Hardware implementation aspects of polar decoders and ultra high-speed LDPC decoders

THÈSE N^{O} 7297 (2016)

PRÉSENTÉE LE 28 OCTOBRE 2016

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE CIRCUITS POUR TÉLÉCOMMUNICATIONS
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Alexios Konstantinos BALATSOUKAS STIMMING

acceptée sur proposition du jury:

Prof. B. Rimoldi, président du jury
Prof. A. P. Burg, directeur de thèse
Prof. W. J. Gross, rapporteur
Prof. I. Tal, rapporteur
Prof. R. Urbanke, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2016

Acknowledgements

I would like to start by thanking my advisor, Prof. Andreas Burg, for his continuous guidance and support throughout the duration of my PhD studies. In particular, I would like to thank him for believing in my potential and agreeing to become my doctoral advisor, for encouraging and enabling me to attend scientific conferences all over the world, for initiating fruitful collaborations with external partners, for always providing meaningful ideas for future research directions, and for acting as a filter between myself and academic politics, thus enabling me to focus almost exclusively on my research.

Taking a small step back in time, I want to thank my undergraduate and MSc advisor, Prof. Athanasios P. Liavas (Department of Electronic and Computer Engineering, Technical University of Crete), who took me by the hand and provided me with all the necessary academic provisions that enabled me to embark on the journey of my PhD. I have nothing but the utmost respect for his personal and professional integrity and his highly systematic and responsible approach to both teaching and research.

I would also like to thank Prof. Bixio Rimoldi for acting as the president of my PhD jury, Prof. Rüdiger Urbanke for serving as an internal examiner, and Prof. Warren J. Gross (Integrated Systems for Information Processing Laboratory, McGill University) and Prof. Ido Tal (Department of Electrical Engineering, Technion - Israel Institute of Technology) for serving as external examiners. Special thanks also go to Prof. Zhengya Zhang (Department of Electrical Engineering and Computer Science, University of Michigan), Prof. Erdal Arıkan (Department of Electrical and Electronics Engineering, Bilkent University), and Prof. Amin Shokrollahi (Algorithmic Mathematics Laboratory, EPFL) for their willingness to serve as examiners for my thesis defense.

The administrative assistant of our laboratory, Ioanna Paniara, as well as the administrative assistants of my doctoral school, Cecilia Chapuis and Corinne Degott, were always very helpful. Many thanks go to them for smoothly taking care of all administrative issues.

Particular thanks go to all of my labmates for endless discussions of varying depth and seriousness on countless topics, both technical and non-technical, as well as lavish culinary feasts and adventurous excursions. They all certainly made these years one of the most memorable periods of my life. In alphabetical order, I would like to thank: Orion Afisiadis, Konstantinos Alexandris, Andrew Austin, Pavle Belanovic, Andrea Bonetti, Jeremy Constantin, Shrikanth Ganapathy, Pascal Giard, Georgios Karakonstantis, Reza Ghanaatian, Pascal Meinerzhagen, Christoph Müller, Nicholas Preyss, Lorenz Schmid, Christian Senning, Adam Teman, Johannes Wüthrich.

I also had the pleasure of working with several brilliant people outside of our laboratory. In particular, I would like to thank Mani Bastani Parizi (Information Theory Laboratory, EPFL) for a very fruitful collaboration on the implementation of polar decoders and for pleasant mealtime discussions. I would also like to thank Mani's advisor, Prof. Emre Telatar, for initiating and encouraging our collaboration. Many thanks also go to Michael Meidlinger and his advisor Prof. Gerald Matz (Institute of Telecommunications, Vienna Institute of Technology) for our collaboration on the implementation of ultra high-speed look-up table based decoders for LDPC codes and for their impeccable integrity and professionalism. I also had the pleasure of working on the implementation of polar decoders with students of Prof. Warren J. Gross (Integrated Systems for Information Processing Laboratory, McGill University) on several occasions. In particular, I would like to thank Seyyed Ali Hashemi, Alexandre Raymond, and Gabi Sarkis for the smooth collaboration and helpful discussions. While we only had the opportunity of collaborating once to this date, I would like to thank Prof. Joseph Cavallaro (Center for Multimedia Communication, Rice University) for his openness and for very pleasant discussions at several conferences.
During my PhD studies, I had the thrilling opportunity of doing an internship at Intel Labs in Hillsboro, Oregon, USA, under the supervision of Dr Farhana Sheikh. I would like to thank her for believing in my technical abilities, for providing me with a very interesting research project to work on, for her guidance throughout my internship, and for a very educational glimpse into the world of leadership and management. I would also like to thank my Intel Labs colleagues Chia-Hsiang Chen, Ching-En (Alex) Lee, and Wei Tang for interesting and honest discussions.
Finally, I would like to thank my parents, Heidi and Georgios, and my girlfriend, Kynthia, for their love, for believing in me and supporting my decisions, and for always being proud of my achievements!

Lausanne, 10 October 2016 Alexios Balatsoukas Stimming

Abstract

The goal of channel coding is to detect and correct errors that appear during the transmission of information. In the past few decades, channel coding has become an integral part of most communications standards as it improves the energy-efficiency of transceivers manyfold while only requiring a modest investment in terms of the required digital signal processing capabilities. The most commonly used channel codes in modern standards are low-density parity-check (LDPC) codes and Turbo codes, which were the first two types of codes to approach the capacity of several channels while still being practically implementable in hardware. The decoding algorithms for LDPC codes, in particular, are highly parallelizable and suitable for high-throughput applications.

A new class of channel codes, called polar codes, was introduced recently. Polar codes have an explicit construction and low-complexity encoding and successive cancellation (SC) decoding algorithms. Moreover, polar codes are provably capacity achieving over a wide range of channels, making them very attractive from a theoretical perspective. Unfortunately, polar codes under standard SC decoding cannot compete with the LDPC and Turbo codes that are used in current standards in terms of their error-correcting performance. For this reason, several improved SC-based decoding algorithms have been introduced. The most prominent SC-based decoding algorithm is the successive cancellation list (SCL) decoding algorithm, which is powerful enough to approach the error-correcting performance of LDPC codes. The original SCL decoding algorithm was described in an arithmetic domain that is not well-suited for hardware implementations and is not clear how an efficient SCL decoder architecture can be implemented. To this end, in this thesis, we re-formulate the SCL decoding algorithm in two distinct arithmetic domains, we describe efficient hardware architectures to implement the resulting SCL decoders, and we compare the decoders with existing LDPC and Turbo decoders in terms of their error-correcting performance and their implementation efficiency.

Due to the ongoing technology scaling, the feature sizes of integrated circuits keep shrinking at a remarkable pace. As transistors and memory cells keep shrinking, it becomes increasingly difficult and costly (in terms of both area and power) to ensure that the implemented digital circuits always operate correctly. Thus, manufactured digital signal processing circuits, including channel decoder circuits, may not always operate correctly. Instead of discarding these faulty dies or using costly circuit-level fault mitigation mechanisms, an alternative approach is to try to live with certain malfunctions, provided that the algorithm implemented by the circuit is sufficiently fault-tolerant. In this spirit, in this thesis we examine decoding of polar codes and LDPC codes under the assumption that the memories that are used within the decoders

are not fully reliable. We show that, in both cases, there is inherent fault-tolerance and we also propose some methods to reduce the effect of memory faults on the error-correcting performance of the considered decoders.
As we already explained, LDPC codes are well-suited for high-speed applications. A new degree of parallelism in LDPC decoding was recently explored by completely unrolling the decoding loop of the LDPC decoder and mapping every decoding iteration directly to hardware, leading to a decoder architecture that can achieve a throughput of over one Terabit per second. However, routing is a severe problem in this kind of architecture as there are typically hundreds of thousands of global wires required in order to connect the various processing blocks within the decoder. In this thesis, we apply an information-theoretic quantization method in order to significantly reduce the bit-width of the quantities that are processed within an unrolled LDPC decoder. Using this approach, we reduce the area and increase the maximum operating frequency, while also significantly reducing the routing congestion.

Key words: Polar codes, successive cancellation list decoding, hardware implementation, VLSI, approximate computing, faulty decoding, LDPC codes, unrolled decoding.

Résumé

L'objectif du codage de canal est de détecter et de corriger les erreurs introduites lors de la transmission de données. Au cours des dernières décennies, le codage de canal est devenu partie intégrante de la plupart des normes de communication car il améliore significativement l'efficacité énergétique des émetteurs-récepteurs au coût d'un modeste investissement en terme de capacité de traitement numérique du signal. Les codes pour canaux les plus communs dans les normes modernes sont les codes à contrôle de parité de faible densité (LDPC) ainsi que les turbo codes. Ces codes furent les deux premiers types de codes à approcher la capacité de plusieurs canaux tout en permettant une implémentation matérielle réalisable. Notamment, les algorithmes de décodage des codes LDPC sont hautement parallélisables et, de ce fait, sont bien adaptés aux applications à haut débit.

Une nouvelle classe de code de canal, appelée codes polaires, fut récemment introduite. Les codes polaires ont une construction explicite ainsi que des algorithmes d'encodage et de décodage, par annulations successives (SC), de faible complexité. De plus, les codes polaires permettent d'atteindre la capacité d'une grande plage de canaux les rendant ainsi très attrayant d'un point de vue théorique. Malheureusement, lorsque décodés avec l'algorithme SC, la performance en terme de correction d'erreurs des codes polaires ne peut compétitionner avec celle des codes LDPC ou des turbo codes utilisés dans les normes de communication actuelles. C'est pourquoi plusieurs algorithmes de décodage améliorés, se basant tout de même sur l'annulation successive, furent introduits. Le plus notoire de ces algorithmes est l'algorithme de décodage par annulations successives de type liste (SCL). C'est un algorithme suffisamment puissant pour permettre aux codes polaires de rivaliser avec la performance de correction d'erreurs des codes LDPC. L'algorithme de décodage SCL originel fut décrit dans un domaine arithmétique mal adapté à l'implémetation matérielle. De ce fait, il n'est pas clair qu'une architecture matérielle efficace pour un décodeur SCL soit réalisable. À cet effet, dans cette thèse, nous reformulons l'algorithme de décodage SCL dans deux domaines arithmétiques distincts et nous décrivons des architectures matérielles efficaces pour l'implémentation des décodeurs SCL résultants. Enfin, nous comparons ces décodeurs avec les décodeurs existants, pour codes LDPC et turbo codes, à la fois en terme de performance de corrections d'erreurs qu'en terme d'efficacité d'implémentation.

Grâce aux progrès de la miniaturisation, la taille des circuits intégrés ne cesse de réduire, et ce, à un rythme remarquable. À mesure que la taille des transistors et des cellules mémoires continues d'être réduite, il devient de plus en plus ardu et onéreux (en termes d'espace et de puissance) d'assurer le perpétuel bon fonctionnement des circuits numériques. Ainsi, les

circuits fabriqués de traitement numérique du signal, incluant les circuits de décodage de canal, peuvent ne pas toujours opérer correctement. Au lieu de jeter ces puces dysfonctionnelles ou d'utiliser des mécanismes coûteux de mitigation de fautes au niveau du circuit, une approche alternative consiste à tenter de faire avec certains dysfonctionnements, en autant que l'algorithme implémenté par le circuit soit suffisamment tolérant aux fautes. C'est dans cet esprit que, dans cette thèse, nous examinons le décodage de codes polaires et de codes LDPC sous l'hypothèse que les mémoires utilisées dans ces décodeurs ne sont pas complètement fiables. Nous montrons que, dans les deux cas, il y a une tolérance inhérente aux fautes et nous proposons également quelques méthodes pour réduire l'effet des fautes mémoires sur la performance de correction d'erreurs des décodeurs considérés.
Tel qu'expliqué précédemment, les codes LDPC sont appropriés pour les applications à haut débit. Un nouveau degré de parallélisme dans le décodage LDPC fut récemment exploré en déroulant complètement la boucle de décodage d'un décodeur LDPC et en faisant un mappage direct de chacune des itérations de décodage en matériel. Cela conduisit à une architecture de décodeur pouvant atteindre un débit de plus d'un térabit par seconde. Cependant, dans ce genre d'architecture, le routage est un problème sévère puisqu'il y a typiquement des centaines de milliers de fils requis afin d'interconnecter les divers blocs de traitement au sein du décodeur. Dans cette thèse, nous appliquons une méthode de quantification issue de la théorie de l'information afin de significativement réduire le nombre de bits requis pour exprimer les valeurs traitées dans un décodeur LDPC déroulé. En utilisant cette approche, nous réduisons l'espace requise et augmentons la fréquence maximale d'opération tout en réduisant significativement la congestion lors du routage.

Mots clefs : Codes polaires, décodage par annulations successives de type liste, implémentation matérielle, VLSI, calcul approximatif, décodage erroné, codes LDPC, décodage déroulé.

Zusammenfassung

Kanalcodierung wird verwendet um Fehler, die bei der Informationsübertragung auftreten können, zu erkennen und zu korrigieren. Kanalcodierung ist ein integraler Bestandteil der meisten Kommunikationsstandards der letzten Jahrzehnten, weil sie deren Energieeffizienz vielfach verbessern kann mit geringen Anforderungen bezüglich der erforderlichen Signalverarbeitungsfähigkeit des Systems. Die meisten modernen Kommunikationsstandards setzen Paritätsprüfungcodes geringer Dichte ("low-density parity-check (LDPC) codes") oder Turbo Codes ein. LDPC und Turbo Codes waren die ersten zwei Arten von praktisch implementierbaren Kanalcodes die sich der Kapazität von mehreren Übertragungskanälen nähern können. Die Decodieralgorithmen für LDPC Codes sind besonders hoch parallelisierbar und sind daher sehr geeignet für Anwendungen die hohen Datendurchsatz erfordern.

Polare Codes sind die jüngste Entwicklung im Bereich der Kanalcodierung. Polare Codes verfügen über eine explizite Konstruktion und sie können mit Hilfe sukzessiver Annullierung ("successive cancellation (SC) decoding") mit geringer Komplexität decodiert werden. Zudem können polare Codes nachweisbar die Kapazität von mehreren Übertragungskanälen erreichen und sind daher aus theoretischer Sicht hochattraktiv. Die Fehlerkorrekturleistung von polaren Codes mit dem SC Decodieralgorithmus steht jedoch der Fehlerkorrekturleistung von LDPC und Turbo Codes leider bedeutend nach. Aus diesem Grund wurden mehrere verbesserte Decodieralgorithmen eingeführt die auf dem SC Algorithmus basieren. Der bekannteste SC-basierte Decodieralgorithmus ist der SC Listenalgorithmus (SCL), der leistungsfähig genug ist um sich der Fehlerkorrekturleistung von LDPC Codes zu nähern. Der SCL Decodieralgorithmus wurde ursprünglich in einem arithmetischen Domain beschrieben, der für Hardwareimplementierungen schlecht geeignet ist. Es ist daher eher unklar wie eine effiziente SCL Decoder-Architektur implementiert werden kann. In dieser Doktorarbeit formulieren wir aus diesem Grund den SCL Algorithmus in zwei unterschiedlichen arithmetischen Domänen die für Hardwareimplementierungen besser geeignet sind. Ausserdem beschreiben wir effiziente Hardwarearchitekturen für die neu formulierten SCL Algorithmen und vergleichen wir diese Architekturen mit existierenden LDPC und Turbo Decodern bezüglich ihrer Fehlerkorrekturleistung und Implementierungseffizienz.

Die Integrationsdichte von integrierten Schaltungen nimmt mit bemerkenswertem Tempo zu aufgrund der andauernden Technologie-Skalierung. Da Transistoren und Speicherzellen immer kleiner werden, wird es zunehmend schwieriger und kostspieliger (bezüglich des Energievebrauchs und der Schaltungsfläche) sicher zu stellen dass digitale Schaltungen immer einwandfrei funktionieren. Aus diesem Grund kann es vorkommen dass hergestellte

digitale Signalverarbeitungsschaltungen, einschliesslich Kanaldecoderschaltungen, fehlerhaft sind. Die fehlerhaften Schaltungen werden üblicherweise entweder weggeworfen oder durch kostspielige Fehlerausgleichsmechanismen abgeschirmt. Ein alternativer Ansatz ist eine bestimmte Anzahl von Störungen zuzulassen, vorausgesetzt dass der implementierte Algorithmus ausreichend fehlertolerant ist. In dieser Doktorarbeit untersuchen wir in diesem Sinne die Decodierung von polaren Codes und LDPC Codes unter der Annahme dass die von der digitalen Schaltung verwendeten Speicherelemente nicht völlig zuverlässig sind. Wir beweisen, dass beide Decoder inhärent Fehlertolerant sind und wir schlagen einige Verfahren vor die die Auswirkung der Speicherfehler auf die Fehlerkorrekturleistung der untersuchten Decoder verringern können.
Wie bereits erklärt wurde sind LDPC Codes gut geeignet für Anwendungen die hohen Datendurchsatz erfordern. Ein neuer Parallelitätsgrad in der Decodierung von LDPC Codes, der durch das vollständige Abrollen der Decodierungsschleife des LDPC-Decoders und durch die direkte Implementierung jeder Decodieriteration in Hardware ermöglicht wird, wurde in letzter Zeit untersucht. Diese Methode ermöglicht die Gestaltung von Decodierungsschaltungen die einen Durchsatz von mehr als einem Terabit pro Sekunde erreichen. Das Leitungsrouting ist ein schwerwiegendes Problem in dieser Art von Architektur, weil typischerweise Hunderttausende von globalen Leitungen erforderlich sind, um die verschiedenen Verarbeitungsblöcke innerhalb des Decoders miteinander zu verbinden. In dieser Doktorarbeit verwenden wir ein informationstheoretisches Quantisierungsverfahren, um die Bitbreite der Daten die in einem abgerollten LDPC Decoder verarbeitet werden deutlich zu reduzieren. Dieser Ansatz reduziert die Schaltungsfläche, erhöht die maximale Betriebsfrequenz und reduziert die Routing-Kongestion der Decodierungsschaltung deutlich.

Stichwörter: Polare Codes, Listenalgorithmus mit sukzessiver Annullierung, Hardwareimplementierung, VLSI, approximierende Berechnung, defekte Decodierung, LDPC Codes, abgerollte Decodierung.

Contents

Acknowledgements i

Abstract (English/Français/Deutsch) iii

List of figures xiii

List of tables xix

1 Introduction 1
- 1.1 Thesis Outline & Contributions . . . 3
- 1.2 Notation and Preliminaries . . . 5
- 1.3 Polar Codes . . . 6
  - 1.3.1 Polarizing Transformation . . . 6
  - 1.3.2 Construction of Polar Codes . . . 9
  - 1.3.3 Decoding of Polar Codes . . . 10
- 1.4 LDPC Codes . . . 19
  - 1.4.1 Construction of LDPC Codes . . . 19
  - 1.4.2 Message-Passing Decoding of LDPC Codes . . . 20

2 Hardware Decoders for Polar Codes 23
- 2.1 LL-Based SCL Decoder . . . 24
  - 2.1.1 Likelihood Representation . . . 24
  - 2.1.2 List SC Decoder Architecture . . . 26
- 2.2 LLR-Based SCL Decoder . . . 30
  - 2.2.1 LLR-Based Path Metric Computation . . . 31
  - 2.2.2 LLR-Based SCL Decoder Hardware Architecture . . . 35
  - 2.2.3 Path Metric Sorting . . . 38
- 2.3 Hardware Implementation Results . . . 48
  - 2.3.1 Quantization Parameters . . . 48
  - 2.3.2 Comparison of Path Metric Sorters . . . 49
  - 2.3.3 LLR-based SCL Decoder: Radix-2L Sorter versus Pruned Radix-2L Sorter 52
  - 2.3.4 LLR-based SCL Decoder: Comparison with LL-based SCL Decoders . . . 53
  - 2.3.5 CRC-Aided SCL Decoder . . . 56
- 2.4 Polar Decoder Survey and Comparison with Existing Decoders . . . 62

2.4.1 Polar Hardware Decoders . . . 62
2.4.2 Comparison of Polar Codes with LDPC and Turbo Codes . . . 70
2.5 Summary . . . 84

3 Faulty Polar and LDPC Channel Decoders 87
3.1 Approximate Computing . . . 87
3.2 Successive Cancellation Decoding with Intentionally Mismatched Polar Codes 88
3.2.1 Complexity-Performance Trade-Offs for SC Decoding of Polar Codes . . . 88
3.2.2 Greedy Optimization Algorithm . . . 92
3.2.3 Numerical Results . . . 94
3.3 Successive Cancellation Decoding of Polar Codes with Faulty Memories . . . 97
3.3.1 Faulty Successive Cancellation Decoding of Polar Codes for the BEC . . . 98
3.3.2 Erasure Probability of Synthetic Channels Under Faulty SC Decoding . . . 101
3.3.3 Frame Erasure Rate Under Faulty SC Decoding . . . 106
3.3.4 Unequal Error Protection . . . 109
3.3.5 Optimal Blocklength Under Faulty SC Decoding . . . 110
3.3.6 Numerical results . . . 111
3.4 Min-Sum Decoding of LDPC Codes with Faulty Memories . . . 114
3.4.1 Channel Model and Memory Fault Model . . . 116
3.4.2 Density Evolution for Faulty Quantized MS Decoding . . . 117
3.4.3 Bit-Error Probability and Decoding Threshold . . . 120
3.4.4 Numerical Results . . . 121
3.5 Summary . . . 123

4 Hardware Decoders for Ultra High-Speed Decoding of LDPC Codes 125
4.1 Mutual Information Based Message Quantization . . . 126
4.1.1 Channel Model and Symmetry Conditions . . . 127
4.1.2 LUT Design via Density Evolution . . . 128
4.2 LUT Design Considerations for Practical Decoders . . . 130
4.2.1 Performance of Min-LUT Decoding . . . 130
4.2.2 Reduced Complexity LUT Structure . . . 131
4.2.3 Design SNR . . . 133
4.2.4 LUT Re-use . . . 134
4.2.5 LUT Input/Output Alphabet Downsizing . . . 136
4.3 LUT-Based Fully Unrolled Decoder Hardware Architecture . . . 138
4.3.1 Decoder Architecture . . . 139
4.3.2 Decoding Latency and Throughput . . . 142
4.3.3 Memory Requirements . . . 142
4.4 Implementation Results . . . 142
4.4.1 Quantization Parameters . . . 143
4.4.2 Adder-based vs. LUT-based Decoder . . . 143
4.5 Summary . . . 145

5 Conclusion & Outlook **147**

Bibliography **162**

List of Figures

1.1 One step of the polarizing transformation applied to two copies of the channel W and generating two synthetic channels $W^{(-)}$ and $W^{(+)}$. 7

1.2 Synthetic channel construction for a polar code of length $N = 2^3 = 8$. Pairs of solid lines represent the + transformation and pairs of dashed lines represent the − transformation. 8

1.3 Implementation of $\boldsymbol{F}^{\otimes n}$ for a polar encoder of length $N = 2^3 = 8$. The application of $\boldsymbol{B}_N$ to the result of this encoding circuit would simply re-arrange the output in the natural ordering, i.e., $\boldsymbol{c} = [\, c_0\, c_1\, \ldots\, c_7\,]$. 9

1.4 The data dependency graph (DDG) for the computation of the LLRs of a polar code with $N = 2^3 = 8$. Solid lines represent application of f_+ and dashed lines represent application of f_-. The partial sums required for the f_+ updates can be computed by using the encoder structure of Figure 1.3 and setting $u_i = \hat{u}_i$ once each estimate becomes available. 12

1.5 Factor graph for belief propagation decoding of polar codes. 16

1.6 Basic computation unit for belief propagation decoding. 17

1.7 Example of a Tanner graph for a $(2,4)$-regular LDPC code of blocklength $N = 6$. 19

1.8 (a) Variable node update for $\mathcal{N}(n) = \{k, k_1, \ldots, k_{d_v-1}\}$ and (b) check node update for $\mathcal{N}(k) = \{n, n_1, \ldots, n_{d_c-1}\}$. 20

1.9 Decision node update for $\mathcal{N}(n) = \{k, k_1, \ldots, k_{d_v}\}$. 21

2.1 High-level overview of the list SC decoder architecture. 26

2.2 Details of the proposed LL-based SCL decoder: (a) pointer memory, (b) metric sorter. 28

2.3 Overview of the SCL decoder architecture. Details on the i, s, p_s, as well as the func & stage and MemAddr components inside the control unit, which are not described in this section, can be found in Section 2.1. The dashed green and the dotted red line show the critical paths for $L = 2$ and $L = 4, 8$ respectively. 36

2.4 Bit-cell copying mechanism controlled by the metric sorter. 37

2.5 Radix-$2L$ sorter for $L = 2$. 40

2.6 Bitonic sorter for $L = 4$. 41

2.7 Pruned radix-$2L$ sorter for $L = 2$. 42

2.8 Pruned bitonic sorter for $L = 4$. The full bitonic sorter requires all the depicted CAS units (cf. Figure 2.6), while in the pruned bitonic sorter all CAS units in red dotted lines can be removed. 43
2.9 Bubble sorter for $2L = 8$. The full bubble sorter requires all the depicted CAS units, while in the simplified bubble sorter all CAS units in red dotted lines can be removed. 46
2.10 The performance of floating-point vs. fixed-point SCL decoders ($L = 1$, i.e., SC decoding, and $L = 2$). $M = 8$ quantization bits are used for the path metric in fixed-point SCL decoders. 49
2.11 The performance of floating-point vs. fixed-point SCL decoders ($L = 4$ and $L = 8$). $M = 8$ quantization bits are used for the path metric in fixed-point SCL decoders. 50
2.12 The performance of LLR-based SCL decoders compared to that of CRC-aided SCL decoders for $L = 2, 4, 8$. 58
2.13 CA-SCLD with $L = 2, 4$, results in the same performance at blocklength $N = 1024$ as the conventional SC decoding with $N = 2048$ and $N = 4096$, respectively. . . 60
2.14 Time complexity vs. area for various decoders for polar codes. All decoders are given for $N = 1024$. The SCL decoder implementations are given for $L = 4$. The area and operating frequency are normalized to 90 nm CMOS technology using standard technology scaling rules. 69
2.15 Throughput vs. power for various decoders for polar codes. The power and operating frequency are normalized to 90 nm CMOS and 1 V using standard technology scaling rules. 70
2.16 Performance of the LDPC code of the IEEE 802.11ad standard compared to polar codes under SC decoding, BP decoding, and SCL decoding (8-bit CRC). 72
2.17 Hardware efficiency of IEEE 802.11ad LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling. 73
2.18 Performance of the LDPC code of the IEEE 802.11ad standard compared to polar codes under SC decoding and BP decoding, and a polar code with $N = 512$ under SCL decoding (8-bit CRC). 74
2.19 Hardware efficiency of IEEE 802.11ad LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER. 74
2.20 Performance of the LDPC code of IEEE 802.11n standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding, and SCL decoding (8-bit CRC). 75
2.21 Hardware efficiency of IEEE 802.11n LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling. 76
2.22 Performance of the LDPC code of IEEE 802.11n standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding ($I = 40$), and SCL decoding (8-bit CRC). 77

2.23 Hardware efficiency of IEEE 802.11n LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER. 77
2.24 Performance of the LDPC code of the IEEE 802.3an standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding, and SCL decoding (8-bit CRC). 78
2.25 Hardware efficiency of IEEE 802.3an LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling. 79
2.26 Performance of the LDPC code of the IEEE 802.3an standard compared to polar codes with $N = 4096$ under SC and BP decoding, and $N = 1024$ under SCL decoding (8-bit CRC). 80
2.27 Hardware efficiency of IEEE 802.3an LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER. 80
2.28 Performance of Turbo code of LTE standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding ($I = 15$), and SCL decoding ($L = 4$, 8-bit CRC). 81
2.29 Hardware efficiency of LTE Turbo decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling. . . 82
2.30 Performance of Turbo code of LTE standard compared to polar codes with $N = 32768$ under SC decoding and BP decoding ($I = 30$), and polar codes with $N = 4096$ under SCL decoding ($L = 4$, 16-bit CRC). 83
2.31 Hardware efficiency of 3GPP LTE Turbo decoder implementations and SC, BP, and SCL polar decoder implementations which have been scaled in order to match the FER performance of the 3GPP LTE Turbo decoders. 83

3.1 Decoding graph for $N = 4$ with channel groups. An optimization variable x_i is associated with each group g_i. Setting $x_i = 1$ corresponds to freezing all channels in g_i. 90
3.2 Tree structure of channel groups with descendants of g_6, i.e., $D(g_6)$, and their corresponding optimization variables. If $x_6 = 1$, then $x_i = 0$ has to be enforced for all $x_i : g_i \in D(g_6)$. 91
3.3 Results from exact solution of (3.11) and of the greedy algorithm for $R = 0.5$, $N = 2^n$, $n = 4,5,6,7$, and transmission over a BEC(0.5). 95
3.4 Solutions of greedy algorithm for $R = 0.5$, $N = 2^n$, $n = 9,10,11,12,13,15$, over a BEC(0.5). 96
3.5 Frame erasure rate performance and performance metric of the *useful* codes for $R = 0.5$ and $N = 2^{10}$. 96
3.6 Synthetic channel construction for a polar code of length $N = 2^2 = 4$ under faulty SC decoding. Solid lines represent the + transformation and dashed lines represent the − transformation. 100

3.7 Sorted $Z_{n,\delta}^{(\mathbf{s})}$, $\mathbf{s} \in \{+,-\}^n$ and $Z_n^{(\mathbf{s})}$, $\mathbf{s} \in \{+,-\}^n$, values for polar codes of length $N = 256, 1024, 4096$, designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and non-faulty decoding, respectively. 112

3.8 Evaluation of P_e^{UB} and P_e^{LB} for polar codes of lengths $N = 256, 1024, 4096$, designed for the BEC(0.5) with $\delta = 10^{-6}$. 113

3.9 Evaluation of P_e^{UB} and P_e^{LB} for $N = 2^n$, $n = 0, \ldots, 12$, and various code rates $R \in \{0.1250, 0.1875, 0.2500\}$ for transmission over a BEC with erasure probability 0.5 under faulty SC decoding with $\delta = 10^{-6}$. 113

3.10 FER for a polar code of length $N = 1024$ designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and $n_{\mathrm{p}} = 0, \ldots, 5$, protected decoding levels. Protecting $n_{\mathrm{p}} = n + 1$ levels is equivalent to using a non-faulty decoder. 114

3.11 FER for polar codes of length $N = 512, 1024, 2048$, designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and $n_{\mathrm{p}} = n - 5$ protected decoding levels. 115

3.12 Message fault model: an incoming b-bit noiseless message of value m is passed through b independent BSC(δ) channels, resulting in the faulty message $e(m)$. 116

3.13 Faulty variable node update for $\mathcal{N}(n) = \{m, m_1, \ldots, m_{d_v-1}\}$ (a) and faulty check node update (b) for $\mathcal{N}(m) = \{n, n_1, \ldots, n_{d_c-1}\}$. 117

3.14 Faulty decision node update for $\mathcal{N}(n) = \{m, m_1, \ldots, m_{d_v}\}$. 117

3.15 Error probability for a (3,6)-regular LDPC code under faulty MS and MS decoding for $\delta = 10^{-5}$ and $\delta = 10^{-6}$. The calculated thresholds are $\sigma_*^2(10, 10^{-5}) = 0.6576$ and $\sigma_*^2(10, 10^{-6}) = 0.6582$. 121

3.16 Decoding threshold for a (3,6)-regular LDPC code under faulty MS and MS decoding for $\delta = 10^{-3}$ for different numbers of quantization bits. 122

4.1 Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code ($\ell_{\max} = 5$). 131

4.2 Six different LUT tree structures. Note that $T_1 \geq_{\mathcal{T}} T_4 \geq_{\mathcal{T}} T_6$, $T_2 \geq_{\mathcal{T}} T_5$, $T_3 \geq_{\mathcal{T}} T_5$, and $T_3 \geq_{\mathcal{T}} T_6$. However, we cannot compare T_2 with T_3 or T_5 with T_6 using the relation $\geq_{\mathcal{T}}$. 132

4.3 FER versus channel SNR for min-LUT decoder at different design SNRs γ for the IEEE 802.3an LDPC code ($\ell_{\max} = 5$, $|\mathcal{L}| = 4$, $|\mathcal{M}| = 3$). 134

4.4 Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code and various LUT re-use patterns $\boldsymbol{r}$ ($\ell_{\max} = 5, |\mathcal{L}| = 2^4, |\mathcal{M}| = 2^3, \gamma = 4.2$ dB). 135

4.5 Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code and various LUT downsizing patterns $\boldsymbol{d}$ ($\ell_{\max} = 5, |\mathcal{L}| = 2^4, \gamma = 4.2$ dB). 136

4.6 Top level decoder architecture processing pipeline. The channel LLRs are the input of the left-hand side and the decoded codeword is obtained as the output of the right-hand side. 139

4.7 (a) The variable node LUT tree that is used in the hardware implementation for the calculation of one output of a variable node of degree $d_v = 6$. This tree is identical to T_6 of Figure 4.2. Each LUT-based variable node contains d_v such LUT trees, one for each combination of $(d_v - 1)$ input messages. (b) The decision node LUT tree that is used in the hardware implementation for the hard decisions taken by each variable node of degree $d_v = 6$. This tree is similar to T_5 of Figure 4.2 with an additional input added to the right LUT of the lowest level. Each LUT-based decision node contains a single decision tree. 141
4.8 FER vs E_b/N_0 for the $N = 2048$ $(6,32)$-regular LDPC code defined in IEEE 802.3an under various decoding algorithms. 143

List of Tables

2.1 Synthesis Results for Radix-2L and Pruned Radix-2L Sorters 50
2.2 Synthesis Results for Bitonic and Pruned Bitonic Sorters 51
2.3 Comparison of Pruned Radix-2L, Pruned Bitonic, and Simplified Bubble Sorters 52
2.4 LLR-based SCL Decoder: Radix-2L vs. Pruned Radix-2L Sorter 52
2.5 Metric Sorter Delay and Critical Path Start- and Endpoints for our LLR-Based SCL Decoder Using the Radix-2L and the Pruned Radix-2L Sorters. 53
2.6 SCL Decoder Synthesis Results ($R = \frac{1}{2}$, $N = 1024$) 53
2.7 Comparison of LLR-based implementation with existing LL-based implementations 54
2.8 Cell Area Breakdown for the LL-Based and the Radix-2L LLR-based SCL Decoders ($R = \frac{1}{2}$, $N = 1024$) 54
2.9 Throughput Reduction in CRC-Aided SCL Decoders 57
2.10 LLR-Based SC Decoder vs. SCL Decoder Synthesis Results 59
2.11 SC Decoder Hardware Implementations 63
2.12 Complexity and Decoding Latency of Different SC Decoder Architectures 64
2.13 BP Decoder Hardware Implementations 65
2.14 SCL Decoder Hardware Implementations ($L = 4$) 69
2.15 Properties of the LDPC and Turbo codes used for comparison. 70
2.16 IEEE 802.11ad LDPC Decoder Implementations. 84
2.17 IEEE 802.11n LDPC Decoder Implementations. 84
2.18 IEEE 802.3an LDPC Decoder Implementations. 85
2.19 3GPP LTE Turbo Decoder Implementations. 85

3.1 Thresholds of various (d_v, d_c)-regular codes under MS and faulty MS decoding for $\alpha = 10$ and $b = 5$ bits. 122

4.1 Comparison of cumulative depth and DE threshold for various tree structures (cf. Figure 4.2). 132
4.2 Synthesis Results for the Adder-based and the LUT-based Decoders 143
4.3 Area Breakdown 144

1 Introduction

Practically all modern communications systems are digital in nature and they use sophisticated digital signal processing techniques that can be readily implemented using digital integrated circuits. The quality of a digital communications system can be described by a quantifiable and intuitive metric, called the *bit-error rate*, which is defined as the average fraction of transmitted bits that are mistaken for a different bit at the receiver due to the noise introduced by the transmission channel. Error-correction coding has become an integral part of digital communications systems, as it can significantly reduce their bit-error rate and, in turn, increase their efficiency.

For example, consider a system whose (uncoded) bit-error rate is P_e. If a single bit is transmitted using this system, it will arrive correctly at the receiver with probability $P_c = 1 - P_e$ and in error with probability P_e. Now consider the case where we transmit the same bit three times over the channel and use a majority rule at the receiver to decode the transmitted bit. In this scenario, the bit will be received correctly if either no instances of the bit are in error or if one instance of the bit is in error and the probability of receiving the bit correctly is

$$P_{c,\text{coded}} = (1-P_e)^3 + 3(1-P_e)^2 P_e = 1 - P_e^2 + 2P_e^3. \tag{1.1}$$

It can be verified that $P_{e,\text{coded}} = 1 - P_{c,\text{coded}} < P_e$ for any $P_c \in (0,1)$. Thus, this repetition code improves the bit-error probability of the system. However, it also reduces the rate of the system, since only one information bit is transmitted for every three coded bits. In general, the code rate is defined as the number of information bits K transmitted over the number of total bits N transmitted, i.e., $R = \frac{K}{N}$. This simple repetition code can be easily generalized to any odd blocklength N with rate $R = \frac{1}{N}$, where a bit error occurs if more than $\frac{N-1}{2}$ of the bit instances are received erroneously. A simple lower bound on the bit-error probability of any code can be derived if we only consider one of the events that lead to a bit-error, namely the case where all N bit instances are received erroneously. This gives us the lower bound $P_{e,\text{coded}} \geq P_e^N$. Thus, a necessary condition for $P_{e,\text{coded}}$ to become arbitrarily small is that N must go to infinity. However, as N goes to infinity, the rate of the repetition code goes to zero, leaving us with the tautological statement that the only way to avoid bit-errors during

transmission over a noisy channel is to not send any bits over this channel.

The belief that arbitrarily reliable transmission of information is only possible if the rate of transmission goes asymptotically to zero was widely held until the seminal work of Shannon in 1948 [1]. Shannon showed that each transmission channel has a capacity and that arbitrarily reliable transmission is in fact possible at any rate that is strictly smaller than the capacity of the channel. Shannon used a random coding argument for his proof which does not enable the construction of practically useful codes, since both the encoding and the decoding complexity of randomly constructed codes are exponential in the blocklength N. While error-correcting codes existed before the work of Shannon, the promise of a fundamental limit that is, at least in principle, achievable essentially gave birth to the field of coding theory.

Classical coding theory studies codes mainly in terms of their algebraic properties, such as the minimum distance, in order to derive bounds on the error-correcting capabilities of these codes. Hamming codes and Reed-Solomon codes are well-known and widely used examples of classical codes. The way of looking at codes changed fundamentally in the 1990s, when algebraic properties gave way to the analysis of codes modeled using sparse graphs and efficient message-passing decoding algorithms. A common term used to describe this paradigm shift is *modern coding theory* [2]. Turbo codes [3] are one of the first examples of modern codes and they are also the first class of codes that was able to approach channel capacity while still being practically implementable. Low-density parity-check (LDPC) codes [4, 5] are another famous example of modern codes that are both capacity-approaching and implementable with reasonable hardware complexity. While both Turbo and LDPC codes have excellent capacity-approaching performance, it has not been shown that they are generally capacity-achieving. The latest breakthrough in channel coding came with Arıkan's polar codes [6], which are provably capacity-achieving over a very wide range of transmission channels.

Since error-correcting codes are an essential part of today's communications systems, the hardware implementation of such systems is a crucial issue. Indeed, a Google Scholar search reveals that there are more than 50'000 publications on the implementation of Turbo decoders and more than 20'000 publications on the implementation of LDPC decoders.[1] However, a similar search for decoder implementations of the more recently invented polar codes only returns slightly more than 200 results. Thus, while it is safe to say that after two decades of research we know how to build efficient Turbo and LDPC decoders for most applications, the same claim can unfortunately not be made for polar decoders, as the field is still in its infancy.

Digital signal processing algorithms and their hardware implementation are commonly treated in isolation. System engineers devise algorithms that are then passed on to the hardware engineers who make sure that they are implemented as efficiently as possible. However, as the node sizes of integrated circuits keep shrinking, this classical approach of layered

[1] We used the search string `CODE+("code*"|"decoder*")+("hardware"|"vlsi"|"fpga"|"asic")`, where `CODE` was either `"turbo"` or `("ldpc"|"low-density parity-check")` or `"polar"`. Specifically for polar codes, we had to append `+"arikan"` to the search string in order to exclude several thousands of publications related to physics and biology.

abstraction not only becomes inefficient, but even inaccurate. This happens because various effects that affect the analog components that are used to build digital circuits actually start manifesting themselves in the operation of the digital circuit. The altered operation of the digital circuit, in turn, directly affects the functionality of the algorithm that it implements. Thus, algorithms and hardware become inextricably intertwined and it becomes imperative to study the behavior of digital signal processing algorithms in a more holistic and cross-layer fashion that takes into account the intricacies of sub-100 nm VLSI technologies. This cross-layer approach can have a big impact on the energy efficiency and manufacturing cost of integrated circuits, as it allows us to find ways to use less reliable and less energy-hungry hardware while still guaranteeing acceptable performance levels for many applications.

1.1 Thesis Outline & Contributions

The contributions of this thesis can be summarized as follows. First, we bridge the gap between Turbo/LDPC decoders and polar decoders by showing how the successive cancellation list decoding algorithm for polar codes, which is particularly interesting due to its superior error rate performance, can be efficiently implemented in hardware. Second, we examine the performance of several channel decoding algorithms in scenarios where the digital hardware that is used to implement them is faulty. Finally, we use a sophisticated quantization method that is inspired by an information-theoretic performance metric in order to design an ultra high-speed LDPC decoder that achieves a decoding throughput of more than one Terabit per second.

In the following, we will briefly outline the contents of each chapter of this thesis and we will summarize the respective main contributions.

Chapter 2: Hardware Decoders for Polar Codes

This chapter deals with the hardware implementation of various successive cancellation list (SCL) decoders for polar codes. We note that this chapter is derived from our works of [7, 8, 9, 10].

More specifically, in Section 2.1 we present the first hardware implementation of an SCL decoder in the literature. This architecture uses a log-likelihood (LL) representation for the internal messages and a smart copying mechanism in order to avoid copying the path likelihoods directly.

In Section 2.2, we present a reformulation of the SCL decoding algorithm in the log-likelihood ratio (LLR) domain, which greatly improves the numerical stability of the SCL decoding algorithm while also decreasing the logic and memory requirements. Moreover, we describe a hardware architecture, that is based on the decoder described in Section 2.1, which exploits the reformulation of SCL decoding in the LLR domain. We also study some properties of the LLR-

based path metrics that are then used in order to greatly improve the hardware implementation of the crucial path selection step of SCL decoding. In Section 2.3, we demonstrate the benefits of the LLR-based formulation of SCL decoding in terms of both the area requirements and the maximum operating frequency of the implemented decoder.

Finally, in Section 2.4 we present a survey on hardware implementations of various decoders for polar codes, covering BP, SC, and SCL decoding. We outline the most important techniques used in the literature so far and we compare the resulting polar decoders with each other. Moreover, we provide an in-depth comparison of polar decoders with existing LDPC and Turbo decoders, both in terms of the error-correcting performance and in terms of the hardware efficiency. Finally, we conclude this section by identifying some interesting and important open problems in the field of hardware decoders for polar codes.

Chapter 3: Faulty Polar and LDPC Channel Decoders

In this chapter we study the performance of both polar and LDPC codes under various approximate computing scenarios. We note that this chapter is derived from our works of [11, 12, 13, 14].

More specifically, in Section 3.2 we propose and formalize a modified construction of polar codes whose goal is to reduce the decoding complexity under successive cancellation decoding by sacrificing the error-correcting performance in a systematic fashion. We show that the modified construction is an NP-hard optimization problem and we propose a greedy algorithm to construct polar codes with large blocklengths. Finally, we demonstrate that, with the proposed code construction method, meaningful performance-complexity trade-offs can be achieved.

In Section 3.3, we study SC decoding of polar codes over the binary erasure channel (BEC) in the case where the memories that are used within the decoder are not fully reliable due to various possible reasons, including manufacturing defects and voltage scaling for power reduction. To this end, we introduce a memory fault model and we show that polarization does not happen in faulty SC decoding in the sense that all synthetic channels become asymptotically fully noisy. Moreover, we generalize an existing lower bound on the frame erasure rate (FER) and we use it, along with a well-known upper bound, in order to easily compute the FER-optimal polar code blocklength under faulty SC decoding. Finally, we present an unequal error-protection mechanism for the faulty memories that can significantly reduce the FER of finite-length polar codes with minimal overhead, while also re-enabling fully reliable communication asymptotically when protecting only a fixed fraction of the total decoder memory.

Finally, in Section 3.4 we study min-sum (MS) decoding of low-density parity-check (LDPC) codes in the case where the memories that are used within the decoder are not fully reliable. To this end, we first prove that the usual density evolution (DE) analysis remains valid under

the considered fault model and we generalize the DE equations for MS decoding to the case of faulty MS decoding. Finally, we use the derived DE equations to quantify the effect of faulty memories on the convergence speed and on the threshold of the decoder and we also show that in the faulty decoding case using more quantization bits does not necessarily lead to better error-correcting performance.

Chapter 4: Hardware Decoders for Ultra High-Speed Decoding of LDPC Codes

In this chapter we are concerned with quantized message-passing decoding of LDPC codes. We note that this chapter is derived from our works of [15, 16].

In Section 4.1 we describe the method that we use to design custom quantized decoding algorithms for any given message quantization bit-width and we explore various design parameter trade-offs. The method can design the variable node and check node update rules based on an information-theoretic criterion. More specifically, the update rules are designed in a way that maximizes the mutual information between each outgoing message and its corresponding codeword bit. Moreover, in Section 4.3 we present a fully unrolled LDPC decoder hardware architecture that greatly benefits from the aforementioned custom decoding algorithms and that can achieve a decoding throughput of more than 1 Terabit per second.

1.2 Notation and Preliminaries

Throughout this thesis, lowercase boldface letters denote vectors. The elements of a vector $\boldsymbol{x}$ are denoted by x_i and $\boldsymbol{x}_l^m$ means the sub-vector $[x_l, x_{l+1}, \ldots, x_m]^T$ if $m \geq l$ and the null vector otherwise. If $\mathcal{I} = \{i_1, i_2, \ldots\}$ is an ordered set of indices, $\boldsymbol{x}_\mathcal{I}$ denotes the sub-vector $[x_{i_1}, x_{i_2}, \ldots]^T$. Sets are denoted using calligraphic letters. If $\mathcal{S}$ is a countable set, $|\mathcal{S}|$ denotes its cardinality. We use $\log(\cdot)$ and $\ln(\cdot)$ to denote the base-2 and the natural logarithm respectively. Random variables are denoted using capital letters and individual realizations of random variables are denoted using the corresponding lowercase letter. Vectors of random variables are denoted by uppercase boldface letters. Uppercase boldface letters also denote matrices, but the distinction between vectors of random variables and matrices is always clear from the context. We use $\mathbb{P}[A]$ to denote the probability of event A and $\mathbb{E}[X]$ to denote the expectation of the random variable X.

Let W denote a binary-input discrete memoryless channel (B-DMC) with input alphabet $\{0,1\}$, output alphabet $\mathcal{Y}$, and transition probabilities $W(y|x),\ x \in \{0,1\},\ y \in \mathcal{Y}$. Let W^N denote N independent uses of W. Two parameters that can be associated with any B-DMC W are the symmetric capacity

$$I(W) \triangleq \sum_{y \in \mathcal{Y}} \sum_{x \in \{0,1\}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)}, \tag{1.2}$$

and the Bhattacharyya parameter

$$Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)}, \tag{1.3}$$

which measure *rate* and *reliability*, respectively. The relation between $I(W)$ and $Z(W)$ is quantified as follows [6]. For any B-DMC W, we have

$$I(W) \geq \log \frac{2}{1+Z(W)}, \tag{1.4}$$

$$I(W) \leq \sqrt{1-Z(W)^2}. \tag{1.5}$$

This means that whenever $I(W)$ goes to 0, $Z(W)$ goes to 1, and whenever $I(W)$ goes to 1, $Z(W)$ goes to 0.

1.3 Polar Codes

In this section, we give an overview of the required background on the construction and decoding of polar codes. More specifically, we first describe the polarizing transformation introduced by Arıkan. Then, we explain how polar codes are constructed by exploiting the properties of the polarizing transformation, as well as how they can be efficiently decoded using various decoding algorithms.

1.3.1 Polarizing Transformation

The main idea behind polar codes is to use a *polarizing transformation* that converts N independent copies of some channel W into N *synthetic channels* which are either better or worse than the original channel W. In the limit of infinite blocklength, it can be shown that channels become either perfectly noiseless or completely noisy. A polar code is then constructed by only using the perfect channels to transmit information and freezing the input of the bad channels to some value that is known at both ends of the communication link. It can also be shown that the fraction of channels that become perfect converges to the mutual information $I(W)$ of the original channel W, meaning that polar codes are capacity achieving. In the following sections, we explain the polarizing transformation in a more formal fashion.

1.3.1.1 Single-Step Polarizing Transformation

Let W denote a binary input memoryless channel with input $u \in \{0,1\}$, output $y \in \mathcal{Y}$, and transition probabilities $W(y|u)$. Assume that we want to transmit two independent and uniformly distributed bits $\boldsymbol{u} = \begin{bmatrix} u_0 & u_1 \end{bmatrix}$ over two independent uses of W and let $\boldsymbol{y} = \begin{bmatrix} y_0 & y_1 \end{bmatrix}$ denote the corresponding noisy outputs. The conditional distribution of $\boldsymbol{y}$ given $\boldsymbol{u}$ is

$$W^2(\boldsymbol{y}|\boldsymbol{u}) \triangleq W(y_0|u_0)W(y_1|u_1). \tag{1.6}$$

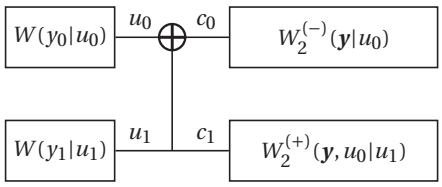


Figure 1.1 – One step of the polarizing transformation applied to two copies of the channel W and generating two synthetic channels $W^{(-)}$ and $W^{(+)}$.

The first step of the polarizing transformation proposed by Arıkan is to apply a linear encoding transformation to $\boldsymbol{u}$ as follows

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}_2, \quad \text{where} \quad \boldsymbol{G}_2 = \boldsymbol{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{1.7}$$

Assume that instead of transmitting $\boldsymbol{u}$, we transmit $\boldsymbol{c}$. In this case, the distribution of $\boldsymbol{y}$ conditioned on $\boldsymbol{u}$ is

$$W_2(\boldsymbol{y}|\boldsymbol{u}) \triangleq W^2(\boldsymbol{y}|\boldsymbol{c}) = W^2(\boldsymbol{y}|\boldsymbol{G}_2\boldsymbol{u}) = W(y_0|u_0 \oplus u_1)W(y_1|u_1). \tag{1.8}$$

The second and final step of the polarizing transformation is to split $W_2(\boldsymbol{y}|\boldsymbol{u})$ into two *synthetic channels* $W^{(-)}$ and $W^{(+)}$ out of $W_2(\boldsymbol{y}|\boldsymbol{u})$ as follows

$$W_2^{(-)}(\boldsymbol{y}|u_0) = \frac{1}{2}\left(W(y_0|u_0)W(y_1|0) + W(y_0|u_0 \oplus 1)W(y_1|1)\right), \tag{1.9}$$

$$W_2^{(+)}(\boldsymbol{y}, u_0|u_1) = \frac{1}{2}\left(W(y_0|u_0 \oplus u_1)W(y_1|u_1)\right). \tag{1.10}$$

In order to intuitively understand the polarizing transformation, imagine that a successive cancellation decoder is used, where u_0 is decoded by considering u_1 as noise, and then u_1 is decoded given a genie-aided decision on u_0. In such a decoder, the channels experienced by u_0 and u_1 are exactly $W_2^{(-)}$ and $W_2^{(+)}$, respectively. It can be shown that [6]

$$I\left(W_2^{(-)}\right) \leq I(W) \leq I\left(W_2^{(+)}\right), \tag{1.11}$$

with equality if and only if $I(W) = 0$ or $I(W) = 1$. In words, one synthetic channel is better than the original channel W with respect to the mutual information, while the other channel is worse than the original channel W. Furthermore, it can be shown that [6]

$$I\left(W_2^{(-)}\right) + I\left(W_2^{(+)}\right) = 2I(W), \tag{1.12}$$

meaning that that total mutual information is preserved by the polarizing transformation.

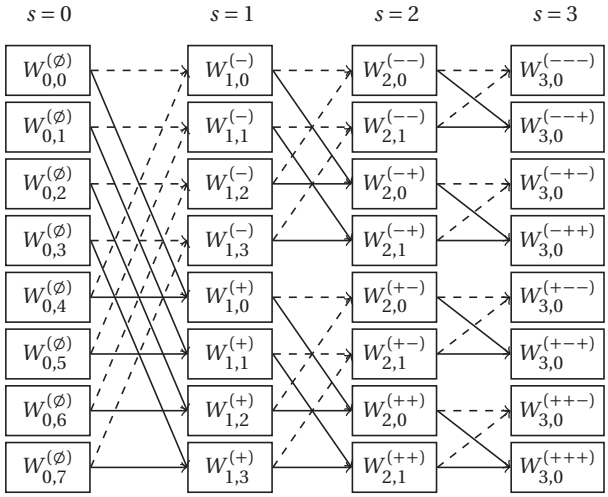


Figure 1.2 – Synthetic channel construction for a polar code of length $N = 2^3 = 8$. Pairs of solid lines represent the + transformation and pairs of dashed lines represent the − transformation.

1.3.1.2 General Polarizing Transformation

The single-step transformation described in the previous section can be generalized to n steps as follows. At step 1 of the polarizing transformation, $N = 2^n$ independent copies of the original channel W, denoted by $W_{0,k}^{(\emptyset)},\ k = 0,\ldots,N-1$, are combined pair-wise in order to generate $N/2$ independent copies of a pair of new synthetic channels denoted by $W_{1,k}^{(+)}$ and $W_{1,k}^{(-)},\ k = 0,\ldots,N/2-1$. As we explained in the previous section, the "+" channels can be shown to be better than the original channel W, while the "-" channels are worse than the original channel W. The same transformation is applied to $W_{1,k}^{(+)}$ and $W_{1,k}^{(-)},\ k = 0,\ldots,N/2-1$ in order to generate $N/4$ independent copies of $W_{2,k}^{(++)}$, $W_{2,k}^{(+-)}$, $W_{2,k}^{(-+)}$ and $W_{2,k}^{(--)},\ k = 0,\ldots,N/4-1$. This procedure is repeated for a total of n steps, until $N = 2^n$ independent channels $W_{n,0}^{(\mathbf{s})},\ \mathbf{s} \in \{+,-\}^n$, are generated. Note that, in general, the notation $W_{s,k}^{(\mathbf{s})}$ implies that $|\mathbf{s}| = s$. An example of the transformation steps is depicted in Figure 1.2 for $n = 3$.

The linear encoding transformation of (1.7) that is applied to $\boldsymbol{u}$ in order to obtain $\boldsymbol{c}$ can be generalized to

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}_N, \quad \text{where} \quad \boldsymbol{G}_N = \boldsymbol{F}^{\otimes n}\boldsymbol{B}_N. \tag{1.13}$$

We note that $\boldsymbol{A}^{\otimes n}$ denotes the n-fold Kronecker product of the matrix $\boldsymbol{A}$ and $\boldsymbol{B}_N$ is a bit-reversal permutation matrix.[2]An example of an encoding circuit for $N = 8$ is shown in Figure 1.3. It can be verified that the circuit contains exactly $N\log N$ nodes and each node needs to be activated once in order to implement the encoding operation $\boldsymbol{u}\mathbf{G}_N$, meaning that encoding can be performed with complexity $O(N\log N)$ [6].

[2]Let $\boldsymbol{v}$ and $\boldsymbol{u}$ be two length $N = 2^n$ vectors and index their elements using binary sequences of length n, $(b_1, b_2, \ldots, b_n) \in \{0,1\}^n$. Then $\boldsymbol{v} = \boldsymbol{B}_N\boldsymbol{u}$ iff $v_{(b_1,b_2,\ldots,b_n)} = u_{(b_n,b_{n-1},\ldots,b_1)}$ for $\forall (b_1, b_2, \ldots, b_n) \in \{0,1\}^n$.

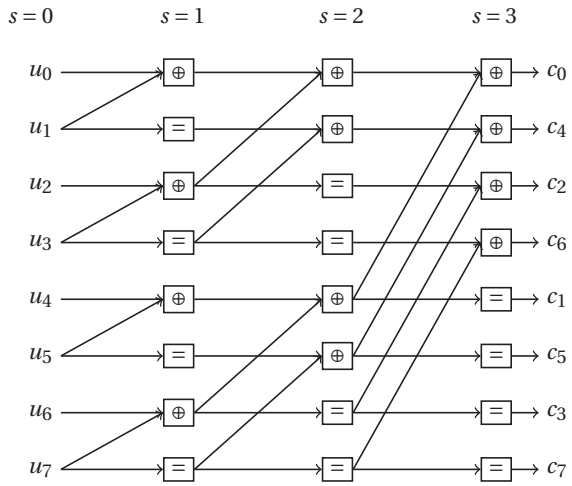


Figure 1.3 – Implementation of $\boldsymbol{F}^{\otimes n}$ for a polar encoder of length $N = 2^3 = 8$. The application of $\boldsymbol{B}_N$ to the result of this encoding circuit would simply re-arrange the output in the natural ordering, i.e., $\boldsymbol{c} = [\, c_0\, c_1\, \ldots\, c_7\,]$.

Arıkan showed that as $n \to \infty$, these synthetic channels *polarize* to 'easy-to-use' B-DMCs [6, Theorem 1]. That is, all except a vanishing fraction of them will be either almost-noiseless channels (whose output is almost a deterministic function of the input) or useless channels (whose output is almost statistically independent of the input). Furthermore, the fraction of almost-noiseless channels is equal to the symmetric capacity of the underlying channel, i.e., the highest rate at which reliable communication is possible through W when the input letters $\{0,1\}$ are used with equal probability [1].

1.3.2 Construction of Polar Codes

Let us define a mapping from $\mathbf{s} \in \{+,-\}^n$ to the integer-valued indices $i \in \{0,\ldots,2^n-1\}$ as follows. First, we construct $\mathbf{b}$ by replacing each $-$ that appears in $\mathbf{s}$ with a 0 and each $+$ that appears in $\mathbf{s}$ with a 1. Then, the index i can be obtained by considering $\mathbf{b}$ as a left-MSB binary representation of i. As this mapping is a bijection, we use $\mathbf{s}$ and i interchangeably. For example, $W_{n,0}^{(--+)}$ and $W_{n,0}^{(1)}$ denote the same synthetic channel. Moreover, when referring to any channel $W_{n,0}^{(i)}$ at stage n of the synthetic channel construction process, we can skip the second subscript for simplicity, since it is identical for all channels, and we can simply write $W_n^{(i)}$ instead of $W_{n,0}^{(i)}$.

Let us fix a blocklength $N = 2^n$ and a code rate $R \triangleq \frac{K}{N}$, $0 < K < N$. Moreover, let $\mathcal{A}$ denote the set of the K channel indices i (equivalently, strings $\mathbf{s}$), that correspond to the K best synthetic channels $W_n^{(i)}$. A polar code of rate R is constructed by transmitting the information vector $\mathbf{u}_\mathcal{A}$ over the K best synthetic channels, while freezing the inputs of the remaining synthetic

channels, i.e., $\mathbf{u}_{\mathcal{A}^c}$, to a value that is known at the receiver.[3] This is equivalent to transmitting the encoded codeword $\mathbf{x} = \mathbf{u}\mathbf{G}_N$ over N independent uses of the initial channel W.

An important issue is how the quality of each synthetic channel can be assessed in order to select the K best channels. For the special case where the original channel W is a binary erasure channel (BEC), both the mutual information and the Bhattacharyya parameters of the the synthetic channels can be calculated analytically using a simple recursive formula [6]. We provide more details on this recursive formula in Section 3.3. For more general channels, in his original paper Arıkan proposed a Monte Carlo based approach to estimate the Bhattacharyya parameters of the synthetic channels [6]. This approach can work for any channel W in principle, but its complexity can be quite high depending on the desired level of accuracy of the Bhattacharyya parameter estimates. More sophisticated methods to construct polar codes, which rely on approximating the synthetic channels in order to efficiently calculate the Bhattacharyya parameters were considered in [17, 18, 19].

1.3.3 Decoding of Polar Codes

In this section, we describe the main decoding algorithms for polar codes. More specifically, in Section 1.3.3.1 we describe the original successive cancellation (SC) decoding algorithm proposed by Arıkan, in Section 1.3.3.2 we describe an improvement of SC decoding called successive cancellation list (SCL) decoding, while in Section 1.3.3.3 we outline belief propagation (BP) decoding of polar codes. Finally, we briefly mention some alternative decoding algorithms in Section 1.3.3.4.

1.3.3.1 Successive Cancellation Decoding

Successive cancellation (SC) decoding is the most basic decoding algorithm for polar codes, which was introduced by Arıkan in his original work [6]. As the name implies, SC decoding takes successive decisions on the information bits. More specifically, the receiver observes the channel output vector $\boldsymbol{y}$ and estimates the elements of the $\boldsymbol{u}_{\mathcal{A}}$ *successively* as follows: Suppose the information indices are ordered as $\mathcal{A} = \{i_1, i_2, \ldots, i_{NR}\}$ (where $i_j < i_{j+1}$). Having the channel output, the receiver has all the required information to decode the input of the synthetic channel $W_n^{(i_1)}$ as $\hat{u}_{i_1}$, since $\boldsymbol{u}_0^{i_1-1}$ is a part of the frozen sub-vector $\boldsymbol{u}_{\mathcal{F}}$. Since this synthetic channel is assumed to be almost-noiseless by construction, we have $\hat{u}_{i_1} = u_{i_1}$ with high probability. Subsequently, the decoder can proceed to index i_2 as the information required for decoding the input of $W_n^{(i_2)}$ is now available. Once again, this estimation is with high probability error-free. As described in Algorithm 1 on a high level, this process is continued until all the information bits have been estimated.

In fact, SC decoding can be viewed as a greedy depth-first search algorithm on a full binary

[3] For symmetric channels, $\mathbf{u}_{\mathcal{A}^c}$ can be the all-zero vector. For asymmetric channels, the choice of $\mathbf{u}_{\mathcal{A}^c}$ may have an impact on the performance of the code [6].

Algorithm 1: SC Decoding [6].

1 **for** $i = 0, 1, \ldots, N-1$ **do**
2 **if** $i \notin \mathcal{A}$ **then** // known frozen bits
3 $\hat{u}_i \leftarrow u_i$;
4 **else** // information bits
5 $\hat{u}_i \leftarrow \arg\max_{u_i \in \{0,1\}} W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | u_i)$;
6 **return** $\hat{\boldsymbol{u}}_{\mathcal{A}}$;

tree. To see this, let

$$\mathcal{U}(\boldsymbol{u}_{\mathcal{F}}) \triangleq \{\boldsymbol{v} \in \{0,1\}^N : \boldsymbol{v}_{\mathcal{F}} = \boldsymbol{u}_{\mathcal{F}}\} \tag{1.14}$$

denote the set of 2^{NR} possible length-N vectors that the transmitter can send. The elements of $\mathcal{U}(\boldsymbol{u}_{\mathcal{F}})$ are in one-to-one correspondence with 2^{NR} leaves of a binary tree of height N. These leaves are constrained to be reached from the root by following the direction u_i at all levels $i \in \mathcal{F}$. Therefore, any decoding procedure is essentially equivalent to picking a *path* from the root to one of these leaves on the binary tree.

