Fast, Flexible, and Area-Efficient Decoders for Polar Codes

Seyyed Ali Hashemi

Department of Electrical and Computer Engineering
McGill University
Montreal, Canada

December 2018

A thesis submitted to McGill University in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Though the waystation be dangerous and the destination far, far away;
There is no road that has no end; grieve not! *Hafez*

Contents

Abstract

Polar codes have received a great deal of attention in the past few years to the extent that they are selected to be included in the 5th Generation of Wireless Communications Standard (5G). Specifically, polar codes were selected as the coding scheme for the Enhanced Mobile Broadband (eMBB) control channel which requires codes of short length. The main bottleneck in the deployment of polar codes in 5G is the design of a decoder which can achieve good error-correction performance, with low hardware implementation cost and high throughput. Successive-Cancellation (SC) decoding was the first algorithm under which polar codes could achieve capacity when the code length is very high. However, for finite practical code lengths, SC decoding falls short in providing a reasonable error-correction performance because of its sub-optimality with respect to the Maximum-Likelihood (ML) decoder. Sphere Decoding (SD) is an algorithm that can achieve the performance of ML decoding with a very high complexity. In order to close the gap between SC and ML decoding, Successive-Cancellation List (SCL) decoding keeps a list of candidates and selects the one with the best Path Metric (PM). Although SCL provides a good error-correction performance, it comes at the cost of higher complexity and lower throughput. In this thesis, we first propose a low complexity SD algorithm which provides a good trade-off between the error-correction performance and the complexity of the decoder for polar codes of short lengths. We then propose algorithms to speed up the SCL decoders. We prove that while these algorithms have much higher throughput than the conventional SCL decoder, they incur no error-correction performance loss. We further propose several techniques to reduce the area occupation in the hardware implementation of SC and SCL decoders by reducing their memory requirements. We solve the flexibility issue of fast SC-based decoders and introduce a completely rate-flexible scheme. Hardware architectures for the proposed algorithms are presented and comparisons with state of the art are made. Finally, we evaluate the performance of polar codes in 5G and we show that polar codes can be used in practical applications by proposing a blind detection scheme with polar codes.

Résumé

Les codes polaires occupent depuis quelques années l'attention de la communauté académique du codage de canal. Cet intérêt s'est étendu à l'industrie puisque les codes polaires prennent part au standard de communications mobiles de cinquième génération (5G). Plus précisément, ils sont sélectionnés comme schéma de codage pour le canal de contrôle du service mobile à large bande amélioré (Enhanced Mobile Broadband (eMBB)), requérant des codes de faible longueur. Le principal obstacle dans le déploiement des codes polaires pour la 5G est la conception d'un décodeur ayant un coût d'implantation matériel faible, tout en présentant à la fois de bonnes performances de correction d'erreurs et un débit élevé. Le décodage à annulation successive (Successive-Cancellation (SC)) est l'algorithme de décodage originel des codes polaires. Il permet d'atteindre la limite de capacité théorique, à condition que la taille du code polaire soit suffisamment grande. Cependant, pour des tailles de trame finies, l'algorithme SC présente des performances de décodage médiocres, provenant de sa sous-optimalité au regard du décodage à maximum de vraisemblance (Maximum-Likelihood (ML)). Le décodage par sphère (Sphere Decoding (SD)) atteint les performances ML au détriment d'une complexité calculatoire importante. Afin de réduire l'écart de performances entre les décodages ML et SC, le décodage à annulation successive par liste (Successive-Cancellation List (SCL)) a été proposé. Son principe réside dans le maintient d'une liste de mots candidats et de la sélection à l'issue du décodage du mot ayant la meilleure métrique. Bien que l'algorithme SCL présente de bonnes performances de décodage, sa complexité calculatoire est plus importante et son débit est réduit par rapport à l'algorithme SC. Dans ces travaux de thèse, nous proposons tout d'abord un algorithme SD à faible complexité calculatoire. Il permet d'obtenir un bon compromis entre sa performance de correction d'erreurs et sa complexité calculatoire pour des codes polaires de faibles tailles. Ensuite, nous proposons différents algorithmes accélérant les décodeurs SCL. Ces algorithmes permettent d'atteindre un débit bien plus élevé que le décoder SCL conventionnel, tout en assurant les mêmes performances de décodage. Ensuite, nous proposons différentes techniques réduisant l'occupation surfacique d'implémentations matérielles de décodeurs SC et SCL. Ces techniques sont basées sur la réduction des besoins mémoire de ces algorithmes. Nous résolvons les problèmes de flexibilité des décodeurs basés sur l'algorithme rapide SC et présentons une méthode compatible avec tout rendement de code. Des architectures matérielles adaptées aux algorithmes proposés sont présentées et des comparaisons avec l'état de l'art sont établies. Finalement, nous évaluons les performances des codes polaires du standard 5G et nous montrons que

les codes polaires peuvent être utilisés dans des applications pratiques en proposant une méthode de détection aveugle.

