EPFL

Thèse n° 10 353

Energy- and Cost-Efficient VLSI DSP Systems Design with Approximate Computing

Présentée le 25 septembre 2020

à la Faculté des sciences et techniques de l'ingénieur
Laboratoire de circuits pour télécommunications
Programme doctoral en microsystèmes et microélectronique

pour l'obtention du grade de Docteur ès Sciences

par

Reza GHANAATIAN JAHROMI

Acceptée sur proposition du jury

Dr M. Mattavelli, président du jury
Prof. A. P. Burg, directeur de thèse
Prof. N. Wehn, rapporteur
Prof. W. Burleson, rapporteur
Prof. D. Atienza, rapporteur

École polytechnique fédérale de Lausanne

2020

Acknowledgements

First of all, I would like to thank my thesis advisor, Prof. Andreas Burg, for giving me the opportunity to join his laboratory and for his continuous support and guidance throughout my doctoral studies. I deeply admire his ability to provide new paths to solve problems, as well as his ability to create a comfortable environment where people can work together with a great passion. I am very grateful for every technical discussion that we had, which always pushed me to look at problems from different perspectives and to discover new opportunities. Thanks to him and owing to the experience that I had at his laboratory, I have a better scientific mentality and I can more easily solve new problems in my future carrier.

I would like to thank Dr. Marco Mattavelli (EPFL) for acting as president of my PhD exam jury, Prof. David Atienza Alonso (EPFL), Prof. Norbert Wehn (University of Kaiserslautern), and Prof. Wayne Burleson (University of Massachusetts, Amherst) for serving as examiners for my thesis defense. Their comments and feedback improved the quality of this manuscript.

I feel very lucky for having conducted my doctoral studies at Telecommunications Circuits Laboratory (TCL), EPFL. There, I had the chance to meet and work with talented colleges and friends who made the laboratory a very stimulating environment to work in and a very friendly environment to live in! They certainly made and continue making TCL a unique place. For the great time we spent together and for all the technical and non-technical discussions that we had, I would like to thank all my current and former colleges in alphabetic order: Orion Afisiadis, Andrew Austin, Alexios Balatsoukas-Stimming, Pavle Belanovic, Andrea Bonetti, Jeremy Constantin, Matthieu Cotting, Shrikanth Ganapathy, Pascal Giard, Robert Giterman, Georgios Karakonstantis, Andreas Kristensen, Yann Kurzo, Sitian Li, Pascal Meinerzhagen, Christoph Müller, Nicholas Preyss, Lorenz Schmid, Adrian Schumacher, Christian Senning, Adam Teman, Marco Widmer, and Ning Xu. I could not omit thanking the administrative assistant of TCL, Ioanna Paniara, for making everything smooth for me, so that I did not spend any time on administrative issues.

I would like to thank all the Iranian friends that directly or indirectly become part of my PhD studies mainly in Switzerland and also in Iran. We had great times together during many unforgettable occasions, I had their support during many aspects of my daily life, and I

certainly learned how to be a better person from them. Without them, my PhD studies would not have been so colorful. Special thanks to all the friends with whom we shared lunches and coffee breaks at EPFL and spent weekends and vacations together outside EPFL during all these years.

Finally, I would like to thank my family for their patience, support, encouragement, and always being proud of me during my PhD studies while I was living abroad. Special thanks to my parents, Karim my father and my life teacher, and Shahnaz my mother and my love teacher. Having their unrestrained love and unconditional support, I always feel strong and blessed. Special thanks to my little sister Negar, for her love and all the sweet chats that we had during these years, to my big brother Ahmad, for his true support, and to my sister-in-law Hosna, for her constant care and the love that she breathes into the family.

Lausanne, August 26th 2020 Reza Ghanaatian Jahromi

Abstract

Although aggressive CMOS technology scaling into the nanometer regime has allowed the realization of complex high performance systems, unfortunately it has also led to process variation, especially in sub-22 nm geometries. Such variations lead to transient and permanent timing and memory failures threatening the correct functionality of future systems. Currently, designers and manufacturers guarantee fault-free operation by introducing redundancy in the form of voltage margins, conservative layout rules, and extra protection circuit measures, which unfortunately lead to a significant increase in the power and cost of the integrated circuits (ICs) diminishing the gain from technology scaling. The high-throughput data processing demand of current and future communication systems and the high-capacity storage requirement of emerging applications, based on artificial intelligence (AI) and machine learning (ML), however, ask to fully utilize the benefit from technology scaling, which aggravate the issue.

Recent trends in the design of ICs propose to achieve this goal by relaxing strict requirements on accuracy and precision and deviating from 100% error-free computation paradigm by exploiting the inherent resilience of many applications through *approximate computing*. In this spirit, a broad set of techniques is used to mitigate variability and improve efficiency during design-time and run-time and across different abstraction layers. The focus of this thesis is on the algorithm and architectural level. To this end, we study three different directions that address approximate computing for fault tolerance to manufacturing issues. Specifically, we consider the usage of faulty storage elements for optimized cost and power, run-time quality adaptation for low power, and design-time optimization while exploiting the inherent resilience of signal processing applications in all the directions.

Due to the dominant role of storage elements in both the area and power of many modern systems-on-chip (SoCs), they are often the source of an unreliable behavior. We propose design methodologies that allow to drop the requirement of 100% reliable memories during test of ICs with such memories while still guaranteeing an average-quality during operation. We provide a realistic memory fault model and a quality assessment methodology that proposes to treat the application quality as a distribution. We evaluate our proposal for some image processing benchmarks using an embedded system. We further fabricate a testchip for a channel decoder

with unreliable memories and provide an extensive analysis based on the chip measurement results, which serve as a proof to our propositions.

Although process-related variations happen at a slow time-scale, some other type of variations might happen at a fast time-scale. Fast variations often occur at run-time due to a data-dependent behavior of algorithms and are another source of an expensive design margin in addition to process variations. In this regard, we propose a run-time adaptive power reduction algorithm for a low-density parity check (LDPC) decoder that is based on an offline statistical analysis of the decoding iterations and is implemented by employing dynamic voltage and frequency scaling (DVFS). This algorithm trades the error-correcting performance for a required power reduction gain while guaranteeing to keep the performance loss below a predefined margin.

Design-time approximate computing techniques have been used during many years to provide a fast and efficient implementation for signal processing algorithms. In that line, we propose multiple techniques at algorithm, architecture, and physical-implementation level to reduce the complexity and enable a high-throughout for an LDPC decoder with an unrolled architecture. The techniques are based on a finite-alphabet message passing algorithm for the decoder and attribute at reducing the complexity of the arithmetic units, storage elements, and the routing wires of the decoder achieving an ultra-high throughout while resulting in a significant reduction in the area and power consumption of the decoder.

Key Words: Digital VLSI Circuits, Nanometer Nodes, Approximate Computing, Ergodic Process, Quality-Yield Analysis, Embedded Systems, Gain-Cell Embedded DRAMs, LDPC Decoder, 28 nm FD-SOI, Run-Time Variations, DVFS, Design-Time Approximation, Finite-Alphabet Decoder, Unrolled Architecture.

Résumé

Bien qu'une mise à l'échelle nanométrique agressive de la technologie CMOS ait permis la réalisation de systèmes complexes de haute performance, elle a malheureusement aussi entraîné des variations de processus, en particulier dans les géométries inférieures à 22 nm. De telles variations entraînent des défaillances transitoires et permanentes de la synchronisation et de la mémoire qui menacent le bon fonctionnement des futurs systèmes. Actuellement, les concepteurs et les fabricants garantissent un fonctionnement sans défaut en introduisant une redondance sous la forme de marges de tension, de règles de disposition conservatrices et de mesures de protection supplémentaires des circuits, ce qui entraîne malheureusement une augmentation significative de la puissance et du coût des circuits intégrés (ICs), diminuant ainsi le gain résultant de la mise à l'échelle de la technologie. La demande de traitement de données à haut débit des systèmes de communication actuels et futurs et les besoins de stockage à grande capacité des applications émergentes, basées sur l'intelligence artificielle (AI) et l'apprentissage automatisé (ML), exigent cependant d'utiliser pleinement les avantages de la mise à l'échelle technologique, ce qui aggrave le problème.

Les tendances récentes dans la conception des ICs proposent d'atteindre cet objectif en assouplissant les exigences strictes en matière d'exactitude et de précision et en s'écartant du paradigme du calcul 100% sans erreur en exploitant la résilience inhérente de nombreuses applications par le calcul approximatif. Dans cet esprit, un large ensemble de techniques est utilisé pour atténuer la variabilité et améliorer l'efficacité pendant la conception et l'exécution et entre les différentes couches de calcul. Cette thèse se concentre sur le niveau algorithmique et architectural. À cette fin, nous considérons trois directions différentes qui abordent le calcul approximatif pour la tolérance aux fautes dues aux problèmes de fabrication. Plus précisément, nous étudions l'utilisation d'éléments de stockage défectueux pour optimiser le coût et la puissance, l'adaptation de la qualité d'exécution pour une faible puissance, et l'optimisation du temps de conception tout en exploitant la résilience inhérente des applications de traitement du signal en considérant toutes les directions.

En raison du rôle dominant des éléments de stockage dans la superficie et la puissance de nombreux systèmes sur puce (SoCs) modernes, ils sont souvent la source d'un comportement peu fiable. Nous proposons des méthodologies de conception qui permettent d'abandonner

l'exigence de mémoires fiables à 100% lors des tests d'ICs avec de telles mémoires tout en garantissant une qualité dans la moyenne en fonctionnement. Nous fournissons un modèle réaliste de défaut de mémoire et une méthodologie d'évaluation de la qualité qui propose de traiter la qualité de l'application comme une distribution. Nous évaluons notre proposition pour certains standards de traitement d'images utilisant un système embarqué. Nous fabriquons en outre une puce de test pour un décodeur de canal avec des mémoires peu fiables et fournissons une analyse approfondie basée sur les résultats des mesures de la puce, qui sert de preuve à nos propositions.

Bien que les variations liées aux processus se produisent à une échelle temporelle lente, d'autres types de variations peuvent se produire à une échelle temporelle rapide. Les variations rapides se produisent souvent au moment de l'exécution en raison d'un comportement dépendant des données pour la qualité de l'application et constituent une autre source de marge de conception coûteuse en plus des variations liées au processus. À cet égard, nous proposons un algorithme de réduction de puissance adaptatif en cours d'exécution pour un décodeur de code LDPC (Low Density Parity Check) qui est basé sur une analyse statistique hors ligne des itérations de décodage et est mis en œuvre en utilisant la mise à l'échelle dynamique de la tension et de la fréquence (DVFS). Cet algorithme échange une performance de correction d'erreur contre un gain de réduction de puissance requis tout en garantissant de maintenir la perte de performance en dessous d'une marge prédéfinie.

Des techniques de calcul approximatif en temps réel ont été utilisées pendant de nombreuses années pour fournir une implémentation rapide et efficace des algorithmes de traitement du signal. Dans cette optique, nous proposons plusieurs techniques au niveau de l'algorithme, de l'architecture et de la mise en œuvre physique afin de réduire la complexité et de permettre un haut débit pour un décodeur LDPC avec une architecture déroulée. Les techniques sont basées sur un algorithme de transmission de messages en alphabet fini pour le décodeur et permettent de réduire la complexité des unités arithmétiques, des éléments de stockage et des fils de routage du décodeur, ce qui permet d'obtenir un débit très élevé tout en réduisant considérablement la surface et la consommation d'énergie du décodeur.

Mots Clefs: Circuits VLSI numériques, Nœuds de Nanomètre, Calcul Approximatif, Processus Ergodique, Analyse de la Qualité et du Rendement, Systèmes Embarqués, eDRAMs, LDPC, 28 nm FD-SOI, Variations de la Durée d'Exécution, DVFS, Approximation du Temps de Conception, Décodeur d'Alphabet Fini, Architecture Déroulée.

Contents

1 Introduction

Technology scaling has progressed to enable integrated circuits with extremely high density enabling systems of tremendous complexity with manageable power consumption. With the continuation of Moore's law [1] for many years, electronic chips have been able to accommodate the growing performance demand and energy-efficiency requirement of many applications. Unfortunately, in the past decade, we have also seen diminishing returns from the most advanced process nodes in terms of performance and power consumption. Furthermore, emerging applications based on artificial intelligence (AI) and machine learning (ML) with high computing demands expedite the increasing performance requirement and aggravate the gap between the diminishing technology scaling benefits and the increased computing demands. The issue lies in the fact that it has become increasingly difficult to guarantee a reliable operation without pessimistic and thus costly design margins in sub-22 nm geometries due to *process variations*.

Fabrication processes become inaccurate, as geometries are scaled down, and lead to *spatial* and *temporal* variations in transistor characteristics that threaten the correct functionality of circuits [2–5]. For instance, transistor channel dopant fluctuations become more acute in smaller geometries that affect the transistor threshold voltage and the leakage power. Such variations can lead to delay variations in digital logic [3], for instance, and may result in incomplete computations, which can ultimately lead to quality of service (QoS) degradation or system failures.

To understand the failure issue, it is important to recognize the types of variation that ultimately lead to unreliable behavior of the circuit. Device parameters vary between devices on a single chip, *inter-die variations*, from chip to chip, *intra-die variations*, and across operating conditions (circuit age, temperature, and supply voltage). Different sources of variation can be classified according to their time scales. While, fast variations such as supply voltage and

input data variations are in the order of micro and nano seconds, respectively, slow variations such as aging effects are in the order of months [6]. Additionally, some of the variations such as process variations are often not even random, but a deterministic consequence of a random and/or unknown phenomenon during production when considering a given die. For example, random process variations lead to failure of some bits in a memory cell, however, these weak bits are often frozen (stuck-at 0/1) and deterministic after manufacturing.

Traditional approaches for tolerating potential performance degradation or circuit failures due to variations suggest frequency/voltage scaling or transistor up-sizing to create a comfortable margin assuming *worst-case* conditions for each source of variability based on developed models [4, 7, 8]. However, such worst-case conditions are usually rare, and hence, such approaches lead to over-design not allowing the full utilization of performance gains obtained from technology scaling [9]. Such approaches do not only cause an increase in power dissipation, but they also limit the circuit performance which could be unacceptable for systems with demanding performance requirements, such as communication systems or systems for emerging applications such as AI and ML.

In the last two decades, many efforts were devoted to the study of the sources and impact of parametric variations [10, 11]. These efforts led for example to new approaches that try to limit the pessimistic guard bands by building models and tools for performing a more accurate timing analysis [12–15], which is merely a way to reduce margins, but not to avoid them entirely. Moving beyond tighter margins, researchers have tried to further gain from violating safety margins while trying to ensure reliable operation by mitigating the impact of rare potential failures. Such techniques are referred to as *better than worst-case* design [16, 17]. One of the first attempts was *Razor* [18], a processor design from University of Michigan and ARM that is based on dynamic detection and correction of timing failures of the critical paths due to below-nominal supply voltage. On similar lines, a processor from Intel [19] places replica circuits within the critical paths of the pipeline stages for detecting dynamic variations and recovering for the errant instructions.

While the aforementioned efforts have improved the circuit efficiency, their potential gain is diminishing in sub-22 nm technology nodes since they still care about detecting and correcting all errors. In order to alleviate the impact of reliability issues even more, many researchers have recently proposed to abandon the conservative and 100% error-free design paradigm, while exploiting the inherent fault-tolerance of many applications through *approximate computing* to avoid the need for conservative margins. In this new paradigm, the strict enforcement of 100% correct functionality is relaxed trading QoS for better cost (energy, yield) efficiency.

Approximate computing is computing efficiently by producing results that are good enough or of sufficient quality [20]. While this core principle is not new and is shared among many fields, recent efforts are devoted to incorporate this principle at many different design levels and for various applications. Therefor, it is useful to define the key principles of the design using

approximate computing techniques. First, a *quality* metric needs to be specified based on the application and it is critical to develop methods to ensure that acceptable quality is maintained while approximate computing techniques are applied. Second, an approximate computing technique should be able to *dynamically adapt* to the input data or to the context in which the outputs are used. More specifically, an approximate computing technique should be quality-configurable across a range of qualities and not be optimized for a fixed quality. Third, all the computations in an application are not equality important in the output quality and thus they may or may not be subject to approximation [7,21,22]. For example, all the computations in an image processing kernel do not equality contribute in shaping the output response and approximations on the more important computations may lead to a non-negligible output quality degradation. Therefore, a notion of *significant-driven* in the computations should be adopted based on how significantly each computation impacts quality, and only apply approximation to the less-significant computations avoiding a catastrophic effect in the output quality.

Beside these key design principles, it is essential to further distinguish between the dynamics and the determinism of the applied approximations or the related uncertainties, which are specifically highlighted in this thesis: *Static* approximate computing techniques are incorporated at design-time and refer to algorithmic and/or architectural modifications to improve efficiency at the cost of quality degradation. The impact of such techniques is identical and fully deterministic for all manufactured chips, and therefore, they are more useful to deal with slow variations with known impact. *Dynamic* approximate computing techniques, however, refer to measures to adapt the quality at run-time or post-manufacturing while dealing with fast variations or variations with an unknown or intractable effect. These specifically also include techniques that deal with differences between manufactured chips or different input data characteristics, which may be neither deterministic nor predictable. Statistical analysis across the input data-set and fault distribution must be used to define their impact on the quality. Consequently, instead of a straightforward quality metric, quality itself must be viewed and analyzed as a distribution.

1.1 Background

Approximate computing is often used to broadly refer to techniques that exploit the intrinsic resilience of applications to enable mitigating variability and improving the efficiency (i.e., cost and power) across different computing layers [6,20]. First, at circuit level, techniques to design approximate circuits that are highly efficient can be developed [23,24]. Next, at architecture/algorithm level, approximate architectures allow for a trade-off between efficiency and output quality by exploiting the resilient nature of many algorithms [25,26]. Finally, at software level, resilient computations within an application can be identified and can be partially pruned or mapped to approximate platforms [27]. We note that the study of ap-

proximate computing techniques at multiple layers can have a considerable impact on the manufacturing cost and energy efficiency of the integrated circuits as it allows to use less reliable and less energy-hungry hardware while still guaranteeing acceptable quality levels. The focus of this thesis is mainly at the algorithm and architecture level.

Some of the proposals and key principles underlying approximate computing can be traced back to some well-established disciplines. For example, many of the algorithms in the domain of signal processing are based on approximations [28]. In this spirit, the very-large-scale integration (VLSI) signal processing domain contains many examples to approximate some specific functions or arithmetic operations. Specifically, approximate recursive algorithms that are used to compute transcendental functions [29], algebraic techniques that are used to approximate complex-valued operations [30], and finite word-length optimizations that are extensively employed to approximate any arithmetic operations [31], are among the well-known examples.

Closely related to the above-mentioned techniques, but motivated more from the circuit side, different proposals exists that modify arithmetic operations slightly in comparison to the original ones resulting in a more efficient implementation. More specifically, it is proposed to approximate the implementation of some core arithmetic functions. This includes, for example, approximate adders [32–35] and approximate multipliers [36, 37]. Along the same line, design automation methods to automatically derive such approximated arithmetic components from a golden design are proposed [38, 39].

Beside the arithmetic and logic units approximation, memory elements are also the focus of approximate computing techniques as they account for a significant fraction of area and energy in many systems-on-chip (SoCs). To this end, various techniques are proposed to enable the usage of memories below their reliability limits while exploiting the inherent error resilience of many applications, which result in a more cost- or power-efficient design. An unreliable static random-access memory (SRAM) with a dynamic quality management at the cost of reliability is, for example, presented in [40] that shows improved energy efficiency. In this work, bits with a more significant impact on the quality are protected while performing voltage scaling to improve overall energy efficiency. A novel embedded DRAM (eDRAM) optimized for a high-performance operation is proposed [41]. An energy-quality trade-off is enabled with a limited refresh to achieve a substantial decrease in power consumption at the cost of an increase in cell-failure probability. The authors of [42] propose to scale the supply voltage of caches beyond their reliability limit while the supply voltage and the number of faulty bits are monitored and controlled in software to achieve the desired energy-quality trade-offs. A similar proposition is made in [43] for an on-chip SRAM in a Binary Neural Network accelerator where the supply voltage is scaled well below the safe operating limit at the cost of accuracy loss by exploiting the intrinsic bit-error resilience of the Binary (single-bit) Neural Networks.

Apart from approximation of different hardware components, it is proposed to exploit the flexibility and robustness of some algorithms through approximate computing. For example, iterative algorithms are a great candidate since the algorithm iterations provide a natural means to trade the output quality for computational efforts. Also, probabilistic and heuristic algorithms embody a trade-off between optimality of results and computational complexity [44]. Communication systems are one area that can benefit from such a type of approximate computing since they contain various iterative and probabilistic algorithms. Further, the system has an inherent robustness [45, 46] against the potential (rare) worst-case events and there is often an acceptable output range, which provides a good flexibility to be exploited by approximation techniques. In addition to this system-level resilience, the fault-tolerance of different components of communication systems have been studied [47–49] and a promising potential for a more cost- and energy-efficient design has been shown at the cost of a negligible communications performance degradation.

1.2 Thesis Contributions & Outline

The contributions of this thesis focus on algorithm and architecture techniques for the design of approximate and efficient hardware for signal processing systems and are summarized as follows: First, we propose a design methodology that allows to drop the requirement of 100% reliable operation and to accept dies with unreliable memory components. We examine the proposed methodology for multiple applications such as image processing kernels and a channel decoder in communication systems. Second, we propose a systematic statistical framework to dynamically adapt the output quality of a channel decoder at run-time, as an example of a dynamic iterative algorithm, that adjusts its effort to exploit circuit-level power reduction mechanism for a significant reduction in the energy consumption. Finally, we propose a static algorithmic and architectural technique at design-time to reduce the complexity of the arithmetic units of a channel decoder resulting in a significant reduction in both design area and energy consumption.

In the following, we outline the chapters of the thesis and we summarize the respective main contributions.

Chapter 2: Approximate Computing with Unreliable Memories by Restoring the Beauty of Randomness

Without pessimistic design margins, process variations result in unreliable operation of digital circuits and lead to QoS degradation. The main challenge while dealing with unreliable circuits lies in defining the output quality degradation due to the variations. In that light, this chapter deals with the effect of unreliable memories in a system and proposes methods to guarantee a minimum output quality for the corresponding application.

We first propose to accept dies with unreliable memories to abandon the requirement of 100% reliable operation, which avoids expensive design margins. Given the importance of a correct quality assessment and thus the difficulty of yield analysis for such an unreliable hardware, we then propose a methodology for analyzing the performance of a population of unreliable dies by performing a detailed quality analysis for each die in the population. This method requires complicated and thus expensive test procedures of the fabricated dies to separate the well-performing dies from dies that deliver poor quality. We, therefore, propose to randomize the errors in the memory to provide an ergodic[1] fault behavior and to remove the need for a complicated parametric test. Finally, we show that this method allows to guarantee a minimum-quality for all the dies that are separated after production with a regular low complexity test procedure.

Chapter 3: Practical Approximate Channel Decoders with Unreliable Memories

Communication systems are one of the domains that can benefit from the approximate computing principles since such systems consist of various fault-tolerant algorithms that deal with noisy data. This chapter demonstrates the ideas from the previous chapter with the example of an low-density parity check (LDPC) decoder in a communication system and provide a proof for them through measurement results.

We propose a LDPC decoder testchip that integrates faulty memories into the datapath to solidly verify the non-ergodic memory fault model and the quality distribution across the population of dies, which were initially shown in the previous chapter. Beside verifying the fault model and quality distribution, we adopt and integrate ideas from the previous chapter to the testchip, but specifically tailor them for the LDPC decoder example, which significantly improve the decoder performance and enable the usage of approximate memories. As such, memory faults are randomized to restore the ergodic quality distribution and to enable independent decoding attempts through creating different logical memories over time, and the binary data representation and memory fault bias are jointly optimized to minimize the impact of memory faults on the quality. The design is the first proof-of-concept demonstration of a signal processing circuit that provides a stable quality in the presence of memory reliability issues.

Chapter 4: DVFS Based Power Managment for LDPC Decoders with Early Termination

Data-dependent variations at run-time are a fast type of variation and another source of expensive design margins taken between the algorithm- and architecture-level. For example, iterative algorithms are often configured for the maximum number of iterations to achieve

[1] A process is ergodic, if the ensemble average corresponds to the time/data average.

the required performance for the worst-case input scenario. We show that such variations can also be exploited for average-case saving through approximate computing. With this in mind, Chapter 4 deals with power management of an approximate LDPC decoder by early termination (ET) of the decoding iterations and shows a systematic methodology of a dynamic energy-quality trade-off adjustment.

We first propose a systematic statistical framework for reducing the energy consumption of an LDPC decoder with dynamic voltage and frequency scaling (DVFS) based ET and characterize the maximum theoretically-achievable savings. By exploiting the iterative nature of the decoding, we then propose an algorithm to dynamically predict the decoding iterations at run-time for each codeword that benefits from the performance-complexity trade-off. This algorithm allows us to depart from the static and 100% predictable operation achieving the desired energy-efficiency level at the cost of a well-controlled output quality, i.e., error-correcting performance, degradation.

Chapter 5: Toward Energy and Area Optimization of High-Throughput LDPC Decoders by Exploiting Quantized Message Passing

Motivated by the static approximate computing techniques, this chapter demonstrates design-time algorithmic and architectural techniques to reduce the complexity improving the area and energy efficiency of an ultra-fast LDPC decoder.

Unrolled LDPC decoders are a promising solution for the ultra-high decoding throughput demands, however, their implementation remains a big challenge due to the very high number of interconnects. Motivated by the historically very successful static approximate computing techniques, we employ an approximate non-uniform quantization scheme to reduce the bit-width of the arithmetic units, which highly reduces the implementation complexity. We then propose an architecture based on serial transfer of the messages to further reduce the interconnect complexity thus enabling the high-throughput implementation of the decoder. We finally adapt a linear floorplan for the unrolled architecture, which results in a physical implementation with a very good area and energy efficiency.

1.3 Selected Publications

This thesis is largely based on the following publications. A complete list of author's publications can be found in the List of Publications as part of the back matter.

Chapter 2:

R. Ghanaatian, M. Widmer, and A. Burg, "Design for Test with Unreliable Memories by Restoring the Beauty of Randomness", *journal publication submitted.*

Chapter 3:

R. Ghanaatian, R. Giterman, A. Bonetti, A. Balatsoukas-Stimming, and A. Burg, "ErgoDEC: A Practical Approximate LDPC Decoder in 28 nm FD-SOI with Unreliable Memories", *journal publication submitted.*

Chapter 4:

R. Ghanaatian and A. Burg, "DVFS based power management for LDPC decoders with early termination", *IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2017.

Chapter 5:

R. Ghanaatian, A. Balatsoukas-Stimming, TC. Müller, M. Meidlinger, G. Matz, and A. Burg, "A 588-Gb/s LDPC Decoder Based on Finite-Alphabet Message Passing", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, Feb. 2018.

A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled LDPC decoder based on quantized message passing", *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2015.

1.4 Third-Party Contributions

All third-party contributions to the work presented in this thesis are listed in this section.

The work in Chapter 2 on approximate computing with unreliable memories was done using an emulation platform that was developed by Marco Widmer. Andrea Bonetti was supporting him during the development. Marco also gave me useful recommendations on how to modify the emulation platform to support different memory types and memory fault models.

I worked in close collaboration with Alexios Balatsoukas-Stimming during the development of the work provided in Chapter 3, i.e., the approximate channel decoder with unreliable memories. He supported and supervised me in the development of the simulation framework for the decoder with faulty memories, and by providing many ideas and suggestions for the project. The fabricated test chip was the result of a collaboration within many people. Andrea Bonetti set up the main floorplan and was the memory cell designer. Hanan Marinberg and

Tzachi Noy, under the supervision of Adam Teman, helped Andrea to automate the design of the dynamic memory arrays for the decoder. Robert Giterman continued the backend and performed sign-offs. He also helped and supported me significantly for the chip measurements. Jonathan Narinx helped Robert Giterman for the backend and designed the PCB for the chip measurement. Finally, the measurement set-up was a customization based on a set-up that was originally developed by Jeremy Constantin and Christoph Müller.

The work in Chapter 5 was the result of a collaboration between our lab and the Institute of Telecommunications at university of Vienna. Micheal Meidlinger and Alexios Balatsoukas-Stimming proposed the theory behind the finite-alphabet decoders and provided a framework to generate a finite-alphabet LDPC decoder for multiple configurations. Alexios Balatsoukas-Stimming supported me during the development of the decoder architecture. Christoph Müller significantly contributed to different phases of the backend flow. Adam Teman provided useful suggestions for the floorplan and the backend.

1.5 Preliminaries

Throughout this thesis, we focus on approximate computing for communication and signal processing systems. We specifically consider LDPC decoders as a key component in modern communication systems that exhibit an inherent error-resilience for the case study of Chapters 3-5. In Chapter 2 and Chapter 3, we also focus specifically on approximate computing for memory elements as they are the point of first failure in nanometer nodes [50]. As such, in this section, we provide some background on embedded memories in modern SoCs and on LDPC decoders. We note that we provide further and more specific background to each chapter at the beginning of the corresponding chapter, while this section reviews some preliminaries for the topics of all the chapters.

1.5.1 Embedded Memories

The increasing need for having more embedded memory on chip to provide a better system integration and to reduce the speed and power penalty of off-chip memory access has shifted the SoC design challenge from the computing logic to the on-chip memory. More specifically, embedded memories typically consume a large share of the total area and the power budget of modern VLSI SoCs in various applications ranging from different microprocessors [51, 52], to communication systems [53] and AI accelerators [54, 55].

Besides the dominance of memories in the silicon area and power consumption of modern SoCs, they are also the first-point-of-failure and limit design margins. For example, commercial SRAM is more susceptible to functional failure as compared to the logic gates when the supply voltage is scaled [50]. Further, due to high density design rules of memories, SRAMs

are often accounted as an important factor that limits the overall manufacturing yield in nanometer nodes even under nominal operating conditions [56]. Therefore, memories are often considered as a priority target for approximate computing techniques to allow for a better energy-efficiency and a higher yield, as it is also considered in Chapter 2 and Chapter 3 of this thesis.

Although most of the techniques that are provided in this thesis to enable approximate computing with unreliable memories are independent of the memory type and the corresponding memory technology, it is useful to briefly discuss the different embedded memory types that are in use today. Broadly speaking, embedded memories can be divided into two main categories: 1) SRAM and 2) eDRAM [57], which are explained in more detail in the following.

SRAM uses a cross-coupled inverter pair to statically retain the stored data. SRAM has been the mainstream memory solution in SoCs during the past decades [58]. The widespread of this solution is mainly because of the availability of SRAM compilers and the fast read/write access time [59]. Unfortunately, SRAM has some drawbacks that limit its usage in some applications. For example, a typical SRAM has a large footprint, i.e., 6 transistors for each bit. Also, the operation of SRAM macros in a low power regime with a scaled supply voltage (for example below 600 mV) encounters several design challenges [60, 61].

To alleviate the large footprint of SRAMs, eDRAMs that are comprised of 1 transistor (1T) and 1 capacitor (1C), are used, which results in a better memory integration density. In eDRAM, the data is stored in the form of electric charges on the capacitor. Unfortunately, the stored data deteriorate due to a leakage and a periodic refresh is therefore required [62]. In addition to this limitation, eDRAM requires special process options to build high-density capacitors [63], which is often not available or requires an increase in the production cost.

In order to alleviate this extra manufacturing cost of 1T-1C eDRAM, a new type of eDRAM, i.e., gain-cell (GC) eDRAM [59], has recently been proposed. The bit-cells in this memory are build from 2-4 transistors [64], and the memory is fully compatible with any standard CMOS technology. From a functional perspective, GC-eDRAM also requires a periodic refresh similarly to conventional 1T-1C eDRAM. For power reduction as well as to reduce the bandwidth overhead for refresh, there is an incentive to operate both 1T-1C eDRAM and GC-eDRAM without conservative refresh margins [41]. However, this power/refresh overhead reduction comes at the cost of few failures in the bit-cells with worst-case leakages.

One of the memory types that has been proposed in recent years to provide a more reliable operation and better integration with logic gates, as compared to SRAM and eDRAM, is standard-cell memory (SCM). SCMs are memory arrays that are described using hardware description languages (HDL) and are mapped to standard-cell libraries. Although the bit-cell in this type of memory has a large footprint, it is widely recognized that the SCMs are better suited for low-power approximate memories to operate at scaled supply voltages [61]. Furthermore,

it has also been shown that they are a good candidate for realization of small-size macros (up to a few kbits) that are needed to be tightly coupled with logic [65].

1.5.2 LDPC Codes and Decoders

Low-density parity check (LDPC) codes are linear block codes for error correction in communication systems that were first introduced by Gallager in 1962 [66]. At that time, however, the complexity of the encoder and the decoder was considered too high for a practical implementation and they were mostly ignored for more than three decades. LDPC codes were rediscovered by MacKay and Neal in 1997 [67]. Today, LDPC codes are adopted and used in numerous communication standards (including the recent 3GPP 5G-NR cellular system) because of their excellent error correcting performance and the availability of their practical and flexible implementation for various throughput and energy requirements.

1.5.2.1 LDPC Code Construction

An LDPC code $\mathscr{C}$ of blocklength N is a set of codeword vectors that satisfies

$$\mathscr{C} = \left\{ \mathbf{c} \in \{0,1\}^N \middle| \mathbf{Hc} = \mathbf{0} \right\}, \tag{1.1}$$

where each row of the matrix multiplication defines a parity-check constraint with modulo-2 operations and $\mathbf{H} \in \{0,1\}^{M \times N}$ is the *parity-check matrix* of the LDPC code. We define the number of non-zero elements in each column and each row of the parity-check matrix as d_v and d_c, respectively. For a matrix to be *low-density*, the number of non-zero elements must be much smaller than the matrix dimension, i.e., $d_c \ll N$ and $d_v \ll M$. In this thesis we only consider *regular* LDPC codes indicating that the parity-check matrix contains exactly d_v non-zero elements per column and d_c non-zero elements per row. Such a code is called a (d_v, d_c)-regular code. The *rate* of an LDPC code, which indicates the fraction of information bit in a codeword vector, is given by $R = 1 - \frac{M}{N}$. For a regular code, the rate can also be derived as $R = 1 - \frac{d_v}{d_c}$. A low-rate code provides better error correction with a higher transmission overhead, while a high-rate code tolerates fewer errors at a lower transmission overhead.

Beside the parity-check matrix, a *Tanner graph* is often used to graphically represent such a code [68]. Tanner graphs are *bipartite* graphs in which the node of the graphs are separated into two distinct sets and edges connect nodes of the two types. The two types of the nodes in a Tanner graph are called *variable nodes (VNs)* (represented with circles) and *check nodes (CNs)* (represented with squares). The CNs are connected to the VNs they check. Specifically, an edge connects CN m to VN n if and only if $\mathbf{H}_{m,n} = 1$. An example of a Tanner graph for a $(2, 4)$-regular LDPC code with a blocklength of $N = 6$ and the corresponding parity-check matrix is illustrated in Fig. 1.1.

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

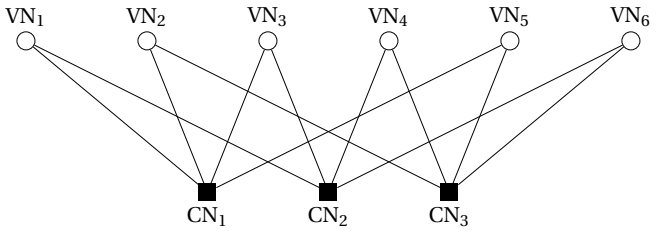


Figure 1.1: Example of a parity check matrix (left) for a (2, 4)-regular LDPC code of blocklength $N = 6$ and the corresponding Tanner graph (right).

1.5.2.2 Decoding of LDPC Codes

The decoding of an LDPC code is a process of translating received codewords into the original codewords that were sent over a communication channel. Since performing an optimal decoding is complex, sub-optimal decoding techniques are used that are based on *iterative message passing (MP)* algorithms [69]. In these algorithms, information is exchanged between VNs and CNs of the Tanner graph over the course of several decoding iterations. Since the parity-check matrix of LDPC code is sparse, the number of messages that are exchanged over the Tanner graph grows linearly with blocklength N, which means the complexity of MP decoding only grows as $\mathcal{O}(N)$ [70].

In the MP algorithm, the nodes act as independent processors, collecting incoming messages and producing outgoing messages. Therefore, each node has a local *update rule*, which computes a mapping from the incoming messages for all the outgoing messages. Depending on the update rules that are employed for the CNs and VNs, different decoding algorithms are identified, among those, *sum-product (SP)* algorithm [71] and *min-sum (MS)* [72] algorithm are two popular examples. SP is the optimum algorithm with respect to the decoder error-correcting performance, however, it involves transcendental functions that impose high complexity for hardware implementation. MS algorithm is an approximation of SP that is more suitable for hardware implementation since it only involves addition and minimum computation. We will provide a detailed mathematical explanation on MS update rules at the beginning of Chapter 3 and Chapter 5.

1.5.2.3 Hardware Architectures for LDPC Decoder

Generally speaking, architectures for an LDPC decoder are comprised of three hardware components: VN and CN processors to compute the update rules, an interconnect network representing the edges of the Tanner graph, and memory elements to store the messages during the decoding [73]. The MP view of the iterative decoder enables different architectural choices based on the Tanner graph. While a parallel implementation uses distinct CN and VN processors with minimal storage requirements, a serial implementation consists of only one CN and VN processor with a significant storage requirement [73].

Different communication standards that utilize LDPC codes require various throughput and complexity (e.g., silicon area and power consumption) trade-offs. In order to meet these diverse requirements, different hardware architectures have been proposed in literature. From a high-level perspective, by starting from a parallel implementation [74], i.e., an *isomorphic architecture*, and applying architectural transformations, different architectures can be obtained. More specifically, by applying *resource sharing* among CN and/or VN processors, block-parallel [75] and partial-parallel [76] architectures are obtained. Such architectures provide a medium throughput and have a modest complexity. The decoders that are used in Chapter 3 and Chapter 4 have a block-parallel architecture. In order to obtain a higher throughput, *iterative decomposition* is applied, which results in an unrolled architecture that has a distinct set of processing nodes for each decoding iteration and thus a high complexity degree [77]. The decoders that are proposed in Chapter 5 have an unrolled architecture.

Beside the architectural transformations that are used to trade throughput for complexity, *early termination (ET)* of the decoding iterations [78] is widely employed to improve the energy-efficiency of LDPC decoders. ET is the process of stopping the decoding iterations when further iterations are unlikely to alter the decoding results, which can be recognized by checking the parity equations in (1.1). In addition to ET, we will provide different techniques based on approximate computing paradigm to further improve the energy-efficiency of an LDPC decoder.

2 Approximate Computing with Unreliable Memories by Restoring the Beauty of Randomness

Process variations result in unreliable operation of the very-large-scale integration (VLSI) circuits and lead to quality of service (QoS) degradations. As we are approaching the most advanced technology nodes, such as 7 nm and 5 nm, it is becoming increasingly challenging to guarantee reliable operations without a conservative and thus costly design margin, which leads to a diminishing gain from technology scaling.

Memory elements are the first point of failure in many VLSI technology nodes due to the higher density design rules [50]. At the same time, embedded memories account for most area and energy-consumption in complex systems-on-chip (SoCs) and in many accelerators. Emerging applications based on artificial intelligence (AI) and machine learning (ML) demand an extremely large amount of memories, ideally embedded on-chip, which aggravates the failure issue. The use of popular and effective low-power design techniques, such as voltage scaling, and the sensitivity of storage elements to voltage scaling further aggravate the situation [79], and therefore, it becomes increasingly challenging to maintain a 100% reliable operation across all corners.

To alleviate the reliability issues, it has been proposed to accept errors in memories, as a flavor of approximate computing, while exploiting the inherent fault-tolerance of many applications. Many publications demonstrate the robustness of various types of algorithms and applications against errors in embedded memories and claim considerable savings in terms of energy or protection overhead for scaled voltages. The authors in [80] describe different energy-quality trade-offs based on static random-access memory (SRAM), which shows an improved energy efficiency by protecting more significant bits while jointly performing voltage scaling. Dynamic random-access memory (DRAM) or embedded DRAM (eDRAM) is considered in [41, 81] with a limited refresh to achieve a substantial decrease in power consumption at the cost of an increase in cell-failure probability. The authors of [42] propose to scale the supply voltage of

caches beyond their reliability limit to achieve energy-quality trade-offs. The error tolerance of a Binary Conventional Neural Network is studied [82] by considering the efficacy of the network coefficient SRAM voltage scaling on the classification accuracy through measuring multiple dies. In [43], it is similarly proposed to scale the supply voltage for an on-chip SRAM in a Binary Neural Network accelerator well below the safe operating limit at the cost of accuracy loss by exploiting the bit-error resilience of the binary network. Even though some of the above studies are based on measurement, most of them assume an *ergodic* fault model in which the time-average quality of a single faulty die is well predicted by the ensemble-average quality across the population of manufactured dies. This model is convenient as it allows to quickly derive a straightforward scalar average-quality metric from Monte-Carlo simulations with random transient faults that are injected to model defects from process variability issues. These errors are often also independent of the stored data and each other, identically distributed, and symmetric.

Unfortunately, the above described widely-made assumption of an ergodic process is seriously flawed and the consideration of an average quality across time/data and across a population of dies has no relevant operational meaning. In fact, each chip with a reliability issue is characterized by a specific failure mode that characterizes, for example, the location and polarity (stuck-at-0 or stuck-at-1) of faulty bits. The specific realization of faults can have a significant impact on the average quality of each individual die, which is not reflected in the scalar average quality metric of the ergodic fault model. We will later also support these claims with measurements in the next chapter.

As an important consequence, the quality difference between dies should be considered during the test procedure of the fabricated dies to discard dies with insufficient quality. Unfortunately, how to achieve this is only partially addressed by very few works. This lack of coverage of this important issue is attributed partially to the widespread acceptance of the ergodic fault model, which ignores the significant quality differences between dies and therefore does not require special attention during test. The work in [83], has indeed recognized that different dies of an unreliable digital circuit will produce a different quality and this matter needs to be accounted for during test. Unfortunately, the procedure proposed in [83] to infer a quality-impact from test results is only applicable to very small circuit components and is far from being able to evaluate the impact of a fault on output quality of a complex algorithm. The work in [84] proposes a test procedure based on a quality metric that is computed from test results and predicts a significant yield enhancement when errors can be tolerated. Unfortunately, this quality metric is very application-specific and is difficult to derive for more complex applications and corresponding circuits.

To be able to drop the requirement of 100% reliable operation during design, manufacturing, and test, it is absolutely necessary to

1. provide a yield analysis across the manufactured chips that requires characterizing the performance of *all the chips* and determining how many chips meet a minimum performance requirement, and

2. provide a strategy (including design-for-test and test) to separate dies that meet a minimum-quality requirement even in the presence of defects in unreliable memories.

We note that throughout this chapter, we assume to have faults only in the memory components, as the dominant part of the modern SoC in terms of area and power. This assumption is important since the derivation of models for the misbehavior of other parts of the circuit, such as the logic part, is difficult and memory faults have a more moderate impact on the performance since different data elements are stored in different memory locations and a single fault affects mostly a single data element. Arithmetic elements on the other hand are typically re-used multiple times which multiplies the impact of any error.

Contributions and Outline Given the importance of a correct performance assessment for the unreliable hardware and the difficulty of yield analysis for such hardware, we propose a generic methodology to analyze the performance of a population of unreliable chips and to deliver reliable performance guarantees. The methodology is based on a design-for-test procedure that guarantees identical time-average behavior for all manufactured dies with the same number of defects by restoring the beauty of randomness. It enables a quality-yield analysis and a simple, low-overhead test strategy that does not require costly per-die quality assessment.

More specifically, the contribution of this chapter are the followings:

- To characterize the faulty manufactured chips, we propose to consider a more complex performance metric, i.e., the individual chip time-average quality instead of the ensemble-average quality, to capture the quality distribution in the entire population of dies.

- Given the quality distribution across the population of dies, we provide a yield analysis to demonstrate how many chips meet a certain time-average quality requirement.

- We propose a new design-for-test strategy that randomizes the deterministic post-fabrication errors to obtain a truly ergodic fault model. This measure not only enables the analytical performance assessment under faulty hardware, but also (and more importantly) facilitates the corresponding test procedure. The proposed randomization scheme is implemented for a faulty memory and synthesized results are reported for overhead comparison.

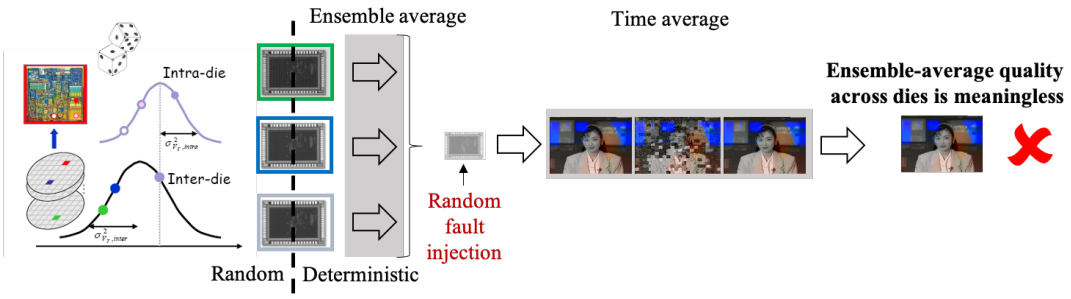


Figure 2.1: The fault model follows a random distribution, however, the fault mode for each chip after manufacturing is deterministic. Therefore a performance evaluation that is according to ensemble-average quality is invalid for the population of chips.

- We demonstrate and evaluate our methodology with practical image processing benchmarks for an embedded system with faulty memories.

The remainder of this chapter is organized as follows. Section 2.1 provides the underlying mathematical analysis for quality assessment, which serves as the background for the chapter. In Section 2.2, we show the detailed quality assessment methodology with unreliable memories for the non-ergodic fault model. Section 2.3 explains the proposed randomization scheme to restore an ergodic behavior. Section 2.4 presents the results, Section 2.5 discusses the impact of the proposed scheme, and Section 2.6 concludes the chapter.

2.1 Background

A convenient approach to analyze the robustness of an application against memory reliability issues is to randomly inject errors either once in the beginning of a simulation or during the simulation. By averaging across many simulation runs (i.e., across many error patterns), an average quality is obtained, as illustrated in Fig. 2.1.

The simplicity of this model lies in the fact that it does not distinguish between the behavior of a given die over different inputs (i.e., time) and the behavior of a population of dies. This lack of distinction between these two dimensions corresponds to an ergodic fault model. In such a model the statistical properties of errors in different dies are the same as those that are relevant for the appearance of errors over time in a given die. Hence, a single scalar average performance metric is sufficient to characterize the quality of an application when errors are injected. More formally, let $\mathscr{P}_n(\mathbf{y}(t),\mathbf{e}_n(t))$ be the output quality of a given die $n \in \mathscr{N}$, where $\mathscr{N}$ denotes the population of dies created by the manufacturing process and $\mathbf{e}_n(t)$ denotes its specific fault-realization. The test-data set $\mathbf{y}(t)$ on which the quality is evaluated is a function of time t and the time-average quality of a given die n as observed during operation of that

die $\bar{\mathscr{P}}_n = \mathbb{E}_t[\mathscr{P}_n(\mathbf{y}(t), \mathbf{e}_n(t))]$ is the quality metric that we must ultimately guarantee for each die that passes the production test.

However, *only for an ergodic fault model* we can replace the time-average $\mathbb{E}_t[\cdot]$ with the widely used ensemble-average $\mathbb{E}_n[\cdot]$. More specifically, let $\bar{\mathscr{P}} = \mathbb{E}_{n,t}[\mathscr{P}_n(\mathbf{y}(t), \mathbf{e}_n(t))]$ be the average quality of an application in the presence of unreliable hardware with random fault injection. Then the time-average quality corresponds to the ensemble/time average quality as

$$\bar{\mathscr{P}}_n \quad = \quad \mathbb{E}_t[\mathscr{P}_n(\mathbf{y}(t), \mathbf{e}_n(t))] \quad = \quad \bar{\mathscr{P}}, \qquad \begin{array}{c} \\ \uparrow \\ \text{only if } \mathbf{e}_n(t) \text{ is ergodic} \end{array} \tag{2.1}$$

if the fault process is indeed ergodic.

2.2 Proposed Application Quality Assessment with Unreliable Memories

In this section, we discuss our proposed quality assessment methodology. To this end, we review the difference between the ergodic and non-ergodic fault process and provide measurement results to support this difference. Given the relevance of the non-ergodic fault process, we then present a quality assessment methodology to reflect this behavior and finally provide the quality results.

2.2.1 Ergodic vs. Non-Ergodic Fault Model

The main issue with this ergodic fault model assumption is that errors arising from process variations are mostly deterministic after manufacturing, at least for a given set of (voltage, temperature) operating conditions. While these conditions may vary, it is still the outcome of the manufacturing which significantly influences the type and locations of the faults. This post-fabrication deterministic behavior is particularly visible for memories and can be observed indirectly in the results of some publications (e.g., [41, 82, 85]).

To illustrate this more explicitly, we provide two examples from dedicated test chips: For SRAM, within-die process variations determine the unstable bits that fail at low voltages. These are clearly different for different chips, but are stable over time as shown by the three fault maps in Fig. 2.2(a) and the percentage of tests in which the faults are visible. For DRAM and embedded DRAM, variations determine the retention time of each individual bit in the memory. Without a pessimistic refresh, weak cells are the first to fail. Fig. 2.2(b) shows how the retention time distribution, and therefore, also the location of the weak cells varies from chip to chip. Both the SRAM and DRAM behavior are clearly non-ergodic.

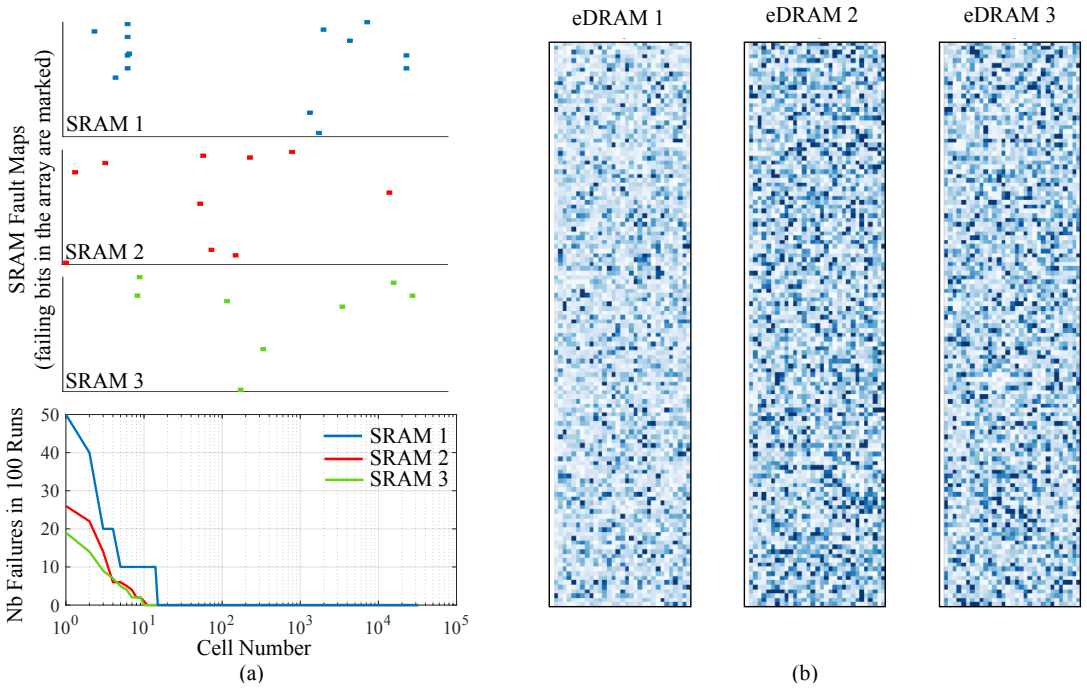


Figure 2.2: Measurements showing the different fault realizations in different chips: a) fault-map of three SRAM macros with the failure rate of the faulty bits across multiple tests, b) data retention time map of three GC-eDRAM macros.

To illustrate the impact of this non-ergodic fault model on the relevance of a quality assessment based on an ergodic fault model, consider the following simple example: A measurement vector $\mathbf{y}$ of length T is stored in an unreliable memory in B-bit 2's-complement format. Our quality metric is the mean-squared error of $\hat{\mathbf{y}}_n \approx \mathbf{y}_n$ which is affected by errors in the memory compared to the original data. A first issue which already illustrates the issues with the ergodic fault model is that for a given bit-failure probability each manufactured die is affected by a different number of errors according to a binomial distribution. However, even for those dies that have the same number of errors K, we can observe significant differences in the quality. With an independent and identically distributed (i.i.d.) ergodic fault model in which K errors manifest as bit-flips, the ensemble-average error of the output is convenient to determine analytically as $\mathbb{E}_n[|\mathbf{y}-\hat{\mathbf{y}}_n|^2] \approx \frac{K}{TB}\sum_{b=0}^{B-1} 2^{2b}$. Unfortunately, it is easy to show that for a arbitrary die n we can only guarantee that $\frac{K}{T} \leq |\mathbf{y}-\hat{\mathbf{y}}_n|^2 \leq \frac{K}{T}2^{2(B-1)}$. These far-apart bounds correspond to bit flips either all in LSBs or all in MSBs representing the best-case and the worst-case scenarios. One is significantly better, the other significantly is worse than the ensemble-average predicted by the ergodic model.

2.2.2 Assessment of the Quality-Yield Trade-Off for Non-Ergodic Fault Models

From the discussion in the previous sub-section, it is evident that the ergodic fault model is not realistic and therefore *the assessment of the ensemble-average quality for a system with*

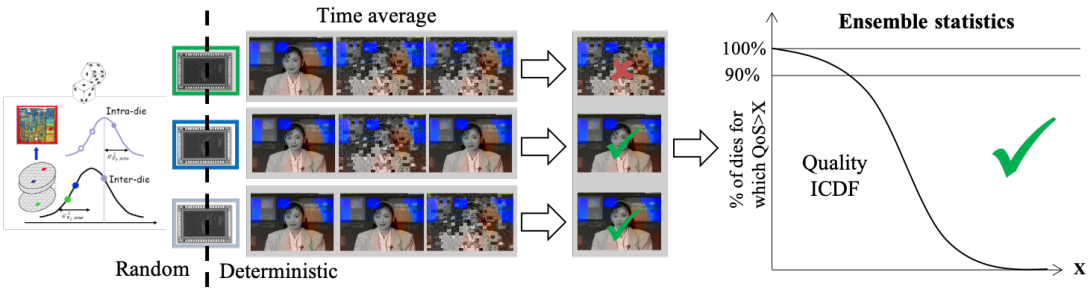


Figure 2.3: The proposed flow for yield assessment. In this flow, the time-average behavior is simulated for all the fault realizations and the ensemble statistics is illustrated as an ICDF.

unreliable (memory) components is meaningless. In fact, even for a good ensemble-average quality, the quality of different dies can be very different and an unknown, possibly a significant percentage of dies may fail to reach the necessary minimum-quality target.

To obtain a more realistic and more meaningful measure of the impact of reliability issues, it is absolutely necessary to perform a *quality-yield analysis.* This analysis generates a population of dies $\mathcal{N}$ with their individual fault patterns $\mathbf{e}_n$ and studies the time-average quality distribution across the entire population as illustrated in Fig. 2.3. The cumulative distribution function (CDF) of the time-average quality then indicates the quality-yield, i.e., the percentage of dies that meet a given minimum time-average quality target.

The procedure to obtain the quality-yield is as follows:

1. First, we group the manufactured dies by the number of faulty memory bits. We define *error ratio* as the number of faulty bits relative to the memory size. Note that for a given bit-failure probability the distribution of the number of errors is a very peaky Bernoulli distribution especially for small error ratios, as illustrated in Fig. 2.4. Hence, there are only very few relevant groups depending on the bit-failure probability and the memory size which can be evaluated separately and then weighted by the probability of occurrence. We approximate these few groups by only one where the error ratio is equal to the bit-failure probability. The following steps are carried out for each group.

2. For a given number of errors, we generate a population of dies $n \in \mathcal{N}$ with their individual fault locations and the type of bit errors $\mathbf{e}_n$. These errors are different deterministic realizations of the fault process, which can be obtained based on silicon measurement statistics, as reported for example in [81, 86]. Here, we consider random location and stuck-at type as the fault parameters for each error ratio.

3. Using *system simulations* with targeted fault injection according to $\mathbf{e}_n$, the benchmark time-average quality performance metric is measured for each die in the population with two nested simulation loops. While the outer loop iterates over the different dies,

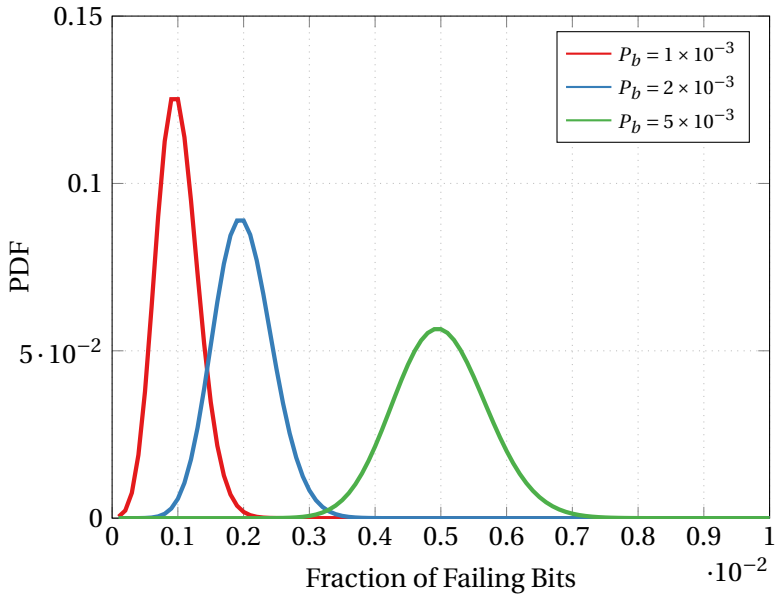


Figure 2.4: The Probability distribution of fraction of affected bits in a memory with size of 10 Kb for multiple bit error probabilities P_b.

i.e., $n \in \mathcal{N}$, the inner loop iterates over the input statistics $\mathbf{y}(t)$, i.e., over time to evaluate the time-average quality $\bar{\mathscr{P}}_n = \mathbb{E}_t[\mathscr{P}_n(\mathbf{y}(t), \mathbf{e}_n(t))]$.

4. Finally, an *analysis* phase is conducted by plotting the CDF of all the measured qualities $\bar{\mathscr{P}}_n$. This CDF illustrates the performance of populations of dies and therefore provides the complete information for yield assessment.

2.2.3 Quality-Yield Assessment FPGA Platform

In order to illustrate the relevance of the yield-quality trade-off we use a software-programmable embedded system that is equipped with an emulator that can mimic faulty memories. To this end, we integrated the emulator presented in [87] into the PULP platform [88], which allows us to evaluate the performance of multiple benchmarks. The system architecture is shown in Fig. 2.7(b). This architecture is a complete system-on-chip (SoC) based on a single core 32-bit RISC-V processor, *PULPissimo* [89], that uses a memory subsystem connected through a single-cycle logarithmic interconnect (system bus). The memory subsystem consists of both 1024 KB reliable and 512 KB faulty memories. The former contains all the program code as well as the critical, e.g., control, data of the selected benchmarks while the latter is used to outsource less-significant large working sets of the benchmarks. While the platform is based on the eDRAM emulator that is publicly available from [87], we use this emulator in this paper only to model reliabilities in SRAM. Hence, we adapt the emulator for this purpose and define

Table 2.1: Description of the analyzed benchmarks.

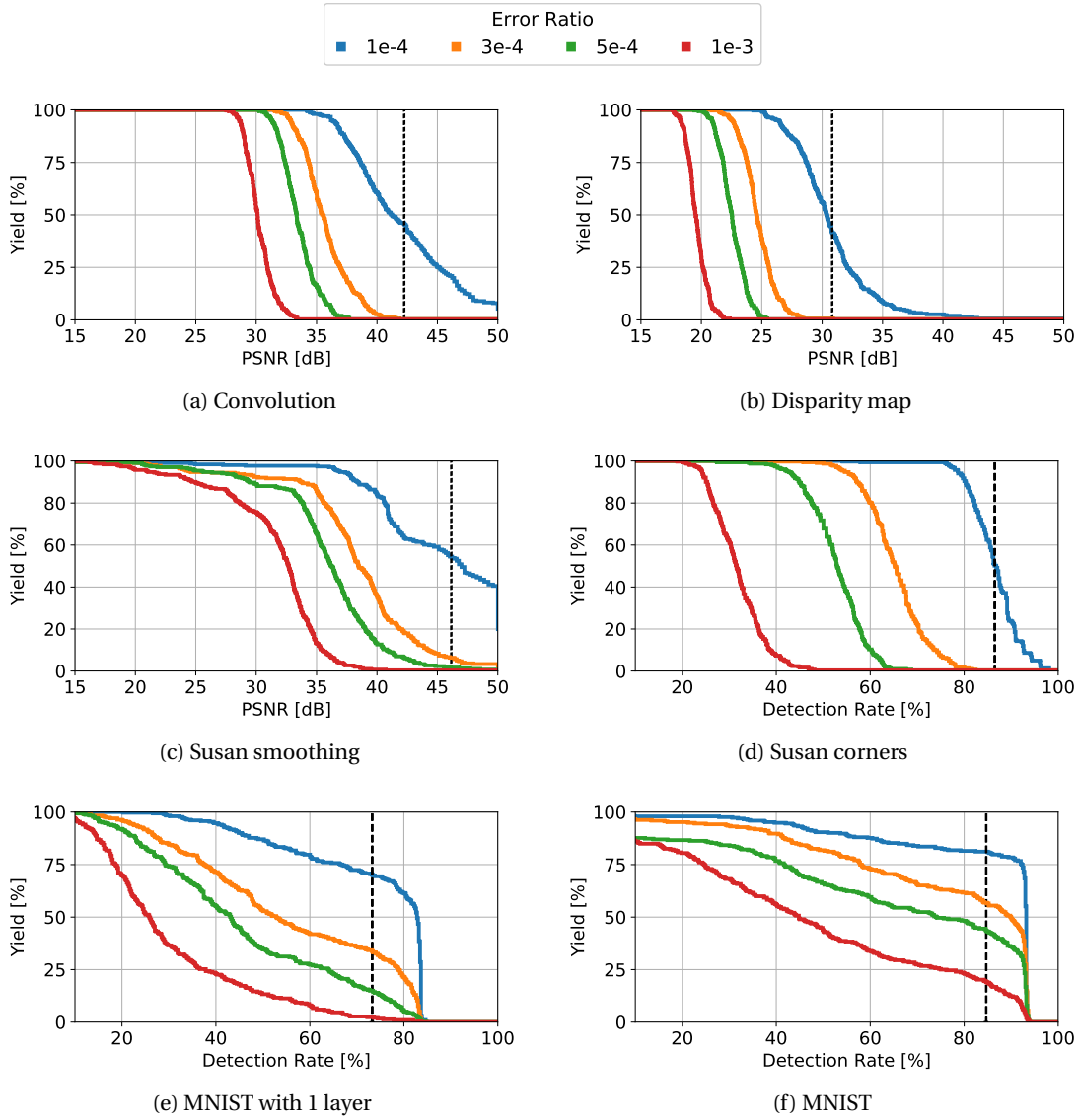
Name	Description	Quality metric
Convolution	Sobel kernel for edge detection on a 128x128 px image	PSNR
Disparity Map [90]	Depth map from two 72x54 px images from a stereo camera	PSNR
Susan Smoothing [91]	Smoothing of a 76x95 px image with SUSAN principle	PSNR
Susan Corners [91]	Corner detection in a 88x66 px image with SUSAN principle	PPV[1]
MNIST [92]	Handwritten digit recognition with a neural network	CID[2]

[1] Fraction of detections being true positives with respect to a reference execution
[2] Percentage of correctly identified digits

directly the bit-error rate instead of using the data-retention-time modeling feature of the emulator.

The emulation platform is implemented on FPGA to accelerate the evaluation of the benchmarks. After programming the FPGA, the evaluation process runs autonomously on the CPU of the emulated embedded system, according to the proposed flow in the previous sub-section. An initialization step is followed by two nested loops to realize Monte-Carlo simulations across multiple instances of the memory fault model and across different input data sets, respectively. More specifically, the outer loop programs the faulty memory emulator with a new fault map and the inner loop iterates over the chosen benchmark and evaluates the quality of the output for each run of the benchmark kernel, which is used to compute the time-average quality for the current fault map. Finally, the summary of these assessments across all fault maps (all simulated chips) allows to provide the yield-quality trade-off by considering the CDF of the quality. This CDF shows the percentage of dies that meet a given minimum quality requirement.

Different embedded benchmarks are implemented in C programming language and are evaluated with the above emulation platform. For each of the analyzed benchmarks, a short description and the used quality metric are reported in Table 2.1. For the non-reliable memory, which contains the less significant (i.e., more robust) data of the benchmark, we consider the fault model explained in the previous sub-section. To characterize the quality for the non-ergodic fault behavior, we then derive 300 different fault maps with the above model, each of which corresponds to a chip, and we derive the quality for each chip.



(a) Convolution

(b) Disparity map

(c) Susan smoothing

(d) Susan corners

(e) MNIST with 1 layer

(f) MNIST

Figure 2.5: Benchmark yield-quality trade-off analysis for the fault model with four different error ratios. The dashed black line shows en ergodic ensemble-average quality for error ratio of 10^{-4}.

2.2.4 Quality-Yield Results

Fig. 2.5 shows the quality-yield trade-off of the different benchmarks for different error ratios (i.e., percentage of defect bits, corresponding for example to different supply voltages). Each benchmark is repeated multiple times with varying and with the same input data.

We observe that for a given error ratio, the quality spreed is very large across individual dies due to the very diverse impact of the different non-ergodic error patterns. Considering the Disparity benchmark as an example, different chips with error ratio of 10^{-4} in the memory span a quality range of $25-42$ dB, while the ergodic average quality, shown with a dashed line,

is 31 dB. More specifically, there exist chips that provide significantly better or more degraded quality compared to this average due to their non-ergodic behavior. Therefore, a minimum quality requirement equal to this average 31 dB only gives a yield around 40% and the rest fails to reach this target. This indicates that the CDF of the quality for different chips is the correct metric for the yield characterization and the ergodic average quality across all the chips is of no operational meaning.

2.3 Restoring the Ergodic Behavior

The quality-yield analysis allows to choose a suitable operating point for the design in terms of the component reliability that optimizes power and other cost metrics under a given minimum quality *and* yield requirement. Targeting the worst-case quality as the requirement would allow to deliver all manufactured dies, i.e., 100% yield, without further testing. Unfortunately, we have also seen that the quality spread across different dies can be very large and the worst-case quality in a population of dies can be very low. Hence, the worst-case quality target is not attractive.

In the following, we first show how a slightly better quality can be guaranteed at the cost of an acceptable yield loss with a complex test procedure. We then show how this achievable quality can be improved and how the yield-loss can be minimized with a small design change and a drastically simplified and feasible test procedure.

2.3.1 Testing for a Minimum Quality Requirement

The difficulty in running a production with a minimum quality requirement that is higher than the pessimistic worst-case quality lies in the need for a parametric test procedure. Such a procedure identifies the quality level for each manufactured die, which can then be compared to the quality threshold to decide if a die should be discarded. This procedure is illustrated as a flow-graph in Fig 2.6. As it can be observed, obtaining the quality level provided by a specific die with a faulty behavior is unfortunately extremely difficult, since it requires to run the application quality benchmark for every fabricated die.

A straightforward approach would be to simply run a quality benchmark on each die as it is used for the quality-yield analysis at design time. Such tests are, however, very time consuming even in a real-time production test set-up, and thus, are uneconomic due to the excessive test time. An alternative method would be to simply keep a table of all the potential error patterns together with the information on the resulting quality. Unfortunately, the number of potential error patterns grows extremely rapidly with the memory size. For example, for a maximum of 5 errors in 1 Kbit of memory, there are more than 10^{12} possible error patterns which is already

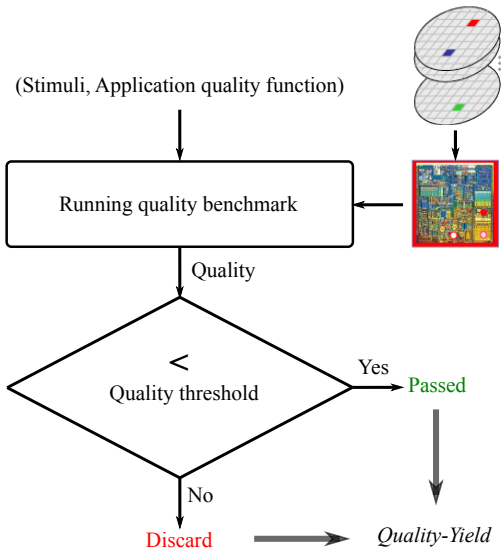


Figure 2.6: Parametric test flow for fabricated dies through running application quality benchmark to separate dies that can pass the minimum quality.

prohibitive. We therefore conclude that identifying the time-average quality of a specific faulty die (with a given and fixed fault pattern) during test is economically not feasible.

2.3.2 Proposed Design-for-Test Strategy: Restore a Random Fault Behavior

The solution to the above-described testability dilemma lies in the observation that the error pattern of each individual die is deterministic. Hence, any averaging over different error patterns is meaningless and cannot be used for representing the quality of each die. To alleviate this issue, we propose to modify the design in such a way that even for a given die (with fixed defect pattern) the time-average behavior of each die corresponds to the ensemble-average over the entire population of dies. Specifically, we propose to randomize the errors by a given defect pattern over different executions of the algorithm. This measure restores the beauty of random fault injection for which the quality is ergodic (i.e., the quality delivered by each die is the same as the average-quality over a population of dies).

More formally, by randomizing the errors over time, the error pattern instance $\mathbf{e}_n$ becomes again time-dependent $\mathbf{e}_n(t)$ for each individual chip. The application ensemble-average quality

$$\begin{aligned}\bar{\mathscr{P}} &= \mathbb{E}_{n,t}[\mathscr{P}_n(\mathbf{y}(t),\mathbf{e}_n(t))] \\ &= \mathbb{E}_t\big[\mathbb{E}_{n|t}[\mathscr{P}_n(\mathbf{y}(t),\mathbf{e}_n(t))]\big] = \mathbb{E}_t[\mathscr{P}_n(\mathbf{y}(t),\mathbf{e}_n(t))] = \bar{\mathscr{P}}_n,\end{aligned} \tag{2.2}$$

is equal to the time-average quality for each individual chip since the stochastic process $\mathbf{e}(t)$ and thus the quality $\mathscr{P}(\cdot)$ are ergodic. We, therefore, make the condition in (2.1) true and conclude a similar result.

2.3.3 Restoring an Ergodic Fault behavior for Memories

In order to realize a random behavior for a faulty memory, error locations should be moved across the memory array. Additionally, error polarities should also be altered randomly to provide a data-independent fault behavior for stuck-at-0/1 faults.

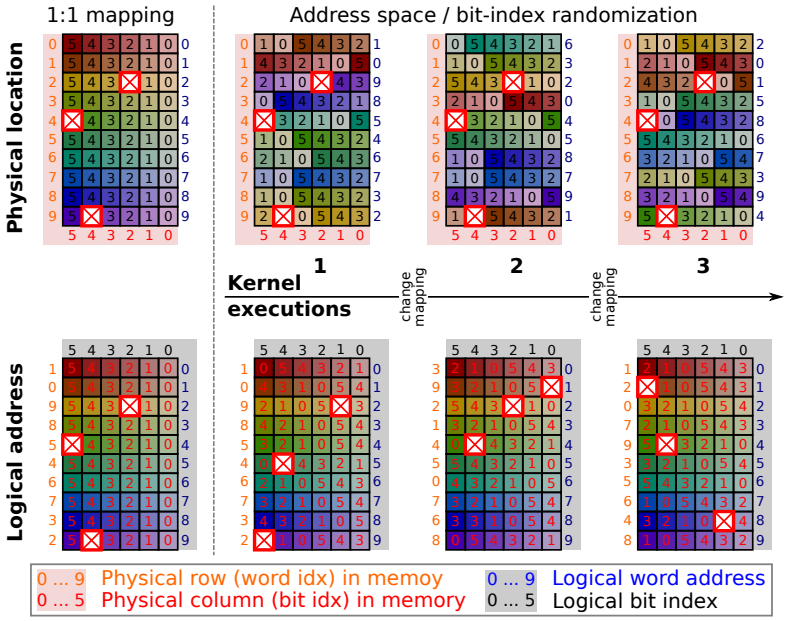
To achieve this randomization, we distinguish between the physical XY-location of a bit in a 2D array of bit-cells on the chip and the logical address including the index of a bit in a word. For the program, only the logical address is relevant, while defects after manufacturing are fixed to a physical location. A straightforward embedded systems typically implements a direct, fixed mapping between logical address and physical location that does not change over time. However, any unique, reversible mapping is valid and the mapping can be changed any time the program or an isolated kernel with memory-intensive local data is restarted and the data in the memory are re-loaded. When the mapping changes, also the location of defects in the logical address space changes as illustrated in Fig. 2.7(a).

To implement an ideal randomized mapping (where each logical bit can be placed in each physical location), the memory must be broken up into 1-bit wide sub-macros which are all individually addressable. Such a memory configuration is shown on the top-left in the system diagram in Fig. 2.7(b). The randomization is controlled by pseudo-random seeds that are changed before each kernel execution. With this seed, hash functions derive scrambled addresses for each sub-macro from the same logical address. A crossbar, controlled by a hashed address then permutes the order of the bits in a word on the data bus. The bits are routed to the memory and are further XORed with pseudo-random bits derived from the address and the seed. This reversible logic operation randomizes the polarity of physical stuck-at-1/0 errors in the logical data. Performing the same operations with the same seed until a new independent kernel execution for all read- and write-operations ensures transparency of the above described mappings.

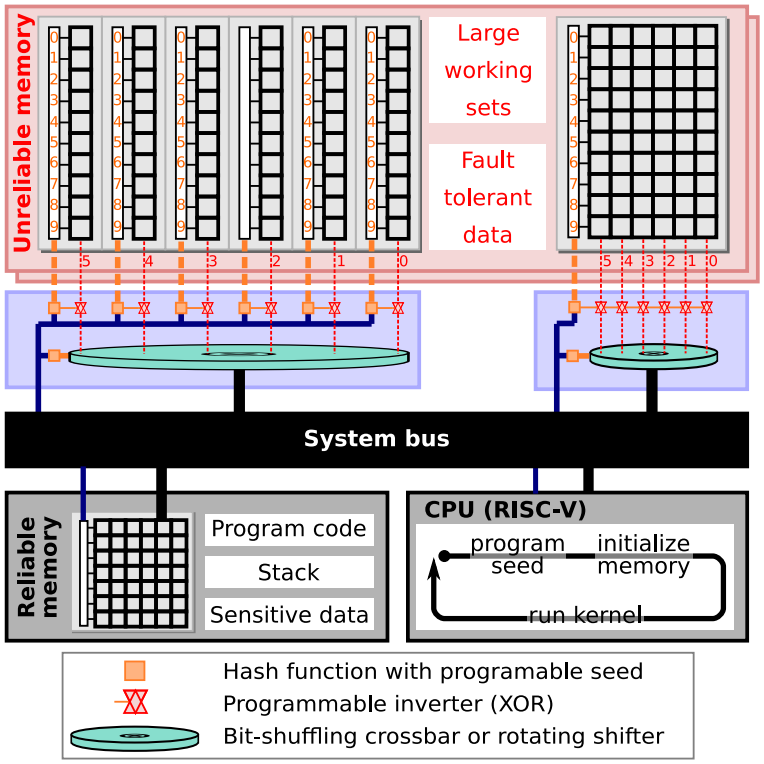
Unfortunately, the ideal randomization is very costly, mainly due to the overhead for the breakup of the memory into sub-macros. We therefore propose a reduced-complexity randomization scheme which leaves memory macros untouched, requires fewer hash function units, and avoids a full crossbar for the randomization of the bit location in the word. The corresponding schematic is shown for the top-right memory in Fig. 2.7(b). A single hash function randomizes the address, the data bits are shuffled with a cyclic shift within a word, and all bits of a word are either inverted or not depending on the address.

2.4 Quality-Yield Trade-Off Results for Ergodic Faults

We re-evaluate the quality-yield for benchmarks of Table 2.1 using the proposed simplified randomization circuit. We follow the procedure described in Section 2.2.2 to obtain the quality-



(a)



(b)

Figure 2.7: (a) Illustration of a 1:1 and randomized mapping of physical locations to logical addresses; (b) System diagram with unreliable memory (top) and logic for ideal (top-left) and simplified (top-right) randomization.

yield. After the initialization for a given number of errors for a group of dies, a fault map for each die is generated (the faulty memory is programmed accordingly). However, we now re-program the seed of the address space randomization logic with a different value for each repetition of a benchmark, as shown in the CPU flow of Fig. 2.7(b). The average quality across these repetitions with different seeds, but with the same fault map (i.e., on the same die) is now the basis for the quality-yield assessment. This analysis reflects the distribution of the average quality that is delivered by the individual dies when operating over an extended time period.

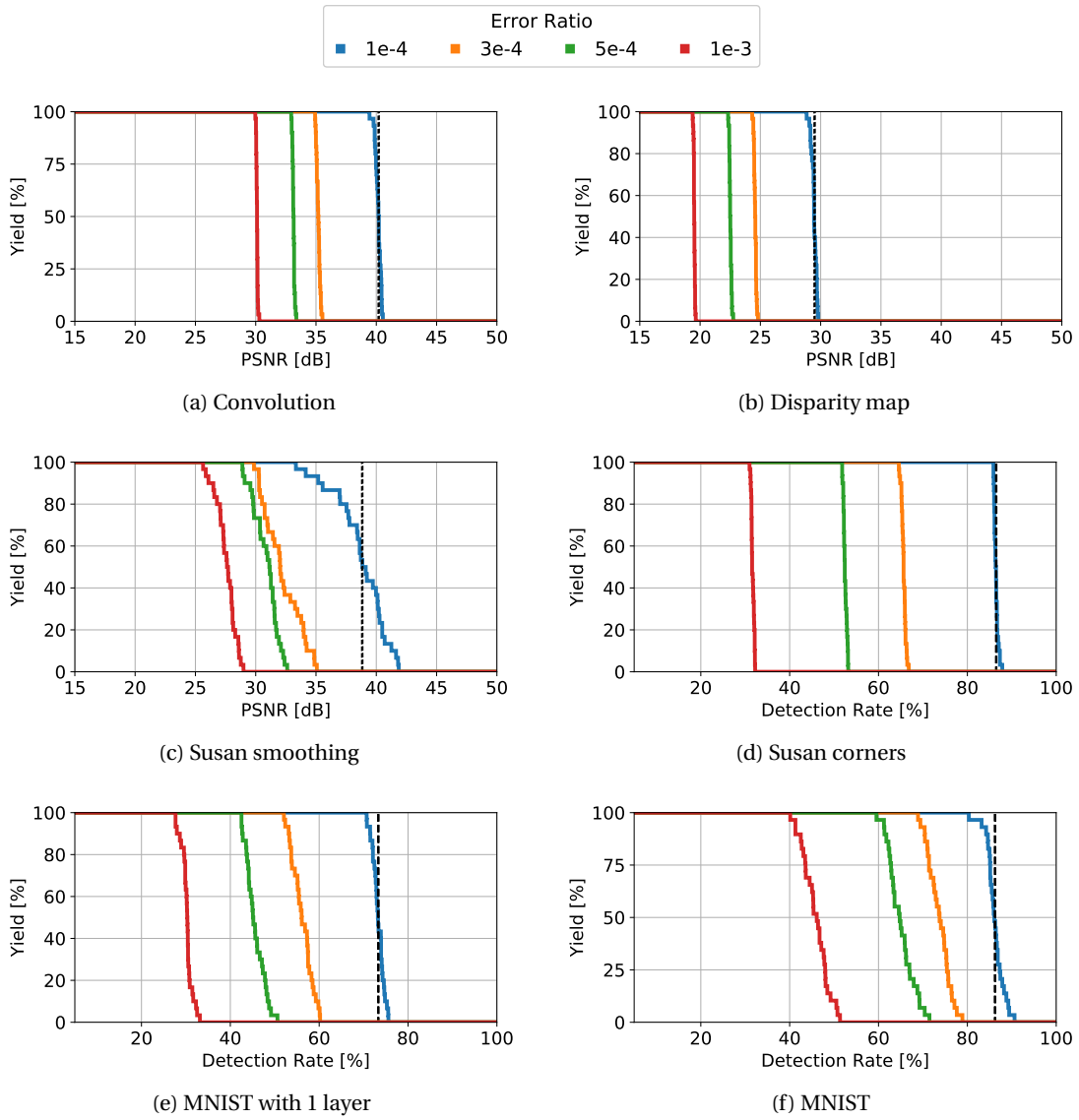
Fig. 2.8 shows the quality-yield results of this analysis for the mentioned benchmarks and for different error ratios. We observe that quality across individual dies shows a very small variance compared to the ensemble-average quality for each error ratio. Considering the Disparity benchmark again as an example, different chips with error ratio of 10^{-4} in the memory provide a quality variance of only 2 dB around the ensemble-average quality, which is 29 dB and shown with a dashed line. This negligible spread indicates that the average quality across multiple benchmark executions on the same die now matches the ensemble-average quality of the entire population of dies. However, while this new stability improves the quality for 50% of the dies, it also degrades the quality of the other 50% in the population. It is noteworthy that the small remaining quality spreed is due to the fact that the simplified fault randomization is not fully optimal.

2.5 Discussion

In the following, we consider the impact of the proposed randomization logic on the testability, the hardware complexity, and other types of variations.

2.5.1 Impact on Testability

The main advantage of the randomization lies in the impact on the required test procedure. Since each die with the same number of defects now provides almost the same quality, no parametric quality test is required anymore. In fact, the time-average quality of each die corresponds closely to the ensemble-average quality across the population of dies. Consequently, we do not need to run the application quality benchmark for each die, as shown in Fig. 2.6, anymore. Instead, only a straightforward standard test is performed on each die which counts the number of defects. The quality of that die then corresponds at least to the ensemble-average quality of the corresponding group of dies with the same number of errors, possibly discounted by a small margin to account for the imperfection of the randomization. This tremendously simplifies the test procedure without significant quality margins or variations.



(a) Convolution

(b) Disparity map

(c) Susan smoothing

(d) Susan corners

(e) MNIST with 1 layer

(f) MNIST

Figure 2.8: Benchmark yield-quality trade-off analysis for the fault model with four different error ratios while the fault for each chip is randomized during multiple execution of the benchmark. The dashed black line shows en ergodic ensemble-average quality for error ratio of 10^{-4}.

2.5.2 Impact on Hardware Complexity

In order to evaluate the hardware overhead of the proposed randomization logic, we integrate the randomization circuit for 32-bit wide SRAM blocks with different capacities. Area results after synthesis for a 28 nm fully depleted silicon on insulator (FD-SOI) technology are reported in Table 2.2. As the memory size increases, the randomization logic remains almost unchanged. Hence, the area overhead becomes already negligible for small memories of only 8–16 KBytes.

Table 2.2: Area overhead of the randomization logic on memories with different sizes.

SRAM size [KB]	**Area** [μm^2]	**Area with randomization** [μm^2]	**Area overhead** [%]
8	15572	16240	4.2
16	31144	31799	2.1
128	249152	250062	0.4

[1] SRAM blocks of 8 KB are stacked to build bigger size memories

2.5.3 Impact on Other Types of Variation

The proposed randomization scheme enable an ergodic quality across a population of dies despite having non-ergodic defects. Although the proposed scheme has originally been motivated by the need to combat the determisitic effect of the process variations, it can also be used as a means to combat the effect of other types of variation that are deterministic (or only slowly varying), but unknown or difficult to model and/or predict. As such, we briefly review some of these variations and how our proposed scheme could be useful in their context.

Process variation can lead to *deterministic but correlated variations* in some of the circuit parameters (e.g., threshold voltage or effective channel length [93]) that may cover a larger area of chip. For the memories, for example, this issue results in a cluster of near-by defected bits rather than distributed randomly-located defected bits due to the correlation in the fabrication process parameters [94]. To alleviate the issue, there have been significant efforts in modeling the variations more accurately and then considering the corresponding effect during the design [93–95]. Unfortunately, it is becoming increasingly challenging to provide such accurate models in more advance technology nodes and thus is difficult to compensate the resulting variations. Our proposed randomization scheme can be more easily used to mitigate these variations. More specifically, the data is shuffled across the memory and is stored on a random location during each read/write. Consequently, the correlation between the defected bits is broken and the cluster of near-by bits transforms to randomly distributed single bits across the memory.

A second example of deterministic (slowly varying) variations is the *aging effect* in *non-volatile memories*. In such memories, the available number of *program/erase cycles* for a reliable operation is limited and excessive stress on some of memory cells would worsen the retention time, which leads to a read failure for those cells [96]. Therefore, such memories are prone to stress-induced variations. To alleviate this issue, the memory controller relocates the data-blocks over the course of different write phases (program cycles) [97, 98]. However, some of the memory cells would still be stressed more frequently, which leads to a failure of those cells. Since the location of the failure changes only slowly, the proposed randomization scheme can be used to alleviate the effect of the deterministic induced errors by shuffling the data across all the memory cells.

A third example for deterministic and slowly changing variations are *local hotspots* in high-performance SoCs. High-performance SoCs, such as multi-core processors, host multiple processing cores, memory macros and interconnect networks on a single die or even on a 3D stack of dies. Due to the high integration density, the power density of such systems is also very high, which is among one of performance limiting factors [99]. Unbalanced workloads create energy hotspots in some of the cores or memory components, which results in reliability issues in certain locations of the chip [100]. To alleviate this issue, most of the processors monitor the temperature in multiple locations of chip and employ dynamic thermal management methods [101] accordingly to minimize the hotspots. While this measure is mainly used to avoid the creation of hotspots, the proposed randomization scheme can be used to mitigate the resulting errors from the remaining hotspots. More specifically, since hotspost change only slowly with the workload, the location of the induced errors are deterministic, which can be altered over time by using the proposed randomization scheme. This approach may on the one hand help to even avoid hotspots, but on the other hand also helps to restore an ergodic fault model. This restoration makes it more straightforward to consider the impact on quality as illustrated in this chapter.

2.6 Conclusion

Memories dominate the area and are the point-of-first-failure in many SoCs in advanced nanometer nodes. Since the design of 100% reliable memories is costly in terms of area, it is interesting to consider dropping the requirement of 100% reliable operation. Unfortunately, any deviation from the conservative design paradigm leads to quality differences between manufactured dies that are difficult to catch in a manageable production test. Previous studies on approximate computing with unreliable memories often neglect this issue. We show how the test issue can be avoided with a simple additional circuit that restores the beauty of random faults that are independent of the manufacturing outcome which equalizes the quality across dies. A complex parametric quality-test procedure is therefore no longer required even with unreliable memories.

3 Practical Approximate Channel Decoders with Unreliable Memories

The realization of complex and high performance systems has been enabled by very-large-scale integration (VLSI) technology scaling. Modern integrated circuits (ICs) in advanced nanometer nodes are able to accommodate extremely complex processing demand of many current and emerging applications. Among those applications, communication systems are a prominent example that require high-throughput data processing and high-capacity data storage, which are facilitated to a great extent by the most advanced process nodes. Unfortunately, the gain from technology scaling is diminishing and designers are pushed to look for new sources of computing efficiency. One of the promising approaches is exploiting the fault-tolerance of communication systems through approximate computing.

Communications systems are designed to operate reliably under channel noise and interference, which indicates an inherent degree of error-resilience in these systems. For this reason, such systems may also be able to cope with additional distortions introduced by unreliable computational resources [45]. In order to better realize the fault-tolerant behavior in communication systems, there have been significant efforts to understand the impact of faults in unreliable hardware on algorithm quality of service. Among the many different components in communications systems, decoders for error-correcting codes are a very good and promising example since they process stochastic signals that are often highly distorted by noise and/or interference.

Memory elements are the most energy- and area-consuming parts in most of the communication system kernels such as channel decoders, and the first point-of-failure while applying approximate computing techniques [50]. As such, in the context of communication systems it is typically assumed that any potential faulty behavior is due to an unreliable memory rather than unreliable logic. To enable the usage of unreliable memories in such systems, researchers have considered different type of memories while operating in a faulty regime.

An unreliable static random-access memory (SRAM) with dynamic quality management is presented in [40] that shows improved energy efficiency at the cost of reliability. An embedded dynamic random-access memory (DRAM) is proposed in [102, 103] for a low-density parity check (LDPC) decoder to achieve a better area and energy efficiency where the memory is designed such that the need for the periodic refresh is eliminated. A similar idea based on gain-cell (GC) embedded DRAM is proposed in [104] to implement high-bandwidth memories for an area and energy-efficient Viterbi decoder. All the above studies unfortunately consider the reliability of the memory itself by only focusing on the memory type or the underlying technology node without considering the algorithm-level reliability.

On the algorithm-level fault-tolerance of communication systems, some studies have focused on an inherent system-level robustness [45], or statistical fault mitigation methods extracted from the corresponding signal [105]. In addition to the system-level fault mitigation, fault-tolerance of different components, such as channel decoder, have been studied analytically under certain assumptions for memory and/or logic faults. In [106] the Gallager A and the sum-product (SP) algorithm for decoding of LDPC codes are analyzed under faulty decoding using density evolution analysis. A similar analysis is provided in [107] for the Gallager B algorithm. Studies of the widely used min-sum (MS) decoding with unreliable memories are presented in [108], [109]. The work of [110] shows that faulty decoding may even be beneficial in certain cases as it can help the decoder escape trapping sets that cause error floors. Other types of codes have also been studied in the context of faulty decoding. For example, the work of [111] examines faulty decoding of spatially-coupled LDPC codes, while the work of [112] studies faulty successive cancellation decoding of polar codes. Unfortunately, most of the mentioned references rely on a mathematical fault model with strict conditions on the ergodicity, independence, and symmetry of the hardware defects, which we will show are not realistic.

In turn, most studies on fault-tolerance of communication systems and channel decoders consider an average performance across both the input and the fault distribution assuming ergodic fault models. While such models are convenient and tractable in simulation and even with analytical tools, they do not necessarily reflect the actual failure modes of the real VLSI circuits, as explained in detail in the previous chapter. For example, although error locations within the memory array are random for a given memory size and operating point, the fault realization is deterministic for each die and very different from die to die after manufacturing, and therefore the ensemble-average performance across different dies is meaningless.

Contributions and Outline We propose an approximate LDPC decoder testchip, *ErgoDEC*, with unreliable memories that are build based on standard-cell memory (SCM) in a 28 nm fully depleted silicon on insulator (FD-SOI) technology, as a proof of concept to the non-ergodic fault model and the ideas initially proposed in the previous chapter. These are further

tailored and applied to an LDPC decoder in this chapter. More specifically, we propose and prove with measurement the efficacy of enabling an ergodic behavior despite the presence of deterministic post-fabrication faults by randomizing the faults in the memory. We also propose to further exploit the resulting random fault behavior to improve the decoder performance by repeating the unsuccessful decoding attempts. Additionally, the memory is designed in a way that the faults appear with a bias toward one logic state that is jointly optimized with the binary data representation according to the corresponding data distribution to minimize the number of effective errors which further improves the decoder performance.

The remainder of this chapter is organized as follows. In Section 3.1, we review the basics of LDPC codes and decoding and the faulty LDPC decoders in literature. We further argue that the widely-assumed ergodic model for the faults in these studies are inaccurate. Section 3.2 presents the baseline decoder architecture and the memory design specifications. We propose a realistic memory fault model and a performance evaluation methodology in Section 3.3, and our proposed improvement schemes to restore the ergodic behavior and minimize the fault effect across the population of dies are explained in Section 3.4 while the integration of the techniques to the ErgoDEC architecture is discussed in Section 3.5. The ErgoDEC chip architecure and the physical implementation details are presented in Section 3.6, the chip specifications and measurement results are provided in Section 3.7. Section 3.8 concludes the chapter.

3.1 LDPC Decoding and Faulty Behavior

3.1.1 LDPC Decoding Background

An LDPC code $\mathscr{C}$ can be defined by its $m \times n$ sparse binary parity-check matrix $\mathbf{H}$ as

$$\mathscr{C} = \left\{ \mathbf{c} \in \{0,1\}^n : \mathbf{Hc} = \mathbf{0} \right\}, \tag{3.1}$$

where additions are performed modulo-2 and $\mathbf{0}$ denotes the all-zeros vector of length m. LDPC codes can also be represented using a *Tanner* graph, which contains nodes of two types, namely variable nodes and check nodes, as explained in Chapter 1. A variable node i is connected to a check j if, and only if, $\mathbf{H}_{ji} = 1$. Quasi-cyclic LDPC (QC-LDPC) codes are a particular class of LDPC codes with a structured $M \times N$ block parity-check matrix that consists of cyclically shifted $Z \times Z$ identity matrices denoted by $\mathbf{I}^{\alpha}$, where Z is the lifting factor of the code and α denotes the shift value. For completeness, we also define the all-zero matrix

$\mathbf{I}^{\infty} = \mathbf{0}_{Z\times Z}$. With these definitions, we can define the parity check matrix

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}^{\alpha_{11}} & \dots & \mathbf{I}^{\alpha_{1N}} \\ \vdots & \ddots & \vdots \\ \mathbf{I}^{\alpha_{M1}} & \dots & \mathbf{I}^{\alpha_{MN}} \end{bmatrix}. \tag{3.2}$$

The corresponding shift coefficients of the parity check matrix are defined in the prototype matrix α of the code. Note that for QC-LDPC codes we have $n = ZN$ and $m = ZM$.

For decoding of QC-LDPC code, most practical hardware decoders use layered offset min-sum (L-OMS) decoding [113]. In the layered decoding schedule, first all the messages flowing into and out of the first layer (i.e., check node) are calculated. Then, the messages flowing into and out of the second layer are calculated, using the information that has already been updated by the first layer, etc. More formally, let Q_i denote the outgoing message at variable node i and let $R_{j,i}$ denote the corresponding incoming message from layer j. When processing layer j, the L-OMS algorithm calculates

$$T_i \leftarrow Q_i^{\text{old}} - R_{j,i}^{\text{old}}, \tag{3.3}$$

$$R_{j,i}^{\text{new}} \leftarrow \max\left(0, \min_{k\in\mathcal{N}_j/i} |T_k| - \beta\right) \prod_{k\in\mathcal{N}_j/i} \text{sign}(T_k), \tag{3.4}$$

$$Q_i^{\text{new}} \leftarrow T_i + R_{j,i}^{\text{new}}, \tag{3.5}$$

for every $i \in \mathcal{N}_j$, where $\mathcal{N}_j/i$ denotes the set of all variable nodes connected to check node j except variable node i, and β is an empirical correction factor called the *offset*. After the values have been updated, we set $Q_i^{\text{old}} \leftarrow Q_i^{\text{new}}$ and $R_{i,j}^{\text{old}} \leftarrow R_{i,j}^{\text{new}}$. An iteration is completed when all layers have been processed. The initial values for Q_i^{old} are the channel LLRs, i.e., $Q_i^{\text{old}} = \ln\left(\frac{p(y_i|x_i=+1)}{p(y_i|x_i=-1)}\right)$, where y_i is the channel output at codeword position i and x_i is the corresponding input. All $R_{j,i}^{\text{old}}$ are initialized to 0. When the maximum number of iterations has been reached, decoding stops and hard decisions are taken based on the sign of each Q_i^{new}.

3.1.2 Problems with Faulty LDPC Decoding Error Models

In the previous chapter, we showed that the typically-assumed expected (or average) performance $\mathscr{P}_{\text{avg}}$ is not a meaningful metric and does not reflect the actual reality of a population of faulty dies, as it does not distinguish between the behavior of a given die over different inputs (i.e., time) and the behavior of a population of dies, which could vary significantly. In this subsection, we use a specific case of the analysis in the previous chapter to formally describe the main issues with the error models in the related literature (e.g., [106–110] and references therein). Let $\mathscr{P}_{\mathscr{C}}(\mathbf{y}, \mathbf{e}, N, \ell)$ denote the codeword error indicator function of an approximate decoder for an LDPC code $\mathscr{C}$ of blocklength N when performing ℓ iterations with

inputs $\mathbf{y} = \mathbf{c} + \mathbf{n}$, where $\mathbf{c} \in \mathscr{C}$ is a codeword and $\mathbf{n}$ is additive noise, and a decoder memory error pattern $\mathbf{e}$. Then, $\mathscr{P}_{\text{avg}}(\mathscr{C}, \ell)$ is given by

$$\mathscr{P}_{\text{avg}}(\mathscr{C}, \ell) = \mathbb{E}_{\mathbf{y},\mathbf{e}} \left[\lim_{N \to \infty} \mathscr{D}_{\mathscr{C}}(\mathbf{y}, \mathbf{e}, N, \ell) \right] \tag{3.6}$$

$$= \mathbb{E}_{\mathbf{e}} \left[\mathbb{E}_{\mathbf{y}|\mathbf{e}} \left[\lim_{N \to \infty} \mathscr{D}_{\mathscr{C}}(\mathbf{y}, \mathbf{e}, N, \ell) \right] \right]. \tag{3.7}$$

The problem is that if we consider different instances of $\mathbf{y}$ to represent the time dimension (i.e., a new noisy codeword is given to the decoder at every time instant) and $\mathbf{e}$ to be the *chip ensemble* (i.e., the ensemble of all possible memory error patterns $\mathbf{e}$), then $\mathscr{D}_{\mathscr{C}}(\mathbf{y}, \mathbf{e}, N, \ell)$ is generally not ergodic, since the different memory error patterns of each chip can have very different (and fixed) effects on the decoder output. As such, the end-user of the decoder does not actually experience the $\mathscr{P}_{\text{avg}}(\mathscr{C}, \ell)$ quality-of-service (QoS) value that is estimated by most asymptotic analysis methods.

To better demonstrate the problem with ergodic fault model assumption in LDPC decoding, we consider an example and provide the corresponding error-correcting performance. We assume a parity-check matrix for a QC-LDPC code that uses blocks of size $Z = 111$ and has $N = 15$ block columns and $M = 3$ block rows. Further, we assume the decoder architecture in [114], which implements L-OMS decoding according to (3.3), (3.4), and (3.5). We run the decoder with 10 iterations and we assume fixed-point bitwidth of $N_Q = N_T = N_R = 6$ for the Q-, T-, and R-memories, respectively.

Fig. 3.1 shows the frame error rate (FER) vs. $\mathrm{E_b/N_0}$ performance of 5 decoder instances (i.e., chips) with different fault maps using a deterministic stuck-at fault model with probability of $P_b = 5 \times 10^{-5}$, which is associated with total two faulty bits in Q-, T-, and R-memories (see section 3.3 for more details on the error model). We also plot the FER performance of the non-faulty decoder in this figure as a reference. The results indicate that the performance of decoders with distinct fault maps can be very different, and therefore, the average ergodic performance is not a relevant metric to capture the behavior of the population of faulty decoder dies.

3.2 LDPC Decoder Architecture

In the section, we describe the decoder architecture that serves as the baseline architecture of the implemented test chip, ErgoDEC, which is used to better understand the effect of memory faults on the decoder performance. The LDPC decoder is based on the previous work [114]. Therefore, we summarize the basic functionality to a degree that is required for the purpose of this chapter. We then explain in detail how the faulty memories are designed and how the baseline architecture is extended for memory fault injection in ErgoDEC.

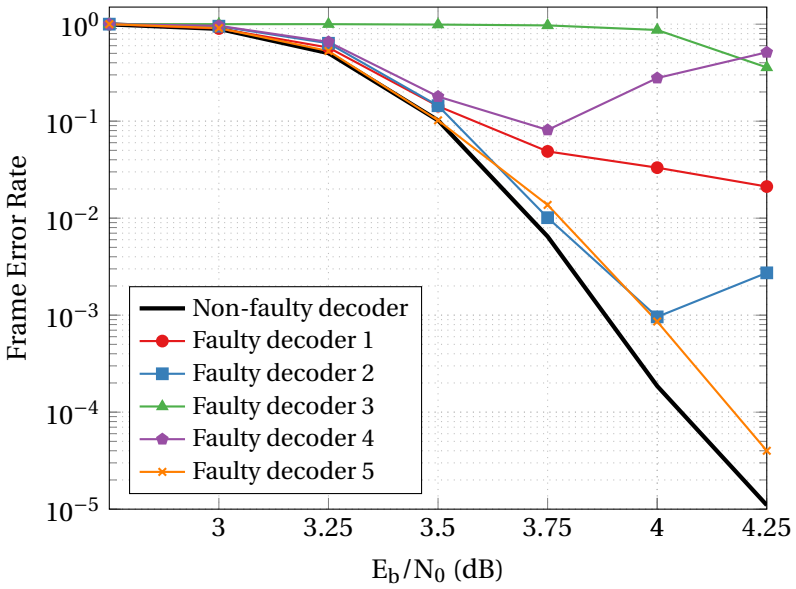


Figure 3.1: Frame error rate of 5 randomly selected faulty LDPC decoder instances compared with the frame error rate of a non-faulty LDPC decoder.

3.2.1 Reference Architecture

The decoder reference architecture is illustrated in Fig. 3.2. The main building blocks of this architecture are *processing units*, which contain Z MIN and Z SEL units and will be explained in the following, a *shifter*, which implements the cyclic shift required for processing each block of the QC-LDPC parity check matrix, and Q-, T-, and R-memories, which store the messages in the L-OMS algorithm, as explained in (3.3), (3.4), and (3.5). The layers are processed one after another while the processing of each layer is decomposed into multiple cycles. The architecture process Z independent check nodes of a layer simultaneously, using Z processing units. To this end, the corresponding Z Q- and R-values are read from the associated Q- and R-memories while Q-values are shifted using cyclic shifter based on the entries of $\mathbf{H}$. The temporary T-values of (3.3) are calculated by the MIN units and stored in the T-memory. Once the MIN units have finished processing all non-zero blocks in a row of the block parity-check matrix, the SEL units use the resulting minimum and second-minimum, sign and T-values to update Z R-values and Q-values according to (3.4) and (3.5). In a purely sequential architecture, the MIN and the SEL phase require one cycle per column in the prototype matrix of the LDPC code, i.e., a total of N clock cycles for each phase. However, the MIN units can start processing blocks of the next row which do not have data dependencies with the rows on which the SEL units are currently operating on. Therefore, by overlapping the two phases of two layers, a single layer can be processed with only $N+2$ clock cycles.

In addition to this overlap, the decoder throughput is increased by using a semi-parallel architecture for processing each layer by increasing the number of processing units as described

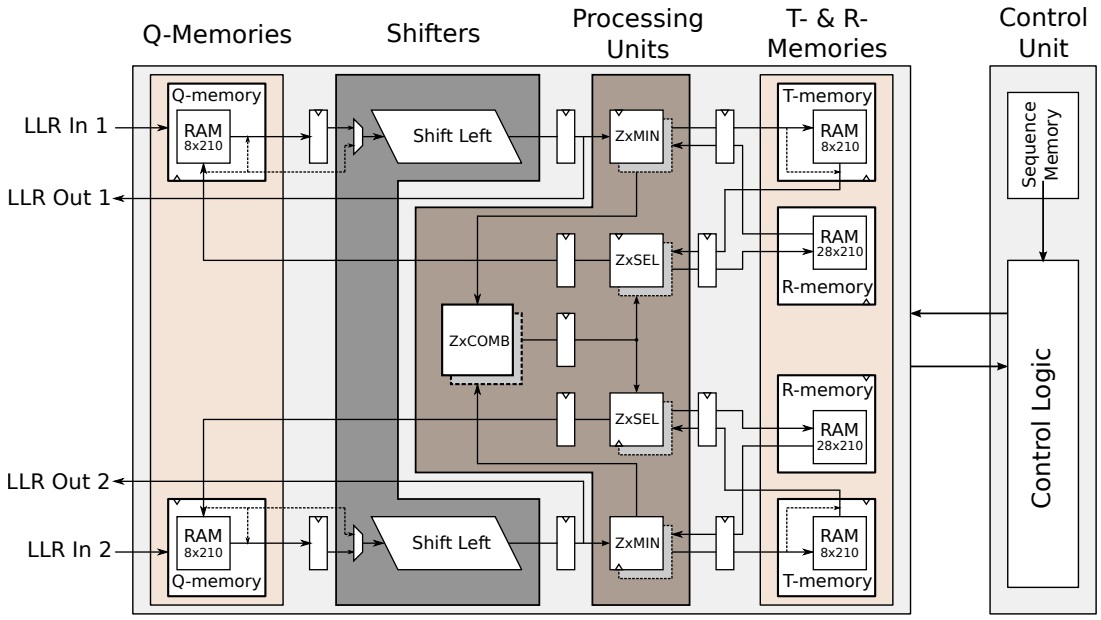


Figure 3.2: Semi-parallel QC-LDPC decoder [114] used as the reference architecture for the implemented chip.

in [114]. In this architecture, we always consider two columns of the prototype matrix in parallel to determine two T-values and to update the minimum and the second minimum with two new candidates. This measure effectively almost cuts the number of cycles per layer into half, i.e., to $\lceil N/2 \rceil + 2$.

In addition to the datapath of the decoder, Fig. 3.2 also shows the control logic. In order to provide the flexibility to support multiple codes, this logic uses a microprogram that is stored in a small sequence memory to manage the datapath. Every iteration of the decoder is controlled by a command from the sequence stored in the memory. The sequence is created offline based on the parity-check matrix of the codes. Every command of the sequence contains all memory addresses which have to be read from and written to, as well as information on which pipeline stages have to be stalled and at which stages forwarding or memory bypassing has to be performed (due to dependencies between rows). One command is issue per cycle and the information contained in each command propagates through the pipeline. The sequence length L is defined as the number of commands that have to be issued in order for the decoder to complete a full iteration of the L-OMS algorithm.

3.2.2 Memory Design

The decoder architecture includes *dynamic SCMs* for the datapath memories. SCMs are memory arrays that are synthesized from standard cells as first proposed in [115] and explained in Chapter 1. The internal architecture of an SCM is essentially a 2D array of latches that

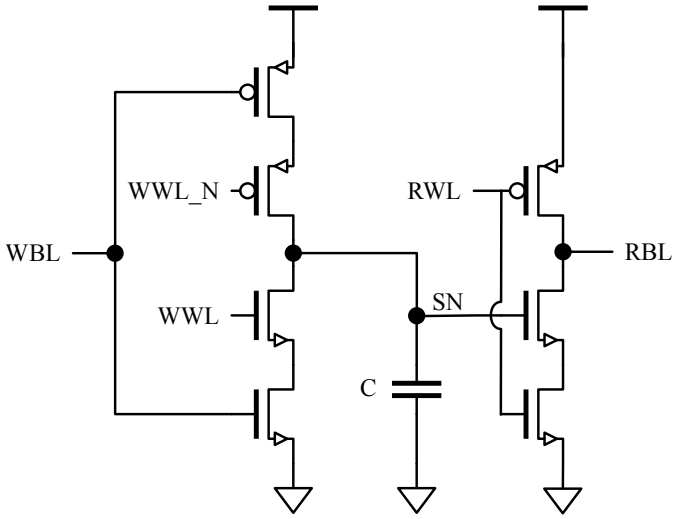


Figure 3.3: GC latch for dynamic SCM.

store the data while each row of latches corresponds to a memory word. The concept of using dynamic SCM as a memory in an accelerator was discussed in [102, 103]. It is widely recognized that they have significant advantages over conventional SRAMs, especially for small macros in accelerators, in terms of power consumption, robustness to voltage scaling, and data locality. SCMs provide an interface that is comparable to that of a two-port SRAM with one read and one write port.

The core component of the ErgoDEC datapath memories is a specific type of SCM, in which the latch is realized based on a dynamic storage mechanism, i.e., a dynamic GC latch, as in [103]. The latch has an integrated NAND gate for the AND-OR MUX tree SCM read-multiplexer as shown in Fig. 3.3. In this latch, the logic level of the write-bit-line (WBL) is copied onto the parasitic capacitance (C) on the storage node (SN) whenever the write-word-line (WWL) and its complement (WWL_N) are asserted. While the read-word-line (RWL) is inactive, the output read-bit-line (RBL) is always driven high and has no impact on the OR tree of the SCMs output MUX. When RWL is asserted, the state of the SN decides on the output level (0 or 1) on the RBL.

This dynamic latch requires 7 transistors as compared to the conservative static CMOS latch that comprises 12 transistors, owing to the fact that the dynamic latch does not include a feedback that maintains the state of the GC storage node. Hence, the charges that are stored on the storage node leak away over time and the memory loses its content when no new data is written into the node. Therefore, it requires periodic refresh operations to avoid the loss of data. The refresh period for a GC based memory is determined by its *data retention time (DRT)* [59, 116] that is defined as the maximum time interval between a write operation and a successful read operation. Both the parasitic capacitance C as well as the leakage currents determine the DRT of the latch. Although the dynamic storage can be used for

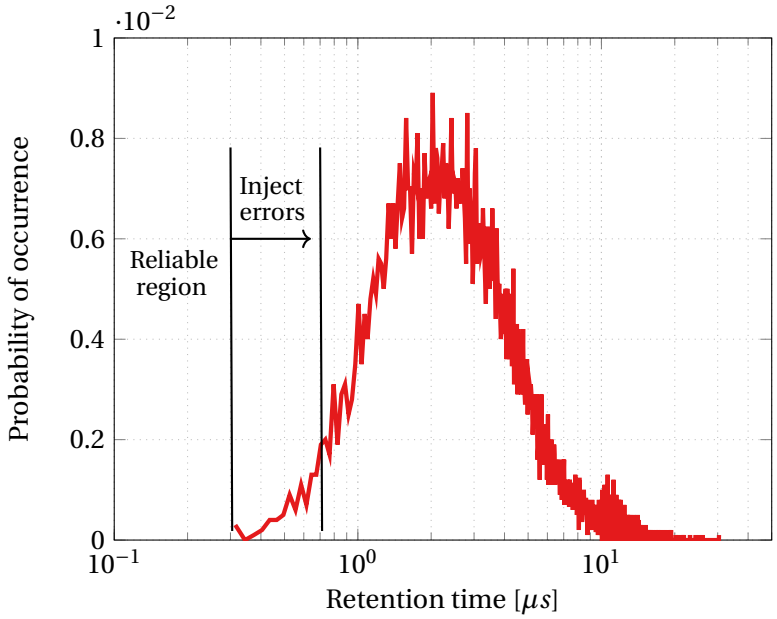


Figure 3.4: DRT distribution for the proposed GC acquired from a Monte-Carlo simulation on a memory with a 10 kbit size and in a typical operating condition.

reliable operation as shown in [102, 103], they can also be used for a controlled fault injection by relaxing the corresponding refresh period and violating the DRT of the dynamic storage elements, as explained in the following.

3.2.2.1 Fault Injection Mechanism

Sub-threshold leakage is the most prominent and has the most substantial impact on the DRT of the above GC latch among the different leakage mechanisms [59]. Since this leakage depends exponentially on the threshold voltage V_T of the write access transistors, which is a manufacturing parameter that is subject to inter-die and intra-die variations, it varies substantially, both between different chips and also between different bit-cells on the same chip, which results in different DRT. The probability distribution of the DRT values for the bit-cell of Fig. 3.3 is shown in Fig. 3.4. The distribution is acquired based on a Monte-Carlo simulation for a memory with a 10 kbit size and a typical operating condition, which shows the large variation among the DRT values. Further, the DRT distribution has a long tail toward zero [117], which leads to a conservative and costly margin in a reliable design approach since it requires consideration of the bit-cell with the worst-case (shortest) retention time across dies, operating conditions, and bit-cells within the array on the same die. However, we purposely exploit these long tails here as the key to enable a graceful degradation of the reliability, i.e., a slow increase of the number of failing bits, over a large tuning range for the lifetime of variables in the memory by reducing the frequency.

Table 3.1: Data lifetime in the T- and R-memories for the considered QC-LDPC code with $N = 15$ and $M = 3$.

Memory	Data lifetime	Min clk for reliable operation with a min DRT= $0.3\,\mu s$
T	$10 \cdot T_{\text{clk}}$	33.3 MHz
R	$30 \cdot T_{\text{clk}}$	100 MHz

3.2.2.2 Data Lifetime in the Memories and Fault Injection

The errors occur in the memory due to DRT violation of the dynamics SCMs, and therefore, timing characteristics of the decoder architecture are particularly relevant. Such errors occur depending on the lifetime of the data in a memory, which is given by the number of cycles between a write and the last read to the same address prior to the next write, N_c, and the period of the clock: $T_{\text{life}} = N_c T_{\text{clk}}$. In the decoder architecture, R-values are refreshed once in each iteration and the T-values are refreshed in each layer, assuming that the prototype matrix of the code does not include $\boldsymbol{I}^{\infty}$ elements. Since the employed semi-parallel architecture processes each layer in $\lceil N/2 \rceil + 2$ clock cycles, the R-values lifetime is $M(\lceil N/2 \rceil + 2) T_{\text{clk}}$ and the T-values lifetime is $(\lceil N/2 \rceil + 2) T_{\text{clk}}$. We summarize the data lifetime values for the considered QC-LDPC code in Table 3.1.

Dynamic bit-cells are designed to provide a very large DRT margin compared to the lifetime of the messages stored in the decoder memories when the decoder operates near its maximum frequency. Therefore, the minimum clock frequency for a reliable operation assuming a minimum DRT is well-below the decoder maximum frequency. This minimum clock frequency is provided in Table 3.1 for a minimum DRT of $\approx 0.3\,\mu s$ from the distribution of Fig. 3.4. This margin ensures that all bits are stored reliably, even for bit-cells with a comparatively high leakage and thus a short DRT. To inject errors, based on the process variations, we can increase the clock period and thereby increase the data lifetime in the memory without changing the DRT of the bit-cells, as illustrated in Fig. 3.4. Due to the long tail of the DRT distribution, the number of failing bit-cells will increase only slowly while lowering the clock frequency enables a gradual increase in the number of failing bit-cells.

3.2.2.3 Selective Protection of Sensitive Bits and Memories

The decoding performance is particularly sensitive to faults that occur in the sign-bit of the sign-magnitude encoded messages. We, therefore, establish a scheme in which the sign-bit (MSB) is always protected and errors can only happen in the magnitude of the messages. This measure ensures a more graceful quality degradation since the most significant bit is protected [118].

We further choose to implement the Q-memories in a fully reliable fashion, while enabling fault injection only for the T- and R-memories. This choice is base on the fact that theoretical studies on the robustness of LDPC decoders in the presence of errors typically assume that errors are injected into the messages, which is aligned with injecting errors in R-memories and the values in T-memories, rather than in Q-memories that stores the (more critical) intrinsic log likelihood ratios (LLRs). In addition to this reason, the data lifetime in the Q-memory may be significantly larger than in the T- and R-memory as the Q-memory is implemented as a ping-pong memory that is used to pre-load channel LLRs and to unload decoding results. This loading and unloading operation may take up to one entire decoding round with multiple iterations. To avoid a loss of data, we implement this memory in a fully reliable fashion using a standard SRAM.

3.2.2.4 Hybrid Static/Dynamic SCM

The T- and R-memories in the decoder architecture are constructed from a hybrid SCM as required to specifically protect the sign-bit for our design. These hybrid SCMs combine conventional static latches for the sign-bits with dynamic GC latches, as explained in the above, for the bits encoding the magnitude of each data word. More specifically, the T- and R-memories are each $Z\,N_R$ ($N_T = N_R$) bits wide and each LLR word of N_R bits contains an MSB implemented as static latch and $N_R - 1$ bits implemented as dynamic GC latch.

3.3 LDPC Decoder Quality Assessment Under memory Faults

In the previous chapter, it has been widely discussed that the ergodic fault model for the memory misbehavior and the associated average-quality performance metric is flawed. Instead, the quality-yield assessment methodology was proposed, where the performance is evaluated across the population of dies and input data and the results were illustrated using the inverse cumulative distribution function (ICDF) of the quality for the population of faulty dies. Since, the memory faults affect the LDPC decoder performance in a similar fashion to the evaluated benchmarks in the previous chapter, i.e., the FER of the decoders with different fault modes are very different as discussed and illustrated in Section 3.1 of this chapter, we apply the described methodology to the LDPC decoder in this chapter. We further discuss a simplified realistic memory fault model that is used throughout the analysis of this chapter.

3.3.1 Simulation Environments

In order to obtain a more meaningful understanding of the memory faults on the error-correcting performance of the decoder, we perform the analysis over a population of dies, similar to our proposal in the previous chapter. This analysis generates population of dies $n \in \mathcal{N}$ with their individual fault patterns $\mathbf{e}_n$ and studies the time-average performance for

each die. More specifically, two nested loops are used to evaluate the performance of the decoder dies. While the outer loop iterates over different dies, the inner loop iterates over input statistics and averages over time to evaluate the error correcting performance of the each decoder die.

We use a simulation model for the decoder, which is a bit-true model of the actual fixed-point architecture, considering also the chosen number representation. Note that this is necessary to model the exact behavior of what is actually stored in memory. Particularly, temporal values T_i are derived and stored (in the T-memories), variable-to-check messages are never actually stored as they are derived locally from previously stored check-to-variable messages (in the R-memories) and from separately stored intrinsic LLRs (in the Q-memories), as explained in (3.3), (3.4), and (3.5). Further, the faulty decoding is enabled by applying (injecting) bit errors during each memory read (in each decoding iteration) according to a *fault map* that describes the fault model realization for each die.

3.3.2 Memory Fault Models

The error probability of each bit in the memory depends on many parameters that are related to the specific memory circuit design, the process, and the operating conditions. We use the bit-error probability P_b, to abstract the implementations and as the input to our simulations [119]. Further, we consider two fault models for the LDPC decoder simulation set-up. The first one is the *ergodic* model based on the typically-assumed model in the related literature, e.g., [106–110], where the errors are assumed to be *random bit-flips* that are independent and identically distributed (i.i.d.) in the memory and appear and disappear randomly over time. However, we modified this model according to the fact that memory errors are always stuck-at, and therefore, we consider an i.i.d. random stuck-at model with equal probability for both polarities and the manufacturing error distribution probability of P_b.

The second model is chosen to better reflect the reality and is more accurate than the commonly assumed ergodic i.i.d. model. To this end, we consider a *non-ergodic* model with *deterministic stuck-at* errors in the memory, where the exact position of the error in the fault map is chosen with a uniform distribution and the error polarities are chosen with equal probabilities for the realizations of the fault map. More specifically, we generate a population of decoders where the T- and R-memories follow this model, but each decoder remains unchanged during the entire simulation. This model is based on the observation that errors are different for each memory as different outcome of the production process, however, they remain stable for that specific memory over time. We will confirm this model later by providing measurement results in Section 3.7.

In addition to the location and polarity of errors, the number of errors K_e for a given manufacturing bit-error probability P_b that appear in a memory instance of a given size N_b is described

by a binomial distribution [119] as

$$K_e \sim \binom{N_b}{K_e} P_b^{K_e} (1-P_b)^{N_b-K_e}. \tag{3.8}$$

We however note that for small bit-error probabilities this distribution is very peaky. Hence, there are only very few relevant groups for the number of failing bits that are weighted by the probability of occurrence depending on the memory size. We approximate these few groups by only one and we define *error ratio* equal to the bit-error probability. Given this approximation, and by multiplying this error ratio to the memory size $P_b N_b$ a fixed number of errors are dictated, which are actually injected in a memory across all instances of the simulation for both of the error models.

3.3.3 Quality Assessment Metrics for LDPC Decoder

For most of benchmarks, when considering the issue of unreliable hardware, quality is assessed based on the deviation from the result by operation of the algorithm on a fully reliable implementation. This deviation can be measured, for example, based on an the mean square error, as it was shown in the previous chapter. However, defining such a metric for a channel decoder is more complex due to the fact that: i) the common quality metric for a decoder, i.e., the FER, is a probability and an Euclidean distance metric to measure the deviation from a probability for the fault-free decoder is not easy to interpret, and ii) the FER performance metric needs to be evaluated over a range of different signal-to-noise ratios (SNRs), and therefore is no longer a scalar.

In the light of the above considerations and inspired by the quality metrics defined in the previous chapter, we define two metrics that provide information on the quality-yield of a channel decoder that is affected by reliability issues:

1. FER-Yield: The FER-yield is based on the assumption that a communication system has a target operating SNR, which is used for performance assessment, but can/should not be changed. This operating SNR is typically chosen to achieve a certain target FER in a regular reliable implementation. Since the SNR is given, we must assume that a certain error-rate performance deviation from the target FER is permissible under hardware errors. We therefore analyze the fraction (yield) of decoders as a function of a relaxed target FER.

2. SNR-Yield: the SNR-yield is based on the assumption that a communication system is build to operate at a fixed target FER, obtained from specifications. The SNR at which this target FER is achieved defines the quality of the implementation. By accepting unreliable operation of the hardware, we accept a certain degradation in the SNR at

which the target FER is reached. We therefore analyze the fraction (yield) of the decoders as a function of the relaxed SNR at which the target FER is reached.

3.3.4 Quality-Yield Results

We demonstrate the performance result for a population of decoder dies using the above assessment methodology and the explained fault models. These experiments are similar to those of the previous chapter, but they are provided for the case of an LDPC decoder with the objective to show their validity with the measurement results in Section 3.7. To this end, we again consider the QC-LDPC code parity-check matrix mentioned at the beginning of this chapter and recall that it uses blocks of size $Z = 111$ and has $N = 15$ and $M = 3$ block columns and rows, respectively (rate = 0.8). Further, we run the decoder with 10 iterations and we assume $N_Q = N_T = N_R = 6$, which results in 8325, and 24975 bits for the faulty part of the T- and R-memories, respectively.

Fig. 3.5 shows the FER vs. SNR for 50 decoder instances representing different realization of the fault model in the population of dies for the two described memory fault models with $P_b = 5 \times 10^{-5}$. The black curve corresponds to the ergodic fault model. The red curves illustrate the performance of the decoders with the non-ergodic fault model and reflect the fact that each die is born very differently with a fixed set of faulty bits. As can be seen from the figure, the ensemble-average performance across the dies is meaningless and does not represent the reality of decoders with non-ergodic faults.

To better demonstrate the difference among the two models, we show the quality-yield results for 1000 decoder instances in Fig. 3.6. The empirical cumulative distribution function (CDF) of the FER at a fixed SNR of 4 dB is shown in Fig. 3.6(a). As it can be seen, the decoders with the non-ergodic fault model span a very large performance range while the decoder with the ergodic model shows only one average-performance. The fraction of decoders with a target FER $< 10^{-3}$ is shown in Fig. 3.6(b). As it can be observed, the required SNR to reach the target FER is different across the population of decoders and even 30% of them can never reach such a target FER. However, the ergodic quality model erroneously suggests that the target FER can be reached, even though with an SNR penalty.

To better understand the effect of memory faults for both models, we analyze the impact of the probability of error P_b in the memory on the decoder performance. For this, we use the quality-yield performance metrics for the considered code. We note that this analysis serves mainly to understand the impact and sensitivity of a change in the number of errors on decoder performance. While, the assumed probability of errors are not derived form a specific type of memory circuit, they are consistent with typical assumptions on error rates of memories with reliability issues over multiple operating conditions.

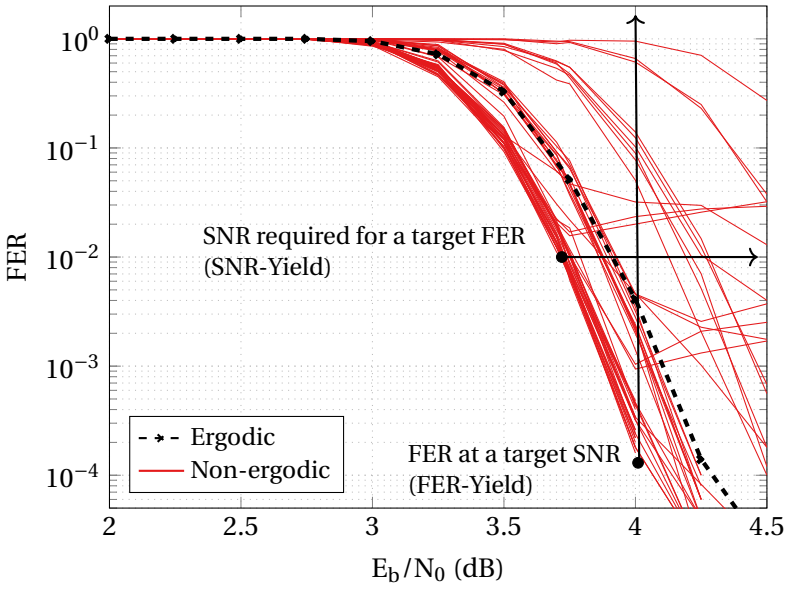


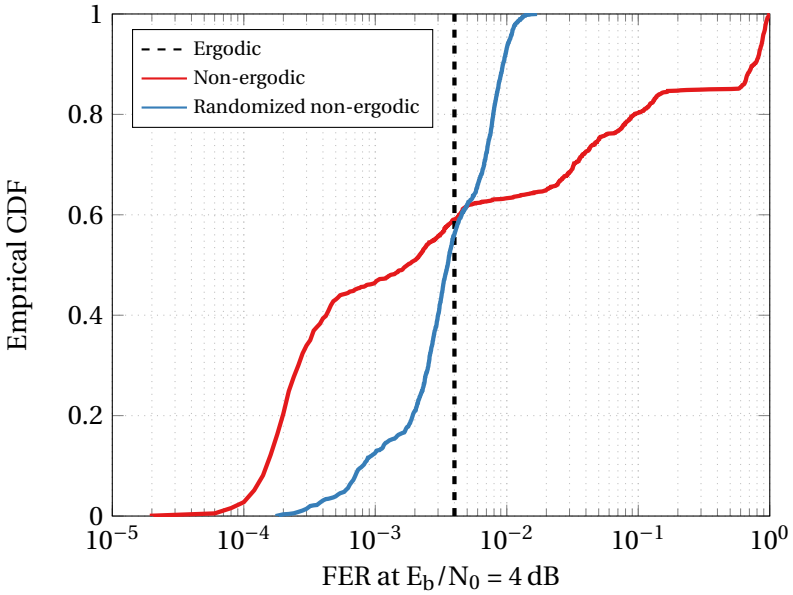
Figure 3.5: Performance evaluation of a population of faulty LDPC decoders through FER vs. E_b/N_0 for ergodic and non-ergodic fault models with bit-error probability $P_b = 5 \times 10^{-5}$.

Fig. 3.7 shows the FER-yield at a similar target SNR of 4 dB and the SNR-yield at a similar target FER of $< 10^{-3}$ for two bit-error probabilities of $P_b = 5 \times 10^{-5}$ and $P_b = 1 \times 10^{-4}$. From the FER-yield, we can see that the higher probability of error results in a lower yield at the given target SNR. Also, from the SNR-yield result, we can observe that the 2× increase in the number of errors leads to an SNR penalty for the ergodic yield model and results in a yield loss for the non-ergodic model. We note that as opposed to the decoder with ergodic fault model, for the non-ergodic case with a higher error probability, the percentage of decoder instances that can reach the target FER even at a high SNR is actually low. This observation can be explained by the fact that for a larger number of errors it becomes more likely that at least few errors land in critical parts of the memory with a non-negligible impact on the overall quality.

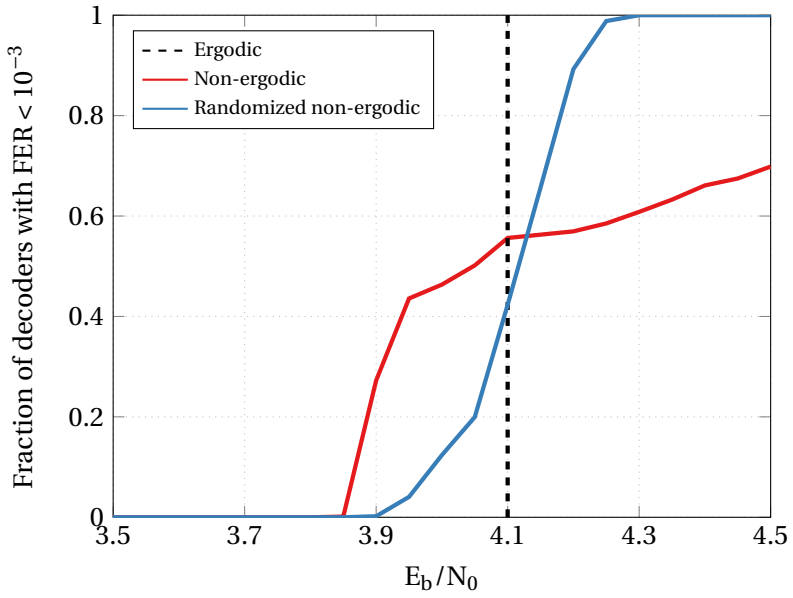
3.4 Improving the Performance Across the Population of Dies

The main issue with the ergodic fault model is that memory errors are mostly deterministic after manufacturing for each individual die, which results in a deterministic, but different performance for each die. Among these dies, there is a considerable quality variation, which would invalidate any average-quality analysis and complicate the quality-yield characterization.

In this section, we discuss our proposed measures to improve the performance across the population of decoders. First, we propose to restore the ergodic behavior across the memory faults by restoring the beauty of randomness, similar to the previous chapter, while we verify

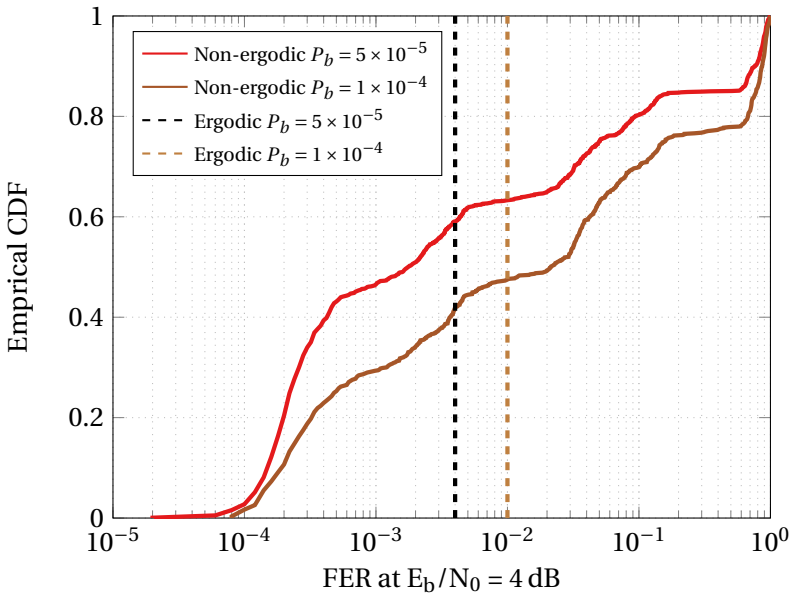


(a) FER-Yiled

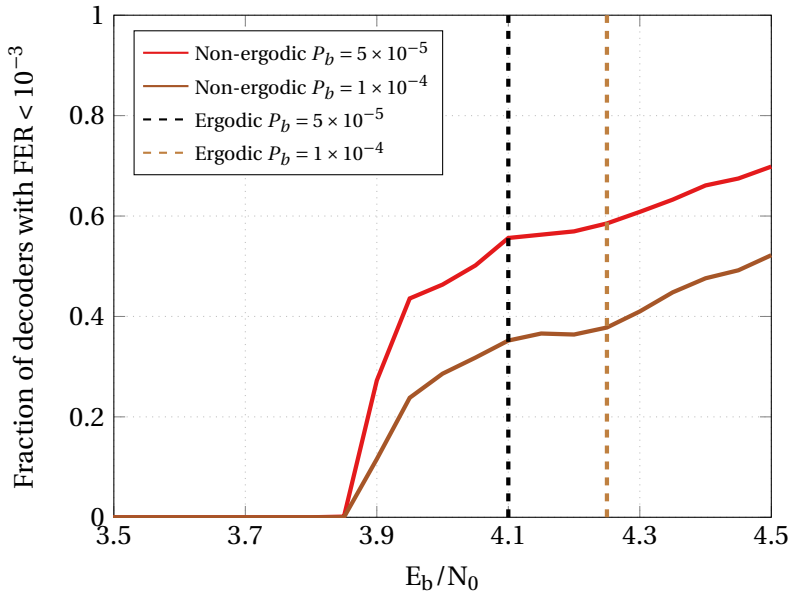


(b) SNR-Yiled

Figure 3.6: Performance evaluation of a population of faulty LDPC decoders through a) empirical cumulative density function of FER at a fixed $E_b/N_0 = 4$ dB, and b) fraction of decoders with FER$< 10^{-3}$, for ergodic and non-ergodic fault models with bit-error probability $P_b = 5 \times 10^{-5}$.



(a) FER-Yiled



(b) SNR-Yiled

Figure 3.7: Performance evaluation of a population of faulty LDPC decoders through a) empirical cumulative density function of FER at a fixed $E_b/N_0 = 4$ dB, and b) fraction of decoders with FER$< 10^{-3}$, for bit-error probabilities of $P_b = 5 \times 10^{-5}$ and $P_b = 1 \times 10^{-4}$.

the effectiveness of this idea with silicon measurement later in Section 3.7. Next, we propose to exploit the randomized memory faults and the resulting behavior of the decoder to improve the performance by repeating the decoding attempts for unsuccessful codewords. Finally, we propose to jointly optimize the binary data representation and memory faults to reduce the number of effective faults and further improve the decoder performance.

3.4.1 Restoring the Ergodic Behavior

Motivated by the above observation and given the fact that memory errors in each individual die are deterministic and thus any averaging across the dies is not legitimate for performance evaluation of each die, we propose to modify the memory faults in a way that the behavior of each die alters over time. More specifically, we propose to randomize the errors between independent subsequent codewords as well as between the subsequent decoding iterations of a codeword. This measure provides a different realization of a random fault map for each execution of the decoder and leads to a more ergodic quality behavior of the faulty hardware. As a result, the time-average behavior of each decoder die corresponds better to the chip ensemble-average over the population of decoder dies. In another words, while the quality of some decoders with a low FER penalty compared to the fault-free decoder degrades, the quality of others with a high FER penalty improves. Overall, the quality variance significantly shrinks, which allows to guarantee a significantly better minimum-quality.

In order to realize a random behavior for the decoder's faulty memories, error locations should be moved across the memory arrays. Additionally, error polarities should be altered randomly to provide randomness in the stuck-at polarity. Since errors cannot be moved freely across the memories, we propose to shuffle the bits in an LLR, shuffle LLRs across a memory word, and shuffle the word addresses over different decoding iterations and codeword decodings. This measure creates a logical memory with a different fault map over time while the physical faults remain the same. If the shuffling is performed randomly each decoding iteration experiences different fault maps, i.e., an almost ergodic process.

Fig. 3.8 illustrates how the proposed randomization scheme is effecting the memory. In this figure, the physical view of the memory is shown on the left with errors in address and bit-index pairs of (2, 2), (4, 5), and (9, 4). By randomizing the physical memory address and the bit index, a logical memory is created that shows a different fault map. Three examples of this logical view are provided on the right side of the figure. In the first example, which corresponds to one realization of the randomization, the above physical address and bit-index pairs are converted into logical pairs of (9, 5), (5, 4), and (2, 1), while this conversion is different for other realizations. Since the logical faults are relevant from the viewpoint of the decoder, the proposed method converts a non-ergodic fault map into an ergodic process.

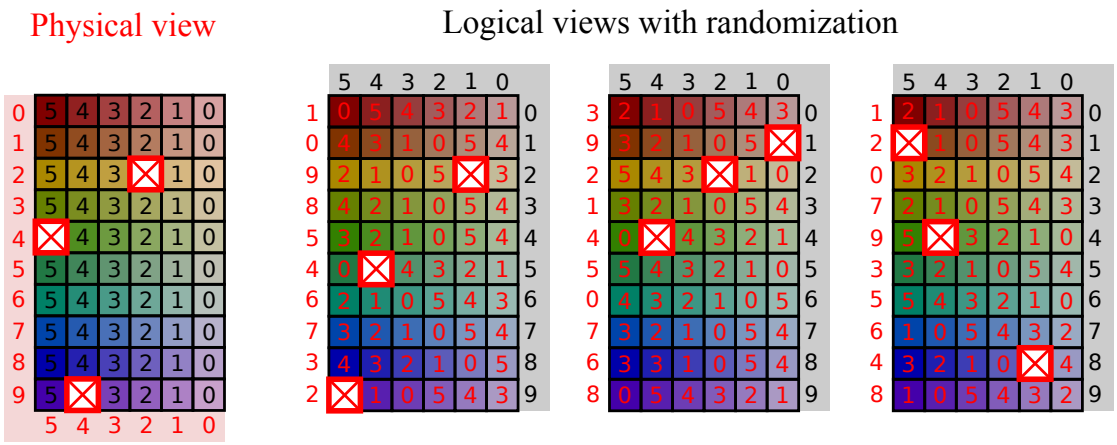


Figure 3.8: Different logical memory fault maps (right) are created for a memory with a constant physical fault map (left).

We re-evaluate the performance of the decoder using the simulation environment while the decoder simulation model is verified so that the memory faults are randomized, as explained. The curves in Fig. 3.6 that are colored in blue, show the quality-yield for 1000 decoders while the proposed memory fault randomization is used during the decoding. In the FER-yield result, we observe that the variance across different dies becomes very small and is heavily reduced as compared to the plot corresponding to the non-ergodic fault model, colored in red. This smaller performance variance indicates that the fault behavior becomes ergodic, and therefore, the time-average behavior for each decoder approximately matches the ensemble-average behavior of all the decoder dies. We can also observe that the performance of inferior decoder dies improves and matches the chip ensemble-average performance. This observation is better illustrated in the SNR-yield results. We observe that all the decoders with a randomized fault model can reach the target FER with a difference in the required SNR, while at least 30% of the decoders with non-ergodic fault model fail to reach this FER even at a very high SNR. Therefore, a minimum performance can be guaranteed for all the dies with the ergodic behavior.

3.4.2 Improving the Performance by Exploiting the Random Behavior of Logical Faults

The proposed randomization technique essentially converts the deterministic physical memory faults into random logical faults. In other words, each decoding attempt experiences a different fault realization, which results in a similar time-average quality across multiple decoder dies, as already discussed. In addition to this ergodic behavior of the decoders, the randomized faults are (ideally) independent from each other, which would result in an independent behavior for different decoding attempts even with an identical codeword. This

property can be exploited to improve the decoder performance, which provides the motivation for our proposition.

Recall that if multiple events B_i are independent the following holds

$$\Pr\Big(\bigcap_i B_i\Big) = \prod_i \Pr(B_i). \tag{3.9}$$

In another words, the joint probability of multiple independent events is the product of the probabilities, which is always smaller than each of their individual probabilities. We therefore propose to exploit the relation in (3.9) to reduce the probability of failure in the decoder. Specifically, we propose to repeat the decoding for the codewords that are unsuccessfully decoded with a different realization of the randomized faults. Since the decoding attempts are (ideally) independent from each other as the result of independent logical faults, the joint probability of an unsuccessful decoding over all repetitions decreases as compared to one repetition. For example, it is less likely to have two subsequent decoding failures as compared to only one failure attempt. Therefore, by repeating the decoding attempts, it becomes more likely that one of the decoding attempts succeeds. In practice, the repetitions can continue until a maximum is reached or the codeword is decoded successfully.

We evaluate the performance over the population of decoders with the randomized non-ergodic faults while we enable the above-explained repetition for the unsuccessful decoding attempts. Note that the unsuccessful decodings can be trivially recognized by monitoring the syndrome (see (3.1)). We allow up to 1, 2, or 3 extra repetitions and we show the FER-yield results in Fig. 3.9. By comparing the plot with 1 extra decoding attempt, colored in green, against the reference plot without any extra attempt, colored in blue, we can observe a significant improvement in the decoder performance, which is up to an order of magnitude for some of the decoders such that the FER penalty compared to the non-faulty decoder, shown with a black dashed curve, becomes small. The improvement saturates as we move to higher number of repetitions due to the fact that the decoding attempts are not completely independent, as they still process an identical codeword. We further see that the variance across multiple decoders is reduced compared to the reference plot since the inferior decoders (with higher frame error probability) get more chances to repeat the decoding as compare to the superior decoders (with lower frame error probability). Such a lower performance variance indicates a higher yield at a target FER. We note that the key ingredient for the success of this idea is the proposed randomization techniques as it allows to realize different independent memory faults and enable the above improvement, while the performance of a decoder with deterministic memory faults would not change by repeating the decoding iterations.

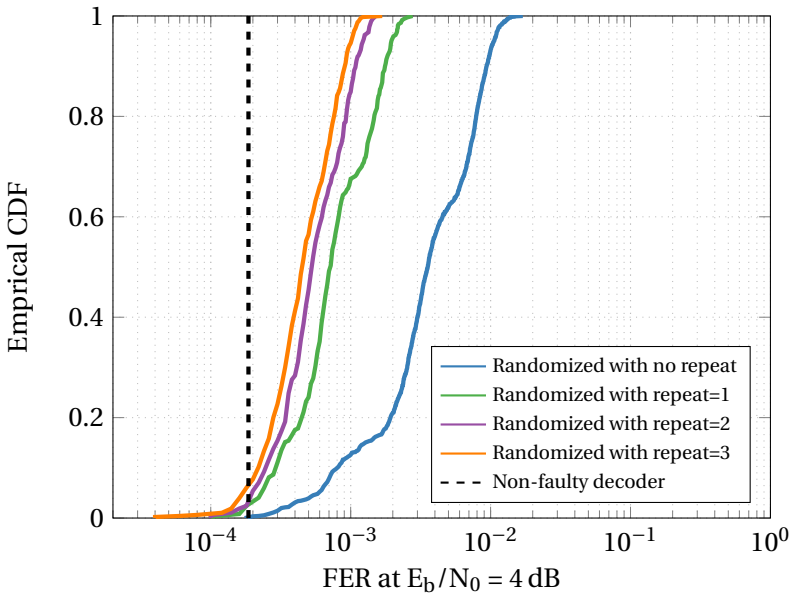


Figure 3.9: Performance evaluation of a population of faulty LDPC decoders through empirical cumulative density function of FER at a fixed $E_b/N_0 = 4$ dB, for non-ergodic fault model with bit-error probability $P_b = 5 \times 10^{-5}$ with fault randomization, while the unsuccessful decoding are repeating 1, 2, or 3 times.

3.4.3 Minimizing the Impact of Memory Faults by Exploiting Binary Data Representation

The non-ergodic fault model reflects the reality of memory faults, where the error positions are fixed after manufacturing, as discussed. In order to minimize the impact of deterministic faults, we adapt the proposition in [119] according to the data distribution in the decoder memories. While [119] only considers binary data representation optimization, we jointly consider and optimize memory faults and the memory binary data representation. To this end, we first define the error probability for each bit of the memory considering the non-ergodic model in a more general form than Section 3.3.2 and then discuss the proposed idea.

3.4.3.1 Formal Definition of The Bit-Error Probability and Memeory Error Model

In Section 3.3.2 we assumed that the bit-error probability for a faulty memory can be defined according to the technology node and operating conditions. Given this bit-error probability, the number of faulty bits are described by a binomial distribution. We then assumed i.i.d. faults across the memory bits and approximated the post-fabricated bit-error probability as $Pr \in \{0,1\}$ to derive the fault map for our simulation analysis.

To generalize the above fault model, we consider a memory of N_b bits and we define the data that is written to the memory as a bit vector $\boldsymbol{d} = [d_1, \cdots, d_{N_b}]$ with $d_i \in \{0,1\}$ and the

corresponding random variable $\boldsymbol{D}$. We also define the error-indicator of bit i $e_i \in \{0,1\}$, and the memory fault map $\boldsymbol{e} = [e_1, \cdots, e_{N_b}]$ with the corresponding random variable $\boldsymbol{E}$. The data in the memory is denoted by $\boldsymbol{d'} = [d'_1, \cdots, d'_{N_b}]$ with $d' \in \{0,1\}$ and random variable $\boldsymbol{D'}$. With these definitions, the error occurrence for each memory bit is described by

$$d'_i = d_i \bigoplus e_i, \tag{3.10}$$

where $\bigoplus$ is the binary XOR operation.

Based on the above definition, we now formally define the error model. We first focus on a specific die. To generalize our definition, we assume that the errors are data-dependent.[1] Hence, we need to condition the error on the data that is written to the memory. We therefore consider the conditional probability mas function (PMF) of error as

$$f_{E_i|\boldsymbol{D}}(e_i, \boldsymbol{d}) = \Pr(E_i = e_i | \boldsymbol{D} = \boldsymbol{d}). \tag{3.11}$$

Unfortunately, the above model is incredibly complex since the large dimension of $\boldsymbol{d}$ (N_b bits) makes the enumeration of all possible conditions even for a single error event on a single bit impossible. Also, some other dependencies, such as history of the memory, are not captured by the conditioning on $\boldsymbol{d}$. To avoid this issue and to circumvent our inability to model the exact dependency of the probability of errors on the entire data in the memory, we marginalize over the data, except for the data in the bit under consideration. With this simplification, we consider only $E_i|D_i$ and its PMF as

$$f_{E_i|D_i}(e_i, d_i) = \Pr(E_i = e_i | D_i = d_i). \tag{3.12}$$

Next, according to the non-ergodic behavior, we need to define $f_{E_i|D_i}(e_i, d_i)$ for each individual die with index $n \in \mathcal{N}$. We therefore consider $f^{(n)}_{E_i|D_i}(e_i, d_i)$ as the PMF of error for die n and suggest to empirically obtain this probability across all the dies through measurements. Under the assumption of no structural preferences for individual bit-cells, we can assume that the parameters of the error follow an i.i.d. distribution. We finally obtain the probability density function (PDF) of manufacturing fault model that leads to the fault map for each die as

$$g_{f_{E|D}}\big(f^{(n)}_{E|D}, e, d\big) = \sum_{i \in N_b} \Pr(f_{E_i|D_i} = f^{(n)}_{E_i|D_i}). \tag{3.13}$$

3.4.3.2 Optimization for Improved Resilience Against Error

We describe our proposition that allows to significantly mitigate the impact of faults on the decoder performance based on the above-described error model. We first consider the

[1] Note that we revisit and verify this assumption in Section 3.7.

impact of the errors on the magnitude of a signed random variable $x \in \mathcal{Z}$ with a B-bit binary representation $[x_B \cdots x_1]$. This binary representation can be achieved via different mappings. For the well-known 2's-complement (2C) and sign-magnitude (SM) mappings, x can be expressed as

$$x^{(2C)} = -x_B 2^{(B-1)} + \sum_{b=1}^{B-1} x_b 2^{(b-1)}, \tag{3.14}$$

and as

$$x^{(SM)} = (-1)^{x_B} + \sum_{b=1}^{B-1} x_b 2^{(b-1)}. \tag{3.15}$$

We consider that the sign-bit is protected, and for simplicity, we assume one error in a memory word.[2] If a word is affected by an error in bit-position ϵ (i.e., $e_\epsilon = 1$), the bit position x_ϵ becomes its binary complement $\bar{x}_\epsilon$. We not that the error magnitude depends on the binary data representation in the memory. For the mappings 2C and SM, since the bits are exponentially weighted, it is straightforward to show the corresponding error magnitude as

$$|x - x_\epsilon| = 2^{\epsilon-1}. \tag{3.16}$$

Since errors in the more significant bits have an exponentially-growing impact on the error magnitude, unequal error protection [120,121], or customized binary data representation [119] have been suggested to minimize the error magnitude.

Motivated by the above observation, and given the fact that the binary data representation in the memory can be customized, we are interested in minimizing the error magnitude. Our interest is however not the impact of a specific error event. Instead, we are interested in minimizing the mean error magnitude $\bar{E}$ across all manufactured dies, across all single-error locations ϵ, and across all possible logic-states of the bit in the error locations d_ϵ . This averaging can be expressed as follows

$$\bar{E} = \mathbb{E}_n\Big\{\mathbb{E}_\epsilon\Big\{\mathbb{E}_{d_\epsilon}\{|x - x_\epsilon|\}\Big\}\Big\}, \tag{3.17}$$

where $\mathbb{E}_z\{\cdot\}$ corresponds to taking the expectation over z. The outer-most expectation runs over all dies in the population. The middle expectation runs over all bits (except the protected sign bit) in a word and we slightly abuse our notation from the initial fault model by limiting the memory size to a single word and setting $i = \epsilon$. Finally, the inner-most expectation runs

[2] Note that multi-bit errors in a single word may occur, but are highly unlikely for low error probabilities.

over the data $d_\epsilon \in \{0,1\}$. Therefore, we can express (3.17) using (3.12) and (3.16) as

$$\bar{E} = \mathbb{E}_n\Big\{\mathbb{E}_\epsilon\Big\{2^{(\epsilon-1)} \cdot \big(\Pr(d_\epsilon = 0)\, f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 0) + \Pr(d_\epsilon = 1)\, f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 1)\big)\Big\}\Big\}. \tag{3.18}$$

The above explicit form for the mean error provides the core motivation for our proposed error-minimization approach. It can be seen that the mean error magnitude is strongly impacted by the relationship between the probability of the data polarities $\Pr(d_\epsilon = d)$ and their corresponding error probabilities $f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = d)$ for $d \in \{0,1\}$. While $\Pr(d_\epsilon = 0) + \Pr(d_\epsilon = 0) = 1$, we refer to $P_b = f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 0) + f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 1)$ as the average error probability across the data. It is commonly noticed that for error magnitude $\bar{E}$, if either the data is symmetric $\big(\Pr(d_\epsilon = 0) = \Pr(d_\epsilon = 0) = 1/2\big)$ or the error probability is symmetric $\big(f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 0) = f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = 1)\big)$, any asymmetry in the other is irrelevant. However, if neither of the two conditions above holds, the mean error magnitude can be influenced by adjusting the asymmetry in either the data or the errors.

Based on the above-described observation, we propose to reduce the impact of error and thereby improve the decoder performance as follows:

- encourage an asymmetric binary data representation, i.e., $\Pr(d_i = d) > \Pr(d_i = \bar{d})$, *and*
- skew the error pattern of the memory to be asymmetric in the opposite direction of data, such that $f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = d) < f^{(n)}_{E_\epsilon|D_\epsilon}(e_\epsilon = 1, d_\epsilon = \bar{d})$.

While the above strategy can be applied to individual bits, the choice of binary data representation dictates the data statistics for all the bits in the data word. Additionally, from (3.18) we note that the impact of errors on the mean error magnitude grows exponentially with the bit index in the word. It is therefore favorable in most cases to reduce on the error contribution of the most significant bits.

In order to apply the above, we first study the data distribution in R-memories of the decoder. We show the corresponding value statistics in Fig. 3.10 for different codewords and channel realizations. We observe that the data in R-memories is tightly distributed around zero and mainly covers the lower magnitude range while it is symmetric around zero. With the knowledge of the data distribution, we analyze the distribution of the bit values. We again consider the well-known 2C and SM mappings and note that for 2C there exist an inherent symmetry around zeros that results in a symmetric distribution of logic-1 and logic-0, i.e., $\Pr(d_i^{(2S)} = 0) = \Pr(d_i^{(2S)} = 1) = 1/2$, independent of the magnitude distribution. However for SM, the probability of logic-1 drops rapidly for the corresponding statistics in Fig. 3.10 when

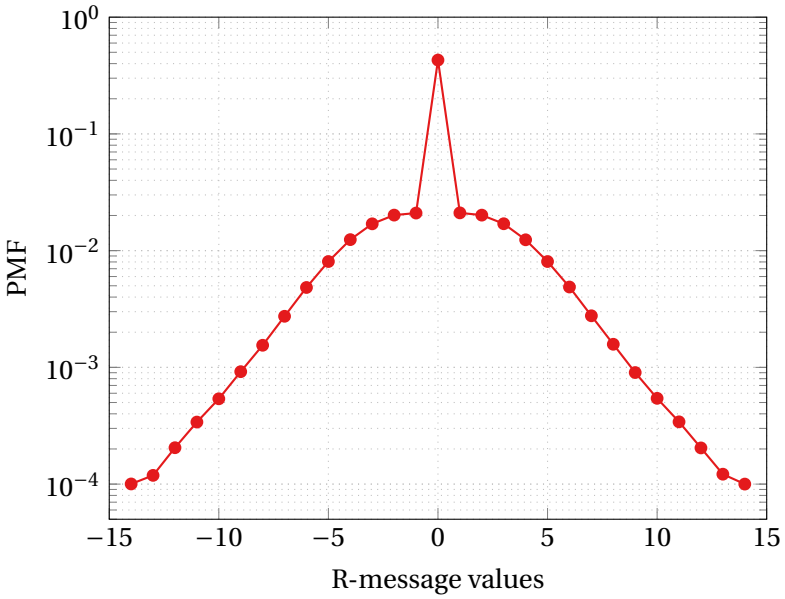


Figure 3.10: Probability of occurrence for R-message values averaged among multiple codewords and channel realizations.

the distribution of magnitude tightens toward zero, i.e., $\Pr(d_i^{(SM)} = 0) > \Pr(d_i^{(SM)} = 1)$ for i toward MSBs.

We illustrate the above observations by a small artificial example. Assume a random variable x which is represented by a 5-bit vector (bit-5 is the sign bit) with binary entries in 2C and SM format. If x is uniformly distributed between -15 and $+15$, all bits in both mappings have 50/50 chance for being 0 or 1. If x is uniformly distributed between -3 and $+3$, all bits in the 2C mapping are still 50/50 distributed across 0 and 1, however, in SM mapping, bits 3 and 4 are always 0.

With the above-provided insight in mind, and by exploiting the R-message distribution in Fig. 3.10, the choice of a SM binary data representation allows us to skew the distribution of the MSBs of the data toward logic-0. By skewing the error pattern of the memory toward logic-1 the mean error magnitude will be minimized, as can be observed by (3.18).

In order to analyze the effectiveness of this approach, we run simulations under the assumption of a skewed fault model while the data is stored in the memory with a SM representation. Fig. 3.11(a) shows the performance of the population of decoders by demonstrating the empirical CDF of FER while comparing two cases for the non-ergodic fault model with $P_b = 5 \times 10^{-4}$. The first plot colored in red, corresponds to a non-skewed fault model with error polarities of 1 and 0 (stuck-at-0 and 1) with equal probabilities. The second plot colored in blue, however, corresponds to a skewed fault model with an error probability of zero for logic-1 indicating only error polarities of 0 (stuck-at-0). It can be clearly observed that the asymmetric errors

provide a considerable FER gain. In fact, for the chosen 4 dB SNR operating point, most dies with symmetric faults and the chosen fault rate of 5×10^{-4} do not manage to decode any codeword correctly, i.e., FER≈ 1. However, by setting a target FER$< 10^{-1}$, we can observe from Fig. 3.11(b) that most of the die, i.e., $\approx 80\%$, with the asymmetric memory faults manage to decode most of the codewords at SNR> 3.75 dB.

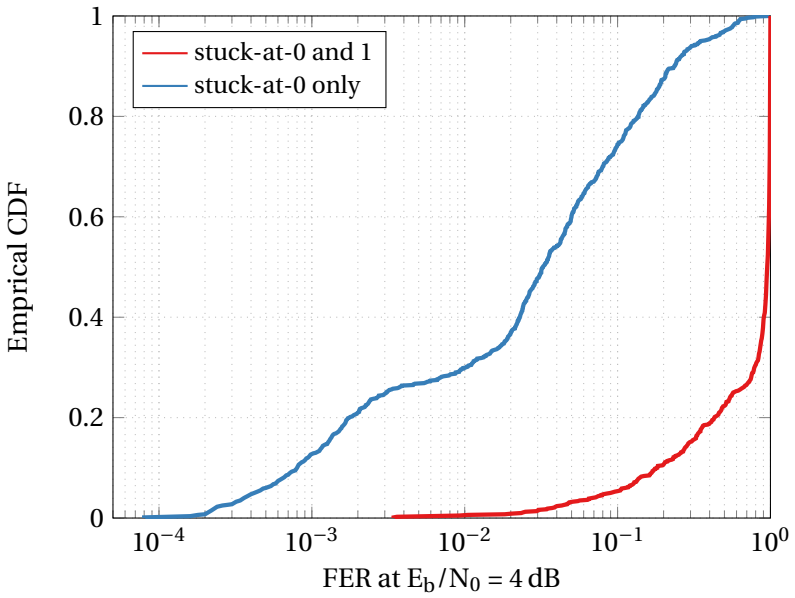
3.5 Integration to ErgoDEC Architecture

In this section, we discuss the integration of the improvement techniques to ErgoDEC architecture and we evaluate the potential overheads. We explain the address and bit-index randomization to enable the ergodic behavior and the imposed overhead, and we discuss the overhead of repeating the unsuccessful decodings. Finally, we explain how the skewed fault model can be implemented.

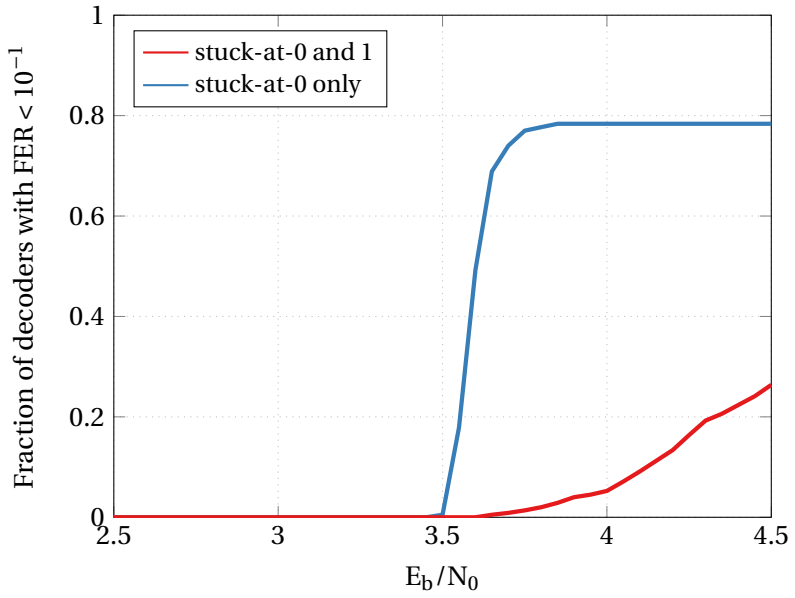
3.5.1 Address and Bit-Index Randomization

In order to realize the ergodic fault statistics in the decoder memories, error locations and polarities should alter over time. More specifically, memory address and data should be scrambled to create different logical memories with random faults over the course of decoder iterations or processing of the layers. In the previous chapter, we proposed to add randomization circuits to the memory of an embedded system. Our proposition for the decoder in the current chapter follows a similar basic idea, however, it is tailored according to the decoder architecture and the faulty memories dimension. More specifically, enabling an ideal randomization for the decoder memories with a wide memory bus imposes a non-negligible overhead as it requires to shuffle the bits across different LLRs in such a wide bus. The overhead unfortunately remains large relative to the memory size since the depth of the decoder memories is low. Therefore, we choose to implement the randomization circuit differently compared to the previous chapter as follows.

We propose to enable the above by integrating randomization circuits to the decoder memory macros at different granularities, i.e., bit-level, LLR-level (note that a memory word spans 111 LLRs each comprised of 6 bits), and address-level, as illustrated in Fig. 3.12. At bit-level, all the bits are XOR-ed with the same random bit to create an inversion in stuck-at errors. At LLR-level, a barrel-shifter is used to rotate the bit orders in an LLR according to a random number. The random number is generated/updated with a LUT-based random number generator, where a seed is used for the initialization. Despite the simplicity of the LUT-based random number generator, it has strong theoretical properties, such as universality and high independence degree, as compared to other random number generation methods [122]. Further, a similar configuration for all the shifters in each memory word is applied and no word-level randomization across the LLRs in a memory word is implemented to reduce the



(a) FER-Yiled



(b) SNR-Yiled

Figure 3.11: Performance evaluation of a population of faulty LDPC decoders through a) empirical cumulative density function of FER at a fixed $E_b/N_0 = 4$ dB, and b) fraction of decoders with FER$< 10^{-1}$, for non-ergodic fault model with bit-error probability $P_b = 5 \times 10^{-4}$ and error polarities of 1 and 0 vs. error polarity of 0 only.

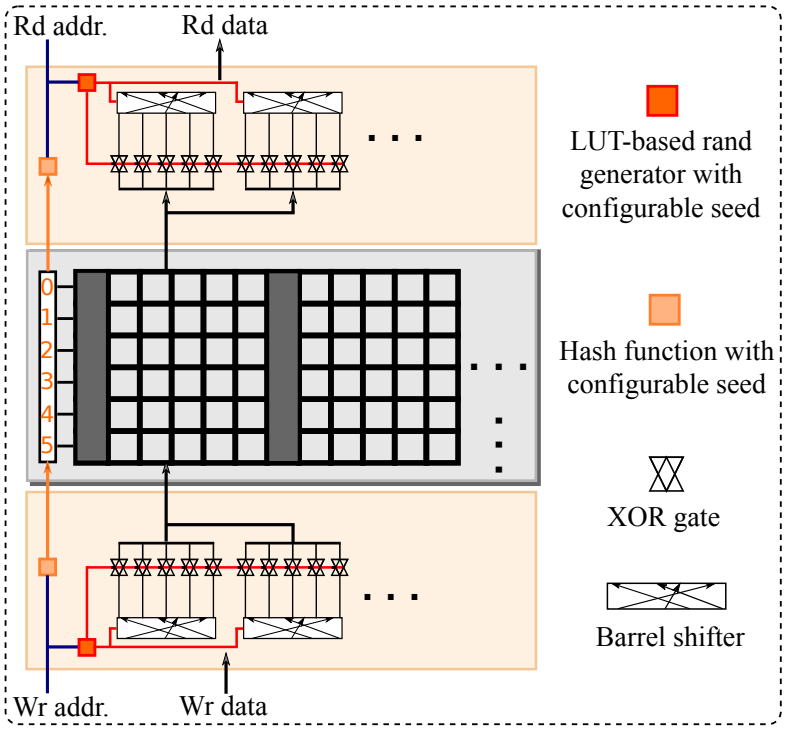


Figure 3.12: The faulty memory macro with randomization logic that is used to create an ergodic fault process.

overhead. At address-level, the memory address is scrambled by a hash function that is also initialized with a seed. All the operations are applied during write and the reverse operations is applied during read to recover the original data.

A new seed is applied for each codeword and is updated during each decoding iteration. The random number engine used for configuring the shifters and XOR gates receives a key from concatenation of the seed and the memory address. Beside ensuring the generation of a different random number for each memory rows and thus enabling a more ergodic fault behavior, this measure provides a natural means to revert the write randomization operation during read without a need for an additional storage element to preserve random numbers during the data lifetime of the datapath memories. We note that as opposed to the random number generator, the seed of the address scrambler hash function remains unchanged during the entire decoding due to the fact that R-messages are updated over the course of iterations and thus the memory address should remain unchanged to avoid any data loss due to overwriting of valid messages.

3.5.2 Repeating Unsuccessful Decoding Attempts

Unsuccessful decoding attempts can trivially be recognized by monitoring the codeword syndrome, which naturally gets computed after each decoding iteration. Note that for the

decoders with layered decoding algorithm, such as the decoder of this chapter, a partial syndrome is computed [123], which can similarly be used for this purpose. Therefore, no extra circuitry is required for detecting the unsuccessful attempts. However, the decoder FSM needs to be modified to repeat the decoding for such codewords while configuring the above-described randomization circuit differently, i.e., with a different seed. Overall, implementation of repeating the decoding attempt does not impose an overhead in hardware.

3.5.3 Optimizing the Binary Data Representation in the Memory and the Memory Faults

Recall that the binary data representation and the memory faults need to be jointly considered to achieve a minimized impact for the memory faults. To this end, we use a SM binary data representation for the messages in decoder allowing to skew the MSB distribution toward logic-0 together with designing a bit-cell that has a lower probability of logic-0 failure than logic-1. We recall the bit-cell schematic in Fig. 3.3 and note that the employed bit-cell features a skewed reliability. In fact, in this schematic logic-0 bits enjoy an infinite retention time as opposed to logic-1 values as the NMOS transistors are typically more leaky than the PMOS and the SN charge degrades toward ground, which skews the fault toward stuck-at-0. Even though, this behavior appears naturally, it can be reverted by artificially inverting all the information bits during write and inverting them back during read in case of a need for a skewed fault behavior toward logic-1.

3.6 Test Chip Architecure and Physical Implementation

In this section, we explain the fabricated test chip architecure and the infrastructure that is added to enable monitoring the faults in the memories. We also discuss the physical implementation of the architecture given the large number and the special dimension of the memories in the design.

3.6.1 Chip-Level Architecture and Operation Modes

An overview of ErgoDEC chip-level architecture is provided in Fig. 3.13. The architecture consist of the decoder core, interface memories for the decoder, a test controller, and a serial interface to access the memories externally, which are explained in details in the following.

The decoder main building blocks are the Q-, T-, R-memories, and the decoder logic, as previously explained. The key feature of ErgoDEC is its ability to closely monitor and analyze the type and the impact of errors in the memory. To this end, T- and R-memories are wrapped into test structures, containing debug memories and logic, that aim at recording the faulty memory behavior during the decoding process. The purpose of this (costly and area consum-

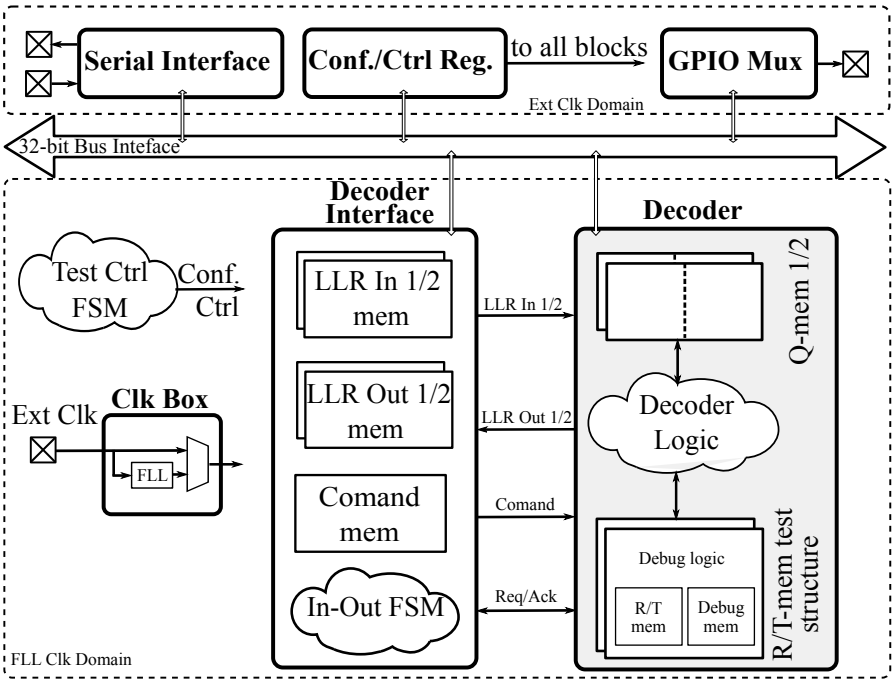


Figure 3.13: ErgoDEC chip-level architecture overview.

ing) measure is to be able to identify errors in the faulty memories during the decoding. To this end, in parallel to the dynamic SCM, during the write operations, the data is loaded into a shadow memory that is built from a SRAM and stores a reliable copy of the data, as shown in Fig. 3.14. During read operations, both the dynamic SCM and the reliable copy are read and compared while the difference is logged in an additional SRAM (difference memory) to obtain a fault map. In addition to storing a reliable copy of the data to track the fault, the shadow memory provides a fully reliable decoder operation even for low frequencies since it can be used as a reliable static datapath memory instead of the dynamic SCM.

The interface memories are comprised of two buffers for each of the input and output LLRs to store two codewords, which allow the decoder to ping-pong between two codewords in a configurable loop. Additionally, the decoder core integrates two pairs of Q-memories for continuous operation with two codewords. Once the LLR buffers are pre-loaded with channel LLRs, the decoder starts by loading its first internal pair of Q-memories. After this initial loading process, the decoding is started. During the decoding process, the second pair of Q-memories can be loaded from the interface buffer. Once the decoding of the first codeword is complete, the decoder starts to decode the second codeword and it dumps the results of the first codeword to the buffer memory and loads again the pair of Q-memories with the channel LLRs of the first codeword. Therefore, the integrated test harness around the LDPC decoder core enables continuous operation with two codewords, which is suitable for an average power measurement. It also allows a single codeword decoding by loading only one of the interface buffers and configuring the decoder for the decoding of one codeword. To

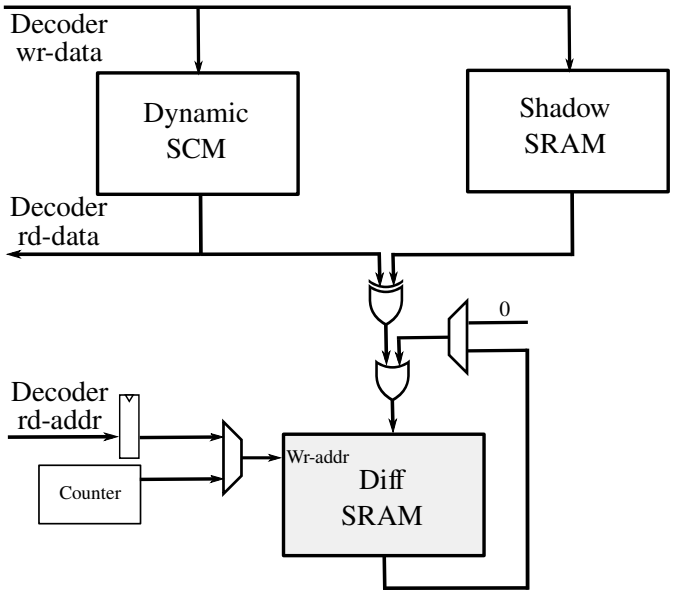


Figure 3.14: Test structure of the T- and R-memory with debug (shadow and difference) memories and logic to enable multiple operating modes.

perform FER measurements with a larger number of different codewords, ErgoDEC interface buffers need to be loaded multiple times with fresh codewords and the results need to be checked externally by a test set-up.

A serial interface provides access to all the storage elements of ErgoDEC, i.e., test structures and interface memories as well as configuration registers, as in Fig. 3.13. While this serial interface requires only few pins, it is also slow and therefore data can neither be provided nor be checked in real-time from outside the chip. Instead, it is used to load the stimuli and the configuration into the corresponding storage elements, trigger the decoder, and read out the result. It is worth noting that parallel-to-serial and serial-to-parallel shift registers are integrated to enable reading from/writing to the memory macros with the wide 666 word length.

ErgoDEC provides multiple operating modes. While a free running mode over repeated codewords is used to measure an average power, multiple runs of the decoder over different codewords is used to measure the FER or memory fault maps by reading the corresponding memories. Further, the test structure around T- and R-memories can be used to record faults in any phase of the decoding process or can log aggregated fault maps over the entire decoding of a codeword (spacial mode). In addition to this mode, a history of a specific address in the faulty memories can be logged, which provides information on the evolution of faults throughout the decoding process (temporal mode).

Table 3.2: List of memories and their sizes in ErgoDEC for the implemented QC-LDPC code with $Z = 111$ and message quantization bit of 6.

Memory	Type	Quantity	Depth×Width
T	SCM	2	8 × 666
R	SCM	2	24 × 666
T shadow	SRAM	2	8 × 666
T diff	SRAM	2	30 × 666
R shadow	SRAM	2	24 × 666
R diff	SRAM	2	24 × 666
Q	SRAM	4	8 × 666
Interface	SRAM	4	16 × 666

3.6.2 Physical Implementation

The physical implementation of ErgoDEC requires special scrutiny since there exist a large number of memory elements with a very wide dimensions, which imposes a large number of global wires and thus a complex routing. To better explain the issue, we list all the memories in ErgoDEC along with their dimension and type in Table 3.2[3]. The SRAMs are based on the standard foundry macros. In order to build the required dimensions, we use macros with dimension of 16 × 222 and 32 × 222 and we stacked 3 macros to build the wordlenght of 666. The SCMs are compiled based on the proposed hybrid static and GC latches [65], and therefore, the SCM macro dimension can be customized according to the required wordlength.

The above large number of physical SRAM macros and their connection to the SCMs and the decoder logic provides multiple floorplan configuration options for the physical implementation. For example, the physical macros corresponding to a memory can be stacked horizontally or vertically to facilitate the local (between the memory and the corresponding logic) or the global (between multiple memories and across the chip) routing, respectively. Additionally, the position of each memory in the chip can be optimized according to its connection to other memories and the decoder logic.

Fig. 3.15 illustrates the proposed physical floorplan for ErgoDEC. In this floorplan, SRAM macros are stacked vertically to reduce the global routing congestion. Also, Q-memories are placed on one side of the decoder logic, while T- and R-memories are placed on the other side to provide an easier access for all the memories to the decoder logic. The interface memories are placed next to the Q-memories and the R- and T- debug memories (shadow and diff) are placed next to the faulty T- and R-memories, which altogether, eliminate unnecessary routings over the decoder core.

[3]Two more memories are used in ErgoDEC to store the decoder commands and decoder sequences, which are not listed in the table since their dimensions are negligible as compared to other memories.

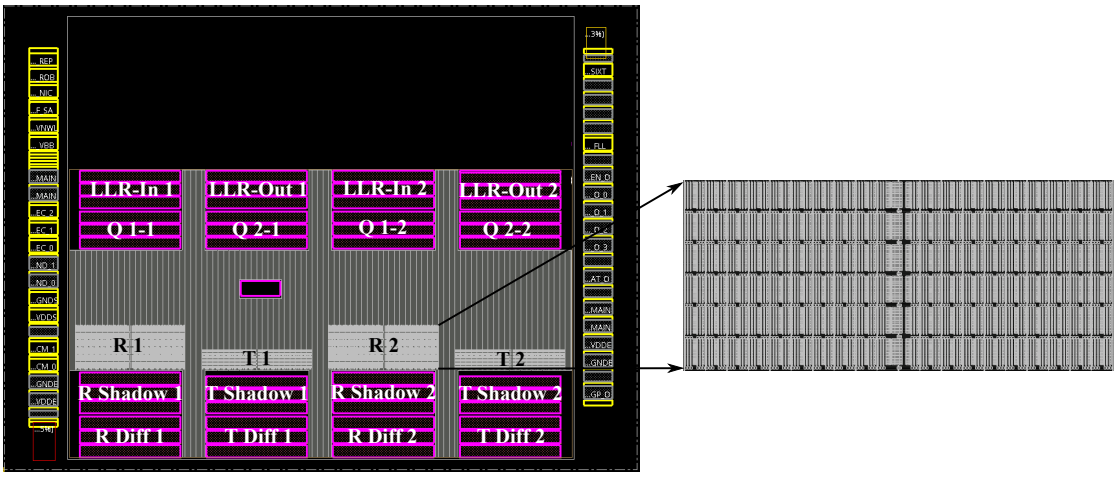


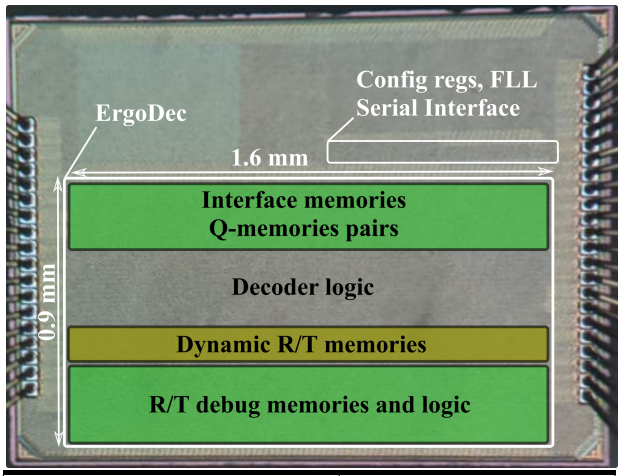
Figure 3.15: The proposed floorplan for ErgoDEC which highlights the SRAMs (annotated with white color) and SCMs (annotated with black color) macro locations (left), and the controlled placed SCMs (right).

The SCMs are composed of the proprietary dynamic latch and standard cells (including a static latch for the sign bits). Therefore, the SCM array can be implemented as part of the standard digital placement and routing flow. However, the memory arrays have a regular structure in its address decoder, clock tree and storage nodes. We therefore, follow the controlled placement methodology of [65] to optimize the physical implementation, as shown in Fig. 3.15 (right). Also, we vertically stack 6 arrays with worldlength of 114 bits[4] to build the required worldlength of 666 bits. This choice is made to match the width of each SCM array with the corresponding debug SRAM macros, as it can be seen in Fig. 3.15. As a result, each bit from the SCM array data port more easily reaches the corresponding index in the SRAM data port with a vertical routing and unnecessary horizontal routing across the SRAM macros and across the SCM arrays is avoided.

3.7 Measurement Results

The ErgoDEC architecture, described in Section 3.6, was fabricated in a 28 nm FD-SOI regular-V_T CMOS technology, utilizing 1.44 mm^2 of a complete 3 mm^2 die. The micrograph and main features of the chip are shown in Fig. 3.16. In addition to the area, we report the reliable frequency ranges and the corresponding power consumptions for two supply voltages. The minimum reliable frequency is the lowest frequency with no memory error (no DRT violation) and the maximum reliable frequency is the highest frequency that the decoder can achieve without setup-timing violation.

[4] Each memory array contains 19 LLR words with 6 bit each. The last macro thereby contains 3 redundant words to keep all macros of the same width, while avoiding breaking data words across macro boundaries for regularity.



Technology	28 FD-SOI	
Die area [mm^2]	3	
Decoder core area [mm^2]	1.44	
Vdd [V]	0.9	0.7
Reliable freq. range [MHz]	57 – 190	10 – 52
Power range [mW]	32.8 – 78.8	4.2 – 18.3

Figure 3.16: ErgoDEC chip micrograph and main features.

A measurement set-up is developed for ErgoDEC that is comprised of a XILINX FPGA on an evaluation board, which communicates with ErgoDEC through the serial interface and with the measurement computer through an RS232 interface. ErgoDEC is placed on a separate board, which contains the power supply, level shifters, and a phase-locked loop (PLL) to adjust the interface clock. The operation clock is generated on-chip using an embedded frequency-locked loop (FLL). This FLL provides the necessary flexibility to adjust the clock in small steps to explore the reliability (i.e., retention time limit) of the embedded memories. Fig. 3.17 shows a picture of the measurement set-up.

The test programs are fully automated and have been developed based on a proprietary IC test and characterization application programming interface (API). Each test program reads the measurement stimuli and configuration data for different test scenarios from the computer and writes them to the chip through the serial interface using the FPGA. The program also reads the output results from the chip back to the computer. In the following, we report the measured results. Our aim is to verify the underlying assumptions that are made for the fault map and validate the proposed performance improvement schemes. To this end, we first study the fault map and then measure the FER of the decoder.

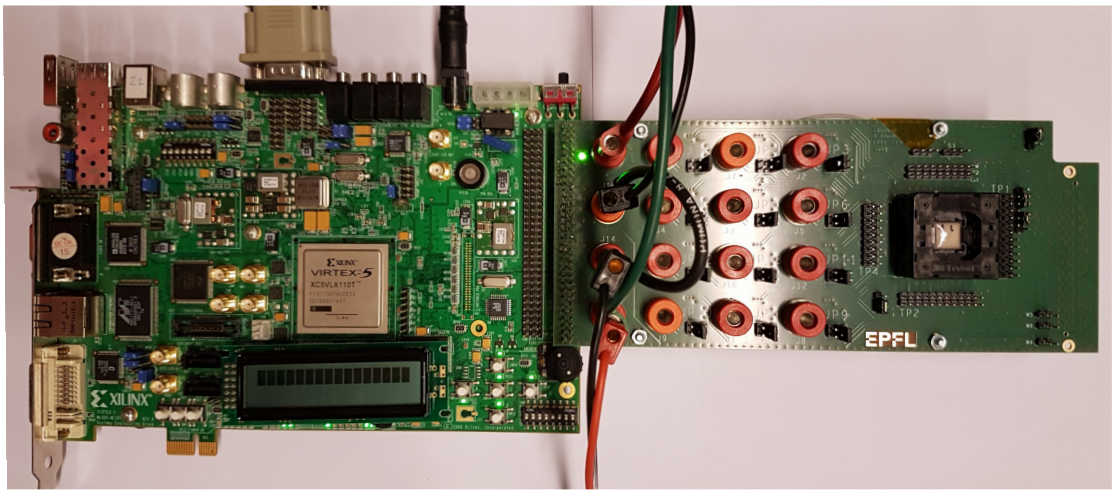


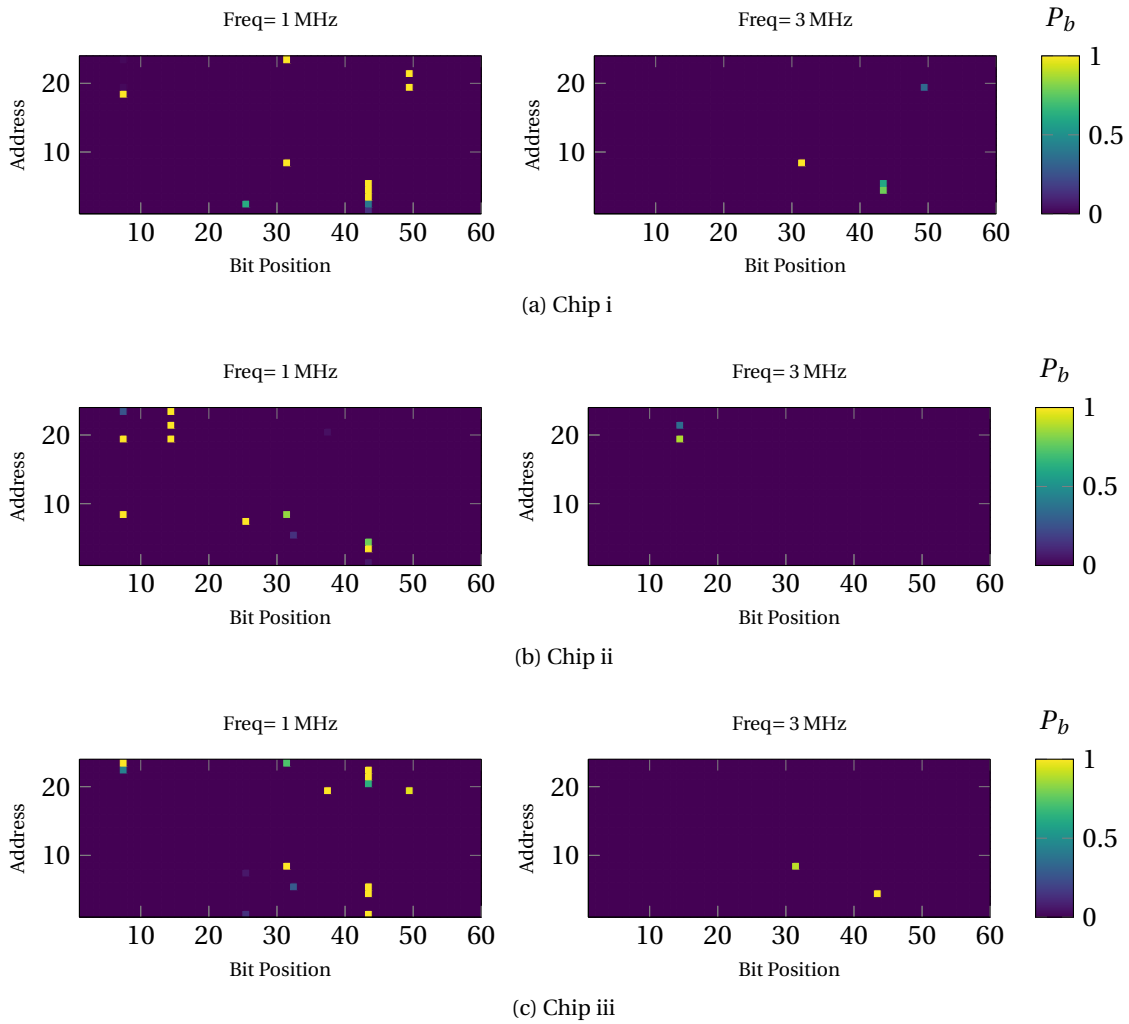
Figure 3.17: ErgoDEC measurement set-up, comprised of a custom validation PCB on the right-hand side hosting the ErgoDEC IC, connected to a Xilinx XUPV5-LX110T FPGA board on the left-hand side.

3.7.1 Fault Model

A key hypothesis underlying the fault model of this chapter is that the majority of errors in memories are caused by process variations. Under this assumption, specific error patterns are imprinted on each individual die during production. These error patterns are different for different dies and remain mostly unchanged during operation, which correspond to a non-ergodic fault behavior.

To verify the above behavior, we provide measurements that focus only on extracting the fault map of ErgoDEC dynamic SCMs using the test structure around the T- and R- memories. To this end, we use the spacial mode of ErgoDEC and read and compare the faulty memory contents against the reliable shadow memory after the first iteration of the decoder to avoid accumulation of errors over multiple iterations.

We obtain the memories fault map by re-running a measurement for an example die with a single test pattern and configuration, i.e., codeword and frequency, and we average over 50 repetitions. Fig. 3.18(a) show the R-memory fault map at one SNR while only 60 bits out of 666 word width are illustrated for better visibility. The brightest color in this figure implies that the cell fails all the time (Pr=1), while the darkest color implies that the cell never fails (Pr=0). The figure shows a clear dependency between the frequency and number of errors, as was intended and explained, in which the fault ratio decreases as the frequency increases since the data lifetime in memory reduces and fewer bit-cells show a faulty behavior. Additionally, we observe that the majority of the failing bits have a failure probability close to one. This observation verifies our assumption for the simplified error model of Section 3.3.2, where the probability of error was approximated to be in $\{0, 1\}$. Further, it confirms stability of the errors



(a) Chip i

(b) Chip ii

(c) Chip iii

Figure 3.18: Fault map of the R-memory for three chips at SNR of 3.7 dB and frequencies of 1 MHz and 3 MHz.

during multiple runs and thus proves the deterministic nature of the error patterns imprinted by the manufacturing.

Following the above observation, we re-run the explained fault map test for multiple dies and illustrate the result for three dies in Fig. 3.18 for comparison. We clearly observe the difference in the faulty bits between these dies, which confirms that errors are different for different dies, as various outcomes of the manufacturing process. In fact, as already explained, the dies (may) show different computation outcomes in the end of the decoding, and therefore, the commonly-assumed performance evaluation based on the chip-ensemble average is meaningless.

In addition to the above, we claimed the asymmetric behavior of the GC latch in Section 3.5, which was exploited to minimize the fault impact. This special design creates a data-dependent

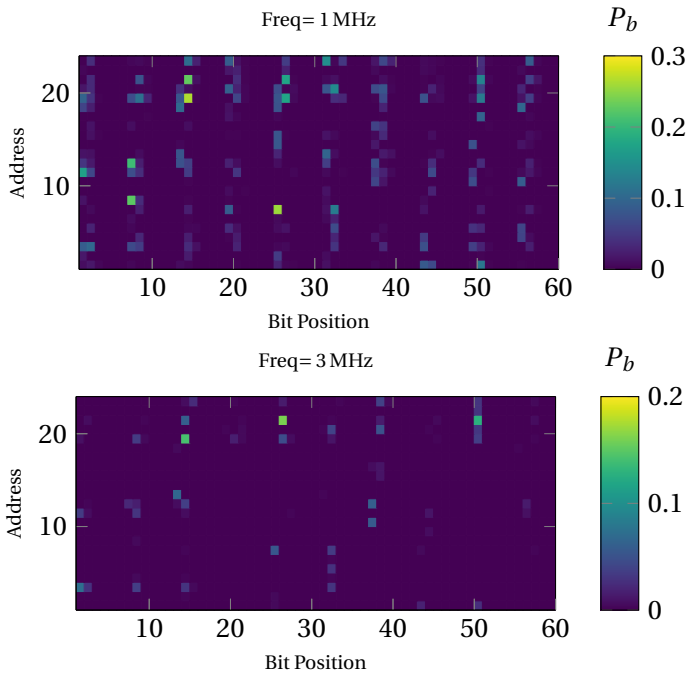


Figure 3.19: Fault map of the R-memory averaged over 500 codewords at SNR of 3.7 dB and frequencies of 1 MHz and 3 MHz.

behavior for memory faults since only logic-1 fails. In order to verify this data-dependent behavior, we analyze the memory fault map for multiple codewords and compare it with the above-described fault map. To this end, we run a measurement for 500 codewords at one SNR and different frequencies and illustrate the R-memory fault map by averaging over all the measurement results in Fig. 3.19. We observe that the probability of failure is not close to one anymore and is significantly lower than that of Fig. 3.18, while more errors are distributed around the memory bits. The higher number of errors with lower probability implies that the faults are data-dependent and different bit-cells fail during the decoding of different codewords.

3.7.2 Decoder Performance

A major claim that motivates the study of this chapter is that the ergodic fault model does not reflect the reality of the manufactured dies and indeed there is a distinction between the quality of different dies. To confirm the non-ergodic assumption, we need to consider the ultimate decoder performance, i.e., the FER, as it was shown in the simulation results in Section 3.4. To this end, we measure the FER of the decoder chips for 17 different dies from two (typical and slow) fabricated wafers to better extract the statistics. In order to have comparable results among all the dies, we first calibrate the operating frequency such that each test die yields the same number of errors (same P_b) in their memories, while the difference between dies only lie in the fault locations and characteristics. The calibrated frequencies for the dies is

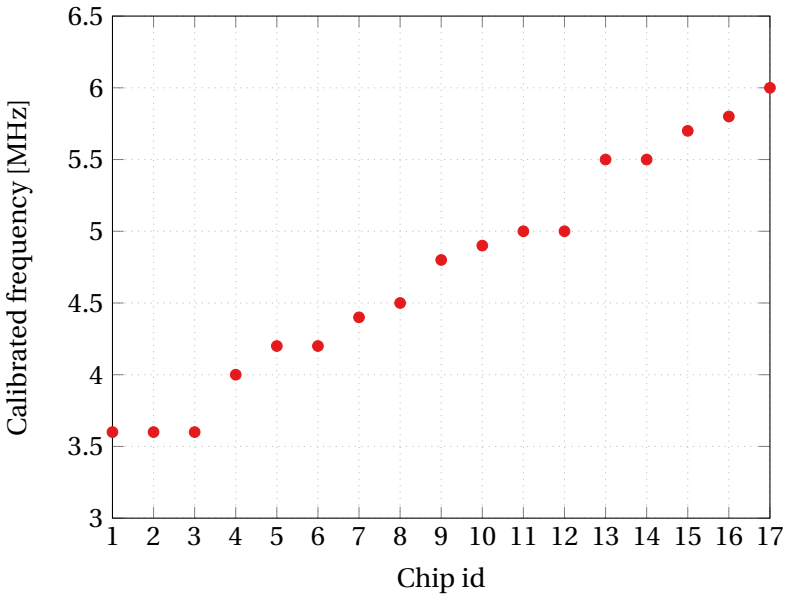
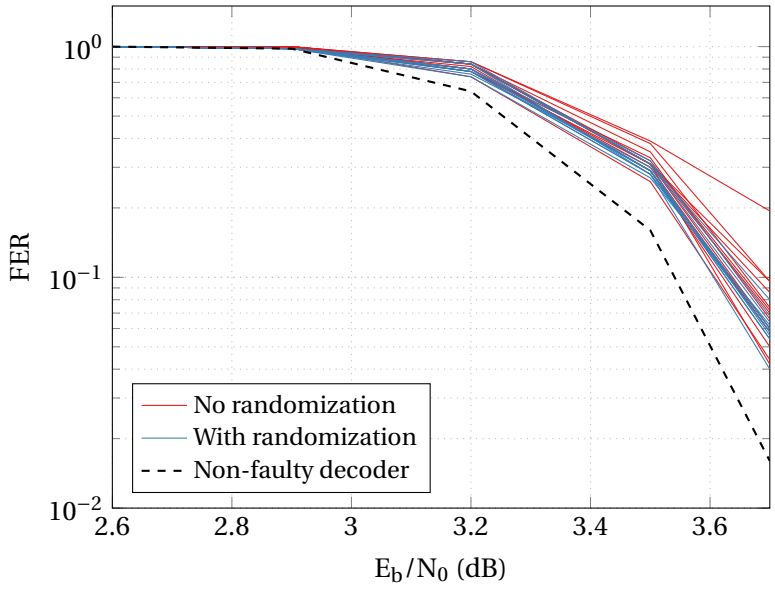


Figure 3.20: The calibrated frequencies for the 17 tested dies for R-memory fault probability of $P_b \approx 5 \times 10^{-4}$.

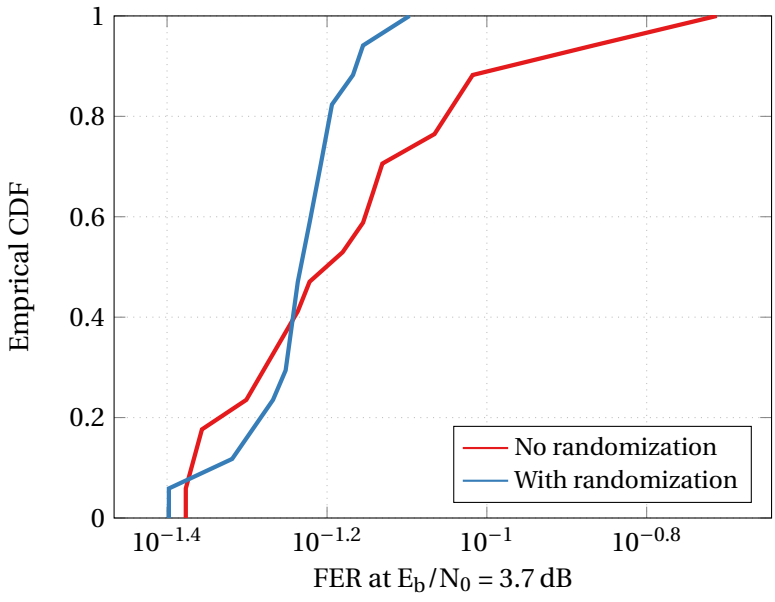
sorted and illustrated in Fig. 3.20. We then measure the FER by reading the decoded codewords from the output LLR interface buffer and compare them against the expected (reference) result for different codewords.

We have also proposed to randomize the memory errors to restore the pre-manufacturing ergodic behavior across the population of manufactured dies. To show the improvement made by the proposed randomization technique, we run the FER measurement with two different configurations. The first configuration relates to the normal operation of the decoder, while the second one corresponds to the case where the randomization circuits are enabled. To this end, the data is XOR-ed with a random number, the LLR bits are shuffled, and the address is scrambled for the T-memories as in Fig. 3.12, however, XOR gates are disabled for the R-memories to benefit from the skewed fault pattern and the strong logic-0 bias of the data in the R-memories.

Fig. 3.21 shows the FER measurement results vs. SNR and the empirical CDF for the dies at a fixed SNR of 3.7 dB for a fault ratio of $P_b = 2.5 \times 10^{-4}$ ($P_b = 5 \times 10^{-4}$) in T-memory (R-memory). The red curves pertain to the normal operation mode of the decoder and the dashed black curve corresponds to the non-faulty decoder from the simulation model. We can see clearly in this figure how the FER performance across SNR is different for different dies, despite the calibration for the same number of errors. We also see the spreed among the dies in the CDF of FER at one SNR. This observation proves the non-ergodic behavior of the quality and thus the decoder performance across the population of decoder dies, as predicted in the simulation results provided in Section 3.3



(a) FER vs. E_b/N_0



(b) FER-Yiled

Figure 3.21: Measured frame error rate results of 17 faulty LDPC decoder chips with R-memory (T-memory) fault probability of $P_b \approx 5\times10^{-4}$ ($P_b \approx 2.5\times10^{-4}$) with and without randomization.

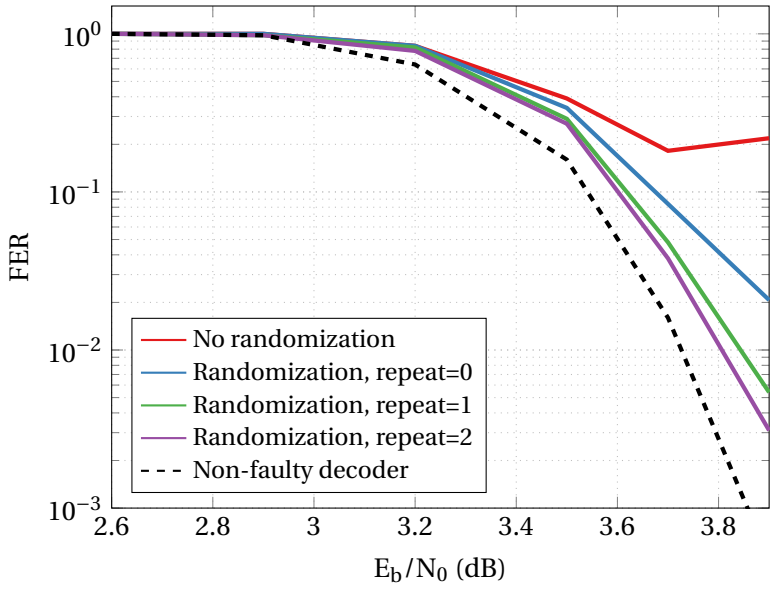


Figure 3.22: Measured frame error rate results of a faulty LDPC decoder chip with R-memory (T-memory) fault probability of $P_b \approx 5 \times 10^{-4}$ ($P_b \approx 2.5 \times 10^{-4}$) while the unsuccessful decoding are repeated 1 or 2 times.

The blue curves in this figure pertain to the case where the randomization circuits are enabled while each die is running at the same calibrated frequency similar to the above. As we can clearly observe, the quality spread is reduced by employing our randomization technique.[5] This smaller quality variance among the dies indicates that the quality becomes ergodic. Therefore, the time-average performance of each die approximates the ensemble-average of the population of dies. This stabilization now enables a confident and extremely easier testing procedure for a minimum quality, as proposed in the previous chapter and further explored in Section 3.4. To this end, dies need only be sorted by the fault ratio in the memory and a minimum quality for all the dies in each group can be guaranteed as they now have a similar time-average quality.

Along with the randomization technique, we have also proposed to repeat the decoding for the unsuccessful codewords with a different fault realization, which showed a significant performance improvement in simulations. To verify this proposition, we run a measurement on the measured die that showed the worst error rate performance without randomization among all the measured dies. We allow up to 2 more decoding repetitions for the unsuccessful codewords, while in each repetition we initialize the random number generator and the hash function with a different seed (see Fig. 3.12) to ensure an independent behavior for the logical memory faults. During the post processing, we consider the codeword as correctly decoded if any of the corresponding decoding attempts were successful. We show the FER vs. SNR for this example chip in Fig. 3.22. As we can observe, the FER improves as we enable

[5] We note that the small divergence in the tail of the blue curve pertain to the fact that the frequency calibration across the dies cannot be made such that all the dies have the exact same probability of error in their memories.

repeating the decoding of unsuccessful codewords compared to the case with no repetition and to the case without randomization. This improvement is significant specially for the curve with 2 extra repetitions, colored in pink, such that the performance of the faulty decoder approaches that of the non-faulty decoder, colored in black. This observation proves the efficacy of our proposition in Section 3.4, and shows a methodology to improve the faulty decoder performance at a negligible overhead while this improvement can only be enabled with the proposed randomization scheme.

3.8 Conclusion

We proposed an approximate ergodic LDPC decoder in a 28 nm FD-SOI technology, ErgoDEC. We showed with measurement that the memory faults as well as the quality across a population of dies are non-ergodic, and therefore, the fault model commonly-assumed in the previous literature is not correct. Beside verifying of the non-ergodic fault model and quality distribution, we proposed and proved with measurement novel approaches to improve the quality of faulty dies by equalizing the quality across the dies and minimizing the impact of memory faults. Altogether, ErgoDEC is the first measured example of an integrated circuit that delivers stable performance across a population of dies despite the presence of errors in its memories. As such, it shows that approximate computing is feasible without a complex test procedure and acceptable quality.

4 DVFS Based Power Managment for LDPC Decoders with Early Termination

While the previous chapters have focused on the issue of unknown, but static (over time) variations, we now focus on uncertainties that do vary naturally over the lifetime of a circuit. Data-dependent variation is such a type of uncertainty, which appears on a fast time-scale at run-time and can affect the output quality especially in iterative (data-dependent) algorithms. Traditionally, even iterative algorithms and corresponding architectures are designed to operate reliably or achieve the minimum required quality for the worst-case scenario of the input data. For instance, the critical path delay is naturally evaluated for the worst-case input transition, or iterative algorithms are often configured for maximum iterations to achieve the required quality for the worst-case scenario. However, such worst-case conditions are usually rare, and hence, such an approach leads to over-design by imposing a conservative design-time margin, which is not required for most of the cases at run-time.

To avoid the costly margin and to design extremely efficient circuits and architectures, which can gain from data-dependent variations, it has been proposed to exploit the flexibility and robustness of some algorithms through dynamic approximate computing [124, 125]. Specifically, iterative algorithms are a great candidate since the algorithm iterations provide an effective means at run-time to adapt to fast data-dependent variations trading the output quality for computational efforts.

As already mentioned in previous chapters, wireless communication is a domain that can benefit from dynamic approximate computing since it contains various iterative and probabilistic algorithms and the system has inherent robustness against the potential rare worst-case events. Among such algorithms, decoders for error-correcting codes are very promising examples since they process stochastic signals that are often highly distorted by noise and/or interference. Further, for such input signals, there is often an acceptable output range thus indicating a high degree of flexibility on the output quality. Among the most well-known cod-

ing schemes, low-density parity check (LDPC) codes have been widely used in high data-rate communication standards [71]. Throughout this chapter, we therefore continue to focus on this error-correcting code. As opposed to the previous chapters, where we have randomized static errors to improve yield, we now show how dynamic approximate computing techniques are used to effectively trade the error-correcting performance for computational complexity by exploiting run-time variations due to the random input data. To this end, we dive slightly deeper into the algorithm and exploit algorithm/architecture interactions.

LDPC codes are commonly decoded using iterative message passing (MP) algorithms through a message exchange process between decoder computation nodes, as already explained. This iterative decoding process for LDPC codes naturally provides an effective means to adjust the performance-complexity trade-off. In practice, the decoder is configured for a maximum number of iterations, $I_{\max}$, which is chosen to achieve the required error-correcting performance in a worst-case signal-to-noise ratio (SNR) scenario. However, the initial iterations will be more significant in decoding the received signals and the following iterations are only useful for recovering the data in low SNR regimes. Thus, early termination (ET) was proposed [78] to avoid redundant iterations by stopping the decoder when extra decoding effort is not required. With this measure, the decoder energy has a natural *first-order data-dependent* behavior since the decoding energy per bit decreases linearly with the actual required number of iterations for each code-block at run-time.

While avoiding redundant iterations with ET effectively reduces the decoding effort per code-block and adapts the computational effort to run-time variations, this measure does not reduce the energy consumed per iteration. However, when fewer iterations are required, more time may be allocated to each iteration without affecting the throughput. *dynamic voltage and frequency scaling (DVFS)* [126, 127] provides an effective means to exploit this additional time to also reduce the energy for each individual iteration. To this end, both frequency and supply voltage are adjusted accordingly to still meet timing requirements while taking advantage of the quadratic dependency of the power on the supply voltage. Together with ET, this measure then results in a *second-order data-dependent* energy behavior, which further improves the energy-efficiency of the decoder at high SNR.

Unfortunately, to apply DVFS to the decoding process, the number of required iterations (RIs) must be *predicted* before or at least during the decoding process of a code-block to properly reduce both voltage and frequency to match the actually required decoding effort without missing the deadline for the decoding of that block. This iteration prediction can be categorized as either *static* or *dynamic*, where static prediction is performed only once during initialization of the decoding process while dynamic predictions are updated after each iteration. The objective is thereby to keep the number of predicted iterations as low as possible to maximize the gain from run-time variations, while avoiding an underestimation of the number of RIs, which would lead to an unsuccessfully decoded data frame and decoder

performance degradations. Different algorithms in both approaches have been proposed to enable the use of DVFS. The authors of [128, 129] proposed a static iteration prediction and the dynamic iteration prediction was later addressed in [130, 131]. Unfortunately, none of these prior works provides a fully systematic approach to jointly analyze and adjust the trade-off between power savings from DVFS with more aggressive prediction strategies on one side and a potential error-rate performance degradation on the other side.

Contributions and Outline Given the importance of the correct trade-off analysis between the potential energy saving and the output error-correcting performance degradation, we propose a systematic statistical framework for reducing the energy consumption of an LDPC decoder with DVFS based ET and characterize the maximum theoretically-achievable saving. We then propose an algorithm, within this framework, to dynamically predict the RI for each codeword that benefits from the performance/complexity trade-off. This algorithm exploits the data-dependent variations at run-time through approximate computing and thereby allows to slightly relax the error-correcting performance by a well-defined margin in order to achieve the desired energy-efficiency improvements.

The remainder of this chapter is organized as follows. Section 4.1 gives a brief introduction to LDPC codes, as well as more details on the decoder power reduction through iteration management. Section 4.2 describes our proposed statistical framework for the energy consumption of an LDPC decoder with iteration prediction. In Section 4.3, our algorithm to minimize the decoder energy consumption is presented and its performance is evaluated in simulation results, and finally, Section 4.4 concludes the chapter.

4.1 Background

A binary LDPC code is a set of codewords which are defined through an $M \times N$ binary-valued sparse parity check matrix as

$$\left\{\mathbf{c} \in \{0,1\}^N | \boldsymbol{H}\mathbf{c} = \mathbf{0}\right\}, \tag{4.1}$$

where all operations are performed modulo 2. If the parity check matrix contains exactly d_v ones per column and exactly d_c ones per row, the code is called a (d_v, d_c)-regular LDPC code. LDPC codes are usually represented with a *Tanner graph* which contains N variable nodes (VNs) and M check nodes (CNs) and VN n is connected to CN m if and only if $\boldsymbol{H}_{mn} = 1$.

As already discussed in the introductory part of this chapter and also previously in the thesis, LDPC codes can be decoded using MP algorithms, where information is exchanged as messages between the VNs and the CNs over the course of several decoding iterations. With early

termination (ET), messages are exchanged until a valid codeword $\hat{\mathbf{c}}$ is found for which the *syndrome* $\mathbf{s} = \boldsymbol{H}\hat{\mathbf{c}} = \mathbf{0}$ or until the maximum number of iterations $I_{\max}$ has been reached.

4.1.1 LDPC Decoder Energy Reduction with ET and DVFS

The total energy expenditure of an LDPC decoder with I decoding iterations per codeword is defined by

$$E(I) = \sum_{i=1}^{I} P_i T_{\text{iter}}, \tag{4.2}$$

where P_i is the decoder power consumption during the i-th iteration and T_{iter} is the time required for this iteration. We note that for high throughput LDPC decoders P_i can be approximated by the switching power of a CMOS circuit as $P_i = \alpha_i C {V_{dd}}^2 f_{\text{CLK}}$, where α_i is the expected value of the decoder-circuit activity factor during the i-th iteration and C is the total capacitance of the switching decoder-circuit elements, V_{dd} is the supply Voltage, and f_{CLK} is the clock frequency [58, 132]. Interestingly, according to our experimental analysis, the activity factor α_i turns out to be independent of the decoder iteration. Furthermore, in a typical system, decoding is performed within a given time limit T that must be sufficient for $I_{\max}$ decoding iterations. If each decoding iteration uses n clock cycles, this requires a clock frequency of $f_{\text{CLK}} = \frac{nI_{\max}}{T}$ and a supply voltage of $\mathrm{f}^{-1}(f_{\text{CLK}})$, where $\mathrm{f}(\cdot)$ relates voltage to the maximum clock frequency of a circuit as shown in Table 4.1. From these relationships, we obtain the baseline energy consumption for decoding one code-block as

$$\bar{E} = \alpha C \left[\mathrm{f}^{-1}\left(\frac{nI_{\max}}{T}\right)\right]^2 nI_{\max} \tag{4.3}$$

With straightforward ET, the number of required iterations (RIs) r_{i} for a codeword becomes a random variable (RV) R_{I} with

$$R_{\mathrm{I}} \sim \Pr_{R_{\mathrm{I}}}(r_{\mathrm{i}}) = \mathbb{P}(R_{\mathrm{I}} = r_{\mathrm{i}}),\ r_{\mathrm{i}} \in \mathscr{I}, \tag{4.4}$$

where $\mathscr{I} = \{1, 2, ..., I_{\max}\}$ and where $\Pr_{R_{\mathrm{I}}}(r_{\mathrm{i}})$ is the empirical probability mass function (pmf) of R_{I}, which can be obtained from Monte-Carlo (MC) simulations. However, the supply voltage is still selected based on the maximum number of RIs to guarantee that the decoding deadline is always met. Hence the average decoding energy per codeword with ET is obtained from

$$\bar{E}_{\text{ET}} = \sum_{r_{\mathrm{i}}=1}^{I_{\max}} \Pr_{R_{\mathrm{I}}}(r_{\mathrm{i}})\, \alpha C \left[\mathrm{f}^{-1}\left(\frac{nI_{\max}}{T}\right)\right]^2 nr_{\mathrm{i}}, \tag{4.5}$$

which trivially shows the first order energy reduction.

If we could further reduce f_{CLK} and the corresponding supply voltage for each codeword to the minimum frequency that is sufficient to just complete $r_{\mathrm{i}} < I_{\max}$ decoding iterations within the time budget T, the total energy can be further reduced

$$\bar{E}_{\mathrm{DVFS}} = \sum_{r_{\mathrm{i}}=1}^{I_{\max}} \Pr_{R_{\mathrm{i}}}(r_{\mathrm{i}})\, \alpha C \left[\mathrm{f}^{-1}\left(\frac{n r_{\mathrm{i}}}{T}\right)\right]^2 n r_{\mathrm{i}}, \tag{4.6}$$

due to the quadratic relationship of the energy and the supply voltage [132] and since the actual number of iterations r_{i} which is always smaller than $I_{\max}$ now enters f^{-1}. This is the principle idea to achieve second-order energy savings with DVFS based ET.

4.1.2 Prior Art

A key requirement to be able to effectively apply DVFS to LDPC decoders with ET is that the number of RIs to accomplish the decoding task can be anticipated to properly select f_{CLK} and V_{dd}. To this end, various algorithms and *prediction metrics* have been studied in the literature. The authors of [128] consider the Hamming weight of the syndrome $\mathbf{s}$ as their prediction metric, since the severity of the noise corruption is correlated with the number of failing parity checks. They then use a look-up table (LUT) to translate this number into an appropriate setting for the DVFS controller assuming that the expected number of iterations is not greater than 1.5 times the average number of decoding iterations in each tabulated prediction metric interval. The prediction accuracy was improved in [129] with a pre-processing phase, defined as three (3) sub-iterations, and by using the updated syndrome $\mathbf{s}$ as the input to the prediction algorithm. Since the effectiveness of a static approach relies only on a single prediction obtained during the early decoding stages, [130] and [131] propose an online power management. Both works first try to fit a linear approximation function for the number of iterations to the Hamming weight of the syndrome $\mathbf{s}$ and then employ that function as the prediction metric to dynamically adjust voltage and frequency accordingly.

Unfortunately, these previous publications do not provide a rigorous mathematical link between their prediction of the number of RIs, the chosen DVFS setting, and the impact on the error correction performance. Furthermore, especially for decoders that rely on layered MP [123], the overhead for the calculation of the syndrome $\mathbf{s}$ is usually not negligible, which makes the power saving offered by ET and DVFS less effective. Therefore, it is essential to simultaneously analyze the power reduction and the performance degradation. To this end, we first formulate the energy expenditure of the decoder with iteration prediction in the following section in a rigorous way and we then elaborate on the corresponding proposed algorithm for DVFS management with a well-defined error rate performance loss in Section 4.3.

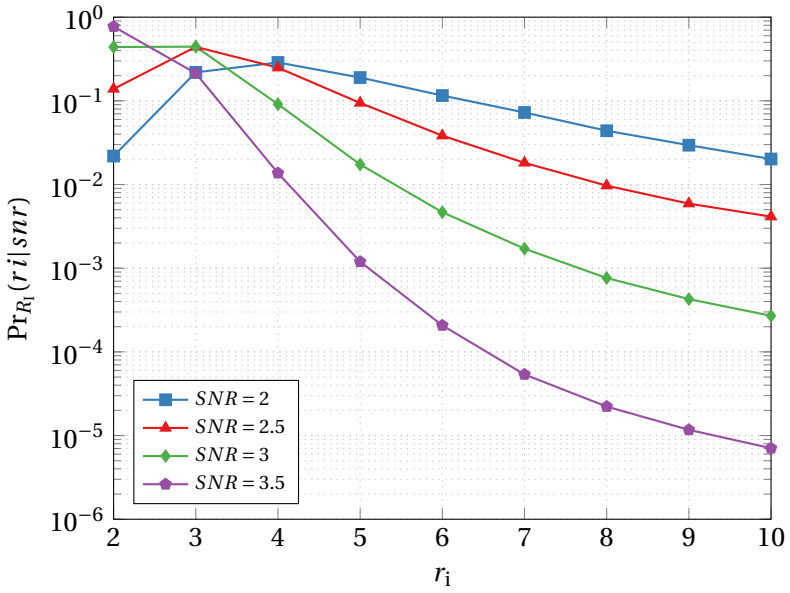


Figure 4.1: Probability distribution of R_I for LDPC decoder for IEEE 802.11ad code conditioned on different SNRs.

4.2 Energy Saving Analysis in LDPC Decoders

The efficiency of DVFS based power reduction for LDPC decoders depends on the accuracy of the iteration prediction algorithm. In essence, an earlier correct choice of the DVFS-setting leads to longer operation in a more energy efficient regime. In this section, we adopt a rigorous statistical model for the number of RIs as well as for the energy consumption for decoding of each codeword and employ that model to formulate the impact of the prediction algorithm on energy efficiency and error rate. For the rest of the chapter, we consider the decoder for the quasi-cyclic (QC)-LDPC code in the IEEE 802.11ad standard [133] for the purpose of simulation, while the analyses remain general and can be applied to any LDPC code. We note that in Chapter 3 we used a similar decoder (and the corresponding architecture), but for a different code than that of this chapter.

4.2.1 Statistical Analysis of LDPC Decoder Iterations

Due to the correlation between the number of RIs and the codeword SNR, we propose to first examine the probability of the number of RIs conditioned on SNR, as $\Pr_{R_\mathrm{I}}(r_\mathrm{i}|snr)$ (cf. (4.4)).

The $\Pr_{R_\mathrm{I}}(r_\mathrm{i}|snr)$ for our decoder for the IEEE 802.11ad LDPC code with layered min-sum (MS) decoding and $I_{\max} = 10$ is illustrated in Fig. 4.1. This figure indicates that the average number of RIs, $\mathbb{E}\{R_\mathrm{I}\}$, decreases as SNR increases and thus more (average) energy saving becomes available at higher SNRs.

4.2.2 Energy Saving of LDPC Decoder with Iteration Prediction

The DVFS based energy saving approach sets the iteration limit (IL) i_l, i.e., the maximum number of allowed iterations for each codeword, and the corresponding decoder frequency and voltage based on some properties of the received data. Hence, we also treat i_l as a RV I_L with the pmf $\Pr_{I_\mathrm{L}}(i_\mathrm{l}) = \mathbb{P}(I_\mathrm{L} = i_\mathrm{l})$. With this, we establish the expectation of total energy consumption $\bar{E}_{I_\mathrm{L}}$ per codeword as the algorithm optimization metric defined by[1]

$$\bar{E}_{I_\mathrm{L}} = T \sum_{i_\mathrm{l}=1}^{I_\mathrm{max}} \Pr_{I_\mathrm{L}}(i_\mathrm{l}) P_{i_\mathrm{l}}\,, \tag{4.7}$$

where we omit the conditioning on SNR for brevity and where $P_{i_\mathrm{l}} = \alpha C\left[\mathrm{f}^{-1}\left(\frac{n i_\mathrm{l}}{T}\right)\right]^2 \frac{n i_\mathrm{l}}{T}$ is the decoding power consumption in the operation mode associated with i_l. Thus, the algorithm effort to minimize the average decoder energy consumption consists of shaping the distribution $\Pr_{I_\mathrm{L}}(i_\mathrm{l})$ to minimize (4.7), while keeping the risk of setting a too restrictive IL, where $r_\mathrm{i} > i_\mathrm{l}$, low.

To understand the theoretical limit on energy reduction of an LDPC decoder, we evaluate (4.7) for the case of a genie-aided prediction algorithm that always chooses the IL i_l to match exactly the number of RI r_i before even starting the first decoding iteration. In this case, we note that $\Pr_{I_\mathrm{L}}(i_\mathrm{l}) = \Pr_{R_\mathrm{I}}(r_\mathrm{i})$. Further, we assume a fine-grained ideal DVFS controller that is able to tune the clock frequency and voltage sufficiently according to the required throughput for the entire decoding period. This allows us to obtain the corresponding minimum average energy expenditure

$$\bar{E}^*_\mathrm{DVFS} = T \sum_{r i=1}^{I_\mathrm{max}} \Pr_{R_\mathrm{I}}(r_\mathrm{i}) P_{r_\mathrm{i}}\,, \tag{4.8}$$

which corresponds to (4.6) when choosing P_{r_i} as the dynamic power in the operation mode in which the IL is set to r_i.

We have calculated (4.8) for the decoder of IEEE 802.11ad code according to an ideal DVFS controller for typical operating conditions in 28 nm FD-SOI, shown in Table 4.1. By plugging P_{r_i} of the corresponding operation mode from this table into (4.8) and the $\Pr_{R_\mathrm{I}}(r_\mathrm{i})$ for $SNR = 3dB$ from Fig. 4.1, the minimum average energy expenditure adds up to $\bar{E}^*_\mathrm{DVFS} = 0.146\,\bar{E}$. This calculation shows the maximum theoretically-available energy reduction at this SNR for the above decoder in 28 nm FD-SOI when the iteration prediction approach is employed. Note that this genie-aided limit has no penalty in terms of error-correction performance.

[1] For this derivation, we assumed that the algorithm sets the IL i_l at the start of the decoding.

Table 4.1: Ideal DVFS controller in 28 nm FD-SOI.

Operation Mode	Voltage	Frequency	Power	Slowdown Factor
M_0	V_{max}	f_{max}	P_{max}	1.0
M_1	$0.95V_{max}$	$0.89f_{max}$	$0.78P_{max}$	1.1
M_2	$0.90V_{max}$	$0.78f_{max}$	$0.62P_{max}$	1.2
M_3	$0.85V_{max}$	$0.68f_{max}$	$0.48P_{max}$	1.4
M_4	$0.80V_{max}$	$0.57f_{max}$	$0.35P_{max}$	1.7
M_5	$0.75V_{max}$	$0.44f_{max}$	$0.24P_{max}$	2.2
M_6	$0.70V_{max}$	$0.35f_{max}$	$0.16P_{max}$	2.8
M_7	$0.65V_{max}$	$0.25f_{max}$	$0.10P_{max}$	4.0
M_8	$0.60V_{max}$	$0.16f_{max}$	$0.05P_{max}$	6.1

4.3 Statistical Based Prediction for Energy Saving in LDPC Decoders

In the previous section we have examined the energy saving offered by DVFS based on a genie-aided iteration prediction. Even though a non-ideal prediction algorithm for the IL reduces the decoder energy consumption compared to ET without DVFS in a similar way, it also degrades the error correction performance when the IL is occasionally set too low ($i_\mathrm{l} < r_\mathrm{i}$). To properly account for this issue, we rigorously formulate the trade-off between the error correction performance penalty and the IL prediction pessimism and propose a prediction algorithm that can be tuned for maximum energy savings with a maximum error rate performance penalty constraint.

4.3.1 SNR Based Iteration Management with Performance Penalty

Since the IL prediction algorithm entails a small performance loss in return for energy saving, the idea of iteration management pertains to the area of approximate computing, as discussed in this thesis. To quantify the performance loss, we formulate the frame-error rate performance degradation that occurs due to inaccurate (optimistic) predictions of the IL. To this end, we first define the probability of a frame error at a given SNR caused by incomplete decoding of a codeword due to pre-mature termination of the decoding procedure when $r_\mathrm{i} > i_\mathrm{l}$ as

$$\mathrm{Pr}_e(i_\mathrm{l}) = \mathbb{P}(R_\mathrm{I} > i_\mathrm{l}), \tag{4.9}$$

where we omit the additional conditioning on the SNR for brevity. We note that $\mathrm{Pr}_e(i_\mathrm{l})$ can be considered as a frame error rate (FER) penalty which can be added to the decoder FER without DVFS to obtain a union upper-bound on the FER with DVFS based ET. By limiting the FER penalty $\mathrm{Pr}_e(i_\mathrm{l})$ to a user-defined threshold (B_u) we can now derive the corresponding

constrained energy-optimal IL $i_{\mathrm{l}\mathrm{opt}}$ as

$$i_{\mathrm{l}\mathrm{opt}} = \min_{\{i_{\mathrm{l}}:\mathrm{Pr}_e(i_{\mathrm{l}})\leq B_u\}} i_{\mathrm{l}}. \tag{4.10}$$

Due to the small number of choices $i_{\mathrm{l}} \in \{1,2,...,I_{\max}\}$, a corresponding DVFS controller can easily evaluate (4.10) with an exhaustive search in a small SNR-specific LUT for $\mathrm{Pr}_e(i_{\mathrm{l}})$. The content of this LUT is derived off-line from (4.4) (cf. Fig. 4.1) as obtained from MC simulations.

4.3.2 Statistical Based Iteration Prediction Algorithm

Unfortunately, we observe that the above-described SNR based prediction of the IL only leads to a small energy efficiency improvement compared to straightforward ET without DVFS if the target FER performance degradation is set to be small. The reason for this behavior is that the tail of the distribution of the number of RIs $\mathrm{Pr}_{R_{\mathrm{I}}}(r_{\mathrm{i}})$ is fat compared to the permissible error rate performance degradation B_u which follows the FER. Hence, conditioning on only the SNR leads to a conservative choice of i_{l} and thus only to small energy savings.

To alleviate this problem, we follow the proposal of the original work in [128] which performs a more fine-grained adjustment of the IL based on a metric C (e.g., the Hamming weight of $\mathbf{s}$) that is derived from each individual received codeword. We adjust our algorithm by introducing the pmf of the number of RIs conditioned on the metric C as $\mathrm{Pr}_{R_{\mathrm{I}}|C}(r_{\mathrm{i}}|c)$ and the pmf of C as $\mathrm{Pr}_C(c)$. We then rewrite (4.9) as

$$\mathrm{Pr}_{\mathrm{e}|\mathrm{C}}(i_{\mathrm{l}}) = \mathbb{P}(R_{\mathrm{I}} > i_{\mathrm{l}}|C = c) \tag{4.11}$$

and adjust the selection of the IL in (4.10) accordingly to obtain the refined IL $i_{\mathrm{l}\mathrm{opt}|C}$ from

$$i_{\mathrm{l}\mathrm{opt}|C} = \min_{\{i_{\mathrm{l}}:\mathrm{Pr}_{e|C}(i_{\mathrm{l}})\mathrm{Pr}_C(c)\leq B_u\}} i_{\mathrm{l}}, \tag{4.12}$$

which keeps the average (across all instances of the metric c) FER penalty from not reserving enough iterations for decoding a frame below B_u. Similar to the SNR based algorithm, this selection can also be performed based on LUTs. However, we now require a separate LUT for different values of c and for each SNR. By partitioning both c and the SNR into intervals, the number of LUTs can be reduced. The intervals can be chosen based on different upper limit for c that result in the different values for the $i_{\mathrm{l}\mathrm{opt}|C}$.

4.3.3 Calculation of the Prediction Metric

We note that there are two parameters that affect the correlation between the metric c and the number of RIs and hence also the accuracy of the IL and the corresponding power savings

for a given error rate degradation: the conditioning *metric c itself* and *when it is calculated* during the decoding process.

A straightforward choice for the conditioning metric is the Hamming weight of the syndrome **s**. Unfortunately, the calculation of this metric for each codeword requires significant overhead in MP decoders with a layered schedule. We therefore propose to use the codeword LLR sign change (LSC) after each iteration, which is the number of changes in the sign-bit of the decoded LLRs compared to the previous iteration, as an alternative metric. This metric comes with only a negligible computational overhead as it can be updated on-the-fly.

Further, we note that the correlation between the conditioning metric and the number of required remaining iterations increases at later iterations of the decoding process. Hence, it can be advantageous to start the decoding process without voltage-frequency down-scaling under the assumption that $I_{\max}$ iterations may be required. The prediction metric c is then only computed after a few initial decoder iterations and DVFS based ET is applied only to the remaining iterations. While in this case a few initial (e.g., one or two) iterations cannot take advantage of DVFS, the subsequent iterations profit from a more reliable and less pessimistic choice of the IL.

4.3.4 Simulation Results

We have simulated the proposed iteration prediction algorithm for the QC-LDPC decoder for the IEEE 802.11ad code, which has a codeword length $N = 672$ with blocks of $Z = 42$ bits, for a rate of $R = 1/2$. We use layered MS decoding with $I_{max} = 10$ and we carry out the first two iterations without DVFS at nominal voltage. After the second iteration, we compute the metric c using either the Hamming weight of the syndrome **s** or the LSC and apply DVFS for the remaining iterations. Furthermore, we always apply regular ET, even on top of DVFS to avoid any unnecessary iterations.

We first illustrate the algorithm efficiency compared to the SNR based DVFS approach. Fig. 4.2 shows the conditional probability distributions of R_{I} times the condition probability, i.e., $\Pr(r_{\mathrm{i}}|c)\Pr(c)$, at $SNR = 3dB$ and for the three intervals of the Hamming weight of **s**, as the condition metric c. These intervals are chosen such that they result in the three most frequent choices of the IL for $B_u = 10^{-3}$ with respect to (4.12). These choices amount to a total of 3, 4, and 5 iterations for the above B_u, as can be derived from Fig. 4.2, and are selected with probabilities of 47%, 11%, and 20% after the second iteration. When comparing the distributions in Fig. 4.2 to those in Fig. 4.1, we notice that the tails are much thinner, which favors the choice of a smaller IL for each condition metric c. Furthermore, we note that the metric interval I_1, which leads to smallest IL, is also the one that is relevant for the majority of the frames.

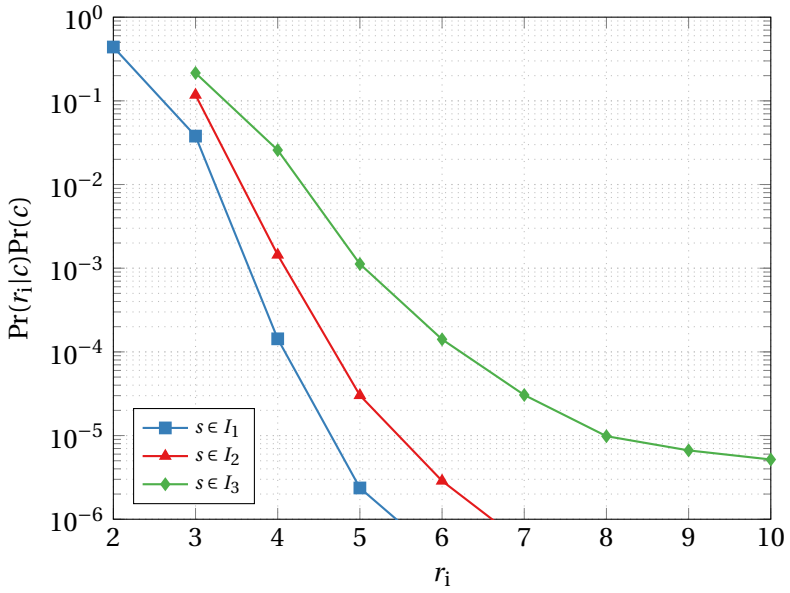
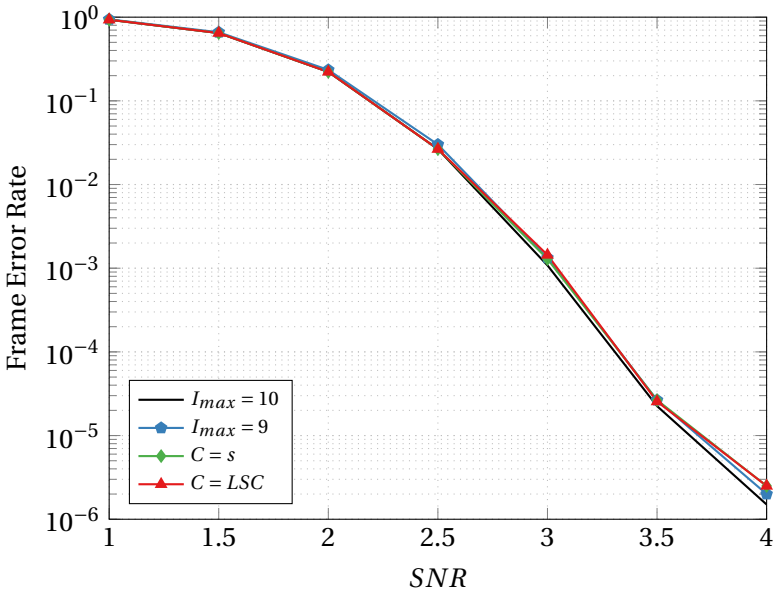


Figure 4.2: Conditional probability distribution of R_I times the conditions probability $\Pr(r_i|c)\Pr(c)$ at $SNR = 3dB$, where $c = |s|$ and where the conditioning is performed after the second iteration for three intervals with $\Pr(s \in I_1) = 0.47$, $\Pr(s \in I_2) = 0.11$, and $\Pr(s \in I_3) = 0.20$.

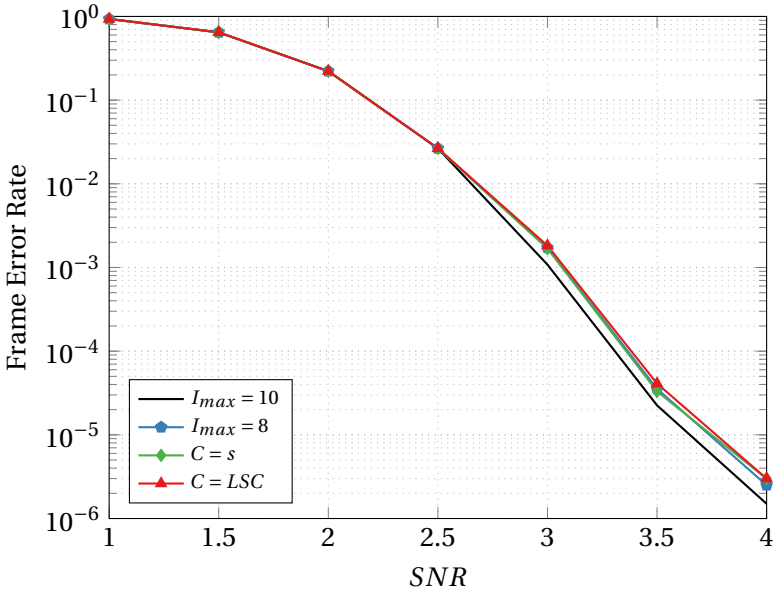
In order to illustrate the effect of the proposed algorithm on the decoder performance, we show the FER for the two discussed conditioning metrics and two approximation levels B_u in Fig. 4.3. For fairness, we also show the FER of a decoder with $I_{\max} < 10$ that benefits from running continuously at an operation mode with lower power than M_0 and has almost the same performance degradation as the decoder with DVFS based ET with $I_{\max} = 10$. Specifically, the decoders with $I_{\max} = 9$ and $I_{\max} = 8$ have similar error rates to the decoder with $I_{\max} = 10$ and iteration prediction algorithm with $B_{u,1}$ and $B_{u,2}$, as shown in Fig. 4.3a and 4.3b, respectively. To determine the impact on power, we first calculate $\Pr(i_l)$ for $i_l = \{3, ..., I_{\max}\}$ at $SNR = 3dB$ and note the results in Table 4.2. We then estimate the average power of the decoder by combining $\Pr(i_l)$ with the power consumption of the corresponding mode[2] from Table 4.1 according to (4.7). We correct the results for the fact that we always run the first two iterations without DVFS in anticipation of $I_{\max}$. We further include the impact of regular ET on top of DVFS to finally calculate the total average energy per codeword, which is shown as the percentage of $\bar{E}$ in Table 4.2.

According to this calculation, regular ET without DVFS reduces the consumed energy per codeword by 62.7%. If we also apply voltage scaling according to iteration prediction with $B_{u,1}$, the average energy is further reduced (compared to the first column in the table) by 22% and 18.5% for the Hamming weight of the syndrome $\mathbf{s}$ and LSC as prediction metrics, respectively. However, this reduction is only 12.6% if we compare to simply reducing the maximum number

[2]The corresponding mode to each entry of Table 4.2 can be found with respect to i_l, $I_{\max}$, and the slowdown factor.



(a)



(b)

Figure 4.3: FER of the decoder for IEEE 802.11ad code with $I_{max} = 10$ and a) the decoder with $I_{max} = 9$ in comparison with the decoders with iteration prediction algorithm with the two conditioning metrics and approximation level $B_{u,1}$; b) the decoder with $I_{max} = 8$ in comparison with the decoders with iteration prediction algorithm with the two conditioning metrics and approximation level $B_{u,2}$.

Table 4.2: The probabilities of IL $\Pr(i_l)$ (shown in %) for the proposed algorithm with two different bounds at $SNR = 3dB$ when conditioning metric is the Hamming weight of the syndrome **s** or LSC, the corresponding average energy expenditure per codeword after ET (shown as percentage of $\bar{E}$, cf. (4.3)), and the corresponding improvement (shown as percentage of $\bar{E}_{\mathrm{ET}}$, cf. (4.5)) for the IEEE 802.11ad decoder.

	no prediction, $Imax =$			prediction with $B_{u,1} = 10^{-3}$		prediction with $B_{u,2} = 2 \times 10^{-3}$	
i_l	10	9	8	$C = \lvert s \rvert$	$C = LSC$	$C = \lvert s \rvert$	$C = LSC$
3	0	0	0	47.8	5.8	47.8	10.5
4	0	0	0	11.9	33.6	27.1	44.5
5	0	0	0	20.2	29	9	23.9
6	0	0	0	4	10.4	9.5	7.5
7	0	0	0	5.7	7.5	2.5	5.1
8	0	0	100	3.7	5.1	1.6	3.4
9	0	100	0	0	0	0	0
10	100	0	0	6.6	8.4	2.6	5
$100 \times \frac{\bar{E}_{\mathrm{ET/DVFS}}}{\bar{E}}$	37.3	32.6	29.6	29.1	30.4	27.9	29.4
$100 \times (1 - \frac{\bar{E}_{\mathrm{DVFS}}}{\bar{E}_{\mathrm{ET}}})$	–	12.6	20.6	**22**	**18.5**	**25.2**	**21.2**

of iteration to $I_{max} = 9$. If we further relax the performance by choosing $B_{u,2}$ for the algorithm with the two metrics or by reducing the maximum number of iteration to $I_{max} = 8$, we get 25.2%, 21.2%, and 20.6% reductions, respectively. Note that the preceding percentages have been calculated with respect to the total energy consumption after regular ET, which are shown in the last line of Table 4.2.

The results prove the efficiency of the proposed iteration prediction algorithm over the simple reduction of I_{max} while both approaches have the same performance penalty. Further, they indicate that using the Hamming weight of the syndrome **s** as the conditioning metric is more beneficial than LSC. However, the computational overhead for the Hamming weight of the syndrome **s** is not negligible and thus we have implemented the algorithm using LSC for the decoder previously presented in [114]. According to our postlayout analysis, the power overhead of the iteration prediction unit is below 1% and therefore the reported energy savings remain legitimate.

4.4 Conclusion

A DVFS based power management framework was proposed in this chapter that provides second-order energy proportionality in an LDPC decoder by exploiting run-time variations.

The algorithm predicts an iteration limit for each codeword according to a conditional probabilities table, which is calculated offline using MC simulation on decoding iterations, and configures the DVFS controller for the appropriate operation mode. The algorithm can deliver variable power reductions for different approximation levels that can be selected through a user-defined upper bound. Implementation results ensure that the proposed framework and the corresponding algorithm notably reduce the power consumption of the decoder while a minimum error correction performance is guaranteed.

5 Toward Energy and Area Optimization of High-Throughput LDPC Decoders by Exploiting Quantized Message Passing

Approximate computing broadly refers to methods that exploit the intrinsic resilience of applications to realize improvement in efficiency of operations. Despite the recent interest, the core principles of approximate computing are not new and the techniques have been widely employed in many applications to avoid unnecessary operations or other (e.g., storage) costs. The history of signal processing is filled with examples that utilize approximate computing principles [28]. Specifically, the domain of very-large-scale integration (VLSI) signal processing contains many examples from approximation of algorithms or some specific functions to approximate arithmetic operations and finite word-length optimization.

Most of the ancient techniques in VLSI signal processing refer to measures at design-time. We refer to these measures as *static approximate computing*. The impact of such techniques is fully known and deterministic and the output quality (degradation) is often measured easily at design-time with analytical considerations or using simulations. Such techniques are then useful to improve the efficiency of the operations. However, they are unrelated to uncertainties or variations. On the contrary, *dynamic approximate computing* techniques are effective while dealing with variations using run-time measures. During the previous chapters of this thesis, we have emphasized on dynamic approximate computing, while the focus of this chapter is on static approximate computing.

Different well-established design-time approximation techniques are used to improve the efficiency of the operations and arithmetic units for signal processing applications. For example, approximate recursive algorithms are used to compute some specific functions, such as transcendental functions [29]. Algebraic techniques are used to approximate complex operations [30]. Finite word-length optimizations are heavily used to approximate any arithmetic operation [31]. Along the same lines, non-uniform quantization is used to improve the signal

to quantization noise level and thus provides a better approximation with a shorter word-length in comparison to uniform quantization [134]. This idea is beneficial for applications that demand a high dynamic range for a signal represented with a finite word-length [135].

Among the applications with a need for high dynamic range numbers, fast decoders for low-density parity check (LDPC) codes have recently attracted a lot of attention [77, 136]. Such decoders are often based on an unrolled architecture with duplicated processing nodes and a complex interconnect network between the processing nodes, which poses a challenge to an efficient implementation. Non-uniform quantization can reduce the complexity and improve the efficiency of the implementation by decreasing the required word-length and thus the interconnect network complexity.

Contributions and Outline Motivated by the static approximate computing techniques, we employ a recent non-uniform quantization scheme to reduce the word-length of the arithmetic units for a high-throughput LDPC decoder, which significantly reduces the implementation complexity. The arithmetic units are optimized based on an information-theoretic criterion to prevent a loss in the decoder performance. This non-uniform quantization scheme is employed in an architecture based on a serial transfer of the message bits, which together enable the high-throughput implementation of the decoder with a very good area and energy efficiency.

The remainder of the chapter is organized as follows. Section 5.1 gives a more detailed introduction to decoding of LDPC codes compared to the previous chapters to follow the details of this contribution, as well as more details on existing high-throughput implementations of LDPC decoders. The associated challenges and limitations are also summarized in this section. Section 5.2 describes the proposed ultra-high throughout decoder architecture that employs a serial message-transfer technique and serves as the baseline architecture for this chapter. In Section 5.3, the approximate computing based algorithm to design a finite-alphabet decoder with non-uniform quantization is explained and applied to the serial message-transfer decoder architecture. Section 5.4 describes the proposed approach for the physical implementation and the timing and area optimization of the serial message-transfer decoders. Finally, Section 5.5 analyzes the implementation results, and Section 5.6 concludes the chapter.

5.1 Background

In this section, we first briefly summarize the fundamentals of LDPC codes and provide further details of the iterative MP algorithm for the decoding, as compared to the previous chapters of the thesis. We then review the state-of-the-art in high speed LDPC decoder architectures to set the stage for the description of our implementation.

5.1.1 LDPC Codes and Decoding Algorithms

A binary LDPC code is a set of codewords which are defined through an $M \times N$ binary-valued sparse parity check matrix as

$$\{\mathbf{c} \in \{0,1\}^N | \boldsymbol{H}\mathbf{c} = \mathbf{0}\}, \tag{5.1}$$

where all operations are performed modulo-2. If the parity check matrix contains exactly d_v ones per column and exactly d_c ones per row, the code is called a (d_v, d_c)-regular LDPC code. Such codes are usually represented with a *Tanner graph*, which contains N variable nodes (VNs) and M check nodes (CNs) and VN n is connected to CN m if and only if $\boldsymbol{H}_{mn} = 1$.

LDPC codes are traditionally decoded using message passing (MP) algorithms, where information is exchanged as messages between the VNs and the CNs over the course of several decoding iterations and where the exchanged messages represent log likelihood ratios (LLRs). At each iteration the message from VN n to CN m is computed using a mapping $\Phi_v : \mathbb{R}^{d_v} \to \mathbb{R}$, which is defined as

$$\mu_{n\to m} = \Phi_v\big(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus m\to n}\big), \tag{5.2}$$

where $\mathcal{N}(n)$ denotes the neighbors of node n in the Tanner graph, $\bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus m\to n} \in \mathbb{R}^{d_v-1}$ is a vector that contains the incoming messages from all neighboring CNs except m, and $L_n \in \mathbb{R}$ denotes the channel LLR corresponding to VN n. Similarly, the CN-to-VN messages are computed using a mapping $\Phi_c : \mathbb{R}^{d_c-1} \to \mathbb{R}$, which is defined as

$$\bar{\mu}_{m\to n} = \Phi_c\big(\boldsymbol{\mu}_{\mathcal{N}(m)\setminus n\to m}\big). \tag{5.3}$$

Fig. 5.1 illustrates the message updates in the Tanner graph. In addition to Φ_v and Φ_c, a third mapping $\Phi_d : \mathbb{R}^{d_v+1} \to \{0,1\}$ is needed to provide an estimate of the transmitted codeword bits in the last VN iteration based on the incoming CN messages and the channel LLR L_n according to

$$\hat{c}_n = \Phi_d(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\to n}). \tag{5.4}$$

Messages are exchanged until a valid codeword has been decoded or until the maximum number of iterations I has been reached.

Among the various MP decoding algorithms, the min-sum (MS) decoding algorithm [72] and its variants (e.g., offset MS, scaled MS) are the most common choices for hardware implementation. For the widely used MS algorithm, the mappings (5.2) and (5.3) are

$$\Phi_v^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = L + \sum_i \bar{\mu}_i, \tag{5.5}$$

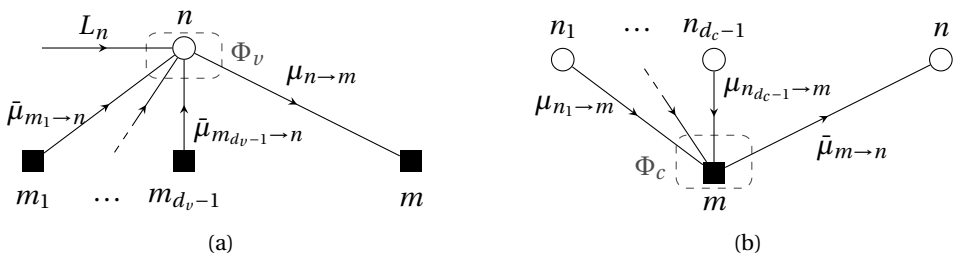


Figure 5.1: (a) VN update and (b) CN update for $\mathcal{N}(n) = \{m, m_1, \ldots, m_{d_v-1}\}$ and $\mathcal{N}(m) = \{n, n_1, \ldots, n_{d_c-1}\}$.

and,

$$\Phi_c^{\mathrm{MS}}(\boldsymbol{\mu}) = \operatorname{sign}\boldsymbol{\mu} \min|\boldsymbol{\mu}|, \tag{5.6}$$

where $\min|\boldsymbol{\mu}|$ denotes the minimum of the absolute values of the vector components and $\operatorname{sign}\boldsymbol{\mu} = \prod_j \operatorname{sign}\mu_j$. The decision mapping Φ_d is defined as

$$\Phi_d^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = \frac{1}{2}\left(1 - \operatorname{sign}\left(L + \sum_i \bar{\mu}_i\right)\right). \tag{5.7}$$

5.1.2 High Throughput LDPC Decoders

Different hardware architectures for LDPC decoders have been proposed in the literature in order to fulfill the power and throughput requirements of various standards. More specifically, various degrees of resource sharing result in flexible decoders with different area requirements. On the one hand, *partial-parallel* LDPC decoders [76, 137] and *block-parallel* LDPC decoders [75, 138] are designed for medium throughput, with modest silicon area. such as decoder architecture considered in Chapter 3 and Chapter 4. *Full-parallel* [74, 139] and *unrolled* LDPC decoders [77, 140], on the other hand, achieve very high throughput (in the order of several tens or hundreds of Gbps) at the expense of large area requirements. In this chapter, we focus on this type of high-throughput decoders.

Several high throughput LDPC decoders have been developed during the past decade in order to satisfy the high data-rate requirements of some applications, such as optical and high-speed Ethernet networks. These decoders usually rely on a full-parallel isomorphic [58] architecture and a flooding schedule, which directly maps the algorithm for *one* iteration to hardware. More specifically, the CN and VN update equations are directly mapped to M CN and N VN processing units and a hard-wired routing network is responsible for passing the messages between them. From an implementation perspective, while such an architecture enables a very high throughput by fully exploiting the inherent parallelism of each iteration, the complexity of the highly unstructured routing network turns out to be a severe bottleneck.

In addition to this routing problem, such full-parallel decoders usually require one or two clock cycles for each iteration and in the worst case as many cycles as the maximum number of iterations for each codeword, which is another throughput limitation factor.

Several solutions have been proposed to alleviate the routing problem in full-parallel decoders, on both architectural and algorithmic levels. The authors of [141, 142] suggest using a bit-serial architecture, which only requires a single wire for each variable-to-check and check-to-variable node connection. While this approach can reduce the routing congestion, it also leads to a significant reduction in the decoding throughput. The decoder in [142], for example, only achieves a throughput of 3.3 Gbps when implemented using a 130 nm CMOS technology. Another architectural technique is reported in [139], where the long wires of the decoder are partitioned into several short wires with pipeline registers. As a result, the critical path is broken down into shorter paths, but the decoder throughput is also affected since more cycles are required to accomplish each iteration. Nevertheless, the decoder of [139] is still able to achieve 13.2 Gbps in 90 nm CMOS with 16 iterations.

On an algorithmic level, the authors of [143] propose a MP algorithm, called *MS split-row threshold*, which uses a column-wise division of the $\boldsymbol{H}$ matrix into S_{pn} partitions. Each partition contains N/S_{pn} VNs and M CNs, and global interconnects are minimized by only sharing the minimum signs between the CNs of each partition. This algorithmic modification was used to implement a full-parallel decoder for the challenging (2048, 1723) LDPC code in 65 nm CMOS, which achieves a throughput of 36.2 Gbps with 11 decoding iterations. Another decoder, reported in [144], uses a hybrid hard/soft decoding algorithm, called *differential binary MP* algorithm, which reduces the interconnect complexity at the cost of some error-correcting performance degradation. A full-parallel (2048, 1723) LDPC decoder using this algorithm was implemented in 65 nm CMOS technology, achieving a throughput of up to 126 Gbps [145]. The work of [146] also proposes another algorithmic level modification, called the probabilistic MS algorithm, where a probabilistic second minimum value is used instead of the true second minimum value to simplify the CN operation and to facilitate high-throughput implementation of full-parallel LDPC decoders. Further, a mix of tree and butterfly interconnect network is proposed in the CN unit to balance the interconnect complexity and the logic overhead and to reduce the routing complexity. The implementation of a decoder for the (2048, 1723) LDPC code with the proposed techniques in 90 nm CMOS technology achieves a throughput of 45.42 Gbps.

To solve the problem of throughput limitations in full-parallel decoders from potentially using multiple iterations for decoding, the work of [140] presents an unrolled full-parallel LDPC decoder exploring an additional degree of parallelism. In the proposed architecture, each decoding iteration is mapped to distinct hardware resources, leading to a decoder with I iterations that can decode one codeword per clock cycle, at the cost of significantly increased area requirements with respect to non-unrolled full-parallel decoders. This unrolled architecture

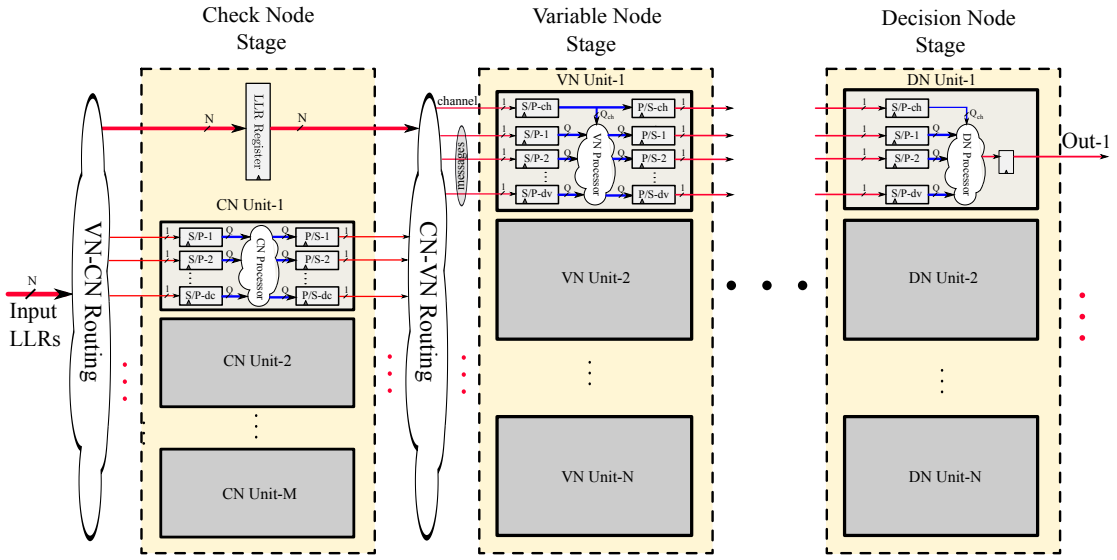


Figure 5.2: Serial message-transfer decoder architecture.

achieves a throughput of 161 Gbps for a $(672, 546)$ LDPC code with $d_v = 3$ and $d_c = 6$, when implemented in a 65 nm CMOS technology. It is noteworthy that an unrolled decoder has 50% reduced wires between adjacent stages compared to a non-unrolled decoder since one stage of variable nodes is connected to one stage of check nodes with uni-directional data flow per decoding iteration. Even though this measure leads to a lower routing congestion, it is still challenging to fully place and route such a decoder. This routing issue becomes more and more severe when considering longer LDPC codes and especially with increasing CN and VN degrees to achieve better error-correcting performance, as required in wireline applications such as for the $(2048, 1723)$ code with $d_v = 6$ and $d_c = 32$ used in the IEEE 802.3an standard [147].

5.2 Serial Message-Transfer LDPC Decoder

In this section, we first propose an unrolled full-parallel LDPC decoder architecture that employs a serial message-transfer technique between CNs and VNs based on conventional fixed-point arithmetic. This architecture is similar to the bit-serial implementations of [141, 142] in the way the messages are transferred, however, the iterations are unrolled to increase the throughput. In the subsequent section we will then show how approximate computing with a custom message format together with corresponding new processing nodes reduces the message-transfer overhead.

5.2.1 Decoder Architecture Overview

An overview of the proposed unrolled serial message-transfer LDPC decoder architecture is shown in Fig. 5.2. As with all unrolled LDPC decoders, each decoding iteration is mapped to a distinct set of N VN and M CN units, which form a processing pipeline. In essence, the unrolled LDPC decoder is a systolic array, in which a new set of N channel LLRs is read in every clock cycle and a decoded codeword is put out in every clock cycle.

Even though both the CNs and VNs can compute their outgoing messages in a single clock cycle, similar to the architecture in [77], in the proposed serial message-transfer architecture each message is transfered one bit at a time between the consecutive variable node and check node stages over the course of Q_{msg} clock cycles, where Q_{msg} is the number of bits used for the messages. More specifically, each CN and VN unit contains a serial-to-parallel (S/P) and parallel-to-serial (P/S) conversion unit at the input and output, respectively, which are clocked Q_{msg} times faster than the processing clock to collect and transfer messages serially, while keeping the overall decoding throughput constant. More details on the architecture of the CN and VN units as well as the proposed serial message-transfer mechanism are provided in the sequel.

5.2.2 Decoder Stages

The unrolled LDPC decoder, illustrated in Fig. 5.2, consists of three types of processing stages, which are described in more detail below. We note that the CN/VN processors of this reference decoder are similar to those of a standard MS decoder, and our modifications for these parts (to realize a finite-alphabet decoder) are discussed in Section 5.3.

5.2.2.1 Check Node Stage

Each check node stage consists of M CN units, each of which contains three components: a CN processor, which implements (5.6) similarly to [77, 140], d_c S/P units for the d_c input messages, and d_c P/S units for the d_c output messages. The CN processor consists of a sorting unit that identifies the two smallest-magnitude messages among all d_c input messages and an output unit which selects the first or the second minimum for each output, along with the appropriate sign. The sorting unit contains 4-input compare-and-select (CAS) units in a tree structure, which identify and output the two smallest values out of the four input values. Moreover, the complete check node stage contains a register bank that is used to store the channel LLRs, which are not directly needed by the check node stage, but nevertheless must be forwarded to the following variable node stage and thus need to be buffered. Hence, no S/P and P/S units are required for the channel LLR buffers in the check node stages as they are simply forwarded serially to the following variable node stage.

5.2.2.2 Variable Node Stage

Each variable node stage consists of N VN units, each of which contains a VN processor and S/P and P/S units at the inputs and outputs, respectively, similar to the CN unit structure. Each VN processor implements the update rule (5.5). Specifically, all input messages are added and then the input message corresponding to each output is subtracted from the sum in order to form the output message, thus implementing the conventional MS update rule.

5.2.2.3 Decision Node Stage

The last variable node stage is called a decision node stage because it is responsible for taking the final hard decisions on the decoded codeword bits. The structure of this stage is similar to a variable node stage, but a decision node (DN) has a simpler version of the VN processor that only computes sum of all inputs and put out its sign, and thus no P/S unit is required at its output.

5.2.3 Message Transfer Mechanism

One of the modifications, compared to [140] and [77], is the serial transfer of the channel and message LLRs between the stages of the decoder, which reduces the required routing resources by factor of Q_{msg}. This modification is applied to make the placement and routing of the decoder feasible, especially for large values of d_v and d_c. To this end, as explained in the previous section, a S/P and a P/S shift register are added to each input and each output of the CN and VN units, as illustrated in Fig. 5.3. We see that the S/P unit consists of a $(Q_{\text{msg}}-1)$-bit shift register and Q_{msg} memory registers, while the P/S unit has Q_{msg} registers with multiplexed inputs. The serial messages are transfered with a fast clock, denoted by CLK_F, that is Q_{msg} times faster than the slow processing clock, denoted by CLK_S. More specifically, at each CN unit and VN unit input, data is loaded serially into the S/P shift register using the fast CLK_F, and after the Q_{msg}-th cycle all message bits are stored in memory registers, clocked by the slow CLK_S. The CN/VN processing can then be performed in one CLK_S cycle and the output messages are saved in the output P/S shift register and transferred serially to the next stage using again CLK_F. At the same time, a new set of messages is serially loaded into the input shift register. We note that for simpler clock tree generation, all registers in Fig. 5.3 are clocked by CLK_F, while CLK_S is actually implemented as a pulsed clock, which is generated using a clock-gating cell controlled by a finite state machine.

5.2.4 Decoder Hardware Complexity and Performance Analysis

In this section, we describe the required memory complexity of the proposed decoder as well as the decoding latency and the throughput. The results from this analysis then motivate a

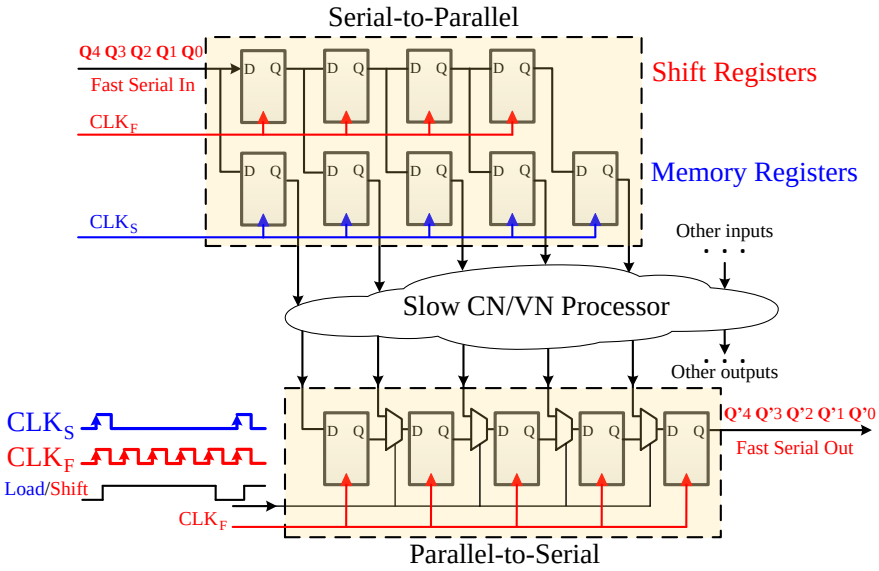


Figure 5.3: The message receive and transfer mechanism by S/P and P/S shift registers, enabled by the fast clock ($\mathrm{CLK_F}$), and the message process enabled by the slow clock ($\mathrm{CLK_S}$) for $Q_{\mathrm{msg}} = 5$.

message approximation that is better and at the same time more accurate than the traditional fixed-point format (see Section 5.3).

5.2.4.1 Memory Requirement

The memory complexity can easily be characterized by counting the number of required registers and can be approximated by

$$R_{\mathrm{tot}} \approx N(d_v + 1)Q(6I - 1), \tag{5.8}$$

where I is the number of decoding iterations (which in unrolled decoders strongly affects the memory requirements) and $Q = Q_{\mathrm{msg}} = Q_{\mathrm{ch}}$, which is often the case for MS LDPC decoders. From (5.8), one can easily see that the quantization bit-width linearly increases the memory requirement for the proposed architecture.

5.2.4.2 Decoding Latency

Since each stage has a delay of two $\mathrm{CLK_S}$ cycles and there are two stages for each decoding iteration, the decoder latency is $4I$ $\mathrm{CLK_S}$ cycles or, equivalently, $4IQ_{\mathrm{max}}$ $\mathrm{CLK_F}$ cycles, where $Q_{\mathrm{max}} = \max(Q_{\mathrm{msg}}, Q_{\mathrm{ch}})$.

5.2.4.3 Decoding Throughput

In the proposed unrolled architecture, one decoder codeword is output in each $\mathrm{CLK_S}$ cycle. Therefore, the coded throughput of the decoder is

$$T = N f_{\mathrm{S_{max}}}, \tag{5.9}$$

where $f_{\mathrm{S_{max}}}$ is the maximum frequency of $\mathrm{CLK_S}$ while the maximum frequency of $\mathrm{CLK_F}$ or simply maximum frequency of the decoder is $f_{\max} = f_{\mathrm{F_{max}}} = Q_{\max} f_{\mathrm{S_{max}}}$. For the proposed architecture, we have

$$f_{\mathrm{S_{max}}} = \left\{ \max\left((Q_{\max} T_{\mathrm{CP,route}}), (T_{\mathrm{CP,VN}}), (T_{\mathrm{CP,CN}}) \right) \right\}^{-1}, \tag{5.10}$$

where $T_{\mathrm{CP,VN}}$ and $T_{\mathrm{CP,CN}}$ are the delay of the critical paths of the CN unit and the VN unit, respectively, and $T_{\mathrm{CP,route}}$ is the critical path delay of the (serial) routing between the decoding stages. Thus, the decoder throughput will be limited by the routing, if the VN/CN delay is smaller than $Q_{\max}$ times the routing delay. Hence, on the one hand, the serial message-transfer decoder alleviates the routing problem by reducing the required number of wires, but on the other hand, the decoder throughput for large quantization bit-widths may be affected, as the serial message-transfer delay will become the limiting factor.

5.3 Finite-Alphabet Serial Message-Transfer LDPC Decoder

Even though the serial message-transfer architecture alleviates the routing congestion of an unrolled full-parallel LDPC decoder, it has a negative impact on both throughput and hardware complexity, as discussed in the previous section. The issue increases with increasing number of bits required to represent the messages for sufficient decoding error rate performance. Hence, it is desirable to approximate the full-precision messages with fewer bits and higher accuracy.

Recently, there has been significant interest in the design of *finite-alphabet* decoders for LDPC codes [77, 136, 148–152]. The main idea behind finite-alphabet LDPC decoders is to start from one or multiple arbitrary message alphabets, which can be encoded with a bit-width that is acceptable from an implementation complexity perspective. The message update rules are then crafted as generic mapping functions to operate on this alphabet, as (5.2), (5.3), and (5.4). The main advantage of such finite-alphabet decoders is that the message bit-width can be reduced significantly with respect to a conventional fixed-point decoder, while maintaining the same error-correcting performance [77, 136], which makes it very promising for the serial message-transfer decoder. The downside of this approach, however, is that the message update rules of finite-alphabet decoders usually cannot be described using fast and area-efficient standard arithmetic circuits.

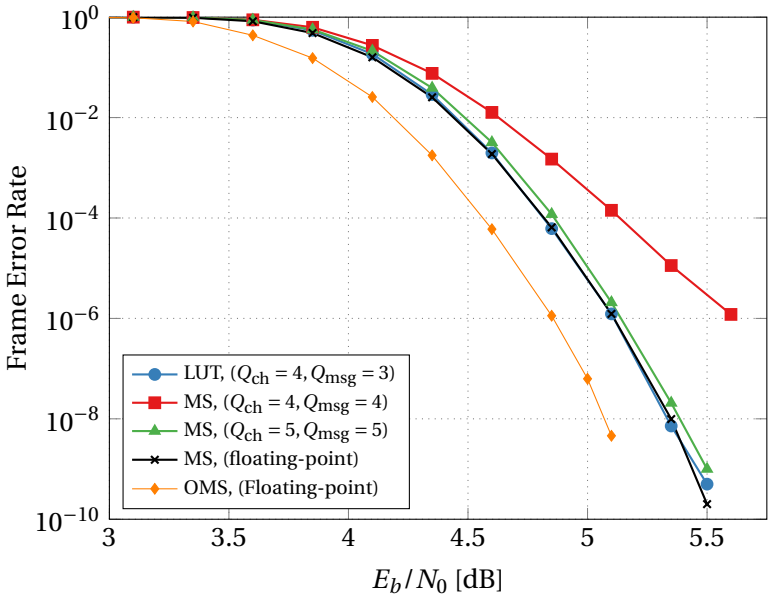


Figure 5.4: Frame error rate of the IEEE 802.3an LDPC code under floating-point MS decoding, fixed-point MS decoding with different bit-widths, LUT based decoding, and floating-point offset min-sum (OMS) decoding (offset=0.5) as reference, all with $I = 5$ decoding iterations.

In this section, we will review the basic idea and our design method for this new type of decoders and then show how the bit-width reduction technique of [77, 136] can be applied in order to increase the throughput and reduce the area of the serial message-transfer architecture.

5.3.1 Mutual Information Based Finite-Alphabet Decoder

In the approach of [77, 136], the standard MP decoding algorithm update rules are replaced by custom update rules that can be implemented as simple look-up tables (LUTs). These LUTs take integer-valued input messages and produce a corresponding output message. Moreover, the input-output mapping that is represented by the LUTs is designed in a way that maximizes the mutual information between the LUT output messages and the codeword bit that these messages correspond to. We note that a similar approach was also used in [150], but the corresponding hardware implementation would have a much higher hardware complexity than the method of [77, 136]. This happens because, contrary to [150], in [77, 136] LUTs are only used for the VNs while the CNs use the standard min-sum update rule.

5.3.2 Error-Correcting Performance and Bit-Width Reduction

In Fig. 5.4, we compare the performance of the IEEE 802.3an LDPC code under floating-point MS decoding, fixed-point MS decoding (with $Q_{ch} = Q_{msg} \in \{4, 5\}$), and LUT-based decoding

(with $Q_{\text{ch}} = 4$ and $Q_{\text{msg}} = 3$) when performing $I = 5$ decoding iterations with a flooding schedule. We also show the performance of a floating-point offset min-sum (OMS) decoder as a reference.[1] We observe that the fixed-point decoder with $Q_{\text{ch}} = Q_{\text{msg}} = 5$ has almost the same performance as the floating-point decoder, while the fixed-point decoder with $Q_{\text{ch}} = Q_{\text{msg}} = 4$ shows a significant loss with respect to the floating-point implementation. Thus, a standard MS decoder would need to use at least $Q_{\text{ch}} = Q_{\text{msg}} = 5$ quantization bits. The LUT-based decoder, however, can match the performance of the floating-point decoder with only $Q_{\text{ch}} = 4$ channel quantization bits and $Q_{\text{msg}} = 3$ message quantization bits.[2] Additionally, with the above quantization bit choices, no noticeable error floor has been observed for both the MS decoder and LUT-based decoder when 10^{10} frames have been transmitted (which corresponds to a BER $\approx 10^{-12}$ with the current block length). We note that, for the LUT-based decoder, the performance in the error floor region can be traded with the performance in the waterfall region by an appropriate choice of the design SNR for the LUTs [136].

5.3.3 LUT-Based Decoder Hardware Architecture

The LUT-based serial message-transfer decoder hardware architecture is very similar to the MS decoder architecture, described in Section 5.2. However, the LUT-based decoder can take advantage of the significantly fewer message bits that need to be transferred from one decoding stage to the next. This reduction reduces the number of CLK_F cycles per iteration, which in turn increases the throughput of the decoder according to (5.10) provided that the CN/VN logic is sufficiently fast. Moreover, the size of the buffers needed for the S/P and P/S conversions is also reduced significantly, which directly reduces the memory complexity of the decoder (see (5.8)).

On the negative side, we remark that the VN units for each variable node stage (decoder iteration) of the LUT-based decoder are different, which slightly complicates the hierarchical physical implementation as we will see later. Furthermore, since $Q_{\text{ch}} > Q_{\text{msg}}$, we now need to transfer the channel LLRs with multiple (two) bits per cycle to avoid the need to artificially limit the number of CLK_F cycles per iteration to Q_{ch} rather than to the smaller Q_{msg}. To reflect this modification, we redefine (5.10) as $Q_{\text{max}} = \max(Q_{\text{msg}}, \lceil \frac{Q_{\text{ch}}}{2} \rceil)$. While this partially parallel transfer of the channel LLRs impacts routing congestion, we note that the overhead is negligible since the number of channel LLRs is small compared to the total number of messages.

[1] The reference simulation was obtained and matched with our simulation by using the open-source simulator provided by: Adrien Cassagne; Romain Tajan; Mathieu Léonardon; Baptiste Petit; Guillaume Delbergue; Thibaud Tonnellier; Camille Leroux; Olivier Hartmann, "AFF3CT: A Fast Forward Error Correction Tool," 2016. [Online]. Available: https://doi.org/10.5281/zenodo.167837

[2] We note that reducing Q_{ch} further results in a non-negligible loss with respect to the floating-point decoder.

5.4 Implementation

Despite the use of a serial message-transfer and the low bit-width approximation of the messages, the physical implementation of the decoders proposed in the previous sections requires special scrutiny since the number of global wires is still significant and the overall area is particularly large. Therefore, in this section, we propose and describe a *pseudo-hierarchical* design methodology to implement the serial message-transfer architecture.

5.4.1 Physical Design

Due to the large number of identical blocks in the decoder architecture, a *bottom-up* flow is expected to provide the best results. The CN, VN, and DN units are first placed and routed individually to build hard macros,[3] and their timing and physical information are extracted. These macros are then instantiated as large cells in the decoder top level. We propose to treat the macros as *custom standard-cells* with identical height to be able to perform the placement using the standard-cell placement engine, rather than the less capable macro placement engine of the backend tool, since in our case the number of hard macro instances is extremely large and the interconnect pattern is complex and highly irregular.

Fig. 5.5 illustrates the proposed physical floorplan for the decoders with the unrolled architecture. In this floorplan, the CN and VN macros within each stage are constrained to be placed into dedicated regions (placement regions in Fig. 5.5a). This measure enforces the high-level structure of systolic array pipeline, but it also leaves freedom to the placement tool to choose the location for the macros in each stage to minimize routing congestion between stages. Note that the linear floorplan has also the advantage of being scalable in the number of iterations since little interaction or interference exists between stages. Furthermore, the CN and VN macros are placed in dedicated rows while the area between these rows is left for repeaters and for the register standard-cells for the channel LLRs in the check node stages, as shown in Fig. 5.5b. We note that the proposed floorplan and the encapsulation of the VN and CN macros as large standard-cells exploit the automated algorithm to optimize both custom and conventional standard-cells placement in order to alleviate the significant routing congestion.

5.4.2 Timing and Area Optimization Flow

Although the synthesis results can give an approximate evaluation for timing and area of the physical implementation, several iterations with different constraints are required to reach an optimal layout. To this end, we propose the methodology illustrated in the flowchart of Fig. 5.6 to effectively implement the serial message-transfer architecture. The main idea behind this methodology is that three main factors directly contribute to the decoder throughput and

[3] Note that for the LUT-based decoder there are different macros for each variable node stage as apposed to the MS decoder.

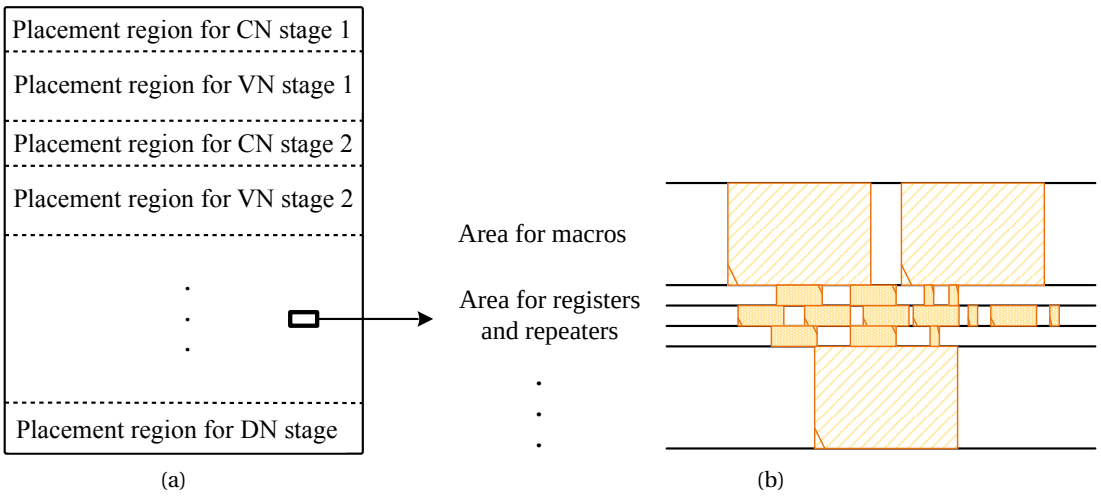


Figure 5.5: The physical floorplan for serial message-transfer architecture, (a) high level overview of the floorplan with dedicated placement regions for each decoder stage; and (b) zoomed in overview showing rows structure for custom macros (large colored blocks) and conventional standard-cells (small colored blocks) placement.

also indirectly to the decoder area, as discussed in Section 5.2 and specifically summarized in (5.10). Our goal is to maximize the throughput with minimum area.

We define the timing constraint applied to $\mathrm{CLK_S}$ as $T_{\mathrm{CSTR,CLK_S}}$, and the timing constrain applied to $\mathrm{CLK_F}$ as $T_{\mathrm{CSTR,CLK_F}}$. The first step is to place and route the CN/VN macros based on $T_{\mathrm{CSTR,CLK_S}}$. This step is followed by the implementation of the decoder using $T_{\mathrm{CSTR,CLK_F}}$. (The initial constraints for the backend are thereby extracted from synthesis timing results.) The fully placed and routed design can provide an accurate routing delay, which will be used to update $T_{\mathrm{CSTR,CLK_F}}$ and then $T_{\mathrm{CSTR,CLK_S}}$ according to (5.10). The updated $T_{\mathrm{CSTR,CLK_S}}$ will be used to re-implement the CN and VN macros within the minimum achievable area.

We note that for a long LDPC code with a large area and long routing delay (such as the one of the IEEE 802.3an standard), the first implementation starts with $T_{\mathrm{CSTR,CLK_S}} < Q_{\max} T_{\mathrm{CSTR,CLK_F}}$. After obtaining a realistic value for the $\mathrm{CLK_F}$ period (and hence for $T_{\mathrm{CSTR,CLK_F}}$) at the end of the implementation, the $T_{\mathrm{CSTR,CLK_S}}$ will be updated to a larger value to approach $T_{\mathrm{CSTR,CLK_S}} \approx Q_{\max} T_{\mathrm{CSTR,CLK_F}}$. Consequently, the CN and VN macro area and thus the decoder area will shrink in the second iteration, which result in larger achievable $\mathrm{CLK_F}$ frequency and hence smaller $T_{\mathrm{CSTR,CLK_F}}$ and $T_{\mathrm{CSTR,CLK_S}}$. The feedback loop will reach the optimum point after a few iterations.

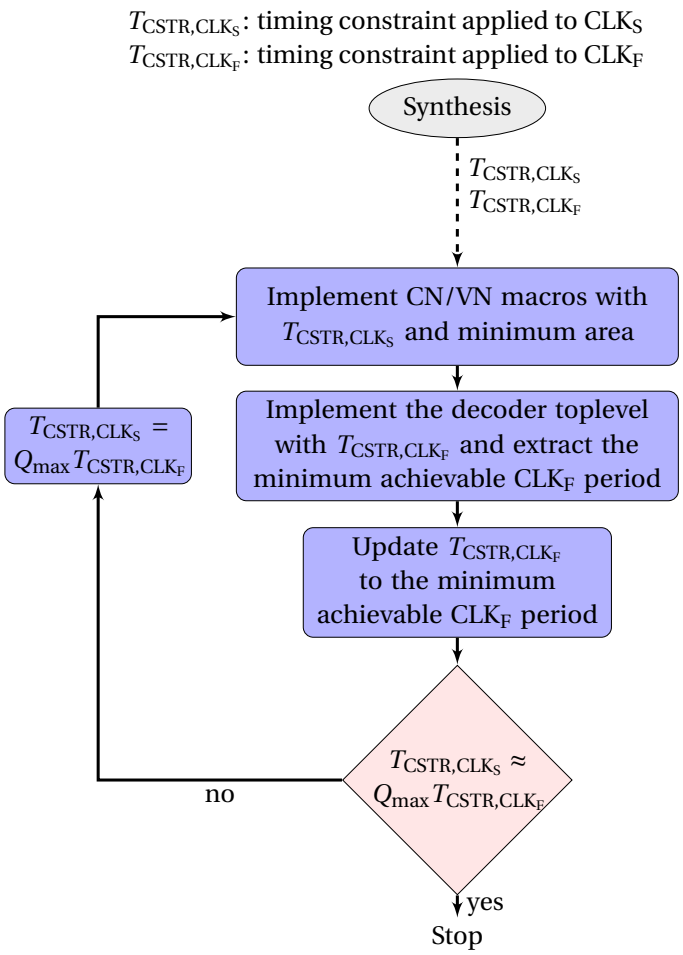
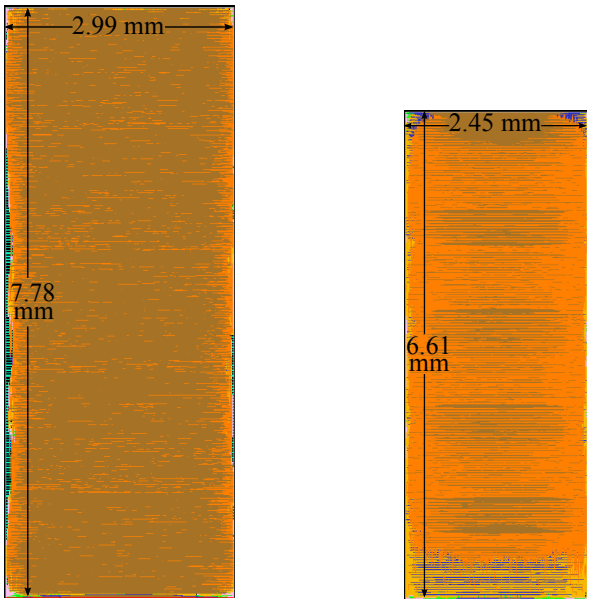


Figure 5.6: The proposed flowchart to optimize timing and area for the serial message-transfer architecture.

5.5 Results and Discussions

To study the impact of the serial message-transfer architecture and the finite-alphabet decoding scheme, we have implemented the proposed architecture by employing the methodology explained in Section 5.4 and we analyzed the results for both MS and LUT-based decoding. We used the parity check matrix of the LDPC code defined in the IEEE 802.3an standard [147], i.e., a $(2048, 1723)$ LDPC code of $R = 0.8413$ with $d_v = 6$ and $d_c = 32$. We used $I = 5$ for both decoders and $Q_{msg} = Q_{ch} = 5$ for the MS decoder and $Q_{msg} = 3$ and $Q_{ch} = 4$ for the LUT-based decoder to achieve the same error-correction performance, as described in Section 5.3. The decoders were synthesized from a VHDL description using Synopsys Design Compiler and placed and routed using Cadence Encounter Digital Implementation. The layouts are shown in Fig. 5.7. The results are reported for a 28 nm FD-SOI library under typical operating conditions ($V_{DD} = 1$ V, $T = 25^\circ$ C).



(a) (b)

Figure 5.7: Layouts for (a) the MS decoder and (b) the LUT-based decoder.

5.5.1 Delay Analysis

In the serial message-transfer architecture, the critical path and, hence, the maximum decoding frequency are defined by (5.10). To investigate the impact of serially transferring the messages on the decoder throughput, we consider the delay of the following register-to-register critical paths for both the MS and LUT-based decoder.

5.5.1.1 CN Critical Path

The CN critical path ($T_{\mathrm{CP,CN}}$) is the path from the S/P memory registers to the P/S shift register within the CN unit. For both decoders, this path is essentially comprised of the logic cells for a sorter tree with a depth of four.

5.5.1.2 VN Critical Path

The VN critical path ($T_{\mathrm{CP,VN}}$) is the path from S/P memory registers to P/S shift register within the VN unit. This path is dominated by an adder tree for the MS decoder and an LUT tree for the LUT-based decoder.

5.5.1.3 Routing Critical Path

The routing critical path ($T_{\mathrm{CP,route}}$) comprises mainly the interconnect wires (and buffers) that connect the CN/VN unit S/P shift register to the VN/CN unit P/S shift register.

Table 5.1: Critical path delays for MS and LUT-based decoder.

Path	MS decoder	LUT-based decoder
CN [ns]	2.38	1.42
VN [ns]	0.96	1.24
Routing [ns]	1.51	1.16

Table 5.1 summarizes the critical path delays of the CN/VN and the routing path. Together with (5.10), the values in the table dictate the maximum achievable frequency for CLK_S and CLK_F, respectively, for both of the decoders with the proposed serial message-transfer architecture. We note that the critical paths are reported after the timing and area constraints for the CN/VN macros and for the decoder toplevel are jointly optimized according to the flow shown in Fig. 5.6. According to (5.10), we observe that in both decoders the message transfer limits the slow clock CLK_S to a period of $5 \times 1.51\,\text{ns} = 7.55\,\text{ns}$ and $3 \times 1.16\,\text{ns} = 3.48\,\text{ns}$ for the MS and the LUT-based decoder, respectively, where 1.51 ns and 1.16 ns are the corresponding minimum CLK_F periods. Consequently, in our flow, the VN and CN units end up as optimized for minimum area only with relaxed and easy to meet timing constraints.

5.5.2 Area Analysis

Fig. 5.8 illustrates the area distribution among the various components after the layout. The area utilization is approximately 67% for both decoders. While almost 62% of the layout is filled with CN/VN macros and registers, the clock tree and routing buffers occupy around 5%. Furthermore, we see a 44% difference in total area between the decoders due to the fact that the total area for CN and VN macros is $14.12\,\text{mm}^2$ in the MS decoder, as opposed to only $9.56\,\text{mm}^2$ in the LUT-based decoder.

To understand this fact, we list the area of each CN/VN macro in Table 5.2. According to this table, the finite-alphabet message passing algorithm leads to significantly smaller CN processors because of two important factors: first, the bit-width reduction of the messages directly affects the data-path area, and second, the quantized messages in the LUT-based decoder are processed directly in the sorter tree of the CNs without the need to compute their absolute values. However, VN processors are less area-efficient in the LUT-based decoder in comparison with the ones of the MS decoder. This is caused by the fact that the LUT-based computations are, in general, less area-efficient than the conventional arithmetic based update rules. Thus, the logic area of the VN in the LUT-based decoder is larger, even though their input/output bit-width is smaller. Another contributing factor in the Table 5.2 is the register area, which is defined by the number of S/P and P/S registers. For those, the 40% reduction of bit-width in the LUT-based decoder is directly noticeable in the register area for both CN and VN units. Altogether, the CN and VN macros in the LUT-based decoder are 58% and 14% smaller, respectively, compared to those of the MS decoder.

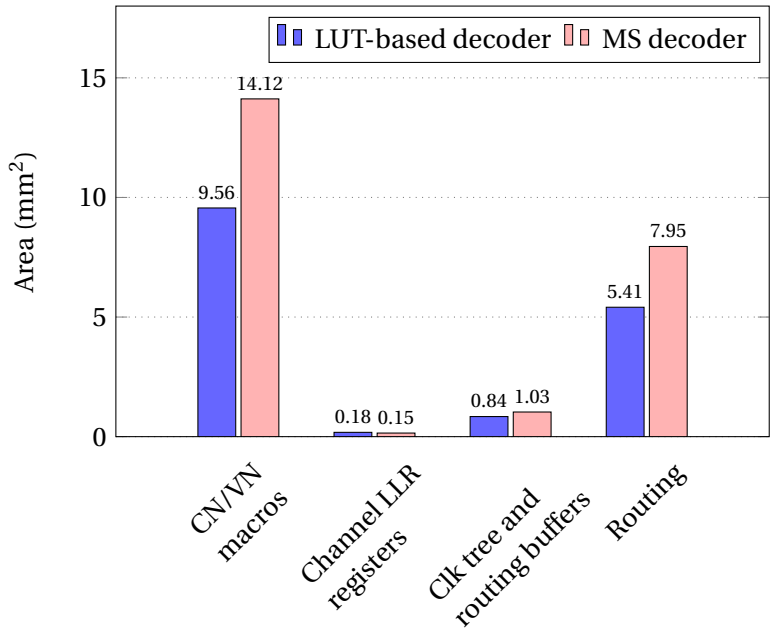


Figure 5.8: Detailed area results for the LUT-based and MS decoder with total area of 16.2 mm^2 and 23.3 mm^2, respectively.

5.5.3 Power Analysis

The energy which is consumed by each decoder is proportional to the capacitance, which in turn is related to the decoder area. Also, the number of required $\mathrm{CLK_F}$ cycles for the serial message-transfer to decode one codeword, which is inversely proportional to the decoding throughput at a constant frequency, directly contributes to the consumed energy for each decoded bit. Therefore, we analyze both the total power and the energy efficiency of the decoders using post-layout vector-based power analysis.[4] The results are reported in Table 5.3. We note that the total powers are calculated at $f_{\max}$ for both decoders. Also, for comparison purpose, we have calculated the total powers at a constant $\mathrm{CLK_F}$ frequency, here $\min(f_{\max,\mathrm{MS}}, f_{\max,\mathrm{LUT}}) = 662$ MHz, for both decoders and note them in the Table 5.3. According to this table, the total power consumption of the LUT-based decoder is 16.2% smaller than that of the MS decoder. Furthermore, by considering the fact that the LUT-based decoder has 66.7% higher throughput than the MS decoder at a similar $\mathrm{CLK_F}$ frequency, the energy efficiency of the LUT-based decoder is almost 2 times better in comparison with the MS decoder.

[4] We first extract the parasitic information of both the hard macros and the top level from the placement and routing tool and then read and link them using a power computation tool to generate the complete parasitic information.

Table 5.2: Detailed area for CN/VN unit*.

Component	MS decoder	LUT-based decoder
CN unit logic [μm^2]	1578	485
CN unit register [μm^2]	1695	971
CN macro [μm^2]	**3607**	**1510**
VN unit logic [μm^2]	315	403†
VN unit register [μm^2]	381	235†
VN macro [μm^2]	**755**	**646**†

* *Logic and register areas are obtained form synthesis, and macro areas are the final post-layout results.*
† *Although the VNs are different for each stage of the LUT-based decoder, their areas are similar and the result for one of them is reported here.*

Table 5.3: Power and energy efficiency comparison for MS and LUT-based decoder.

	MS decoder	LUT-based decoder
Total power @ $f_{\max}$ [mW]	12248	13350
Total power @ 662 MHz [mW]	12248	10257
Leakage power [mW]	7.44	5.27
Energy efficiency [pJ/bit]	45.2	22.7

5.5.4 Summary and Comparison to the State-of-the-Art

The final post-layout results for our MS and LUT-based decoders and also for some other recently implemented decoders are summarized in Table 5.4. Our LUT-based decoder runs at a maximum CLK_F frequency of $f_{\max,LUT} = 862$ MHz and delivers a sustained throughput of 588 Gbps, while it occupies 16.2 mm^2 area and dissipates 22.7 pJ/bit. Compared to the MS decoder, the LUT-based decoder is 1.4× smaller, 2.2× faster, and thus 3.1× more area efficient. It also has 16.2% lower power dissipation and 2× better energy efficiency, when the decoding throughout is taken into account.

The work in [140] is the only other unrolled full-parallel decoder in literature, but it is designed for the IEEE 802.11ad [133] code, which has a shorter block length and smaller node degrees ($d_v = 3$ and $d_c = 6$ as opposed to $d_v = 6$ and $d_c = 32$ for the code used in the design reported in this chapter). The work of [76], [143, 146, 154], and [153] are for the same IEEE 802.3an code considered in this chapter, but with partial-parallel and full-parallel architectures. The proposed LUT-based decoder has more than an order of magnitude higher throughput compared to [143] and [146], and three times higher throughput compared to [154], while the maximum throughput of the proposed decoder is maintained for all SNR scenarios as it does not require early termination to achieve a high throughput. The area efficiency of the proposed unrolled full-parallel architecture, however, is inferior to the one of the decoders in [143, 146] and [153] with full-parallel architecture due to the repeated routing overhead between the decoder stages in our design.

Table 5.4: Implementation results for MS and LUT-based decoder and comparison with other works.

	MS decoder	LUT-based decoder	[140]	[76]	[143]	[146]	[153]	[154]
Process technology	28 nm FD-SOI		65 nm CMOS	65 nm CMOS low-power	65 nm CMOS	90 nm CMOS	90 nm CMOS	90 nm CMOS
Supply voltage [V]	1.0		1.2	1.2	1.3	0.9	1.2	1.0
LDPC code	(2048,1723)		(672, 546)	(2048,1723)	(2048,1723)	(2048,1723)	(2048,1723)	(2048,1723)
Node degree (d_v, d_c)	(6, 32)		(3, 6)	(6, 32)	(6, 32)	(6, 32)	(6, 32)	(6, 32)
Algorithm	min-sum	finite-alphabet	min-sum	offset min-sum with post processor	split-row	normalized probablistic min-sum	reduced-complexity min-sum	delayed stochastic
$I_{\max}$	5		9	8	11	9	30	–
Quantization bits	5	**3**	4	4	5	4	6	5
E_b/N_0 @ BER= 10^{-7} [dB]	4.97	4.95	–	4.25	4.55	4.4	4.32	4.7
Architecture	unrolled full-parallel		unrolled full-parallel	partial-parallel	full-parallel	full-parallel	full-parallel	full-parallel
Core area [mm^2]	23.3	16.2	12.9	5.05	4.84	9.6	3.84	3.93
Area utilization [%]	66.4	65.9	76	84.5	97	91	–	93
Max. frequency ($f_{\max}$) [MHz]	662	862	257	700	195	199.6	226	750
Latency [ns]	151	69.6	105	137	56.4	45.09	–	800
Throughput @ $I_{\max}$ [Gbps]	271	**588**	160.8	13.3	36.3	45.42	12.8	172.4*
Power @ $f_{\max}$ [mW]	12248	13350	5360	2800	1359	1110	1040	–
Area eff. [Gbps/mm^2]	11.6	36.3	12.5	2.63	7.5	4.73	3.34	43.86
Energy per bit @ $I_{\max}$ [pJ/bit]	45.2	22.7	33.3	210.5	37.4	24.44	81.2	–
Scaled area eff.† [Gbps/mm^2]	11.6	36.3	156.4	32.9	93.8	157.1	110.9	1456.8
Scaled energy per bit‡ [pJ/bit]	45.2	22.7	10	63	9.5	9.4	17.5	–

* *Maximum throughput @ $E_b/N_0 = 5.5$ dB (Note that throughput @ I_{max} is not reported in the original paper)*

† *Scaling is done by S^3 where S is the relative dimension to 28 (Note that this is very rough and optimistic since it does not apply to the interconnects)*

‡ *Scaling is done by $1/SU^2$ where U is the relative voltage to 1.0*

5.6 Conclusion

An ultra high throughput LDPC decoder with a serial message-transfer architecture and based on non-uniform quantization of messages, inspired by static approximate computing techniques, was proposed to achieve the highest decoding throughput in literature. The proposed decoder architecture is an unrolled full-parallel architecture with serialized messages for CN/VN units, which was enabled by employing S/P and P/S shift registers at the inputs and outputs of each unit. The proposed quantized message passing algorithm replaces conventional MS, resulting in 40% reduction in message bit-width without any performance penalty. This algorithm was implemented by using generic LUTs instead of adders for VNs while the CNs remained unchanged compared to MS decoding. Placement and routing results in 28 nm FD-SOI show that the LUT-based serial message-transfer decoder delivers 0.588 Tbps throughput and is 3.1 times more area efficient and 2 times more energy efficient in comparison with the MS decoder with serial message-transfer architecture.

6 Conclusions and Outlook

The gap created by diminishing returns from technology scaling on the one hand, and expanding computing demand on the other hand, has led to a quest for a new source of efficiency in computing. Approximate computing is a paradigm that can serve as a promising solution to optimize computing platforms by adding a new dimension to the design space. Throughout this thesis, we identified two groups for approximate computing techniques, namely static and dynamic, and we proposed multiple algorithmic- and architectural-level techniques within these groups for the design of approximate and efficient hardware. More specifically, we examined the effect of unreliable memory components on the output quality of multiple applications. Further, we proposed a statistical framework for run-time adaptation of the quality-performance trade-off for an iterative algorithm. Finally, we exploited algorithm and architectural techniques at design time to optimize arithmetic units of a decoder in a communication system.

In the remainder of this concluding chapter, we will summarize the results of each chapter and we will describe some interesting open problems and future research directions that we have identified for each of the topics that were investigated in this thesis.

Chapter 2: Approximate Computing with Unreliable Memories by Restoring the Beauty of Randomness

A design-for-test methodology is proposed that allows to drop the requirement of 100% reliable operation for dies with defects in their memory elements, as the dominant part of many modern SoCs and the point-of-first-failure in advanced nano-meter technologies. The methodology facilitates the test procedure for faulty dies to a great-extent by equalizing the quality across dies and thus removing the need for a complex parametric test for all the

dies. The results for the proposed method were demonstrated and evaluated with practical image processing benchmarks for an embedded system with faulty memories. The key idea underlying the proposition was to add a small additional circuit that restores the beauty of random faults that are independent of the manufacturing outcome of the memory. This additional circuit creates different logical memories with the same physical fault map across the algorithm iterations.

Chapter 3: Practical Approximate Channel Decoders with Unreliable Memories

An approximate LDPC decoder in a 28 nm fully depleted silicon on insulator (FD-SOI) technology, ErgoDEC, was proposed, which was used to verify the assumption that errors are statically determined by the manufacturing outcome. The chip employed gain-cell (GC) embedded dynamic random-access memorys (DRAMs) as the faulty memory while the operating frequency was used as a knob to tune the fault ratio during the decoding operation. Beside serving the verification of the fault model, ErgoDEC demonstrates novel approaches to improve the performance and enable the usage of approximate memories, including the randomization introduced in Chapter 2. To this end, asymmetric logic-level distribution in the data in combination with an asymmetric reliability in the memory are used to minimize the memory faults impacts, and memory faults are randomized to enable the ergodic quality distribution and independent decoding attempts through creating different logical memories over time. As a result, the time-average behavior of each chip matched the ensemble-average across the population of chips.

Chapter 4: DVFS Based Power Managment for LDPC Decoders with Early Termination

A power management framework to control the energy-quality trade-off of an LDPC decoder was proposed by exploiting run-time variations in the LDPC algorithm. The framework sets the iteration limit on a per-codeword basis and configures a dynamic voltage and frequency scaling (DVFS) controller for the corresponding appropriate operation mode. The iteration limits are selected according to a conditional probabilities table of the decoding iterations, which is calculated offline using Monte-Carlo (MC) simulation. The framework can deliver variable power reductions for different output quality degradation levels. The results showed that a noticeable power reduction is achieved while a minimum quality is ensured.

Chapter 5: Toward Energy and Area Optimization of High-Throughput LDPC Decoders by Exploiting Quantized Message Passing

An ultra high throughput LDPC decoder with a serial message-transfer architecture and extremely optimized processing nodes based on non-uniform quantization of the decoding messages, inspired by approximate computing techniques, was proposed and the performance was evaluated for the challenging (2048, 1723) LDPC decoder. The proposed decoder was based on a min-LUT decoding algorithm and an unrolled full-parallel architecture with serial interconnect between processing nodes. The decoding algorithm was based on a quantized message passing algorithm, which replaces conventional min-sum decoding by using generic LUTs instead of standard arithmetic units for the variable nodes (VNs) while the check nodes (CNs) remains unchanged, resulting in 40% reduction in messages bit-width. Placement and routing results for the min-LUT decoder in 28 nm FDSOI showed 3.1 times more area efficiency and 2 times more energy efficiency in comparison to the min-sum decoder while achieving the best-in-class throughput.

Outlook

The central idea in the work of Chapter 2 and Chapter 3 attributes to the non-ergodic quality distribution of the population of dies and how to equalize such a distribution across the dies. As a result of this equalization, the quality of superior dies was degraded and the quality of inferior dies was improved while both of the qualities approached the die ensemble-average quality. The fault randomization succeeds to transform the quality of each die across the quality distribution of the population by shuffling the memory faults. In fact, an ideal randomization is capable of realizing any quality transformation from a single die. Even though the original purpose of the randomization was to enable the ensemble-average quality for each die in the population, it can be further exploited to realize the best quality across the quality distribution for each die in the population. The naive approach to enable this idea is to try all the possible realizations of the randomization for the input data-set and select the one with the best quality. However, this approach can heavily be improved and a more smart methodology can be used to *learn* the best quality across the quality distribution.

For the work of Chapter 4, LDPC decoders are used that are a great example of iterative algorithms where incrementing the number of iterations improves the output quality but increases the energy consumption. The proposed framework in this chapter delivers a second-order energy reduction for such an algorithm. The general idea behind this framework is the early prediction of the required iteration, which comes with a probability of miss prediction for the case of LDPC decoder, as illustrated in this chapter. However, this idea can be applied to other iterative algorithms in the domain of signal processing systems. The critical remark is to find an upper bound for the quality degradation or to ensure a minimum quality. This

minimum quality may be calculated analytically or using MC simulation, as provided in this chapter.

The work of Chapter 5 is based on an unrolled full-parallel architecture. Such architectures enable ultra-high throughput implementation of LDPC decoders by exploiting another degree of parallelism in such decoders. While the throughput of such architecture improves, the energy efficiency degrades, as compared to the full-parallel architecture. The degradation in energy efficiency occurs due to the fact that early termination of the decoding iterations cannot be trivially employed as all the iterations are mapped to distinct hardware units. However, early termination can be implemented in a different way for such an architecture by applying clock-gating to the pipeline registers that correspond to the extra decoding iterations. This practice would reduce the energy expenditure for most of the codewords and thus would improve the energy efficiency.

The unrolled LDPC decoder architecture is a perfect candidate for the constraints of the min-LUT decoding algorithm, however, it is an architecture that is quite specialized to ultra-high throughput applications. Thus, it would be useful to identify other kinds of LDPC decoder architecture, which could also benefit from the reduced quantization bit-width offered by the min-LUT decoding algorithm, as a flavor of static approximate computing. Finally, the main idea behind this algorithm could be applied to other applications, such as deep Neural Networks, to reduce the complexity of the processing nodes.

Bibliography

[1] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No. 03CH37451)*. IEEE, 2003, pp. 338–342.

[3] K. A. Bowman, S. G. Duvall, and J. D. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of solid-state circuits*, vol. 37, no. 2, pp. 183–190, 2002.

[4] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao, "The impact of NBTI on the performance of combinational and sequential circuits," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 364–369.

[5] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718–1751, 2010.

[6] A. Rahimi, L. Benini, and R. K. Gupta, "Variability mitigation in nanometer CMOS integrated systems: A survey of techniques from circuits to software," *Proceedings of the IEEE*, vol. 104, no. 7, pp. 1410–1448, 2016.

[7] S. Bhunia and S. Mukhopadhyay, *Low-power variation-tolerant design in nanometer silicon*. Springer, 2010.

[8] S. Chandra, A. Raghunathan, and S. Dey, "Variation-aware voltage level selection," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 5, pp. 925–936, 2011.

[9] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical analysis and optimization for VLSI: Timing and power*. Springer Science & Business Media, 2006.

[10] A. Asenov, A. Cathignol, B. Cheng, K. McKenna, A. Brown, A. Shluger, D. Chanemougame, K. Rochereau, and G. Ghibaudo, "Origin of the asymmetry in the magnitude of the

statistical variability of n-and p-channel poly-Si gate bulk MOSFETs," *IEEE Electron Device Letters*, vol. 29, no. 8, pp. 913–915, 2008.

[11] S. Reda and S. R. Nassif, "Analyzing the impact of process variations on parametric measurements: Novel models and applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 375–380.

[12] M. Miranda, B. Dierickx, P. Zuber, P. Dobrovoln, F. Kutscherauer, P. Roussel, and P. Poliakov, "Variability aware modeling of SoCs: From device variations to manufactured system yield," in *2009 10th International Symposium on Quality Electronic Design*. IEEE, 2009, pp. 547–553.

[13] N. Aymerich, A. Asenov, A. Brown, R. Canal, B. Cheng, J. Figueras, A. González, E. Herrero, S. Markov, M. Miranda *et al.*, "New reliability mechanisms in memory design for sub-22nm technologies," in *2011 IEEE 17th International On-Line Testing Symposium*. IEEE, 2011, pp. 111–114.

[14] M. Cho, S. T. Kim, C. Tokunaga, C. Augustine, J. P. Kulkarni, K. Ravichandran, J. W. Tschanz, M. M. Khellah, and V. De, "Postsilicon voltage guard-band reduction in a 22 nm graphics execution core using adaptive voltage scaling and dynamic power gating," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 50–63, 2016.

[15] D. Mishagli, E. Blokhina, T. J. Brazil, and S. Hollands, "Analytical approach to statistical logic cell delay analysis and its extension to a timing graph," in *The 2018 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Monterey, California, 15-16 March 2018*. ACM, 2018.

[16] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. ACM, 2005, pp. 2–7.

[17] J. Cong, H. Duwe, R. Kumar, and S. Li, "Better-than-worst-case design: Progress and opportunities," *Journal of computer science and technology*, vol. 29, no. 4, pp. 656–663, 2014.

[18] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, 2004.

[19] K. A. Bowman, J. W. Tschanz, S.-L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, 2010.

[20] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 120.

[21] I. J. Chang, D. Mohapatra, and K. Roy, "A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications," *IEEE transactions on circuits and systems for video technology*, vol. 21, no. 2, pp. 101–112, 2011.

[22] D. Mohapatra, G. Karakonstantis, and K. Roy, "Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, 2009, pp. 195–200.

[23] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[24] K. Palem and A. Lingamneni, "Ten years of building broken chips: The physics and engineering of inexact computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, pp. 1–23, 2013.

[25] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[26] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, "ERSA: Error resilient system architecture for probabilistic applications," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 1560–1565.

[27] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener, "Programming with relaxed synchronization," in *Proceedings of the 2012 ACM workshop on Relaxing synchronization for multicore and manycore scalability*, 2012, pp. 41–50.

[28] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate signal processing," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 15, no. 1-2, pp. 177–200, 1997.

[29] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.

[30] R. G. Lyons, *Understanding digital signal processing, 3/E*. Pearson Education India, 2004.

[31] R. Yates, "Fixed-point arithmetic: An introduction," *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[32] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[33] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: imprecise adders for low-power approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design*. IEEE, 2011, pp. 409–414.

[34] V. Camus, J. Schlachter, and C. Enz, "A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. Ieee, 2016, pp. 1–6.

[35] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Block-based carry speculative approximate adder for energy-efficient applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 1, pp. 137–141, 2019.

[36] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*. IEEE, 2011, pp. 346–351.

[37] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1435–1441, 2017.

[38] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[39] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and applications of approximate circuits by gate-level pruning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[40] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, "SRAM for error-tolerant applications with dynamic energy-quality management in 28 nm CMOS," *IEEE Journal of Solid-state circuits*, vol. 50, no. 5, pp. 1310–1323, 2015.

[41] R. Giterman, A. Fish, N. Geuli, E. Mentovich, A. Burg, and A. Teman, "An 800-MHz mixed-V_T 4T IFGC embedded DRAM in 28-nm CMOS bulk process for approximate storage applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2136–2148, 2018.

[42] M. Shoushtari *et al.*, "Exploiting partially-forgetful memories for approximate computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, March 2015.

[43] A. Di Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, "Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications," in *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 30–4.

[44] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[45] G. Karakonstantis, C. Roth, C. Benkeser, and A. Burg, "On the exploitation of the inherent error resilience of wireless systems under unreliable silicon," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 510–515.

[46] N. Zompakis, A. Papanikolaou, P. Raghavan, D. Soudris, and F. Catthoor, "Enabling efficient system configurations for dynamic wireless applications using system scenarios," *International journal of wireless information networks*, vol. 20, no. 2, pp. 140–156, 2013.

[47] R. A. Abdallah and N. R. Shanbhag, "Error-resilient low-power viterbi decoder architectures," *IEEE transactions on signal processing*, vol. 57, no. 12, pp. 4906–4917, 2009.

[48] M. May, M. Alles, and N. Wehn, "A case study in reliability-aware design: A resilient LDPC code decoder," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 456–461.

[49] Y. Liu, T. Zhang, and J. Hu, "Design of voltage overscaled low-power trellis decoders in presence of process variations," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 3, pp. 439–443, 2009.

[50] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 24, no. 12, pp. 1859–1880, 2005.

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

[52] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[53] A. Burg, S. Haene, M. Borgmann, D. Baum, T. Thaler, F. Carbognani, S. Zwicky, L. Barbero, C. Senning, P. Greisen *et al.*, "A 4-stream 802.11 n baseband transceiver in 0.13 μm CMOS," in *2009 Symposium on VLSI Circuits*. IEEE, 2009, pp. 282–283.

[54] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 340–352.

[55] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[56] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[57] P. A. Meinerzhagen, "Novel approaches toward area- and energy-efficient embedded memories," Ph.D. dissertation, EPFL, 2014.

[58] H. Kaeslin, *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*. Cambridge University Press, 2008.

[59] P. Meinerzhagen, A. Teman, R. Giterman, N. Edri, A. Burg, and A. Fish, *Gain-cell Embedded DRAMs for Low-power VLSI Systems-on-chip*. Springer, 2018.

[60] B. H. Calhoun and A. P. Chandrakasan, "A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation," *IEEE journal of solid-state circuits*, vol. 42, no. 3, pp. 680–688, 2007.

[61] P. Meinerzhagen, S. Y. Sherazi, A. Burg, and J. N. Rodrigues, "Benchmarking of standard-cell based memories in the Sub-VT domain in 65-nm CMOS technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 173–182, 2011.

[62] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1524–1537, 2014.

[63] Y. Leblebici and S.-M. Kang, *CMOS digital integrated circuits: analysis and design*. McGraw-Hill, 1996.

[64] A. Teman, P. Meinerzhagen, A. Burg, and A. Fish, "Review and classification of gain cell edram implementations," in *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*. IEEE, 2012, pp. 1–5.

[65] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Controlled placement of standard cell memory arrays for high density and low power in 28nm FD-SOI," in *The 20th Asia and South Pacific Design Automation Conference*. IEEE, 2015, pp. 81–86.

[66] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[67] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[68] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[69] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219–230, 1998.

[70] A. K. B. Stimming, "Hardware implementation aspects of polar decoders and ultra high-speed ldpc decoders," Ph.D. dissertation, EPFL, 2016.

[71] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," vol. 45, no. 2, pp. 399–431, 1999.

[72] M. P. Fossorier, M. Mihaljević, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," vol. 47, no. 5, pp. 673–680, 1999.

[73] C. Roth, A. Cevrero, C. Studer, Y. Leblebici, and A. Burg, "Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 1772–1775.

[74] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," vol. 37, no. 3, pp. 404–412, 2002.

[75] C. Roth, P. Meinerzhagen, C. Studer, and A. Burg, "A 15.8 pJ/bit/iter quasi-cyclic LDPC decoder for IEEE 802.11n in 90 nm CMOS," in *IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2010, pp. 1–4.

[76] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolić, "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," vol. 45, no. 4, pp. 843–855, 2010.

[77] A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled LDPC decoder based on quantized message passing," in *IEEE Int. Workshop on Signal Process. Syst. (SiPS)*, Oct 2015, pp. 1–6.

[78] J. Li, X. hu You, and J. Li, "Early stopping for LDPC decoding: convergence of mean magnitude (CMM)," *IEEE Communications Letters*, vol. 10, no. 9, pp. 667–669, Sept 2006.

[79] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-driven near-threshold SRAM design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 11, pp. 1590–1598, Nov 2010.

[80] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Approximate SRAMs with dynamic energy-quality management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2128–2141, June 2016.

[81] R. Giterman, A. Fish, N. Geuli, E. Mentovich, A. P. Burg, and A. Teman, "An 800 Mhz mixed-VT 4T gain-cell embedded DRAM in 28 nm CMOS bulk process for approximate computing applications," *IEEE European Solid State Circuits Conf. (ESSCIRC)*, pp. 308–311, 2017.

[82] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, "Bit error tolerance of a CIFAR-10 binarized convolutional neural network processor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[83] M. A. , "Intelligible test techniques to support error-tolerance," in *13th Asian Test Symposium*, Nov 2004, pp. 386–393.

[84] M. A. Breuer and H. Zhu, "An illustrated methodology for analysis of error tolerance," *IEEE Design Test of Computers*, vol. 25, no. 2, pp. 168–177, March 2008.

[85] R. Liu, H. Wu, Y. Pang, H. Qian, and S. Yu, "Experimental characterization of physical unclonable function based on 1 kb resistive random access memory arrays," *IEEE Electron Device Letters*, vol. 36, no. 12, pp. 1380–1383, 2015.

[86] D. Bortolotti, H. Mamaghanian, A. Bartolini, M. Ashouei, J. Stuijt, D. Atienza, P. Vandergheynst, and L. Benini, "Approximate compressed sensing: ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor," in *Int. Symp. Low Power Electronics and Design*. ACM, 2014, pp. 45–50.

[87] M. Widmer, A. Bonetti, and A. Burg, "FPGA-Based emulation of embedded DRAMs for statistical error resilience evaluation of approximate computing systems," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. ACM/IEEE, 2019, pp. 1–6.

[88] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. Beignè *et al.*, "Energy-efficient near-threshold parallel computing: The pulpv2 cluster," *Ieee Micro*, vol. 37, no. 5, pp. 20–31, 2017.

[89] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "μDMA: An autonomous i/o subsystem for iot end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2017, pp. 1–8.

[90] S. K. Venkata *et al.*, "SD-VBS: The san diego vision benchmark suite," in *2009 IEEE IISWC*, Oct 2009, pp. 55–64.

[91] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *2001 IEEE IWWC*, Dec 2001, pp. 3–14.

[92] A. Carter, "MNIST neural network in C," available at *https://github.com/AndrewCarterUK/mnist-neural-network-plain-c*.

[93] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[94] D. Boning and S. Nassif, "Models of process variations in device and interconnect," *Design of high performance microprocessor circuits*, p. 6, 2000.

[95] V. M. Van Santen, J. Martin-Martinez, H. Amrouch, M. M. Nafria, and J. Henkel, "Reliability in super-and near-threshold computing: A unified model of RTN, BTI, and PV," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 293–306, 2017.

[96] E. I. Vatajelu, H. Aziza, and C. Zambelli, "Nonvolatile memories: Present and future challenges," in *2014 9th International Design and Test Symposium (IDT)*. IEEE, 2014, pp. 61–66.

[97] F. Mühlbauer, L. Schröder, P. Skoncej, and M. Schölzel, "Handling manufacturing and aging faults with software-based techniques in tiny embedded systems," in *2017 18th IEEE Latin American Test Symposium (LATS)*. IEEE, 2017, pp. 1–6.

[98] M. Schölzel and P. Skoncej, "Software-based repair for memories in tiny embedded systems," in *2015 20th IEEE European Test Symposium (ETS)*. IEEE, 2015, pp. 1–2.

[99] P. K. Chundi, Y. Zhou, M. Kim, E. Kursun, and M. Seok, "Hotspot monitoring and temperature estimation with miniature on-chip temperature sensors," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[100] M. F. Reza, D. Zhao, H. Wu, and M. Bayoumi, "Hotspot-aware task-resource co-allocation for heterogeneous many-core networks-on-chip," *Computers & Electrical Engineering*, vol. 68, pp. 581–602, 2018.

[101] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici, "Dynamic thermal management in 3D multicore architectures," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 1410–1415.

[102] Y. S. Park, D. Blaauw, D. Sylvester, and Z. Zhang, "Low-power high-throughput LDPC decoder using non-refresh embedded DRAM," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 3, pp. 783–794, 2014.

[103] P. Meinerzhagen, A. Bonetti, G. Karakonstantis, C. Roth, F. Giirkaynak, and A. Burg, "Refresh-free dynamic standard-cell based memories: Application to a QC-LDPC decoder," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 1426–1429.

[104] W. Choi, G. Kang, and J. Park, "A refresh-less eDRAM macro with embedded voltage reference and selective read for an area and power efficient Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 10, pp. 2451–2462, 2015.

[105] C. Roth, C. Studer, G. Karakonstantis, and A. Burgi, "Statistical data correction for unreliable memories," in *2014 48th Asilomar Conference on Signals, Systems and Computers*. IEEE, 2014, pp. 1890–1894.

[106] L. R. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, 2011.

[107] S. S. T. Yazdi, H. Cho, and L. Dolecek, "Gallager B decoder on noisy hardware," *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1660–1673, 2013.

[108] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of LDPC codes under unreliable message storage," *IEEE Communications Letters*, vol. 18, no. 5, pp. 849–852, 2014.

[109] C. K. Ngassa, V. Savin, and D. Declercq, "Min-sum-based decoders running on noisy hardware," in *2013 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2013, pp. 1879–1884.

[110] P. Ivaniš and B. Vasić, "Error errore eicitur: A stochastic resonance paradigm for reliable storage of information on unreliable media," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3596–3608, 2016.

[111] J. Mu, A. Vosoughi, J. Andrade, A. Balatsoukas-Stimming, G. Karakonstantis, A. Burg, G. Falcao, V. Silva, and J. R. Cavallaro, "The impact of faulty memory bit cells on the decoding of spatially-coupled LDPC codes," in *2015 49th Asilomar Conference on Signals, Systems and Computers*. IEEE, 2015, pp. 1627–1631.

[112] A. Balatsoukas-Stimming and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2322–2332, 2017.

[113] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for LDPC decoding," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4076–4091, 2007.

[114] A. Balatsoukas-Stimming, N. Preyss, A. Cevrero, A. Burg, and C. Roth, "A parallelized layered QC-LDPC decoder for IEEE 802.11 ad," in *IEEE International New Circuits and Systems Conference (NEWCAS)*. IEEE, 2013, pp. 1–4.

[115] P. Meinerzhagen, C. Roth, and A. Burg, "Towards generic low-power area-efficient standard cell based memory architectures," in *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, 2010, pp. 129–132.

[116] R. Giterman, A. Bonetti, E. V. Bravo, T. Noy, A. Teman, and A. Burg, "Current-Based Data-Retention-Time characterization of Gain-Cell embedded DRAMs across the design and variations space," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.

[117] A. Teman, G. Karakonstantis, R. Giterman, P. Meinerzhagen, and A. Burg, "Energy versus data integrity trade-offs in embedded high-density logic compatible dynamic memories," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 489–494.

[118] S. Ganapathy, G. Karakonstantis, A. Teman, and A. Burg, "Mitigating the impact of faults in unreliable memories for error-resilient applications," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[119] C. Roth, C. Benkeser, C. Studer, G. Karakonstantis, and A. Burg, "Data mapping for unreliable memories," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2012, pp. 679–685.

[120] I. Lee, J. Kwon, J. Park, and J. Park, "Priority based error correction code (ECC) for the embedded SRAM memories in H. 264 system," *Journal of Signal Processing Systems*, vol. 73, no. 2, pp. 123–136, 2013.

[121] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, "13.8 A 32kb SRAM for error-free and error-tolerant applications with dynamic energy-quality management in 28nm CMOS," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 244–245.

[122] M. Pătraşcu and M. Thorup, "The power of simple tabulation hashing," *Journal of the ACM (JACM)*, vol. 59, no. 3, pp. 1–50, 2012.

[123] C. Studer, N. Preyss, C. Roth, and A. Burg, "Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes," in *2008 42nd Asilomar Conference on Signals, Systems and Computers*. IEEE, 2008, pp. 1137–1142.

[124] Q. Zhang, F. Yuan, R. Ye, and Q. Xu, "Approxit: An approximate computing framework for iterative methods," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[125] A. Roldao-Lopes, A. Shahzad, G. A. Constantinides, and E. C. Kerrigan, "More flops or more precision? accuracy parameterizable linear equation solvers for model predictive control," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE, 2009, pp. 209–216.

[126] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*. IEEE, 1990, pp. 238–239.

[127] T. Kuroda, K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda *et al.*, "Variable supply-voltage scheme for low-power high-speed CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, pp. 454–462, 1998.

[128] W. Wang and G. Choi, "Minimum-energy LDPC decoder for real-time mobile application," in *Proceeding on the Conference on Design, Automation and Test in Europe (DATE)*. EDA Consortium, 2007, pp. 343–348.

[129] W. Wang, G. Choi, and K. K. Gunnam, "Low-power VLSI design of LDPC decoder using DVFS for AWGN channels," in *International Conference on VLSI Design*. IEEE, 2009, pp. 51–56.

[130] E. Amador, R. Knopp, V. Rezard, and R. Pacalet, "Dynamic power management on LDPC decoders," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2010, pp. 416–421.

[131] X. Zhang, F. Cai, and C. R. Shi, "Low-power LDPC decoding based on iteration prediction," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2012, pp. 3041–3044.

[132] N. H. Weste and D. M. Harris, *CMOS VLSI design: a circuits and systems perspective*. Addison-Wesley/Pearson, 2011.

[133] "ISO/IEC/IEEE International Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band (adoption of IEEE Std 802.11ad-2012)," *ISO/IEC/IEEE 8802-11:2012/Amd.3:2014(E)*, pp. 1–634, March 2014.

[134] E. Lai, *Practical digital signal processing*. Elsevier, 2003.

[135] M. Gautschi, M. Schaffner, F. K. Gürkaynak, and L. Benini, "4.6 A 65nm CMOS 6.4-to-29.2 pJ/FLOP 0.8 V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 82–83.

[136] M. Meidlinger, A. Balatsoukas-Stimming, A. Burg, and G. Matz, "Quantized message passing for LDPC codes," in *Asilomar Conf. on Signals, Syst., and Comput. (ACSSC)*, Nov 2015, pp. 1606–1610.

[137] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm^2 10GBASE-T Ethernet LDPC decoder chip in 90 nm CMOS," in *IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2010, pp. 1–4.

[138] T.-C. Kuo and A. N. Willson Jr, "A flexible decoder IC for WiMAX QC-LDPC codes," in *IEEE Custom Integrated Circuits Conf. (CICC)*, 2008, pp. 527–530.

[139] N. Onizawa, T. Hanyu, and V. C. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," vol. 18, no. 3, pp. 482–489, 2010.

[140] P. Schlafer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *IEEE Int. Workshop on Signal Process. Syst. (SiPS)*, 2013, pp. 153–158.

[141] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," in *IEEE Custom Integrated Circuits Conf. (CICC)*, 2007, pp. 459–462.

[142] A. Darabiha, A. C. Carusone, and R. Kschischang, "Power reduction techniques for LDPC decoders," vol. 43, no. 8, pp. 1835–1845, 2008.

[143] T. Mohsenin, D. N. Truong, and B. M. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," vol. 57, no. 5, pp. 1048–1061, 2010.

[144] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," vol. 57, no. 9, pp. 2518–2523, 2009.

[145] K. Cushon, S. Hemati, C. Leroux, S. Mannor, and W. J. Gross, "High-throughput energy-efficient LDPC decoders using differential binary message passing," vol. 62, no. 3, pp. 619–631, 2014.

[146] C.-C. Cheng, J.-D. Yang, H.-C. Lee, C.-H. Yang, and Y.-L. Ueng, "A fully parallel LDPC decoder architecture using probabilistic min-sum algorithm for high-throughput applications," vol. 61, no. 9, pp. 2738–2746, 2014.

[147] "IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," IEEE Std. 802.3an, Sep. 2006.

[148] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders—part i: Decoding beyond belief propagation on the binary symmetric channel," vol. 61, no. 10, pp. 4033–4045, 2013.

[149] B. Kurkoski, K. Yamaguchi, and K. Kobayashi, "Noise thresholds for discrete LDPC decoding mappings," in *IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2008, pp. 1–5.

[150] F. J. C. Romero and B. M. Kurkoski, "Decoding LDPC codes with mutual information-maximizing lookup tables," in *IEEE Int. Symp. on Inf. Theory (ISIT)*, Jun. 2015, pp. 426–430.

[151] J. Lewandowsky and G. Bauch, "Information-optimum LDPC decoders based on the information bottleneck method," *IEEE Access*, vol. 6, pp. 4054–4071, 2018.

[152] J. Lewandowsky, M. Stark, and G. Bauch, "Optimum message mapping LDPC decoders derived from the sum-product algorithm," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[153] F. Angarita, J. Valls, V. Almenar, and V. Torres, "Reduced-complexity min-sum algorithm for decoding LDPC codes with low error-floor," vol. 61, no. 7, pp. 2150–2158, 2014.

[154] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of LDPC codes," vol. 59, no. 11, pp. 5617–5626, 2011.

List of Figures

List of Tables

Curriculum Vitae

Name:	**Reza GHANAATIAN JAHROMI**
Date of Birth:	13.05.1988
Nationality:	Iranian
Address:	EPFL, STI-IEL-TCL, Station 11
	CH-1015 Lausanne
E-Mail:	reza.ghanaatian@epfl.ch
	reza.ghanaatian@gmail.com

Education

02.2015 – 08.2020 **École Polytechnique Fédérale de Lausanne**, Lausanne (VD), CH
Ph.D Degree in Electrical Engineering

09.2010 – 08.2012 **Sharif University of Technology**, Tehran, IR
Master's Degree in Digital Systems

09.2006 – 08.2010 **Khaje Nasir Toosi University of Technology**, Tehran, IR
Bachelor's Degree in Electrical Engineering

Professional Experience

02.2015 – 08.2020 **École Polytechnique Fédérale de Lausanne**, Lausanne (VD), CH
Doctoral Assistant at Telecommunications Circuits Laboratory

07.2019 – 10.2019 **Swisscom AG**, Bern, CH
Technical Intern at Enterprise Architecture & Innovation

09.2012 – 06.2014 **Electronic Research Center, Sharif University of Technology**, Tehran, IR
FPGA-based system developer

09.2010 – 08.2012 **Sharif University of Technology**, Tehran, IR
Research Assistant at Advanced Integrated Circuit Design Laboratory

List of Publications

Journal Papers

R. Ghanaatian, M. Widmer, and A. Burg, "Design for Test with Unreliable Memories by Restoring the Beauty of Randomness", *journal publication submitted.*

R. Ghanaatian, R. Giterman, A. Bonetti, A. Balatsoukas-Stimming, and A. Burg, "ErgoDEC: A Practical Approximate LDPC Decoder in 28 nm FD-SOI with Unreliable Memories", *journal publication submitted.*

R. Ghanaatian, A. Balatsoukas-Stimming, TC. Müller, M. Meidlinger, G. Matz, and A. Burg, "A 588-Gb/s LDPC Decoder Based on Finite-Alphabet Message Passing", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 2018.

R. Ghanaatian, M. Shabany and M. H. Shoreh, "A High-Throughput VLSI Architecture for Real-Time Optical OFDM Systems With an Efficient Phase Equalizer", *IEEE Canadian Journal of Electrical and Computer Engineering*, 2014. *

Conference Papers

R. Ghanaatian, O. Afisiadis, M. Cotting, and A. Burg, "LoRa Digital Receiver Analysis and Implementation", *International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2019*, Brighton, UK. *

R. Ghanaatian, V. Jamali, A. Burg, and R. Schober, "Feedback-Aware Precoding in Millimeter Wave MIMO Systems", *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) 2019*, Istanbul, Turkey. *

R. Ghanaatian and A. Burg, "DVFS based power management for LDPC decoders with early termination", *IEEE International Workshop on Signal Processing Systems (SiPS) 2017*, Lorient, France.

R. Ghanaatian, P. N. Whatmough, J. Constantin, A. Teman and A. Burg, "A Low-Power Correlator for Wakeup Receivers with Algorithm Pruning through Early Termination", *IEEE International Symposium on Circuits and Systems (ISCAS) 2016*, Montreal, QC, Canada. *

A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled LDPC decoder based on quantized message passing", *IEEE Workshop on Signal Processing Systems (SiPS) 2015*, Hangzhou, China.

M. H. Shoreh, R. Ghanaatian and J. A. Salehi, "Channel Estimation and Iterative Equalization for Long-Haul Coherent Optical OFDM Communication Systems", *International Conference on Telecommunications (ConTEL) 2015*, Graz, Austria. *

R. Ghanaatian, M. Shabany and M. Sharifkhani, "An Efficient High-Throughput VLSI Architecture for a Synchronization Block Applied to Real-Time Optical OFDM Systems", *IEEE International Symposium on Circuits and Systems (ISCAS) 2014*, Melbourne VIC, Australia. *

* The contents of these publications do not form a part of this thesis.