In particular, an optimal maximum-likelihood (ML) decoder, associates each path with its likelihood (or any other *path metric* which is a monotone function of the likelihood) and picks the path that maximizes this metric by exploring *all* possible paths

$$\hat{\boldsymbol{u}}_{\mathrm{ML}} = \arg\max_{\boldsymbol{v} \in \mathcal{U}(\boldsymbol{u}_{\mathcal{F}})} W_n(\boldsymbol{y} | \boldsymbol{v}). \tag{1.15}$$

Clearly such an optimization problem is computationally infeasible as the number of paths, $|\mathcal{U}(\boldsymbol{u}_{\mathcal{F}})|$, grows exponentially with the blocklength N.

The SC decoder, in contrast, finds a sub-optimal solution by maximizing the likelihood via a *greedy* one-time-pass through the tree: starting from the root, at each level $i \in \mathcal{A}$, the decoder extends the existing path by picking the child that maximizes the *partial likelihood* $W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | u_i)$.

SC Decoding Complexity The computational task of the SC decoder is to calculate the pairs of likelihoods $W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | u_i),\ u_i \in \{0,1\}$, needed for the decisions in line 5 of Algorithm 1. Since the decisions are binary, it is sufficient to compute the *decision log-likelihood ratios (LLRs)*,

$$\mathsf{LLR}_n^{(i)} \triangleq \ln\left(\frac{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | 0)}{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | 1)}\right), \qquad i \in \{0, \ldots, N-1\}. \tag{1.16}$$

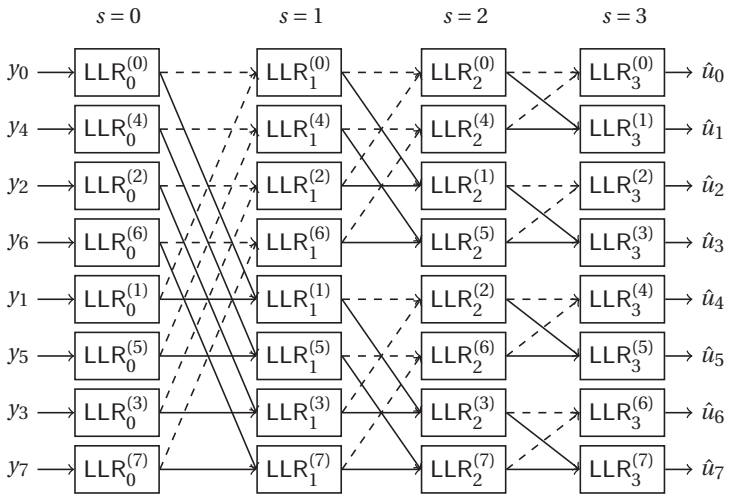


Figure 1.4 – The data dependency graph (DDG) for the computation of the LLRs of a polar code with $N = 2^3 = 8$. Solid lines represent application of f_+ and dashed lines represent application of f_-. The partial sums required for the f_+ updates can be computed by using the encoder structure of Figure 1.3 and setting $u_i = \hat{u}_i$ once each estimate becomes available.

It can be shown (see [6, Section VII] and [20]) that the decision LLRs (1.16) can be computed via the recursions,

$$\mathsf{LLR}_s^{(2i)} = f_-\big(\mathsf{LLR}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{LLR}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}\big), \tag{1.17}$$

$$\mathsf{LLR}_s^{(2i+1)} = f_+\big(\mathsf{LLR}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{LLR}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}, \mathsf{u}_s^{(2i)}\big), \tag{1.18}$$

for $s = n, n-1, \ldots, 1$, where $f_- : \mathbb{R}^2 \to \mathbb{R}$ and $f_+ : \mathbb{R}^2 \times \{0,1\} \to \mathbb{R}$ are defined as

$$f_-(\alpha, \beta) \triangleq \ln\Big(\frac{e^{\alpha+\beta}+1}{e^{\alpha}+e^{\beta}}\Big), \tag{1.19a}$$

$$f_+(\alpha, \beta, u) \triangleq (-1)^u \alpha + \beta, \tag{1.19b}$$

respectively. The recursions terminate at $s = 0$ where

$$\mathsf{LLR}_0^{(i)} \triangleq \ln\Big(\frac{W(y_i|0)}{W(y_i|1)}\Big), \qquad \forall\, i \in \{0, \ldots, N-1\},$$

are *channel LLRs*. The *partial sums* $\mathsf{u}_s^{(i)}$ are computed starting from $\mathsf{u}_n^{(i)} \triangleq \hat{u}_i,\ \forall\, i \in \{0, \ldots, N-1\}$ and setting

$$\mathsf{u}_{s-1}^{(2i-[i \bmod 2^{s-1}])} = \mathsf{u}_s^{(2i)} \oplus \mathsf{u}_s^{(2i+1)},$$
$$\mathsf{u}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])} = \mathsf{u}_s^{(2i+1)},$$

for $s = n, n-1, \ldots, 1$, which is equivalent to a step-wise encoding of the vector $\hat{\boldsymbol{u}}$. In essence, the SC decoding algorithm consists of a forward step in which the LLRs are updated, and a

feedback part in which the partial sums are updated. The data dependency graph (DDG) for an SC polar decoder with $N = 8$ is shown in Figure 1.4. We note that, while Figure 1.4 may suggest that each level can be processed in parallel similarly to the implementation of the fast Fourier transform, in reality this is not the case due to the hidden dependencies stemming from the feedback part of the decoder.

The entire set of $N \log N$ LLRs $\mathsf{LLR}_s^{(i)}, s \in \{1, \ldots, n\}, i \in \{0, \ldots, N-1\}$, can be computed using $O(N \log N)$ updates since from each pair of LLRs at *stage s*, a pair of LLRs at stage $s+1$ is calculated using f_- and f_+ update rules (see Figure 1.4). Additionally the decoder must keep track of $N \log N$ partial sums $\mathsf{u}_s^{(i)}, s \in \{0, \ldots, n-1\}, i \in \{0, \ldots, N-1\}$, and update them after decoding each bit $\hat{u}_i$, which is also achievable using $O(N \log N)$ updates.

Remark. While the update rule f_+ given in (1.19b) is simple to implement in hardware, the exact update rule f_- given in (1.19a) is much more involved. To this end, in all hardware implementations f_- is approximated as

$$f_-(\alpha, \beta) \approx \operatorname{sign}(\alpha) \operatorname{sign}(\beta) \min\{|\alpha|, |\beta|\}. \tag{1.20}$$

This approximation is a hardware-friendly function as it involves only the easy-to-implement $\min\{\cdot, \cdot\}$ operation (compared to f_- which involves exponentiation and logarithms). This approximation is called the min-sum (MS) approximation and it is also very commonly used in message-passing decoding of low-density parity-check (LDPC) codes (cf. Section 1.4).

1.3.3.2 Successive Cancellation List Decoding

The *successive cancellation list (SCL)* decoding algorithm, introduced in [21], converts the greedy one-time-pass search of SC decoding into a breadth-first search under a complexity constraint in the following way: At each level $i \in \mathcal{A}$, instead of extending the path in only one direction, the decoder is duplicated into two parallel *decoding threads* continuing in both possible directions. However, in order to avoid the exponential growth of the number of decoding threads, as soon as the number of parallel decoding threads reaches L, at each step $i \in \mathcal{A}$, only L threads corresponding the L most likely paths (out of $2L$ tentative paths) are retained. We note that, although it is not necessary, L is usually a power of 2. The decoder eventually finishes with a *list* of L candidates $\hat{\boldsymbol{u}}[\ell]$, $\ell \in \{0, \ldots, L-1\}$, corresponding to L (out of 2^{NR}) paths on the binary tree and declares the most likely of them as the final estimate. This procedure is formalized in Algorithm 2. Simulation results in [21] show that for a $(2048, 1024)$ polar code, a list size of $L = 32$ is sufficient to have a close-to-ML block-error probability.

We note that in [21] SCL decoding is described in terms of the synthetic channel likelihoods $W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell] | u), \forall \ell \in \mathcal{L}, u \in \{0, 1\}$. These likelihoods can be computed using a recursion whose schedule is identical to the one that we described in Section 1.3.3.1 to compute the

Algorithm 2: SC List Decoding [21]

$\mathcal{L} \leftarrow \{0\}$; // start with a single active thread
for $i = 0, 1, \ldots, N-1$ **do**
  if $i \notin \mathcal{A}$ **then** // known frozen bits
    $\hat{u}_i[\ell] \leftarrow u_i$ for $\forall \ell \in \mathcal{L}$;
  else // information bits
    if $|\mathcal{L}| < L$ **then** // duplicate all the threads
      foreach $\ell \in \mathcal{L}$ **do**
        `duplicatePath(`ℓ`);`
    else
      Compute $P_{\ell,u} = W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell] | u)$, for $\forall \ell \in \mathcal{L}$ and $\forall u \in \{0,1\}$;
      $\tau \leftarrow$ the median of $2L$ numbers $P_{\ell,u}$;
      foreach $\ell \in \mathcal{L}$ *such that* $P_{\ell,0} < \tau$ *and* $P_{\ell,1} < \tau$ **do**
        Kill the thread ℓ and set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\ell\}$;
      for $\ell \in \mathcal{L}$ **do**
        if $P_{\ell,u} > \tau$ *while* $P_{\ell,u\oplus 1} < \tau$ **then**
          $\hat{u}_i[\ell] \leftarrow u$;
        else // both $P_{\ell,0}$ and $P_{\ell,1}$ are $\geq \tau$
          `duplicatePath(`ℓ`);`
$\ell^* \leftarrow \arg\max_{\ell \in \mathcal{L}} W_n^{(N-1)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{N-1}[\ell] | \hat{u}_N[\ell])$;
return $\hat{\boldsymbol{u}}_{\mathcal{A}}[\ell^*]$;
subroutine `duplicatePath(`ℓ`)`
  Copy the thread ℓ into a new thread $\ell' \notin \mathcal{L}$;
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell'\}$;
  $\hat{u}_i[\ell] \leftarrow 0$;
  $\hat{u}_i[\ell'] \leftarrow 1$;

LLRs, but the update rules are different. More specifically, we have

$$\mathsf{L}_s^{(2i)} = f_-\big(\mathsf{L}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{L}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}\big), \tag{1.21}$$

$$\mathsf{L}_s^{(2i+1)} = f_+\big(\mathsf{L}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{L}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}, \mathsf{u}_s^{(2i)}\big), \tag{1.22}$$

where $\mathsf{L}_s^{(i)}$ are *pairs* of likelihoods, i.e.,

$$\mathsf{L}_s^{(i)} = \Big(W_s^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}|0), W_s^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}|1)\Big), \tag{1.23}$$

and

$$f_-(a,b) \triangleq \left(\frac{1}{2}\left(a(0)b(0)+a(1)b(1)\right), \frac{1}{2}\left(a(1)b(0)+a(0)b(1)\right)\right), \tag{1.24}$$

$$f_+(a,b,u) \triangleq \left(\frac{1}{2}a(u)b(0), \frac{1}{2}a(1-u)b(1)\right). \tag{1.25}$$

We note that these two update rules correspond exactly to the transition probabilities of the $W^{(-)}$ and $W^{(+)}$ channels created by the one-step polarizing transformation in (1.9)–(1.10). The recursions terminate at $s = 0$ where

$$\mathsf{L}_0^{(i)} \triangleq \big(W(y_i|0), W(y_i|1)\big), \qquad \forall\, i \in \{0,\dots,N-1\},$$

are the *channel likelihoods*. The computation of the partial sums for each path is identical to Section 1.3.3.1.

While a naive implementation of SCL decoder would have a decoding complexity of at least $\Omega(L\cdot N^2)$ due to $\Theta(L\cdot N)$ *duplications* of data structures of size $\Omega(N)$ in lines 8 and 18 of Algorithm 2, a clever choice of data structures together with the recursive nature of computations enables the authors of [21] to use a copy-on-write mechanism and implement the decoder in $O(L\cdot N\log N)$ complexity.

CRC-Aided Successive Cancellation List Decoder It was observed in [21] that when the SCL decoder fails, in most of the cases, the correct path (corresponding to $\boldsymbol{u}_{\mathcal{A}}$) is among the L paths the decoder has ended up with. The decoding error only happens because there exists another candidate path that is more likely and is thus selected in line 19 of Algorithm 2 (note that in such situations the ML decoder would also fail). They, hence, conclude that the performance of polar codes would be significantly improved if the decoder were assisted for its final choice.

One way of assisting the SCL decoder is by adding r more non-frozen bits (i.e., creating a polar code of rate $R + r/N$ instead of rate R) to the underlying polar code and then setting the last r non-frozen bits to an r-bit CRC of the first NR information bits (note that the *effective* information rate of the code is unchanged). The SCL decoder, at line 19, first discards the paths that do not pass the CRC and then chooses the most likely path among the remaining ones. Since the CRC can be computed very efficiently [22, Chapter 7], this measure does not

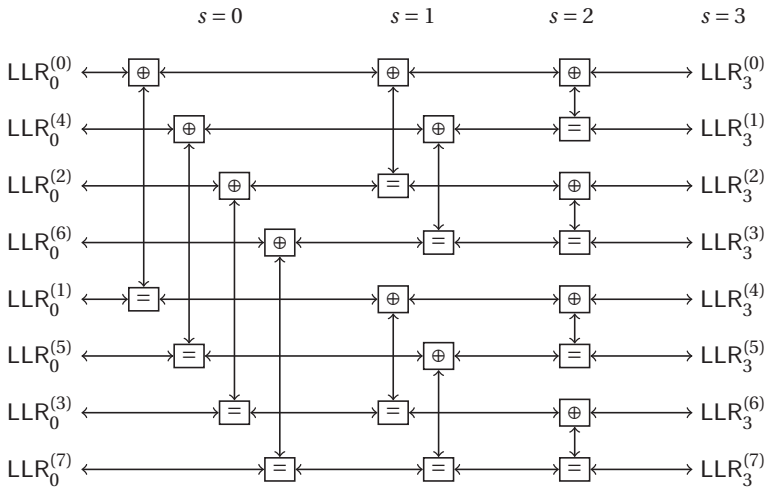


Figure 1.5 – Factor graph for belief propagation decoding of polar codes.

notably increase the computational complexity of the decoder. This modification is called the CRC-aided SCL (CA-SCL) decoder. The empirical results of [21] show that a $(2048, 1024)$ concatenated polar code with a 16-bit CRC decoded using a list decoder with list size of $L = 32$, outperforms the existing state-of-the-art WiMAX $(2304, 1152)$ LDPC code [23].

1.3.3.3 Belief Propagation Decoding

Even though SC and SCL decoding are very structured and have low decoding complexity, they have certain drawbacks. First, both SC and SCL decoding produce hard outputs (i.e., binary decisions) and cannot easily be used in an effective way in iterative receivers [24]. Moreover, it is difficult to highly parallelize SC and SCL decoding due to their serial nature. Thus, achieving high decoding throughput with SC and SCL decoders is challenging and quite sophisticated methods need to be employed, as we will explain in more detail in Section 2.4. A simple way of achieving both high decoding throughput and producing soft outputs for iterative receivers is to use belief propagation (BP) decoding on the polar code's factor graph, which can be derived from the encoding graph of Figure 1.3 [6] and is depicted in Figure 1.5.

The factor graph of Figure 1.5 contains two types of nodes, namely "=" nodes and "$\oplus$" nodes. Due to their similarity with the nodes found in the Tanner graph of an LDPC code (cf. Section 1.4), we call these two types of nodes *variable nodes* (VNs) and *check nodes* (CNs), respectively. Each node has three edges connected to it and on each edge one incoming and one outgoing message are transmitted. More specifically, for the basic computational structure shown in Figure 1.6, there are four left-to-right (LR) messages, four right-to-left (RL) messages, and two internal (I) messages. Out of the four LR and RL messages, two are incoming and two are outgoing. The update rules to calculate these messages are identical to the update rules of a VN and a CN of degree three, as explained in Section 1.4. More specifically, using the

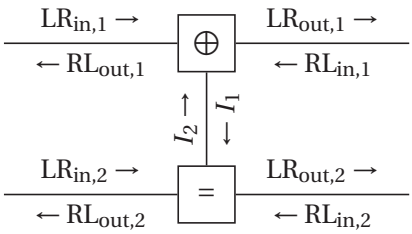


Figure 1.6 – Basic computation unit for belief propagation decoding.

min-sum approximation for the CN, we have

$$I_1 = f_-\left(\mathrm{LR}_{\mathrm{in},1}, \mathrm{RL}_{\mathrm{in},1}\right), \tag{1.26}$$
$$\mathrm{RL}_{\mathrm{out},1} = f_-\left(\mathrm{RL}_{\mathrm{in},1}, I_2\right), \tag{1.27}$$
$$\mathrm{LR}_{\mathrm{out},1} = f_-\left(\mathrm{LR}_{\mathrm{in},1}, I_2\right), \tag{1.28}$$

where

$$f_-(a,b) = \operatorname{sign} a \cdot \operatorname{sign} b \cdot \min(|a|,|b|). \tag{1.29}$$

For the VN, we have

$$I_2 = f_+\left(\mathrm{LR}_{\mathrm{in},2}, \mathrm{RL}_{\mathrm{in},2}\right), \tag{1.30}$$
$$\mathrm{RL}_{\mathrm{out},2} = f_+\left(\mathrm{RL}_{\mathrm{in},2}, I_1\right), \tag{1.31}$$
$$\mathrm{LR}_{\mathrm{out},2} = f_+\left(\mathrm{LR}_{\mathrm{in},2}, I_1\right), \tag{1.32}$$

where

$$f_+(a,b) = a + b. \tag{1.33}$$

One decoding iteration consists of the activation of all $N \log N$ nodes exactly once. Thus, the per-iteration decoding complexity of BP decoding is $O(N \log N)$, which is identical to the decoding complexity of SC decoding. However, several decoding iterations need to be performed in order to get useful error-correcting performance, so the overall decoding complexity of BP decoding is $O(\ell_{\max} N \log N)$, where $\ell_{\max}$ is the number of performed decoding iterations. The order of activation of the nodes is defined by the decoding schedule. Several different decoding schedules are possible, the most common being bi-directional (i.e., a left-to-right pass followed by a right-to-left pass over the nodes) and uni-directional (i.e., only left-to-right or right-to-left passes over the nodes).

1.3.3.4 Other Decoding Algorithms

A few other decoding algorithms for polar codes have been proposed in the literature. While these alternative decoding algorithms are not directly related to the present thesis, we still

believe they are noteworthy and we mention them briefly.

Soft cancellation (SCAN) decoding [25] is an SC-based soft-output decoding algorithm. The main idea behind SCAN decoding is to replace the hard-decision feedback part of the SC decoder with a soft-decision feedback part. In essence, the SCAN decoder is a BP decoder with an SC decoding schedule. Compared with BP decoding, SCAN decoding requires significantly fewer iterations to converge and thus has a lower average computational complexity. On the other hand, due to the serial decoding schedule, SCAN decoding is not as highly parallelizable as BP decoding, making the implementation of high-throughput hardware decoders challenging. As SCAN decoding has very similar error-correcting performance to SC decoding, for the hardware comparison we group SCAN decoders together with SC decoders. However, it is important to note that SCAN decoding naturally generates soft outputs for iterative decoding.

Successive cancellation stack (SCS) decoding [26] is similar to SCL decoding in the sense that it follows multiple paths on the decoding tree. However, the tree search strategy is different, as only the path with the best metric (i.e., highest likelihood) is expanded at each step, instead of all L paths simultaneously. The error correcting performance of SCS decoding can be close to that of SCL decoding, but with much lower average computational complexity. Unfortunately, the variable runtime and the large memory requirements of SCS decoding reduce its attractiveness for hardware implementation.

Successive cancellation flip (SCF) decoding [27] is a CRC-aided tree search decoding algorithm for polar codes which, similarly to SCL and SCS decoding, explores multiple decoding paths. The main motivation behind SCF decoding is the observation that for most erroneous codewords a single bit is in error. SCF decoding first performs standard SC decoding once and then uses a CRC to detect (with high probability) whether the decoded codeword was correct or not. If the decoded codeword was correct, decoding halts. If the decoded codeword was not correct, standard SC decoding re-starts but one of the T least reliable bit decisions is flipped over a maximum of T decoding restarts (or iterations). At each iteration, the CRC is used to check whether the decoded codeword was correct. SCF decoding has similar computational complexity to SCS decoding and worse error correcting performance, but much lower memory requirements. Moreover, similarly to SCS decoding, SCF decoding has a variable runtime which may make it unattractive for hardware implementation as the hardware would still need to be tailored to also accommodate for the worst case execution times.

Sphere decoding of polar codes [28] is another tree-search decoding algorithm, which is inspired by the MIMO detection algorithm of the same name. The main idea behind sphere is decoding is that a complexity-constrained tree search is performed in order to find all candidate codewords that lie within a multi-dimensional sphere of radius r (the distance is commonly measured using the Euclidean distance metric). A combination of sphere decoding with SCL decoding was also proposed in [29].

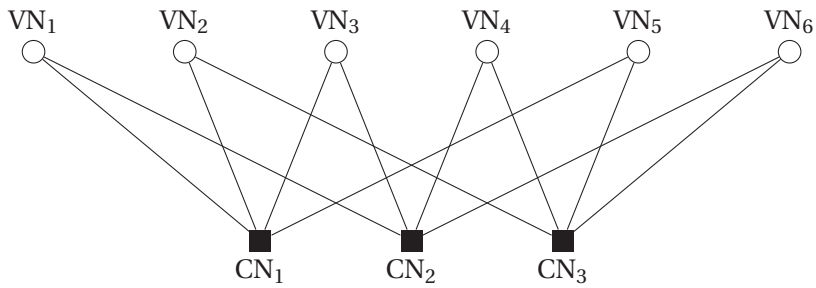


Figure 1.7 – Example of a Tanner graph for a $(2,4)$-regular LDPC code of blocklength $N = 6$.

1.4 LDPC Codes

LDPC codes are linear block codes that were first introduced by Gallager in 1962 [4]. At the time, however, the decoding complexity of LDPC codes was considered too high to be of practical interest and they were forgotten for more than thirty years. LDPC codes were rediscovered by MacKay and Neal in 1997 [5] and, since then, they have been adopted by numerous communications standards, such as IEEE 802.3an (10 Gbps Ethernet) [30], IEEE 802.11n (Wi-Fi) [31], IEEE 802.11ad (WiGig) [32], and DVB-S2 (digital video broadcasting) [33], to name a few.

1.4.1 Construction of LDPC Codes

An LDPC code $\mathcal{C}$ of blocklength N is the set of $N \times 1$ codeword vectors

$$\mathcal{C} = \left\{ \boldsymbol{c} \in \{0,1\}^N \middle| \boldsymbol{H}\boldsymbol{c} = \boldsymbol{0} \right\}, \tag{1.34}$$

where all operations are performed modulo 2 and $\boldsymbol{H} \in \{0,1\}^{M \times N}$ is a matrix which is called the *parity-check matrix* of the LDPC code. The parity-check matrix $\boldsymbol{H}$ is sparse in the sense that the number of non-zero elements grows linearly with N (while the size of the matrix grows quadratically with N). The *design rate* of the code is given by $R = 1 - \frac{M}{N}$, and it is identical to the actual rate provided that $\boldsymbol{H}$ has full rank.

If $\boldsymbol{H}$ contains exactly d_v ones per column and exactly d_c ones per row, then the corresponding LDPC code is called a (d_v, d_c)-regular LDPC code. Better error-correcting performance can be achieved by designing *irregular* LDPC codes, where the column and row weights of the parity-check matrix are not constant. As in this thesis we only employ regular LDPC codes, we refer the interested reader to the work of [34] for more information on irregular LDPC codes.

The parity-check matrix $\boldsymbol{H}$ also forms an incidence matrix for a Tanner graph which contains N variable nodes (VNs) and M check nodes (CNs). VN n is connected to CN k if and only if $\boldsymbol{H}_{mn} = 1$. An example of a Tanner graph for a $(2,4)$-regular LDPC code with a blocklength of $N = 6$ is illustrated in Figure 1.7.

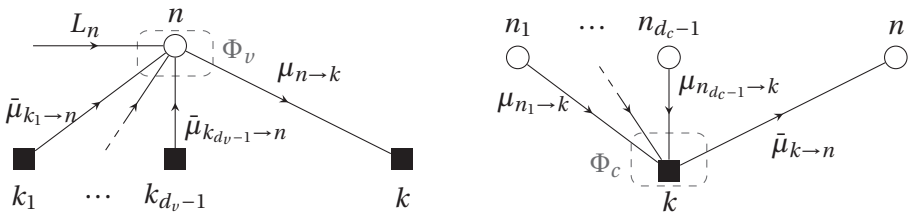


Figure 1.8 – (a) Variable node update for $\mathcal{N}(n) = \{k, k_1, \ldots, k_{d_v-1}\}$ and (b) check node update for $\mathcal{N}(k) = \{n, n_1, \ldots, n_{d_c-1}\}$.

1.4.2 Message-Passing Decoding of LDPC Codes

LDPC codes are traditionally decoded using message-passing (MP) algorithms, where information is exchanged between the VNs and the CNs of the Tanner graph over the course of several decoding iterations. There exist several ways to schedule the passing of these messages, but in this thesis we only focus on the standard *flooding* schedule, where one decoding iteration consists of the computation of all $d_v N$ VN-to-CN messages and N decision messages, followed by the computation of all $d_c M$ CN-to-VN messages. Due to the sparsity of the parity-check matrix, the number of messages that are exchanged over the Tanner graph edges grows linearly with the blocklength N, meaning that the decoding complexity of MP algorithms grows as $O(N)$. In this section, we first describe MP decoding in a very general way and then give examples of two particularly popular MP decoding algorithms, namely sum-product (SP) decoding and min-sum (MS) decoding.

In general, both the message update rules and the message alphabets may change from one decoding iteration to the next. Thus, we let the VN-to-CN and the CN-to-VN message alphabet at iteration ℓ be denoted by $\mathcal{M}^{(\ell)}$ and the channel LLR alphabet be denoted by $\mathcal{L}$. Moreover, at each iteration the messages from VN n to CN k are computed using a mapping which is defined as

$$\mu_{n\to k}^{(\ell)} = \Phi_v^{(\ell)}\big(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus k\to n}^{(\ell-1)}\big), \quad \forall\, n \in \{1,\ldots,N\},\ k \in \mathcal{N}(n), \tag{1.35}$$

where $\mathcal{N}(n)$ denotes the neighbors of node n in the Tanner graph, $\bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus k\to n}$ is a vector that contains the incoming messages from all neighboring CNs except k, and L_n denotes the channel LLR corresponding to VN n. For iteration $\ell = 1$, the vector $\bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\setminus k\to n}^{(0)}$ is the all-zero vector. Similarly, the CN-to-VN messages at iteration ℓ are computed using a mapping which is defined as

$$\bar{\mu}_{k\to n}^{(\ell)} = \Phi_c^{(\ell)}\big(\boldsymbol{\mu}_{\mathcal{N}(k)\setminus n\to k}^{(\ell)}\big), \quad \forall\, k \in \{1,\ldots,M\},\ n \in \mathcal{N}(k). \tag{1.36}$$

In addition to Φ_v and Φ_c, a third mapping Φ_d is needed to provide an estimate of the transmitted codeword bit based on the incoming check node messages and the channel LLR L_n

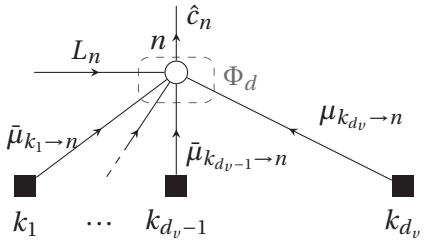


Figure 1.9 – Decision node update for $\mathcal{N}(n) = \{k, k_1, \ldots, k_{d_v}\}$.

$$\hat{c}_n^{(\ell)} = \Phi_d^{(\ell)}\big(L_n, \bar{\boldsymbol{\mu}}_{\mathcal{N}(n)\to n}^{(\ell)}\big), \quad n \in \{1, \ldots, N\}. \tag{1.37}$$

Figure 1.8 and Figure 1.9 illustrate the message updates in the Tanner graph (we have omitted the iteration index for clearer illustration). We note that different choices for the mappings (1.35)–(1.37) result in different MP decoding algorithms.

1.4.2.1 Sum-Product Decoding

For the case of the sum-product (SP) algorithm, which is asymptotically optimal with respect to the bit error rate, we have $\mathcal{L} = \mathcal{M}^{(\ell)} = \mathbb{R},\ \forall \ell = 1, 2, \ldots$, and the message update mappings read

$$\mu = \Phi_v^{\mathrm{SP}}(L, \bar{\boldsymbol{\mu}}) = L + \sum_i \bar{\mu}_i, \tag{1.38}$$

$$\bar{\mu} = \Phi_c^{\mathrm{SP}}(\boldsymbol{\mu}) = 2\tanh^{-1}\left(\prod_i \tanh\left(\frac{\mu_i}{2}\right)\right). \tag{1.39}$$

The decision mapping Φ_d is defined as

$$\hat{c} = \Phi_d^{\mathrm{SP}}(L, \bar{\boldsymbol{\mu}}) = \frac{1}{2}\left(1 - \operatorname{sign}\left(L + \sum_i \bar{\mu}_i\right)\right). \tag{1.40}$$

We note that we have omitted the iteration index ℓ in the update rules for simplicity since they do not change from one iteration to the next.

1.4.2.2 Min-Sum Decoding

For the min-sum (MS) algorithm, which is widely used in hardware implementations due to the simplicity of the CN update rule, we also have $\mathcal{L} = \mathcal{M}^{(\ell)} = \mathbb{R},\ \forall \ell = 1, 2, \ldots$ and the message

update mappings read

$$\mu = \Phi_v^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = L + \sum_i \bar{\mu}_i, \tag{1.41}$$

$$\bar{\mu} = \Phi_c^{\mathrm{MS}}(\boldsymbol{\mu}) = \left(\prod_i \operatorname{sign} \mu_i \right) \min |\boldsymbol{\mu}|, \tag{1.42}$$

where $\min|\boldsymbol{\mu}|$ denotes the minimum of the absolute values of the vector elements of $\boldsymbol{\mu}$. The decision mapping Φ_d is defined as

$$\hat{c} = \Phi_d^{\mathrm{MS}}(L, \bar{\boldsymbol{\mu}}) = \frac{1}{2}\left(1 - \operatorname{sign}\left(L + \sum_i \bar{\mu}_i\right)\right). \tag{1.43}$$

We note that we have omitted the iteration index ℓ in the update rules for simplicity since they do not change from one iteration to the next.

1.4.2.3 Quantized Min-Sum Decoding

In hardware implementations of LDPC decoders, the message alphabets $\mathcal{L}$ and $\mathcal{M}^{(\ell)}$ are usually chosen to be relatively small and they usually do not change over the course of the decoding iterations due to implementation complexity considerations, so for most implementations it is safe to assume that $\mathcal{L} = \mathcal{M}^{(1)} = \mathcal{M}^{(2)} = \ldots = \mathcal{M}$. A very common approach is to use uniform b-bit symmetric quantization for both the channel LLRs and the message LLRs, meaning that $|\mathcal{M}| = 2^b$. In actual LDPC decoder hardware implementations, $4 \leq b \leq 7$ are common values (see, e.g., [35] and references therein). A uniform quantizer is defined by

$$q_\Delta(x) = \operatorname{sign}(x)\Delta \left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor, \tag{1.44}$$

where Δ denotes the quantization step. Assume that the set of quantization levels $\mathcal{M} = \{m_0, m_1, \ldots, m_{2^b-1}\}$ is sorted so that $m_i < m_{i+1}, \forall\, i \in \{0, \ldots, 2^b - 2\}$. Then, the corresponding quantization intervals are

$$t_i = \left(\frac{m_{i-1} + m_i}{2}, \frac{m_i + m_{i+1}}{2} \right], \; i = 0, \ldots, 2^b - 2, \tag{1.45}$$

where $m_{-1} = -\infty$ and $m_{2^b-1} = +\infty$. LLR saturation is commonly used in order for the results of (1.35)–(1.37) to remain within the alphabet $\mathcal{M}$. Specifically, the results of (1.35)–(1.37) that are smaller than m_0 or larger than m_{2^b-2} are saturated to m_0 and m_{2^b-2}, respectively.

2 Hardware Decoders for Polar Codes

In his seminal work [6], Arıkan constructed the first class of error correcting codes with a systematic construction that can achieve the capacity of any symmetric binary-input discrete memoryless channel (B-DMC) with efficient encoding *and decoding* algorithms. In particular, Arıkan proposed a low-complexity successive cancellation (SC) decoder and proved that the block-error probability of *polar codes* under SC decoding vanishes as their blocklength goes to infinity. Several hardware architectures for SC decoding of polar codes have recently been presented in the literature [20, 36, 37, 38, 39, 40, 41]. The first SC decoder ASIC was presented in [42], and some simplifications of Arıkan's original SC decoding algorithm are studied in [43, 44, 45, 46].

Unfortunately, even though polar codes are asymptotically optimal, they do not perform well at low-to-moderate blocklengths. This is to a certain extent due to the sub-optimality of the SC decoding algorithm. To partially compensate for this sub-optimality, Tal and Vardy proposed the successive cancellation list (SCL) decoder (cf. Section 1.3.3.2) whose computational complexity is shown to scale identically to the SC decoder with respect to the blocklength [21].

The block-error probability of SCL decoding for polar codes can be improved even further if one uses *modified polar codes* [21, 26], which are constructed by concatenating a polar code with a cyclic redundancy check (CRC) code as an outer code. Adding the CRC increases neither the computational complexity of the encoder nor that of the SCL decoder by a notable amount, while reducing the block-error probability significantly, making the error-rate performance of the modified polar codes under SCL decoding comparable to the state-of-the-art LDPC codes [21]. In [47] an adaptive variant of the CRC-aided SCL decoding algorithm is proposed in order to further improve the block-error probability of modified polar codes while maintaining the average decoding complexity at a moderate level. The SCL decoding algorithm in [21] is described in terms of likelihoods, which is a completely valid high-level description, but it unfortunately makes the hardware implementation of SCL decoding uneconomic due to severe numerical stability problems.

2.1 LL-Based SCL Decoder

2.1.1 Likelihood Representation

As discussed in Section 1.3.3.1, SC decoding can be carried out in the log-likelihood ratio (LLR) domain because at each step of decoding the decisions are binary. LLRs provide reduced storage requirements, increased numerical stability, as well as simplified computations with respect to a likelihood based implementation. Hence, in most channel decoders they are essential for an efficient hardware realization. However, the original SCL decoder, in lines 10–18 of Algorithm 2, has to choose the L most likely children out of $2L$ children of L different parents (see [48, Figure 3] for an illustration). For the necessary comparisons the decision log-likelihood *ratios* $\mathsf{LLR}_n^{(i)}$ alone are not sufficient. For this reason, the original SCL decoding algorithm is described using likelihoods in [21]. In their software implementation, the authors of [21] explain that, in order to avoid underflows, at each intermediate step of the updates the likelihoods are scaled by a common factor such that $P_{\ell,u}$ in line 10 of Algorithm 2 is proportional to $W(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell]|u)$ [21].

For our initial SCL decoder hardware implementation, we propose to reformulate the SCL decoding algorithm in the log-likelihood (LL) domain. Using LLs provides improved numerical stability, leading to lower bit-width requirements for the hardware implementation, and, as we will show, it also simplifies the f_- and f_+ update rules. In particular, we use negative LLs, which, since all likelihoods are positive and smaller than 1, are always positive numbers and do not require a sign bit to make the binary representation more compact. Assuming BPSK-modulated transmission over an AWGN channel with noise variance σ^2, the negative channel LLs are

$$\mathsf{LL}_0^{(i)} = \big(-\ln W(y_i|0), -\ln W(y_i|1)\big) = \left(\frac{(y_i-1)^2}{2\sigma^2} + \ln\sqrt{2\pi\sigma^2}, \frac{(y_i+1)^2}{2\sigma^2} + \ln\sqrt{2\pi\sigma^2}\right) \tag{2.1}$$

The decision LLs can then be calculated recursively as

$$\mathsf{LL}_s^{(2i)} = f_-\big(\mathsf{LL}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{LL}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}\big), \tag{2.2}$$

$$\mathsf{LL}_s^{(2i+1)} = f_+\big(\mathsf{LL}_{s-1}^{(2i-[i \bmod 2^{s-1}])}, \mathsf{LL}_{s-1}^{(2^s+2i-[i \bmod 2^{s-1}])}, \mathsf{u}_s^{(2i)}\big), \tag{2.3}$$

where $\mathsf{LL}_s^{(i)}$ are *pairs* of log-likelihoods, i.e.,

$$\mathsf{LL}_s^{(i)} = \left(-\ln W_s^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}|0), -\ln W_s^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}|1)\right). \tag{2.4}$$

Using negative LLs, the update rules f_- and f_+ of (1.24) and (1.25) are re-written as

$$f_-(a,b) \triangleq (\min^*(a(0)+b(0), a(1)+b(1)), \min^*(a(1)+b(0), a(0)+b(1))), \tag{2.5}$$

$$f_+(a,b,u) \triangleq (a(u)+b(0), a(1-u)+b(1)), \tag{2.6}$$

where $\min^*(a,b) = \min(a,b) + \ln\left(1+e^{-|a-b|}\right)$. The computation of the partial sums for each

Algorithm 3: LL-based SCL Decoding

$\mathcal{L} \leftarrow \{0\}$; // start with a single active path
for $i = 0, 1, \ldots, N-1$ **do**
  if $i \notin \mathcal{A}$ **then** // known frozen bits
    $\hat{u}_i[\ell] \leftarrow u_i$ for $\forall \ell \in \mathcal{L}$;
  else // information bits
    if $|\mathcal{L}| < L$ **then** // duplicate all the paths
      foreach $\ell \in \mathcal{L}$ **do**
        duplicatePath(ℓ);
    else
      Compute $P_{\ell,u} = \ln\left(W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell]|u)\right)$, for $\forall \ell \in \mathcal{L}$ and $\forall u \in \{0,1\}$;
      $\tau \leftarrow$ the median of $2L$ numbers $P_{\ell,u}$;
      foreach $\ell \in \mathcal{L}$ *such that* $P_{\ell,0} > \tau$ *and* $P_{\ell,1} > \tau$ **do**
        Kill the path ℓ and set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\ell\}$;
      for $\ell \in \mathcal{L}$ **do**
        if $P_{\ell,u} > \tau$ *while* $P_{\ell,u\oplus 1} < \tau$ **then**
          $\hat{u}_i[\ell] \leftarrow u$;
        else // both $P_{\ell,0}$ and $P_{\ell,1}$ are $\geq \tau$
          duplicatePath(ℓ);
$\ell^* \leftarrow \arg\min_{\ell \in \mathcal{L}} \ln\left(W_n^{(N-1)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{N-1}[\ell]|\hat{u}_N[\ell])\right)$;
return $\hat{\boldsymbol{u}}_{\mathcal{A}}[\ell^*]$;
subroutine duplicatePath(ℓ)
  Copy the path ℓ into a new path $\ell' \notin \mathcal{L}$;
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell'\}$;
  $\hat{u}_i[\ell] \leftarrow 0$;
  $\hat{u}_i[\ell'] \leftarrow 1$;

path is identical to Section 1.3.3.1.

In order to simplify the hardware implementation, the f_- function of (2.5) is approximated by ignoring the $\ln(\cdot)$ term in the $\min^*$ function. Thus, f_- becomes

$$f_-(a,b) \approx (\min(a(0)+b(0), a(1)+b(1)), \min(a(1)+b(0), a(0)+b(1))), \tag{2.7}$$

which is easily implementable in hardware as it only involves additions and finding the minimum of two values. This approximation is known as the *max-log* approximation [49] when standard LLs are used. Since we use negative LLs, we call (2.7) the *min-log* approximation.

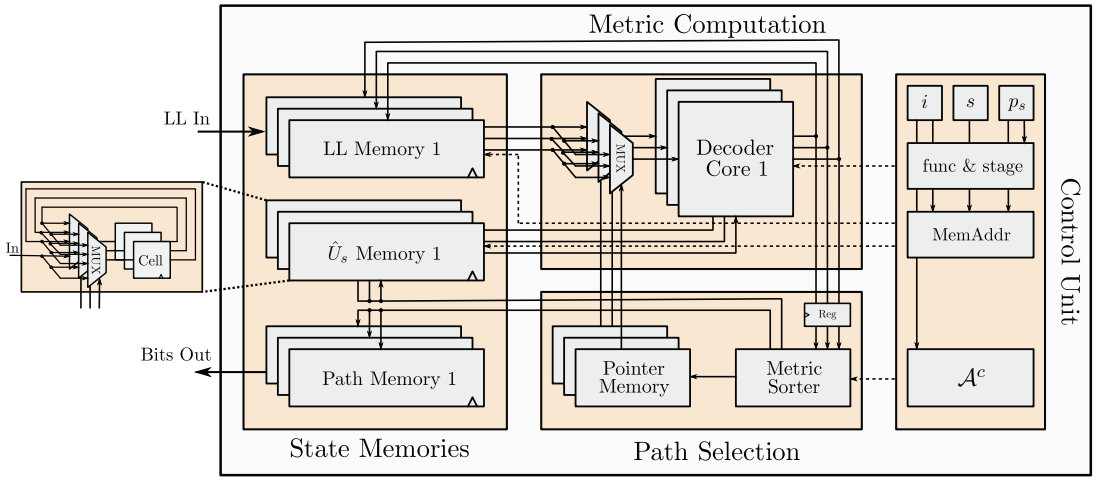


Figure 2.1 – High-level overview of the list SC decoder architecture.

Moreover, let $c, d > 0$ be constants. Then, for any $a, b \geq 0$, we have

$$\min(ca + d, cb + d) = c\min(a, b) + d, \tag{2.8}$$

$$(ca + d) + (cb + d) = c(a + b) + 2d. \tag{2.9}$$

In other words, with the approximation of (2.7) the update rules become linear, so we can ignore the additive and multiplicative constants in (2.1) without affecting the ordering of the path metrics. Thus, our decoder can safely use the following channel LLs

$$\mathsf{LL}_0^{(i)} = \left((y_i - 1)^2, (y_i + 1)^2\right), \tag{2.10}$$

which are significantly easier to handle by the quantization step due to their more limited dynamic range with respect to the original channel LLs of (2.1).

The proposed LL-based decoding algorithm is summarized in Algorithm 3. We note that, in general, LL-based SCL decoding differs from likelihood based SCL decoding as described in Algorithm 2 in only very few operations: Since the largest likelihoods (i.e., closest to 1) correspond to the smallest negative LLs (i.e., close to 0), the comparisons that are used to decide which paths to duplicate and which to discard in lines 11–18 as well as the choice of the most likely path in line 19 have to be inverted. Moreover, the path metric computation of line 10 also has to be adapted to the use of LLs.

2.1.2 List SC Decoder Architecture

For each of the L decoding paths, the intermediate LLs $\mathsf{LL}_s^{(i)},\ s = 1, \ldots, n,\ i = 0, \ldots, N-1$, the partial sums $\mathsf{u}_s^{(i)}, s = 0, \ldots, n,\ i = 0, \ldots, N-1$, and the path itself $\hat{\boldsymbol{u}}$ are stored in memories. We call these three memories collectively the *state-memories*. The content of each memory forms the *state* of each path. It was shown in [38] that by re-using memory positions, the $N \log N$

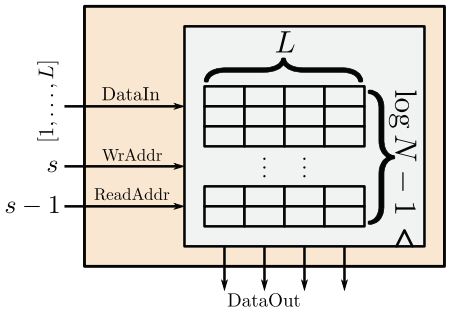
intermediate LLs produced for each path during SC decoding can be stored by only using approximately $2N$ memory positions, while using a similar re-use argument one can show that the partial sums can be stored using only approximately N memory positions. Finally, each of the L paths requires N memory positions. After the path selection step of line 11 of Algorithm 2, each of the initial L paths is either discarded, kept, or duplicated, depending on whether it has zero, one, or two child nodes in the set of L out of $2L$ largest metrics, respectively. In order to duplicate a path, in a straightforward implementation, its state is copied from one state-memory to another state-memory with some differences between the two copies that correspond to the two different choices for $\hat{u}_i$. It was shown in [21] that list SC decoding can be performed with complexity $O(LN\log N)$ when using a *lazy copy* technique.

While the lazy copy mechanism used in [21] is sufficient to ensure that the decoding complexity is $O(LN\log N)$, it is not ideal for a hardware implementation as it still requires copying the internal LLs, which is costly in terms of power, decoding latency, and silicon area. In our implementation we use an auxiliary *pointer memory* in order to improve this lazy copy technique. More specifically, our hardware architecture contains L physical LL memory banks for the intermediate LLs. Each decoder core $\ell \in \{1,\ldots,L\}$ always writes its output LLs to the same physical memory bank ℓ. However, when a path is split into two paths, both paths can read LLs from the same physical memory bank that corresponds to the parent path, while still writing their produced LLs to their own private memory banks. The pointers keep track of which physical memory bank each decoder core has to read from at each given stage of the DDG. This way, splitting a path is equivalent to copying some of the contents of this small pointer memory, instead of copying actual LLs from one physical memory bank to another.

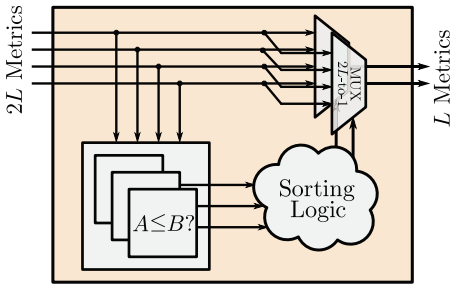
The proposed LL-based list SC decoder consists of three main components. The first component is the metric computation unit (MCU), which calculates the metrics for each path using the standard SC procedure. The second component, called the *state-memories* component, consists of L state-memories, which the MCU uses to compute the $2L$ path metrics. Moreover, a third component manages the tree search by performing *path selection* based on the metrics that are calculated by the MCU. An overview of the proposed list SC decoder architecture is shown in Figure 2.1. The MCU contains L *SC decoder cores*, which perform the metric calculation based on the state that they are supplied with. Multiplexers are responsible for redirecting the correct LLs to each decoder core, according to the entries of the pointer memory. The path selection unit contains a sorter (cf. Section 2.1.2.2) which finds the L best metrics out of $2L$ options, along with the path index and the value of $\hat{u}_i[\ell]$ from which they resulted, and the pointer memory, which manages the memory read access of the SC decoder cores.

2.1.2.1 LL Quantization

Since the LLs are positive numbers and (2.7) and (2.6) only involve additions, as SC decoding of a polar code of length $N = 2^n$ moves towards stage n, the dynamic range of the LLs increases. When an LL pair saturates, it is useless for making a decision meaning that, when using LLs, it is crucial to avoid saturation. In (2.7) and (2.6), two numbers with the same dynamic range



(a) Overview of the pointer memory architecture.



(b) Overview of the metric sorter architecture.

Figure 2.2 – Details of the proposed LL-based SCL decoder: (a) pointer memory, (b) metric sorter.

are added. The simplest way to avoid all saturations is to increase the number of bits used to store the LLs by one bit per stage. This way, the only performance degradation with respect to the floating point implementation comes from the quantization of the channel LLs. More specifically, let Q_{LL} denote the number of bits used for the quantization of the channel LLs. Using the LL quantization scheme described previously, we have $Q_{\max} = Q_{\text{LL}} + n$, where $Q_{\max}$ denotes the maximum LL bit-width that is required by the decoder.

2.1.2.2 Decoder Building Blocks

Metric Computation Unit The architecture of the SC decoder cores contained in the MCU is derived from the semi-parallel log-likelihood ratio (LLR) based architecture of [38], which was modified to implement the LL-based SC decoding update rules. Each decoder core consists of P processing elements (PEs) that operate on up to P nodes of each stage of the DDG in parallel. The stages of the DDG that contain more than P nodes are processed in parts over multiple clock cycles. The PEs implement both (2.5) and (2.6) simultaneously. An additional input is used to choose between the f_- and f_+ outputs. Due to the conservative choice of $Q_{\max}$, no overflow checks are needed in the PEs. The MCU contains L L-to-1 multiplexers, which are controlled by the pointer memory in the path selection unit and redirect the correct LLs to each SC decoder core. The maximum LL bit-width $Q_{\max}$ determines the bit-width of the arithmetic components within the PEs.

Control Unit The control unit is mainly responsible for generating the read and write addresses for the LL memory and for stalling the decoder cores for one clock cycle whenever an information bit is encountered in order to perform the path management step. For this reason, three counters track the index i of the bit that is currently being decoded, the current stage s within the decoding graph, and the current part within the stage p_s for the stages that require more than one cycle to be processed. All control signals and memory addresses are generated based on (i, s, p_s) and the set of frozen bits $\mathcal{A}^c$, exactly as in [38].

Memory Unit *LL Memory:* SC decoding can be implemented by storing $2(N-1)$ LL pairs [38], requiring a total of $4(N-1)$ data words. The N first pairs that correspond to the channel LLs are never overwritten during SC decoding. Thus, only one copy of the channel LL memory is needed, from which all decoder cores can read. The remaining $N-1$ memory position pairs have to be distinct for each path $\ell \in \{0,\ldots,L-1\}$, meaning that we need $L(N-1)$ distinct memory position pairs for the internal LLs. Thus, the total number of required memory position pairs is $(L+1)N-L$.

Path Memory: The path memory consists of L N-bit registers, denoted by $\hat{\boldsymbol{u}}[\ell]$, $\ell \in \{0,\ldots,L-1\}$. When a path ℓ needs to be duplicated, the contents of $\hat{\boldsymbol{u}}[\ell]$ are copied to $\hat{\boldsymbol{u}}[\ell']$, where ℓ' corresponds to an inactive path (cf. line 25 of Algorithm 4). The decoder is stalled for one clock cycle in order to perform the required copy operations by means of N $L \times L$ crossbars which connect each $\hat{\boldsymbol{u}}[\ell]$, $\ell \in \{0,\ldots,L-1\}$ with all other $\hat{\boldsymbol{u}}[\ell']$, $\ell' \in \{0,\ldots,L-1\}$. The copy mechanism is presented in detail in Figure 3, where we show how each memory bit-cell is controlled based on the results of the metric sorter. After path ℓ has been duplicated, one copy is extended with the bit value $\hat{u}_i[\ell] = 0$, while the other is updated with $\hat{u}_i[\ell'] = 1$.

Partial Sum Memory: The partial sum memory consists of L partial sum networks (PSNs), where each PSN is implemented as in [38]. When a path $\ell \in \{0,\ldots,L-1\}$ needs to be duplicated, the contents of the PSN ℓ are copied to another PSN ℓ', where ℓ' corresponds to an inactive path (cf. line 25 of Algorithm 4). Copying is performed in parallel with the copy of the path memory in a single clock cycle by using N $L \times L$ crossbars which connect each PSN $\ell \in \{0,\ldots,L-1\}$ with all other PSNs $\ell' \in \{0,\ldots,L-1\}$. If PSN ℓ was duplicated, one copy is updated with the bit value $\hat{u}_i[\ell] = 0$, while the other copy is updated with $\hat{u}_i[\ell'] = 1$. If a single copy of PSN ℓ was kept, then this copy is updated with the value of $\hat{u}_i[\ell]$ that corresponds to the surviving path.

Path Selection Unit For the path selection step, the $2L$ metrics are sorted in a single cycle. To minimize the delay, a radix-$2L$ sorter was implemented by extending the architecture presented in [50] to support finding of the L smallest values, instead of only the 2 smallest values. This sorter requires $2L(2L-1)/2$ comparators of $Q_{\max}$-bit quantities. Since a single sorter is needed, minimizing its size is not critical. The architecture of the metric sorter is presented in Figure 2.2(b).

Address Translation Unit The address translation unit contains a pointer memory with $L \times (\log N - 1)$ elements which can take on L distinct values. We need $\lceil \log L \rceil$ bits for the representation of the L distinct values. In total, the pointer memory contains $L\lceil \log L \rceil (\log N - 1)$ bits. For $L = 2,4$ and $N = 1024$, this translates to 18 and 72 bits, which is negligible. This memory also has the copying functionality that the partial sum and path memories provide. The architecture of the pointer memory is presented in Figure 2.2(a).

2.1.2.3 Decoding Schedule and Latency

In order to decode a codeword, the channel LLs are first loaded into the channel LL memory. Then, the MCU is activated in order to compute the $2L$ decision LLs (cf. Line 10 of Algorithm 3) for each codeword bit $i \in \{0,\dots,N-1\}$. Since the MCU uses the semi-parallel architecture of [42], the number of clock cycles required to compute all N sets of $2L$ decision LLs is $2N + \frac{N}{P}\log\frac{N}{4P}$. Every time the $2L$ decision LLs have been computed, the path selection unit is activated in order to find the best L paths. The sorting could be carried out in the same clock cycle as the computation of the $2L$ LL pairs, but we have found that this increases the critical path through the decoder significantly. Thus, a register is added between the output of the MCU and the metric sorter in order to reduce the length of the critical path. Unfortunately, decoding cannot proceed before the choice of paths is made, hence an idle cycle has to be introduced every time the output of the metric sorter is needed. This happens NR times per codeword. Thus, by modifying the expression found in [42], the number of cycles required to decode one codeword is now

$$D_{\text{SCL}}(N,P,\mathcal{A}) = (2+R)N + \frac{N}{P}\log\frac{N}{4P}, \tag{2.11}$$

where the rate can be calculated from $\mathcal{A}$ as $R = \frac{|\mathcal{A}|}{N}$. The overhead with respect to the case where we do not add a register is NR clock cycles, or approximately $\frac{RN}{2N} = 50R$ percent if we ignore the second term in 2.11, which is usually small. Nevertheless, our studies show that adding the register leads to an overall higher throughput due to a much higher achievable clock frequency.

2.2 LLR-Based SCL Decoder

In the previous section, we have described a baseline LL-based SCL decoder hardware architecture that provides some numerical stability and quantization bit-width gains with respect to a likelihood-based implementation. Ideally, however, one would like to be able to implement the SCL decoder using LLRs. LLRs provide additional numerical stability as well as lower memory requirements since pairs of LLs can be compressed into a single LLR value. Moreover, many processing blocks in practical receivers process the data in the form of LLRs. Therefore, the LLR-based SCL decoder can readily be incorporated into existing systems.

To this end, we first prove in Section 2.2.1 that the SCL decoding algorithm can in fact be formulated entirely in the LLR domain, thus enabling area-efficient and numerically stable implementations of SCL decoding. We discuss our SCL decoder hardware architecture in Section 2.2.2 and we leverage some useful properties of the LLR-based formulation in order to simplify various metric sorters (implementing the sorting step of SCL decoding) by avoiding unnecessary comparisons in Section 2.2.3. Next, in Section 2.3 we see that the LLR-based implementation leads to a significant reduction of the size of our LL-based hardware architecture of Section 2.1, as well as to an increase of its maximum operating frequency. We also

compare our decoder with the recent SCL decoder architectures of [51, 52] and show that our decoder can have more than 100% higher throughput per unit area than those architectures.

Finally, we show that a CRC-aided SCL decoder can be implemented by incorporating a CRC unit into our decoder, with almost no additional hardware cost, in order to achieve significantly lower block-error probabilities with respect to the original SCL decoding algorithm. As we will see, for a fixed information rate, the choice of CRC length is critical in the design of the modified polar code to be decoded by a CRC-aided SCL decoder. In Section 2.3.5 we provide simulation results showing that for small list sizes a short CRC will improve the performance of SCL decoder while larger CRCs will even degrade the performance compared to a polar code without CRC. As the list size gets larger, one can increase the length of the CRC in order to achieve considerably lower block-error probabilities.

An interesting question, which is, to the best of our knowledge, still unaddressed in the literature, is whether it is better to use SC decoding with long polar codes or SCL decoding with short polar codes. In Section 2.3.5.4 we study two examples of long polar codes that have the same block-error probability under SC decoding as our $(1024, 512)$ modified polar codes under CRC-aided SCL decoding. By comparing the synthesis results of the corresponding decoders, we observe that, while the SCL decoders have a lower throughput due to the sorting step, they also have a significantly lower decoding latency than the SC decoders.

2.2.1 LLR-Based Path Metric Computation

Algorithm 1 and Algorithm 2 of Section 1.3 are both valid high-level descriptions of SC and SCL decoding, respectively. However, for implementing these algorithms, the stability of the computations is crucial. The SCL algorithm summarized in Section 1.3.3.2 is described in terms of likelihoods which are *not* safe quantities to work with; a decoder implemented using the likelihoods is prone to underflow errors as they are typically very small numbers.[1]

Considering the binary tree picture of SC decoding provided in Section 1.3.3.1, the decision LLRs $\mathsf{LLR}_n^{(i)}$ (1.16) summarize all the necessary information for choosing the most likely child among two children of the same parent node at level i. We also saw that having this type of decisions in the conventional SC decoder allows to implement the computations in the LLR domain using numerically stable operations. However the SCL decoder, in lines 10–18 of Algorithm 2, has to choose the L most likely children out of $2L$ children of L different parents (see [48, Figure 3] for an illustration). For these comparisons the decision log-likelihood *ratios* $\mathsf{LLR}_n^{(i)}$ alone are not sufficient.

Consequently, the software implementation of the decoder in [21] implements the decoder in the likelihood domain by rewriting the LLR-based recursions of Section 1.3.3.1 to compute pairs of likelihoods $W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1} | u_i)$, $u_i \in \{0,1\}$ from pairs of channel likelihoods $W(y_i|x_i), x_i \in \{0,1\}, i \in \{0,\ldots,N-1\}$. To avoid underflows, at each intermediate step of the updates the likeli-

[1] As noticed in [21], it is not difficult to see that $W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_0^{i-1} | u_i) \leq 2^{-i}$.

hoods are scaled by a common factor such that $P_{\ell,u}$ in line 10 of Algorithm 2 is proportional to $W(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell]|u)$ [21].

Alternatively, such a normalization step can be avoided by performing the computations in the log-likelihood (LL) domain, i.e., by computing the pairs $\ln\big(W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}^{i-1}[\ell]|u)\big)$, $u \in \{0,1\}$, for $i \in \{0,\dots,N-1\}$, as a function of channel log-likelihood pairs $\ln(W(y_i|x_i))$, $x_i \in \{0,1\}$, $i \in \{0,\dots,N-1\}$, as shown in Section 2.1. Log-likelihoods provide some numerical stability, but still involve some issues compared to the log-likelihood *ratios* as we shall discuss in Section 2.2.2.

Luckily, we shall see that the decoding paths can still be ordered according to their likelihoods using all of the past decision LLRs $\mathsf{LLR}_n^{(j)}$, $j \in \{0,1\cdots,i\}$, and the trajectory of each path as summarized in the following theorem.

Theorem 1. *For each path ℓ and each level $i \in \{0,\dots,N-1\}$ let the* path-metric *be defined as:*

$$\mathsf{PM}_\ell^{(i)} \triangleq \sum_{j=0}^{i} \ln\big(1+e^{-(1-2\hat{u}_j[\ell])\cdot\mathsf{LLR}_n^{(j)}[\ell]}\big), \tag{2.12}$$

where

$$\mathsf{LLR}_n^{(i)}[\ell] = \ln\left(\frac{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell]|0)}{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell]|1)}\right),$$

is the log-likelihood ratio of bit u_i given the channel output $\boldsymbol{y}$ and the past trajectory of the path $\hat{\boldsymbol{u}}_0^{i-1}[\ell]$ from the root of the tree to the current node.

If all the information bits are uniformly distributed in $\{0,1\}$, for any pair of paths ℓ_1, ℓ_2,

$$W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell_1]|\hat{u}_i[\ell_1]) < W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}[\ell_2]|\hat{u}_i[\ell_2])$$

if and only if

$$\mathsf{PM}_{\ell_1}^{(i)} > \mathsf{PM}_{\ell_2}^{(i)}.$$

In view of Theorem 1, one can implement the SCL decoder using L parallel low-complexity *and stable* LLR-based SC decoders as the underlying building blocks and, in addition, keep track of L path-metrics. The metrics can be updated successively as the decoder proceeds by setting

$$\mathsf{PM}_\ell^{(i)} = \phi\big(\mathsf{PM}_\ell^{(i-1)}, \mathsf{LLR}_n^{(i)}[\ell], \hat{u}_i[\ell]\big), \tag{2.13a}$$

where the function $\phi : \mathbb{R}_+^2 \times \{0,1\} \to \mathbb{R}_+$ is defined as

$$\phi(\mu,\lambda,u) \triangleq \mu + \ln\big(1+e^{-(1-2u)\lambda}\big). \tag{2.13b}$$

As shown in Algorithm 4, the paths can be compared based on their likelihood using the values

of the associated path metrics $\mathsf{PM}_\ell^{(i)}$ as a proxy.

Algorithm 4: LLR-based formulation of SCL Decoding

$\mathcal{L} \leftarrow \{0\}$; // start with a single active path
$\mathsf{PM}_0^{(0)} \leftarrow 0$;
for $i = 0, 1, \ldots, N-1$ **do**
  Compute $\mathsf{LLR}_n^{(i)}[\ell]$ for $\forall \ell \in \mathcal{L}$; // parallel SC decoders
  if $i \notin \mathcal{A}$ **then** // frozen bits
    $\big(\hat{u}_i[\ell], \mathsf{PM}_\ell^{(i)}\big) \leftarrow \big(u_i, \phi(\mathsf{PM}_\ell^{(i-1)}, \mathsf{LLR}_n^{(i)}[\ell], u_i)\big)$ for $\forall \ell \in \mathcal{L}$; // cf. (2.13b)
  else // information bits
    Set $P_{\ell,u} \leftarrow \phi(\mathsf{PM}_\ell^{(i-1)}, \mathsf{LLR}_n^{(i)}, u)$ for $\forall \ell \in \mathcal{L}$ and $\forall u \in \{0,1\}$; // cf (2.13b)
    if $|\mathcal{L}| < L$ **then** // duplicate all the paths
      foreach $\ell \in \mathcal{L}$ **do**
        `duplicatePath(`ℓ`);`
    else
      $\tau \leftarrow$ the median of $2L$ numbers $P_{\ell,u}$;
      foreach $\ell \in \mathcal{L}$ *such that* $P_{\ell,0} > \tau$ *and* $P_{\ell,1} > \tau$ **do**
        Kill the path ℓ and set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\ell\}$;
      for $\ell \in \mathcal{L}$ **do**
        if $P_{\ell,u} > \tau$ *while* $P_{\ell,u\oplus 1} < \tau$ **then**
          $\big(\hat{u}_i[\ell], \mathsf{PM}_\ell^{(i)}\big) \leftarrow (u, P_{\ell,u})$;
        else // both $P_{\ell,0}$ and $P_{\ell,1}$ are $\leq \tau$
          `duplicatePath(`ℓ`);`
$\ell^* \leftarrow \operatorname{argmin}_{\ell \in \mathcal{L}} \mathsf{PM}_\ell^{(N)}$;
return $\hat{\boldsymbol{u}}_{\mathcal{A}}[\ell^*]$;
subroutine `duplicatePath(`ℓ`)`
  Copy the path ℓ into a new path $\ell' \notin \mathcal{L}$;
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell'\}$;
  $\big(\hat{u}_i[\ell], \mathsf{PM}_\ell^{(i)}\big) \leftarrow (0, P_{\ell,0})$;
  $\big(\hat{u}_i[\ell'], \mathsf{PM}_{\ell'}^{(i)}\big) \leftarrow (1, P_{\ell,1})$;

Before proving Theorem 1 let us provide an intuitive interpretation of our metric. Since

$$\ln(1+e^x) \approx \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \geq 0, \end{cases} \tag{2.14}$$

the update rule (2.13) is well-approximated if we replace ϕ with $\tilde{\phi} : \mathbb{R}_+^2 \times \{0,1\} \to \mathbb{R}_+$ defined as

$$\tilde{\phi}(\mu, \lambda, u) \triangleq \begin{cases} \mu & \text{if } u = \frac{1}{2}[1 - \operatorname{sign}(\lambda)], \\ \mu + |\lambda| & \text{otherwise.} \end{cases} \tag{2.15}$$

We also note that $\frac{1}{2}[1-\text{sign}(\mathsf{LLR}_n^{(i)}[\ell])]$ is the direction that the LLR (given the past trajectory $\hat{\boldsymbol{u}}_0^{i-1}[\ell]$) suggests. This is the same decision that a SC decoder would have taken if it were to estimate the value of u_i at step i given the past set of decisions $\hat{\boldsymbol{u}}_0^{i-1}[\ell]$ (cf. line 5 in Algorithm 1). Equation (2.15) shows that if at step i the ℓth path does not follow the direction suggested by $\mathsf{LLR}_n^{(i)}[\ell]$ it will be penalized by an amount that is approximately equal to $|\mathsf{LLR}_n^{(i)}[\ell]|$.

With such an intuitive interpretation, one might immediately conclude that the path that SC decoder would follow will always have the lowest penalty hence is always declared as the output of the SCL decoder. However, this reasoning is correct only if *all* the elements of $\boldsymbol{u}$ are information bits. As soon as the decoder encounters a frozen bit, the path metric is updated based on the likelihood of that frozen bit, given the past trajectory of the path and the a-priori known value of that bit (cf. line 6 in Algorithm 4). This can penalize the SC path by a considerable amount, if the value of that frozen bit does not agree with the LLR given the past trajectory (which is an indication of a preceding erroneous decision), while keeping some other paths unpenalized.

We devote the rest of this section to the proof of Theorem 1.

Lemma 1. *If U_i is uniformly distributed in $\{0,1\}$, then,*

$$\frac{W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_0^{i-1}|u_i)}{\mathbb{P}[\boldsymbol{U}_0^i = \boldsymbol{u}_0^i|\boldsymbol{Y}=\boldsymbol{y}]} = 2\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}].$$

Proof. Since $\mathbb{P}[U_i = u_i] = \frac{1}{2}$ for $\forall\, u_i \in \{0,1\}$,

$$\begin{aligned}\frac{W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_0^{i-1}|u_i)}{\mathbb{P}[\boldsymbol{U}_0^i = \boldsymbol{u}_0^i|\boldsymbol{Y}=\boldsymbol{y}]} &= \frac{\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}, \boldsymbol{U}_0^i = \boldsymbol{u}_0^i]}{\mathbb{P}[U_i=u_i]\mathbb{P}[\boldsymbol{U}_0^i = \boldsymbol{u}_0^i|\boldsymbol{Y}=\boldsymbol{y}]} \\ &= \frac{\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}]\mathbb{P}[\boldsymbol{U}_0^i = \boldsymbol{u}_0^i|\boldsymbol{Y}=\boldsymbol{y}]}{\mathbb{P}[U_i=u_i]\mathbb{P}[\boldsymbol{U}_0^i = \boldsymbol{u}_0^i|\boldsymbol{Y}=\boldsymbol{y}]} = 2\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}].\end{aligned}$$

□

Proof of Theorem 1. It is sufficient to show that

$$\mathsf{PM}_\ell^{(i)} = -\ln\Big(\mathbb{P}[\boldsymbol{U}_0^i = \hat{\boldsymbol{u}}_0^i[\ell]|\boldsymbol{Y}=\boldsymbol{y}]\Big). \tag{2.16}$$

Having shown (2.16), Theorem 1 will follow as an immediate corollary to Lemma 1 (since the channel output $\boldsymbol{y}$ is fixed for all decoding paths). Since the path index ℓ is fixed on both sides of (2.12) we will drop it in the sequel. Let

$$\Lambda_n^{(i)} \triangleq \frac{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_0^{i-1}|0)}{W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_0^{i-1}|1)} = \frac{\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}, \boldsymbol{U}_0^{i-1} = \hat{\boldsymbol{u}}_0^{i-1}, U_i = 0]}{\mathbb{P}[\boldsymbol{Y}=\boldsymbol{y}, \boldsymbol{U}_0^{i-1} = \hat{\boldsymbol{u}}_0^{i-1}, U_i = 1]}$$

(the last equality follows since $\mathbb{P}[U_i=0] = \mathbb{P}[U_i=1]$), and observe that showing (2.16) is

equivalent to proving

$$\mathbb{P}[\boldsymbol{U}^i = \hat{\boldsymbol{u}}_0^i | \boldsymbol{Y} = \boldsymbol{y}] = \prod_{j=0}^{i} \left(1 + (\Lambda_n^{(j)})^{-(1-2\hat{u}_j)}\right)^{-1}. \tag{2.17}$$

Since

$$\begin{aligned}\mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i-1} = \hat{\boldsymbol{u}}_0^{i-1}] &= \sum_{\hat{u}_i \in \{0,1\}} \mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i} = \hat{\boldsymbol{u}}_0^{i}] \\ &= \mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i} = \hat{\boldsymbol{u}}_0^{i}]\left(1 + (\Lambda_n^{(i)})^{-(1-2\hat{u}_i)}\right),\end{aligned}$$

$$\begin{aligned}&\mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i} = \hat{\boldsymbol{u}}_0^{i}] \\ &= \left(1 + (\Lambda_n^{(i)})^{-(1-2\hat{u}_i)}\right)^{-1} \mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i-1} = \hat{\boldsymbol{u}}_0^{i-1}].\end{aligned} \tag{2.18}$$

Repeated application of (2.18) (for $i-1, i-2, \ldots, 0$) yields

$$\mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{U}_0^{i} = \hat{\boldsymbol{u}}_0^{i}] = \prod_{j=0}^{i} \left(1 + (\Lambda_n^{(j)})^{-(1-2\hat{u}_i)}\right)^{-1} \mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}].$$

Dividing both sides by $\mathbb{P}[\boldsymbol{Y} = \boldsymbol{y}]$ proves (2.17). □

2.2.2 LLR-Based SCL Decoder Hardware Architecture

In this section, we show how the LLR-based path metric derived in the previous section can be exploited in order to derive a very efficient LLR-based SCL decoder hardware architecture. To this end, we give a detailed description of each unit of our LLR-based SCL decoder architecture, which, similarly to the LL-based decoder described in Section 2.1, essentially consists of L parallel SC decoders along with a path management unit which coordinates the tree search. Moreover, we highlight the advantages of an LLR-based SCL decoder hardware architecture over the LL-based decoder architecture described in Section 2.1.

Our SCL decoder consists of five units: the *memories unit*, the *metric computation unit* (MCU), the *metric sorting unit*, the *address translation unit*, and the *control unit*. An overview of the SCL decoder is shown in Figure 2.3.

2.2.2.1 LLR and Path Metric Quantization

In principle, similarly to the LL-based decoder, the dynamic range of the internal LLRs also increases slightly as the decoding process moves along the stages of the DDG. However, since the LLRs are both positive and negative, the LLRs do not grow as rapidly as the LLs. Moreover, saturations are not as critical when using LLRs as they are when using LLs. For this reason, both the channel LLRs and the internal LLRs are quantized using a Q_{LLR}-bit signed uniform quantizer with step size $\Delta = 1$. The path metrics are unsigned numbers which are quantized

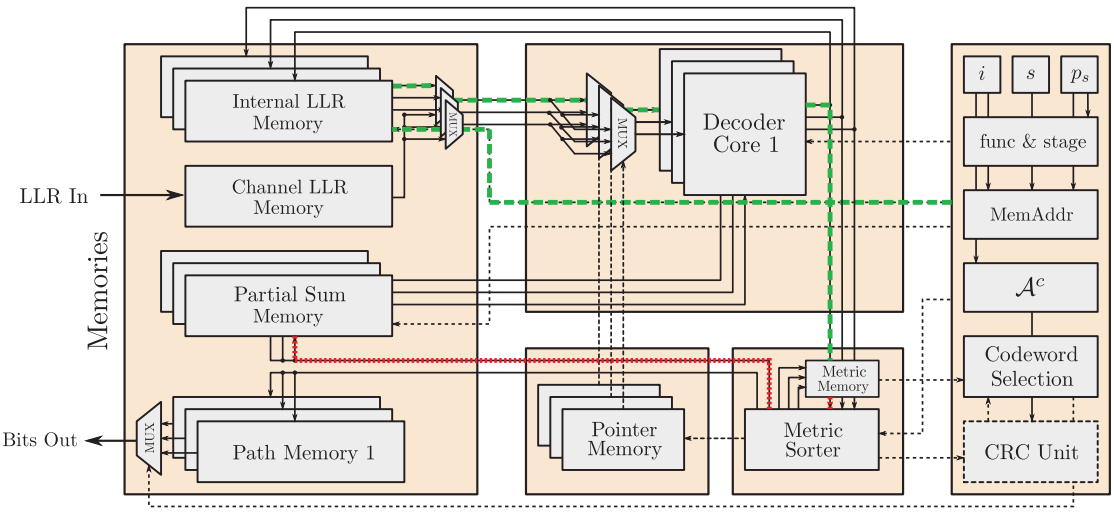


Figure 2.3 – Overview of the SCL decoder architecture. Details on the i, s, p_s, as well as the func & stage and MemAddr components inside the control unit, which are not described in this section, can be found in Section 2.1. The dashed green and the dotted red line show the critical paths for $L = 2$ and $L = 4, 8$ respectively.

using M bits. Since the path metrics are initialized to 0 and, in the worst case, they are incremented by $2^{Q_{\text{LLR}}-1} - 1$ for each bit index i, the maximum possible value of a path metric is $N(2^{Q_{\text{LLR}}-1} - 1) = 2^{n+Q_{\text{LLR}}-1} - 2^n < 2^{n+Q_{\text{LLR}}-1}$. Hence, at most $M = n + Q_{\text{LLR}} - 1$ bits are sufficient to ensure that there will be no overflows in the path metric. In practice, any path that gets continuously harshly penalized will most likely be discarded. Therefore, as we will see in Section 2.3, much fewer bits are sufficient in practice for the quantization of the path metrics.

2.2.2.2 Decoder Building Blocks

Metric Computation Unit The computation of the L decision LLRs (line 4 of Algorithm 4), which are required to update the path metrics $\mathsf{PM}_\ell^{(i)}$, can be fully parallelized. Consequently, the MCU consists of L parallel SC decoder cores which implement the LLR-based SC decoding update rules and compute the L decision LLRs using the semi-parallel SC decoder architecture of [38] with P PEs. Each decoder core reads its input LLRs from one of the L physical LLR memory banks based on an address translation performed by the pointer memory (described in more detail in Section 2.1.2.2). When the L decision LLRs have been computed, the MCUs wait for one clock cycle. During this single clock cycle, the path metrics $\mathsf{PM}_\ell^{(i)}$ are updated and sorted. Moreover, based on the result of metric sorting, the partial sum, path, and pointer memories are also updated in the same clock cycle, as described in the sequel.

Memory Unit As in the LL-based SCL decoder of Section 2.1, the proposed LLR-based SCL decoder contains a path memory, a partial sum memory, as well as an address translation unit that implements the lazy copy mechanism of [21]. In the proposed LLR-based SCL decoder,

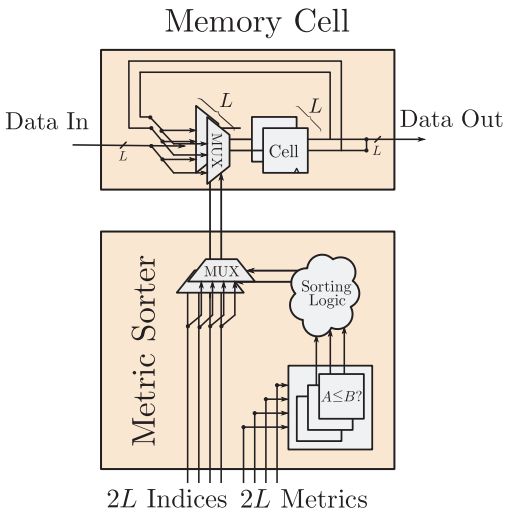


Figure 2.4 – Bit-cell copying mechanism controlled by the metric sorter.

these memories are identical to the LL-based SCL decoder and, thus, we do not describe them again. The LL memory found in the LL-based decoder is replaced by an LLR memory in the LLR-based SCL decoder of this section.

LLR Memory: The channel LLRs are fixed during the decoding process of a given codeword, meaning that an SCL decoder requires only one copy of the channel LLRs. These are stored in a memory which is $\frac{N}{P}$ words deep and $Q_{\text{LLR}}P$ bits wide. On the other hand, the internal LLRs of the intermediate stages of the SC decoding (metric computation) process are different for each path $\ell \in \{0,\ldots,L-1\}$. Hence we require L physical LLR memory banks with $N-1$ memory positions per bank. All LLR memories have two reads ports, so that all P PEs can read their two Q_{LLR}-bit input LLRs simultaneously. Here, register based storage cells are used to implement all the memories.

Metric Sorting Unit The metric sorting unit contains a *path metric memory* and a *path metric sorter*. The path metric memory stores the L path metrics $\mathsf{PM}_{\ell}^{(i)}$ using M bits of quantization for each metric. In order to find the median τ at each bit index i (line 13 of Algorithm 4), the path metric sorter sorts the $2L$ candidate path metrics $P_{\ell,u}$, $\ell \in \{0,\ldots,L-1\}$, $u \in \{0,1\}$ (line 8 of Algorithm 4). The path metric sorter takes the $2L$ path metrics as an input and produces the sorted path metrics, as well as the path indices ℓ and bit values u which correspond to the sorted path metrics as an output. Since decoding cannot continue before the surviving paths have been selected, the metric sorter is a crucial component of the SCL decoder. Hence, we will discuss various sorter architectures in detail in Section 2.2.3.

Control Unit The control unit generates all memory read and write addresses as in [38]. Moreover, the control unit contains the codeword selection unit and the optional CRC unit.

The CRC unit contains L r-bit CRC memories, where r is the number of CRC bits. A bit-serial implementation of a CRC computation unit is very efficient in terms of area and path delay, but it requires a large number of clock cycles to produce the checksum. However, this computation delay is masked by the bit-serial nature of the SCL decoder itself and, thus, has no impact on the number of clock cycles required to decode each codeword. Before decoding each codeword, all CRC memories are initialized to r-bit all-zero vectors. For each $\hat{u}_i[\ell],\ i \in \mathcal{A}$, the CRC unit is activated to update the CRC values. When decoding finishes, the CRC unit declares which paths $\ell \in \{0,\ldots,L-1\}$ pass the CRC.[2] When a path is duplicated the corresponding CRC memory is copied by means of $L \times L$ crossbars (like the partial sums and the path memory).

If the CRC unit is present, the codeword selection unit selects the most likely path (i.e., the path with the smallest metric $\mathsf{PM}_\ell^{(i)}$) out of the paths that pass the CRC. If the CRC unit is not present or if all paths fail the CRC, the codeword selection unit simply chooses the most likely path.

2.2.2.3 Decoding Schedule and Latency

The schedule of the LLR-based decoder is practically identical to the schedule of the LL-based decoder. One small difference is that metric sorting may require a slightly higher number of total clock cycles, depending on which metric sorter is used (cf. Section 2.2.3.5). Let the total number of cycles required for metric sorting at all information indices $i \in \mathcal{A}$ be denoted by $D_{\mathrm{MS}}(\mathcal{A})$. Then, our SCL decoder requires

$$D_{\mathrm{SCL}}(N,P,\mathcal{A}) = 2N + \frac{N}{P}\log\frac{N}{4P} + D_{\mathrm{MS}}(\mathcal{A}) \tag{2.19}$$

cycles to decode each codeword.

2.2.3 Path Metric Sorting

From our preliminary implementation results, we saw that, even for relatively modest list sizes (e.g., $L \geq 4$), the maximum (critical) delay path of our architecture passes through the metric sorter, thus reducing the maximum operating frequency of the decoder described in Section 2.1. Fortunately, it turns out that the LLR-based path metric we introduced in Theorem 1 has some properties (which the LL-based path metric lacks) that can be used to simplify the sorting task.

To this end, we note that the $2L$ real numbers that have to be sorted in line 13 of Algorithm 4 are not arbitrary. Half of them are the previously existing path-metrics (which are already

[2] We note that it is possible for multiple paths to pass the CRC, since a CRC cannot detect all possible error patterns.

sorted as a result of decoding the preceding information bit) and the rest are obtained by adding positive real values (the absolute value of the corresponding LLRs) to the existing path metrics. Moreover, we do not need to sort *all* these $2L$ potential path metrics; a sorted list of the L smallest path metrics is sufficient.

Hence, the sorting task of the SCL decoder can be formalized as follows. Given a sorted list of L path metrics from the previous decoding step

$$\mu_0 \leq \mu_1 \leq \cdots \leq \mu_{L-1}, \tag{2.20}$$

a list of metrics of size $2L$, $\boldsymbol{m} = [m_0, m_1, \cdots, m_{2L-1}]$, is created by setting

$$m_{2\ell} := \mu_\ell \quad \text{and} \quad m_{2\ell+1} := \mu_\ell + a_\ell, \qquad \ell \in \{0, \ldots, L-1\}, \tag{2.21}$$

where $a_\ell \geq 0$, for $\forall\, \ell \in \{0, \ldots, L-1\}$. The path selection problem is equivalent to finding a sorted list of L smallest elements of $\boldsymbol{m}$ when the elements of $\boldsymbol{m}$ have the following two properties: for $\forall\, \ell \in \{0, 1, \cdots, L-2\}$,

$$m_{2\ell} \leq m_{2(\ell+1)}, \tag{2.22a}$$

$$m_{2\ell} \leq m_{2\ell+1}, \tag{2.22b}$$

for $\ell \in \{0, 1, \ldots, L-2\}$.

Note that (2.22a) and (2.22b) imply that out of $\binom{2L}{2} = L(2L-1)$ unknown pairwise relations between the elements of $\boldsymbol{m}$, L^2 are known (every even-indexed element is smaller than all its following elements). Hence, in principle the sorting complexity can be reduced by approximately a factor of two.

Remark. For LLR-based SCL decoding, in order for the assumptions on the list structure to hold, besides the above mentioned problem, a general sorting problem of size L needs to be solved infrequently in order to ensure that (2.22a) holds. This happens because the path metrics are also updated when frozen bits are encountered, but these are not passed through the metric sorter in order to keep them sorted.

We note that this problem can be solved by using a sorter that finds the L smallest elements of a list with properties (2.22a) and (2.22b), $L-1$ times in a row. In particular, let $a_0, a_1, \ldots, a_{L-1}$ be arbitrary real numbers. For $\ell = 0, 1, \ldots, L-2$, set $m_{2\ell} := -\infty$ and $m_{2\ell+1} = a_\ell$. Finally set $m_{2L-2} := a_{L-1}$ and $m_{2L-1} := +\infty$. It is easy to check that (2.22a) and (2.22b) hold for the list $\boldsymbol{m}$ and the ordered L smallest elements of this list are $[-\infty, -\infty, \ldots, \min_{0 \leq \ell \leq L-1} a_\ell]$. Thus, we can find the minimum of up to L arbitrary real numbers using such a sorter. Consequently, using the sorter $L-1$ times in a row we can sort an arbitrary set of L real numbers. For the pruned radix-$2L$ sorter in particular, we show in Section 2.2.3.2 that it can be re-used in order to solve a generic sorting problem of size L by simply carefully re-arranging the input values.

Another simple solution is to instantiate a generic sorter for the sorting problem of size L and

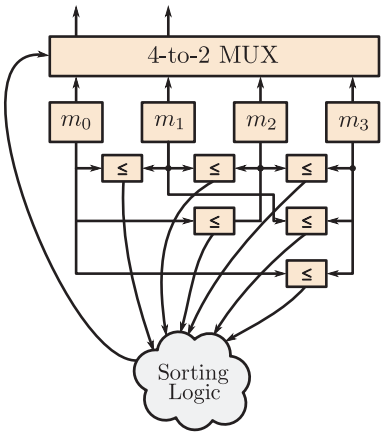


Figure 2.5 – Radix-2L sorter for $L = 2$.

a more specialized sorter that fully utilizes (2.22a) and (2.22b) for the sorting problem of size $2L$. As we show in Section 2.3.2, this solution still results in an overall area and maximum operating frequency improvement with respect to the case where a generic sorter is used to solve the sorting problem of size $2L$ (note that such a sorter can also trivially solve the sorting problem of size L by setting the remaining L inputs to $+\infty$). This result is not unexpected, since the complexity of all the generic sorters that we consider scales super-linearly.

In the following, we first briefly review the existing path metric sorter architectures and we then exploit the properties (2.22a) and (2.22b) in order to simplify existing sorters and also introduce a modified version of the well-known bubble sort algorithm that can be efficiently parallelized for the particular application of LLR-based SCL decoding.

2.2.3.1 Existing Metric Sorter Architectures

Radix-2L Sorter In Chapter 2.1, we used a radix-2L sorter, which blindly compares every pair of elements $(m_\ell, m_{\ell'})$ and then combines the results to find the L smallest elements. This solution requires $\binom{2L}{2} = L(2L-1)$ comparators together with L $2L$-to-1 multiplexers (see Figure 2.5). The *sorting logic* combines the results of all comparators in order to generate the control signal for the multiplexers (cf. [50] for details). The maximum path delay of the radix-2L sorter is mainly determined by the complexity of the sorting logic, which in turn depends on the number of comparator results that need to be processed.

Bitonic Sorter For the SCL decoder of [53], the authors used a bitonic sorter [54]. A bitonic sorter that can sort $2L$ arbitrary numbers consists of $(\log L + 1)$ *super-stages*. Each super-stage $s \in \{1, 2, \ldots, \log L + 1\}$ contains s *stages*. The total number of stages is therefore

$$s_{\text{tot}}^{\text{BT}} = \sum_{s=1}^{\log L+1} s = \frac{1}{2}(\log L + 1)(\log L + 2). \tag{2.23}$$

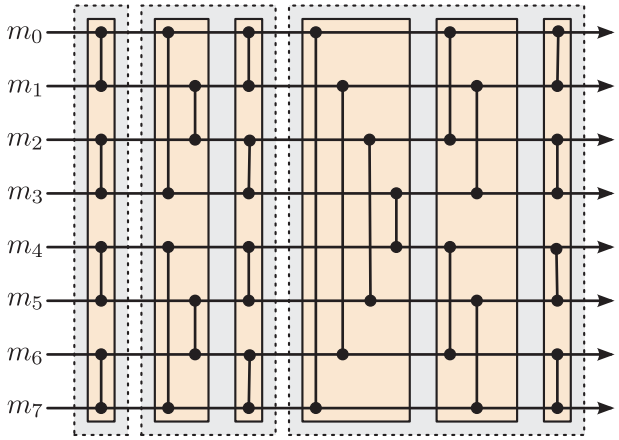


Figure 2.6 – Bitonic sorter for $L = 4$.

The length of the critical path of the sorter is determined by the number of stages. Each stage contains L compare-and-select (CAS) units consisting of one comparator and a 2-to-2 MUX. Thus, the total number of CAS units in a bitonic sorter is

$$c_{\text{tot}}^{\text{BT}} = \frac{L}{2}(\log L + 1)(\log L + 2). \tag{2.24}$$

Thus, the scaling behavior of the bitonic sorter in terms of the number of comparators is superior to the scaling behavior of the radix-$2L$ sorter. An example of a bitonic sorting network that can sort $2L = 8$ numbers is given in Figure 2.6. Each vertical connection between two horizontal lines denotes a CAS unit, whose two inputs are the values that can be found on the two endpoints of the vertical connection. The bitonic sorter can sort its $2L$ inputs in an ascending or in a descending order, depending on whether the individual CAS units sort their two input values in an ascending or in a descending order.

2.2.3.2 Pruned Radix-$2L$ Sorter

The *pruned radix-2L* sorter presented in this section reduces the complexity of the sorting logic of the radix-$2L$ sorter and, thus, also the maximum path delay, by eliminating some pairwise comparisons whose results are either already known or irrelevant.

Proposition 1. *In order to find the L smallest elements of* $\boldsymbol{m}$*, it is sufficient to use a pruned radix-2L sorter that involves only* $(L-1)^2$ *comparators. This sorter is obtained by*

(a) removing the comparisons between every even-indexed element of $\boldsymbol{m}$ *and all following elements, and*

(b) removing the comparisons between m_{2L-1} *and all other elements of* $\boldsymbol{m}$*.*

Proof. Properties (2.22a) and (2.22b) imply $m_{2\ell} \leq m_{\ell'}$ for $\forall \ell' > 2\ell$. Hence, the outputs of these comparators are known. Furthermore, as we only need the first L elements of the list

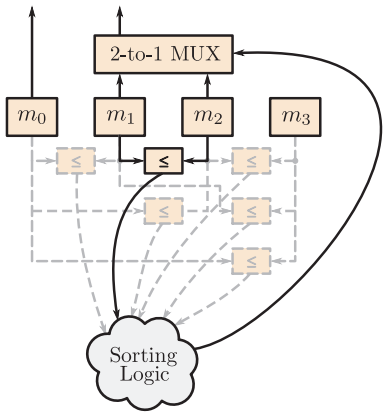


Figure 2.7 – Pruned radix-2L sorter for $L = 2$.

sorted and m_{2L-1} is never among the L smallest elements of $\boldsymbol{m}$, we can always replace m_{2L-1} by $+\infty$ (pretending the result of the comparisons involving m_{2L-1} is known) without affecting the output of the sorter.

In step (a) we have removed $\sum_{\ell=0}^{L-1}(2L-1-2\ell) = L^2$ comparators and in step (b) $(L-1)$ comparators (note that in the full sorter m_{2L-1} is compared to all $(2L-1)$ preceding elements, but L of them correspond to even-indexed elements whose corresponding comparators have already been removed in step (a)). Hence we have $L(2L-1) - L^2 - (L-1) = (L-1)^2$ comparators. □

Besides the $(L-1)^2$ comparators, the pruned radix-2L sorter requires $L-1$ $(2L-2)$-to-1 multiplexers (see Figure 2.7).

As discussed in Section 2.2.3, a general sorting problem of size L needs to be solved infrequently in order to keep the L path metrics sorted when exiting a cluster of frozen bits. The existing pruned radix-2L sorter can be used for sorting L arbitrary positive numbers as follows.

Proposition 2. *Let $a_0, a_1, \ldots, a_{L-1}$ be L non-negative numbers. Create a list of size $2L$ as*

$$\boldsymbol{b} \triangleq [0, a_0, 0, a_1, \ldots, 0, a_{L-2}, a_{L-1}, +\infty].$$

Feeding this list to the pruned radix-$2L$ sorter will result in an output list of the form

$$[\underbrace{0, 0, \ldots, 0}_{L-1 \text{ zeros}}, a_{(0)}, a_{(1)}, \ldots, a_{(L-1)}, +\infty]$$

where $a_{(0)} \leq a_{(1)} \leq \cdots \leq a_{(L-1)}$ is the ordered permutation of $a_0, a_1, \ldots, a_{L-1}$.

Proof. It is clear that the assumptions (2.22a) and (2.22b) hold for $\boldsymbol{b}$. The proof of Proposition 1 shows if the last element of the list is additionally known to be the largest element, the pruned radix-2L sorter sorts the entire list. □

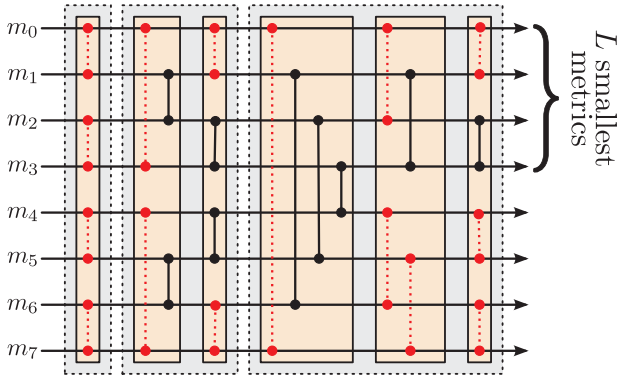


Figure 2.8 – Pruned bitonic sorter for $L = 4$. The full bitonic sorter requires all the depicted CAS units (cf. Figure 2.6), while in the pruned bitonic sorter all CAS units in red dotted lines can be removed.

Note that while the same comparator network of a pruned radix-$2L$ sorter is used for sorting L numbers, L separate L-to-1 multiplexers are required to output the sorted list.

2.2.3.3 Pruned Bitonic Sorter

As was the case with the radix-$2L$ sorter, the known relations between the elements can be exploited to simplify the bitonic sorter. In particular, due to (2.22b), the results of all sorters in stage 1 are already known. Thus, stage 1 can be removed completely from the sorting network. Moreover, the result of all comparators whose one input is m_0 are also known, since m_0 is, by construction, always the smallest element of $\boldsymbol{m}$. Furthermore, since m_{2L-1} is never among the L smallest elements of the list, all comparisons involving m_{2L-1} are irrelevant and the corresponding CAS units can be removed. Finally, we can remove the $L/2$ last CAS units of the $\log L$ final stages of super-stage $\log L + 1$, since they are responsible for sorting the last L elements of $\boldsymbol{m}$ while we are only interested in its first L elements. The unnecessary CAS units for $2L = 8$ are illustrated with dotted red lines in Figure 2.8.

Since only stage 1 is completely removed from the sorting network, the number of stages in the pruned bitonic sorter is,

$$s_{\text{tot}}^{\text{PBT}} = s_{\text{tot}}^{\text{BT}} - 1 = \frac{1}{2}(\log L + 1)(\log L + 2) - 1. \tag{2.25}$$

Therefore, the delay of the pruned bitonic sorter is only slightly smaller than that of the full bitonic sorter, especially for large list sizes L.

To compute the number of CAS units in a pruned bitonic sorter, we note that the first super-stage is eliminated completely. In all remaining super-stages except the last one, the 2 CAS units per stage that are connected to m_0 and m_{2L-1} are removed, since m_0 is always the smallest element and m_{2L-1} is never among the L smallest elements. In the last super-stage, we can remove the CAS units connected to m_0 plus all the CAS units in the second half of the

Algorithm 5: The Bubble Sort Algorithm

```
1 while exists ℓ such that m_ℓ > m_{ℓ+1} do
2     for ℓ = 2L − 1 to 1 do
3         if m_ℓ < m_{ℓ−1} then
4             Swap m_ℓ and m_{ℓ−1};
5 return m
```

last $\log L$ stages since they contribute in sorting the L largest elements of the list, which we are not interested in. Hence, the total number of CAS units in the pruned bitonic sorter can be shown to be equal to

$$c_{\text{tot}}^{\text{PBT}} = \left(\frac{L}{2} - 1\right)(\log L)(\log L + 2) + 1. \tag{2.26}$$

By examining the ratio between $c_{\text{tot}}^{\text{BT}}$ and $c_{\text{tot}}^{\text{PBT}}$ we can conclude that, similarly to the maximum delay, the relative reduction in the number of comparators also diminishes with increasing list size L.

2.2.3.4 Bubble Sorter

Even though bubble sort [55, Chapter 2] is a generally inefficient sorting algorithm, it turns out to be a suitable candidate for our particular problem. More precisely, properties (2.22a) and (2.22b) result in a specific data dependency structure of the algorithm enabling an efficient parallel hardware implementation of the sorter. Furthermore, since we only require the sorted list of L smallest elements of $\boldsymbol{m}$ (rather than sorting the entire list $\boldsymbol{m}$) we can simplify the sorter by only implementing the first half of the rounds of the bubble sorting algorithm.

The bubble sort algorithm is formalized in Algorithm 5. It is clear that Algorithm 5 sorts the full list of $2L$ inputs. By restricting the **while** condition as "exists $\ell \in \{0, 1, \ldots, L-1\}$ such that $m_\ell > m_{\ell+1}$" one can simplify the algorithm to only output the first L ordered elements of the list $\boldsymbol{m}$.

Lemma 2. *Let m_ℓ^t denote the element at position ℓ of the list at* the beginning of round *t of the **while** loop in Algorithm 5 and,*

$$\mathcal{B}_t \triangleq \left\{\ell \in \{1, 2, \ldots, 2L-1\} : m_\ell^t < m_{\ell-1}^t\right\}. \tag{2.27}$$

Then (2.22a) *and* (2.22b) *imply that for all $t \geq 1$,*

(i) $\mathcal{B}_t$ does not contain adjacent indices,

*(ii) the **if** body (line 4) is executed at round t iff $\ell \in \mathcal{B}_t$,*

(iii) $\mathcal{B}_{t+1} \subseteq \mathcal{B}_t + 1$, where addition of a constant with a set is defined as $\mathcal{X} + a \triangleq \{x + a : x \in \mathcal{X}\}$.

Proof. To prove the lemma, we will prove that for all $t \geq 1$,

$$m_\ell^t \geq m_{\ell-2}^t \qquad \text{for all } \ell \in \mathcal{B}_t. \tag{2.28}$$

We first show that (2.28) implies (i)–(iii) and then prove (2.28).

(i) Suppose $\ell \in \mathcal{B}_t$, hence, $m_\ell^t < m_{\ell-1}^t$ and (2.28) implies $m_\ell^t \geq m_{\ell-2}^t$. Thus $m_{\ell-1}^t > m_{\ell-2}^t$ which implies $\ell - 1 \notin \mathcal{B}_t$.

(ii) Note that the element at position ℓ of the list is changed if and only if line 4 is executed for indices ℓ or $\ell+1$. We use strong induction on ℓ to prove (2). Clearly line 4 is executed for the first time for an index $\ell^* = \max \mathcal{B}_t$.

Assume line 4 is executed for some index ℓ. This implies $m_\ell < m_{\ell-1}$ before execution of this line when $m_{\ell-1} = m_{\ell-1}^t$ (since line 4 has not been executed for ℓ nor for $\ell-1$ so far). Now if $\ell+1 \notin \mathcal{B}_t$, then $m_\ell^t = m_\ell$ as well by the induction assumption (since line 4 is not executed for index $\ell+1$) hence $m_\ell^t < m_{\ell-1}^t$ which means $\ell \in \mathcal{B}_t$. Otherwise, $\ell+1 \in \mathcal{B}_t$, implies line 4 is executed for $\ell+1$. Since $\mathcal{B}_t$ does not contain adjacent elements, line 4 is *not* executed for $\ell+2$. This implies $m_\ell = m_{\ell+1}^t \geq m_{\ell-1} = m_{\ell-1}^t$ by (2.28) which contradicts the assumption of loop being executed for ℓ.

Conversely, assume $\ell \in \mathcal{B}_t$. Since $\ell+1 \notin \mathcal{B}_t$, line 4 is not executed for $\ell+1$ by assumption. Hence once the **for** loop is executed for index ℓ, $m_\ell = m_\ell^t$ and (as we justified before) $m_{\ell-1} = m_{\ell-1}^t$. Therefore, $m_\ell < m_{\ell-1}$ and line 4 is executed for ℓ.

(iii) Using (i) and (ii), we can explicitly write the time-evolution of the list as

$$m_\ell^{t+1} = \begin{cases} m_{\ell-1}^t, & \text{if } \ell \in \mathcal{B}_t, \\ m_\ell^t, & \text{if } \ell \notin \mathcal{B}_t \text{ and } \ell+1 \notin \mathcal{B}_t, \\ m_{\ell+1}^t & \text{if } \ell+1 \in \mathcal{B}_t. \end{cases} \tag{2.29}$$

Therefore, if $\ell \in \mathcal{B}_t$, $m_\ell^{t+1} = m_{\ell-1}^t > m_\ell^t = m_{\ell-1}^{t+1}$, and $\ell \notin \mathcal{B}_{t+1}$. Pick $\ell \in \mathcal{B}_{t+1}$. We shall show this requires $\ell - 1 \in \mathcal{B}_t$. Since $\ell \notin \mathcal{B}_t$ as we just showed in (iii), (2.29) yields

$$m_\ell^{t+1} = \begin{cases} m_\ell^t & \text{if } \ell+1 \notin \mathcal{B}_t, \\ m_{\ell+1}^t & \text{if } \ell+1 \in \mathcal{B}_t, \end{cases} \tag{2.30}$$

$$m_{\ell-1}^{t+1} = \begin{cases} m_{\ell-2}^t & \text{if } \ell-1 \in \mathcal{B}_t, \\ m_{\ell-1}^t & \text{if } \ell-1 \notin \mathcal{B}_t. \end{cases} \tag{2.31}$$

Equation (2.30), together with (2.28) imply $m_\ell^{t+1} \geq m_{\ell-1}^t$. Now if $\ell - 1 \notin \mathcal{B}_t$, by (2.31), $m_{\ell-1}^{t+1} = m_{\ell-1}^t \leq m_\ell^{t+1}$. Hence $\ell \notin \mathcal{B}_{t+1}$.

It remains to show (2.28) holds for all $t \geq 1$ by induction. The claim holds for $t = 1$ by construc-

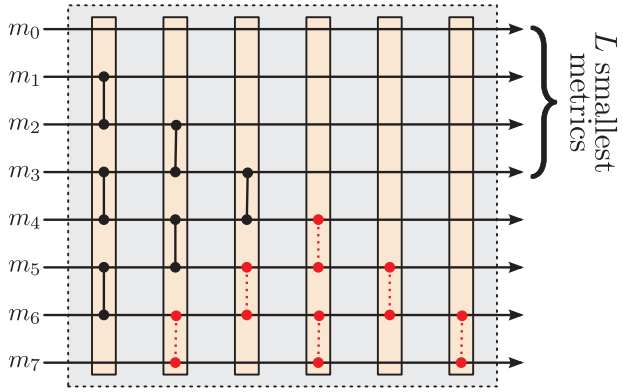


Figure 2.9 – Bubble sorter for $2L = 8$. The full bubble sorter requires all the depicted CAS units, while in the simplified bubble sorter all CAS units in red dotted lines can be removed.

tion; $\mathcal{B}_1 \subseteq \{2,4,\ldots,2L-2\}$ (because of (2.22b)) and (2.22a) is equivalent to (2.28) for $t = 1$.

Pick $\ell \in \mathcal{B}_{t+1}$. Assuming (2.28) holds for t, we know $m_\ell^{t+1} \geq m_{\ell-1}^t$ (as we just showed). Furthermore, since $\ell - 1 \in \mathcal{B}_t$ (and $\ell - 2 \notin \mathcal{B}_t$ due to (2)), (2.29) yields $m_{\ell-2}^{t+1} = m_{\ell-1}^t \leq m_\ell^{t+1}$. □

Property (ii) means we can replace the condition of the **if** block by $m_\ell^t \leq m_{\ell-1}^t$ without changing the algorithm. In other words, to determine whether we need to swap adjacent elements or not we can take a look at the values stored at that positions at the beginning of each round of the outer **while** loop. Furthermore, property (i) guarantees that each element, at each round, participates in at most one swap operation. As a consequence the inner **for** loop can be executed in parallel. Finally, property (iii) together with the initial condition $\mathcal{B}_1 \subseteq \{2,4,\ldots,2L-2\}$ implies that at odd rounds CAS operations take place only between the even-indexed elements and their preceding elements while at even rounds CAS operations take place only between the odd-indexed elements and their preceding elements.

Given the above considerations, we can implement the sorter in hardware as follows. The sorter has $2L-2$ *stages*, each of them implementing a round of bubble sort (i.e., an iteration of the **while** loop in Algorithm 5).[3] At round t of the bubble sort the first t elements are unchanged and the sorter will have a *triangular* structure. Since in our setting, round 1 is already eliminated, each stage t, $t = 1,2,\ldots,2L-2$ only moves the elements at indices $t, t+1,\ldots,2L-1$. Each stage implements the execution of the inner **for** loop in parallel using the required number of CAS units. In Figure 2.9 we show the structure of the sorter for $2L = 8$. Using simple counting arguments we can show that the full bubble sort requires $L(L-1)$ CAS units.

So far we have only discussed about the implementation of a sorter that sorts the entire list $\boldsymbol{m}$. However, we only need the first L ordered elements of the list. Hence, we can simplify the full

[3] In general the bubble sort terminates in up to $2L-1$ rounds but in our particular problem instance, since $m_0 = \mu_0$ is the smallest element of the list, the first round is eliminated.

sorter as follows. The first obvious simplification is to eliminate all the stages $L, L+1, \ldots, 2L-2$ since we know that after round $L-1$ of the bubble sort, the first L elements of the list correspond to an ordered list of the L smallest elements of the original list. Thus, the total number of required stages for this simplified bubble sorter is

$$s_{\text{tot}}^{\text{B}} = L - 1. \tag{2.32}$$

Furthermore, we note that, due to property (i), each element of the list at each round of the algorithm is moved at most by one position. Consider the elements at positions $2L-t, 2L-t+1, \ldots, 2L-1$ at round t. Since at most $L-t$ rounds of bubble sort are executed (including the current round), these elements cannot be moved to the first half of the list. Hence, we can eliminate the CAS units involving elements at indices $2L-t, 2L-t+1 \ldots, 2L-1$ at each stage $t = 1, 2, \ldots, L-1$ as well. The simplified bubble sorter thus requires

$$c_{\text{tot}}^{\text{B}} = \frac{1}{2} L(L-1) \tag{2.33}$$

CAS units. In Figure 2.9 the parts of the sorter that can be eliminated are drawn with red dotted lines.

2.2.3.5 Latency of Metric Sorting

We assume that the sorting procedure is carried out in a single clock cycle. A decoder based on any of the full sorters that solve the generic sorting problem of size $2L$, only needs to sort the path metrics for the information indices. Hence, the total sorting latency of such an implementation, measured in clock cycles, is

$$D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| = NR. \tag{2.34}$$

Using any of the pruned/simplified sorters, however, results in additional latency. This happens because additional sorting steps are required at the end of each contiguous set of frozen indices in order to ensure that property (2.22a) holds, as described in Section 2.2.3. Let $F_C(\mathcal{A})$ denote the number of *clusters* of frozen bits for a given information set $\mathcal{A}$.[4] The metric sorting latency using any of the pruned/simplified sorters, measured in clock cycles, is then

$$D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| + F_C(\mathcal{A}) = NR + F_C(\mathcal{A}). \tag{2.35}$$

[4] More precisely we assume $\mathcal{F} = \bigcup_{j=1}^{F_C(\mathcal{A})} \mathcal{F}_j$ such that (i) $\mathcal{F}_j \cap \mathcal{F}_{j'} = \emptyset$ if $j \neq j'$, i.e., $\{\mathcal{F}_j : j = 1, \ldots, F_C(\mathcal{A})\}$ is a partition of $\mathcal{F}$; (ii) for every j, $\mathcal{F}_j$ is a contiguous subset of $\{0, \ldots, N-1\}$; and (iii) for every pair $j \neq j'$, $\mathcal{F}_j \cup \mathcal{F}_{j'}$ is *not* a contiguous subset of $\{0, \ldots, N-1\}$. It can be easily checked that such a partition always exists and is unique.

2.3 Hardware Implementation Results

In this section, we present synthesis results for the SCL decoder architectures described in this chapter. For a fair comparison with [52], we use a TSMC 90 nm technology with a typical timing library (1 V supply voltage, 25° C operating temperature). All synthesis runs are performed with timing constraints that are not achievable, in order to assess the maximum achievable operating frequency of each design, as reported by the synthesis tool. For our synthesis results, we have used $P = 64$ PEs per SC decoder core, as in [38]. The hardware *efficiency* is defined as the throughput per unit area and it is measured in Mbps/mm^2. The decoding throughput of all decoders is:

$$T_{\text{SCL}}(N, P, \mathcal{A}, f) = \frac{f \cdot N}{D_{\text{SCL}}(N, P, \mathcal{A})}, \tag{2.36}$$

where f is the operating frequency of the decoder.

We first compare the various path metric sorters that were described in Section 2.2.3 in isolation. Then, we examine the effect of using a simplified metric sorter on our LLR-based SCL decoder and we compare our LLR-based decoder with our LL-based decoder in order to demonstrate the improvements obtained by moving to an LLR-based formulation of SCL decoding. Finally, we compare our LLR-based decoder with the LL-based decoder of [52] (since [52] is an improved version of [53], we do not compare directly with [53]) and [51]. A direct comparison with the SCL decoders of [56, 57] is unfortunately not possible, as the authors do not report their synthesis results in terms of mm^2. Finally, we provide some discussion on the effectiveness of a CA-SCLD.

2.3.1 Quantization Parameters

Before we continue with the implementation results, we need to examine the quantization bit-widths required by each type of decoder in order to ensure similar error-correcting performance for a fair comparison. In Figure 2.10 and Figure 2.11, we present the FER of floating-point and fixed-point implementations of an LL-based and an LLR-based SCL decoder for a $(1024, 512)$ polar code as a function of SNR.[5] For the floating-point simulations we have used the exact implementation of the decoder, i.e., for computing the LLRs the update rule f_- of (1.19a) is used and the path metric is iteratively updated according to (2.13). In contrast, for the fixed-point simulations we have used the MS approximation of the decoder given in (1.20)) and the approximated path metric update rule of (2.15).

We observe that the LL-based and the LLR-based SCL have practically indistinguishable FER performance when quantizing the channel LLs and the channel LLRs with $Q_{\text{LL}} = 4$ bits and $Q_{\text{LLR}} = 6$ bits respectively. Moreover, in our simulations we observe that the performance of the LL and the LLR-based SCL decoder is degraded significantly when $Q_{\text{LLR}} < 6$ and $Q_{\text{LL}} < 4$,

[5] The code is optimized for $E_b/N_0 = 2$ dB and constructed using the Monte-Carlo method of [6, Section IX].

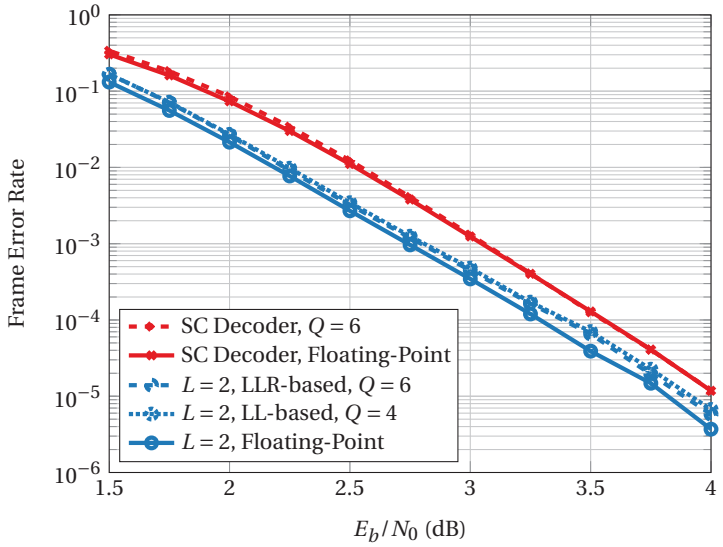


Figure 2.10 – The performance of floating-point vs. fixed-point SCL decoders ($L = 1$, i.e., SC decoding, and $L = 2$). $M = 8$ quantization bits are used for the path metric in fixed-point SCL decoders.

respectively. As discussed in Section 2.2.2.1, metric quantization requires at most $M = n + Q_{\text{LLR}} - 1$ bits for the LLR-based SCL decoder. However, in practice, much fewer bits turn out to be sufficient. For example, in our simulations for $N = 1024$ and $Q_{\text{LLR}} = 6$, setting $M = 8$ leads to the same performance as the worst-case $M = 15$, while setting $M = 7$ results in a significant performance degradation due to metric saturation. Thus, all synthesis results of this section are obtained for $Q_{\text{LL}} = 4$ for the LL-based decoder of Section 2.1, and $Q_{\text{LLR}} = 6$ and $M = 8$ for the LLR-based decoder of Section 2.2 for a fair (i.e., iso-FER) comparison.

The authors of [51] do not provide the FER curves for their fixed-point implementation of SCLD and the authors of [52] only provide the FERs for a CA-SCLD [52, Figure 2]. Nevertheless, we assume their quantization schemes will not result in a *better* FER performance for a *standard* SCLD than that of [7] since they both implement exactly the same algorithm as in [7] (using a different *architecture* than [7]).

2.3.2 Comparison of Path Metric Sorters

In this section, we compare the various path metric sorters described in Section 2.2.3 in isolation and for various list sizes, ranging from $L = 2$ up to $L = 32$ and using a path metric bit-width of $M = 8$ bits. We first compare the radix-$2L$ sorter and the bitonic sorter with their simplified counterparts, and then we compare all simplified sorters with each other. These synthesis results are useful in order to decide which path metric sorter should be used depending on the considered scenario. As we will see, the optimal decoder in terms of both operating frequency and area strongly depends on the employed list size L.

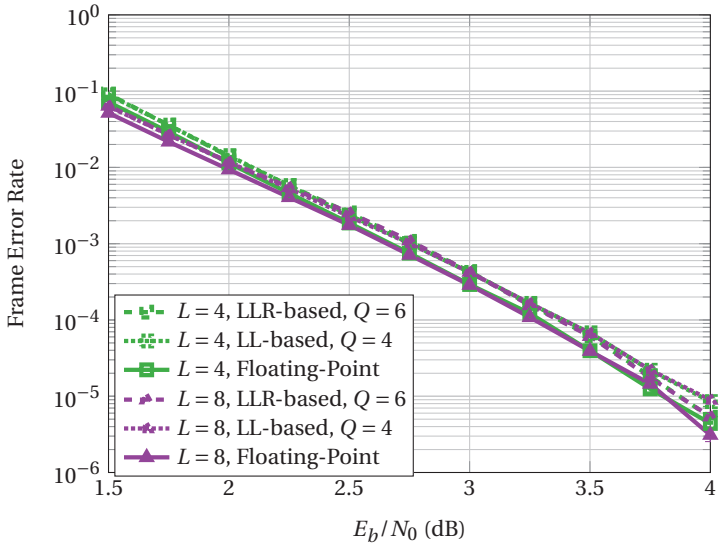


Figure 2.11 – The performance of floating-point vs. fixed-point SCL decoders ($L = 4$ and $L = 8$). $M = 8$ quantization bits are used for the path metric in fixed-point SCL decoders.

Table 2.1 – Synthesis Results for Radix-2L and Pruned Radix-2L Sorters

	Radix-2L		Pruned Radix-2L	
	Freq. (MHz)	Area (μm^2)	Freq. (MHz)	Area (μm^2)
$L = 2$	2128	3007	4545	608
$L = 4$	1111	12659	2083	3703
$L = 8$	526	50433	1031	18370
$L = 16$	229	238907	372	70746
$L = 32$	n/a*	n/a*	145	376945

* For $L = 32$ the synthesis tool ran out of memory on a machine with 48GB of RAM, most likely due to the lack of structure in the circuits that generate the control signals for the multiplexers in the radix-2L sorter.

2.3.2.1 Radix-2L Sorter vs. Pruned Radix-2L Sorter

In Table 2.1 we present synthesis results for the radix-2L sorter and the pruned radix-2L sorter. We observe that the pruned radix-2L sorter is at least 63% smaller and at least 56% faster than the full radix-2L sorter for all considered list sizes.

2.3.2.2 Bitonic Sorter vs. Pruned Bitonic Sorter

In Table 2.2 we present synthesis results for the bitonic sorter of [53] and the pruned bitonic sorter presented in this paper. We observe that, as discussed in Section 2.2.3.3, the improvement in terms of both area and operating frequency are diminishing as the list size L is increased. Nevertheless, even for $L = 32$ the pruned bitonic sorter is 5% faster and 14% smaller than the full bitonic sorter.

Table 2.2 – Synthesis Results for Bitonic and Pruned Bitonic Sorters

	Bitonic		Pruned Bitonic	
	Freq. (MHz)	Area (μm^2)	Freq. (MHz)	Area (μm^2)
$L = 2$	1370	2109	4545	608
$L = 4$	676	8745	952	3965
$L = 8$	347	27159	478	20748
$L = 16$	214	82258	256	69769
$L = 32$	157	238721	166	205478

2.3.2.3 Simplified Bubble Sorter vs. Pruned Radix-$2L$ and Pruned Bitonic Sorter

In Table 2.3 we present synthesis results for the simplified bubble sorter described in Section 2.2.3.4. We observe that, for $L \leq 8$, the simplified bubble sorter has a lower delay than the pruned bitonic sorter. This happens because, as can be verified by evaluating (2.23) and (2.32), for $L \leq 8$ the bubble sorter has fewer stages than the pruned bitonic sorter while for $L > 8$ the situation is reversed. A similar behavior can be observed for the area of the sorters, where the bubble sorter remains smaller than the pruned bitonic sorter for $L \leq 16$.

We also observe that, for $L \leq 16$, the pruned radix-$2L$ sorter is faster than the other two sorters and similar in area to the pruned bitonic sorter, while the simplified bubble sorter is significantly smaller. Thus, for $L \leq 16$ the pruned bitonic sorter is not a viable option, while trade-offs between speed and area can be made by using either the pruned radix-$2L$ sorter or the simplified bubble sorter. For $L = 32$, however, the pruned bitonic sorter has a higher operating frequency *and* a smaller area than the other two sorters.

For the remainder of this section we only provide synthesis results for various SCL decoders with list size up to $L = 8$, where the only simplified metric sorters of interest are the pruned radix-$2L$ sorter and the bubble sorter. Since the sorter area is generally small compared to the rest of the decoder, we chose to only use the larger pruned radix-$2L$ sorter which is, however, significantly faster than the bubble sorter.

Recall that in Section 2.2.3 we mentioned that an LLR-based SCL decoder with a simplified sorter needs to solve a general sorting problem of size L after exiting a group of frozen synthetic channels. We have explained that a simple solution for this problem is to use a simplified sorter for the problem of size $2L$ and a general sorter for the problem of size L. From Table 2.1 and Table 2.2 we observe that this solution results in a higher overall operating frequency and, in most cases, a lower area compared to the case where we use a general sorter for the problem of size $2L$ (in which case we do not need the smaller sorter of size L). For example, consider an SCL decoder with $L = 4$ which requires a simplified sorter for $L = 4$ and a general sorter for $L = 2$. Using a radix-$2L$ sorter, the operating frequency is limited by the pruned radix-$2L$ sorter. Hence, adding the smaller full radix-$2L$ sorter does not affect the maximum operating frequency. Moreover, the combined area of the pruned radix-$2L$ sorter for $L = 4$ and the full radix-$2L$ sorter for $L = 2$ is smaller than the area of the full radix-$2L$ sorter for $L = 4$.

Table 2.3 – Comparison of Pruned Radix-2L, Pruned Bitonic, and Simplified Bubble Sorters

	Pruned Radix-2L		Pruned Bitonic		Simplified Bubble	
	Freq. (MHz)	Area (μm^2)	Freq. (MHz)	Area (μm^2)	Freq. (MHz)	Area (μm^2)
$L=2$*	4545	608	4545	608	4545	608
$L=4$	2083	3703	952	3965	1388	2756
$L=8$	1031	18370	478	20748	534	11726
$L=16$	372	70746	256	69769	247	51159
$L=32$	145	376945	166	205478	127	212477

* For $L=2$ it can easily be seen that all three sorters are equivalent.

Table 2.4 – LLR-based SCL Decoder: Radix-2L vs. Pruned Radix-2L Sorter

	Radix-2L Sorter			Pruned Radix-2L Sorter		
	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$	$L=8$
Freq. (MHz)	847	758	415	848	794	637
Lat. (Cyc./bit)	2.53	2.53	2.53	2.59	2.59	2.59
T/P (Mbps)	335	299	164	328	307	246
Area (mm^2)	0.88	1.75	3.87	0.9	1.78	3.85
Efficiency	380	171	42	364	172	64

2.3.3 LLR-based SCL Decoder: Radix-2L Sorter versus Pruned Radix-2L Sorter

One may expect an LLR-based SCL decoder using the pruned radix-2L sorter to always outperform an LLR-based SCL decoder using the non-pruned radix-2L sorter. However, the decoder equipped with the pruned radix-2L sorter needs to stall slightly more often to perform the additional sorting steps after groups of frozen bits. In particular, a $(1024,512)$ polar code contains $F_C(\mathcal{A}) = 57$ groups of frozen bits. Therefore, the total sorting latency for the pruned radix-2L sorter is $D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| + F_C(\mathcal{A}) = 569$ cycles (see (2.35)), while the total sorting latency for the non-pruned radix-2L sorter is $D_{\text{MS}}(\mathcal{A}) = |\mathcal{A}| = 512$ cycles (see (2.34)). Thus, the SCL decoder with the pruned sorter exhibits a latency of $D_{\text{SCL}}(N,P,\mathcal{A}) = 2649$ cycles, which is an increase of approximately 2% compared to the decoder equipped with a full radix-2L sorter. Therefore, if using the pruned radix-2L does not lead to a more than 2% higher clock frequency, the decoding throughput will actually be reduced.

As can be observed in Table 2.4, this is exactly the case for $L=2$, where the LLR-based SCL decoder with the pruned radix-2L sorter has a 2% *lower* throughput than the LLR-based SCL decoder with the full radix-2L sorter. However, for $L \geq 4$ the metric sorter starts to lie on the critical path of the decoder and therefore using the pruned radix-2L sorter results in a significant increase in throughput of up to 50% for $L=8$.

To provide more insight into the effect of the metric sorter on our SCL decoder, in Table 2.5 we present the metric sorter delay and the critical path start- and endpoints of each decoder of Table 2.4. The critical paths for $L=2$ and $L=4,8$, are also annotated in Figure 2.3 with green dashed lines and red dotted lines, respectively. We denote the register of the controller which stores the internal LLR memory read address by R_{IM}. Moreover, let $D_{\hat{U}_s}$ and D_M denote

Table 2.5 – Metric Sorter Delay and Critical Path Start- and Endpoints for our LLR-Based SCL Decoder Using the Radix-2L and the Pruned Radix-2L Sorters.

	Radix-2L Sorter			Pruned Radix-2L Sorter		
	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$	$L=8$
Delay (ns)	0.50*	0.80	1.83	0.50*	0.54	1.09
CP Startpoint	R_{IM}	D_M	D_M	R_{IM}	R_{IM}	D_M
CP Endpoint	D_M	$D_{\hat{U}_s}$	$D_{\hat{U}_s}$	D_M	D_M	$D_{\hat{U}_s}$

* Note that the true delay of the pruned radix-2L sorter is always smaller than the delay of the radix-2L sorter. However, for $L=2$, both sorters meet the synthesis timing constraint, which was set to 0.50 ns.

Table 2.6 – SCL Decoder Synthesis Results ($R=\frac{1}{2}$, $N=1024$)

	LLR-Based			LL-Based			LL-Based [52][a]			LL-Based [51][b]	
	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$
Technology	TSMC 90nm			TSMC 90nm			TSMC 90nm			Scaled to 90nm[c]	
Freq. (MHz)	847	794	637	794	730	408	507	492	462	361	289
Lat. (Cycles/bit)	2.53	2.59	2.59	2.53	2.53	2.53	2.53	2.53	3.03	1.00	1.00
T/P (Mbps)	335	307	246	314	288	161	200	194	153	362	290
Area (mm^2)	0.88	1.78	3.58	1.38	2.62	5.38	1.23	2.46	5.28	2.03	4.10
Efficiency	380	172	69	227	110	30	163	79	29	178	71

[a] The synthesis results in [52] are provided with up to 16 PEs per path. The reported numbers in this table are the corresponding synthesis results using 64 PEs per path and are courtesy of the authors of [52].
[b] The authors of [51] use 3 quantization bits for the channel LLs and a tree SC architecture, while [7, 52] use 4 quantization bits for the channel LLs and a semi-parallel architecture with $P=64$ PEs per path.
[c] We use the standard assumption that area scales as s^2 and frequency scales as $1/s$, where s is the feature size.

a register of the partial sum memory and the metric memory, respectively. From Table 2.5, we observe that, for $L=2$, the radix-2L sorter does not lie on the critical path of the decoder, which explains why using the pruned radix-2L sorter does not improve the operating frequency of the decoder. For $L \geq 4$ the metric sorter does lie on the critical path of the decoder and using the pruned radix-2L sorter results in a significant increase in the operating frequency of up to 53%. It is interesting to note that using the pruned radix-2L sorter eliminates the metric sorter completely from the critical path of the decoder for $L=4$. For $L=8$, even the pruned radix-2L sorter lies on the critical path of the decoder, but the delay through the sorter is reduced by 40%.

2.3.4 LLR-based SCL Decoder: Comparison with LL-based SCL Decoders

In Table 2.6, we compare our LLR-based decoder with the LL-based decoders of [52] and [51] along with our LL-based decoder of Section 2.1. For the comparisons, we pick our LLR-based SCL decoder with the best hardware efficiency for each list size, i.e., for $L=2$ we pick the SCL decoder with the radix-2L sorter, while for $L=4,8$, we pick the SCL decoder with the pruned radix-2L sorter. Moreover, we pick the decoders with the best hardware efficiency from [51], i.e., the 4b-rSCL decoders.

Table 2.7 – Comparison of LLR-based implementation with existing LL-based implementations

	LL-Based			LLR-Based		
	$L=2$	$L=4$	$L=8$	$L=2$	$L=4$	$L=8$
Freq. (MHz)	794	730	408	847	758	415
Lat. (Cyc./bit)	2.53	2.53	2.53	2.53	2.53	2.53
T/P (Mbps)	314	288	161	335	299	164
Area (mm^2)	1.38	2.62	5.38	0.88	1.75	3.87
Efficiency	227	110	30	380	171	42

Table 2.8 – Cell Area Breakdown for the LL-Based and the Radix-2L LLR-based SCL Decoders ($R=\frac{1}{2}$, $N=1024$)

	LL-Based	LLR-Based	Reduction
List Size		$L=2$	
Total Area (mm^2)	1.38	0.88	36%
Memory (mm^2)	1.07	0.80	25%
MCU (mm^2)	0.28	0.06	79%
Metric Sorter (mm^2)	1.34×10^{-3}	0.75×10^{-3}	44%
Other (mm^2)	0.03	0.02	50%
List Size		$L=4$	
Total Area (mm^2)	2.62	1.75	33%
Memory (mm^2)	1.92	1.57	18%
MCU (mm^2)	0.54	0.11	80%
Metric Sorter (mm^2)	13.92×10^{-3}	9.23×10^{-3}	33%
Other (mm^2)	0.15	0.06	60%
List Size		$L=8$	
Total Area (mm^2)	5.38	3.87	28%
Memory (mm^2)	4.08	3.46	15%
MCU (mm^2)	0.82	0.18	78%
Metric Sorter (mm^2)	70.65×10^{-3}	54.05×10^{-3}	24%
Other (mm^2)	0.41	0.18	56%

2.3.4.1 Comparison with LL-Based Decoder of Section 2.1

Our LL-based architecture of Section 2.1 and the LLR-based architecture with the radix-2L sorter presented in Section 2.2 are identical except that the former uses LLs while the latter uses LLRs. Therefore, by comparing these two architectures we can specifically identify the improvements in terms of area and decoding throughput that arise directly from the reformulation of SCL decoding in the LLR domain.

We recall that the cycle count for our SCL decoder using the radix-2L sorter when decoding a $(1024,512)$ polar code is $D_{\text{SCL}}(N,P,\mathcal{A})=2592$ cycles (see (2.19) and (2.34)).

From Table 2.7, we see that our LLR-based SCL decoder occupies 36%, 33%, and 28% less area than our LL-based SCL decoder for $L=2$, $L=4$, and $L=8$, respectively. We present the area breakdown of the LL-based and the LLR-based decoders in Table 2.8 in order to identify where the area reduction mainly comes from and why the relative reduction in area decreases with increasing list size L. The *memory* area corresponds to the combined area of the LLR (or LL) memory, the partial sum memory, and the path memory. We observe that, in absolute terms,

the most significant savings in terms of area come from the memory for $L = 2$ and from the MCU for $L = 4, 8$. On the other hand, in relative terms, the biggest savings in terms of area always come from the MCU with an average area reduction of 79%. The relative reduction in the memory area decreases with increasing list size L. This happens because each bit-cell of the partial sum memory and the path memory contains L-to-L crossbars, whose size grows quadratically with L, while the LL (and LLR) memory grows only linearly in size with L. Thus, the size of the partial sum memory and the path memory, which are not affected by the LLR-based reformulation, becomes more significant as the list size is increased, and the relative reduction due to the LLR-based formulation is decreased. Similarly, the relative reduction in the metric sorter area decreases with increasing L, because the LLR-based formulation only decreases the bit-width of the $L(2L-1)$ comparators of the radix-$2L$ sorter but it does not affect the size of the sorting logic, which dominates the sorter area as the list size is increased.

From Table 2.7, we observe that the operating frequency (and, hence, the throughput) of our LLR-based decoder is 7%, 3%, and 2% higher than that of our LL-based SCL decoder of [7] for $L = 2$, $L = 4$, and $L = 8$, respectively. Even though in this comparison we did not use any of the simplified sorters, the operating frequency of the LLR-based SCL decoder is increased for all list sizes because the bit-width of all quantities involved in decoding is reduced quite significantly due to the LLR-based reformulation of the SCL decoding algorithm.

Due to the aforementioned improvements in area and decoding throughput, we conclude that the LLR-based reformulation of SCL decoding leads to hardware decoders with 67%, 55%, and 40% better hardware efficiency than their corresponding LL-based decoders, for $L = 2$, $L = 4$, and $L = 8$, respectively.

2.3.4.2 Comparison with Other Existing Decoders

From Table 2.6 we observe that our LLR-based SCL decoder has an approximately 28% smaller area than the LL-based SCL decoder of [52] for all list sizes. Moreover, the throughput of our LLR-based SCL decoder is up to 70% higher than the throughput achieved by the LL-based SCL decoder of [52], leading to a 137%, 118%, and 120% better hardware efficiency for $L = 2$, $L = 4$ and $L = 8$, respectively. The synthesis results of [51] are given for a 65 nm technology, which makes a fair comparison difficult. Nevertheless, in order to enable as fair a comparison as possible, we scale the area and the frequency to a 90 nm technology in Table 2.6. Moreover, the authors of [51] only provide synthesis results for $L = 2$ and $L = 4$. In terms of area, we observe that our decoder is approximately 57% smaller than the decoder of [51] for all list sizes. We also observe that for $L = 2$ our decoder has a 7% lower throughput than the decoder of [51], but for $L = 4$ the throughput of our decoder is 6% higher than that of [51]. Overall, the hardware efficiency of our LLR-based SCL decoder is 115% and 142% better than that of [51] for $L = 2$ and $L = 4$ respectively.

2.3.5 CRC-Aided SCL Decoder

As discussed in Section 1.3.3.2, the performance of the SCL decoder can be significantly improved if it is assisted for its final choice by means of a CRC which rejects some incorrect codewords from the final set of L candidates. However, there is a trade-off between the length of the CRC and the performance gain. A longer CRC, rejects more incorrect codewords but, at the same time, it degrades the performance of the inner polar code by increasing its rate [21]. Hence, the CRC improves the overall performance if the performance degradation of the inner polar code is compensated by rejecting the incorrect codewords in the final list.

2.3.5.1 Choice of CRC

We picked three different CRCs of lengths $r = 4$, $r = 8$ and $r = 16$ from [58] with generator polynomials:

$$g(x) = x^4 + x + 1, \tag{2.37a}$$
$$g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1, \text{ and} \tag{2.37b}$$
$$g(x) = x^{16} + x^{15} + x^2 + 1, \tag{2.37c}$$

respectively and evaluated the empirical performance of the SCL decoders of list sizes of $L = 2$, $L = 4$, $L = 8$, aided by each of these three CRCs in the regime of $E_b/N_0 = 1$ dB to $E_b/N_0 = 4$ dB. The results are shown in Figure 2.12.

For $L = 2$, using either the CRC-4 or the CRC-8 (represented by generator polynomials (2.37a) and (2.37b) respectively) improves the performance of the standard SCL decoder. In contrast, for the CRC-16 the performance degradation of the inner polar code becomes dominant at $E_b/N_0 \leq 2.75$ dB causing the CA-SCLD to perform slightly worse than the standard SCL decoder. At higher SNRs the performance of the CA-SCLD with CRC-16 is better than a standard SCL decoder but not better than that of a CA-SCLD with shorter CRCs. The CRC-aided SCL decoders with CRC-4 and CRC-8 have almost the same block-error probability (the block-error probability of the CA-SCLD with CRC-8 is only marginally better than that of the CA-SCLD with CRC-4 at $E_b/N_0 \geq 3.25$ dB). Given this observation and the fact that increasing the length of CRC decreases the throughput of the decoder (see Section 2.3.5.2), we conclude that the CRC-4 of (2.37a) is a reasonable choice for a CA-SCLD with list size $L = 2$.

For $L = 4$, allocating $r = 8$ bits for the CRC of (2.37b) turns out to be the most beneficial option. CRC-4 and CRC-8 will lead to almost identical FER at $E_b/N_0 \leq 2.25$ dB while CRC-8 improves the FER significantly more than CRC-4 at higher SNRs. Furthermore, CRC-16 leads to the same performance as CRC-8 at high SNRs and worse performance than CRC-8 in low-SNR regime.

Finally, for $L = 8$ we observe that CRC-16 of (2.37c) is the best candidate among the three different CRCs in the sense that the performance of the CA-SCLD which uses this CRC is significantly better than that of the decoders using CRC-4 or CRC-8 for $E_b/N_0 > 2.5$ dB, while

Table 2.9 – Throughput Reduction in CRC-Aided SCL Decoders

		$L=2$	$L=4$	$L=8$
	Freq. (MHz)	847	794	637
SCLD	$\lvert\mathcal{A}\rvert$	512	512	512
	$F_C(\mathcal{A})$	57	57	57
	Lat. (Cycles)	2592	2649	2649
	T/P (Mbits/s)	335	307	246
CA-SCLD	$\lvert\mathcal{A}\rvert$	516	520	528
	$F_C(\mathcal{A})$	55	54	52
	Lat. (Cycles)	2596	2654	2660
	T/P (Mbits/s)	334	306	245
	Reduction (%)	0.2	0.2	0.4

all three decoders have almost the same FER at lower SNRs (and they all perform better than a standard SCL decoder).

2.3.5.2 Throughput Reduction

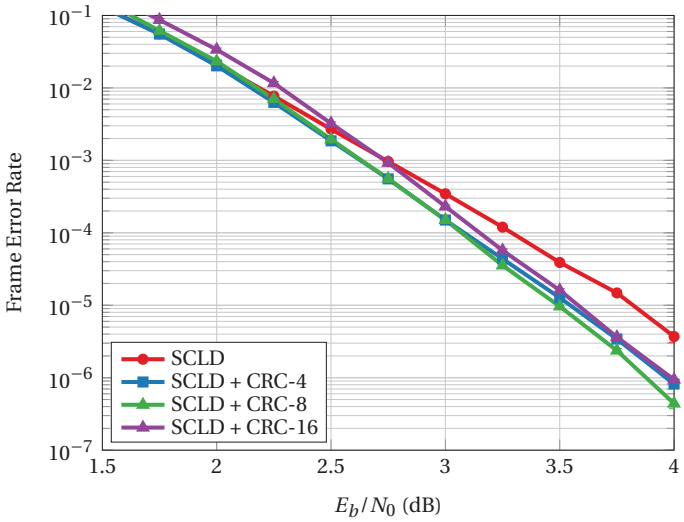
Adding r bits of CRC increases the number of information bits by r, while reducing the number of groups of frozen channels by *at most* r. As a result, the sorting latency is generally increased, resulting in a decrease in the throughput of the decoder. In Table 2.9 we have computed this decrease in the throughput for different decoders and we see that the CRC-aided SCL decoders have slightly (at most 0.4%) reduced throughput. For this table, we have picked the best decoder at each list size in terms of hardware efficiency from Table 2.4.

2.3.5.3 Effectiveness of CRC

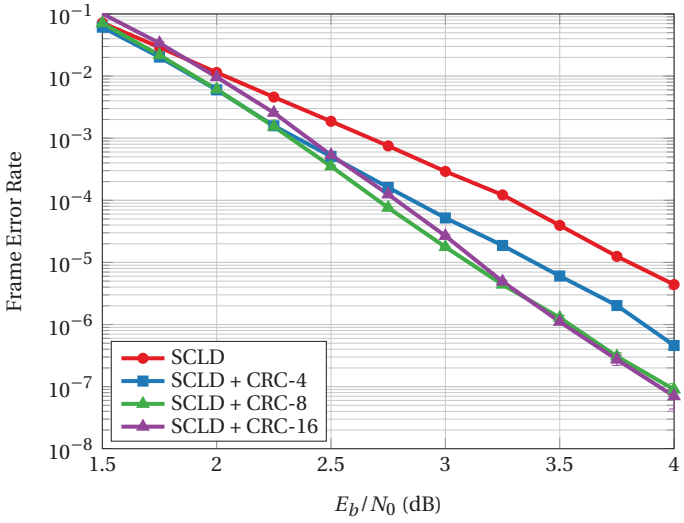
The area of the CRC unit for all synthesized decoders is in less than 1 μm^2 for the employed TSMC 90 nm technology. Moreover, the CRC unit does not lie on the critical path of the decoder. Therefore, it does not affect the maximum achievable operating frequency. Thus the incorporation of a CRC unit is a highly effective method of improving the performance of an SCL decoder. For example, it is interesting to note that the CA-SCLD with $L=2$ has a somewhat lower FER than the standard SCL decoder with $L=8$ (in both floating-point and fixed-point versions) in the regime of $E_b/N_0 > 2.5$ dB. Therefore, if a FER in the range of 10^{-3} to 10^{-6} is required by the application, using a CA-SCLD with list size $L=2$ is preferable to a standard SCL decoder with list size $L=8$ as the former has more than five times higher hardware efficiency.

2.3.5.4 SC Decoding or SCL Decoding?

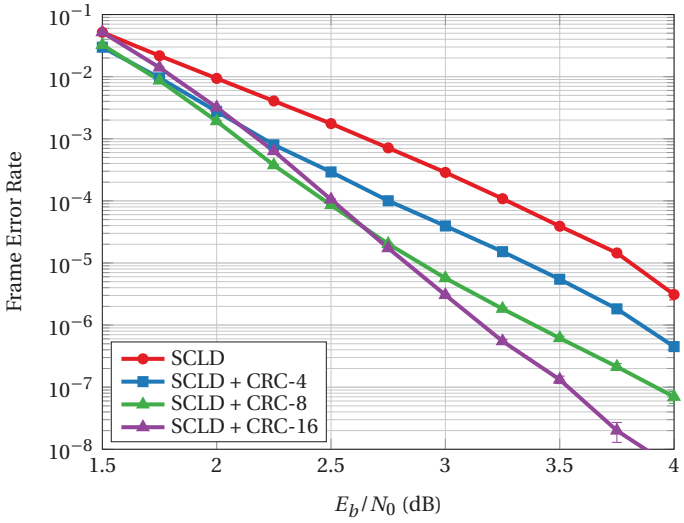
Modern communication standards sometimes allow very long blocklengths to be used. For example, the DVB-S2 standard [33] for digital video broadcasting over satellite links uses LDPC codes of blocklength up to $N=64800$. The error-rate performance of polar codes under



(a) $L = 2$



(b) $L = 4$



(c) $L = 8$

Figure 2.12 – The performance of LLR-based SCL decoders compared to that of CRC-aided SCL decoders for $L = 2, 4, 8$.

Table 2.10 – LLR-Based SC Decoder vs. SCL Decoder Synthesis Results

	SC	CA-SCLD $L = 2$, CRC-4	SC	CA-SCLD $L = 4$, CRC-8
N	2048	1024	4096	1024
Freq. (MHz)	870	847	806	794
Lat. (Cyc./bit)	2.05	2.54	2.06	2.59
Lat. (Cyc.)	4192	2596	8448	2654
T/P (Mbps)	425	334	391	306
Area (mm^2)	0.78	0.88	1.51	1.78

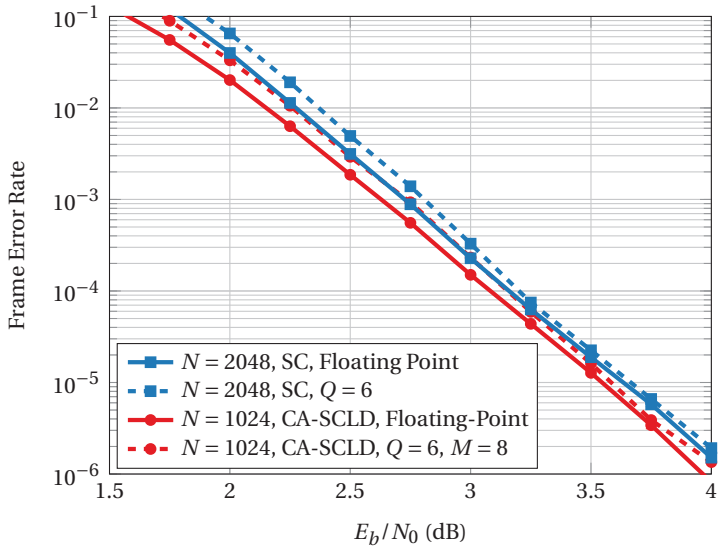
conventional SC decoding is significantly improved if the blocklength is increased. However, a long blocklength implies long decoding latency and large decoders. Thus, an interesting question is whether it is better to use a long polar code with SC decoding or a shorter one with SCL decoding, for a given target block-error probability. In order to answer this question, we first need to find some pairs of short and long polar codes which have approximately the same block-error probability under SCL and SC decoding, respectively to carry out a fair comparison.

In Figure 2.13a we see that a $(2048, 1024)$ polar code has almost the same FER under SC decoding as a $(1024, 512)$ modified polar code under CA-SCLD with list size $L = 2$ and CRC-4 of (2.37a). Similarly, in Figure 2.13b we see that a $(4096, 2048)$ polar code has almost the same FER under SC decoding as an $(1024, 512)$ modified polar code decoded under CA-SCLD with list size $L = 4$ and CRC-8 of (2.37b).

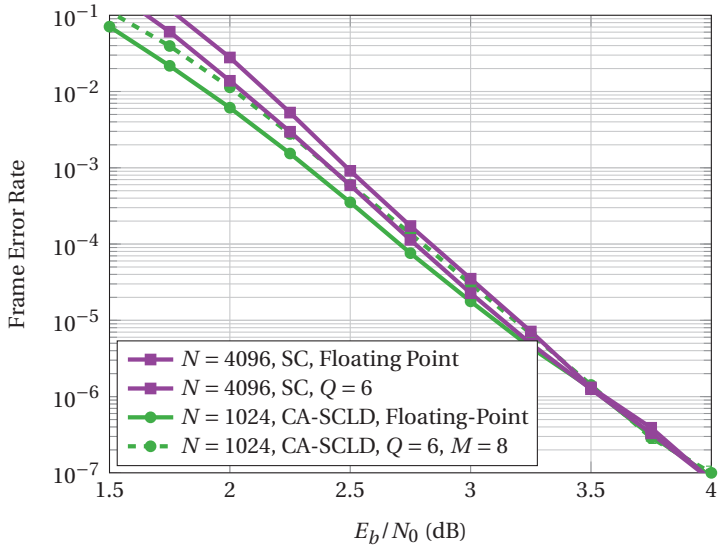
As mentioned earlier, our SCL decoder architecture is based on the SC decoder of [38]. In Table 2.10 we present the synthesis results for the SC decoder of [38] at block lengths $N = 2048$ and $N = 4096$ and compare them with that of our LLR-based SCL decoder, when using the same TSMC 90nm technology and identical operating conditions. For all decoders, we use $P = 64$ PEs per path and $Q_{\text{LLR}} = 6$ bits for the quantization of the LLRs.

First, we see that the SCL decoders occupy an approximately 15% larger area than their SC decoder counterparts. This may seem surprising, as it can be verified that an SC decoder for a code of length LN requires more memory (LLR and partial sum) than the memory (LLR, partial sum, and path) required by an SCL decoder with list size L for a code of length N, and we know that the memory occupies the largest fraction of both decoders. This discrepancy is due to the fact that the copying mechanism for the partial sum memory and the path memory still uses $L \times L$ crossbars, which occupy significant area. It is an interesting open problem to develop an architecture that eliminates the need for these crossbars.

Moreover, we observe that both SC decoders can achieve a slightly higher operating frequency than their corresponding SCL decoders, although the difference is less than 3%. However, the per-bit latency of the SC decoders is about 20% smaller than that of the SCL decoders, due to the sorting step involved in SCL decoding. The smaller per-bit latency of the SC decoders combined with their slightly higher operating frequency, make the SC decoders have an almost 27% higher throughput than their corresponding SCL decoders.



(a) A (2048, 1024) polar code under SC decoding versus a (1024, 512) modified polar code under CA-SCLD with $L = 2$ and CRC-4 with generator polynomial (2.37a).



(b) A (4096, 2048) polar code under SC decoding versus a (1024, 512) modified polar code under CA-SCLD with $L = 4$ and CRC-8 with generator polynomial (2.37b).

Figure 2.13 – CA-SCLD with $L = 2, 4$, results in the same performance at blocklength $N = 1024$ as the conventional SC decoding with $N = 2048$ and $N = 4096$, respectively.

However, from Table 2.10 we see that the SCL decoders have a significantly lower per-codeword latency. More specifically, the SCL decoder with $N = 1024$ and $L = 2$ has a 38% lower per-codeword latency than the SC decoder with $N = 2048$, and the SCL decoder with $N = 1024$ and $L = 4$ has a 68% lower per-codeword latency than the SC decoder with $N = 4096$. Thus, for a fixed FER, our LLR-based SCL decoders provide a solution of reducing the per-codeword latency at a small cost in terms of area, rendering them more suitable for low-latency applications than their corresponding SC decoders.

2.4 Polar Decoder Survey and Comparison with Existing Decoders

In the previous section, we have presented two hardware architectures for SCL decoding of polar codes. However, there also exist several other decoding algorithms for polar codes, where the most popular are the standard SC decoding algorithm and the BP decoding algorithm, which are both described in Section 1.3. Over the past few years, significant advances have been made in the hardware implementation of SC, BP, as well as SCL decoders for polar codes, leading to improvements in both the achievable throughput and in the area requirements. Most hardware implementations target application-specific integrated circuits (ASICs), but there also exist several implementations of polar decoders on field-programmable gate arrays (FPGAs) and even some high-speed software implementations for software-defined radio (SDR) applications.

In this section, we summarize and systematically categorize the techniques that have been used and the results that have been presented in the polar hardware decoder literature, mainly focusing on the ASIC implementations. Moreover, we compare the error-correcting performance and hardware efficiency of polar decoders with several LDPC and Turbo hardware decoders that are used in current standards. More specifically, in Section 2.4.1, we summarize the techniques that lead to the advances in the implementation of hardware polar decoders and we compare the resulting decoders in terms of their area and energy efficiency. Section 2.4.2, on the other hand, focuses on a comparison of the error-correcting performance and the hardware efficiency of polar decoders with LDPC and Turbo codes used in existing telecommunications standards. More specifically, we present an in-depth comparison with the LDPC codes used in the IEEE 802.11ad (WiGig) [32], IEEE 802.11n (Wi-Fi) [31], and IEEE 802.3an (10 Gb/s Ethernet) [30] standards, as well as the Turbo code used in the 3GPP LTE [59] standard.

We note that error-correcting performance comparisons can sporadically already be found in the literature. For example, [60] compared an FPGA-based BP polar decoder with an FPGA-based decoder for the Turbo code of the IEEE 802.16e standard. Moreover, [40] compared an FPGA-based SC decoder with an FPGA-based decoder for the LDPC code of the IEEE 802.3an standard. The authors of [21] compared the error-correcting performance of SCL decoding with the error-correcting performance of the LDPC code used in the IEEE 802.16e standard. Finally, [61] compared SCL decoding with the LDPC codes used in the IEEE 802.11n and IEEE 802.3an standards. However, these comparisons were not systematic and no comparison of the corresponding hardware implementations was made.

2.4.1 Polar Hardware Decoders

2.4.1.1 Successive Cancellation Hardware Decoders

In this section, we provide an overview of the evolution of SC decoder hardware implementation over the past years. A summary of all SC decoder VLSI implementation results, along with

Table 2.11 – SC Decoder Hardware Implementations

Work	Main Contribution	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	T/P (Mb/s)	Power (mW)
Leroux *et al.* [62]	Linear memory requirement.	1024	65	0.36	500	239	n/a
Mishra *et al.* [42]	First ASIC, two-bit decoding.	1024	180	1.71	150	98	67
Leroux *et al.* [38]	Semi-parallel architecture.	1024	65	0.31	500	246	n/a
Fan *et al.* [41]	Efficient partial sum computation.	1024	65	0.07	1010	497	n/a
Yuan *et al.* [63]	Two-bit decoding with precomputation.	1024	45	0.64	750	500	n/a
Lin *et al.* [64]	SCAN decoder.	1024	90	0.97	571	958	n/a
Che *et al.* [65]	Implementation of fast-SSC decoding.	1024	45	0.28	1040	2001	n/a
Dizdar *et al.* [66]	Single-cycle implementation.	1024	90	3.21	2	2560	191
Giard *et al.* [67]	Fast-SSC decoding for low-rate codes.	1024	65	0.69	600	1860	215
Giard *et al.* [68]	Unrolled decoder implementation.	1024	65	3.22	361	18489	534
Lin *et al.* [69]	Fast-SSC SCAN decoder.	1024	90	1.01	471	1435	n/a

the main contribution of each work, is given in Table 2.11.

First Steps In his original paper, Arıkan already alluded to a high-level *isomorphic* SC decoder architecture, where each of the $N \log N$ nodes of the DDG is directly mapped to a *processing element* (PE) in hardware. The output of each PE is stored in a register, meaning that $N \log N$ registers are required for storage. Assuming that each node operation requires a single clock cycle, it can be shown that such an architecture requires $2N-2$ clock cycles to decode a single codeword [20] (i.e., it has a *latency* of $2N-2$ clock cycles). Since each node only needs to be activated once to decode a single codeword, almost all of the nodes are idle throughout most of the decoding procedure.

More efficient SC decoder hardware architectures were proposed in [20, 62]. More specifically, it was pointed out that at stage s of the DDG, only 2^s PEs can be activated simultaneously. Thus, a *tree* architecture with a full binary tree of PEs of depth $n-1$ has the same latency as an isomorphic architecture with $N \log N$ PEs. We note that the PEs of the tree decoder need to support both the f_+ and the f_- update functions and it was assumed in [20, 62] that their complexity is two times larger than the complexity of the PEs used in the isomorphic architecture. Through a similar resource sharing argument, it can be shown that $2N-1$ registers are sufficient to store the intermediate values produced by the PEs. Furthermore, since only a single stage of the DDG is activated at each clock cycle and the maximum number of nodes that are activated at once is 2^{n-1}, the tree decoder can be further simplified to a *line* decoder architecture with only 2^{n-1} PEs, which has the same latency as a tree decoder.

It was observed in [38] that the stages that need the fewest PEs (i.e., they are the least parallelizable) actually account for most of the decoding latency, as they are activated most often. Thus, using a *semi-parallel* decoder architecture that only instantiates $P < 2^{n-1}$ PEs has a small impact on the decoding latency, provided that P is not too small, while providing significant savings in terms of hardware resources. More specifically, the decoding latency for the semi-parallel architecture with P PEs is $2N + \frac{N}{P} \log\left(\frac{N}{4P}\right)$ clock cycles.

The aforementioned SC decoder architectures are summarized in Table 2.12. We observe that

Table 2.12 – Complexity and Decoding Latency of Different SC Decoder Architectures

	PEs	Registers	Latency (Cycles)
Isomorphic [6, 20, 62]	$N \log N$	$N(\log N + 1)$	$2N-2$
Tree [20, 62]	$2N-2$	$2N-1$	$2N-2$
Line [20, 62]	N	$2N-1$	$2N-2$
Semi-parallel [38]	P	$2N-1$	$2N + \frac{N}{P} \log\left(\frac{N}{4P}\right)$

the line architecture has the same decoding latency as the isomorphic and tree architectures, but with a much lower cost in terms of hardware. Moreover, the semi-parallel architecture provides meaningful trade-offs between hardware complexity and decoding latency. Thus, it is not surprising that practically all of the SC (and SCL) decoders that followed [20, 62, 38] use either the line architecture of the semi-parallel architecture.

Multi-Bit Hardware Decoders The successive nature of the SC decoding algorithm makes large-scale parallelization, which is necessary for high-throughput decoding, challenging. Hence, the line and semi-parallel SC decoder architectures suffer from relatively high decoding latency and, thus, low decoding throughput. One way to reduce the decoding latency is to decode multiple bits simultaneously. For example, in the work of [42], which presented the first ASIC implementation of an SC decoder, two bits are decoded simultaneously. In particular u_{2i+1} can be decoded in parallel with u_{2i} by pre-computing both possible outcomes of the f_- function (i.e., for $u_{2i} = 0$ and for $u_{2i} = 1$) and selecting the appropriate result when $\hat{u}_{2i}$ becomes available. For two-bit decoding, the additional cost for the pre-computation and the selection of u_{2i+1} is negligible both in terms of hardware resources and in terms of the critical path. Moreover, the decoding latency is reduced by $N/2$ clock cycles, effectively increasing the throughput of the SC decoder by 25%. Two-bit decoding was also considered in [63]. This approach can be generalized to decode more than two bits simultaneously, as described in the context of SCL decoding in [51]. However, the returns are diminishing as the hardware cost increases exponentially in the number of simultaneously decoded bits, but the decoding latency only decreases linearly.

Fast-SSC Hardware Decoders It is possible to reduce the complexity of multi-bit decoders by exploiting the pattern of frozen and non-frozen bit channels. As a simple example, if we know that u_{2i} is a frozen bit, then we do not need to pre-compute two values to decode u_{2i+1}, since we know that $u_{2i} = 0$. As a more involved example, consider a polar code of length $N = 4$ where only u_3 is an information bit. It can easily be verified from the encoding process that this is in fact a repetition code of length $N = 4$, where the only two valid codewords are $\boldsymbol{c}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$ and $\boldsymbol{c}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$. Instead of traversing the DDG using the standard SC decoding algorithm, which would take 7 clock cycles with a semi-parallel architecture with $P = 2$, it is possible to directly decode this repetition code by summing up all the input LLRs and taking a hard decision on the sum in a single clock cycle. Other patterns, such as single parity-check codes, can be identified and decoded efficiently, saving a significant

Table 2.13 – BP Decoder Hardware Implementations

Work	Main Contribution	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	Max. Iter.	Max. T/P (Mb/s)	Sust. T/P (Mb/s)	Power (mW)
Park *et al.* [72]	Unidirectional schedule.	1024	65	1.48	300	15	4676	2048	478
Yuan *et al.* [73]	Early termination.	1024	45	1.04	500	40	4500	2588	990
Abbas *et al.* [74]	Sub-graph freezing.	1024	45	0.75	197	15	1683	1683	n/a
Lin *et al.* [75]	Adaptive quantization.	1024	65	1.40	769	5	7870	7870	442

amount of clock cycles. This is the main idea behind the *fast-simplified SC* (fast-SSC) decoding algorithm [40], where dedicated decoders are used in order to simultaneously decode groups of bits that have various frozen and non-frozen bit patterns.

Unrolled Hardware Decoders The main idea behind unrolled decoders is that the SC decoding recursion can be (completely or partially) unrolled and mapped to hardware. The first fully unrolled SC decoder, based on the fast-SSC decoding algorithm, was presented in [70] and it achieves a decoding throughput of 237 Gb/s, but at a significant cost in terms of hardware complexity and power.[6] Partially unrolled fast-SSC decoders were presented in [67], which reduce the hardware complexity while still achieving multi-Gb/s throughputs. A disadvantage of the standard unrolled decoders of [70, 67] is that they can only decode a single polar code (i.e., only a fixed code rate and blocklength), while non-unrolled SC decoders are inherently flexible. However, there may still be practical applications since there are examples of standards (e.g., IEEE 802.3an) that only define a single code. However, a flexible unrolled decoder architecture was also presented in [68]. This decoder can decode a master polar code and several shorter polar codes of various rates with a small penalty in terms of the area requirements with respect to a non-flexible decoder. Moreover, a single-cycle unrolled decoder that remains flexible by not applying any code-specific simplifications was presented in [66].

SCAN Hardware Decoders As explained in Section 1.3.3.4, a SCAN decoder is essentially a BP decoder with an SC schedule. An FPGA implementation of SCAN decoding was presented in [71]. A SCAN decoder that also includes ideas from fast-SSC decoding was presented in [64], leading to a soft-output decoder with significantly lower area requirements than conventional BP decoders. Some more ideas from fast-SSC decoding were added to the same decoder in the work of [69], further increasing the decoding throughput.

2.4.1.2 Belief Propagation Hardware Decoders

In this section, we describe the main ideas behind VLSI implementations of BP polar decoders in the literature. The main contributions and results of each paper are summarized in Table 2.13. We note that in Table 2.13 we make a distinction between the maximum throughput

[6] We note that the decoder of [70] was implemented on an FPGA and is thus not directly included in our comparison.

and the sustained throughput of each BP decoder. All presented BP decoders use some form of early termination, meaning that the average number of iterations is lowered, but the runtime of the BP decoder becomes variable. Thus, some codeword buffers are needed in order for the decoder to be able to sustain its maximum throughput, but all presented area results exclude these buffers. Thus, for fair comparison with SC and SCL decoders, we also present the sustained throughput, which is the throughput of the BP decoder when it always uses its maximum decoding iterations.

First Steps The first ASIC of a BP polar decoder was presented in [72]. This decoder uses a single column of N bi-directional PEs that calculate both the left-to-right and the right-to-left messages at the same time. This allows the decoder to process all $N \log N$ nodes of the DDG in $\log N$ clock cycles, thus effectively performing one BP iteration per $\log N$ clock cycles. Furthermore, using bidirectional PEs also reduces the message storage memory requirements by 50%. Moreover, the authors of [72] use a latch-based memory and a bit-splitting register file with logic-in-memory circuits, which reduce congestion and lead to a high area utilization of 85%.

Early Termination When decoding LDPC codes, early termination is performed using the parity-check matrix $\mathbf{H}$, since for any valid codeword $\mathbf{c}$ we have $\mathbf{Hc} = \mathbf{0}$, and decoding is usually terminated when $\mathbf{H\hat{c}} = \mathbf{0}$, where $\hat{\mathbf{c}}$ denotes the codeword estimate produced by the LDPC decoder. However, with polar codes the decoder output is an estimate of the information vector $\hat{\mathbf{u}}$, which cannot directly be used to detect a valid codeword. For this reason, other early termination methods have been explored in the literature. For example, the authors of [72] already used a simple early termination scheme, where decoding halts when the bit-decisions do not change for three consecutive BP iterations. More early termination methods were explored in [73], where a $\boldsymbol{G}_N$ matrix based and a *minimum LLR* based early termination method is proposed. In the $\boldsymbol{G}_N$ matrix based early termination method, hard decisions are taken on both sides of the DDG, leading to an estimate of $\hat{\mathbf{u}}$ and an estimate of $\hat{\mathbf{c}}$ and if $\hat{\mathbf{u}}\mathbf{G} = \hat{\mathbf{c}}$ holds, decoding halts. The minimum LLR criterion examines all the decision LLRs and if the minimum absolute value of the decision LLRs is above some predefined threshold β, then the decisions are assumed to be sufficiently reliable and decoding halts. A method to adapt the threshold β to the channel SNR is also presented in [72]. An alternative early termination method, which is based on *subgraph freezing*, was presented in [74]. In this early termination method, hard decisions are taken on sub-vectors of $\hat{\mathbf{c}}$, which correspond to constituent codes of the polar code. These sub-vector decisions are frozen (i.e., no longer updated in the BP schedule) if re-encoding them leads to an information sub-vector where the frozen bit positions have the correct frozen values. Decoding halts when all sub-vectors of $\hat{\mathbf{c}}$ have been frozen.

2.4.1.3 Successive Cancellation List Hardware Decoders

In this section, we provide an overview of the evolution of SCL decoder hardware implementation over the past years. A summary of all SCL decoder VLSI implementation results, along with the main contribution of each work, is given in Table 2.14.

First Steps The first log-likelihood (LL) based hardware implementation of an SCL decoder was presented in [7]. We note that this decoder is part of our work and it was described in detail in Section 2.1. The proposed architecture essentially consists of L semi-parallel SC decoders that compute the L path metrics in parallel. The L paths only interact when stage n of the DDG is reached, at which point the $2L$ candidate path metrics need to be sorted and each of the L active paths is either duplicated, discarded, or just kept active. A smart copying is used, which copies pointers to banks of LLs instead of copying the LLs themselves, thus saving saving a large amounts of crossbars that would be needed to directly copy the LL banks from one path to another. A similar architecture that also supports CRC-aided decoding, but does not use the smart copying mechanism, was presented in [53, 52].

LLR-Based SCL Decoders As explained in Section 1.3.3.2, the original SCL decoding algorithm was described using likelihoods. While this description is mathematically valid, it causes significant numerical problems and leads to costly hardware implementations. For this reason, the first hardware implementations of [7, 53, 76] reformulated the SCL decoding algorithm in the log-likelihood domain. While this reformulation provides increased numerical stability with respect to the likelihood based formulation, it still requires large amounts of storage and complex message update rules.

SC decoding can be reformulated in the LLR domain in a straightforward manner, but the reformulation of the SCL decoding algorithm turns out to be more complicated. To this end, the authors of [8, 9] introduced a cumulative path metric that is updated iteratively based on the decision LLRs. This enables the hardware implementation of the SCL decoding algorithm using L parallel LLR-based SC decoder cores, which are both more compact and faster than their log-likelihood based counterparts. We note that this decoder is also part of our work and it was described in detail in Section 2.2. The same LLR-based cumulative path metric was later derived independently in [77].

Path Metric Sorting In SCL decoding, when reaching a node of stage n of the DDG that corresponds to an information bit, the L active paths are expanded into $2L$ candidate paths, out of which the L paths with the best path metric are kept and the rest are discarded. The most common approach in the literature is to sort the $2L$ paths with respect to their path metrics and to simply select the L first paths. We note that sorting can be ascending or descending, depending on the exact definition of the path metric, but the sorting procedure is essentially the same in both cases. This step of the SCL decoding algorithm becomes increasingly complex

as the list size L is increased and, in most hardware architectures, the path metric sorting unit is the limiting factor in terms of the critical path.

The first SCL decoder hardware architecture of [7] used a simple radix-$2L$ sorting network, while subsequent implementations used more sophisticated sorting networks, such as the bitonic sorting network [53, 51, 52] and the Batcher odd–even mergesort network [77]. The LLR-based path metric introduced in [7, 77] has some properties that can also simplify the task of path metric sorting. These properties were exploited in [10, 78] in order to derive sorting networks that are tailored to SCL decoding, which have lower hardware complexity and shorter critical paths than their more generic counterparts. Finally, the authors of [79, 80] considered approximating the path sorting step, leading to simplified hardware implementations at the cost of some error correction performance degradation.

Multi-bit Hardware SCL Decoders The multi-bit decoding approaches of [42, 63] can be readily extended to SCL decoding. For example, [51] considered multi-bit decoding and presented a hardware architecture for LL-based SCL decoding. The multi-bit approach was later extended to LLR-based SCL decoding for two [81] and multiple [82] simultaneously decoded bits. A similar approach, which groups multiple bits into symbols and transforms the SCL decoder to a symbol-based SCL decoder was presented in [83] and is shown to offer similar decoding throughput improvements compared to standard multi-bit decoding, but with lower decoding complexity.

Fast-SSC Based Hardware SCL Decoders The family of fast-SSC decoders is not applicable verbatim to the LLR-based SCL decoder. The reason for this complication is that, the path metric must be updated even when frozen bits are encountered. Nevertheless, an SCL decoder hardware implementation based on fast-SSC with some approximations of the path metric updates was presented in [80]. An additional important algorithmic step in the direction of incorporating more techniques from fast-SSC into SCL decoding was recently made in [61]. However, the authors of [61] did not provide a hardware implementation of their described algorithm.

2.4.1.4 Polar Decoder Comparison

For the hardware comparison, the main metrics of interest are: area (mm^2), decoding throughput (Mb/s), and power (mW). Unfortunately, as can be seen in Tables 2.11–2.14, power results for polar decoders are scarce, making a useful comparison with existing decoders difficult. Thus, for the comparison of polar decoders with LDPC and Turbo decoders we mainly consider the area and the decoding time complexity (which is the inverse of the decoding throughput). These metrics are shown on a double-logarithmic plot where the area is on the vertical axis and the time complexity is on the horizontal axis. We note that hardware efficiency is defined as unit area per decoded bit and is measured in mm^2/bits/s. Thus, on the aforementioned plot,

Table 2.14 – SCL Decoder Hardware Implementations ($L = 4$)

Work	Main Contribution	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	T/P (Mb/s)
Balatsoukas-Stimming *et al.* [7]	First architecture, efficient path copying.	1024	90	2.62	730	288
Lin *et al.* [53]	First CRC-aided decoder.	1024	90	3.02	657	216
Balatsoukas-Stimming *et al.* [9]	LLR-based implementation.	1024	90	1.78	794	307
Fan *et al.* [79]	Low-complexity path metric sorting.	1024	90	n/a	n/a	n/a
Hashemi *et al.* [84]	Reduced memory requirements.	2048	90	1.36	500	164
Lin *et al.* [52]	Efficient memory and metric sorting.	1024	90	1.13	476	173
Lin *et al.* [80]	Fast-SSC decoding with approximations.	1024	90	3.83	403	1140
Xiong *et al.* [85]	Partial ML decoding, flexibility.	1024	90	1.89	409	1094
Yuan *et al.* [51]	Multi-bit decision LL-based decoding.	1024	65	2.14	400	401
Xiong *et al.* [83]	Symbol-based SCL decoding.	1024	90	1.21	500	313
Yuan *et al.* [82]	Multi-bit decision LLR-based decoding.	1024	65	1.18	360	675

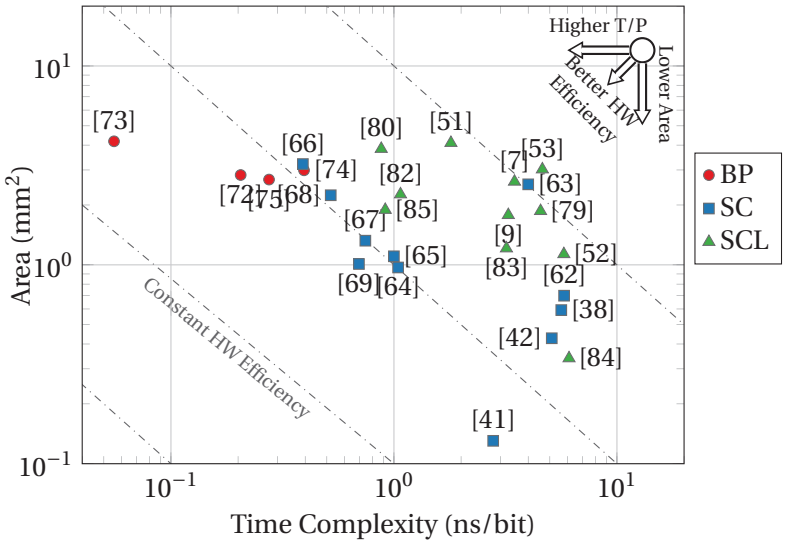


Figure 2.14 – Time complexity vs. area for various decoders for polar codes. All decoders are given for $N = 1024$. The SCL decoder implementations are given for $L = 4$. The area and operating frequency are normalized to 90 nm CMOS technology using standard technology scaling rules.

lines with a slope of -1 correspond to iso-hardware efficiency lines. In Figure 2.14, we provide a summary of the area and time complexity of VLSI implementations of SC, BP, and SCL polar decoders. All synthesis results are scaled to a 90 nm CMOS technology. We use standard Dennard scaling laws [86], so that the area scales as s^2 and the operating frequency scales as $1/s$, where s is the technology feature size. We observe that BP decoders generally provide very high throughputs, although they are matched by some of the most recent fast-SSC based SC decoders. We note, however, that some fast-SSC decoders only work for a specific rate and that BP decoding provides soft output values, which are required for iterative receivers. Moreover, the BP decoders also generally have the highest area requirements of all decoders. SCL decoders generally have the lowest throughput of all decoders, as well as higher area requirements than SC decoders and similar area requirements to BP decoders. However, they provide significantly improved error-correcting performance with respect to both SC and BP

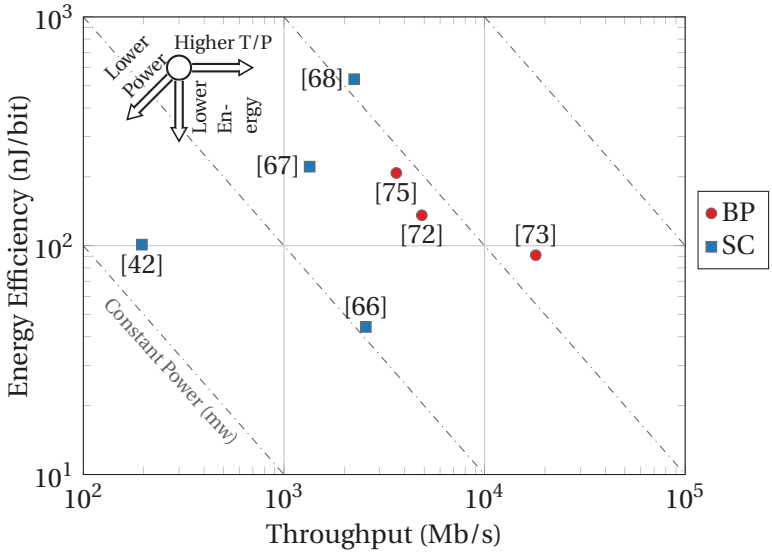


Figure 2.15 – Throughput vs. power for various decoders for polar codes. The power and operating frequency are normalized to 90 nm CMOS and 1 V using standard technology scaling rules.

Table 2.15 – Properties of the LDPC and Turbo codes used for comparison.

	Blocklength	Throughput	Performance	Rates
IEEE 802.11n	Short-medium	Medium	Medium	Medium-high
IEEE 802.11ad	Short	High	Medium	Medium-high
IEEE 802.3an	Medium	Very high	Good	High
3GPP LTE	Short-long	Medium	Very Good	Low-high

decoding.

In Figure 2.15, we plot the energy efficiency of the few SC and BP decoder implementations that report power results as a function of the decoding throughput. We note that the lines with slope -1 in this plot correspond to iso-power lines. We observe that the SC decoders have significantly lower power requirements than the BP decoders. However, the throughput of the reported SC decoders is also significantly lower than the throughput of the BP decoders, leading to similar energy efficiency numbers of both types of decoders.

2.4.2 Comparison of Polar Codes with LDPC and Turbo Codes

In this section, we compare polar code decoders with decoder for the LDPC and Turbo codes used in some current communications standards, both in terms of error-correcting performance and in terms of their corresponding hardware implementations. More specifically, we perform a comparison with the LDPC codes used in the IEEE 802.11ad (WiGig) [32], IEEE 802.11n (Wi-Fi) [31], and IEEE 802.3an (10 Gb/s Ethernet) [30] standards, as well as the Turbo code used in the 3GPP LTE [59] standard. These codes were selected to cover a wide range of

scenarios in terms of blocklength, throughput, and error-correcting performance, as summarized in Table 2.15.

2.4.2.1 Comparison Setup

For the comparison of the error-correcting performance we use floating-point versions of the decoding algorithms, since the quantization parameters of the hardware decoders are usually chosen so that the performance loss with respect to the floating-point implementation is negligible. Moreover, for all simulations the encoded codewords are modulated using BPSK and they are transmitted over an AWGN channel. For almost all decoders for polar codes and LDPC codes we use the (scaled or offset) *min-sum* approximation for check node updates. The scaling and/or offset factor is given, whenever applicable. The Turbo decoder for the Turbo code of the LTE standard also uses the corresponding log-likelihood domain approximation, which we refer to as the *max-log* approximation. All polar codes are designed using the Monte Carlo based method of Arıkan [6]. In order to speed up our simulations of BP decoding for polar codes, we used the **G** matrix based early termination method of [73], which has negligible impact on the error-correcting performance. For the SCL decoders, we use the following CRC polynomials

$$\text{CRC-8}: \quad g_8(x) = x^8 + x^5 + x^4 + x^3 + 1, \tag{2.38}$$

$$\text{CRC-16}: \quad g_{16}(x) = x^{16} + x^{15} + x^2 + 1, \tag{2.39}$$

$$\text{CRC-32}: \quad g_{32}(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + + x^5 + x^4 + x^3 + 1. \tag{2.40}$$

The comparison of hardware decoders is performed in two stages. First, we compare the existing polar decoders with the decoders for the LDPC or Turbo code in question by only considering technology scaling to normalize the area and operating frequency. All decoders are scaled to a 90 nm CMOS technology. As previously, we use Dennard scaling laws [86], so that the area scales as s^2 and the operating frequency scales as $1/s$, where s is the technology feature size. For all comparisons, we also provide tables with the original (unscaled) results as found in the literature for completeness.

In the second stage, we perform a hardware comparison by selecting parameters for the polar decoders (e.g., blocklength, list size, number of iterations) that lead to an error-correcting performance that is as close as possible to that of the competing LDPC or Turbo codes. In other words, the second stage is an *iso-FER* comparison. In order to scale the area of the reference polar decoders with the blocklength and list size (in addition to technology scaling), we make the following assumptions: The area of the SC and SCL decoders scales linearly with the blocklength, and the area of the BP decoders scales as $N \log N$. The area of the SCL decoders scales linearly with the list size. The decoding latency of the BP decoders scales linearly with the maximum number of iterations. As it is very difficult to predict the frequency scaling

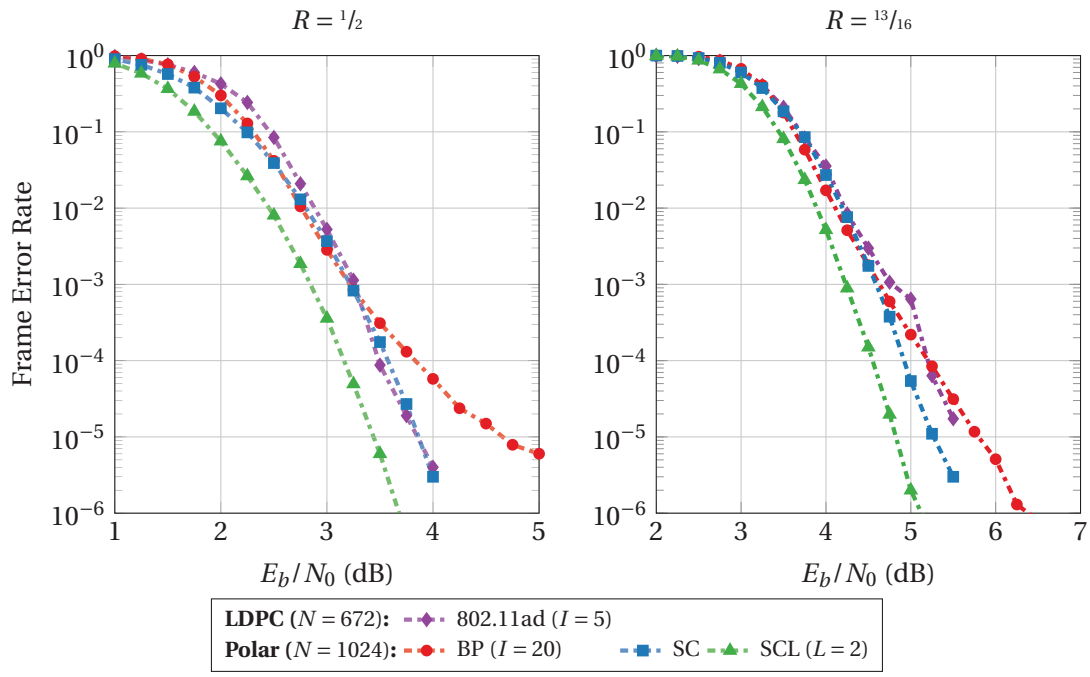


Figure 2.16 – Performance of the LDPC code of the IEEE 802.11ad standard compared to polar codes under SC decoding, BP decoding, and SCL decoding (8-bit CRC).

with respect to the aforementioned parameters, we use the original, non-scaled, operating frequency results.

2.4.2.2 Polar Codes vs. IEEE 802.11ad LDPC Codes

The IEEE 802.11ad standard [32] uses quasi-cyclic (QC) LDPC codes with a blocklength of $N = 672$ and code rates $R \in \{\frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{13}{16}\}$. We simulated the performance of this LDPC code using a layered offset min-sum decoding algorithm with a maximum of $I = 5$ iterations and an offset of $\beta = 0.2$, which are numbers commonly found in the literature, as can be seen in in Table 2.16. We provide a comparison for the lowest rate $\left(R = \frac{1}{2}\right)$ and the highest rate $\left(R = \frac{13}{16}\right)$ found in the IEEE 802.11ad standard.

Plain Comparison We first compare the performance of the IEEE 802.11ad LDPC code with the performance of polar codes with blocklength $N = 1024$. The polar codes are decoded using BP decoding with $I = 20$ maximum iterations and a scaling factor of $\alpha = 0.9375$, SC decoding, and SCL decoding with $L = 2$ and a CRC of 8 bits. The codes for $R = \frac{1}{2}$ and $R = \frac{13}{16}$ were designed for an SNR of 1 dB and 4 dB, respectively. In Figure 2.16, we observe that SC decoding has similar performance to the IEEE 802.11ad LDPC code for both $R = \frac{1}{2}$ and $R = \frac{13}{16}$. BP decoding, on the other hand, has approximately 0.75 dB worse performance than the IEEE 802.11ad LDPC codes for $R = \frac{1}{2}$ at a FER of 10^{-5}, while for $R = \frac{13}{16}$ the performance of polar codes and

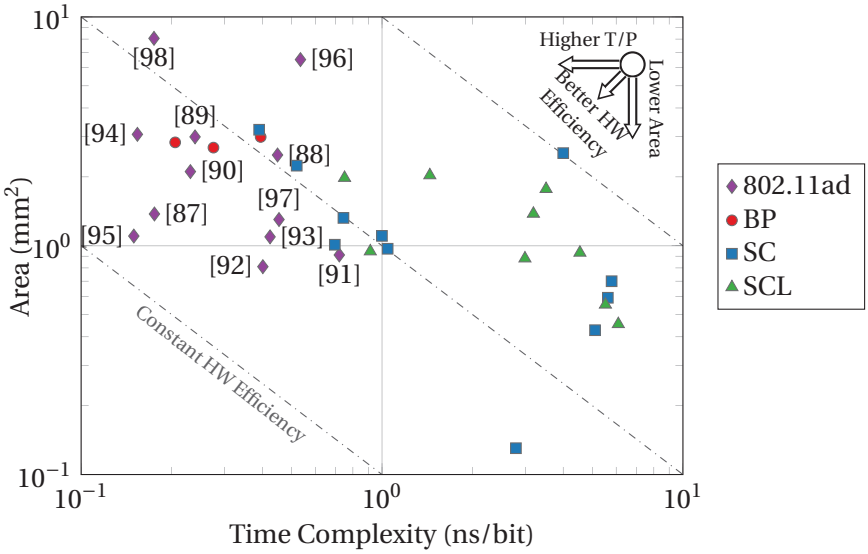


Figure 2.17 – Hardware efficiency of IEEE 802.11ad LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling.

the IEEE 802.11ad LDPC code is very similar. Finally, SCL decoding with $L = 2$ has better performance than the IEEE 802.11ad LDPC codes, resulting in a gain of approximately 0.25 dB at a FER of 10^{-5} for both $R = \frac{1}{2}$ and $R = \frac{13}{16}$.

In Figure 2.17, we compare the hardware efficiency of several IEEE 802.11ad LDPC decoders with the hardware efficiency of SC, BP, and SCL decoders for polar codes, when only considering technology scaling and not the iso-FER case. The original results for the LDPC decoders are summarized in Table 2.16. We observe that the best SC and all BP based polar decoders compete well in terms of area, throughput, and hardware efficiency with the LDPC decoders. SCL decoders, on the other hand, have lower hardware efficiency in general, mainly due to their lower throughput.

Iso-FER Comparison The iso-FER comparison in this case is relatively simple, since SC and BP decoding with $N = 1024$ already perform very similarly to the LDPC codes of the IEEE 802.11ad standard. SCL decoding with $N = 1024$ and $L = 2$, on the other hand, performs better than the LDPC code of the IEEE 802.11ad standard, meaning that the blocklength of the polar code can potentially be reduced. In Figure 2.18, we observe that SCL decoding with $L = 2$, a CRC of 8 bits, and a blocklength of $N = 512$ is indeed sufficient to match the error-correcting performance of the longer LDPC code. We note that the $N = 512$ codes used for SCL decoding for $R = \frac{1}{2}$ and $R = \frac{13}{16}$ were designed for an SNR of 1 dB and 4 dB, respectively.

In Figure 2.19 we observe that, when considering the iso-FER case, the hardware efficiency of SC and BP decoders is unaffected, while the SCL decoders have an improved area efficiency, due to the reduced area requirements from the shorter blocklength, which is, in some cases,

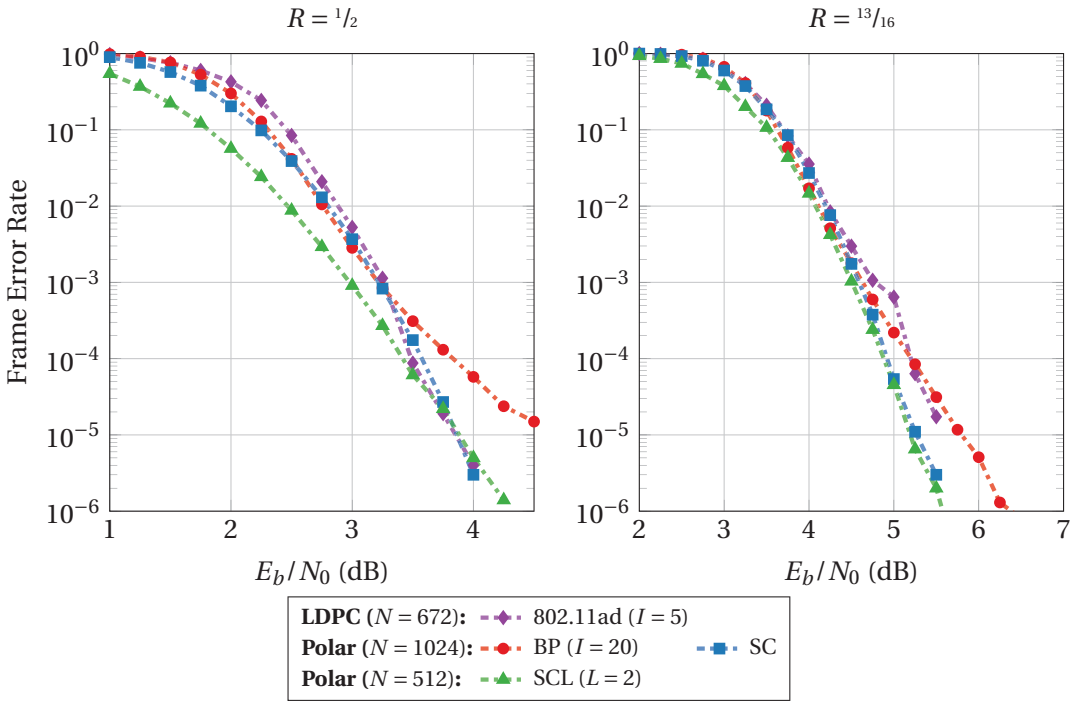


Figure 2.18 – Performance of the LDPC code of the IEEE 802.11ad standard compared to polar codes under SC decoding and BP decoding, and a polar code with $N = 512$ under SCL decoding (8-bit CRC).

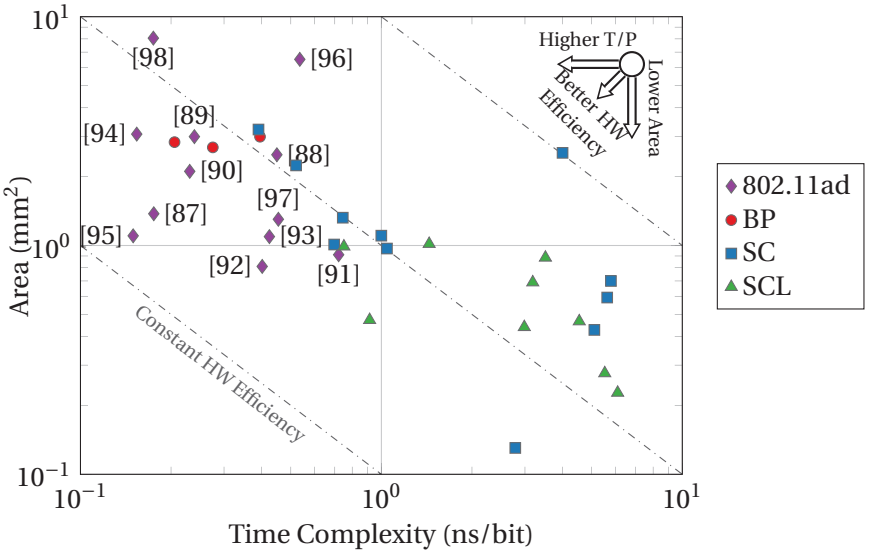


Figure 2.19 – Hardware efficiency of IEEE 802.11ad LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER.

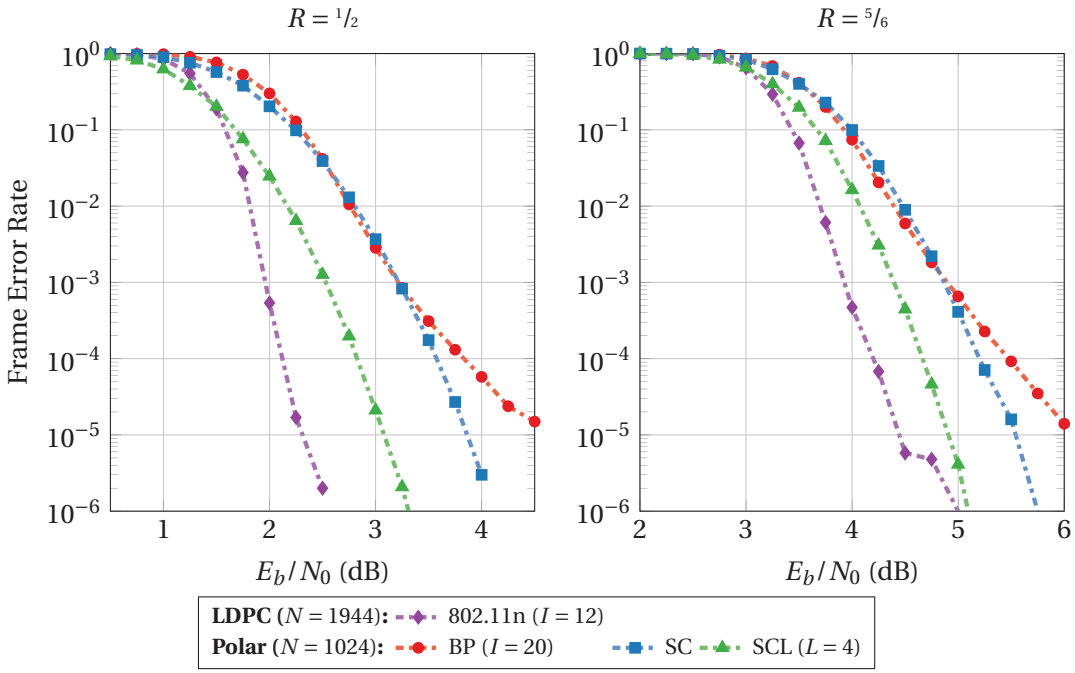


Figure 2.20 – Performance of the LDPC code of IEEE 802.11n standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding, and SCL decoding (8-bit CRC).

comparable to that of the IEEE 802.11ad LDPC decoders. However, the throughput of most SCL decoders is still not comparable to that of the IEEE 802.11ad LDPC decoders.

2.4.2.3 Polar Codes vs. IEEE 802.11n LDPC Codes

The IEEE 802.11n standard [31] uses QC-LDPC codes with blocklengths of $N \in \{648, 1296, 1944\}$ and code rates $R \in \{\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}\}$. We simulated the performance of this LDPC code using a layered offset min-sum decoding algorithm with a maximum of $I = 12$ iterations and an offset of $\beta = 0.5$, which are numbers commonly found in the literature as can be seen in Table 2.17. We provide a comparison for $N = 1944$ and for the lowest rate $\left(R = \frac{1}{2}\right)$ and the highest rate $\left(R = \frac{5}{6}\right)$ found in the IEEE 802.11n standard.

Plain Comparison We first compare the performance of the IEEE 802.11n LDPC code with the performance of polar codes with blocklength $N = 1024$. The polar codes are decoded using BP decoding with $I = 20$ maximum iterations and a scaling factor of $\alpha = 0.9375$, SC decoding, and SCL decoding with $L = 4$ and a CRC of 8 bits. The polar codes for $R = \frac{1}{2}$ and $R = \frac{5}{6}$ were designed for an SNR of 1 dB and 5 dB, respectively. In Figure 2.20, we observe that polar codes under BP decoding have a 2.25 dB and 1.5 dB loss at a FER of 10^{-5} with respect to the LDPC code of the IEEE 802.11n standard for $R = \frac{1}{2}$ and $R = \frac{5}{6}$, respectively. Polar codes under SC decoding perform slightly better at low FERs, having a loss of 1.5 dB and 1 dB compared to the LDPC code at a FER of 10^{-5} for $R = \frac{1}{2}$ and $R = \frac{5}{6}$, respectively. Polar codes under SCL decoding

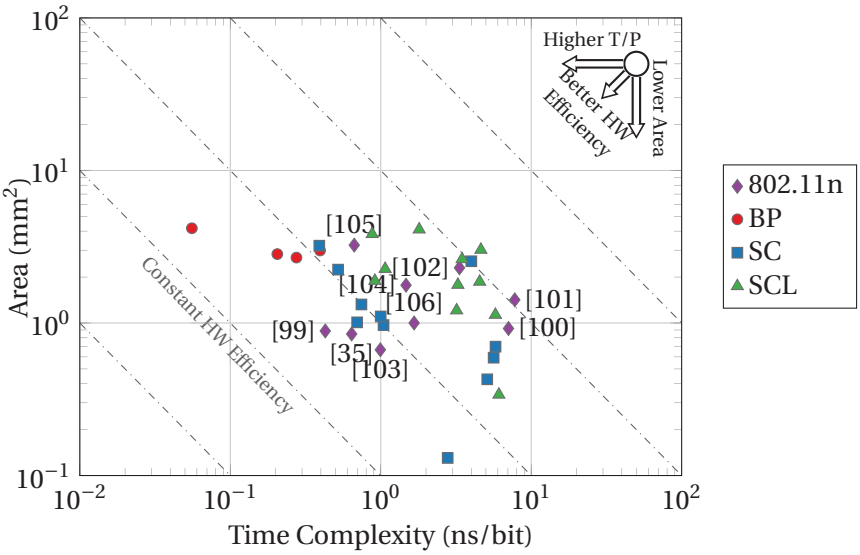


Figure 2.21 – Hardware efficiency of IEEE 802.11n LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling.

with $L = 4$ provide the best performance with a loss of only 0.75 dB and 0.5 dB at a FER of 10^{-5} for $R = \frac{1}{2}$ and $R = \frac{5}{6}$, respectively.

In Figure 2.21, we compare the hardware efficiency of several IEEE 802.11n LDPC decoders with the hardware efficiency of SC, BP, and SCL decoders for polar codes, when only considering technology scaling and not the iso-FER case. The original results for the IEEE 802.11n LDPC decoders are summarized in Table 2.17. We observe that the best SC and BP decoders have practically the same hardware efficiency with the best IEEE 802.11n LDPC decoders. Moreover, the SCL decoders, which can more closely match the IEEE 802.11n LDPC decoders in terms of the error-correcting performance, also have hardware efficiencies that are close to several IEEE 802.11n LDPC decoders.

Iso-FER Comparison In Figure 2.22, we observe that a polar code with $N = 8192$ under SC decoding has a small loss of 0.5 dB with respect to the IEEE 802.11n LDPC code with $N = 1944$ at a FER of 10^{-5} for $R = \frac{1}{2}$, while the error-correcting performance for $R = \frac{5}{6}$ is very similar. Moreover, a polar code with $N = 1024$ under SCL decoding with $L = 8$ and an 8-bit CRC has practically identical performance with the aforementioned polar code with $N = 8192$ under SC decoding for both $R = \frac{1}{2}$ and $R = \frac{5}{6}$. Unfortunately, the polar code with $N = 8192$ under BP decoding cannot reach the performance of the IEEE 802.11n LDPC code, even when a maximum of $I = 40$ iterations are performed. We note that the polar codes with $N = 8192$ used for SC and BP decoding were designed for an SNR of -1 dB 3 dB for $R = \frac{1}{2}$ and $\frac{5}{6}$, respectively, while the polar codes with $N = 1024$ used for SCL decoding with $L = 8$ were designed for an SNR of 0 dB and 4 dB for $R = \frac{1}{2}$ and $\frac{5}{6}$, respectively.

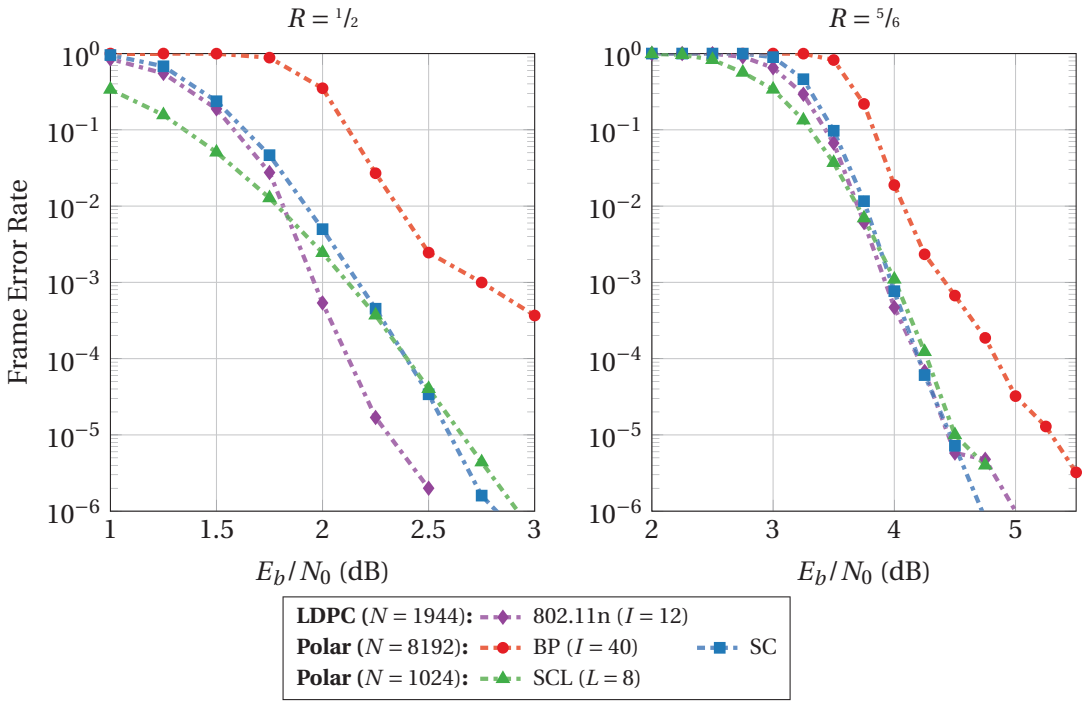


Figure 2.22 – Performance of the LDPC code of IEEE 802.11n standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding ($I = 40$), and SCL decoding (8-bit CRC).

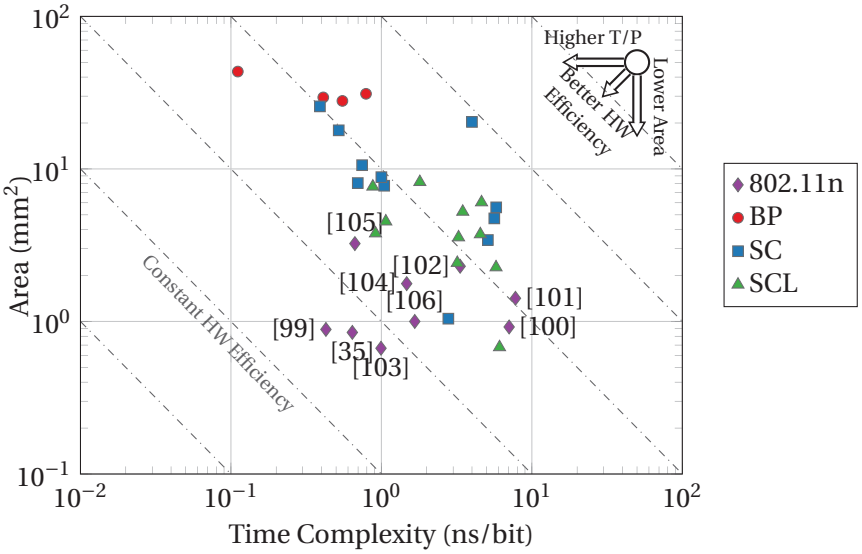


Figure 2.23 – Hardware efficiency of IEEE 802.11n LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER.

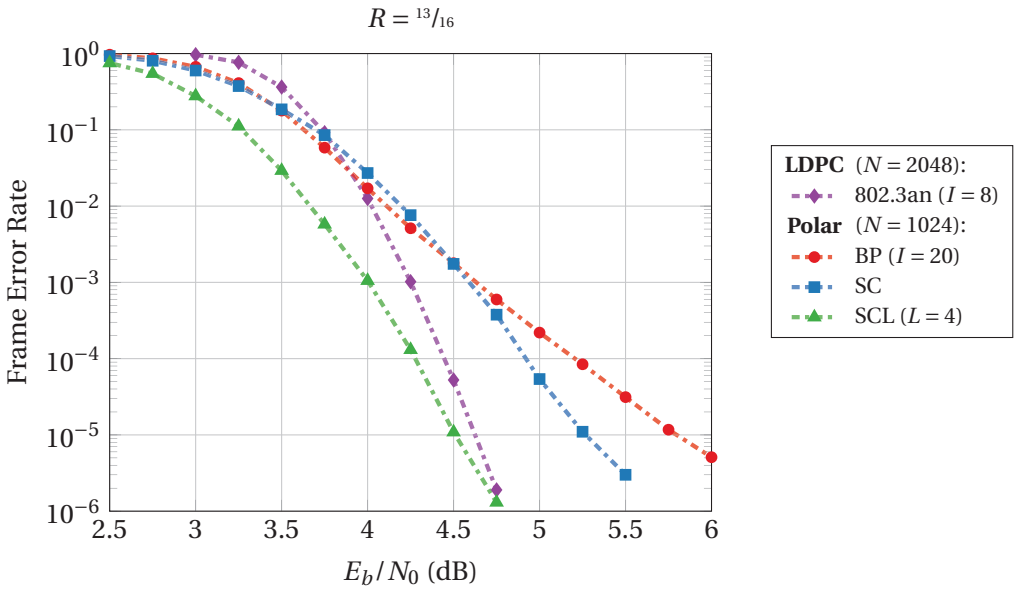


Figure 2.24 – Performance of the LDPC code of the IEEE 802.3an standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding, and SCL decoding (8-bit CRC).

In Figure 2.23, we compare the hardware efficiency of several IEEE 802.11n LDPC decoders found in the literature with the hardware efficiency of SC, BP, and SCL decoders for polar codes, which have been scaled in order to attempt to match the FER performance of the LDPC decoders. In the iso-FER case, we observe that, on average, the SCL decoders have the highest hardware efficiency out of the polar decoders. Both the SC and the BP decoders have significantly higher area requirements when trying to match the FER performance of the IEEE 802.11n LDPC codes. Finally, we observe that, on average, the IEEE 802.11n LDPC decoders have a slightly higher hardware efficiency than the polar decoders.

2.4.2.4 Polar Codes vs. IEEE 802.3an LDPC Codes

The IEEE 802.3an standard [30] uses a $(6,32)$-regular LDPC code with blocklength $N = 2048$ and code design rate $R = \frac{13}{16}$. In our simulations, the LDPC code is decoded using a flooding sum-product decoder with $I = 8$ maximum decoding iterations, which is a number that is commonly found in the literature as can be seen in Table 2.18 (we note that 4-5 layered iterations provide similar error-correcting performance to 8-10 flooding iterations).

Plain Comparison We first compare the performance of the IEEE 802.3an LDPC code with the performance of polar codes with blocklength $N = 1024$. The polar codes are decoded using BP decoding with $I = 20$ maximum iterations and a scaling factor of $\alpha = 0.9375$, SC decoding, and SCL decoding with $L = 4$ and CRC of 8 bits. The polar code for $R = \frac{13}{16}$ was designed for an SNR of 4 dB. In Figure 2.24, we observe that the polar code under SC and BP decoding has

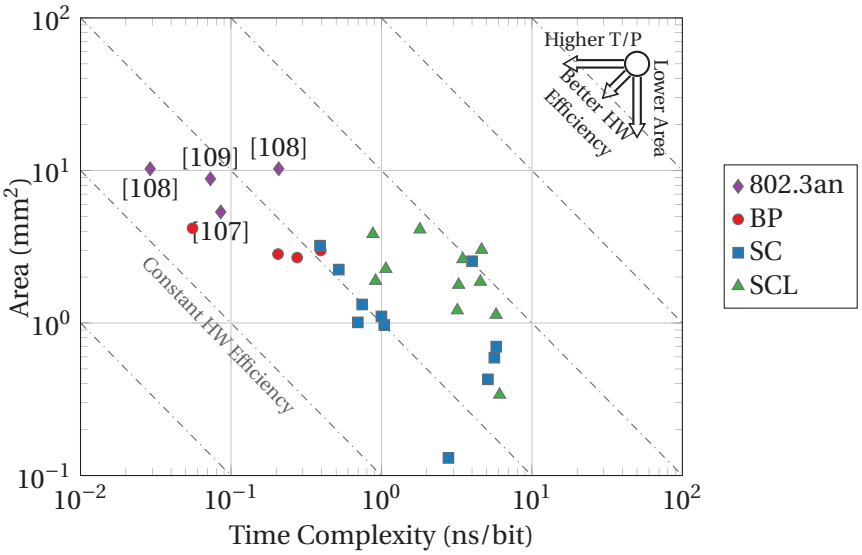


Figure 2.25 – Hardware efficiency of IEEE 802.3an LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling.

a loss of approximately 0.75 dB and 1.25 dB with respect to the LDPC code at a FER of 10^{-5}, respectively. SCL decoding, on the other hand, provides slightly superior performance than the IEEE 802.3an LDPC code for FERs down to 10^{-6}.

In Figure 2.25, we compare the hardware efficiency of several IEEE 802.3an LDPC decoders with the hardware efficiency of SC, BP, and SCL decoders for polar codes, when only considering technology scaling and not the iso-FER case. The original results for the LDPC decoders are summarized in Table 2.18. Even though the SC and BP polar decoders have lower throughput than the IEEE 802.3an LDPC decoders, they also have significantly lower area requirements, leading to similar hardware efficiency. SCL decoders, on the other hand, have lower hardware efficiency in general, mainly due to their lower throughput.

Iso-FER Comparison SCL decoding with $N = 1024$, $L = 4$, and an 8-bit CRC already performs better than the IEEE 802.3an LDPC code down to a FER of 10^{-6}. In Figure 2.26, we observe that a polar code with $N = 4096$ under SC decoding has better error-correcting performance than the IEEE 802.3an LDPC code down to a FER of 10^{-6}. BP decoding with $I = 40$ for the same polar code, however, has a small loss of 0.5 dB with respect to the IEEE 802.3an LDPC code at a FER of 10^{-5}. We note that the polar code for $N = 4096$ and $R = \frac{13}{16}$ used for SC and BP decoding was designed for an SNR of 3 dB.

In Figure 2.27, we compare the hardware efficiency of several IEEE 802.3an LDPC decoders found in the literature with the hardware efficiency of SC, BP, and SCL decoders for polar codes, which have been scaled in order to attempt to match the FER performance of the LDPC decoders. In the iso-FER case, we observe that, on average, the polar decoders have lower

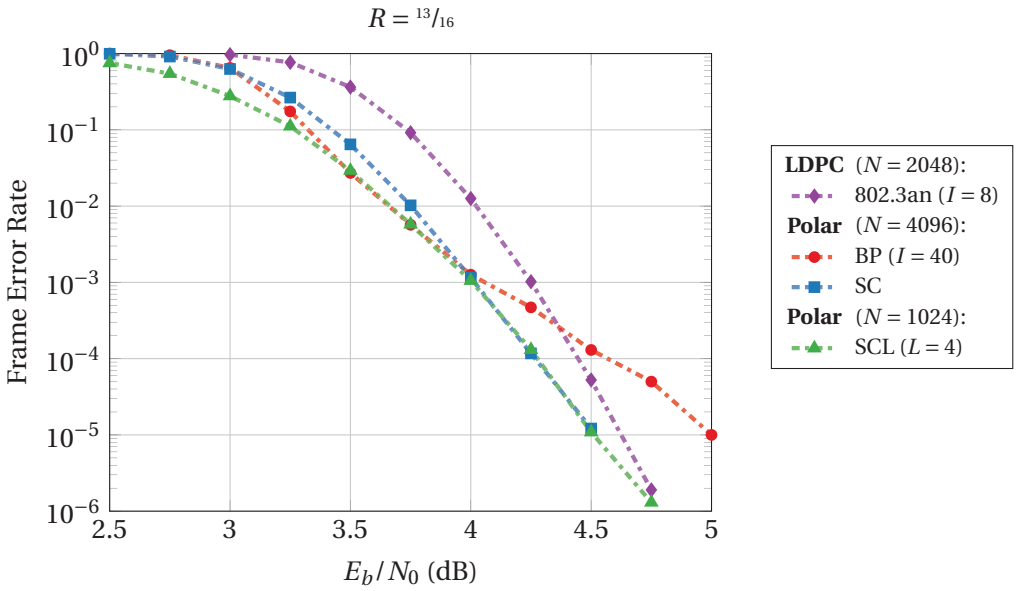


Figure 2.26 – Performance of the LDPC code of the IEEE 802.3an standard compared to polar codes with $N = 4096$ under SC and BP decoding, and $N = 1024$ under SCL decoding (8-bit CRC).

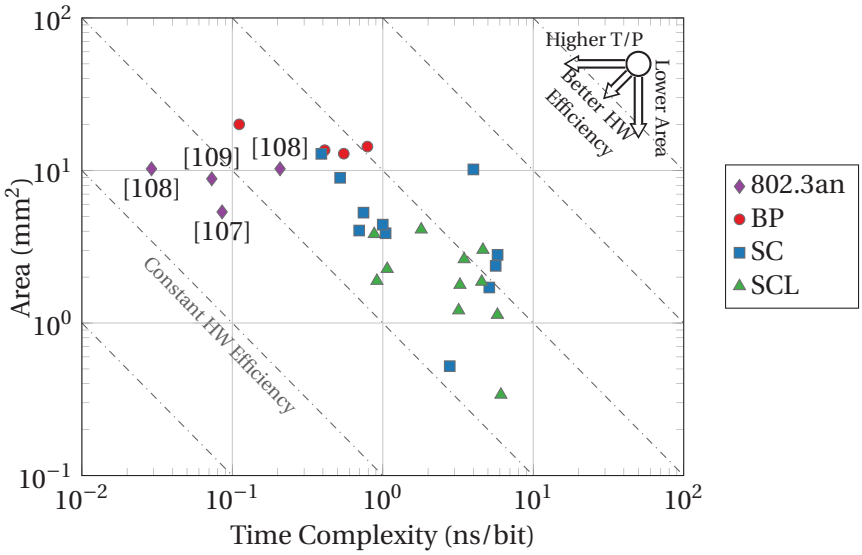


Figure 2.27 – Hardware efficiency of IEEE 802.3an LDPC decoder implementations and SC, BP, and SCL polar decoder implementations when scaling for iso-FER.

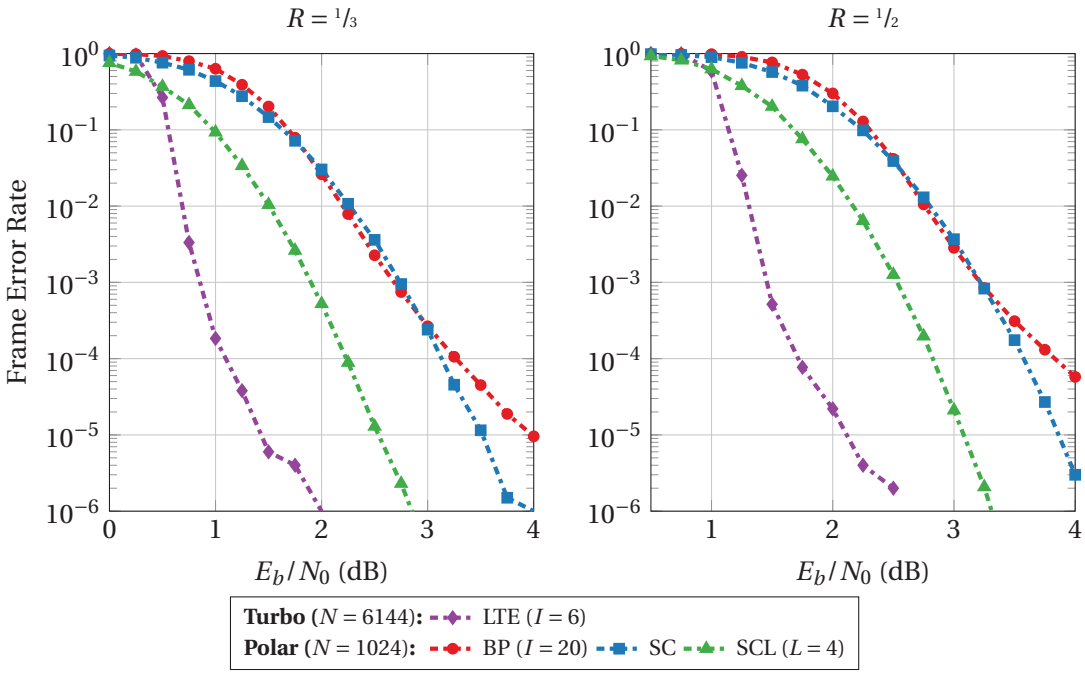


Figure 2.28 – Performance of Turbo code of LTE standard compared to polar codes with $N = 1024$ under SC decoding, BP decoding ($I = 15$), and SCL decoding ($L = 4$, 8-bit CRC).

hardware efficiency than the IEEE 802.3an LDPC decoders, while all of the polar decoders have very similar hardware efficiency. In terms of decoding throughput, only the BP decoders and a few SC decoders can approach the IEEE 802.3an LDPC decoders, albeit with slightly higher area requirements.

2.4.2.5 Polar Codes vs. 3GPP LTE Turbo Codes

The 3GPP LTE standard [59] defines a baseline Turbo code with rate $R = \frac{1}{3}$ and information bit interleaver block sizes ranging from $K = 40$ to $K = 6144$ bits. Multiple code rates are supported, both higher and lower than $R = \frac{1}{3}$, which are obtained by puncturing and parity bit repetition, respectively. We simulated the performance of this Turbo code for the largest supported interleaver length $K = 6144$ under max-log decoding with $I = 6$ iterations, which is a number that is commonly found in the hardware implementation literature, as can be seen in Table 2.19. We note that an interleaver length of $K = 6144$ leads to a codeword blocklength $N = 12288$ for rate $R = \frac{1}{2}$ and a codeword blocklength of $N = 18432$ for rate $R = \frac{1}{3}$. We provide a comparison for $R = \frac{1}{3}$ and $R = \frac{1}{2}$.

Plain Comparison We first compare the performance of the 3GPP LTE Turbo code with the performance of a polar code with blocklength $N = 1024$. The polar codes for $R = \frac{1}{3}$ and $R = \frac{1}{2}$ were designed for an SNR of -2 dB and 1 dB, respectively. The polar codes are decoded using

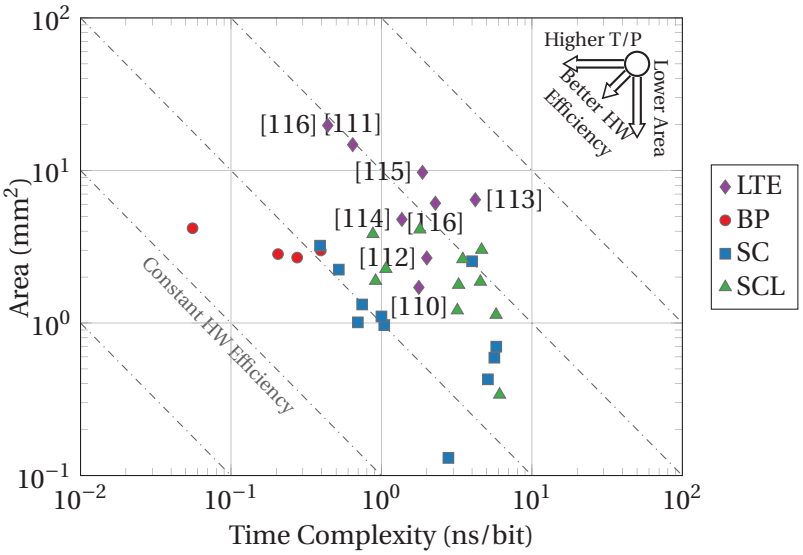


Figure 2.29 – Hardware efficiency of LTE Turbo decoder implementations and SC, BP, and SCL polar decoder implementations when only considering technology scaling.

BP decoding with $I = 20$ maximum iterations and a scaling factor of $\alpha = 0.9375$, SC decoding, and SCL decoding with $L = 4$ and CRC of 8 bits. In Figure 2.28, we observe that, for both examined rates, polar codes under SC and BP decoding have a loss of approximately 2 dB with respect to the 3GPP LTE Turbo code at a FER of 10^{-5}. Polar codes under SCL decoding, on the other hand, have a lower loss of approximately 1 dB with respect to the 3GPP LTE Turbo code at a FER of 10^{-5}.

In Figure 2.29, we compare the hardware efficiency of several 3GPP LTE Turbo decoders found in the literature with the hardware efficiency of SC, BP, and SCL decoders ($L = 4$) for polar codes, when only considering technology scaling and not the iso-FER case. The original non-scaled results for the LTE decoders are summarized in Table 2.19. We observe that most SC and BP decoders are faster and have better area efficiency than the LTE decoders. Moreover, most SCL decoders have lower decoding throughput but also lower area requirements than their LTE decoder counterparts, leading to slightly better hardware efficiency on average.

Iso-FER Comparison In Figure 2.30, we observe that a polar code with $N = 16384$ under SC decoding provides similar error-correcting performance with the LTE Turbo code with $K = 6144$ at a FER of 10^{-5} for both $R = \frac{1}{3}$ and $R = \frac{1}{2}$ and a polar code with $N = 2048$ under SCL decoding with $L = 8$ and an 8-bit CRC provides similar error-correcting performance with the LTE Turbo code with $K = 6144$ at a FER of 10^{-5} for both $R = \frac{1}{3}$ and $R = \frac{1}{2}$. We note, however, that at higher FERs the LTE Turbo code has better performance than the polar codes. The polar codes only match the performance of the LTE Turbo code at low FERs because the latter exhibits a relatively high error floor. Unfortunately, the polar code with $N = 16384$ under BP decoding cannot reach the performance of the LTE Turbo code, even when a maximum of

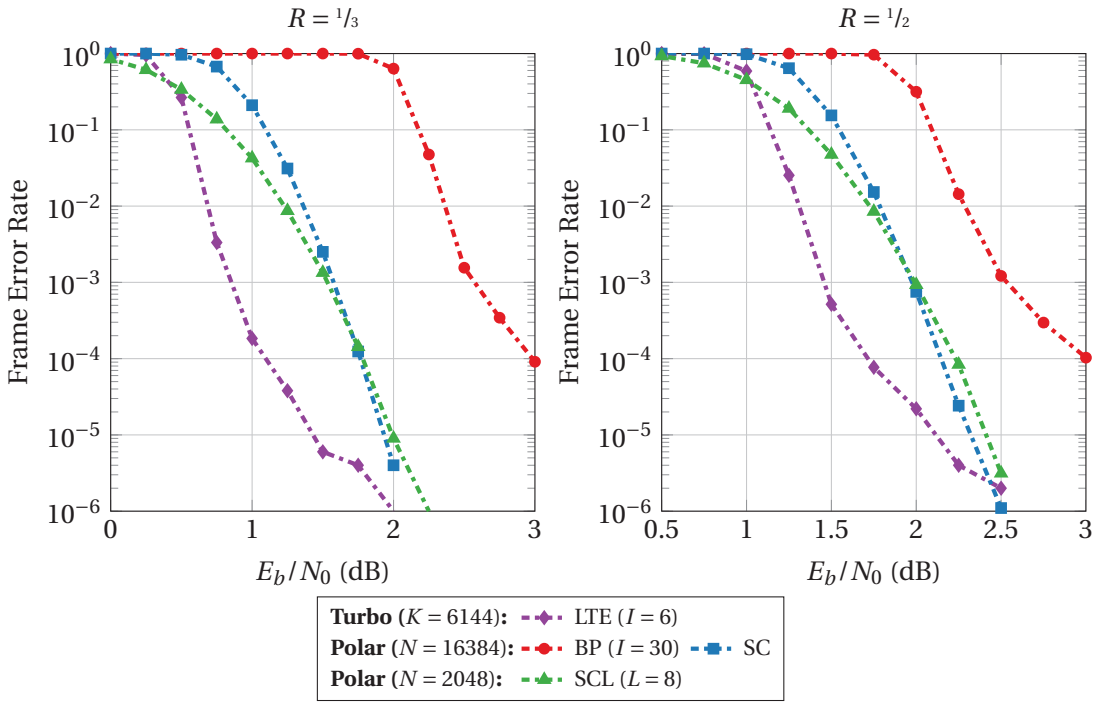


Figure 2.30 – Performance of Turbo code of LTE standard compared to polar codes with $N = 32768$ under SC decoding and BP decoding ($I = 30$), and polar codes with $N = 4096$ under SCL decoding ($L = 4$, 16-bit CRC).

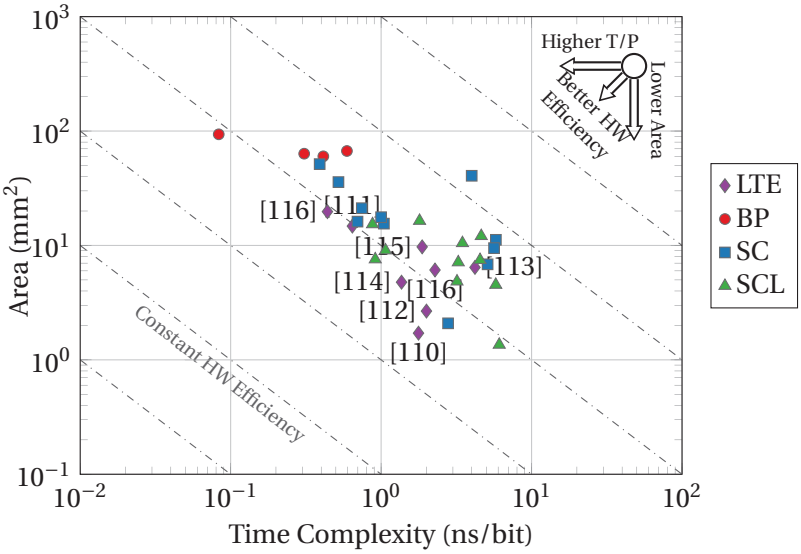


Figure 2.31 – Hardware efficiency of 3GPP LTE Turbo decoder implementations and SC, BP, and SCL polar decoder implementations which have been scaled in order to match the FER performance of the 3GPP LTE Turbo decoders.

$I = 30$ iterations are performed.

In Figure 2.30, we compare the hardware efficiency of several 3GPP LTE Turbo decoders found in the literature with the hardware efficiency of scaled SC, BP, and SCL decoders for polar codes, which have been scaled in order to attempt to match the FER performance of the 3GPP LTE Turbo decoders. In the iso-FER case, we observe that, on average, the SCL decoders have the best hardware efficiency among the polar decoders. Both the SC and BP decoders have high area requirements when trying to match the FER performance of the LTE Turbo codes. We also observe that, on average, the LTE Turbo decoders have a similar hardware efficiency to the polar decoders.

2.4.2.6 Original LDPC and LTE Decoder Results

This section contains tables with the original (unscaled) results from the papers which were used in our comparison of LDPC and Turbo decoders with polar decoders.

Table 2.16 – IEEE 802.11ad LDPC Decoder Implementations.

Work	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	Schedule	Iter.	T/P (Mb/s)	Voltage (V)	Power (mw)
Shrirani-Mehr *et al.* [87]	672	65	0.72	235	Layered	5	7900	n/a	n/a
Weiner *et al.* [88]	672	65	1.30	150	Flooding	15	3080	0.8	84
Yen *et al.* [89]	672	65	1.56	197	Layered	5	5790	1.0	361
Ajaz *et al.* [90]	672	65	1.10	215	Layered	6	6000	1.1	210
Balatsoukas-Stimming *et al.* [91]	672	40	0.18	850	Layered	5	3120	n/a	n/a
Li *et al.* [92]	672	40	0.16	500	Layered	5	5600	0.9	99
Li *et al.* [93]	672	40	0.22	500	Layered	5	5300	0.9	136
Park *et al.* [94]	672	65	1.60	540	Flooding	10	9000	1.1	783
Ajaz *et al.* [95]	672	65	0.57	400	Layered	7	9250	1.1	273
Weiner *et al.* [96]	672	28	0.63	65	Flooding	15	6000	0.7	38
Li *et al.* [97]	672	28	0.13	400	Layered	3	7070	0.8	54
Li *et al.* [98]	672	28	0.78	470	Layered	5	18400	0.9	166

Table 2.17 – IEEE 802.11n LDPC Decoder Implementations.

Work	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	Schedule	Iter.	T/P (Mb/s)	Voltage (V)	Power (mW)
Gunnam *et al.* [99]	1944	130	1.85	500	Layered	15	1618	n/a	238
Rovini *et al.* [100]	1944	65	0.48	240	Layered	12	196	1.2	168
Rovini *et al.* [101]	1944	65	0.74	240	Layered	12	178	1.2	234
Studer *et al.* [35]	1944	180	3.39	208	Layered	5	780	n/a	2886
Sun *et al.* [102]	2304	65	1.20	400	Layered	10	415	0.9	180
Jin *et al.* [103]	1944	180	2.67	250	Layered	10	503	1.8	463
Roth *et al.* [104]	1944	90	1.77	346	Layered	10	679	1.0	107
Sun *et al.* [105]	1944	45	0.81	815	Layered	15	3000	n/a	n/a
Meinerzhagen *et al.* [106]	1944	90	1.00	307	Layered	10	600	1.0	88

2.5 Summary

In Section 2.1 we have reformulated the SCL decoding algorithm for polar codes in the LL-domain, improving its numerical stability and also significantly reducing the required bit-

Table 2.18 – IEEE 802.3an LDPC Decoder Implementations.

Work	N	Tech. (nm)	Area (mm^2)	Freq. (MHz)	Schedule	Iter.	T/P (Mb/s)	Voltage (V)	Power (mW)
Cevrero *et al.* [107]	2048	90	5.35	137	Layered	4	11690	1.2	1559
Zhang *et al.* [108]	2048	65	5.35	100	Flooding	8	6670	0.7	144
Zhang *et al.* [108]	2048	65	5.35	700	Flooding	8	47700	1.2	2800
Bao *et al.* [109]	2048	130	18.40	278	Layered	5	9480	1.2	774

Table 2.19 – 3GPP LTE Turbo Decoder Implementations.

Work	K	Tech. (nm)	Area (mm^2)	Freq. (MHz)	Iter.	T/P (Mb/s)	Voltage (V)	Power (mW)
Studer *et al.* [110]	6144	130	3.57	n/a	6	391	1.2	789
Ilnseher *et al.* [111]	6144	65	7.70	450	6	2150	1.1	n/a
Chen *et al.* [112]	6144	65	1.39	512	6	692	1.2	635
Lin *et al.* [113]	6144	40	1.27	252	6	535	0.9	218
Belfanti *et al.* [114]	6144	65	2.49	410	6	1013	1.2	966
Shrestha *et al.* [115]	6144	45	2.43	600	6	1067	0.8	870
Wang *et al.* [116]	6144	90	6.10	625	8	438	1.0	272
Wang *et al.* [116]	6144	90	19.75	625	8	2274	1.0	1450

width for a fixed-point hardware implementation. Moreover, we have presented the first SCL decoder hardware implementation in the literature which uses a smart copying mechanism to avoid copying the path LLs. In Section 2.2, we have introduced an LLR-based path metric for SCL decoding of polar codes, which enables the implementation of an LLR-based SCL decoder that is even more numerically stable than its LL-based counterpart. We note that the LLR-based path metric is not specific to SCL decoding and can be applied to any other tree-search based decoder (e.g., stack SC decoding [48]). Moreover, we have shown that we can simplify the sorting task of the SCL decoder by using various simplified sorters which exploit the properties of the LLR-based path metric. We have also described an efficient hardware architecture for an LLR-based SCL decoder that significantly outperforms the existing LL-based hardware decoders both in terms of throughput and in terms of area, leading to a substantial increase in hardware efficiency of up to 137%. Finally, we have shown that adding the CRC unit to the decoder and using CA-SCLD is an easy way of increasing the hardware efficiency of our SCL decoder at a given block-error probability as the list size can be decreased. Specifically, our CA-SCLD at list size $L = 2$ has somewhat lower block-error probability *and* more than five times better hardware efficiency than our standard SCLD at list size $L = 8$.

Path metric sorting is an important aspect of SCL decoding, especially when considering polar codes with relatively short blocklength (e.g., $N \leq 256$) and large list sizes (e.g., $L \geq 16$) for use in low-latency and/or low-power and low-rate applications, as the sorting step can dominate the overall complexity of the decoder. Even though we used the properties of the LLR-based path metric to simplify various sorters in Section 2.2, it was recently shown in [78] that further simplifications are in fact possible. It remains an important open problem to fully optimize the path metric sorting step of SCL decoding. As can be seen from the comparison of Section 2.4.2, SCL decoders cannot yet match the high throughput numbers reported for SC and BP decoders. This is partly due to the fact that fast-SSC decoding [40] has not yet been fully applied to SCL decoding. Since our LLR-based SCL decoder uses L SC decoders, it seems

evident that any architectural and algorithmic improvements made to the SC decoder itself will be beneficial to the LLR-based SCL decoder as well. However, the family of fast-SSC decoders is not applicable verbatim to the LLR-based SCL decoder. This happens because, in order to keep the path metric updated, we need to calculate the LLRs even for the frozen bits. An important step in this direction was recently made in [61], but the hardware implementation of a fast-SSC based SCL decoder is an essential next step.

In Section 2.4.2 we have presented a literature survey on hardware decoders for polar codes that included BP, SC, and SCL decoders in which we outlined the most important algorithmic and architectural techniques that have been used to date. Moreover, we have compared the polar codes with LDPC and Turbo decoders for existing communications standards, such as IEEE 802.11ad [32], IEEE 802.11n [31], and IEEE 802.3an [30], and 3GPP LTE [59]. In most cases, BP and SC decoding are not powerful enough and more complex algorithms, such as SCL decoding, are needed in order to match the error-correcting performance of the LDPC and Turbo codes. Moreover, we have seen that the polar decoders that can match the error-correcting performance of LDPC and Turbo codes usually have lower hardware efficiency than their LDPC and Turbo decoder counterparts. The low hardware efficiency stems mainly from the low throughput that these decoders achieve, and not so much from their area requirements. In conclusion, while significant improvements have been achieved over the past few years in the polar decoding literature, further work is required in order to match and surpass existing channel coding solutions. In particular, the direction of increasing the throughput of SCL decoders seems promising, since SCL decoders have the lowest area requirements and generally the best hardware efficiency out of the polar decoders in all iso-FER comparisons of Section 2.4.1.4.

3 Faulty Polar and LDPC Channel Decoders

3.1 Approximate Computing

Approximate computing [117] is a computing paradigm in which the requirements for reliable and predictable operation of integrated circuits and software are relaxed. This approach is motivated by the observation that, for many applications, exact computations are not always necessary and that allowing for a small and acceptable degradation in the quality of the produced output can result in disproportionately large computation energy savings. Moreover, allowing for some faults in integrated circuits can improved their production yield significantly, since faulty dies do not necessarily have to be discarded.

An important distinction has to be made between *intentionally approximate* and *unintentionally faulty* operation since, even though these two modes of operation are fundamentally different, they are often treated similarly by researchers in the field. Unintentionally faulty operation results from factors that are difficult to control, such as radiation or manufacturing defects and unintentional side-effects of energy-saving techniques (such as aggressive voltage scaling [118]), that affect the correct operation of integrated circuits. Intentionally approximate operation, on the other hand, is controllable and it may be caused by circuit design techniques, such as using approximate adders and multipliers [119], or even algorithmic modifications that simplify the operation of a circuit on a much higher level [120]. Almost all signal processing systems are intentionally approximate on at least one level, since they mostly operate on *quantized* and often *approximated* versions of the algorithms that they implement. Intentional and unintentional factors are, of course, not mutually exclusive and they may co-exist in an approximate computing system.

In this chapter, we examine three approximate computing scenarios in the context of channel coding, which involve both intentionally approximate and unintentionally faulty operation. More specifically, in Section 3.2 we present a modified construction for polar codes that aims to reduce the complexity of SC decoding while sacrificing the error-correcting performance of the code in a highly controlled and systematic fashion. Then, in Section 3.3 we study SC decoding of polar codes when the memories that are used to store the messages involved in

the decoding process are unintentionally faulty. Finally, in Section 3.4 we provide a similar analysis for MS decoding of LDPC codes under unintentionally faulty message storage.

3.2 Successive Cancellation Decoding with Intentionally Mismatched Polar Codes

As explained in Section 1.3.3.1, the complexity of SC decoding scales like $O(N \log N)$ since the DDG of the algorithm contains $N \log N$ nodes and each node is activated exactly once. However, complexity reduction of SC decoding can be achieved by not activating the nodes in the DDG that are only connected to frozen synthetic channels, since the results of these computations are never used by the decoder. This technique, called *simplified SC decoding*, was first proposed in [121] and later improved in several works (e.g., [40]).

In all simplified SC decoders, the amount of complexity reduction that can be achieved by skipping unnecessary node computations highly depends on the distribution of frozen and information bit locations in the polar code. Arıkan's original polar code construction [6] only focuses on maximizing the reliability of the information bits. A few altered polar-like code constructions to support low-complexity decoding based on [121] have already been proposed in the literature [122, 123] and their objective is also to trade error-correction performance for decoding complexity by slightly changing the set of information bits $\mathcal{A}$, while keeping the code rate fixed. The main idea behind all the altered code constructions is to exchange the locations of a few frozen bits and information bits in order to get more bit patterns that are favorable in terms of decoding latency. The polar code construction method in [122] first defines a small set of bit locations which contains the $n_s - h$ least reliable information bit locations along with the h most reliable frozen bit locations. Then, in order to keep the rate fixed, it performs an exhaustive search over all $\binom{n_s}{h}$ possible combinations of the n_s elements containing exactly h frozen bit locations and selects the combination that leads to the smallest decoding latency. A similar greedy algorithm is presented in [123] for polar codes with more general code lengths of the form $N = l^n$, $l \geq 2$.

In this chapter, we first formalize the altered polar code construction problem as a binary integer linear program. Consequently, we show that finding the polar code with the lowest decoding complexity under an error-correction performance constraint is an NP-hard problem. For this reason, we describe a greedy approximation algorithm which provides reasonable complexity-performance trade-offs at low complexity even for polar codes with very large blocklengths.

3.2.1 Complexity-Performance Trade-Offs for SC Decoding of Polar Codes

In this section, we first introduce the metrics that are used in order to quantify the error-correcting performance and the complexity of a polar code with a given set of information indices $\mathcal{A}$. Then, we use these metrics in order to formulate an optimization problem that aims

to maximize the complexity reduction while satisfying a given error-correcting performance constraint. By varying the performance constraint, various complexity-performance trade-offs can be achieved.

3.2.1.1 Complexity and Performance Metrics

Complexity metric: We use the total number of computations that can be saved by pruning the DDG, denoted by c, as a complexity metric. As explained previously, the value of this metric depends on the set of information indices $\mathcal{A}$ and we explain how it can be computed as part of the formulation of the optimization problem.

Let the blocklength N and the rate $R = \frac{k}{N},\ k \in \{0,\dots,N\}$, be fixed. In order to simplify notation, in this section we denote the mutual information values of the N synthetic channels by $I_i,\ i = 0\dots,N-1$, meaning that

$$I_i = I\left(W_n^{(i)}\right),\ i = 0\dots,N-1. \tag{3.1}$$

Performance metric: We use the sum mutual information of the set of non-frozen channels as a performance metric, i.e.,

$$m = \sum_{i\in\mathcal{A}} I_i = N\cdot I(W) - \sum_{i\in\mathcal{A}^c} I_i. \tag{3.2}$$

Note that the polar code construction originally proposed by Arıkan [6] essentially maximizes m under the constraint $|\mathcal{A}| = k$ and let $m_{\max}$ denote this maximum, i.e.,

$$m_{\max} = \max_{\mathcal{A}:|\mathcal{A}|=k} \sum_{i\in\mathcal{A}} I_i. \tag{3.3}$$

Since $I_i \geq 0,\ 0 \leq i \leq N-1$, the maximization amounts to selecting the channel indices with the k largest I_i values.

Our choice of performance metric requires some intuitive justification. Let $Z(W)$ denote the Bhattacharyya parameter of a channel W and let $Z_i = Z(W^{(i)})$. It is known that $\sum_{i\in\mathcal{A}} Z_i$ is an upper bound on the probability of block error [6]. It was shown in [124] that, for the BEC, this upper bound is tight. Moreover, for the BEC we have $I_i = 1 - Z_i$, hence by maximizing $\sum_{i\in\mathcal{A}} I_i$ one can minimize the block-error probability. Similarly, by placing a constraint on $\sum_{i\in\mathcal{A}} I_i$, we are implicitly placing a constraint on $\sum_{i\in\mathcal{A}} Z_i$, which is directly related to the block-error probability. So, for the case of the BEC, the metric that we use has an explicit relation with the probability of block error. For more general channels, one intuitively expects that there is at least an implicit relation between the two quantities. Ideally, one would like to use the probability of block error itself as a metric, but, to the best of our knowledge, this cannot be described analytically as a function of $\mathcal{A}$, and especially not in a linear way which is necessary to enable a simple formulation of the optimization problem. Moreover, as we will show in Section 3.2.3.3, the error-correcting performance of the various altered polar codes that we

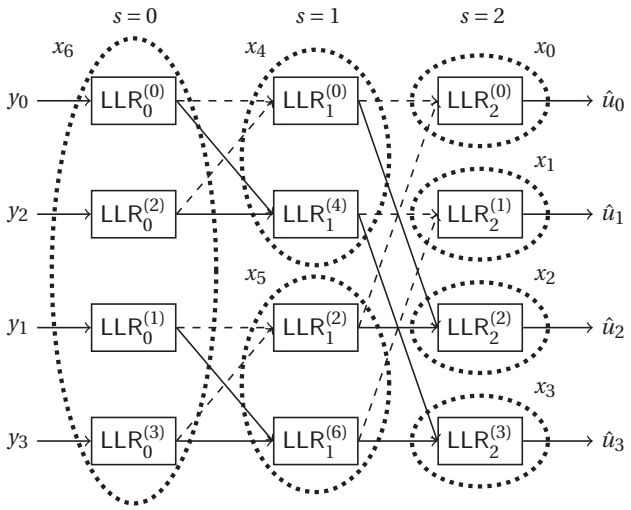


Figure 3.1 – Decoding graph for $N = 4$ with channel groups. An optimization variable x_i is associated with each group g_i. Setting $x_i = 1$ corresponds to freezing all channels in g_i.

construct degrades gracefully with increasing complexity reduction.

3.2.1.2 Optimization Problem Formulation

From a complexity perspective, it is favorable to form clusters of $2^l,\ l \in \mathbb{N}$, frozen channels in order to maximize pruning of node computations according to [121]. In this section, we describe an optimization problem which constructs a polar code of rate R, in a way that maximizes c while ensuring that m is larger than a pre-defined performance constraint $m' \geq 0$. To this end, the indices of the N channels are grouped into clusters of $1, 2, \ldots, N$, consecutive channels as illustrated in Fig. 3.1, where the illustration of the groups has been spread across the stages of the data dependency graph to reduce congestion. Let the set of all the groups be denoted by $\mathcal{G}$. We have

$$|\mathcal{G}| = N \sum_{j=0}^{n} 2^{-j} = 2N - 1. \tag{3.4}$$

We associate each of the groups $g_i \in \mathcal{G}$ with a binary optimization variable $x_i,\ i = 0, \ldots, 2N-2$. The assignment $x_i = 1$ means that all synthetic channels contained in group i are frozen. Each group also has a rate cost, denoted by $f_i,\ i = 0, \ldots, 2N-2$. This rate cost is equal to the number of channel indices that are contained in g_i, i.e., $f_i = |g_i|$, and it reflects the rate loss incurred by setting $x_i = 1$. This leads to the *rate constraint*

$$\sum_{i=0}^{2N-2} f_i x_i = N - k. \tag{3.5}$$

Observe that, if in the example of Fig. 3.1, say, $x_6 = 1$, then the rate cost f_6 is paid. However,

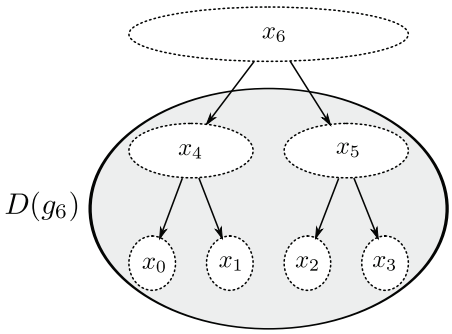


Figure 3.2 – Tree structure of channel groups with descendants of g_6, i.e., $D(g_6)$, and their corresponding optimization variables. If $x_6 = 1$, then $x_i = 0$ has to be enforced for all $x_i : g_i \in D(g_6)$.

due to the tree structure of the groups, f_6 includes the rate costs for freezing the channels in groups g_0 to g_5, So, when $x_i = 1$ for any non-leaf group, $x_i = 0$ has to be enforced for all the descendants of this group in order not to count any rate costs more than once. Let the descendants of group $g_i \in \mathcal{G}$ be denoted by $D(g_i)$. An example is illustrated in Fig. 3.2. Let $\mathcal{X} = \{(i, j) : g_i \in \mathcal{G} \backslash \{\text{leaves}\}, g_j \in D(g_i)\}$. Since $x_i \in \{0, 1\}$, the *mutual exclusiveness* constraint can be formalized as

$$x_i + x_j \leq 1, \ \forall (i, j) \in \mathcal{X}. \tag{3.6}$$

Moreover, we have

$$|\mathcal{X}| = N \sum_{i=1}^{\log N - 1} (\log N - i) 2^{-i} = 2(\log N - 1)N + 2. \tag{3.7}$$

From (3.4) and (3.7), it can be seen that the number of variables grows linearly with the code length and the number of constraints in (3.6) grows as $N \log N$. Each group $g_i \in \mathcal{G}$ has an associated gain in the number of computations, denoted by c_i, $i = 0, \ldots, 2N - 2$. This gain is the number of computations that is saved via pruning if all the channels in this group are frozen. Let $s(g_i) \in \{0, \ldots, \log N - 1\}$ denote the stage to which group $g_i \in \mathcal{G}$ corresponds. For example, in Fig. 3.1, group g_4 corresponds to stage 1. Then, we have

$$c_i = (n - s(g_i) + 1) 2^{n - s(g_i)}, \quad i = 0, \ldots, 2N - 2. \tag{3.8}$$

Due to (3.6), no complexity gain is counted more than once. Finally, freezing the channels in group $g_i \in \mathcal{G}$ results in a loss in total mutual information, denoted by m_i, with

$$m_i = \sum_{j \in g_i} I_j, \quad i = 0, \ldots, 2N - 2. \tag{3.9}$$

again, due to (3.6), no mutual information loss is counted more than once. A *performance*

constraint $m \geq m'$, $m' \geq 0$, is enforced, which can equivalently be written as

$$\sum_{i=0}^{2N-2} x_i m_i \leq N \cdot I(W) - m'. \tag{3.10}$$

An optimization problem which maximizes the complexity gain, while ensuring that the resulting code has rate R and satisfies the performance constraint, can be formulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=0}^{2N-2} c_i x_i \\ \text{subject to} \quad & \sum_{i=0}^{2N-2} f_i x_i = N - k \\ & \sum_{i=0}^{2N-2} x_i m_i \leq N \cdot I(W) - m' \\ & x_i + x_j \leq 1, \ \forall (i,j) \in \mathcal{X} \\ & x_i \in \{0,1\}, \quad i = 0, \ldots, 2N-2 \end{aligned} \tag{3.11}$$

The above problem is an instance of the multidimensional 0–1 knapsack problem [125], which is known to be NP-hard in general. If m' is chosen carefully so that $m' \leq m_{\max}$, then (3.11) is always feasible. Moreover, for $m' = m_{\max}$, the optimization problem reduces to the construction proposed by Arıkan, while $m' = 0$ results in a construction that maximizes the number of saved computations while completely disregarding performance. By varying m' between these two extremal values, various complexity-performance trade-offs can be achieved.

3.2.2 Greedy Optimization Algorithm

In order to solve (3.11) for practically relevant blocklengths, like $2^{10} \leq N \leq 2^{20}$, in reasonable time, we present a greedy algorithm that takes advantage of the structure of the problem to provide useful solutions with negligible runtime.

3.2.2.1 Greedy Algorithm Description

Our greedy algorithm consists of three steps, namely the *greedy maximization* step, the *feasibility* step, and the *post-processing* step. In the first step, the goal is to greedily maximize the objective function while satisfying all inequality constraints. The second step ensures that the equality constraint is also satisfied, while the last step finalizes and improves the solution. Recall that $k' = N - k$ is the number of bits that need to be frozen. Let k'_{bin} denote the $\log N + 1$ bit left-MSB binary representation of k' and let $k'_{\text{bin}}(j)$, $0 \leq j \leq \log N - 1$, denote the j-th bit of k'_{bin}. The greedy maximization step is inspired by the following observation.

Proposition 1. *If there were no performance constraint present in (3.11), the problem could be solved exactly as follows.*

1. *Set* $j = 0$ *and* $x_i = 0,\ 0 \le i \le 2N-2$.
2. *If* $k'_{bin}(j) = 1$, *then set* $x_i = 1$ *for one* $g_i : s(g_i) = j$, *denoted by* $g_{i'}$, *and set* $x_i = 0$ *for all remaining* $g_i : s(g_i) = j$. *Remove all* $x_i : g_i \in D(g_{i'})$ *from the problem.*
3. *Set* $j = j+1$ *and go to 2. until* $j > \log N$.

Proof. By eliminating all $x_i : g_i \in D(g_{i'})$ from the problem at step 2, we guarantee that the mutual exclusiveness constraint is not violated. If $k'_0(j) = 1$ then the k' required bits are frozen in the first iteration of the above loop and the algorithm can safely terminate. Moreover, stage 1 contains two groups, of which only one can be frozen, and for each group in stage j there are two groups in stage $j+1$, so that step 2 can always be executed. We now show that any optimal solution must freeze at most one group per stage. Suppose that, for some solution, more than one groups were frozen in some stage j. Then, it is possible to replace any two frozen groups at stage j with some frozen group at stage $j+1$ without violating any constraint. Based on (3.8), for the complexity gains we have

$$2 \cdot \left((n-j+1)2^{n-j} \right) = (n-j+1)2^{n-j+1} < (n-j+2)2^{n-j+1},\ \forall\, j \ge 0, \tag{3.12}$$

so this would strictly increase the objective function, meaning that the original solution could not have been optimal. Since all groups in stage j contain $2^{(n-j)}$ bits and the binary representation of k' is unique, it follows that the only way to freeze exactly k' channels by freezing at most one group per stage, thus satisfying the rate constraint, is to freeze the groups according to the pattern dictated by k'_{bin}. □

3.2.2.2 Greedy maximization step

The greedy maximization step is different than the procedure of Proposition 1 in that it ensures that the performance constraint is satisfied. In the following procedure, k'_{bin} is again initialized to $\log N$ bit right-MSB binary representation of k', but $k'_{\text{bin}}(j) \in \mathbb{N}$.

1. Set $j = 0$ and $x_i = 0,\ 0 \le i \le 2N-2$.
2. If $k'_{\text{bin}}(j) \ge 1$, then try the following.
   - 2.1. Find the $g_i : s(g_i) = j$ with the smallest m_i in stage j and set $x_i = 1$.
   - 2.2. If $\sum_i x_i m_i \le N \cdot I(W) - m'$, then remove all $x_i : g_i \in D(g_{i'})$ from the problem, set $k'_{\text{bin}}(j) = k'_{\text{bin}}(j) - 1$, and go to 2.
   - 2.3. Else, set $k'_{\text{bin}}(j+1) = k'_{\text{bin}}(j+1) + 2$, set $x_i = 0$, and go to 3.
3. Set $j = j+1$ and go to 2. until $j > \log N$.

At step 2.3., we set $k'_{\text{bin}}(j+1) = k'_{\text{bin}}(j+1) + 2$ because for each group that could not be frozen at stage j due to the performance constraint, we need to freeze two groups at stage $(j+1)$ in

order to (hopefully) satisfy the rate constraint. Unfortunately, there is no longer a guarantee that the procedure will be able to freeze exactly k' bits as required to satisfy the rate constraint. However, the mutual exclusiveness and performance constraints are guaranteed to be met.

3.2.2.3 Feasibility step

The second step of the algorithm sacrifices the objective function in a systematic step-by-step fashion until the solution is feasible, i.e., until the rate constraint is satisfied. Let k'' denote the number of additional bits that need to be frozen after the greedy maximization step is finished so that the rate constraint is satisfied, i.e., $k'' = k' - \sum_i f_i x_i$.

If $k'' > 0$, then the feasibility step starts greedily unfreezing frozen groups to free up mutual information. More and more groups are unfrozen until the total number of unfrozen groups that can be frozen at stage n is equal to k'' plus the number of variables in the groups that were unfrozen so far. Since during this step only groups at stage n are refrozen which provide the smallest complexity gain, no direct effort is made to minimize the loss in the objective function. The feasibility step starts at stage $\lceil \log k'' \rceil + 1$, because by unfreezing a group in this stage it is possible to satisfy the rate constraint in a single step, thus making an indirect effort to minimize the objective function loss. Subsequently, all stages up to n are visited, and the procedure continues with stages 0 to $\lceil \log k'' \rceil$, thus visiting all stages, if required. If $m' \leq m_{\max}$, the feasibility step is guaranteed to find a feasible solution.

3.2.2.4 Post-processing step

The post-processing step identifies pairs of consecutive frozen groups at each stage j and replaces them with their parent group at stage $j-1$, which strictly improves the objective function as we saw in (3.12) without violating any of the constraints.

3.2.3 Numerical Results

3.2.3.1 Exact Reference Solution for Short Polar Codes

Even though (3.11) is NP-hard, relatively small instances can still be solved by using standard branch-and-bound methods. For simplicity in calculating the mutual information values $I_i,\ i = 0,\ldots,N-1$, we present results only for the BEC(p), where p denotes the erasure probability. However, the proposed approach can be used for any other channel and input distribution, provided that $I_i,\ i = 0,\ldots,N-1$, are available. Moreover, given $I_i,\ i = 0,\ldots,N-1$, the complexity of (3.11) and of the greedy algorithm presented in Section 3.2.2 does not depend on the type of channel. We assume that the capacity achieving input distribution is used, so that $I(W) = 1-p$.

The solutions obtained by solving (3.11) for various $0 \leq m' \leq m_{\max}$ exactly as well as by using

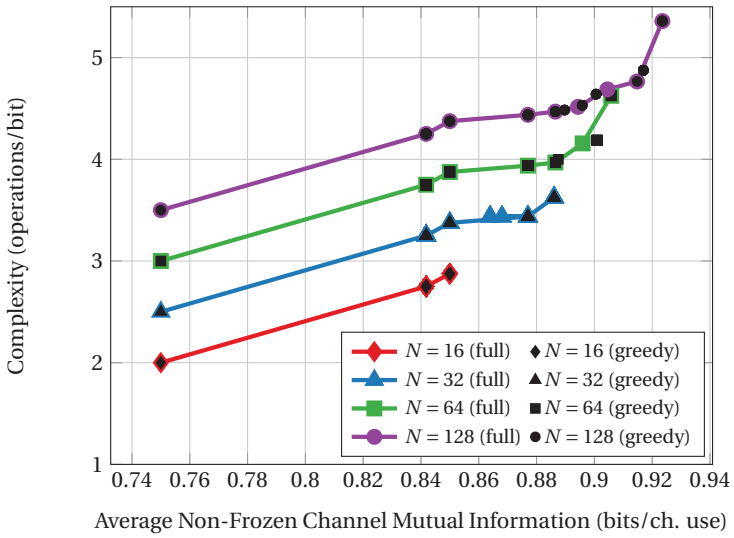


Figure 3.3 – Results from exact solution of (3.11) and of the greedy algorithm for $R = 0.5$, $N = 2^n$, $n = 4,5,6,7$, and transmission over a BEC(0.5).

the greedy algorithm for various constraints and blocklength up to $N = 2^7$ and for $R = 0.50$ are compared in Fig. 3.3. We use the complexity in operations per bit on the vertical axis and the average mutual information on the horizontal axis. The former can be easily obtained from any solution x^* as $\frac{1}{N}\left(N\log N - \sum_{i=0}^{2N-2} c_i x_i^*\right)$, while the latter is equal to $1 + \frac{1}{RN}\sum_{i=0}^{2N-2} m_i x_i^*$. We observe that the greedy algorithm is able to find most of the optimal solutions for small instances of the problem.

3.2.3.2 Greedy Algorithm for Long Polar Codes

The solutions found by the greedy algorithm are presented in Fig. 3.4 for various blocklengths and for $R = 0.50$. For $N = 2^{20}$ the average running time of the greedy algorithm is less than 10^2 seconds on an Intel Core i7 870 processor running at 2.93 GHz, which is negligible given that the optimization is carried out offline. We observe that the rightmost part of the curve is relatively steep, thus providing favorable trade-offs. For a fixed blocklength, the codes corresponding to some solution points can be chosen and stored in order to provide the system with online performance-complexity trade-offs. Moreover, during the design phase one can choose the solution with the best performance among all blocklengths that satisfies a given complexity constraint.

3.2.3.3 Error-Correcting Performance Degradation

In principle, it is possible that a solution of (3.11) contains a very bad channel in $\mathcal{A}$. This would lead to a catastrophic failure of the code, resulting in a block error rate (BLER) close to 1. This

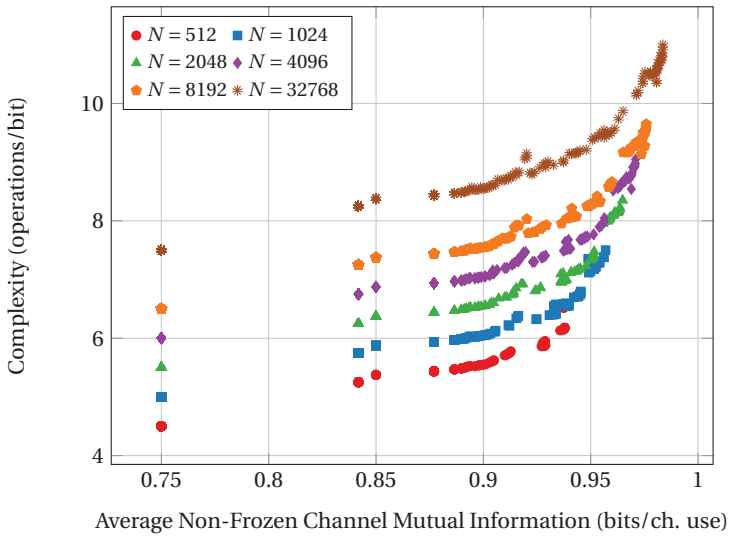


Figure 3.4 – Solutions of greedy algorithm for $R = 0.5$, $N = 2^n$, $n = 9, 10, 11, 12, 13, 15$, over a BEC(0.5).