Acknowledgments

As I am starting to write this Acknowledgments section, I am trying to name all the people who helped, supported, and encouraged me during my Ph.D. studies and it seems to me that they are countless. In fact, I am not even sure where to start from. So I stick to tradition and start by thanking my supervisor, Professor Warren Gross who in the first place, gave me the chance to study Ph.D. under his supervision, and in the second place, walked me through this challenging journey. He always had a smile and our meetings were in a very friendly atmosphere. Thank you Professor for providing me with an amazing Ph.D. life.

I would like thank my Ph.D. committee members, Professor Brett Meyer and Professor Ioannis Psaromiligkos for their helpful advices regarding my research plans.

I would like to thank Carlo Condo but I do not know how to thank him. Take a look at the list of my publications and you will see his footsteps in almost every part of this thesis. He was always available when you needed him. No matter if he was on vacation, it was a weekend, or late at night. I always remember the time we spent together discussing ideas, working on the papers, or getting ready for presentations. His knowledge was super useful to me, especially in the hardware design. He was literally my co-supervisor. No need to say that we enjoyed his voice when he used to sing along the song being played on his headphones! Thanks Carlo.

I would like to thank Arash Ardakani. We sat side by side during the whole duration of Ph.D. and we almost had every single lunch together. Although we were working on two completely different topics, he was the one with whom I had the chance to discuss about research the most. His vast knowledge of hardware design enabled me to reach a better understanding of its challenges. He was a friend that I could talk to any time about anything. We made tea twice a day so he is the only one who can tell you how many cups of tea I had as a Ph.D. student! Thanks Arash.

I would like to thank Gabi Sarkis and Pascal Giard. I was lucky to have started my Ph.D. when Gabi and Pascal were members of the lab. They were very patient in answering all of my very basic [mostly stupid!] questions. Thanks Gabi and Pascal.

I would like to thank Furkan Ercan. We had extensive discussions about research and he was always helpful. Furkan always had a story to tell and bring joy to the lab and he was available inside and outside of school. He is also a great photographer! Saygılar abi!

I would like to thank Nghia Doan. In fact, I thought I was working hard until Nghia joined our lab. He is very hard-working and he is very passionate to learn. I really enjoyed working with him.

Thanks Nghia.

I would like to thank other members of our lab from past till present, Harsh Aurora [with his jokes!], Thibaud Tonnellier [the Résumé section is his work!], Adam Cavatassi, Elie Ngomseu Mambou, Jerry Ji, Hooman Jarollahi, and François Leduc-Primeau, with whom I had the chance to discuss and learn from.

I now want to go outside the lab and thank Marco Mondelli. His deep knowledge of information theory enabled me to look at the problems from a different perspective. Although we met in person only once in Barcelona, we had numerous Skype calls to discuss about research which resulted in multiple publications. Thanks Marco.

I would like to thank Professor Hamed Hassani and Professor Rüdiger Urbanke with whom I had the chance to work and learn from.

I would like to thank Alexios Balatsoukas-Stimming. I was very fortunate to write one of my first papers in collaboration with him. He was very helpful and friendly. We met several times in different conferences and I always enjoyed talking to him. I remember the trip I had with him and Orion Afisiadis to the 17-Mile Drive, and the jar of candy that we won at a trivia night! Thanks Alexios.

I would like to thank my M.Sc. advisor, Professor Behrouz Nowrouzian from whom I learned how to conduct research and how to write.

I would also like to thank the staff at the Department of Electrical and Computer Engineering of McGill who were always helpful.

Aside from research, I had the chance of having a lot of very good friends at McGill. I would like to thank the members of the group "patogh" with whom I had lunch and discussed about the non-Ph.D. life. