and performance evaluation, 330
 error tolerant block mapping, 328–329
 SNM values, 324
 variation-aware voltage scaling analysis, 329
 workload effect analysis, 317
 Negative-bias temperature instability (NBTI), 2, 118, 123
 aging, 386, 531
 degradation, 531–532
 and HCI-related aging, 412
 online health monitoring, 118
 in PMOS transistors, 531
 sensors, 534
 Network on chip (NoC), 59, 196, 236
 Non-functional properties, 17, 29, 94, 119, 590
 Nonlinear programming, 375
 Non-silent data corruption (NSDC), 211
 Non-volatile memories (NVM), 88, 506, 511

O

- Object-oriented programming model, 96
 On-chip memories, 123, 164, 166, 399–401, 510
 On-chip monitor, 385
 Online test
 ATPG tool, 282
 experimental evaluation, 284–286
 generation and runtime scheduling, 282–283
 MISR, 284
 on-demand PORT, 283
 PORT (*see* Post-configuration online tests (PORT))
 PRET (*see* Pre-configuration online tests (PRET))
 system functionality, 281
 TC tests, 284
 test configurations for CLBs, 284, 285
 Online Test Strategies for Reliable Reconfigurable Architectures (OTERA), 280
 OpenCL execution model, 536–537
 Open Systems Interconnection (OSI) model, 2
 Optimal Priority Assignment (OPA), 152
 Optimization
 CRA, 465
 MPSoC, 461
 reliability analysis, 460–461
 uncertainty-aware, 462, 470–473
 Out-of-order, *see* Superscalar processors

P

- Partitioned Strict Priority Preemptive (SPP), 60
 Peak signal-to-noise ratio (PSNR), 45–46, 190
 Performance compensation, 50
 Permanent faults
 FAP, 489–490
 FAP+T technique, 490
 Persistent memory (PM), 86
 durability on demand, 104
 reliable transactions, 104
 system model, 104–105
 Peukert's law, 553
 Phase Change Memory (PCM), 104
 Post-configuration online tests (PORT)
 effectiveness, 284
 frequencies, 285
 functional tests, 282
 module diversification, 286
 on-demand, 283

- performance loss and worst case test
 latency, 286
 test flow, 283
- Potentially critical tasks (PCT), 200
- Power-aware fault-tolerance (PAFT) technique, 578, 586
- Power consumption
 circuit switching activity, 567
 static power, 566
- Power-efficiency, 155
- Power grid network
 DC effective, 366
 mesh-structured P/G network, 366
 multi-segment wires, 366
 nodal voltages, 367
 power supply network, 367
- Power-management techniques, 389
- Power monitor, 582
- Power-reliability management, 573, 576, 585, 586
- Power-reliability-performance management, 576
 chip power constraint, 577
 code version constraint, 578
 cores power constraint, 578
 design-time code selection, 579–580
 DRV5, 576
 dsReliM leverages, 576
 optimization goal, 577
 system reliability, 577
 tasks timing constraint, 578
 V-f level, 578
- Power-reliability-performance tradeoffs
 at hardware level
 application reliability, 571
 core-to-core frequency, 570
 power consumption, 571
 V-f levels, 571
 at software level
 applications, 572
 power consumption, 572
- Power supply unit (PSU), 106
- Pre-configuration online tests (PRET)
 accelerator configurations (ACs), 282
 application performance loss, 284–285
 array-based structural test approach, 282
 effectiveness, 284
 module diversification, 286
 on-demand, 283
 test configurations (TCs), 282
 test flow, 283
- Probabilistic error propagation, 3, 4
- Probabilistic Wallace-tree multiplier (PWTM), 51
- Process Control Modules (PCM), 404
- Processing elements (PE), 21
- Processor aging
 aging estimation, 556–557
 aging mechanisms, 555
 countermeasure, 555–556
 design software, 557
- Processor delay, 361
- Process variation, 147–148
 of NTC circuits, 304
 SER, 6T and 8T SRAM cells, 315
 SNM degradation analysis, 311–312
 in SRAM cells, 305–307
- Profitable semiconductor scaling, 34
- Program counter (pc), 222
- Prohibit and propagation rules, 39
- Protection switching, 197–199
- Q**
- Quality-of-control (QoC) indices, 545
 and battery usage, 560
 constraints, 560
- Quality-of-Service (QoS)
 adaptive, 440
 extended dynamic flow, 441
 standard flow, 440
 wireless transmission system, 438, 440
- R**
- Radiation-induced soft errors, 247, 250, 263, 272, 304, 307, 309, 313, 435
- Random Telegraph Noise (RTN), 192
- Rate capacity effect
 cell voltage, 552
 concept, 552
 discharge current, 553
- Read-only memory (ROM), 91
- Real-time systems, 154
- Reconfigurable baseline architecture, 280–281
- Reconfigurable monitor, 385
- Redundant multithreading (RMT), 147
- Reliability
 aging, 343
 analysis techniques, 458, 460–461
 BTI-induced aging, 345
 C2MOS flip-flop, 344
 compositional approaches, 461
 cross-level analysis, 458, 459, 462, 474
 embedded systems, 460
 ML, 481
 module diversification, 288, 290
 optimization, 460–461

- Reliability (cont.)**
- post-aging delay, 344
 - reconfigurable baseline architecture, 280
 - reliability-improving techniques, 458
 - RIF, 290
 - SRAMs, 513
 - Reliability abstraction level (RAL), 464–466
 - Reliability and execution time, 587
 - Reliability-aware reconfigurable cache architecture, 166–167
 - Reliability block diagrams (RBDs), 5
 - Reliability-energy tradeoff, 450, 510
 - Reliability-heterogeneous cores, 162
 - Reliability improvement factor (RIF), 290
 - Reliability Information Interchange Format (RIIF), 6
 - Reliability management, 451, 565, 571
 - Reliability modeling, 6, 13, 203
 - Reliability Profit Function (RPF), 165
 - Reliability techniques
 - DMRR technique, 569
 - SEDR, 569
 - spatial correlation, 568
 - TMR, 570
 - total application, 568
 - Reliability-Timing Optimizing Technique (RTO), 148
 - Reliability-Timing Penalty (RTP), 16, 141
 - Reliability type qualifiers, 39
 - Reliable Computing Base (RCB), 30, 34–35, 59, 88
 - Reliable CPS Design for Unreliable Hardware Platforms, 57
 - Reliable deep learning
 - design-time steps, 485–486
 - DNN-based applications, 485
 - permanent faults
 - FAP, 489–490
 - FAP+T technique, 490
 - resilience of DNNs
 - to memory faults, 488–489
 - to permanent faults, 486–487
 - timing error mitigation techniques, 488
 - to timing faults, 488
 - run-time steps, 486
 - timing fault mitigation
 - per-layer voltage underscaling, 492–493
 - TE-Drop, 491–492
 - Re-order Buffer (ROB), 168
 - Resilience actuators
 - algorithmic, 452
 - application-specific, 445, 452
 - dynamic protection mechanism, 441
 - extended dynamic QoS flow, 441
 - hardware operating point, 442
 - low-level hardware techniques, 442
 - power state switching, 444
 - quantitative comparison, 445
 - Resilience articulation point (RAP) model, 255, 437
 - application-and system-level software, 2
 - application-level optimization, 23–24
 - application-level software, 2
 - complex situations, 6
 - cross-layer optimization, 2, 3
 - data vulnerability analysis and mitigation
 - binary classification, 19
 - code and data flow analyses, 16
 - context of execution, 16
 - data flow analyses, 17, 18
 - FEHLER system, 18
 - horizontal propagation, 17
 - invalid assignments, 18, 19
 - logic errors, 16
 - memory bit flips, 16
 - non-functional properties, 17
 - propagation analysis, 18
 - device level, 4
 - divide and conquer strategies, 2
 - dynamic testing (*see* Dynamic testing)
 - electronic components, 6
 - environmental and operating conditions, 6
 - error probability, 7
 - error transformation function, 5
 - generic model, 6
 - graph theory, 4
 - hardware/software complexity, 2
 - hardware/software system abstraction, 4
 - logic testing, 3
 - MTTF, 5
 - probability, 4
 - reliability analysis approaches, 5
 - semiconductor material and device levels, 3, 4
 - SoC, 2
 - SRAM (*see* SRAM)
 - system design, 7
 - transformation functions, 4
 - Variability Expedition program, 2
 - Resilience of DNNs
 - to memory faults, 488–489
 - to permanent faults, 486–487
 - to timing faults, 488
 - Resistive random-access memory (RRAM), 104
 - Resource management, 280, 432
 - Response-time analysis
 - error recovery, 74–75

- fork-join task, 67–70
independent tasks, 70–74
time interval, 67
- Retention time, 449, 450
- Robustness, 450, 480, 497
- Robustness constraint model, 141–142
- Run-time hardening mode
base solution, 582
ILP solvers, 580
initial hardening mode assignment, 581
initial mapping, 581
overview, 581
PAFT technique, 580
TDP constraint, 582, 583
- Run-time manager, 156
- Runtime reconfiguration, 280–281
- Runtime system
accelerator placement, 293–295
FAMERE, 42–43
PRET and PORT, 282–283
test-SI, 283
- S**
- Safety-critical systems, 88, 117, 248, 249, 256, 460, 557
- Safety of the intended functionality (SoiF), 249
- Scheduling, 154
- Scratchpad memories (SPMs), 221
- Secure deep learning
defences against security attacks
GAN-based framework, 497
noisy visual data, 497
security attacks on DNNs
adversarial perturbation attacks, 493–494
backdoor attacks, 495
- Security
data/IP stealing, 482
data manipulation, 481
definition, 481
denial-of-service, 481
reliability and security vulnerabilities, 481–482
- Selective data redundancy, 156
- Selective flip-flop optimization
aging and voltage-drop analysis, 349–350
gate-level simulation, 350
methodology, 349
VLSI design flow, 349
- Selective flip-flop replacement step, 351
- Self-balancing robot system, 270–272
- Semantic annotation of criticality, 35–37
- Semantics of errors, 33–35
- Settling time, 545, 548, 560
- Side information (SI), 207, 217
- Silent data corruption (SDC), 24, 87, 256, 258
- Simulation-based fault injection, 252
- Simultaneous Redundant Threading (SRT), 149
- Single-bit error correction (SEC), 216, 224
- Single error correcting-double error detecting (SEC-DED), 105, 167
- Single-error detecting (SED), 216, 223–224
- Single event effects (SEEs), 57
- Single event upsets (SEUs), 2
data memory, 220–221
instruction memory, 218–220
- Single point of failure (SPOF), 86
- Soft error, 566, 572
aging-induced SNM degradation, 308
AVF analysis, 310
neutron-induced, 316
radiation-induced, 304
in SRAM cells, 307
- Soft-error reliability (SER), 234
big-little, 240–242
- CPU and GPU fabrication processes, 236
- frequency scaling, 236
- integrated CPU–GPU, 242–244
- optimization, 238–240
- probability, 236
- supply voltage, 236
- task scheduling, 235, 240
- utilization control, 236, 238
- Software-Defined Error Correcting Codes (SDECC)
abstraction layers, 208
analysis, 210
architecture, 211, 212, 221–222
candidate codewords, 208–210
data recovery policy, 212–213
hybrid hardware/software technique, 216
- parity++
application characteristics, 224
architecture, 227–228
cache capacity, 230
correction, 226–227
error detection, 226–227
error protection, 224
experimental methodology, 228–229
Gem5 simulations, 229
generator matrix, 226
Hamming code, 225
Internet-of-Things (IoT), 230
memory capacity, 230
polynomial coding methods, 225

- Software-Defined Error Correcting Codes
 (SDECC) (*cont.*)
 SECDED, 229
 unequal message protection code, 224
 reliability evaluation
 methodology, 213–214
 recovery breakdown, 214–216
 SEUs (*see* Single event upsets (SEUs))
 side information (SI), 209
 soft fault recovery, 222–223
 terms and notation, 208, 209
 UL-ELC, 216–218
 Software error detection with re-execution
 (SEDR), 569
 Software-implemented fault injection, 251
 Software-level fault injection, 252
 Software transactional memory (STM), 86, 105
 Special Instructions (SIs), 279
 Special priority program (SPP), 589–590
 Spike simulator, 213
 Spin Transfer Torque Magnetic RAM
 (STT-RAM)
 alternative solutions, 506
 basic cell structure, 508
 error rate vs. energy consumption trade-off,
 509–511
 FlexRel approach
 effectiveness, 519–520
 energy consumption, 521
 FlexRel-equipped cache, 516
 static energy consumption, 522
 write patterns, 513–515
 non-volatile technology, 516–519
 read disturbance, 506
 read operation, 508
 reliability, 511–512
 retention errors, 506
 susceptibility, 432
 TMR, 509
 write errors, 506
 Spin-transfer-torque magnetoresistive random-access memory (STT-MRAM), 104
 Stability, 8, 16, 17
 State compression techniques, 174–175
 Static analyses, 36
 Static Mapping Library (StML) approach, 130
 Static Pattern-Based Reliable Execution, 50
 Static random-access memories (SRAMs), 105
 accelerated transistor aging, 305
 aging and process variation, 309
 approximate DRAM, 449
 cross-layer approach for soft error resilience, 261–262
 error function, 7
 fast fault injection, 255–256
 heterogeneous cache design, 303
 NTC circuits, 304
 particle strikes, 8–10
 process variation, 305–307
 pull-down transistors (PD), 8
 read delay/WTV, 10–11
 self-balancing robot system, 270–272
 SER rate, 6T and 8T SRAM cells, 314,
 315, 319
 SNM degradation estimation, 311
 soft error rate, 307
 SPICE model, 313
 state variables, 7
 STT-RAM, 432
 supply voltage drops, 11–12
 SVNM, 10
 technology scaling effects, 308–309
 tolerant SRAM cell design, 303
 6-transistor (6T), 7
 8-transistor (8T), 7
 unreliability sources, 304
 Static stress, 278, 298, 299
 Stress-aware placement method (STRAP),
 291–292, 295, 297–300
 Stress distribution, 290, 291, 294, 298, 370
 Structural test, 280
 Success trees (STs), 458
 Superscalar processors
 cloud servers, 168
 data-centers, 168
 experimental setup, 169
 methodology
 FPVF, 171, 172
 heterogeneous reliability modes,
 172–174
 state compression techniques,
 174–175
 vulnerability analysis, 169–171
 Supply voltage fluctuation, 398
 Synopsys design compiler, 361
 System-level management
 task migration, 194–195
 voltage scaling, 193–194
 System level reliability analysis, 463–464
 System model, 63
 System-on-Chip (SoC), 2
 approximate DRAM, 451
 LSI scaling, 389
 RAP model, 437
 System reliability, 577
 Systems with Dynamic Real-Time Guarantees,
 152

T

- Task migration, 118, 193–195, 198, 199, 202
Task model, 63–65
Technology scaling, 8, 203, 308–309, 389, 435, 452, 458, 505
Temperature monitor, 397
Temporal variability, 527
Thermal design power (TDP), 165, 565
Thermal estimation, 184, 187
Thermal measurement, 187–189
Thermal simulations, 185–187
Time Demand Analysis (TDA), 152
Time-Dependent Dielectric Breakdown (TDDB), 192
Time-Division Multiplexing (TDM), 60
Time to failure (TTF) estimation, 366
Timing correctness, 15, 141, 145, 591
Timing error
 per-layer voltage underscaling, 492–493
 TE-Drop, 491–492
Timing faults, 132, 488
Timing Optimizing Technique (TO), 148
Tradeoffs
 hardening modes
 DMRR and TMR techniques, 573
 reliability techniques, 572
 system-wide reliability, 573
 V-f level, 573
 software hardening
 application execution, 574
 reliability and execution time, 574
 reliability-driven software, 573
 TDP constraint, 575
Transient EM-induced stress estimation
 eigenfunctions, 371
 growth phase, 372
 incubation phase, 371–372
 interconnect, 370, 373
 nucleation phase, 370
 nucleation time, 370
Transient faults, 57, 86, 89, 105, 234, 242, 280, 567
Triple modular redundancy (TMR), 20, 58, 122, 125, 147, 435, 570
Type deduction graph (TDG), 41

U

- Ultra-Lightweight Error-Localizing Codes (UL-ELC), 216
Ultra Thin Body and Box Fully Depleted Semiconductor on Insulator (UTBB-FDSOI) technology, 132

Uncertainty

- correlation, 459, 466, 470
 dominance criteria, 470, 471
 embedded systems, 462
 modeling, 467
 system reliability, 459
 system's response and characteristics, 459
 uncertainty-aware optimization, 462–463, 470–474
 uncertainty-aware reliability analysis, 462, 468–470

Uniform voltage scaling (UVOS), 51

V

- Very Large Scale Integr. (VLSI)
design
 aging modeling (*see* Aging)
 modern designs, 409
 robustness, 409
VirTherm3D project, 181
Virtualized network interface controller (VNIC), 197
Virtualized NoC adapter (VNA), 197
VLSI reliability, 409
Voltage, 19, 23, 51
Voltage-drop resilient flip-flop design, 345
Voltage scaling, 193–194
 DNNs, 488
 DRVS, 576
 supply voltage scalability, 329
 system-level thermal management techniques, 193–194
 TED-based approaches, 488

W

- Watchdog timer (WDT), 270
Wireless baseband processing
 communication systems, 436–437
 dynamic behavior, 438–440
 dynamic resilience actuators, 440–442
 EDS circuits, 438
 hardware errors, modeling, 437
 multiple-antenna MIMO systems, 442–443
 reliable transmission, 437
 unreliable hardware, effects, 438
Wireless communication systems, 436–437
Worst-case deviation, 54
Worst-case execution time (WCET), 63, 153
Worst-Case Response Time (WCRT), 61
Write energy, 510, 516, 519, 521

X

- Xilinx Design Language (XDL), 131

Y

Yield modeling and optimization

hotspot detection

conventional simulation-based, 420

data efficiency, 423

lithography detection tasks, 421–422

machine learning, 421

model trustworthiness, 423–424

neural network, 421

pattern matching, 420–421

performance modeling

performance optimization, 419–420

semi-supervised modeling, 418–419

sparse modeling, 417–418