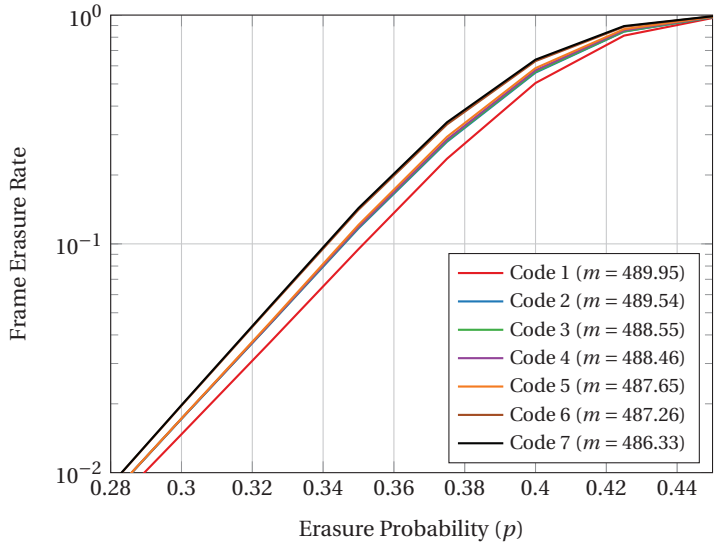


Figure 3.5 – Frame erasure rate performance and performance metric of the *useful* codes for $R = 0.5$ and $N = 2^{10}$.

problem can be circumvented by adding the following additional constraints to (3.11)

$$(1-x_i)\cdot h_i = 0,\ i = 1,\ldots,2N-1, \tag{3.13}$$

where $h_i = 1$ if $g_i \in \mathcal{G}$ contains a channel with $I_i \leq m''$, where m'' is chosen as the lowest acceptable mutual information of the channels used for the information bits, and $h_i = 0$ otherwise. However, we have observed in simulations that the useful codes (a code is said to be *useful* if it lies on the Pareto frontier of the set of obtained solutions) have a performance which degrades gracefully with decreasing values of the performance metric. An example of this behavior for $N = 2^{10}$ can be seen in Fig. 3.5, where code 1 corresponds to the standard construction of [6], while codes 2 to 8 provide different performance-complexity trade-offs.

3.3 Successive Cancellation Decoding of Polar Codes with Faulty Memories

Uncertainties in the manufacturing process of integrated circuits are expected to play a significant role in the design of very-large-scale integration systems in the nanoscale era [126, 127, 128]. Due to these uncertainties, it will become increasingly difficult to guarantee the correct behavior of integrated circuits at the gate level, meaning that the hardware may become *faulty* in the sense that data is not always processed or stored correctly. Moreover, aggressive voltage scaling, which is commonly used to reduce the energy consumption of integrated circuits, can increase the occurrence of undesired faulty behavior [129]. Traditional methods to ensure accurate hardware behavior, such as using larger transistors or circuit-level error correcting codes, are costly both in terms of area and power.

In this section we study successive cancellation decoding of polar codes for transmission over the BEC under an erasure-based fault model for the internal storage elements in the decoder hardware. Contrary to the previous section, in this section the faulty behavior is *unintentional*. We show that, under this fault model, fully reliable communication is no longer possible. Furthermore, by studying the polarization process, we show that synthetic channel ordering with respect to both the channel erasure probability and the internal decoder erasure probability still holds. We also adapt the lower bound on the FER derived in [124] to the case of such faulty decoding, and we use it in order to derive the FER-optimal blocklength for a polar code of a given rate, and for a given channel and decoder erasure (i.e., fault) probability. Finally, we introduce a simple unequal error protection method, which is shown to re-enable asymptotically fully reliable communication by protecting only a constant fraction of the decoder. In the finite blocklength regime, our proposed fault-tolerance method significantly improves the FER performance with very low hardware protection overhead.

3.3.1 Faulty Successive Cancellation Decoding of Polar Codes for the BEC

Successive cancellation decoding of polar codes can be greatly simplified for the case of the BEC as follows. Without loss of generality, we assume the output alphabet of the BEC W to be $\mathcal{Y} = \{-1, 0, +1\}$, where 0 denotes an erasure, while -1 corresponds to the binary input 1 and $+1$ corresponds to the binary input 0. For transmission over the BEC, the update functions f_+ and f_- can be re-defined as

$$f_-(a,b) = ab, \tag{3.14}$$

$$f_+(a,b,u) = \left\lfloor \frac{(-1)^u a + b}{2} \right\rceil, \tag{3.15}$$

where u denotes a *partial sum*, which is the modulo-2 sum of some of the previously decoded bits, $\lfloor \cdot \rceil$ denotes the rounding operation, and we use $\lfloor -0.5 \rceil = -1$ and $\lfloor 0.5 \rceil = 1$ for tie-breaking. When level n is reached, the output message will either be correct (i.e., -1 or $+1$), or an erasure. If the final output message is correct, we can derive the corresponding bit value for $\hat{\mathbf{u}}_i$ and proceed with decoding. If the final output message is an erasure, the decoder halts and declares a block erasure. We note that in the latter case the decoder could make a random decision and attempt to continue decoding.

3.3.1.1 Erasure Probability of Synthetic Channels

Let $Z_{s,k}^{(\mathbf{s})} \triangleq Z\left(W_{s,k}^{(\mathbf{s})}\right)$ denote the Bhattacharyya parameter of the synthetic channel $W_{s,k}^{(\mathbf{s})}$. When W is a BEC(p), its Bhattacharyya parameter is equal to the erasure probability, i.e., $Z\left(W_{0,k}^{(\emptyset)}\right) = Z(W) = p$. Moreover, all synthetic channels generated at step s are also BECs and their Bhattacharyya parameters (equivalently, their erasure probabilities) can be calculated recursively based on the Bhattacharyya parameters of the channels at step $(s-1)$ as [6]

$$Z_{s,k}^{(\mathbf{s}-)} = Z_{s-1,k}^{(\mathbf{s})} + Z_{s-1,k+2^{n-s}}^{(\mathbf{s})} - Z_{s-1,k}^{(\mathbf{s})} Z_{s-1,k+2^{n-s}}^{(\mathbf{s})}, \tag{3.16}$$

$$Z_{s,k}^{(\mathbf{s}+)} = Z_{s-1,k}^{(\mathbf{s})} Z_{s-1,k+2^{n-s}}^{(\mathbf{s})}, \tag{3.17}$$

where $s = 1, \ldots, n,\ k = 0, \ldots, 2^{n-s} - 1$. The channels $W_{s,k}^{(\mathbf{s})},\ k = 0, \ldots, 2^{n-s} - 1$, are independent copies of the same type of channel, meaning that their erasure probabilities are identical. Thus, if we are only interested in the erasure probability of a specific *type* $\mathbf{s}$ of channel we can simplify (3.16) and (3.17) by omitting the index k as

$$Z_s^{(\mathbf{s}-)} = T^-\left(Z_{s-1}^{(\mathbf{s})}\right) \triangleq 2Z_{s-1}^{(\mathbf{s})} - \left(Z_{s-1}^{(\mathbf{s})}\right)^2, \tag{3.18}$$

$$Z_s^{(\mathbf{s}+)} = T^+\left(Z_{s-1}^{(\mathbf{s})}\right) \triangleq \left(Z_{s-1}^{(\mathbf{s})}\right)^2, \tag{3.19}$$

with $Z_0^{(\emptyset)} = p$. The vector containing all $Z_s^{(\mathbf{s})}$, $\mathbf{s} \in \{+,-\}^s$, variables is denoted by $\mathbf{Z}_s$.

Moreover, as in [6, 130], we define the polarization random process ϵ_n as

$$\epsilon_s = Z_s^{(\mathbf{s})}, \tag{3.20}$$

with $\mathbb{P}[\mathbf{S} = \mathbf{s}] = \frac{1}{2^s}$, i.e., ϵ_s is equally likely to be equal to the erasure probability of any of the 2^s distinct types of synthetic channels at step s of the polarizing transformation. The random process ϵ_s can be written equivalently as

$$\epsilon_s = \begin{cases} T^-(\epsilon_{s-1}) & \text{w.p. } 1/2, \\ T^+(\epsilon_{s-1}) & \text{w.p. } 1/2, \end{cases} \tag{3.21}$$

with $\epsilon_0 = Z(W) = p$. It was shown in [6] that ϵ_s converges almost surely to a random variable $\epsilon_\infty \in \{0, 1\}$, with $P(\epsilon_\infty = 0) = I(W) = 1 - p$, where $I(W)$ denotes the symmetric capacity of the BEC W.

Finally, let us define a binary erasure indicator variable $E_{s,k}^{(\mathbf{s})}$ for which $E_{s,k}^{(\mathbf{s})} = 1$ if and only if the output of the synthetic channel $W_{s,k}^{(\mathbf{s})}$ is an erasure and $E_{s,k}^{(\mathbf{s})} = 0$ otherwise. It is clear that $\mathbb{E}\left[E_{s,k}^{(\mathbf{s})}\right] = Z_{s,k}^{(\mathbf{s})}$. The indicator variables can also be determined recursively as follows [124]

$$E_{s,k}^{(\mathbf{s}-)} = E_{s-1,k}^{(\mathbf{s})} + E_{s-1,k+2^{n-s}}^{(\mathbf{s})} - E_{s-1,k}^{(\mathbf{s})} E_{s-1,k+2^{n-s}}^{(\mathbf{s})}, \tag{3.22}$$

$$E_{s,k}^{(\mathbf{s}+)} = E_{s-1,k}^{(\mathbf{s})} E_{s-1,k+2^{n-s}}^{(\mathbf{s})}. \tag{3.23}$$

Similarly to the Bhattacharyya parameters, if we are only interested in the statistics of the indicator variable for a channel of a specific type $\mathbf{s}$, we can simplify (3.22) and (3.23) as

$$E_s^{(\mathbf{s}-)} = {E_{s-1}^{(\mathbf{s})}}' + {E_{s-1}^{(\mathbf{s})}}'' - {E_{s-1}^{(\mathbf{s})}}' {E_{s-1}^{(\mathbf{s})}}'', \tag{3.24}$$

$$E_s^{(\mathbf{s}+)} = {E_{s-1}^{(\mathbf{s})}}' {E_{s-1}^{(\mathbf{s})}}'', \tag{3.25}$$

where ${E_{s-1}^{(\mathbf{s})}}'$ and ${E_{s-1}^{(\mathbf{s})}}''$ denote two independent realizations of $E_{s-1}^{(\mathbf{s})}$ [124]. The vector containing all $E_s^{(\mathbf{s})}$ indicator variables is denoted by $\mathbf{E}_s$.

3.3.1.2 Faulty SC Decoding of Polar Codes for the BEC

All current SC decoder hardware implementations (e.g., [20, 42, 131, 38]) require a full binary tree of *memory elements* (MEs) of depth n, which store the messages that result from the update rules at each level of the decoder tree. The total number of MEs required by such decoders is

$$N_{\text{ME}} = \sum_{s=0}^{n} 2^{n-s} = 2^{n+1} - 1 = 2N - 1 \in O(N). \tag{3.26}$$

The processing elements (PEs), which apply the update rules, can also have a full binary tree structure for a fully-parallel implementation [20], although semi-parallel implementations are also possible [38]. A fully-parallel implementation requires $N - 1$ PEs, while in a semi-parallel

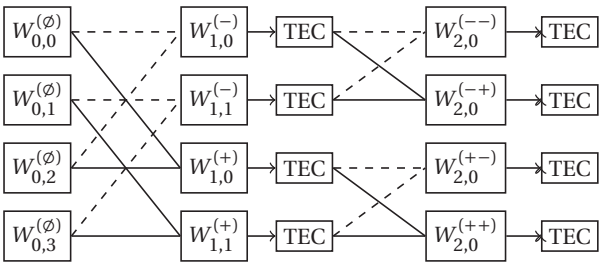


Figure 3.6 – Synthetic channel construction for a polar code of length $N = 2^2 = 4$ under faulty SC decoding. Solid lines represent the + transformation and dashed lines represent the − transformation.

implementation the number of PEs is restricted to $P < N - 1$.

We model faulty decoding as additional *internal* erasures within the memory elements of the decoder that store the messages between the decoding stages, which may be caused either by faulty PEs or by faulty MEs (or both) and we assume, without loss of generality, that they manifest themselves when an output message is written to an ME. Moreover, we assume that these erasures are *transient* in the sense that whenever an ME is written to, the internal erasures occur independently of any previous internal erasures. The partial sums, which are required by the f^+ update rule, also need to be stored in a memory, which however is typically smaller than the memory required to store the messages. Moreover, due to the partial sum recursive update rules [6], a single erasure in a partial sum will result in erasures in all following partial sums and we can intuitively see that the sensitivity of the SC decoder with respect to faults in the partial sum memory is high. Thus, in this work we assume that the partial sum memory is fault-free.

Under the above assumptions, the internal erasures can occur at the output of all synthetic channels of a polar code of blocklength n, i.e., $W_{s,k}^{(\mathbf{s})},\ s = 1,\ldots,n,\ \mathbf{s} \in \{+,-\}^s,\ k = 0,\ldots,2^{n-s}-1$. Moreover, the internal erasures occur independently of the message value and with probability δ. Let us define a ternary-input erasure channel (TEC) with input alphabet $\mathcal{X} = \{-1, 0, +1\}$ and output alphabet $\mathcal{Y} = \mathcal{X}$ and the following transition probabilities

$$\mathbb{P}[0|0] = 1, \tag{3.27}$$

$$\mathbb{P}[0|-1] = \mathbb{P}[0|+1] = \delta, \tag{3.28}$$

$$\mathbb{P}[+1|+1] = \mathbb{P}[-1|-1] = 1-\delta, \tag{3.29}$$

where the probabilities of all remaining transitions are equal to zero.

Using the above TEC, our error model can be represented as a cascade of a BEC with a TEC, as shown in Figure 3.6, where $W_{s,k}^{(\mathbf{s})}$ results from the non-faulty polarizing channel transformation applied to a pair of channels $W_{s-1,k}^{(\mathbf{t})}$ and $W_{s-1,k+2^{n-s}}^{(\mathbf{t})}$ (where $\mathbf{t}$ is a prefix of $\mathbf{s}$) and "TEC" represents the internal erasures caused by the faulty SC decoder. We denote this cascaded compound channel by $W_{s,k,\delta}^{(\mathbf{s})}$ in order to make the dependence on δ explicit. It is easy to check

that for $\delta = 0$ we get a non-faulty decoder, while for $\delta = 1$ all messages are always erasures leading to a fully faulty decoder. Thus, it is mainly interesting to study the decoder for $\delta \in (0,1)$.

In order to have a more rigorous definition of the internal erasure fault model, let us define the binary erasure indicator variable $\Delta_{s,k}^{(\mathbf{s})}$, where $\Delta_{s,k}^{(\mathbf{s})} = 1$ iff the TEC that comes after $W_{s,k}^{(\mathbf{s})}$ in Figure 3.6 causes an internal erasure at channel $W_{s,k}^{(\mathbf{s})}$, and $\Delta_{s,k}^{(\mathbf{s})} = 0$ otherwise. By definition, we have $\mathbb{P}\left[\Delta_{s,k}^{(\mathbf{s})} = 1\right] = \delta$, thus $\mathbb{E}\left[\Delta_{s,k}^{(\mathbf{s})}\right] = \delta$ and $\mathrm{var}\left[\Delta_{s,k}^{(\mathbf{s})}\right] = \delta(1-\delta)$. Since the internal erasures are assumed to be transient, all $\Delta_{s,k}^{(\mathbf{s})}$ are independent. Due to the cascaded BEC-TEC structure, we can rewrite (3.22) and (3.23) using $\Delta_{s,k}^{(\mathbf{s})}$ as

$$E_{s,k,\delta}^{(\mathbf{s}-)} = E_{s-1,k,\delta}^{(\mathbf{s})} + E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} - E_{s-1,k,\delta}^{(\mathbf{s})} E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} + \left(\overline{E_{s-1,k,\delta}^{(\mathbf{s})} + E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} - E_{s-1,k,\delta}^{(\mathbf{s})} E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})}}\right) \Delta_{s,k}^{(\mathbf{s}-)}, \tag{3.30}$$

$$E_{s,k,\delta}^{(\mathbf{s}+)} = E_{s-1,k,\delta}^{(\mathbf{s})} E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} + \left(\overline{E_{s-1,k,\delta}^{(\mathbf{s})} E_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})}}\right) \Delta_{s,k}^{(\mathbf{s}+)}. \tag{3.31}$$

Again, if we are only interested in the statistics of the indicator variable for a channel of a specific type $\mathbf{s}$, we can simplify (3.30) and (3.31) as

$$E_{s,\delta}^{(\mathbf{s}-)} = {E_{s-1,\delta}^{(\mathbf{s})}}' + {E_{s-1,\delta}^{(\mathbf{s})}}'' - {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}'' + \left(\overline{{E_{s-1,\delta}^{(\mathbf{s})}}' + {E_{s-1,\delta}^{(\mathbf{s})}}'' - {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}''}\right) \Delta_{s}^{(\mathbf{s}-)}, \tag{3.32}$$

$$E_{s,\delta}^{(\mathbf{s}+)} = {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}'' + \left(\overline{{E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}''}\right) \Delta_{s}^{(\mathbf{s}+)}. \tag{3.33}$$

where ${E_{s-1,\delta}^{(\mathbf{s})}}'$ and ${E_{s-1,\delta}^{(\mathbf{s})}}''$ denote two independent realizations of $E_{s-1,\delta}^{(\mathbf{s})}$ and $\Delta_{s}^{(\mathbf{s}-)}$ and $\Delta_{s}^{(\mathbf{s}+)}$ denote a realization of $\Delta_{s,k}^{(\mathbf{s}-)}$ and $\Delta_{s,k}^{(\mathbf{s}+)}$, respectively. The vector containing all $E_{s,\delta}^{(\mathbf{s})}$ indicator variables is denoted by $\mathbf{E}_{s,\delta}$.

We note that in a fully-parallel implementation, each ME has a corresponding PE, and our erasure-based fault model can take erasures in both the MEs and the PEs into account simultaneously. In a semi-parallel implementation, on the other hand, the MEs are significantly more than the PEs (i.e., typically $P \ll 2N-1$, as in [38] where $N = 1024$ and $P = 64$), so it is reasonable to assume that faults stem only from the MEs, as the PEs can be made reliable with circuit-level protection techniques at a relatively low cost.

3.3.2 Erasure Probability of Synthetic Channels Under Faulty SC Decoding

Using the fault model introduced in the previous section, we can rewrite the recursive expressions for $Z_{s,k}^{(\mathbf{s})}$ (i.e., (3.16) and (3.17)) in order to obtain a recursive expression for the erasure probability of the synthetic channels in the faulty case, which we denote by $Z_{s,k,\delta}^{(\mathbf{s})} \triangleq \mathbb{E}\left[E_{s,k,\delta}^{(\mathbf{s})}\right]$.

Specifically, we have

$$Z_{s,k,\delta}^{(\mathbf{s}-)} = Z_{s-1,k,\delta}^{(\mathbf{s})} + Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} - Z_{s-1,k}^{(\mathbf{s})} Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})},$$
$$+ \left(\overline{Z_{s-1,k,\delta}^{(\mathbf{s})} + Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} - Z_{s,k}^{(\mathbf{s})} Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})}}\right)\delta \tag{3.34}$$

$$Z_{s,k,\delta}^{(\mathbf{s}+)} = Z_{s-1,k,\delta}^{(\mathbf{s})} Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})} + \left(\overline{Z_{s-1,k,\delta}^{(\mathbf{s})} Z_{s-1,k+2^{n-s},\delta}^{(\mathbf{s})}}\right)\delta, \tag{3.35}$$

with $Z_{0,k,\delta}^{(\emptyset)} = p,\ k = 0,\ldots,2^n-1$. The channels $W_{s,k,\delta}^{(\mathbf{s})},\ k = 0,\ldots,2^{n-s}-1$, are independent copies of the same type of channel, meaning that their erasure probabilities are identical. Thus, if we are only interested in the erasure probability of a specific *type* $\mathbf{s}$ of channel we can simplify (3.16) and (3.17) by omitting the index k as

$$Z_{s,\delta}^{(\mathbf{s}-)} = T_\delta^- \left(Z_{s-1,\delta}^{(\mathbf{s})}\right) \triangleq 2Z_{s-1,\delta}^{(\mathbf{s})} - \left(Z_{s-1,\delta}^{(\mathbf{s})}\right)^2$$
$$+ \left(\overline{2Z_{s-1,\delta}^{(\mathbf{s})} - \left(Z_{s-1,\delta}^{(\mathbf{s})}\right)^2}\right)\delta, \tag{3.36}$$

$$Z_{s,\delta}^{(\mathbf{s}+)} = T_\delta^+ \left(Z_{s-1,\delta}^{(\mathbf{s})}\right) \triangleq \left(Z_{s-1,\delta}^{(\mathbf{s})}\right)^2 + \overline{\left(Z_{s-1,\delta}^{(\mathbf{s})}\right)^2}\delta, \tag{3.37}$$

with $Z_{0,\delta}^{(\emptyset)} = p$. The vector containing all $Z_{s,\delta}^{(\mathbf{s})},\ \mathbf{s} \in \{+,-\}^s$, variables is denoted by $\mathbf{Z}_{s,\delta}$. The random process ϵ_s can be rewritten for the faulty case as

$$\epsilon_{s,\delta} = \begin{cases} T_\delta^+(\epsilon_{s-1,\delta}) & \text{w.p. } 1/2, \\ T_\delta^-(\epsilon_{s-1,\delta}) & \text{w.p. } 1/2, \end{cases} \tag{3.38}$$

with $\epsilon_{0,\delta} = Z(W) = p$.

3.3.2.1 Properties of T_δ^+ and T_δ^-

In this section, we show some properties of the T_δ^+ and T_δ^- transformations, which will be useful to prove two negative results in the following section, as well as to interpret some of the numerical results of Section 3.3.6.

Property 1. *For $T_\delta^+(\epsilon)$ and $T_\delta^-(\epsilon)$, we have*

(i) $T_\delta^+(\epsilon) \geq \delta,\ \forall \epsilon, \delta \in [0,1]$,

(ii) $T_\delta^-(\epsilon) \geq \delta,\ \forall \epsilon, \delta \in [0,1]$,

Proof. For $T_\delta^+(\epsilon)$, we have

$$\epsilon^2 + (1-\epsilon^2)\delta \geq \delta \Leftrightarrow \tag{3.39}$$
$$(1-\delta)\epsilon^2 \geq 0, \tag{3.40}$$

which indeed holds for any $\epsilon, \delta \in [0,1]$. Similarly, for $T_\delta^-(\epsilon)$, we have

$$2\epsilon - \epsilon^2 + (1 - 2\epsilon + \epsilon^2)\delta \geq \delta \Leftrightarrow \tag{3.41}$$

$$(1-\delta)(2\epsilon - \epsilon^2) \geq 0, \tag{3.42}$$

which indeed holds for any $\epsilon, \delta \in [0,1]$. □

Property 2. *The fixed points of $T_\delta^+(\epsilon)$ are $\epsilon = 1$ and $\epsilon = \frac{\delta}{1-\delta}$. The unique fixed point of $T_\delta^-(\epsilon)$ for $\epsilon \in [0,1]$ is $\epsilon = 1$.*

Proof. The above property can easily be shown by solving $T_\delta^+(\epsilon) = \epsilon$ and $T_\delta^-(\epsilon) = \epsilon$ for ϵ, respectively, and noting that one solution of $T_\delta^-(\epsilon) = \epsilon$ is negative. □

Moreover, the following two properties of the process $\epsilon_{s,\delta}$ give us some first insight into the effect that the faulty decoder has on the decoding process.

Property 3. *The process $\epsilon_{s,\delta}$, $s = 0, 1, \ldots$, defined in* (3.38) *is a submartingale.*

Proof. Since $\epsilon_{s,\delta}$ is bounded, it holds that $\mathbb{E}(|\epsilon_{s,\delta}|) < \infty$. Moreover we have

$$\mathbb{E}(\epsilon_{s,\delta}|\epsilon_{s-1,\delta}) = \frac{1}{2}\left(T_\delta^+(\epsilon_{s-1,\delta}) + T_\delta^-(\epsilon_{s-1,\delta})\right) \tag{3.43}$$

$$= \frac{1}{2}\Big((1-\epsilon_{s-1,\delta}^2)\delta + 2\epsilon_{s-1,\delta} + (1 - 2\epsilon_{s-1,\delta} + \epsilon_{s-1,\delta}^2)\delta\Big) \tag{3.44}$$

$$= \epsilon_{s-1,\delta} + (1-\epsilon_{s-1,\delta})\delta \geq \epsilon_{s-1,\delta}. \tag{3.45}$$

□

Property 4. *For the expectation of the process $\epsilon_{s,\delta}$, $s = 0, 1, \ldots$, defined in* (3.38) *we have*

$$\mathbb{E}(\epsilon_{s,\delta}) = 1 - (1-\epsilon_0)(1-\delta)^s, \tag{3.46}$$

Proof. From the proof of Property 3, we know that

$$\mathbb{E}(\epsilon_{s,\delta}|\epsilon_{s-1,\delta}) = \epsilon_{s-1,\delta} + (1-\epsilon_{s-1,\delta})\delta. \tag{3.47}$$

By taking the expectation with respect to $\epsilon_{s-1,\delta}$ on both sides of (3.47), we have

$$\mathbb{E}(\epsilon_{s,\delta}) = \mathbb{E}(\epsilon_{s-1,\delta}) + (1 - \mathbb{E}((\epsilon_{s-1,\delta}))\delta \tag{3.48}$$

$$= (1-\delta)\mathbb{E}(\epsilon_{s-1,\delta}) + \delta, \tag{3.49}$$

with $\mathbb{E}(\epsilon_{0,\delta}) = \epsilon_{0,\delta} = p$. The solution of this recurrence relation is

$$\mathbb{E}(\epsilon_{s,\delta}) = 1 - (1-p)(1-\delta)^s. \tag{3.50}$$

□

Specifically, this tells us that, contrary to [6], the average erasure probability is not preserved by $T_\delta^+(\epsilon)$ and $T_\delta^-(\epsilon)$. Thus, even if fully reliable transmission were possible in the limit of infinite blocklength, the polar code would not be capacity achieving since $\mathbb{P}\left[\epsilon_{s,\delta} = 0\right] < 1-p$, meaning that the fraction of noiseless channels would be strictly smaller than the capacity of the BEC.

3.3.2.2 Impact on Synthetic Channel Polarization

Unfortunately, as the following proposition asserts, fully reliable transmission under faulty decoding is not possible.

Proposition 3. *Let $\mathcal{Q}$ denote the sample space of the process $\epsilon_{s,\delta}$ and let $\epsilon_{s,\delta}(q)$, $q \in \mathcal{S}$, denote a specific realization of $\epsilon_{s,\delta}$. Polarization does not happen under faulty SC decoding for the BEC in the sense that $\nexists q \in \mathcal{Q}$ such that $\epsilon_{s,\delta}(q) \xrightarrow{s\to\infty} 0$.*

Proof. This is a direct consequence of Property 1, since all $\epsilon_{s,\delta}(q)$ are produced by repeated applications of T_δ^+ and T_δ^- to $\epsilon_{0,\delta} = p$, so that $\epsilon_{s,\delta}(q) \geq \delta$, $\forall q \in \mathcal{Q}$. □

It turns out that we can prove the following stronger result, which states that, under faulty SC decoding over the BEC, almost all channels become asymptotically useless.

Proposition 4. *For the process $\epsilon_{s,\delta}$, $s = 0, 1, \ldots$, defined in* (3.38)*, we have $\epsilon_{s,\delta} \xrightarrow{\text{a.s.}} 1$.*

Proof. From Property 3, we know that $\epsilon_{s,\delta}$ is a bounded submartingale. Thus, it converges a.s. to some limiting random variable ϵ_∞. Moreover, from Property 4 we have

$$\mathbb{E}(\epsilon_{s,\delta}) = 1 - (1-p)(1-\delta)^s, \tag{3.51}$$

which directly implies that $\lim_{s\to\infty} \mathbb{E}(\epsilon_{s,\delta}) = 1$, since, by assumption, $\delta \in (0,1)$. Equivalently, and since $\epsilon_{s,\delta} \in [0,1]$, we can write

$$\lim_{s\to\infty} \mathbb{E}(|\epsilon_{s,\delta} - 1|) = 0, \tag{3.52}$$

which means, by definition, that $\epsilon_{s,\delta} \xrightarrow{L^1} 1$. Moreover, $\epsilon_{s,\delta} \xrightarrow{L^1} 1$ implies that $\epsilon_{s,\delta} \xrightarrow{\mathbb{P}} 1$. Since we know, due to the submartingale property, that $\epsilon_{s,\delta}$ also converges almost surely and almost sure convergence implies convergence in probability, all the aforementioned limits must be identical and we can conclude that $\epsilon_{s,\delta} \xrightarrow{\text{a.s.}} 1$. □

3.3.2.3 Synthetic Channel Ordering

In the case of non-faulty decoding, there exists a partial ordering of the synthetic channels with respect to the BEC erasure probability p. In order to explain this ordering, we first need to define the notion of "η-goodness".

Definition 1. A synthetic channel $W_s^{(\mathbf{s})}$ is said to be "η-good" if $Z_s^{(\mathbf{s})} \leq \eta$.

In the non-faulty case, it is easy to see that both $T^+(\epsilon)$ and $T^-(\epsilon)$ are increasing in ϵ, $\forall \epsilon \in [0,1]$. Thus, a synthetic channel that is η-good for a BEC with erasure probability p_1, will also be η-good for a BEC with erasure probability p_2 when $p_2 \leq p_1$.

In this section, we show that under faulty decoding the partial ordering with respect to the BEC parameter p is preserved and we show that a similar partial ordering exists with respect to the decoder erasure probability δ. To this end, in the following two properties we examine the monotonicity of $T_\delta^-(\epsilon)$ and $T_\delta^+(\epsilon)$ with respect to ϵ and δ.

Property 5. *Both $T_\delta^-(\epsilon)$ and $T_\delta^+(\epsilon)$ are*

(i) Increasing in ϵ, $\forall \epsilon \in [0,1]$.

(ii) Increasing in δ, $\forall \delta \in [0,1]$.

Proof. (i) $T_\delta^+(\epsilon)$ can be re-written as

$$T_\delta^+(\epsilon) = \epsilon^2 + (1-\epsilon^2)\delta \tag{3.53}$$
$$= \epsilon^2(1-\delta) + \delta. \tag{3.54}$$

Thus, for any fixed $\delta \in [0,1]$, $T_\delta^+(\epsilon)$ is clearly increasing in ϵ for any $\epsilon \in [0,1]$. Similarly, $T_\delta^-(\epsilon)$ can be re-written as

$$T_\delta^-(\epsilon) = 2\epsilon - \epsilon^2 + (1-2\epsilon+\epsilon^2)\delta \tag{3.55}$$
$$= (2\epsilon - \epsilon^2)(1-\delta) + \delta. \tag{3.56}$$

Thus, the partial derivative of $T_\delta^-(\epsilon)$ with respect to ϵ can easily be calculated as

$$\frac{\partial T_\delta^-(\epsilon)}{\partial \epsilon} = 2(1-\epsilon)(1-\delta), \tag{3.57}$$

which, for any fixed $\delta \in [0,1]$, is non-negative $\forall \epsilon \in [0,1]$.

(ii) Both $T_\delta^-(\epsilon)$ and $T_\delta^+(\epsilon)$ are linear functions of δ with a non-negative coefficient, so they are increasing $\forall \delta \in \mathbb{R}$. □

Proposition 5 (Monotonicity with respect to p)**.** *Let $p_1, p_2 \in (0,1)$, $p_2 \leq p_1$ and $\delta \in (0,1)$. A synthetic channel that is η-good for a decoder with a fixed erasure probability δ over a BEC with*

erasure probability p_1 is also η-good for the same decoder over a BEC with erasure probability p_2.

Proof. The erasure probability of any synthetic channel $W_{s,\delta}^{(\mathbf{s})}$ can be calculated by repeated applications of T_δ^- and T_δ^+ starting from p as

$$Z_{s,\delta}^{(\mathbf{s})}(p) = T_\delta^{s_s}\left(T_\delta^{s_{s-1}}\left(\cdots\left(T_\delta^{s_1}(p)\right)\right)\right), \tag{3.58}$$

where $\mathbf{s} = [s_s, s_{s-1}, \ldots, s_1]$ and $s_i \in \{+,-\}$, $i = 1, \ldots, s$. Since from Property 5(i) we know that both $T_\delta^-(\epsilon)$ and $T_\delta^+(\epsilon)$ are increasing with respect to ϵ, any composition of the two functions will also be increasing. Thus

$$Z_{s,\delta}^{(\mathbf{s})}(p_2) \leq Z_{s,\delta}^{(\mathbf{s})}(p_1) \leq \eta. \tag{3.59}$$

□

The following proposition states that there also exists a partial ordering of the synthetic channels with respect to the decoder erasure probability δ. This is a useful property, as it ensures that, for any given polar code, a decoder with internal erasure probability δ_2 will not perform worse than a decoder with internal erasure probability δ_1, where $\delta_2 \leq \delta_1$.

Proposition 6 (Monotonicity with respect to δ)**.** *Let $\delta_1, \delta_2 \in (0,1)$, $\delta_2 \leq \delta_1$ and $\epsilon \in (0,1)$. A synthetic channel that is η-good for a decoder with erasure probability δ_1 over a BEC with a fixed erasure probability ϵ is also η-good for a decoder with erasure probability δ_2 over the same channel.*

Proof. Similarly to the proof of Proposition 5, the proof stems directly from the monotonicity of $T_\delta^-(\epsilon)$ and $T_\delta^+(\epsilon)$ with respect to δ shown in Property 5(ii). □

3.3.3 Frame Erasure Rate Under Faulty SC Decoding

In this section, we adapt the framework of [124] to the case of faulty decoding in order to derive a lower bound on the frame erasure probability under faulty decoding. Let $P_e(\mathcal{A}_n)$ denote the frame erasure rate (FER) of a polar code of length 2^n with information set $\mathcal{A}_n$. From [6], we have the general upper bound

$$P_e(\mathcal{A}_n) \leq \sum_{\mathbf{s}\in\mathcal{A}_n} Z_n^{(\mathbf{s})} \triangleq P_e^{\mathrm{UB}}. \tag{3.60}$$

Furthermore, from [124] we have the lower bound

$$P_e(\mathcal{A}_n) \geq \sum_{\mathbf{s}\in\mathcal{A}_n} Z_n^{(\mathbf{s})} - \frac{1}{2}\sum_{\substack{\mathbf{s},\mathbf{t}\in\mathcal{A}_n:\\ \mathbf{s}\neq\mathbf{t}}} \left(Z_n^{(\mathbf{s})} Z_n^{(\mathbf{t})} + C_n^{(\mathbf{s},\mathbf{t})}\right) \triangleq P_e^{\mathrm{LB}} \tag{3.61}$$

where $\mathbf{C}_n \triangleq [C_n^{(\mathbf{s},\mathbf{t})} : \mathbf{s},\mathbf{t} \in \{+,-\}^n]$ denotes the covariance matrix of the random vector $\mathbf{E}_n$, where $C_n^{(\mathbf{s},\mathbf{t})} \triangleq \text{cov}[E_n^{(\mathbf{s})} E_n^{(\mathbf{t})}]$. It was shown in [124] that, in the non-faulty case, the elements of $\mathbf{C}_s, s = 1,\ldots,n$, can be calculated recursively from the elements of $\mathbf{C}_{s-1}$ and $Z_{s-1}^{(\mathbf{s})}$ as follows

$$C_s^{(\mathbf{s}-,\mathbf{t}-)} = 2\overline{Z_{s-1}^{(\mathbf{s})} Z_{s-1}^{(\mathbf{t})}} C_{s-1}^{(\mathbf{s},\mathbf{t})} + C_{s-1}^{(\mathbf{s},\mathbf{t})^2}, \tag{3.62}$$

$$C_s^{(\mathbf{s}-,\mathbf{t}+)} = 2\overline{Z_{s-1}^{(\mathbf{s})}} Z_{s-1}^{(\mathbf{t})} C_{s-1}^{(\mathbf{s},\mathbf{t})} - C_{s-1}^{(\mathbf{s},\mathbf{t})^2}, \tag{3.63}$$

$$C_s^{(\mathbf{s}+,\mathbf{t}-)} = 2Z_{s-1}^{(\mathbf{s})} \overline{Z_{s-1}^{(\mathbf{t})}} C_{s-1}^{(\mathbf{s},\mathbf{t})} - C_{s-1}^{(\mathbf{s},\mathbf{t})^2}, \tag{3.64}$$

$$C_s^{(\mathbf{s}+,\mathbf{t}+)} = 2Z_{s-1}^{(\mathbf{s})} Z_{s-1}^{(\mathbf{t})} C_{s-1}^{(\mathbf{s},\mathbf{t})} + C_{s-1}^{(\mathbf{s},\mathbf{t})^2}, \tag{3.65}$$

with $C_0^{(\emptyset,\emptyset)} = p(1-p)$. In the case of reliable decoding, the second sum in (3.61) goes to zero as n is increased [124] if $R = \frac{|\mathcal{A}_n|}{2^n} < 1-p$, so that

$$P_e(\mathcal{A}_n) \approx \sum_{\mathbf{s}\in\mathcal{A}_n} Z_n^{(\mathbf{s})}. \tag{3.66}$$

We can use the upper and lower bounds of (3.61) and (3.60) for the case of faulty decoding by replacing $Z_n^{(\mathbf{s})}$ with $Z_{n,\delta}^{(\mathbf{s})}$, and $C_n^{(\mathbf{s},\mathbf{t})}$ with $C_{n,\delta}^{(\mathbf{s},\mathbf{t})}$, where $\mathbf{C}_{n,\delta}^{(\mathbf{s},\mathbf{t})} \triangleq [C_{n,\delta}^{(\mathbf{s},\mathbf{t})} : \mathbf{s},\mathbf{t} \in \{+,-\}^n]$, is the covariance matrix of the random vector $\mathbf{E}_{n,\delta}$. In the case of faulty decoding, as n is increased, we know from Proposition 4 that almost all $Z_{n,\delta}^{(\mathbf{s})} Z_{n,\delta}^{(\mathbf{t})}$, $\mathbf{s},\mathbf{t} \in \mathcal{A}_n$, are equal to 1. Moreover, the non-diagonal elements of $C_{n,\delta}^{(\mathbf{s},\mathbf{t})}$ still converge to 0 for any $\mathbf{s},\mathbf{t}$, as almost all indicator variables become deterministic like in the fault-free case. Thus, for some n the lower bound of (3.61) becomes negative and can be replaced by the trivial lower bound $P_e(\mathcal{A}_n) \geq \max_{\mathbf{s}\in\mathcal{A}_n} Z_{n,\delta}^{(\mathbf{s})}$. Similarly, for some n the upper bound of (3.60) becomes greater than 1, so it can be replaced by the trivial upper bound $P_e(\mathcal{A}_n) \leq 1$. Clearly though, since $Z_{n,\delta}^{(\mathbf{s})}$ converges to 1 as n grows for almost all $\mathbf{s} \in \{+,-\}^n$, we have $\lim_{n\to\infty} P_e(\mathcal{A}_n) = 1$ for any $\mathcal{A}_n$ such that $\lim_{n\to\infty} \frac{|\mathcal{A}_n|}{2^n} \nrightarrow 0$.

3.3.3.1 Lower Bound on $P_e(\mathcal{A}_n)$ Under Faulty SC Decoding

We already have an efficient way to calculate $Z_{n,\delta}^{(\mathbf{s})}$ recursively (i.e., (3.36) and (3.37)), but, in order to evaluate P_e^{LB}, we still need to find an efficient way to calculate $\mathbf{C}_{n,\delta}$. To this end, we first introduce a property which we then combine with the results of [124] in order to obtain a recursive expression for $\mathbf{C}_{s,\delta}$, $s = 1,\ldots,n$.

Property 6. *Let X, Y denote two arbitrary random variables. Let Δ_1, Δ_2 denote two random variables with $\Delta_1, \Delta_2 \in \{0,1\}$ and $\mathbb{E}[\Delta_1] = \mathbb{E}[\Delta_2] = \delta$ that are independent of X, Y and of each other. Then, we have*

$$cov[X + (1-X)\Delta_1, Y + (1-Y)\Delta_2] = (1-\delta)^2 cov[X,Y]. \tag{3.67}$$

Proof. For simpler notation, let us define $X' \triangleq X + (1-X)\Delta_1$ and $Y' \triangleq Y + (1-Y)\Delta_2$. We then

have

$$\begin{aligned}
\operatorname{cov}\left[X', Y'\right] &= \mathbb{E}[X'Y'] - \mathbb{E}[X']\mathbb{E}[Y'] && (3.68)\\
&= \mathbb{E}[(1-\Delta_1)X + \Delta_1)((1-\Delta_2)Y + \Delta_2)] \\
&\quad - \mathbb{E}[(1-\Delta_1)X + \Delta_1]\mathbb{E}[(1-\Delta_2)Y + \Delta_2] && (3.69)\\
&\overset{(*)}{=} \mathbb{E}\left[(1-\Delta_1)(1-\Delta_2)\right](\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]) && (3.70)\\
&\overset{(**)}{=} (1-\delta)^2 \operatorname{cov}[X, Y], && (3.71)
\end{aligned}$$

where for $(*)$ we have used the independence of Δ_1 and Δ_2 from X and Y, while for $(**)$ we have used the independence between Δ_1 and Δ_2. □

Proposition 7. *The covariance matrix of the random vector* $\mathbf{E}_{s,\delta}$*, denoted by* $\mathbf{C}_{s,\delta} \triangleq [C_{s,\delta}^{(\mathbf{s},\mathbf{t})} : \mathbf{s}, \mathbf{t} \in \{+,-\}^s]$*, where* $\mathbf{C}_{s,\delta} \triangleq cov\left[E_{s,\delta}^{(\mathbf{s})} E_{s,\delta}^{(\mathbf{t})}\right]$*, can be computed in terms of* $\mathbf{C}_{s-1,\delta}$ *and* $\mathbf{Z}_{s-1,\delta}$ *as follows:*

$$C_{s,\delta}^{(\mathbf{s}-,\mathbf{t}-)} = (1-\delta)^2\left(2\overline{Z_{s-1,\delta}^{(\mathbf{s})} Z_{s-1,\delta}^{(\mathbf{t})}} C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})} + {C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})}}^2\right), \tag{3.72}$$

$$C_{s,\delta}^{(\mathbf{s}-,\mathbf{t}+)} = (1-\delta)^2\left(2\overline{Z_{s-1,\delta}^{(\mathbf{s})}} Z_{s-1,\delta}^{(\mathbf{t})} C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})} - {C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})}}^2\right), \tag{3.73}$$

$$C_{s,\delta}^{(\mathbf{s}+,\mathbf{t}-)} = (1-\delta)^2\left(2 Z_{s-1,\delta}^{(\mathbf{s})} \overline{Z_{s-1,\delta}^{(\mathbf{t})}} C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})} - {C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})}}^2\right), \tag{3.74}$$

$$C_{s,\delta}^{(\mathbf{s}+,\mathbf{t}+)} = (1-\delta)^2\left(2 Z_{s-1,\delta}^{(\mathbf{s})} Z_{s-1,\delta}^{(\mathbf{t})} C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})} + {C_{s-1,\delta}^{(\mathbf{s},\mathbf{t})}}^2\right), \tag{3.75}$$

with $C_0^{(\emptyset,\emptyset)} = p(1-p)$.

Proof. To avoid unnecessary repetition, we prove the result only for (3.75), as the remaining relations (3.72)–(3.74) can be derived in the same way. Recall that, in the case of faulty decoding, from (3.33) we have

$$E_{s,\delta}^{(\mathbf{s}+)} = {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}'' + \left(1 - {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}''\right)\Delta_s^{(\mathbf{s}+)}, \tag{3.76}$$

$$E_{s,\delta}^{(\mathbf{t}+)} = {E_{s-1,\delta}^{(\mathbf{t})}}' {E_{s-1,\delta}^{(\mathbf{t})}}'' + \left(1 - {E_{s-1,\delta}^{(\mathbf{t})}}' {E_{s-1,\delta}^{(\mathbf{t})}}''\right)\Delta_s^{(\mathbf{t}+)}. \tag{3.77}$$

Let us define $X \triangleq {E_{s-1,\delta}^{(\mathbf{s})}}' {E_{s-1,\delta}^{(\mathbf{s})}}''$, $Y \triangleq {E_{s-1,\delta}^{(\mathbf{t})}}' {E_{s-1,\delta}^{(\mathbf{t})}}''$, $\Delta_s^{(\mathbf{s}+)} \triangleq \Delta_1$, and $\Delta_s^{(\mathbf{t}+)} \triangleq \Delta_2$. Then, we can rewrite (3.76) as

$$E_{n,\delta}^{(\mathbf{s}+)} = X + (1-X)\Delta_1, \tag{3.78}$$

$$E_{n,\delta}^{(\mathbf{t}+)} = Y + (1-Y)\Delta_2, \tag{3.79}$$

where X and Y are identical to the update rule for $E_s^{(\mathbf{s}+)}$ and $E_s^{(\mathbf{t}+)}$ in the fault-free case given in (3.25), respectively. Using $\mathbb{E}\left[\Delta_s^{(\mathbf{s}+)}\right] = \mathbb{E}\left[\Delta_s^{(\mathbf{t}+)}\right] = \delta$, along with the fact that $\Delta_s^{(\mathbf{s}+)}$ and $\Delta_s^{(\mathbf{t}+)}$ are independent by assumption, we can apply Proposition 6 to the update formula for $\operatorname{cov}[X, Y]$ from [124] given in (3.65), in order to obtain (3.75). □

It is intuitively pleasing to note that, for $\delta = 0$ (i.e., for fault-free decoding), the expressions in (3.72)–(3.75) become identical to the expressions in (3.62)–(3.65).

3.3.4 Unequal Error Protection

As mentioned in Section 3.3, standard methods employed to enhance the fault tolerance of circuits, such as using larger transistors or circuit-level error correcting codes, are costly in terms of both area and power if the entire circuit needs to be protected. With this in mind, we note that in SC decoding of polar codes not all levels in the tree of MEs are of equal importance, meaning that it may suffice to employ *partial protection* of the decoder against hardware-induced errors. In fact, we shall see in Proposition 8, a careful application of such a protection method allows polarization to happen even in a faulty decoder while protecting only a constant fraction of the total decoder MEs.

Let n_{p} denote the number of levels that are protected, starting from level n of the tree (i.e., the root) and going towards the leaves. We assume that for these n_{p} levels we have $\delta = 0$. Let N_{p} denote the total number of protected MEs, where

$$N_{\mathrm{p}} = \begin{cases} \sum_{j=0}^{n_{\mathrm{p}}-1} 2^j = 2^{n_{\mathrm{p}}} - 1, & n_{\mathrm{p}} > 0, \\ 0, & n_{\mathrm{p}} = 0. \end{cases} \tag{3.80}$$

If we set $n_{\mathrm{p}} = (n+1) - n_{\mathrm{u}}$, where $n_{\mathrm{u}} > 0$ is a *fixed* number of unprotected levels, then the fraction of the decoder that is protected converges to a constant as n grows. Indeed, we have

$$\lim_{n\to\infty} \frac{N_{\mathrm{p}}}{N_{\mathrm{ME}}} = \lim_{n\to\infty} \frac{2^{(n+1)-n_{\mathrm{u}}} - 1}{2^{n+1} - 1} = 2^{-n_{\mathrm{u}}}. \tag{3.81}$$

In this case, the process $\epsilon_{s,\delta}$ can be rewritten as

$$\epsilon_{s,\delta} = \begin{cases} T_\delta^+(\epsilon_{s-1,\delta}), & \text{w.p. } 1/2, \\ T_\delta^-(\epsilon_{s-1,\delta}), & \text{w.p. } 1/2, \end{cases} \quad \text{if } s = 1,\dots,n_{\mathrm{u}}, \\ \begin{cases} T^+(\epsilon_{s-1,\delta}), & \text{w.p. } 1/2, \\ T^-(\epsilon_{s-1,\delta}), & \text{w.p. } 1/2, \end{cases} \quad \text{if } s = n_{\mathrm{u}}+1,\dots,n. \tag{3.82}$$

The following proposition asserts that the protection of a constant fraction of the decoder is sufficient to ensure that polarization happens as n grows.

Proposition 8. *Setting $n_p = s - n_u$ for any fixed n_u suffices to ensure that $\epsilon_{s,\delta}$ converges almost surely to a random variable $\epsilon_\infty \in \{0,1\}$. However, the unprotected levels result in a rate loss $\Delta R(\delta, p, n_u)$, in the sense that $P(\epsilon_\infty = 0) = 1 - p - \Delta R(\delta, p, n_u)$, which can be calculated in closed form as*

$$\Delta R(\delta, p, n_u) = (1 - (1-\delta)^{n_u})(1-p). \tag{3.83}$$

Proof. The process $\epsilon_{s,\delta}$ as defined in (3.82) is a submartingale for $s \le n_{\mathrm{u}}$, but it becomes a

martingale for $s > n_{\mathrm{u}}$. Thus, for $s > n_{\mathrm{u}}$ we have $\mathbb{E}(\epsilon_{s,\delta}) = \mathbb{E}(\epsilon_{n_{\mathrm{u}},\delta})$. Using the arguments from [6], we can show that $\epsilon_{s,\delta}$ converges almost surely to a random variable $\epsilon_\infty \in \{0,1\}$ with $P(\epsilon_\infty = 0) = 1 - \mathbb{E}(\epsilon_{n_{\mathrm{u}}}) \leq 1 - p$. Equivalently, $P(\epsilon_\infty = 0) = 1 - p - \Delta R(\delta, \epsilon, n_{\mathrm{u}})$ for $\Delta R(\delta, \epsilon, n_{\mathrm{u}}) = \mathbb{E}(\epsilon_{n_{\mathrm{u}}}) - p$. Using the closed form expression for $\mathbb{E}(\epsilon_{s,\delta})$ from Property 4, we get

$$\begin{aligned} \Delta R(\delta, p, n_{\mathrm{u}}) &= \mathbb{E}(\epsilon_{n_{\mathrm{u}}}) - p && (3.84)\\ &= 1 - (1-p)(1-\delta)^{n_{\mathrm{u}}} - p && (3.85)\\ &= \left(1 - (1-\delta)^{n_{\mathrm{u}}}\right)(1-p). && (3.86) \end{aligned}$$

□

Proposition 8 implies that, when partial protection of the decoder is employed, polar codes are still not capacity achieving, but they can nevertheless be used for reliable transmission at any rate R such that $R < 1 - p - \Delta R(\delta, p, n_{\mathrm{u}})$.

3.3.5 Optimal Blocklength Under Faulty SC Decoding

In the finite blocklength regime, which is of practical interest, there are two clashing effects occurring. On one side, we have the polarization process, which tends to *decrease* the FER of the code as the blocklength is increased, but on the other side we have the internal erasures of the decoder which tend to *increase* the FER of the code as the blocklength is increased. From Proposition 4 we already know that, as the blocklength is increased towards infinity, the latter effect dominates and the resulting polar code becomes asymptotically useless. However, there must exist at least one blocklength which minimizes the FER and it is of great practical interest to identify this length.

Since this is a finite-length problem with practical applications, there will usually be a predefined maximum blocklength $n_{\max}$ for which a decoder is implementable with acceptable complexity. Thus, for a given $n_{\max}$, we define $\mathcal{N} = \{0, \ldots, n_{\max}\}$ as the set of n values of interest. For a given code rate R, we define the n^* which leads to the blocklength with the lowest FER under faulty decoding $N^* = 2^{n^*}$ as

$$n^* = \arg\min_{n \in \mathcal{N}} P_e(\mathcal{A}_n). \tag{3.87}$$

A simple way to identify the optimal blocklength is to perform extensive Monte-Carlo simulations of the codes for all $n \in \mathcal{N}$. However, we can find the solution more efficiently by using the bounds on $P_e(\mathcal{A}_n)$ given by (3.60) and (3.61). First, we study the special case where $p < \delta$. More specifically, the following proposition shows that, when $p < \delta$, it is optimal in terms of the FER to use *uncoded* transmission, as the faulty decoder can only increase the FER.

Proposition 9. *If $p < \delta$, then $n^* = 0$.*

Proof. The FER for $n = 0$ (i.e., uncoded transmission) over a BEC(p) is equal to p. From Property 1, we know that $Z_{n,\delta}^{(\mathbf{s})} \geq \delta,\ \forall \mathbf{s} \in \{+,-\}^n$. Since $p < \delta$ by assumption, we have $Z_{n,\delta}^{(\mathbf{s})} > p,\ \forall \mathbf{s} \in \{+,-\}^n$. Thus, using the trivial lower bound on the FER, i.e., $P_e^{\text{LB}} = \max_{\mathbf{s} \in \mathcal{A}_n} Z_{n,\delta}^{(\mathbf{s})}$, we can see that $P_e^{\text{LB}} > p$ for any $\mathcal{A}_n$ such that $|\mathcal{A}_n| > 0$. Thus, in this special case coded transmission with any blocklength such that $n > 0$ and at any rate $R > 0$, leads to a higher FER than uncoded transmission. □

In general, we can efficiently evaluate $P_e^{\text{UB}}(\mathcal{A}_n)$ and $P_e^{\text{LB}}(\mathcal{A}_n)$ for all $n \in \mathcal{N}$ for a given rate R [124]. Using these values, we can deduce whether there exists a single $n \in \mathcal{N}$ satisfying the following inequality

$$P_e^{\text{UB}}(\mathcal{A}_n) \leq P_e^{\text{LB}}(\mathcal{A}_{n'}),\ \forall n' \in \mathcal{N}. \tag{3.88}$$

If there exists such a unique $n \in \mathcal{N}$, then clearly this is the optimal n^*. Otherwise, we need to examine (via Monte-Carlo simulations) all $n \in \mathcal{N}$ for which $P_e^{\text{UB}}(\mathcal{A}_n)$ and $P_e^{\text{LB}}(\mathcal{A}_n)$ overlap, i.e., for which $\exists n' \in \mathcal{N}$ and $\exists \text{B} \in \{\text{UB}, \text{LB}\}$ such that

$$P_e^{\text{LB}}(\mathcal{A}_{n'}) \leq P_e^{\text{B}}(\mathcal{A}_n) \leq P_e^{\text{UB}}(\mathcal{A}_{n'}). \tag{3.89}$$

3.3.6 Numerical results

In this section we provide some numerical results to explore the process $\epsilon_{s,\delta}$, as well as the FER performance of polar codes constructed based on this process. Moreover, we use the FER bounds derived in Section 3.3.3 in order to find the optimal blocklength for polar a polar code under faulty SC decoding and we explore the effectiveness of the unequal error protection scheme described in Section 3.3.4.

Remark Most of the results in this section are presented for a decoder erasure probability of $\delta = 10^{-6}$. From Property 1, we know that the erasure probability of the synthetic channels is lower bounded by δ. Moreover, from (3.60), we know that the frame error rate is upper bounded by the sum of the erasure probabilities of the synthetic channels used to transmit information. In the numerical experiments we did, we saw that the same number also provides a good lower bound for most code rates. Thus, have we selected $\delta = 10^{-6}$ as this leads to frame error rates that are practically relevant for the blocklengths that we have considered.

3.3.6.1 Bhattacharyya Parameters

In Figure 3.7, we show the sorted values $Z_{n,\delta}^{(\mathbf{s})},\ \mathbf{s} \in \{+,-\}^n$, for polar codes with $n = 8, 10, 12$, designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$. We observe that we always have $Z_{n,\delta}^{(\mathbf{s})} \geq \delta$, as predicted by Property 1. Moreover, $\epsilon = \frac{\delta}{1-\delta}$ is a fixed point of $T_\delta^+(\epsilon)$, but it is not a fixed point of $T_\delta^-(\epsilon)$ (whereas $\epsilon = 1$ is a fixed point for both), resulting in the staircase-like

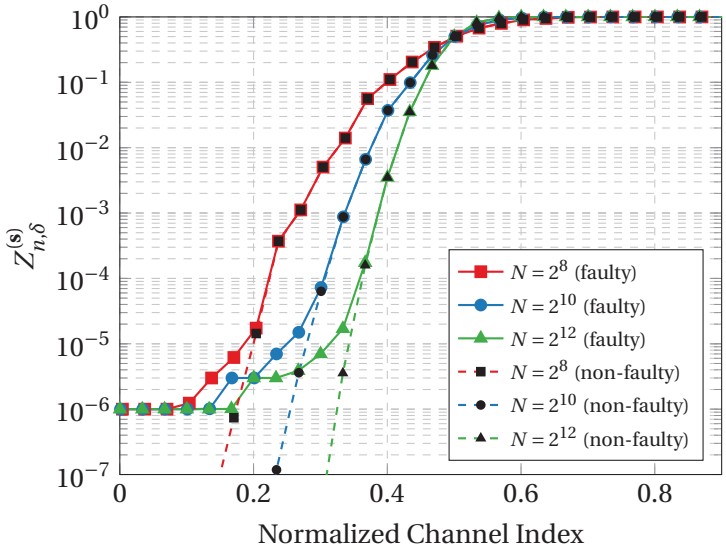


Figure 3.7 – Sorted $Z_{n,\delta}^{(\mathbf{s})}$, $\mathbf{s} \in \{+,-\}^n$ and $Z_n^{(\mathbf{s})}$, $\mathbf{s} \in \{+,-\}^n$, values for polar codes of length $N = 256, 1024, 4096$, designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and non-faulty decoding, respectively.

structure that we can observe in Figure 3.7.

3.3.6.2 Frame Erasure Rate

In Figure 3.8, we present the evaluation of P_e^{UB} and P_e^{LB} as a function of R and for $N = 256, 1024, 2048$, for a faulty SC decoder with $\delta = 10^{-6}$ and transmission over the BEC(0.5). We observe that, especially for low rates, P_e^{UB} and P_e^{LB} are practically indistinguishable. For rates $R > 0.30$ we start observing a difference between the lower bound and the upper bound, while for $R > 0.40$ both the upper bound and the lower bound break down and should be replaced by their trivial versions $P_e^{\mathrm{UB}} = 1$ and $P_e^{\mathrm{LB}} = \max_{\mathbf{s} \in \mathcal{A}_n} Z_{n,\delta}^{(\mathbf{s})}$. Moreover, we observe that over a wide range of rates the FER under SC decoding actually increases when the blocklength is increased, contrary to the fault-free case where increasing the blocklength generally decreases the FER. This can be explained if we recall that $Z_{n,\delta}^{(\mathbf{s})} \geq \delta$. Thus, by increasing the blocklength while keeping the rate fixed, we are increasing the number of terms in (3.66), and since some of these terms do not decrease beyond some point, the value of the sum can increase.

3.3.6.3 Optimal Blocklength

An example of the evaluation of P_e^{UB} and P_e^{LB} for $N = 2^n$, $n = 4, \ldots, 12$, and code rates $R \in \{0.1250, 0.1875, 0.2500\}$ (where $K = \lceil RN \rceil$) is shown in Figure 3.9 under faulty SC decoding with $\delta = 10^{-6}$ over a BEC(0.5). We observe that the bounds are tight enough in this case so that there always exists a unique $n \in \mathcal{N}$ that satisfies (3.88). Thus, for $R = 0.1250$ the optimal blocklength is $N = 128$, for $R = 0.1875$ the optimal blocklength is $N = 256$, and finally for $R = 0.2500$ the

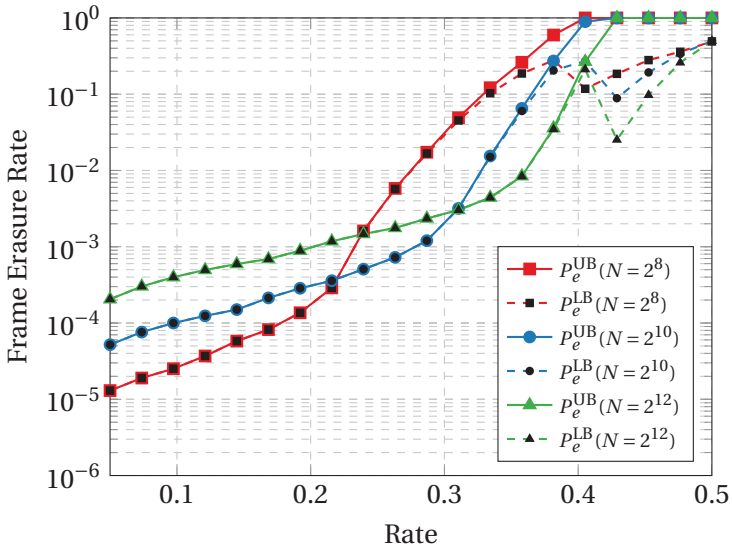


Figure 3.8 – Evaluation of P_e^{UB} and P_e^{LB} for polar codes of lengths $N = 256, 1024, 4096$, designed for the BEC(0.5) with $\delta = 10^{-6}$.

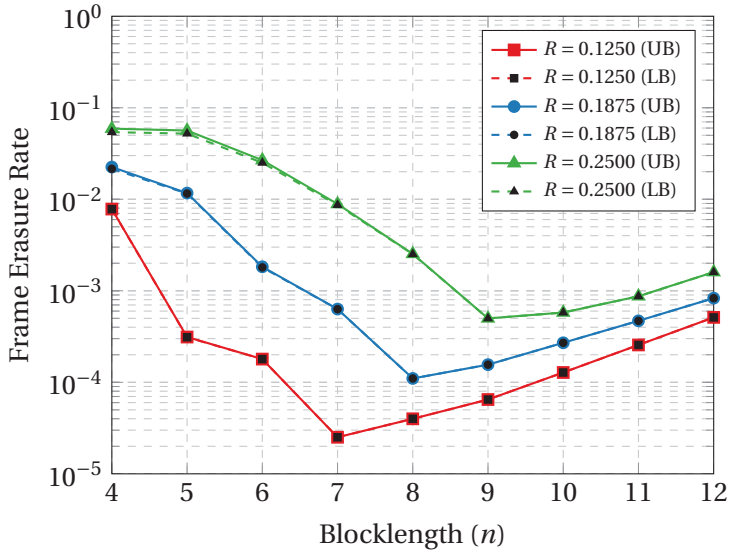


Figure 3.9 – Evaluation of P_e^{UB} and P_e^{LB} for $N = 2^n$, $n = 0, \ldots, 12$, and various code rates $R \in \{0.1250, 0.1875, 0.2500\}$ for transmission over a BEC with erasure probability 0.5 under faulty SC decoding with $\delta = 10^{-6}$.

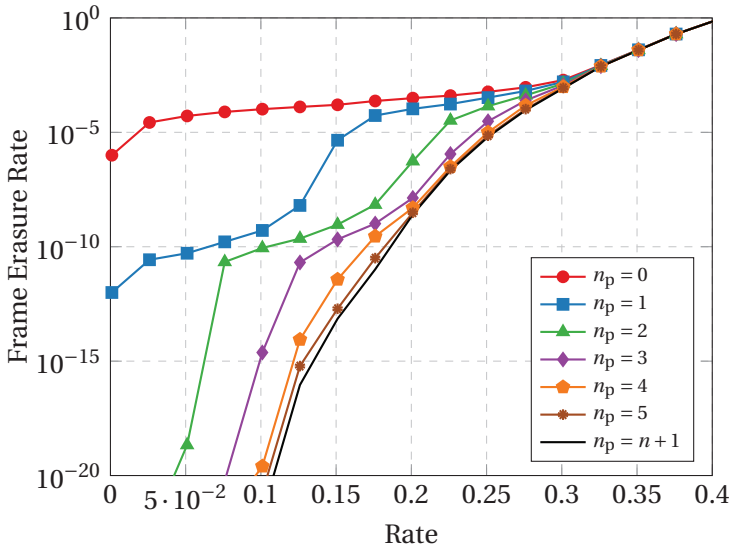


Figure 3.10 – FER for a polar code of length $N = 1024$ designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and $n_p = 0, \ldots, 5$, protected decoding levels. Protecting $n_p = n+1$ levels is equivalent to using a non-faulty decoder.

optimal blocklength is $N = 512$.

3.3.6.4 Unequal Error Protection

The effect of the partial protection for a finite length code is illustrated in Figure 3.10, where we present $P_e^{\mathrm{UB}}(\mathcal{A}_n)$ for $N = 2^{10} = 1024$ and $\delta = 10^{-6}$ when $n_p = 0, \ldots, 5$, levels of the tree are protected. To improve readability, we intentionally omit $P_e^{\mathrm{LB}}(\mathcal{A}_n)$ from the figure. However, we have already seen that the bounds are tight, especially for low rates, so using only the upper bound is sufficient to illustrate the effect of unequal error protection. We observe that protecting only the root node already improves the performance significantly, especially for the lower rates. When $n_p = 5$, the performance of the faulty SC decoder is almost identical to the non-faulty decoder in the examined FER region and it is remarkable that this performance improvement is achieved by protecting only $\frac{N_p}{N_{\mathrm{ME}}} = \frac{31}{2047} \approx 1.5\%$ of the decoder. Moreover, in Figure 3.11, we present $P_e^{\mathrm{UB}}(\mathcal{A}_n)$ for $N = 512, 1024, 2048$, and $\delta = 10^{-6}$ with $n_p = n - 5$, so that the protected part for each N is fixed to approximately 1.5% of the decoder. We observe that, contrary to the results of Section 3.3.6, increasing the blocklength actually decreases $P_e(\mathcal{A}_n)$ in the examined FER region, as in the case of the non-faulty decoder.

3.4 Min-Sum Decoding of LDPC Codes with Faulty Memories

Apart from polar codes, there has also been significant interest in studying the performance of LDPC codes under decoding where messages are not always computed or stored correctly. An important reason for this interest is that LDPC decoders are dominated by memory which

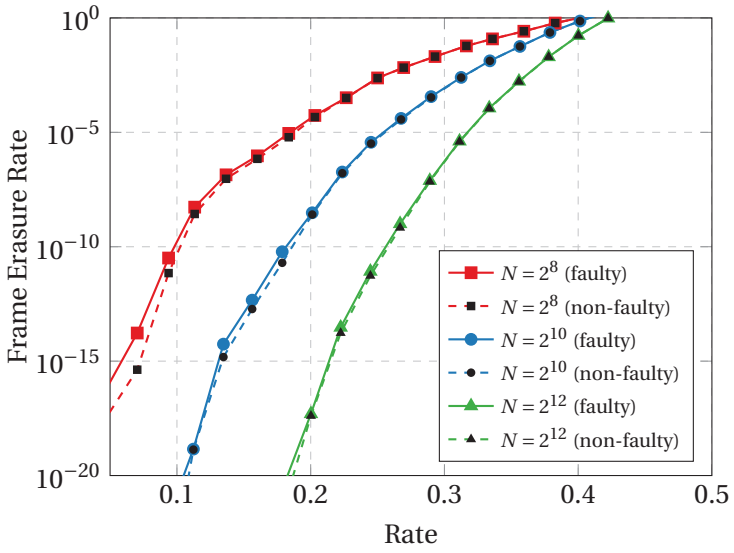


Figure 3.11 – FER for polar codes of length $N = 512, 1024, 2048$, designed for the BEC(0.5) under faulty SC decoding with $\delta = 10^{-6}$ and $n_{\mathrm{p}} = n - 5$ protected decoding levels.

is error prone. Moreover, LDPC codes can be studied analytically using a technique called *density evolution* (DE). Density evolution tracks the average probability density functions of the messages exchanged between the variable and check nodes at each decoding iteration in the limit of infinite blocklength [132]. DE operates under the assumption that all messages are independent because it can be shown that the Tanner graph of a randomly constructed LDPC code is asymptotically cycle-free. A concentration result guarantees that the performance of individual codes chosen from an ensemble is close to the ensemble average performance with high probability [132].

In [133] the Gallager A and the sum-product algorithms are analyzed under faulty decoding and an important concentration result is proven that makes the DE analysis valid and meaningful in the case of faulty decoding. Similar analyses are provided in [134, 135] for the Gallager B algorithm, while more general finite-alphabet decoders were considered in [136].

The aforementioned studies provide important insight into the behavior of LDPC codes under unreliable iterative decoding. However, since the study of faulty decoders is practically motivated, it is also important to study specific decoders which are widely used in practice, such as the min-sum decoder [137]. Moreover, the distribution of the faults for the non-binary message alphabet decoders studied in [133, 136] is not chosen based on a model that could describe a hardware implementation reasonably well. Thus, in this section we introduce a bit-level memory fault model and we derive the corresponding density evolution analysis for faulty decoding of LDPC codes using a quantized version of the min-sum decoding algorithm.

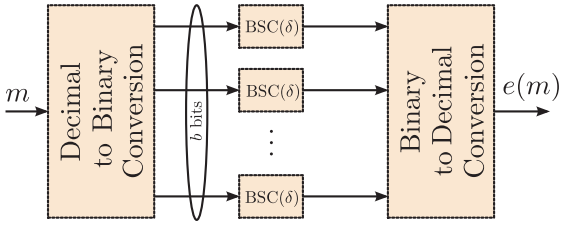


Figure 3.12 – Message fault model: an incoming b-bit noiseless message of value m is passed through b independent BSC(δ) channels, resulting in the faulty message $e(m)$.

3.4.1 Channel Model and Memory Fault Model

3.4.1.1 Channel Model

We assume that transmission of each codeword $\mathbf{c} \in \mathcal{C}$ takes place over an additive white Gaussian noise (AWGN) channel using binary phase-shift keying (BPSK) modulation, which can be modeled as

$$y_i = x_i + w_i,\ w_i \sim \mathcal{N}\left(0, \sigma^2\right),\ i = 1, \ldots, N, \tag{3.90}$$

where $x_i = 1 - 2c_i$. Moreover, for an AWGN channel the channel LLR is computed as

$$L(y_i) \triangleq \ln \frac{p(y_i|x_i = +1)}{p(y_i|x_i = -1)} = \ln \frac{p(y_i|c_i = 0)}{p(y_i|c_i = 1)} = -\frac{2y_i}{\sigma^2}. \tag{3.91}$$

3.4.1.2 Fault Model

We assume quantized MS decoding as described in Section 1.4.2.3 using a sign-magnitude binary representation for all message values $m \in \mathcal{M}$. The memory read errors are modeled as independent and identically distributed (i.i.d.) random bit-flips. Thus, all faults are transient, as in [133], and independent of the stored message. More precisely, at each iteration, each bit of the binary representation of the messages used to compute (1.41)–(1.43) is passed through a binary symmetric channel (BSC) with crossover probability δ, denoted by BSC(δ). We denote the set of all possible binary error patterns by $\mathcal{E}$ and the resulting faulty message after applying $e \in \mathcal{E}$ to a message of value m by $e(m)$. The distribution of the error patterns is

$$\mathbb{P}(e) = \delta^{\mathrm{w_H}(e)} (1 - \delta)^{b - \mathrm{w_H}(e)}, \quad e \in \mathcal{E}, \tag{3.92}$$

where $\mathrm{w_H}(e)$ denotes the Hamming weight of e. The fault model and its application to quantized MS decoding are illustrated in Figure 3.12, Figure 3.13, and Figure 3.14.

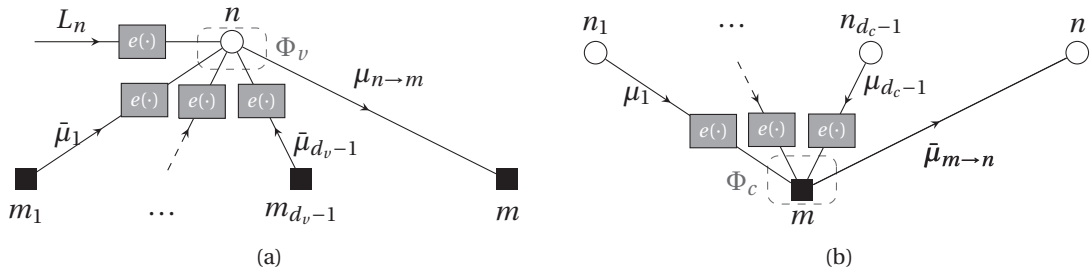


Figure 3.13 – Faulty variable node update for $\mathcal{N}(n) = \{m, m_1, \dots, m_{d_v-1}\}$ (a) and faulty check node update (b) for $\mathcal{N}(m) = \{n, n_1, \dots, n_{d_c-1}\}$.

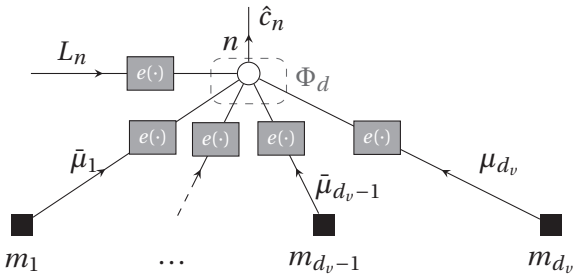


Figure 3.14 – Faulty decision node update for $\mathcal{N}(n) = \{m, m_1, \dots, m_{d_v}\}$.

3.4.2 Density Evolution for Faulty Quantized MS Decoding

In this section, we derive the DE equations for faulty quantized MS decoding of (d_v, d_c)-regular LDPC codes. First, we need to ensure that some important properties that make the DE analysis valid and meaningful still hold in the case of faulty decoding. Specifically, the existence of transient errors using the error model introduced in Section 3.4.1.2 does not affect the asymptotic cycle-free property of the decoding graph and with our error model the faulty messages are independent, because the corresponding non-faulty messages from which they are derived are independent and the errors affecting a specific message are independent of the message value.

3.4.2.1 Restriction to the All-One Modulated BPSK Codeword

An additional important property that makes the DE analysis tractable is *channel symmetry* and *decoder update rule symmetry*. If both the channel and the decoder are symmetric (in a sense that will be explained in the sequel), then the DE analysis can be restricted to the all-zero LDPC codeword, or, equivalently, the all-one modulated BPSK codeword [132]. We will now show that both symmetries hold in faulty quantized MS decoding.

The AWGN channel is symmetric in the sense that

$$L(-y_i) = -L(y_i). \tag{3.93}$$

It can easily be seen that the following symmetries hold for the MS update rules

$$\Phi_v^{\mathrm{MS}}(-L, -\bar{\mu}_1, \ldots, -\bar{\mu}_{d_v-1}) = -\Phi_v^{\mathrm{MS}}(L, \bar{\mu}_1, \ldots, \bar{\mu}_{d_v-1}), \tag{3.94}$$

$$\Phi_c^{\mathrm{MS}}(b_1\mu_1, \ldots, b_{d_c-1}\mu_{d_c-1}) = \prod_{i=1}^{d_c-1} b_i \Phi_c^{\mathrm{MS}}(\mu_1, \ldots, \mu_{d_c-1}), \tag{3.95}$$

where $b_i \in \{\pm 1\},\ i = 1, \ldots, d_c - 1$. Under the update rule symmetry defined in (3.94) and (3.95) and under channel symmetry, as defined in (3.93), the probability of bit error is independent of the transmitted codeword [132]. Thus, the asymptotic analysis of MS decoding can be restricted to the all-one BPSK codeword. The following proposition ensures that the same simplification can be applied to faulty quantized MS decoding with our error model.

Proposition 2. *When messages are represented in sign-magnitude form, MS decoder symmetry is preserved under faulty decoding with read errors modeled as i.i.d. bit-flips.*

Proof. Due to quantizer symmetry, we have $q_\Delta\left(L(-y_i)\right) = q_\Delta(-L(y_i)) = -q_\Delta(L(y_i))$, so channel symmetry holds. Moreover, when using sign-magnitude representation where "+0" and "−0" exist as distinct values, it holds that

$$e(-m) = -e(m), \quad \forall e \in \mathcal{E},\ m \in \mathcal{M}. \tag{3.96}$$

Thus, for the variable node update rule, we have

$$\Phi_v^{\mathrm{MS}}\left(-e(L), -e(\bar{\mu}_1), \ldots, -e(\bar{\mu}_{d_v-1})\right) = -\Phi_v^{\mathrm{MS}}\left(e(L), e(\bar{\mu}_1), \ldots, e(\bar{\mu}_{d_v-1})\right) \tag{3.97}$$

Similarly, for $b_i \in \{\pm 1\},\ i = 1, \ldots, d_c - 1$, we have

$$\Phi_c^{\mathrm{MS}}\left(e(b_1\mu_1), \ldots, e(b_{d_c-1}\mu_{d_c-1})\right) = \prod_{i=1}^{d_c-1} b_i \Phi_c^{\mathrm{MS}}\left(e(\mu_1), \ldots, e(\mu_{d_c-1})\right) \tag{3.98}$$

meaning that update rule symmetry holds for both variable nodes and check nodes. Moreover, we assume that, whenever $m = 0$ appears, a uniform random choice between "+0" and "−0" is made, so that the bit error rate when $m = 0$ is always $1/2$ independently of the codeword bit value. □

3.4.2.2 Density Evolution for Quantized MS Decoding

Since we have shown that all required properties hold for the DE analysis to be valid, we can now proceed with the formulation of the DE equations. We first describe DE for non-faulty MS decoding and we then extend it to the case of faulty decoding.

Let $p^{\ell}(m)$ and $q^{\ell}(m)$ denote the probability mass functions (PMFs) of the VN-to-CN and the CN-to-VN messages at iteration $\ell \geq 1$, respectively, and let $p^0(m)$ denote the PMF of the channel LLR messages assuming that the all-one BPSK codeword was transmitted. We have

$$p^0(l_i) = \frac{1}{\sqrt{8\pi\sigma^{-2}}} \int_{t_i} e^{-\left(x-\frac{2}{\sigma^2}\right)^2 \cdot \frac{\sigma^2}{4}} dx, \quad \forall l_i \in \mathcal{M}, \tag{3.99}$$

where t_i denotes the quantization interval corresponding to m_i, as defined in (1.45). The CN-to-VN message density is given by

$$q^{\ell}(m) = \begin{cases} \Phi_-^{\ell}(m) - \Phi_-^{\ell}(m-1), & m < 0, \\ 1 - \left(1 - p^{\ell}(0)\right)^{d_c - 1}, & m = 0, \\ \Phi_+^{\ell}(m+1) - \Phi_+^{\ell}(m), & m > 0, \end{cases} \tag{3.100}$$

where $\Phi_-^{\ell}(m)$ and $\Phi_+^{\ell}(m)$ are defined as

$$\Phi_+^{\ell}(m) = \sum_{\substack{k=0, \\ k \text{ even}}}^{d_c-1} \binom{d_c-1}{k} \left(A_+^{\ell}(m)\right)^k \left(A_-^{\ell}(m)\right)^{d_c-k-1} \tag{3.101}$$

$$\Phi_-^{\ell}(m) = \sum_{\substack{k=0, \\ k \text{ odd}}}^{d_c-1} \binom{d_c-1}{k} \left(A_+^{\ell}(m)\right)^k \left(A_-^{\ell}(m)\right)^{d_c-k-1} \tag{3.102}$$

and

$$A_+^{\ell}(m) = \sum_{x=m}^{l_{2^b-2}} p^{\ell}(x), \quad m > 0, \tag{3.103}$$

$$A_-^{\ell}(m) = \sum_{x=l_0}^{m} p^{\ell}(x), \quad m < 0. \tag{3.104}$$

The VN-to-CN message density is given by

$$p^{\ell}(m) = p^0(m) \otimes \left(q^{(\ell-1)}(m)\right)^{\otimes(d_v-1)}, \tag{3.105}$$

where $\otimes$ denotes the convolution and $q^0(m) = \delta[m]$, where $\delta[m]$ is the Kronecker delta function. The density of the quantity used for bit decisions is given by

$$d^{\ell}(m) = p^0(m) \otimes \left(q^{(\ell-1)}(m)\right)^{\otimes d_v}. \tag{3.106}$$

When applying (3.105) and (3.106), any probability mass that corresponds to values smaller than l_0 or larger than l_{2^b-2} is added to the mass corresponding to l_0 or l_{2^b-2}, respectively.

3.4.2.3 Density Evolution for Faulty Quantized MS Decoding

Let $f_\delta(P)(m)$ denote the probability of a faulty message m, $m \in \mathcal{M}$, where P is the distribution of the non-faulty messages m', i.e., P can be p^ℓ or q^ℓ. We have

$$f_\delta(P)(m) = \sum_{\substack{e\in\mathcal{E}, m'\in\mathcal{M}: \\ e(m')=m}} P(m')\mathbb{P}(e). \tag{3.107}$$

For each value m, there are 2^b pairs (e, m') such that $e(m') = m$. Since there are $2^b - 1$ values for m,[1] evaluating $f_\delta(P)$ requires the calculation of approximately 2^{b+1} terms.

Unreliable memory reads cause errors in the *input* messages of (1.41)–(1.43). Thus, DE for faulty MS decoding with transient memory read errors can be formulated by replacing the p^ℓ and q^ℓ distributions that appear on the right-hand side of (3.100)–(3.106) with $f_\delta\left(p^\ell\right)$ and $f_\delta\left(q^\ell\right)$, respectively.

3.4.3 Bit-Error Probability and Decoding Threshold

Let $P_e^\ell(\sigma^2)$ denote the bit-error probability at iteration ℓ when transmission takes place over an AWGN channel with noise variance σ^2. Under the all-zero codeword assumption, a bit-error occurs when the bit-decision taken by (1.43) is equal to 1, or, equivalently, when the decision LLR is negative. We note that when the decision LLR is exactly equal to zero, the bit is decoded as 0 or 1 randomly and with equal probability. Since the PMF of the decision LLRs is known from the DE analysis, the bit-error probability $P_e^\ell(\sigma^2)$ can be easily computed as follows

$$P_e^\ell(\sigma^2) \triangleq \frac{1}{2}d^\ell(0) + \sum_{m=l_0}^{l_{2^{b-1}-2}} d^\ell(m). \tag{3.108}$$

In non-faulty decoding, the *decoding threshold* corresponds to the worst channel parameter for which asymptotically error-free transmission is possible in the sense that the bit-error probability converges to zero. More specifically, for the AWGN channel the decoding threshold is defined as [132]

$$\sigma_*^2 \triangleq \sup\left\{\sigma^2 \geq 0 : \lim_{\ell\to\infty} P_e^\ell(\sigma^2) = 0\right\}. \tag{3.109}$$

Under faulty decoding, in some cases the bit-error probability is lower bounded by a strictly non-zero quantity [133], making the threshold definition of (3.109) meaningless. Thus, the threshold for faulty decoding was re-defined in [133] as

$$\sigma_*^2(\eta) \triangleq \sup\left\{\sigma^2 \geq 0 : \lim_{\ell\to\infty} P_e^\ell(\sigma^2) \leq \eta\right\}, \tag{3.110}$$

[1] Recall that the decimal value 0 corresponds to two binary patterns (i.e., "+0" and "−0").

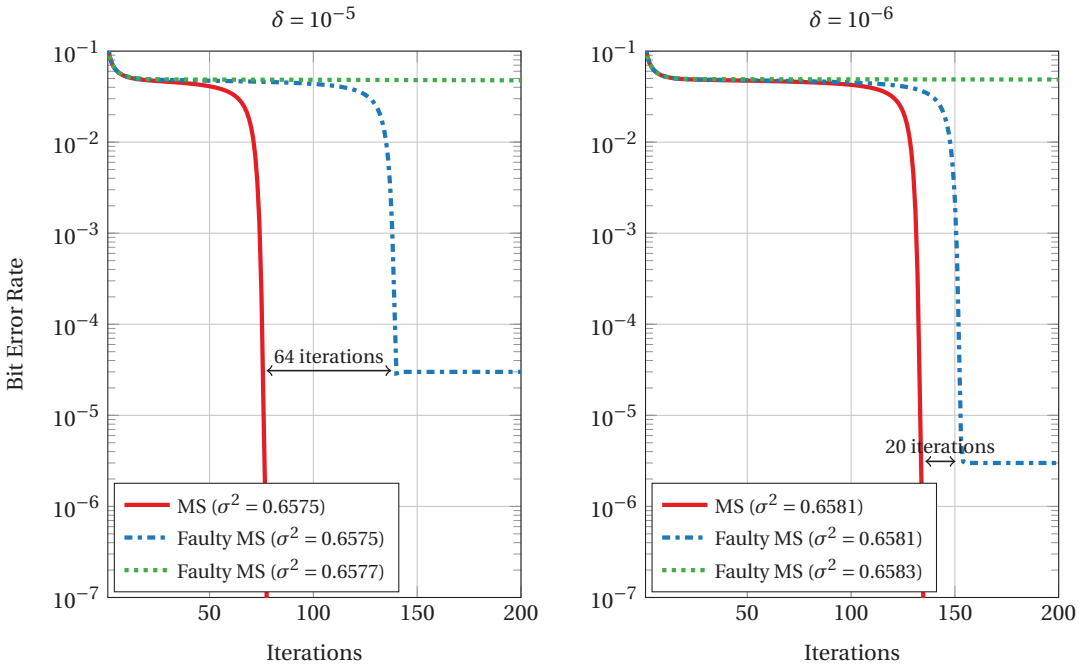


Figure 3.15 – Error probability for a $(3,6)$-regular LDPC code under faulty MS and MS decoding for $\delta = 10^{-5}$ and $\delta = 10^{-6}$. The calculated thresholds are $\sigma^2_*(10, 10^{-5}) = 0.6576$ and $\sigma^2_*(10, 10^{-6}) = 0.6582$.

where η is some target bit-error probability.

3.4.4 Numerical Results

For all examined (d_v, d_c)-regular ensembles and σ^2, our numerical results consistently show that $P_e^{\ell}\left(\sigma^2\right) \geq \delta$. This observation can help us in choosing a meaningful value for η. Specifically, we choose $\eta = \alpha\delta$, for some $\alpha > 1$. If α is chosen so that $\alpha\delta$ lies within the waterfall region of the code, then the value of α does not have a significant effect on the computed threshold. To make the dependence on α and δ explicit, we denote the threshold by $\sigma^2_*(\alpha, \delta)$.

3.4.4.1 Bit Error Rate

The evolution of $P_e^{\ell}(\sigma^2)$ as a function of ℓ for the $(3,6)$-regular ensemble and for two indicative cases of $\delta = 10^{-5}$ and $\delta = 10^{-6}$ under MS and faulty MS decoding is presented in Fig. 3.15. The error floor for faulty MS decoding is very apparent. This visualization also enables us to calculate the overhead, in terms of additional iterations, introduced by faulty decoding. In Fig. 3.15, the faulty MS decoder for $\delta = 10^{-5}$ requires 64 more iterations than the MS decoder to achieve the same bit error probability when operating slightly below the faulty MS decoder's threshold, which is $\sigma^2_*(10, 10^{-5}) = 0.6576$. Moreover, the faulty MS decoder for $\delta = 10^{-6}$ requires 20 more iterations than the MS decoder to achieve the same bit error probability

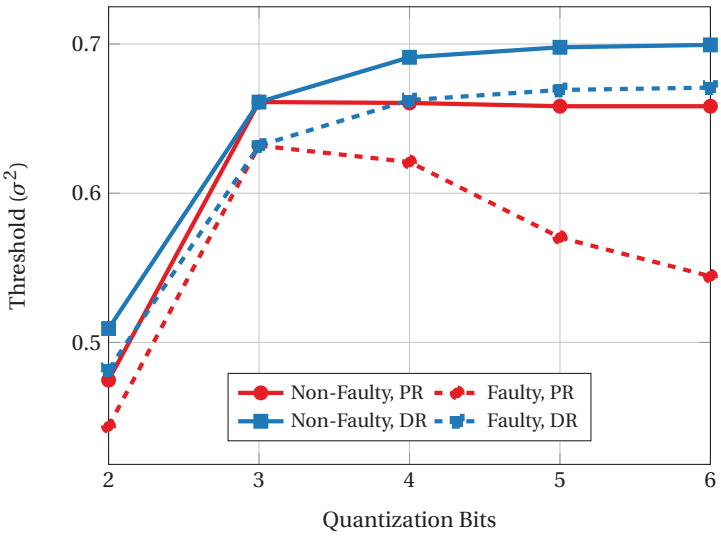


Figure 3.16 – Decoding threshold for a (3,6)-regular LDPC code under faulty MS and MS decoding for $\delta = 10^{-3}$ for different numbers of quantization bits.

Table 3.1 – Thresholds of various (d_v, d_c)-regular codes under MS and faulty MS decoding for $\alpha = 10$ and $b = 5$ bits.

(d_v, d_c)		δ	10^{-3}	10^{-4}	10^{-5}	10^{-6}
(3,6)	MS	$\sigma^2_*(\alpha,\delta)$	0.6579	0.6579	0.6579	0.6582
	F-MS	$\sigma^2_*(\alpha,\delta)$	0.5703	0.6518	0.6576	0.6582
(4,8)	MS	$\sigma^2_*(\alpha,\delta)$	0.5486	0.5486	0.5486	0.5486
	F-MS	$\sigma^2_*(\alpha,\delta)$	0.5077	0.5446	0.5482	0.5486
(5,10)	MS	$\sigma^2_*(\alpha,\delta)$	0.4793	0.4793	0.4793	0.4793
	F-MS	$\sigma^2_*(\alpha,\delta)$	0.4473	0.4761	0.4790	0.4792
(6,12)	MS	$\sigma^2_*(\alpha,\delta)$	0.4320	0.4320	0.4320	0.4320
	F-MS	$\sigma^2_*(\alpha,\delta)$	0.4041	0.4292	0.4317	0.4320

when operating slightly below the faulty MS decoder's threshold, which is $\sigma^2_*(10, 10^{-6}) = 0.6582$. In Fig. 3.15, we see that for a smaller δ, the difference in iterations is smaller, as intuitively expected. Finally, for both $\delta = 10^{-5}$ and $\delta = 10^{-6}$ when operating above the corresponding threshold, the bit-error rate very quickly floors at very high value.

3.4.4.2 Decoding Threshold

In Table 3.1, we present $\sigma^2_*(\alpha,\delta)$ under MS and faulty MS decoding for various (d_v, d_c)-regular ensembles of rate 0.5 and for various values of δ, with $\alpha = 10$ and $b = 5$ bits. The maximum number of iterations is set to $\ell_{\max} = 200$. Quantization is performed with $\Delta = 1$. For fair comparison, the definition in (3.110) was used for both MS and faulty MS decoding.

It is interesting to note that the threshold generally decreases when d_v and d_c are increased, but the resulting code ensembles seem to be more resilient to errors. The loss in $\sigma^2_*(\alpha,\delta)$ as δ is increased is smaller for larger (d_v, d_c) pairs.

3.4.4.3 Are More Quantization Bits Always Better?

In faulty decoding it cannot be claimed in advance that increasing the number of quantization bits b will result in better performance, since by increasing b we also increase the number of faults in the decoder. The additional bits can be used either to increase the dynamic range (DR) or to increase the precision (PR) of the messages. The DR case corresponds to quantization with a fixed step size, while in the PR case the quantization step is a function of b.

In Fig. 3.16 we present indicative threshold results for the DR case with $\Delta_{\text{DR}} = 1$, as used in Section 3.4.4.2, and for the PR case with $\Delta_{\text{PR}} = 2^{3-b}$, which we empirically found to provide good performance with fault-free decoding in both cases, and $\delta = 10^{-3}$ and $\alpha = 10$. We observe that increasing the dynamic range does not offer any benefits after $b = 3$ for fault-free decoding in the examined scenario, and that PR quantization provides better performance than DR quantization. More importantly, however, in the DR case the performance actually degrades for $b \geq 3$. This behavior can be explained intuitively as follows. In the DR case, bit-flips in the additional bits cause increasingly larger errors in the message values, whereas in the PR case these errors become smaller when b is increased.

3.5 Summary

In this chapter we have studied the application of approximate computing concepts to channel coding applications. We have examined applications that are both intentionally approximate and unintentionally faulty.

More specifically, in Section 3.2.1, we have shown how to achieve fine-grained trade-offs between complexity and performance of SC decoding of polar codes by reformulating the frozen channel selection step of the standard polar code construction procedure as a 0-1 knapsack problem. Moreover, we have described a low-complexity greedy algorithm, which is tailored to fit our specific knapsack problem instance. This greedy algorithm was used to approximately solve the optimization problem in order to construct polar codes of blocklength up to $N = 2^{20}$ that provide varying levels of performance-complexity trade-offs.

In Section 3.3, we have studied faulty SC decoding of polar codes for the BEC, where the hardware-induced errors are modeled as additional erasures within the decoder. We have shown that, under this model, fully reliable communication is not possible at any rate. Furthermore, we have shown that, in order for partial ordering of the synthetic channels with respect to the BEC parameter p to hold, the internal erasure probability of the decoder has to be approximately smaller than the erasure probability of the BEC. Moreover, we have derived a lower bound on the frame erasure rate and we used this lower bound in order to optimize the blocklength of polar codes under faulty SC decoding. Finally, we have proposed an error protection scheme which re-enables asymptotically error-free transmission by protecting only a constant fraction of the decoder. Finally, our unequal error protection scheme was shown to significantly improve the performance of the faulty SC decoder for finite-length codes by

protecting as little as 1.5% of the decoder.

In Section 3.4, we have studied MS decoding of LDPC codes under unreliable message storage, where the hardware-induced errors in the memory are modeled as i.i.d. bit-flips. We have derived the DE equations for an MS decoder using our fault model, and we have provided numerical results on the threshold and the BER of a faulty MS decoder with various fault rates. Moreover, we have demonstrated that, in the context of faulty MS decoding, increasing the number of quantization bits only leads to improved performance when the additional bits are used in order to enhance the precision of the decoder and not the dynamic range. We have also observed that the decoding threshold decreases when the variable node and check node degrees are increased, but the resulting LDPC code ensembles seem to be more resilient to errors.

4 Hardware Decoders for Ultra High-Speed Decoding of LDPC Codes

The excellent error-correcting performance of low-density parity-check (LDPC) codes, along with the availability of low-complexity and highly parallel decoding algorithms and corresponding hardware architectures makes them an attractive choice for many high throughput communication systems. LDPC codes are usually decoded using message-passing (MP) schemes such as the sum-product (SP) and the min-sum (MS) algorithms, as explained in Section 1.4. Both of the aforementioned decoding algorithms involve real-valued infinite-precision messages. However, practical implementations require finite-precision message representations in order to keep the implementation complexity at acceptable levels. To this end, the decoder messages are typically uniformly quantized and represented using 4 to 7 bits per message (see, e.g., [35] and references therein). Lower message resolutions tend to deteriorate the error rate performance of the code severely, especially in the error floor regime at high signal-to-noise ratios (SNRs) [138].

Instead of starting from a decoding algorithm and using uniform quantization of a given bit-width, it is also possible to start from a given bit-width and design a decoding algorithm specifically for that bit-width. We call this approach *quantized decoding*. Previous work on quantized MP algorithms for LDPC decoding has shown that decoders which are designed to operate directly on message alphabets of finite size can lead to improved performance. There are numerous different approaches towards the design of such decoders. For example, the authors of [139], [140] and [141] consider look-up table (LUT) based update rules that are designed such that the resulting decoders can correct most of the error events contributing to the error floor. However, their design is restricted to codes with column weight 3 and to binary output channels. In [138] a quasi-uniform quantization was proposed which extends the dynamic range of the messages at later iterations and improves the error floor performance. Unfortunately, the design of [138] still relies on the conventional message update rules and therefore does not reduce the required message bit-width. Furthermore, the authors of [142, 143] consider message updates based on an information theoretic fidelity criterion. While [139], [140], and [138] analyze the performance of their decoding schemes by means of FER simulations, [142] only provides density evolution results and [143] focuses solely on the

algorithm for designing the message update rules. To the best of our knowledge, none of the above schemes have been assessed in terms of their impact on hardware implementations.

In this chapter, we use the idea of operating on messages that are not directly associated with numerical values in order to present a novel min-LUT algorithm that replaces the variable node (VN) update of the MS algorithm with a look-up table (LUT) designed to maximize the local information flow through the code's Tanner graph as in [143]. Moreover, we examine the effects of the design SNR on the error-correcting performance and we develop several complexity reduction techniques, such as using a tree structure for the LUTs, LUT re-use, and alphabet downsizing. We demonstrate the various design and performance trade-offs by means of both density evolution (DE) analysis and frame error rate (FER) simulations. Finally, we design, implement, and synthesize a fully unrolled LDPC decoder based on our LUT design algorithm and compare our results with our re-implementation of the only other existing fully unrolled LDPC decoder architecture [144], which is based on the conventional MS algorithm.

4.1 Mutual Information Based Message Quantization

Since floating-point arithmetic is too complex for most practical hardware implementations of LDPC decoders, the real-valued messages of the MS algorithm are usually discretized using a small number of uniformly spaced quantization levels. Together with the well-established two's complement and sign-magnitude binary encoding, the uniform quantization leads to highly efficient arithmetic circuits. However, this kind of quantization is not necessarily the best choice in terms of the error-correcting performance of the resulting decoder, as several other (non-uniform) quantization options are available.

Recently, efforts have been made to design decoders that explicitly account for finite message and channel log-likelihood ratio (LLR) alphabets [139, 142]. Instead of arithmetic computations such as (1.41) and (1.42), the update rules for these decoders are implemented as LUTs. There are numerous approaches to the design of such LUTs. In the following, we present an algorithm that combines the conventional MS algorithm and the purely LUT-based approach of [142]. In this min-LUT algorithm, the VN updates are realized as LUTs, whereas the CN updates follow the standard MS update rule of (1.42). Our choice is motivated by the following three observations: First, the CN degree can be much larger than the VN degree, especially for high code rates. Consequently, without further simplifications, CN LUTs are far more complex than VN LUTs as the size of a single LUT implementing the node update rule grows exponentially in the number of inputs. Second, for the MS algorithm, the VN update (1.41) typically increases the dynamic range of the messages whereas the CN update (1.42) preserves the dynamic range, so that message saturation is not an issue at CNs. Moreover, replacing the VN update (1.41) with a LUT eliminates the need for a message representation that can be interpreted as a numeric value. However, as will be explained in (4.1.1), the outputs of the LUT-based VN can be sorted in such a way that the CN update (1.41) can be performed based on the LUT output labels, making the actual numerical values of the messages irrelevant

and enabling the use of a conventional MS-based CN. The LUT design for the VN updates is based on the method of [142] and it is based on density evolution. More specifically, as will be explained in the sequel, given the CN-to-VN message distributions of the previous iterations, one can design the VN LUTs for each iteration in a way that maximizes the mutual information between the VN output messages and the codeword bit corresponding to the VN in question.

4.1.1 Channel Model and Symmetry Conditions

As explained in Section 3.4, LDPC codes can be studied analytically using a technique called *density evolution* (DE), which tracks the average probability density functions of the messages exchanged between the variable and check nodes at each decoding iteration in the limit of infinite blocklength [132]. There are two main properties that make the DE procedure tractable. First, the incoming messages at check nodes and variable nodes are assumed to be independent so that their evolution can be tracked more easily. It can be shown that the Tanner graph of a randomly constructed LDPC code is asymptotically cycle-free, so this assumption is in fact true. The second property is that the conventional decoding algorithms, such as SP and MS decoding, have certain symmetry properties which, when paired with a symmetric transmission channel, mean that the error probability of the decoding algorithms is independent of the transmitted codeword. Thus, the DE analysis can be restricted to the all-zero codeword, greatly reducing its complexity. In the remainder of this section, we will formalize these properties.

In order to initialize the DE procedure, the LLR distribution at the decoder input needs to be known. Similarly to Section 3.4, we assume that transmission of each codeword $\mathbf{c} \in \mathcal{C}$ takes place over an additive white Gaussian noise (AWGN) channel using binary phase-shift keying (BPSK) modulation, which can be modeled as

$$y_i = x_i + w_i,\ w_i \sim \mathcal{N}\left(0, \sigma^2\right),\ i = 1, \ldots, N, \tag{4.1}$$

where $x_i = 1 - 2c_i$. We note that, since x_i and c_i are equivalent, they can be used interchangeably. Moreover, for an AWGN channel the quantized channel LLRs are computed as

$$\mathrm{LLR}(y_i) \triangleq q_\Delta\left(\ln \frac{p(y_i|x_i = +1)}{p(y_i|x_i = -1)}\right) = q_\Delta\left(\ln \frac{p(y_i|c_i = 0)}{p(y_i|c_i = 1)}\right) = q_\Delta\left(-\frac{2y_i}{\sigma^2}\right), \tag{4.2}$$

where the uniform symmetric b-bit quantizer q_Δ is defined in Section 1.4.2.3. We note that, while our decoder design is exemplified for the BI-AWGN channel, it applies to any symmetric binary input channel that is followed by a symmetric quantizer. The quantizer q_Δ induces a symmetric PMF $p_{\mathrm{LLR}|X}(l|x)$ that can in turn be used to define the reproducer values of the quantized LLRs as

$$L \triangleq \log \frac{p_{\mathrm{LLR}|X}(l|0)}{p_{\mathrm{LLR}|X}(l|1)}, \tag{4.3}$$

hence $p_{L|X}(-l|0) = p_{L|X}(l|1)$. Similarly, we can assign reproducer values

$$\mu \triangleq \log \frac{p^{(\ell)}_{M|X}(\mu|0)}{p^{(\ell)}_{M|X}(\mu|1)} \tag{4.4}$$

to the output message labels of the VN LUTs at iteration ℓ. We assume that the number $|\mathcal{L}|$ and $|\mathcal{M}|$ of channel and internal messages is even. When the reproducer values are in ascending order, i.e.,

$$L_1 < L_2 < \cdots < L_{|\mathcal{L}|}, \quad \text{and} \quad \mu_1 < \mu_2 < \cdots < \mu_{|\mathcal{M}|}, \tag{4.5}$$

for $L_i \in \mathcal{L},\ i = 1,\ldots,|\mathcal{L}|$, and $\mu_i \in \mathcal{M},\ i = 1,\ldots,|\mathcal{M}|$, the identities

$$L_k \equiv -L_{|\mathcal{L}|-k+1}, \qquad \mu_j \equiv -\mu_{|\mathcal{M}|-j+1}, \tag{4.6}$$

follow from the symmetry of $p_{L|X}(l|x)$ and the MP algorithm (cf. [132], Definition 1) and they associate each label $k \in \{1,\ldots,|\mathcal{L}|\}$ and $j \in \{1,\ldots,|\mathcal{M}|\}$ with a sign. Specifically, $L_k,\ k \in \left\{1,\ldots,\frac{|\mathcal{L}|}{2}\right\}$ and $L_j,\ j \in \left\{1,\ldots,\frac{|\mathcal{M}|}{2}\right\}$ can be said to have a negative sign, while the remaining values have positive signs. Based on this association and the ordering (4.5), the MS CN update (1.42) can be performed directly on the message labels; the reproducer values (4.3)–(4.4) are not needed for decoding. However, (4.4) bears an interesting interpretation: As the messages become more informative over the course of iterations, implying more concentrated densities $p^{(\ell)}_{M|X}(\mu|x)$, the reproducer values grow in magnitude. Using different LUTs for different iterations is thus similar to using different message representations for different iterations, an approach which has already been used successfully in [138].

The symmetry of the MP algorithm discussed above is guaranteed, if the designed VN LUT at any iteration ℓ satisfies

$$\Phi^{(\ell)}_v(-L,-\bar{\mu}_1,\ldots,-\bar{\mu}_{d_v-1}) = -\Phi^{(\ell)}_v(L,\bar{\mu}_1,\ldots,\bar{\mu}_{d_v-1}). \tag{4.7}$$

This identity can be reformulated based on (4.6) as a symmetry relation involving only labels.

4.1.2 LUT Design via Density Evolution

In this section, we show how the VN update rules for a finite-alphabet LDPC decoder can be optimized by tracking the evolution of the message distributions over the course of the decoding iterations for a (d_v, d_c)-regular LDPC code. We first describe how the distribution of the CN-to-VN messages can be computed based on the distribution of the incoming VN-to-CN messages. We then use the joint distribution of the CN-to-VN messages in order to design a locally optimal VN update rule. Using this rule, the distribution of the VN-to-CN messages can be computed based on the distribution of the incoming CN-to-VN messages. We note that the derivation of the VN-to-CN messages and the CN-to-VN messages is similar to the

derivation of Section 3.4. However, there is one important difference: in order to design the VN LUTs we need to calculated the joint distribution of the VN *input messages*, whereas in Section 3.4 it sufficed to directly calculated the distribution of the output messages. For this reason, the notation in this section is slightly more involved that the notation of Section 3.4.

4.1.2.1 CN-to-VN Messages

Let $\boldsymbol{\mu} = \left(\mu_1, \ldots, \mu_{d_c-1}\right)$ denote the $(d_c - 1)$ incident VN-to-CN messages that are involved in the update of a certain CN k. We wish to calculate the distribution of the CN-to-VN message from CN k to a particular VN n, which is associated with a codeword bit c_n. If the Tanner graph is cycle-free, then the individual input messages μ_j of the CN at iteration ℓ are i.i.d. conditioned on their corresponding transmitted bit c_j, and their distributions are denoted by $p^{(\ell)}_{M|X}(\mu_j|c_j)$. The joint distribution of the $(d_c - 1)$ incoming messages at CN k conditioned on the transmitted bit value c_n corresponding to the recipient VN n (cf. Figure 1.8) thus reads

$$p^{(\ell)}_{\boldsymbol{M}|C}(\boldsymbol{\mu}|c_n) = \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{c}:\bigoplus_{j=1}^{d_c-1} c_j = c_n} \left(\prod_{j=1}^{d_c-1} p^{(\ell)}_{M|C}(\mu_j|c_j) \right). \tag{4.8}$$

Using the MS update rule (1.42), the distribution of the outgoing CN-to-VN message is then given by

$$Cp^{(\ell)}_{\overline{M}|C}(\bar{\mu}|c_n) = \sum_{\boldsymbol{\mu}:\,\mathrm{sign}(\boldsymbol{\mu})\min|\boldsymbol{\mu}|=\bar{\mu}} p^{(\ell)}_{\boldsymbol{M}|C}(\boldsymbol{\mu}|c_n), \tag{4.9}$$

which was already calculated in detail in (3.100) of Section 3.4.

4.1.2.2 VN-to-CN Messages

Let $\bar{\boldsymbol{\mu}} = \left(\bar{\mu}_1, \ldots, \bar{\mu}_{d_v-1}\right)$ denote the $(d_v - 1)$ incident CN-to-VN messages that are involved in the update of a certain VN n, which is in turn associated with a codeword bit c_n. Then, the joint distribution of the VN input messages and the channel LLR is given by

$$p^{(\ell)}_{L,\overline{\boldsymbol{M}}|C}(l, \bar{\boldsymbol{\mu}}|c_n) = p_{L|C}(l|c_n) \sum_{\boldsymbol{c}:\, c_1 = \cdots = c_{d_v-1} = c_n} \left(\prod_{j=1}^{d_v-1} p^{(\ell)}_{\overline{M}|X}(\bar{\mu}_j|c_j) \right). \tag{4.10}$$

Given this joint input distribution, we want to construct an update rule $\Phi^{(\ell)}_v$ that maximizes the mutual information $I^{(\ell)}\big(\Phi(L, \overline{\boldsymbol{M}}); C\big) \triangleq I(M^{(\ell)}; C)$, i.e.,

$$\Phi^{(\ell)}_v = \arg\max_{\Phi} I^{(\ell)}\big(\Phi(\mathsf{L}, \overline{\boldsymbol{M}}); C\big), \tag{4.11}$$

where $M^{(\ell)}$ is a random variable describing any one the variable node output messages (note that all d_v output messages have the same distribution) and the maximization is performed over all deterministic mappings Φ in the form of (1.35) that respect the symmetry condition

(4.7). Hence, the resulting update rule $\Phi_v^{(\ell)}$ maximizes the local information flow between the VN and the CN. We use a dynamic programming algorithm that solves (4.11) with complexity $O\big(|\mathcal{M}|^{3d_v}\big)$ which was provided in [143]. Using the update rule (4.11), we can compute the conditional distribution of the VN-to-CN message in the next iteration

$$p_{M|C}^{(\ell+1)}(\mu_n|c_n) = \sum_{(l,\bar{\boldsymbol{\mu}}):\Phi_v^{(\ell)}(l,\bar{\boldsymbol{\mu}})=\mu_n} p_{\mathsf{L},\overline{\boldsymbol{M}}|C}^{(\ell)}(l,\bar{\boldsymbol{\mu}}|c_n). \tag{4.12}$$

Using the distribution $p_{M|C}^{(\ell+1)}(\mu_n|c_n)$, we can in turn calculate the distribution of the CN-to-VN messages at iteration $(\ell+1)$ and design the VN update rule $\Phi_v^{(\ell+1)}$. This procedure is repeated until the update rules for all $\ell_{\max}$ iterations are designed.

4.2 LUT Design Considerations for Practical Decoders

In the previous section, we have described our proposed LUT design method on a relatively abstract level. However, there are several issues that need to be addressed from a practical point of view in order to enable the implementation of efficient hardware LDPC decoders using the proposed min-LUT algorithm. In particular, one significant issue with LUT-based decoders is that the circuit complexity of LUTs can be significantly higher than that of conventional arithmetic circuits, such as adders and comparators, especially when many inputs are involved. Thus, in this section we focus mainly on methods for complexity reduction of the LUT implementation. The various complexity-performance trade-offs are illustrated in this section by Monte Carlo simulations for the frame-error rate (FER). We use the parity-check matrix of the LDPC code defined in the IEEE 802.3an standard [30], which is a $(6,32)$-regular LDPC code of design rate $R = \frac{13}{16}$ and blocklength $N = 2048$.

4.2.1 Performance of Min-LUT Decoding

In this section, we first briefly demonstrate the error-correcting performance that can be achieved using our proposed min-LUT decoder, skipping some of the practical details on how the examined min-LUT decoder was actually designed, as these details are thoroughly treated in the following sections. To this end, in Figure 4.1 we compare the error-correcting performance of both floating point and fixed point versions of the standard MS decoding algorithm with the error-correcting performance of our proposed min-LUT decoder when performing a maximum of $\ell_{\max} = 5$ decoding iterations. We observe that the fixed point MS decoder needs a message alphabet size of at least $|\mathcal{M}| = 2^5$ in order to closely match the performance of the floating point MS decoder.[1] Moreover, we can see that the min-LUT decoder with channel LLR alphabet size $|\mathcal{L}| = 2^4$ and message alphabet size $|\mathcal{M}| = 2^3$ outperforms even the floating point version of the MS decoder. Finally, we observe the min-LUT decoder with channel LLR alphabet size $|\mathcal{L}| = 2^4$ and message alphabet size $|\mathcal{M}| = 2^3$ is

[1] We note that this should not be unexpected, since the MS algorithm is sub-optimal while the min-LUT decoding algorithm is highly optimized.

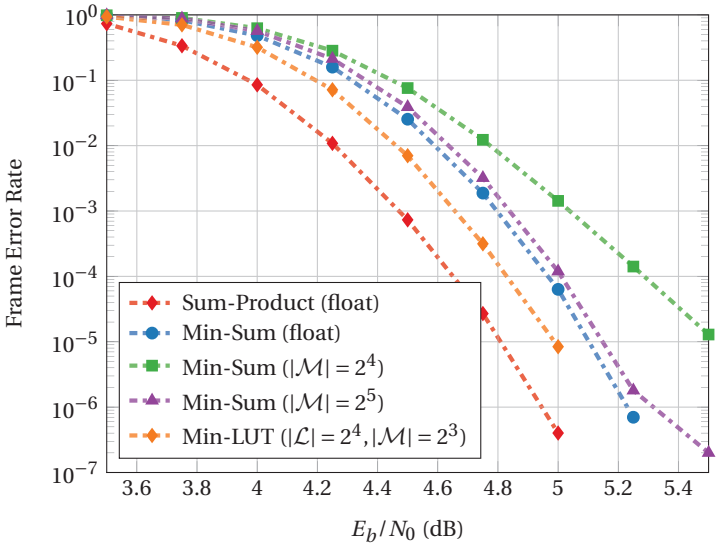


Figure 4.1 – Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code ($\ell_{\max} = 5$).

only 0.2 dB worse than the floating point SP decoder.

4.2.2 Reduced Complexity LUT Structure

Since the number of input configurations for the VN update $\Phi_v^{(\ell)}$ equals $|\mathcal{M}^{(\ell)}|^{d_v}$, a single-stage LUT would be prohibitively complex for codes with even moderate VN degree d_v. A similar problem occurs with the decision LUT that implements (1.37). To overcome this limitation, we propose to use nested (i.e., multi-level LUT) update rules. For example, for $d_v = 6$ a possible LUT decomposition could take the form

$$\Phi(L, \bar{\mu}_1, \ldots, \bar{\mu}_5) = \Phi_1\big(L, \Phi_2(\bar{\mu}_1, \bar{\mu}_2, \bar{\mu}_3), \Phi_3(\bar{\mu}_4, \bar{\mu}_5)\big). \tag{4.13}$$

Any such nesting can be represented graphically by a tree, e.g., tree T_2 in Figure 4.2 for the example of (4.13). The LUT design procedure for this tree starts by designing the LUTs for the leaves and progressively moves towards the root of the tree. Since we assume i.i.d. messages, the ordering of the arguments in the nesting is irrelevant for the mutual information and we consider nestings that differ only in the ordering as equivalent. While the nested structure clearly reduces complexity, it is not clear a priori which tree structure to prefer over another. In what follows, we provide guidelines on how to choose the tree structure based on information-theoretic arguments as well as on a heuristic metric.

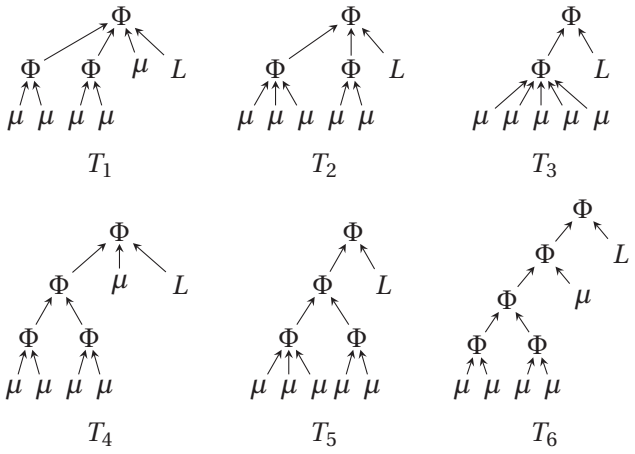


Figure 4.2 – Six different LUT tree structures. Note that $T_1 \geq_{\mathcal{T}} T_4 \geq_{\mathcal{T}} T_6$, $T_2 \geq_{\mathcal{T}} T_5$, $T_3 \geq_{\mathcal{T}} T_5$, and $T_3 \geq_{\mathcal{T}} T_6$. However, we cannot compare T_2 with T_3 or T_5 with T_6 using the relation $\geq_{\mathcal{T}}$.

Table 4.1 – Comparison of cumulative depth and DE threshold for various tree structures (cf. Figure 4.2).

T	T_1	T_2	T_3	T_4	T_5	T_6
λ	10	11	11	14	16	19
σ^*	0.5330	0.5328	0.5327	0.5313	0.5309	0.5305

A Partial Ordering

Let the tree T_1 represent a specific nesting and let T_2 be a refinement of T_1. Graphically, a refinement of nesting corresponds to the placement of new nodes between parent and child nodes. Furthermore, let $\mathcal{Q}_j$ denote the set of all LUTs that respect the nesting induced by some tree T_j. By construction, any LUT in $\mathcal{Q}_2$ also conforms with the nesting associated with T_1. Thus, $\mathcal{Q}_1 \supseteq \mathcal{Q}_2$ and

$$\max_{\Phi \in \mathcal{Q}_1} I^{(\ell)}\big(\Phi(L, \overline{M}); C\big) \geq \max_{\Phi \in \mathcal{Q}_2} I^{(\ell)}\big(\Phi(L, \overline{M}); C\big).$$

Consequently, tree refinement defines a partial ordering $\geq_{\mathcal{T}}$, effectively inducing a hierarchy in terms of maximum information flow. Clearly, not all tree structures can be compared in terms of the relation $\geq_{\mathcal{T}}$. An example of various LUT trees and their corresponding ordering with respect to $\geq_{\mathcal{T}}$ is given in Figure 4.2.

A Heuristic Metric

The data processing inequality states that processing can only reduce mutual information. Intuitively, for maximum information flow the paths from the input leaves to the root output should be as short as possible. We thus define the cumulative depth $\lambda(T)$ of a tree T as the sum

of distances of all leaf nodes to the root node. DE simulations confirmed that cumulative depth is in our case useful for ranking tree structures. More specifically, Table 4.1 shows how a larger λ corresponds with a lower DE threshold (cf. Section 4.2.5.1 on how the decoding threshold can be calculated). However, the threshold differences are small and our simulations have shown that all the trees presented here perform similarly in terms of error rate. While there were small differences conforming with the ordering discussed above, they are not significant enough to serve as a basis for choosing the tree structure. Rather, we recommend choosing the tree based on its silicon complexity. Trees that are close to full binary trees are preferable because they have short critical paths with low complexity LUTs and at the same time have small cumulative depth λ.

Position of the Channel LLR in the VN Tree

The mutual information between the CN-to-VN messages and the coded bits is initially zero and increases over the course of iterations until at some iteration $I^{(\ell')}(\overline{M};C) \geq I(L;C)$. Using a similar argument as before, we can conclude that until iteration ℓ' the channel LLR should be placed close to the root node to ensure a large information flow. After iteration ℓ', the CN-to-VN messages tend to carry more information than the channel LLR an thus should be gradually placed closer to the root node. Our simulations indeed showed that this strategy provides the best FER performance. However, the loss as compared to the case where the channel LLR stays at the root node is only relevant for a large number of iterations (e.g., $\ell_{\max} > 20$), meaning that, in order to simplify the design procedure by reducing the number of possible parameters, we recommend to place the channel LLR at the root of the tree for all iterations.

4.2.3 Design SNR

In order to design the VN LUTs with our proposed method, the channel LLR distribution $p_{L|C}(l|c)$ needs to be known as a starting point for the DE process. In general, the channel LLR distribution usually depends on some channel parameter, such as the noise variance σ^2 (equivalently, the SNR) for AWGN channels. Thus, a decoder that is designed for a specific *design* SNR γ may not necessarily work well for other SNRs.

To illustrate this behavior, we plot the FER performance of the LDPC code used in the IEEE 802.3an standard for various design SNRs γ in Figure 4.3. The examined decoder performs $\ell_{\max} = 5$ decoding iterations using a channel LLR alphabet of size $|\mathcal{L}| = 2^4$ and a message alphabet of size $|\mathcal{M}| = 2^3$. We observe that, by increasing the design SNR, we can trade off performance in the waterfall region against performance in the error floor region. The explanation is straight-forward, as we intuitively expect, decoders that are designed for bad channels to work better for bad channels (i.e., low SNR) and decoders that are designed for good channels to work better for good channels (i.e., high SNR). More specifically, when designing a decoder for a low SNR, at each decoding iteration the message densities computed

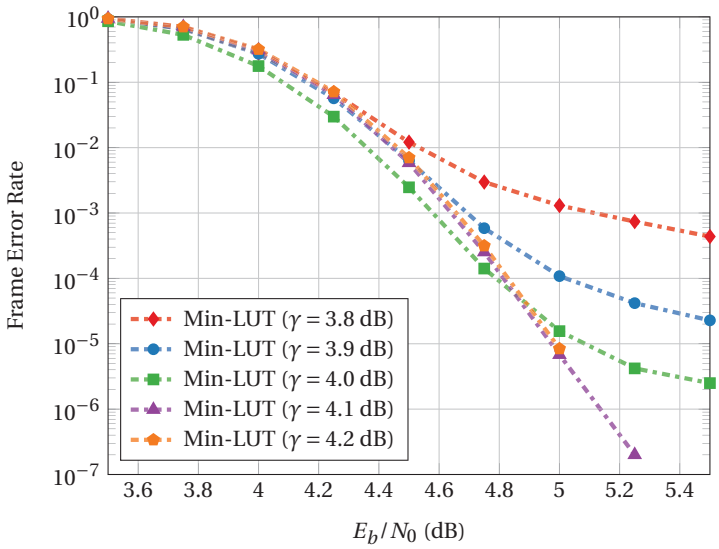


Figure 4.3 – FER versus channel SNR for min-LUT decoder at different design SNRs γ for the IEEE 802.3an LDPC code ($\ell_{\max} = 5$, $|\mathcal{L}| = 4$, $|\mathcal{M}| = 3$).

by DE are concentrated around smaller values than when designing a decoder for a higher SNR. Thus, when operating a decoder that is designed for a low SNR at a much higher SNR, significant message saturation occurs which can lead to the pronounced error floors observed in Figure 4.3. Our simulations also indicate that, if the design SNR γ is chosen carefully, then excellent error-correcting performance is maintained over a wide range of actual SNR values. Re-designing the LLR quantizer or the entire decoder for different SNRs would of course improve the error-correcting performance, but it would also substantially increase the decoder hardware implementation cost as many more distinct LUTs would need to be implemented.

4.2.4 LUT Re-use

The method described in Section 4.1.2 produces a distinct VN LUT for each iteration. While this does not affect silicon complexity for a decoder in which the individual iterations are unrolled (cf. Section 4.3), standard non-unrolled decoders would need to implement multiple distinct LUTs in the hardware that computes the message updates for all iterations. In order to reduce the hardware complexity, it would be desirable to re-use the LUTs for a particular iteration in order to also compute the message updates for a few of the following iterations. Since the message distributions usually do not change drastically from one iteration to the next, this approach is actually possible. Let us define a *re-use pattern* $\boldsymbol{r}$ as follows

$$\boldsymbol{r} = \left[r^{(1)} \quad r^{(2)} \quad \dots \quad r^{(\ell_{\max})} \right], \tag{4.14}$$

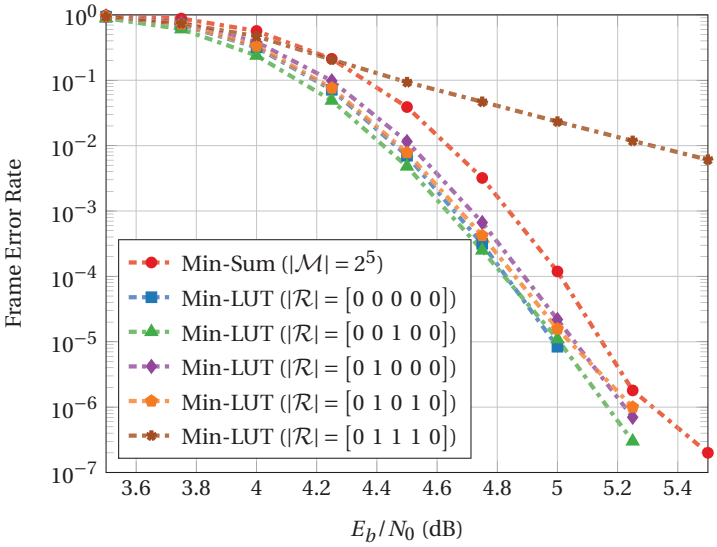


Figure 4.4 – Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code and various LUT re-use patterns $\boldsymbol{r}$ ($\ell_{\max} = 5, |\mathcal{L}| = 2^4, |\mathcal{M}| = 2^3, \gamma = 4.2$ dB).

where $r^{(\ell)} = 1$ if at iteration ℓ the VN LUT from iteration $(\ell - 1)$ is re-used, and $r^{(\ell)} = 0$ if a new VN LUT is to be designed for iteration ℓ. Clearly, $r^{(1)} = 0$ must always hold as there is no VN LUT from iteration $\ell = 0$ to be re-used at iteration $\ell = 1$. Moreover, $r^{(\ell_{\max})} = 1$ is only permissible in conjunction with LUT downsizing (cf. Section 4.2.5), as the output of the last VN LUT has to be binary by definition of the decision rule (1.37).

In Figure 4.4 we compare the error-correcting performance of both floating point and fixed point versions of the standard MS decoding algorithm with the error-correcting performance of our proposed min-LUT decoder when performing a maximum of $\ell_{\max} = 5$ decoding iterations and using various LUT re-use patterns $\boldsymbol{r}$. We observe that re-using one LUT (i.e., patterns $\boldsymbol{r} = \left[0\ 0\ 1\ 0\ 0\right]$ and $\boldsymbol{r} = \left[0\ 1\ 0\ 0\ 0\right]$ where four distinct LUTs are used) does not degrade the error-correcting performance of the min-LUT decoder significantly. In fact, for some SNR values the performance with LUT re-use can even be slightly better than the performance without LUT re-use. The explanation of this effect is an open question, but we conjecture that it originates from the overly optimistic message distributions of DE, which tends to overestimate the speed of convergence with respect to finite-length codes that are not cycle-free. Even when employing a LUT re-use pattern that leads to three distinct LUTs being used (i.e., $|\boldsymbol{r}| = \left[0\ 1\ 0\ 1\ 0\right]$), the error-correcting performance degradation of the resulting min-LUT decoder is negligible. However, we observe that, when moving to the most extreme re-use pattern that is admissible for $\ell = 5$, i.e., $\boldsymbol{r} = \left[0\ 1\ 1\ 1\ 0\right]$ where only two distinct LUTs are used, the error-correcting performance degradation is detrimental.

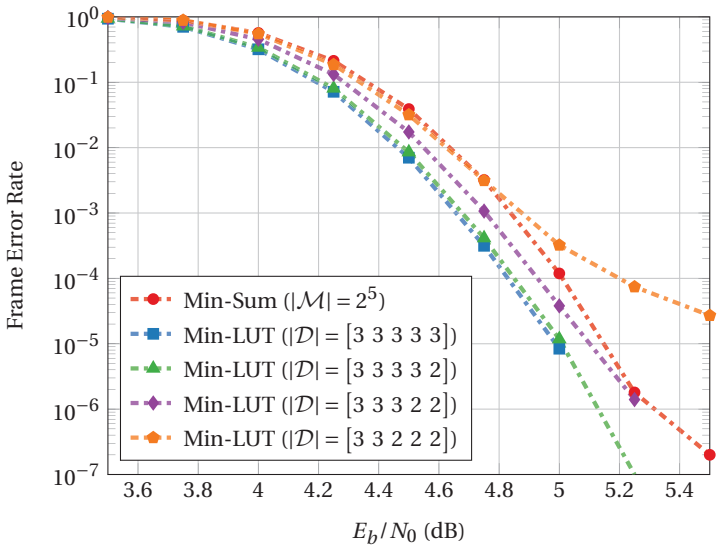


Figure 4.5 – Performance comparison of floating point and fixed point min-sum decoding with our proposed min-LUT decoder for the IEEE 802.3an LDPC code and various LUT downsizing patterns $\boldsymbol{d}$ ($\ell_{\max} = 5, |\mathcal{L}| = 2^4, \gamma = 4.2$ dB).

4.2.5 LUT Input/Output Alphabet Downsizing

As already mentioned in Section 1.4.2, in general the size of the message alphabets can change over the course of the decoding iterations. Thus, another means of reducing LUT complexity is LUT alphabet downsizing, i.e., $|\mathcal{M}^{(\ell')}| \leq |\mathcal{M}^{(\ell)}|$ for $\ell' > \ell$. The idea here is that the messages undergo a gradual hardening while being passed through the decoder before culminating into the binary-output decision mapping (1.37). Let us define a *downsizing pattern* $\boldsymbol{d}$ as follows

$$\boldsymbol{d} = \left[d^{(1)} \quad d^{(2)} \quad \dots \quad d^{(\ell_{\max})} \right], \tag{4.15}$$

where $d^{(\ell)} = \log\left|\mathcal{M}^{(\ell)}\right|$. For the VN LUTs of iteration ℓ, the bit-width of the input messages is $d^{(\ell)}$, while the bit-width of the output messages is $d^{(\ell+1)}$ to ensure compatibility with the VN LUT of iteration $(\ell+1)$. Finally, the VN LUT of iteration $\ell_{\max}$, also called a decision node (DN) LUT, has an output bit-width of one bit by definition of the decision mapping (1.37).

In Figure 4.5 we compare the error-correcting performance of both floating point and fixed point versions of the standard MS decoding algorithm with the error-correcting performance of our proposed min-LUT decoder when performing a maximum of $\ell_{\max} = 5$ decoding iterations and using various LUT downsizing patterns $\boldsymbol{d}$. We observe that downsizing the LUT alphabet size only during the last iteration from $|\mathcal{M}^{(4)}| = 2^3$ to $|\mathcal{M}^{(4)}| = 2^2$, i.e., $\boldsymbol{d} = \left[3\ 3\ 3\ 3\ 2\right]$, has practically no impact on the error-correcting performance of the resulting min-LUT decoder. When $\boldsymbol{d} = \left[3\ 3\ 3\ 2\ 2\right]$, on the other hand, the degradation in the error-correcting performance is more visible, although the resulting min-LUT decoder still performs similarly to the floating point MS decoder. Finally, when more aggressive downscaling is performed, as in the case

of the downscaling pattern $\boldsymbol{d} = \left[3\ 3\ 2\ 2\ 2\right]$, the error-correcting performance degradation becomes very apparent and an error floor starts to appear at a FER of 10^{-3}.

We note that LUT re-use and LUT alphabet downsizing can be combined, but not in an arbitrary fashion as, e.g., reducing the message resolution at a certain iteration ℓ prevents re-use of the LUT of iteration $(\ell-1)$. More formally, we call a pair of LUT re-use and LUT downsizing patterns *compatible* if for any $\ell \in \{2,\dots,\ell_{\max}-1\}$ such that $d^{(\ell-1)} \neq d^{(\ell)}$ or $d^{(\ell)} \neq d^{(\ell+1)}$, we have $r^{(\ell)} = 0$. Finally, $r^{(\ell_{\max})} = 1$ is only permissible if $d^{(\ell_{\max}-1)} = d^{(\ell_{\max})} = 1$, as the last VN is always a special DN with binary output.

4.2.5.1 Decoding Threshold and Decoder Design Procedure

In the previous sections, we have demonstrated the performance of the min-LUT decoder and we have examined the effect of various design parameter choices on the error-correcting performance of the resulting min-LUT decoder. In this section, we describe the LUT design procedure in more detail. To this end, let us first define the noise threshold σ^* of a (d_v, d_c)-regular LDPC code ensemble with at most $\ell_{\max}$ decoding iterations as

$$\sigma^* = \sup\left\{\sigma \geq 0\colon\ I(M^{(\ell)}; C) > 1-\epsilon \text{ for some } \ell \leq \ell_{\max}\right\}, \tag{4.16}$$

where $\epsilon > 0$ is pre-defined tolerance parameter.

Algorithm 6 summarizes the individual steps of a bisection algorithm that uses DE to design the VN LUTs for each decoding iteration and to calculate the corresponding decoding threshold σ^*. The following parameters need to be defined in order to run Algorithm 6.

1. The desired number of maximum iterations $\ell_{\max}$ and the VN and CN degrees (d_v, d_c)
2. The LUT downsizing pattern $\boldsymbol{d}$ (cf. Section 4.2.5) and a compatible LUT re-use pattern $\boldsymbol{r}$ (cf. Section 4.2.4).
3. The LUT tree structure for the variable nodes and the decision nodes (cf. Section 4.2.2).
4. The search interval $[\sigma_{\min}, \sigma_{\max}]$, the desired accuracy $\Delta\sigma > 0$ for the bisection search procedure, and the tolerance parameter ϵ for the threshold calculation (cf. (4.16)).

Algorithm 6 produces the threshold σ^* of the considered (d_v, d_c)-regular LDPC code under the designed min-LUT decoding algorithm, as well as the sequence of VN LUTs that lead to this decoding threshold. If one is only interested in designing a VN LUT sequence for a particular design SNR γ (equivalently, for a particular σ^2), it suffices to run lines (3)-(14) of Algorithm 6 for the particular value of σ^2.

Algorithm 6: Density Evolution based LUT design

Data: Search interval $\ell_{\max}$, (d_v, d_c), $\boldsymbol{d}$, $\boldsymbol{r}$, $[\sigma_{\min}, \sigma_{\max}]$, $\Delta\sigma$, ϵ

while $\sigma_{\max} - \sigma_{\min} > \Delta\sigma$ **do**
  $\sigma \leftarrow (\sigma_{\max} - \sigma_{\min})/2$;
  Get $p_{L|C}(l|c)$ corresponding to BI-AWGN(σ^2);
  achievable $\leftarrow$ false;
  for $\ell = 1, \ldots, \ell_{max}$ **do**
    Update CN-to-VN distribution (4.8)–(4.9);
    Build the product distribution (4.10);
    if $\boldsymbol{r}_\ell == 0$ **then**
      Design LUT update $\Phi_v^{(\ell)}$ (4.11) with input bit-width $\boldsymbol{d}_\ell$ and output bit-width $\boldsymbol{d}_{\ell+1}$;
    else
      Re-use LUT from iteration $(\ell - 1)$, i.e., $\Phi_v^{(\ell)} = \Phi_v^{(\ell-1)}$;
    Update VN-to-CN distribution (4.12);
    if $I(M^{(\ell)}; C) > 1 - \epsilon$ **then**
      achievable $\leftarrow$ true;
  if *achievable* **then**
    $\sigma_{\min} \leftarrow \sigma$
  else
    $\sigma_{\max} \leftarrow \sigma$
$\sigma^* \leftarrow \sigma$;

Result: Threshold σ^*, LUT sequence $\Phi_v^{(1)}, \ldots, \Phi_v^{(\ell)}$

4.3 LUT-Based Fully Unrolled Decoder Hardware Architecture

In the previous sections, we have described an algorithm that can construct locally optimal VN update rules in the form of LUTs for a given message bit-width for each iteration for any given (d_v, d_c)-regular LDPC code. In this section, we describe an LDPC decoder hardware architecture that takes full advantage of the min-LUT decoding algorithm in order to significantly increase the hardware efficiency and the throughput of the decoder with respect to similar existing LDPC decoders.

Most LDPC decoder architectures are either partially parallel, meaning that fewer than N VNs and M CNs are instantiated, or fully parallel, meaning that N VNs and M CNs are instantiated and re-used several times in order to perform all $\ell_{\max}$ decoding iterations. Using a LUT-based decoder with a carefully designed quantization scheme can significantly reduce the memory required to store the messages exchanged by the VNs and CNs in both of these LDPC decoder architecture types, mainly due to the reduced message bit-width required to achieve the same FER performance with a conventional quantized MS decoder. However, both for partially parallel and for fully parallel decoders, separate LUTs would be required within each VN for each one of the performed decoding iterations, significantly increasing the area requirements of each VN, and thus possibly outweighing the gain in the memory area.

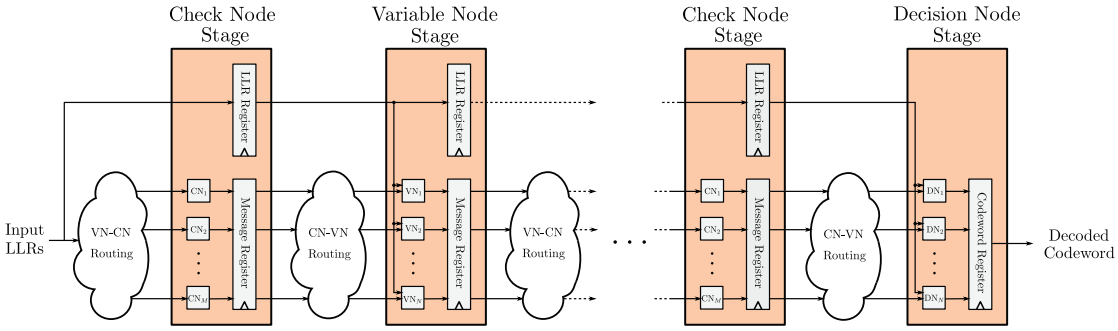


Figure 4.6 – Top level decoder architecture processing pipeline. The channel LLRs are the input of the left-hand side and the decoded codeword is obtained as the output of the right-hand side.

An additional degree of parallelism was recently explored in [144], where a *fully unrolled* LDPC decoder was presented. This decoder instantiates N VNs and M CNs for each and every iteration of the decoding algorithm, leading to a total of $N\ell_{\max}$ instantiated VNs and $M\ell_{\max}$ instantiated CNs. While such a fully unrolled decoder requires significant hardware resources, it also has a very high throughput since it is possible to output one decoded codeword in each clock cycle and the clock period can be made arbitrarily small with appropriate pipelining. Thus, the hardware efficiency (i.e., throughput per unit area) of the fully unrolled decoder presented in [144] turns out to be significantly better than the hardware efficiency of partially parallel and fully parallel (non-unrolled) approaches. Since in a fully unrolled LDPC decoder architecture dedicated VNs and CNs are instantiated for each iteration, it is a very suitable candidate for the application of our LUT-based decoding algorithm, where distinct VN LUTs are required for each decoding iteration. In this section, we describe the hardware architecture of our proposed fully unrolled LUT-based LDPC decoder. This hardware architecture is similar to the architecture used in [144]. However, the most important difference are the optimized LUT-based variable node and the significantly reduced bit-width of all quantities involved in the decoding process that lead to reduced memory and routing requirements, while maintaining similar error-correcting performance.

4.3.1 Decoder Architecture

An overview of our decoder architecture is shown in Figure 4.6. Each decoding iteration $\ell \in \{1,\ldots,\ell_{\max}\}$ is mapped to a distinct set of N VNS and M CNs, which then form a processing pipeline. In essence, a fully unrolled LDPC decoder is a systolic array in which data flows from left to right. A new set of N channel LLRs can be read in each clock cycle, and a new decoded codeword is output in each clock cycle. The decoding latency as well as the maximum frequency depend on the number of performed iterations as well as the number of pipeline registers present in the decoder. Our decoder consist of three types of stages, namely the CN stage, the VN stage, and the DN stage, which are described in detail in the following sections. As long as a steady flow of input channel LLRs can be provided to the decoder, there is no

control logic required apart from the clock and reset signals.

4.3.1.1 Check Node Stage

Each CN stage contains M check node units, as well as Md_c $d^{(\ell)}$-bit registers which store the CN output messages, where $d^{(\ell)}$ denotes the number of bits used to represent the CN output messages at iteration ℓ (cf. Section 4.2.5). Moreover, each CN stage contains N $d^{(0)}$-bit channel LLR registers which are used to forward the channel LLRs required by the following variable node stages, where $d^{(0)}$ denotes the number of bits used to represent the channel LLRs.

As already discussed in Section 4.1.1, due to (4.6), the CN update can be performed directly on the message labels instead of the message values. If we use the natural numbering $\{0,\ldots,2^{d^{(\ell)}}-1\}$ for the message labels and a sign-magnitude binary representation, we can use a check node architecture which is practically identical to the check node architecture used in [144] if we consider the MSB to be the sign bit. In this case, when the MSB is 0 the number is negative, while when the MSB is 1 the number is positive (cf. Section 4.1.1). More specifically, each check node consists of a sorting unit that identifies the two smallest messages among the absolute values of 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 in order to efficiently implement the CN update rule (1.42). The sorting unit contains 4-input compare-and-select (CS) units in a tree structure, which identify and output the two smallest values out of the four input values [144].

4.3.1.2 Variable Node Stage

The VN stage for iteration ℓ contains N variable node units, as well as Nd_v $d^{(\ell)}$-bit registers that store the variable node output messages. Moreover, each VN stage contains N $d^{(0)}$-bit channel LLR registers which are used to forward the channel LLRs to the following CN stage, so that they can then be used by the VN stage for iteration $(\ell+1)$.

In the reference variable node architecture used in the adder-based decoder of [144], all input messages are added up using an adder tree and then the input message corresponding to each output is subtracted from the sum in order to form the output message, thus efficiently implementing the conventional MS update rule given in (1.41). In order to avoid overflows, in our implementation of [144] the bit-width of the internal signals is increased by one bit for each addition. The final result is then saturated so that the output has the same bit-width as the input.

For our LUT-based decoder the adder tree is replaced by d_v LUT trees, each of which computes one of the d_v outputs of the variable node. One possible LUT-tree structure for a code with $d_v = 6$ is shown in Figure 4.7a, where μ denotes an internal message from a check node and L denotes the channel LLR. Keeping the number of inputs of each LUT as low as possible

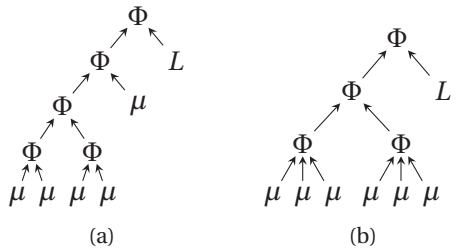


Figure 4.7 – (a) The variable node LUT tree that is used in the hardware implementation for the calculation of one output of a variable node of degree $d_v = 6$. This tree is identical to T_6 of Figure 4.2. Each LUT-based variable node contains d_v such LUT trees, one for each combination of $(d_v - 1)$ input messages.
(b) The decision node LUT tree that is used in the hardware implementation for the hard decisions taken by each variable node of degree $d_v = 6$. This tree is similar to T_5 of Figure 4.2 with an additional input added to the right LUT of the lowest level. Each LUT-based decision node contains a single decision tree.

ensures that the size of the LUTs, which grows exponentially with the number of inputs, is manageable for the automated logic synthesis process.

4.3.1.3 Decision Node Stage

The VN that corresponds to the final decoding iteration is called a *decision node* (DN). The DN stage contains N decision nodes, as well N single-bit registers that store the decoded codeword bits. The DN stage does not contain channel LLR registers, as there are no subsequent decoding stages where the channel LLRs would be used. The architecture of a decision node is generally simpler than that of a variable node, as a single output value (i.e., the decoded bit) is calculated instead of d_v distinct outputs.

More specifically, in the reference architecture of [144], the decision metric of (1.37) is already calculated as part of the variable node update rule. However, for the decision node, there is no need to subtract each input message from the sum in order to generate d_v distinct output messages. It suffices to check whether the sum is positive or negative, and output the corresponding decoded codeword bit.

In our LUT-based decoder a LUT tree is designed whose tree node has an output bit-width of a single bit, which is the corresponding decoded codeword bit. An example of a decision LUT tree for a decision node that corresponds to a code with $d_v = 6$ is shown in Figure 4.7b. Each decision node contains a single LUT tree, in contrast with the variable nodes which contain d_v LUT trees.

4.3.2 Decoding Latency and Throughput

Our LUT-based architecture contains pipeline registers at the output of each stage (VN, CN, and DN). Thus, for a given maximum number of decoding iterations $\ell_{\max}$, the decoding latency is $2\ell_{\max}$ clock cycles. Since one decoded codeword is output in each clock cycle, the decoding throughput of the decoder, measured in Gbits/s, is given by

$$T = Nf, \tag{4.17}$$

where f denotes the operating frequency of the decoder measured in GHz. The operating frequency of the decoder is limited by the combinational path with the highest delay

4.3.3 Memory Requirements

Each pipeline stage except the DN stage requires $Nd^{(0)}$ channel LLR registers. Since there are $(2\ell-1)$ stages when excluding the DN stage, the total number of registers required to store and forward the channel LLRs is $(2\ell-1)Nd^{(0)}$. Moreover, the VN and CN stages for iteration ℓ require $Nd_v d^{(\ell)}$ and $Md_c d^{(\ell)}$ registers to store their output messages, respectively. Finally, the DN stage requires N registers to store the decoded codeword bits. Thus, the total number of register bits required by our LUT-based decoder can be calculated as

$$B_{\text{tot}} = (2\ell_{\max}-1)Nd^{(0)} + \left(\sum_{\ell=1}^{\ell_{\max}-1}\left(Nd_v d^{(\ell)} + Md_c d^{(\ell)}\right)\right) + \left(Md_c d^{(\ell_{\max})} + N\right). \tag{4.18}$$

In the case where $d^{(1)} = d^{(2)} = \ldots = d^{(\ell_{\max})} = d$ and taking into account the fact that $Nd_v d^{(\ell)} = Md_c d^{(\ell)}$, (4.18) can be simplified to

$$B_{\text{tot}} = (2\ell_{\max}-1)N(d_v d + d^{(0)}) + N. \tag{4.19}$$

Naturally, both (4.18) and (4.19) can also be used to calculate the register bits required by an adder-based MS architecture with the same pipeline register structure.

4.4 Implementation Results

In this section, we present synthesis results for a fully unrolled LUT-based LDPC decoder and we compare them with synthesis results of our implementation of a fully unrolled adder-based MS LDPC decoder. As for the simulations of Section 4.2, we have used the parity-check matrix of the LDPC code defined in the IEEE 802.3an standard [30], which is a $(6,32)$-regular LDPC code of design rate $R = \frac{13}{16}$ and blocklength $N = 2048$. For the adder-based MS decoder and the LUT-based decoder, a total of $\ell_{\max} = 5$ decoding iterations are performed, as we can observe from Figure 4.8 that no significant gain is achieved when increasing the number of iterations to, e.g., $\ell_{\max} = 10$. All synthesis results are obtained by using a TSMC 90 nm CMOS library under typical operating conditions (1 V supply voltage, 25°C operating temperature).

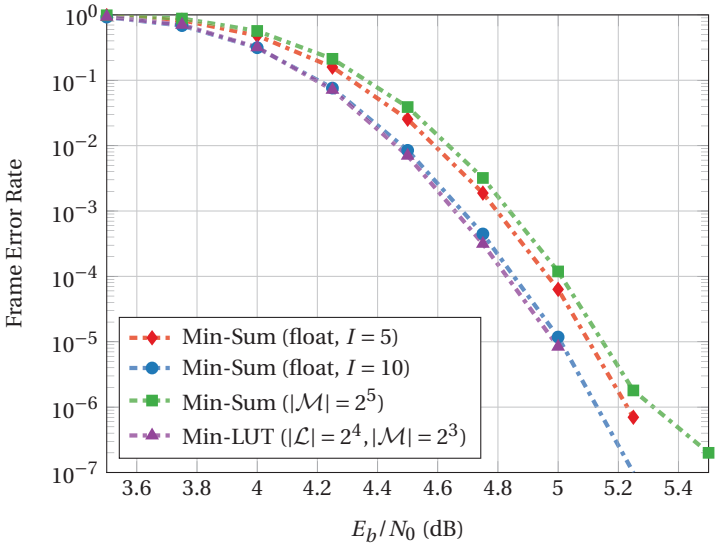


Figure 4.8 – FER vs E_b/N_0 for the $N = 2048$ $(6,32)$-regular LDPC code defined in IEEE 802.3an under various decoding algorithms.

Table 4.2 – Synthesis Results for the Adder-based and the LUT-based Decoders

	Adder-based MS	LUT-based
Logic Area	35.63 mm^2	33.79 mm^2
Operating Frequency	495 MHz	813 MHz
Decoding Latency	20.20 ns	12.30 ns
Coded Throughput	1014 Gbps	1665 Gbps
Area Efficiency	28.46 Gbps/mm^2	49.27 Gbps/mm^2

4.4.1 Quantization Parameters

For the LUT-based decoder, we have used $d^{(0)} = 4$ bits for the representation of the channel LLRs and $d_{(1)} = d_{(2)} = \ldots = d_{(\ell_{\max})} = 3$ bits for the representation of the internal messages, as this leads to an error correction performance that is very close the floating-point MS decoder (cf. Figure 4.8). Both decoders perform $\ell_{\max} = 5$ decoding iterations. For the variable nodes, we use the LUT tree structure of Figure 4.7a and for the decision nodes we use the LUT tree structure of Figure 4.7b. The design SNR is set to 4.2 dB. For the adder-based MS decoder which serves as a reference, we use $d^{(0)} = 5$ bits for the representation of the channel LLRs and $d_{(1)} = d_{(2)} = \ldots = d_{(\ell_{\max})} = 5$ bits for the representation of the internal messages, as this leads to practically the same FER performance for the LUT-based and the adder-based MS decoder, as can be seen in Figure 4.8.

4.4.2 Adder-based vs. LUT-based Decoder

We present synthesis results for the adder-based and the LUT-based decoders in Table 4.2. For fair comparison, we synthesized both designs for various clock constraints and selected the

Table 4.3 – Area Breakdown

	Adder-based MS	LUT-based
	Check Node Stage	
Check Nodes	2.77 mm^2	1.11 mm^2
Pipeline Registers	1.11 mm^2	0.70 mm^2
Total	3.88 mm^2	1.81 mm^2
	Variable Node Stage	
Variable Nodes ℓ_1	2.35 mm^2	4.62 mm^2
Variable Nodes ℓ_2	2.35 mm^2	4.78 mm^2
Variable Nodes ℓ_3	2.35 mm^2	4.64 mm^2
Variable Nodes ℓ_4	2.35 mm^2	4.68 mm^2
Pipeline Registers	1.11 mm^2	0.57 mm^2
Total ℓ_1	3.46 mm^2	5.32 mm^2
Total ℓ_2	3.46 mm^2	5.48 mm^2
Total ℓ_3	3.46 mm^2	5.34 mm^2
Total ℓ_4	3.46 mm^2	5.38 mm^2
	Decision Node Stage	
Decision Nodes	2.35 mm^2	3.21 mm^2
Pipeline Registers	0.03 mm^2	0.03 mm^2
Total	2.38 mm^2	3.24 mm^2
	Top-Level Decoder	
Logic Area	25.58 mm^2	27.46 mm^2
Register Area	10.05 mm^2	6.33 mm^2
Total Area	35.63 mm^2	33.79 mm^2

result with the highest hardware efficiency (Gbps/mm^2) for each design. These results should not be regarded in absolute terms, as the placement and routing of such a large design is highly non-trivial and will increase the area and the delay of both designs significantly. However, it is safe to make relative comparisons, especially when considering the fact that the LUT-based decoder will be easier to place and route due to the fact that it requires approximately 40% fewer wires for the interconnect between the VN, CN, and DN stages. We observe that the LUT-based decoder is approximately 8% smaller as well as 64% faster than the adder-based MS decoder. As a result, the area efficiency of the LUT-based decoder is 73% higher than that of the adder-based MS decoder. For both designs, the critical path goes through the CN, but in the LUT-based decoder the delay is smaller due to the reduced bit-width.

We show the area breakdown of the LUT-based and the adder-based decoders in Table 4.3. We observe that the VN stage area of the LUT-based decoder varies significantly over the iterations, even though the LUT tree structures are identical. This is not unexpected, since the contents of the LUTs are different for different iterations and the resulting logic circuits can have very different complexities. We also see that the VN stage of the LUT-based decoder is larger than the VN stage of the adder-based decoder. Moreover, we observe that the CN stage of the LUT-based decoder is approximately 53% smaller than the CN stage of the adder-based

decoder due to the bit-width reduction enabled by the optimized LUT design. The reduction in the CN stage is larger than the increase in the VN stage, leading to an overall reduction in area for our proposed LUT-based decoder. From Table 4.3 we can see that this reduction stems mainly from the reduced number of required registers, as the area occupied by the logic of each decoder is similar.

4.5 Summary

In this chapter, we have described a method that can be applied to design a discrete message-passing decoder for LDPC codes by replacing the standard VN update rules with locally optimal LUT-based update rules which are designed using an information-theoretic criterion. Moreover, we have examined the effect of various LUT design parameters, such as the design SNR and the LUT tree structure, on the error-correcting performance of the resulting min-LUT decoder. Using the IEEE 802.3an LDPC code, we have demonstrated that the min-LUT error-correcting performance can be superior to that of standard MS decoding even with 40% smaller message resolutions. Moreover, we have presented a hardware architecture for a LUT-based fully unrolled LDPC decoder which can reduce the area and increase the operating frequency compared to a conventional adder-based unrolled MS decoder by 8% and 64%, respectively, due to the significantly reduced bit-width required to achieve identical error correction performance. Finally, the LUT-based decoder requires approximately 40% fewer wires, which simplifies the routing step, which is a known problem in fully parallel architectures.

5 Conclusion & Outlook

In this thesis, we have investigated some topics that form an interplay between abstract information theoretic concepts and hardware implementation. More specifically, in Chapter 2 we have examined the hardware implementation of decoders for polar codes, where we have shown that a high-level algorithmic transformation leads to significant improvements and optimization opportunities in the hardware implementation of a successive cancellation list decoder for polar codes. Moreover, in Chapter 3 we have analyzed the performance of channel decoders under various approximate computing scenarios by using various tools from information and coding theory, such as density evolution. Finally, in Chapter 4 we have implemented an ultra high-speed fully unrolled decoder for LDPC codes by using an optimized message quantization scheme that maximizes the transfer of information between the variable nodes and the check nodes of the LDPC code.

In the remainder of this concluding chapter, 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: Hardware Decoders for Polar Codes

Path metric sorting is an important aspect of SCL decoding, especially when considering polar codes with relatively short blocklength (e.g., $N \leq 256$) and large list sizes (e.g., $L \geq 16$) for use in low-latency and/or low-power and low-rate applications, as the sorting step can dominate the overall complexity of the decoder. Even though we used the properties of the LLR-based path metric to simplify various sorters in Section 2.2, it was recently shown in [78] that further simplifications are in fact possible. It remains an important open problem to fully optimize the path metric sorting step of SCL decoding.

As can be seen from the comparison of Section 2.4.2, SCL decoders cannot yet match the high throughput numbers reported for SC and BP decoders. This is partly due to the fact that fast-SSC decoding [40] has not yet been fully applied to SCL decoding. Since our LLR-based SCL decoder uses L SC decoders, it seems evident that any architectural and algorithmic

improvements made to the SC decoder itself will be beneficial to the LLR-based SCL decoder as well. However, the family of fast-SSC decoders is not applicable verbatim to the LLR-based SCL decoder. This happens because, in order to keep the path metric updated, we need to calculate the LLRs even for the frozen bits. An important step in this direction was recently made in [61], but the hardware implementation of a fast-SSC based SCL decoder is an essential next step. In particular, the direction of increasing the throughput of SCL decoders seems promising, since SCL decoders have the lowest area requirements and generally the best hardware efficiency out of the polar decoders in all iso-FER comparisons of Section 2.4.1.4.

It may also be interesting to identify different operating regimes such as where polar codes may be able to compete better with existing channel coding solutions. Short blocklengths are of particular interest for low-latency communications, and both Turbo and LDPC codes are known to exhibit poor performance at short blocklengths, both in the waterfall and in the the error floor region.

Chapter 3: Faulty Polar and LDPC Channel Decoders

It would be beneficial to extend the modified polar code construction of Section 3.2 to other decoding algorithms that further simplify SC decoding, such as fast-SSC [40]. An attempt in this direction was recently made in [67], where a human-guided exhaustive search approach was used in order to use a modified polar code in conjunction with the fast-SSC decoding algorithm. A more systematic solution of this problem that is similar to the approach of Section 3.2 remains an interesting open problem.

Another interesting open problem is to study the faulty SC decoder of Section 3.3 over more general channels, such as the AWGN channel. In order to achieve this, the lower bound of [124] would have to be extended to more general channels, which is a very challenging open problem in itself that has with wide applications. Moreover, a suitable error model, such as the one we used in Section 3.4, would have to be introduced and analyzed.

In Section 3.4 we saw that the decoding threshold for a (d_v, d_c)-regular LDPC code ensemble decreases when the variable node and check node degrees are increased, but the resulting LDPC code ensembles seem to be more resilient to errors. This interesting trade-off between threshold and error resilience motivates the design of irregular LDPC codes that are tailored to faulty MS decoding, which is an open problem worth pursuing.

Moreover, we note that, while the fault models that we have used in Section 3.3 an Section 3.4 are quite accurate for faults of a transient nature, like radiation-induced faults, they are most likely not the best way to model other kinds of failures. For example, manufacturing defects often lead to “stuck-at” faults, where the value of a particular bit is always either 0 or 1. These errors can be converted to random bit-flips easily by randomly flipping the value that is stored in the faulty bit-cell and then flipping it again when it is read. However, their nature is not generally transient, meaning that a faulty bit-cell will always be faulty. It is possible to

randomize the physical write addresses in order to make the faults appear transient, but this may incur significant hardware overhead for the address randomization. The introduction of more comprehensive and realistic fault models is an important next step in the general field of approximate computing. A particularly interesting problem in this direction is to investigate whether it is better to use more sophisticated models in order to describe the operation of faulty hardware or to slightly modify the faulty hardware itself (e.g., by using address randomization as described previously) in order to make it behave more closely to simple fault models like, for example, the random bit-flip models used in this thesis, which are more manageable from an analytical perspective.

Chapter 4: Hardware Decoders for Ultra High-Speed Decoding of LDPC Codes

In Chapter 4 we have only considered regular LDPC codes and it would be beneficial to extend the LUT design method to irregular LDPC codes, which are known to generally exhibit better performance than regular LDPC codes [34]. Moreover, it would be useful to examine the performance of the min-LUT decoding algorithm for LDPC codes that are shorter than the code used in the IEEE 802.3an standard, as the LUT design method assumes message independence and this assumption is usually violated more strongly in short LDPC codes than in longer LDPC codes. While the unrolled LDPC decoder presented in this chapter is a very good match for the constraints of the min-LUT decoding algorithm, 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 architectures which could also benefit from the reduced quantization bit-width offered by our min-LUT decoding algorithm. Finally, in Chapter 4 we have only presented synthesis results for the designed fully unrolled LDPC decoder. We note that the physical design of such a large architecture, and particularly the placement and routing step, is also a highly non-trivial and important problem if such a decoder is to be implemented in silicon.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[2] T. Richardson and R. Urbanke, *Modern Coding Theory*. New York, NY, USA: Cambridge University Press, 2008.

[3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[4] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[5] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar 1997.

[6] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inform. Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[7] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Trans. Circuits Syst. II*, vol. 61, no. 8, pp. 609–613, May 2014.

[8] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 3903–3907.

[9] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Trans. Signal Processing*, vol. 63, no. 19, pp. 5165–5179, Oct 2015.

[10] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "On metric sorting for successive cancellation list decoding of polar codes," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1993–1996.

[11] A. Balatsoukas-Stimming, G. Karakonstantis, and A. Burg, "Enabling complexity-performance trade-offs for successive cancellation decoding of polar codes," in *IEEE International Symposium on Information Theory*, June 2014, pp. 2977–2981.

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

[13] A. Balatsoukas-Stimming and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," in *International Symposium on Information Theory and its Applications (ISITA)*, Oct 2014, pp. 448–452.

[14] A. Balatsoukas-Stimming and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," *IEEE Trans. Inform. Theory*, 2016 (under review).

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

[16] M. Meidlinger, A. Balatsoukas-Stimming, A. Burg, and G. Matz, "Quantized message passing for LDPC codes," in *2015 49th Asilomar Conference on Signals, Systems and Computers*, Nov 2015, pp. 1606–1610.

[17] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inform. Theory*, vol. 59, no. 10, Oct. 2013.

[18] R. Pedarsani, S. H. Hassani, I. Tal, and E. Telatar, "On the construction of polar codes," in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2011, pp. 11–15.

[19] I. Tal, "On the construction of polar codes for channels with moderate input alphabet sizes," in *IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 1297–1301.

[20] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1665–1668.

[21] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inform. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[22] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, ser. North-Holland Mathematical Library. North-Holland, 1978.

[23] "IEEE standard for air interface for broadband wireless access systems," *IEEE Std 802.16TM-2012*, Aug. 2012.

[24] J. Hagenauer, "The Turbo principle: Tutorial introduction and state of the art," in *International Symposium on Turbo Codes*, Sep. 1997, pp. 1–11.

[25] U. U. Fayyaz and J. R. Barry, "Low-complexity soft-output decoding of polar codes," *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 958–966, May 2014.

[26] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, Oct. 2012.

[27] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Asilomar Conference on Signals, Systems and Computers*, Nov. 2014, pp. 2116–2120.

[28] K. Niu, K. Chen, and J. Lin, "Low-complexity sphere decoding of polar codes based on optimum path metric," *IEEE Commun. Lett.*, vol. 18, no. 2, pp. 332–335, Feb. 2014.

[29] S. A. Hashemi, C. Condo, and W. J. Gross, "List sphere decoding of polar codes," in *Asilomar Conference on Signals, Systems and Computers*, Nov. 2015, pp. 1346–1350.

[30] "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-2006 (Amendment to IEEE Std 802.3-2005)*, pp. 1–167, 2006.

[31] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, 2009.

[32] "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, 2014.

[33] "Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)," *ETSI EN 302 307 V1.2.1 (2009-08)*, pp. 1–78, 2009.

[34] T. J. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb 2001.

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

[36] A. J. Raymond and W. J. Gross, "Scalable successive-cancellation hardware decoder for polar codes," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2013, pp. 1282–1285.

[37] A. Pamuk and E. Arıkan, "A two phase successive cancellation decoder architecture for polar codes," in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2013, pp. 957–961.

[38] C. Leroux, A. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Processing*, vol. 61, no. 2, pp. 289–299, Jan. 2013.

[39] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. Signal Processing*, vol. 61, no. 10, pp. 2429–2441, Mar. 2013.

[40] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.

[41] Y. Fan and C.-Y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Trans. Signal Processing*, vol. 62, no. 12, pp. 3165–3179, Jun. 2014.

[42] A. Mishra, A. J. Raymond, L. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS," in *IEEE Asian Solid State CircuitsConference (A-SSCC)*, Nov. 2012, pp. 205–208.

[43] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Oct. 2011.

[44] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *IEEE International Conference on Communications (ICC)*, Jun. 2012, pp. 3471–3475.

[45] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725–728, Apr. 2013.

[46] C. Zhang and K. K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Trans. Circuits Syst. II*, vol. 61, no. 2, pp. 115–119, Feb. 2014.

[47] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2044–2047, Dec. 2012.

[48] K. Chen, K. Niu, and J. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Trans. Commun.*, vol. 61, no. 8, pp. 3100–3107, Aug. 2013.

[49] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE International Conference on Communications*, vol. 2, Jun. 1995, pp. 1009–1013.

[50] L. G. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *IEEE Trans. VLSI Syst.*, vol. 20, no. 12, pp. 2342–2346, Dec. 2012.

[51] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. VLSI Syst.*, vol. 23, no. 10, pp. 2268–2280, 2015.

[52] J. Lin and Z. Yan, "An efficient list decoder architecture for polar codes," *IEEE Trans. VLSI Syst.*, vol. 23, no. 11, pp. 2508–2518, Nov. 2015.

[53] J. Lin and Z. Yan, "Efficient list decoder architecture for polar codes," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 1022–1025.

[54] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conf.*, vol. 32, 1968, pp. 307–314.

[55] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*, 2nd ed. MIT Press, 2001.

[56] C. Zhang, X. You, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 209–212.

[57] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2014, pp. 1–6.

[58] Wikipedia. (2016, Jul.) Polynomial representations of cyclic redundancy checks — wikipedia, the free encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Polynomial_representations_of_cyclic_redundancy_checks

[59] "Technical Specification Group Radio Access Network; E-UTRA; Multiplexing and Channel Coding (Release 10) 3GPP, TS36.212, Rev. 10.0.0," *3GPP*, 2011.

[60] A. Pamuk, "An FPGA implementation architecture for decoding of polar codes," in *International Symposium on Wireless Communication Systems (ISWCS)*, Nov. 2011, pp. 437–441.

[61] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, "Fast list decoders for polar codes," *IEEE J. Select. Areas Commun.*, vol. 34, no. 2, pp. 318–328, Feb. 2016.

[62] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Journal of Signal Processing Systems*, vol. 69, no. 3, pp. 305–315, Jul. 2012.

[63] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.

[64] J. Lin, C. Xiong, and Z. Yan, "Reduced complexity belief propagation decoders for polar codes," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2015, pp. 1–6.

[65] T. Che, J. Xu, and G. Choi, "TC: Throughput centric successive cancellation decoder hardware implementation for polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 991–995.

[66] O. Dizdar and E. Arıkan, "A high-throughput energy-efficient implementation of successive-cancellation decoder for polar codes using combinational logic," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 3, pp. 436–447, Mar. 2016.

[67] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast low-complexity decoders for low-rate polar codes," *arXiv:1603.05273 [cs, math]*, Mar. 2016, arXiv: 1603.05273. [Online]. Available: http://arxiv.org/abs/1603.05273

[68] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Multi-mode unrolled architectures for polar decoders," *arXiv:1505.01459 [cs]*, May 2016, arXiv: 1505.01459. [Online]. Available: http://arxiv.org/abs/1505.01459

[69] J. Lin, J. Sha, L. Li, C. Xiong, Z. Yan, and Z. Wang, "A high throughput belief propagation decoder architecture for polar codes," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 153–156.

[70] P. Giard, G. Sarkis, C. Thibeault, and W. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.

[71] G. Berhault, C. Leroux, C. Jego, and D. Dallet, "Hardware implementation of a soft cancellation decoder for polar codes," in *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Sep. 2015, pp. 1–8.

[72] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *IEEE Symposium on VLSI Circuits*, Jun. 2014, pp. 1–2.

[73] B. Yuan and K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Trans. Signal Processing*, vol. 62, no. 24, pp. 6496–6506, Dec. 2014.

[74] S. M. Abbas, Y. Fan, J. Chen, and C.-Y. Tsui, "Low complexity belief propagation polar code decoder," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2015, pp. 1–6.

[75] S. Sun and Z. Zhang, "Architecture and optimization of high-throughput belief propagation decoding of polar codes," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 165–168.

[76] C. Zhang, X. You, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 209–212.

[77] B. Yuan and K. K. Parhi, "Successive cancellation list polar decoder using log-likelihood ratios," in *Asilomar Conference on Signals, Systems and Computers*, Nov. 2014, pp. 548–552.

[78] B. Kong, H. Yoo, and I. C. Park, "Efficient sorting architecture for successive cancellation list decoding of polar codes," *IEEE Trans. Circuits Syst. II*, vol. 63, no. 7, pp. 673–677, 2016.

[79] Y. Fan, J. Chen, C. Xia, C.-Y. Tsui, J. Jin, H. Shen, and B. Li, "Low-latency list decoding of polar codes with double thresholding," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015, pp. 1042–1046.

[80] J. Lin, C. Xiong, and Z. Yan, "A high throughput list decoder architecture for polar codes," *IEEE Trans. VLSI Syst.*, vol. 24, no. 6, pp. 2378–2391, Jun. 2016.

[81] B. Yuan and K. K. Parhi, "Reduced-latency LLR-based SC list decoder for polar codes," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2015, pp. 107–110.

[82] B. Yuan and K. K. Parhi, "LLR-based successive-cancellation list decoder for polar codes with multi-bit decision," *IEEE Trans. Circuits Syst. II*, vol. PP, no. 99, pp. 1–1, 2016.

[83] C. Xiong, J. Lin, and Z. Yan, "Symbol-decision successive cancellation list decoder for polar codes," *IEEE Trans. Signal Processing*, vol. 64, no. 3, pp. 675–687, Feb. 2016.

[84] S. A. Hashemi, A. Balatsoukas-Stimming, P. Giard, C. Thibeault, and W. J. Gross, "Partitioned successive-cancellation list decoding of polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2016, pp. 957–960.

[85] C. Xiong, J. Lin, and Z. Yan, "A multimode area-efficient SCL polar decoder," *IEEE Trans. VLSI Syst.*, vol. PP, no. 99, May 2016.

[86] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974.

[87] H. Shirani-Mehr, T. Mohsenin, and B. Baas, "A reduced routing network architecture for partial parallel LDPC decoders," in *Asilomar Conference on Signals, Systems and Computers*, Nov. 2011, pp. 2192–2196.

[88] M. Weiner, B. Nikolic, and Z. Zhang, "LDPC decoder architecture for high-data rate personal-area networks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2011, pp. 1784–1787.

[89] S.-W. Yen, S.-Y. Hung, C.-L. Chen, H.-C. Chang, S.-J. Jou, and C.-Y. Lee, "A 5.79-Gb/s energy-efficient multirate LDPC codec chip for IEEE 802.15.3c applications," *IEEE J. Solid-State Circuits*, vol. 47, no. 9, pp. 2246–2257, Sep. 2012.

[90] S. Ajaz and H. Lee, "Reduced-complexity local switch based multi-mode QC-LDPC decoder architecture for Gbit wireless communication," *Electronics Letters*, vol. 49, no. 19, pp. 1246–1248, Sep. 2013.

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

[92] M. Li, F. Naessens, P. Debacker, P. Raghavan, C. Desset, M. Li, A. Dejonghe, and L. Van der Perre, "An area and energy efficient half-row-paralleled layer LDPC decoder for the 802.11ad standard," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2013, pp. 112–117.

[93] M. Li, F. Naessens, M. Li, P. Debacker, C. Desset, P. Raghavan, A. Dejonghe, and L. Van der Perre, "A processor based multi-standard low-power LDPC engine for multi-Gbps wireless communication," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2013, pp. 1254–1257.

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

[95] S. Ajaz and H. Lee, "Multi-Gb/s multi-mode LDPC decoder architecture for IEEE 802.11ad standard," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Nov. 2014, pp. 153–156.

[96] M. Weiner, M. Blagojevic, S. Skotnikov, A. Burg, P. Flatresse, and B. Nikolic, "A scalable 1.5-to-6Gb/s 6.2-to-38.1mW LDPC decoder for 60GHz wireless networks in 28nm UTBB FDSOI," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2014, pp. 464–465.

[97] M. Li, Y. Lee, Y. Huang, and L. Van der Perre, "Area and energy efficient 802.11ad LDPC decoding processor," *Electronics Letters*, vol. 51, no. 4, pp. 339–341, 2015.

[98] M. Li, J. W. Weijers, V. Derudder, I. Vos, M. Rykunov, S. Dupont, P. Debacker, A. Dewilde, Y. Huang, L. V. d. Perre, and W. V. Thillo, "An energy efficient 18Gbps LDPC decoding processor for 802.11ad in 28nm CMOS," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Nov. 2015, pp. 1–5.

[99] K. Gunnam, G. Choi, W. Wang, and M. Yeary, "Multi-rate layered decoder architecture for block LDPC codes of the IEEE 802.11n wireless standard," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2007, pp. 1645–1648.

[100] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes," in *IFIP International Conference on Very Large Scale Integration*, Oct. 2007, pp. 236–241.

[101] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A scalable decoder architecture for IEEE 802.11n LDPC codes," in *IEEE Global Telecommunications Conference (GLOBECOM)*, Nov. 2007, pp. 3270–3274.

[102] Y. Sun, J. R. Cavallaro, and T. Ly, "Scalable and low power LDPC decoder design using high level algorithmic synthesis," in *IEEE International SOC Conference (SOCC)*, Sep. 2009, pp. 267–270.

[103] J. Jin and C. y. Tsui, "An energy efficient layered decoding architecture for LDPC decoder," *IEEE Trans. VLSI Syst.*, vol. 18, no. 8, pp. 1185–1195, Aug. 2010.

[104] 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 Conference (A-SSCC)*, Nov. 2010, pp. 1–4.

[105] Y. Sun, G. Wang, and J. R. Cavallaro, "Multi-layer parallel decoding algorithm and VLSI architecture for quasi-cyclic LDPC codes," in *IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 1776–1779.

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

[107] 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 Conference (A-SSCC)*, Nov. 2010, pp. 1–4.

[108] Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, Apr. 2010.

[109] D. Bao, X. Chen, Y. Huang, C. Wu, Y. Chen, and X. Y. Zeng, "A single-routing layered LDPC decoder for 10GBASE-T Ethernet in 130nm CMOS," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2012, pp. 565–566.

[110] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 8–17, Jan. 2011.

[111] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE advanced base station applications," in *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Aug. 2012, pp. 21–25.

[112] X. Chen, Y. Chen, Y. Li, Y. Huang, and X. Zeng, "A 691 Mbps 1.392mm^2 configurable radix-16 turbo decoder ASIC for 3GPP-LTE and WiMAX systems in 65nm CMOS," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Nov. 2013, pp. 157–160.

[113] C.-Y. Lin, C.-C. Wong, and H.-C. Chang, "A 40 nm 535 Mbps multiple code-rate turbo decoder chip using reciprocal dual trellis," *IEEE J. Solid-State Circuits*, vol. 48, no. 11, pp. 2662–2670, Nov. 2013.

[114] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser, and Q. Huang, "A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS," in *Symposium on VLSI Circuits (VLSIC)*, Jun. 2013, pp. 284–285.

[115] R. Shrestha and R. Paily, "High-throughput turbo decoder with parallel architecture for LTE wireless communication standards," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 9, pp. 2699–2710, Sep. 2014.

[116] G. Wang, H. Shen, Y. Sun, J. Cavallaro, A. Vosoughi, and Y. Guo, "Parallel interleaver design for a high throughput HSPA+/LTE multi-standard turbo decoder," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 5, pp. 1376–1389, May 2014.

[117] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[118] R. Gonzalez, B. Gordon, and M. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE J. Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, Aug. 1997.

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

[120] Y. Emre and C. Chakrabarti, "Energy and quality-aware multimedia signal processing," *IEEE Trans. Multimedia*, vol. 15, no. 7, pp. 1579–1593, Nov. 2013.

[121] A. Alamdar-Yazdi and F. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.

[122] Z. Huang, C. Diao, and M. Chen, "Latency reduced method for modified successive cancellation decoding of polar codes," *Electronics Letters*, vol. 48, no. 23, pp. 1505–1506, Nov 2012.

[123] L. Zhang, Z. Zhang, X. Wang, C. Zhong, and L. Ping, "Simplified successive-cancellation decoding using information set reselection for polar codes with arbitrary blocklength," *IET Communications*, vol. 9, no. 11, pp. 1380–1387, Jul. 2015.

[124] M. Bastani Parizi and E. Telatar, "On the correlation between polarized BECs," in *IEEE International Symposium on Informtion Theory (ISIT)*, Jul. 2013, pp. 784–788.

[125] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, May 2004.

[126] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[127] O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, "Impact of parameter variations on circuits and microarchitecture," *IEEE Micro*, vol. 26, no. 6, pp. 30–39, Nov. 2006.

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

[129] A. Bhavnagarwala, S. Kosonocky, C. Radens, K. Stawiasz, R. Mann, Q. Ye, and K. Chin, "Fluctuation limits & scaling opportunities for CMOS SRAM cells," in *IEEE International Electron Devices Meeting*, Dec. 2005, pp. 659–662.

[130] S. H. Hassani and R. Urbanke, "Polar codes: robustness of the successive cancellation decoder with respect to quantization," in *IEEE International Symposium on Information Theory*, July 2012, pp. 1–6.

[131] C. Zhang and K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. Signal Processing*, vol. 61, no. 10, pp. 2429–2441, May 2013.

[132] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599–618, Feb 2001.

[133] L. R. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Trans. Inform. Theory*, vol. 57, no. 7, pp. 4427–4444, Jul. 2011.

[134] S. M. Tabatabaei Yazdi, H. Cho, and L. Dolecek, "Gallager B decoder on noisy hardware," *IEEE Trans. Commun.*, vol. 61, no. 5, pp. 1660–1672, May 2013.

[135] F. Leduc-Primeau and W. J. Gross, "Faulty Gallager-B decoding with optimal message repetition," in *Allerton Conference on Communication, Control, and Computing*, Oct. 2012, pp. 549–556.

[136] C. H. Huang and L. Dolecek, "Analysis of finite-alphabet iterative decoders under processing errors," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013, pp. 5085–5089.

[137] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University, Linköping, Sweden, 1996.

[138] X. Zhang and P. Siegel, "Quantized iterative message passing decoders with low error floor for LDPC codes," *IEEE Trans. Commun.*, vol. 62, no. 1, pp. 1–14, Jan. 2014.

[139] S. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders—Part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

[140] D. Declercq, B. Vasic, S. Planjery, and E. Li, "Finite alphabet iterative decoders – Part II: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4046–4057, October 2013.

[141] F. Cai, X. Zhang, D. Declercq, S. Planjery, and B. Vasić, "Finite alphabet iterative decoders for LDPC codes: Optimization, architecture and analysis," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 5, pp. 1366–1375, May 2014.

[142] B. Kurkoski, K. Yamaguchi, and K. Kobayashi, "Noise thresholds for discrete LDPC decoding mappings," in *IEEE Global Telecommunications Conference (GLOBECOM)*, Nov. 2008.

[143] B. Kurkoski and H. Yagi, "Quantization of binary-input discrete memoryless channels," *IEEE Trans. Inform. Theory*, vol. 60, no. 8, pp. 4544–4552, Aug. 2014.

[144] P. Schläfer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2013, pp. 153–158.

Alexios Balatsoukas-Stimming

Telecommunications Circuits Laboratory
EPFL-STI-IEL-TCL
Office ELG 011, Station 11
1015 Lausanne, Switzerland
http://tcl.epfl.ch

Citizenship: Greek, German
Date of Birth: May 4th, 1986
Phone: +41 767 39 68 20
E-mail: alexios.balatsoukas@epfl.ch
Website: http://people.epfl.ch/alexios.balatsoukas

Research Interests

Coding theory and practice, hardware for signal processing and communications, full-duplex communications.

Education

2016 **PhD**, Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

2012 **MSc**, Electronic and Computer Engineering, Technical University of Crete (TUC), Greece.

2010 **Diploma**, Electronic and Computer Engineering, Technical University of Crete, Greece.

Research Experience

Jul 2012–today **Research Assistant, Telecommunications Circuits Lab, EPFL, Switzerland.**
- Conceived algorithms and VLSI architectures for decoding of polar codes.
- Designed a Tbps LDPC decoder via optimized quantization.
- Analyzed decoding of error-correcting codes with faulty hardware.
- Measured and mitigated RF and baseband impairments on a full-duplex MIMO testbed.
- Authored 26 peer-reviewed publications with 162 citations to date (Google Scholar profile).

Tools: MATLAB, C, VHDL, ModelSim, Design Compiler, perl, LabVIEW, LaTeX.

Jun 2015–Aug 2015 **Graduate Technical Intern, Intel Labs, Hillsboro, USA.**
- Worked on large-scale MIMO detection for next-generation 5G systems.
- Filed one U.S. patent and authored one conference publication.
- Recognized by manager for making and meeting commitments, fostering innovation and creative thinking, as well as flawless execution.

Tools: MATLAB, Verilog, ModelSim, Design Compiler.

Mar 2010–Jun 2012 **Research Assistant, Telecommunications Lab, TUC, Greece.**
- Designed LDPC codes for the relay and multiple access channels.
- Designed and implemented an FPGA-based fully-parallel LDPC decoder.

Tools: MATLAB, C, VHDL, Xilinx EDK, LaTeX.

Jul 2011–Aug 2011 **Intern, Algorithmic Research in Network Information Lab, EPFL, Switzerland.**
- Explored subtree decomposition for network coding and developed relevant MATLAB code.

Tools: MATLAB, LaTeX.

Teaching Experience

2012–today	**Teaching Assistant, EPFL.** EE442 – Wireless Receivers: Algorithms and Architectures (2012-2015). EE542 – Advanced Wireless Receivers: Algorithms and Architectures (2013-2015).
2010–2012	**Teaching Assistant, TUC.** TEL301 – Digital Telecommunication Systems I (2010, 2011). TEL311 – Digital Telecommunication Systems II (2010). TEL313 – Sound and Music Processing (2010, 2012). TEL413 – Introduction to Optimization Theory (2011). TEL415 – Wireless Communications (2011).

Student Supervision

Cristina Teodorescu	MSc semester project, 2016, "A low-power and low-throughput LUT-based LDPC decoder"
Johannes Wüthrich	BSc internship, 2015, "An FPGA-based accelerator for rapid simulation of SC decoding of polar codes."
Orion Afisiadis	MSc thesis, 2014, "A low-complexity improved successive cancellation decoder for polar codes."
Afshin Mardani	MSc thesis, 2014, "Exploiting the resilience of error correcting codes under reliability constraints," co-supervised with Dr. Georgios Karakonstantis.
Hamedeh Jafari	MSc semester project, 2013, "Exploiting the error-resilience of polar decoders under unreliable silicon," co-supervised with Dr. Georgios Karakonstantis.

Honors and Awards

2016	Invited poster at Graduation Day event, ITA 2016.
2015	Best student paper award (2nd prize), IEEE ICECS 2015.
2015	Best student paper award finalist, IEEE ISCAS 2015.
2014	Exemplary reviewer, IEEE Wireless Communications Letters.
2013	Best student paper award (2nd prize), IEEE ICECS 2013.
2013	Exemplary reviewer, IEEE Wireless Communications Letters.
2011	Special Graduate Studies Scholarship, Technical University of Crete.
2010	Two-year Scholarship for Graduate Studies in Greece, Onassis Foundation.
	Special Graduate Studies Scholarship, Technical University of Crete.
2009	Scholarship of Academic Excellence (top of class), State Scholarships Foundation of Greece.
	Scholarship of Academic Excellence (top of class), Technical University of Crete.

Professional Service

Reviewer	IEEE Transactions on Communications IEEE Transactions on Wireless Communications IEEE Transactions on Signal Processing IEEE Journal on Selected Areas in Communications IEEE Wireless Communications Letters IEEE Communications Letters IEEE Transactions on Circuits and Systems I: Regular Papers IEEE Transactions on Circuits and Systems II: Express Briefs EURASIP Journal on Wireless Communications and Networking AIMS Advances in Mathematics of Communications IEEE ICC 2012, FPL 2013, IEEE ICECS (2013, 2015), IEEE SAM 2014, IEEE ISCAS (2015, 2016), IEEE ISIT (2015).
TPC	EUSIPCO (2014, 2015)
Web Chair	IEEE WoWMoM 2012, 2013, 2014 & 2015 ViDEv workshops.

Select Graduate Coursework

EPFL	Information Theory and Coding, Advanced Probability, Statistical Physics for Communications and Computer Science.
TUC	Probability and Random Processes, Information Theory and Coding, Detection and Estimation Theory, Convex Optimization, Reconfigurable Digital Systems.

Languages

Greek	Mother tongue.
German	Mother tongue.
English	Fluent (Cambridge CPE).
French	Fair (DELF B1).

Personal Interests

Arts	Part of a hobbyist band with four self-produced albums, British comedy aficionado.
Sports	Alpine skiing & snowboard, tennis, jogging, and motorcycling.

List of Publications

Journals

J1. **A. Balatsoukas-Stimming** and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," *IEEE Transactions on Information Theory*, Feb. 2016 (submitted)

J2. P. Giard, **A. Balatsoukas-Stimming**, G. Sarkis, C. Thibeault, and W. J. Gross, "Low-complexity polar decoders for low-rate codes," *Springer Journal of Signal Processing Systems*, Mar. 2016

J3. **A. Balatsoukas-Stimming**, A. C. M. Austin, P. Belanovic, and A. Burg, "Baseband and RF hardware impairments in full-duplex wireless systems: Experimental characterisation and suppression," *EURASIP Journal on Wireless Communications and Networking (Special Issue on Experimental Evaluation in Wireless Communications)*, May 2015

J4. **A. Balatsoukas-Stimming**, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Transactions on Signal Processing*, Mar. 2015

J5. **A. Balatsoukas-Stimming**, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Aug. 2014

J6. **A. Balatsoukas-Stimming** and A. Burg, "Density evolution for min-sum decoding of LDPC codes under faulty message storage," *IEEE Communications Letters*, May 2014

Conferences

C1. O. Afisiadis, A. C. M. Austin, **A. Balatsoukas-Stimming**, and A. Burg, "Analysis of full-duplex wireless links with asymmetric capacity requirements," in *IEEE Vehicular Technology Conference*, May 2017 (submitted)

C2. P. Giard, **A. Balatsoukas-Stimming**, T. C. Müller, A. Burg, C. Thibeault, and W. Gross, "A multi-Gbps unrolled hardware list decoder," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2016 (accepted)

C3. A. C. M. Austin, **A. Balatsoukas-Stimming**, and A. Burg, "Digital predistortion of power amplifier non-linearities for full-duplex transceivers," in *IEEE International workshop on Signal Processing advances in Wireless Communications (SPAWC)*, Jul. 2016

C4. F. Sheikh, **A. Balatsoukas-Stimming**, and C.-H. Chen, "High-throughput lattice reduction for large-scale MIMO systems based on Seysen's algorithm," in *IEEE International Conference on Communications (ICC)*, May 2016

C5. O. Afisiadis, A. C. M. Austin, **A. Balatsoukas-Stimming**, and A. Burg, "Sliding window spectrum sensing for full-duplex cognitive radios with low access-latency," in *IEEE Vehicular Technology Conference*, May 2016

C6. S. A. Hashemi, **A. Balatsoukas-Stimming**, P. Giard, C. Thibeault, and W. J. Gross, "Partitioned successive-cancellation list decoding of polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016

C7. P. Giard, G. Sarkis, **A. Balatsoukas-Stimming**, Y. Fan, C.-Y. Tsui, A. Burg, C. Thibeault, and W. J. Gross, "Hardware decoders for polar codes: An overview," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016

C8. J. Wüthrich, **A. Balatsoukas-Stimming**, and A. Burg, "An FPGA-based accelerator for rapid simulation of SC decoding of polar codes," in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Dec. 2015 (**best paper award**, 2nd prize)

C9. M. Meidlinger, **A. Balatsoukas-Stimming**, A. Burg, and G. Matz, "Quantized message passing for LDPC codes," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2015

C10. 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 *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2015

C11. **A. Balatsoukas-Stimming**, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled LDPC decoder based on quantized message passing," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2015

C12. **A. Balatsoukas-Stimming**, M. Bastani Parizi, and A. Burg, "On metric sorter architectures for list successive cancellation decoding of polar codes," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015 (**best paper award finalist**)

C13. O. Afisiadis, **A. Balatsoukas-Stimming**, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2014

C14. **A. Balatsoukas-Stimming** and A. Burg, "Faulty successive cancellation decoding of polar codes for the binary erasure channel," in *International Symposium on Information Theory and Applications (ISITA)*, Oct. 2014

C15. **A. Balatsoukas-Stimming**, G. Karakonstantis, and A. Burg, "Enabling complexity-performance tradeoffs for successive cancellation decoding of polar codes," in *IEEE International Symposium on Information Theory (ISIT)*, Jun. 2014

C16. K. Alexandris, **A. Balatsoukas-Stimming**, and A. Burg, "Measurement-based characterization of residual self-interference on a full-duplex MIMO testbed," in *IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*, Jun. 2014

C17. **A. Balatsoukas-Stimming**, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014

C18. **A. Balatsoukas-Stimming**, P. Belanovic, K. Alexandris, and A. Burg, "On self-interference suppression methods for low-complexity full-duplex MIMO," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2013

C19. **A. Balatsoukas-Stimming**, N. Preyss, A. Cevrero, A. Burg, and C. Studer, "A parallelized layered QC-LDPC decoder for IEEE 802.11ad in 40 nm CMOS," in *IEEE International New Circuits and Systems Conference (NEWCAS)*, Jun. 2013

C20. **A. Balatsoukas-Stimming** and A. Dollas, "FPGA-based design and implementation of a multi-Gbps LDPC decoder," in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012

Posters, Demos & Invited Talks

I1. **A. Balatsoukas-Stimming**, R. Ghanaatian, and A. Burg, "Complexity and energy efficiency of LDPC decoders and an initial comparison to polar codes." *Workshop on energy-efficiency in error-correction coding*, Télécom ParisTech, Paris, France, Jun. 2016

I2. **A. Balatsoukas-Stimming**, "Faulty successive cancellation decoding of polar codes for the binary erasure channel." McGill University, Montréal, Canada, May 2016

I3. A. Burg, **A. Balatsoukas-Stimming**, M. Meidlinger, and G. Matz, "Information optimized quantized message passing to enable ultra-high-speed (Tbps) LDPC decoding," in *Information Theory and Applications (ITA) Workshop*, Feb. 2016

P4. **A. Balatsoukas-Stimming**, "From coding theory to coding practice: Hardware implementation of polar decoders and Terabit/s LDPC decoders," in *Information Theory and Applications (ITA) Workshop Graduation Day Poster*, Feb. 2016

D5. A. C. M. Austin, O. Afisiadis, **A. Balatsoukas-Stimming**, and A. Burg, "Concurrent spectrum sensing and transmission for cognitive radio using self-interference cancellation," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Jun. 2015

I6. **A. Balatsoukas-Stimming** and A. Burg, "Polar decoding with unreliable memories," in *Information Theory and Applications (ITA) Workshop*, Feb. 2015

I7. **A. Balatsoukas-Stimming**, C. Studer, and A. Burg, "Characterization of min-sum decoding of LDPC codes on unreliable silicon," in *Information Theory and Applications (ITA)*, Feb. 2014

D8. P. Belanovic, **A. Balatsoukas-Stimming**, and A. Burg, "A multipurpose testbed for full-duplex wireless communications," in *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Dec. 2013 (**best paper award**, 2nd prize)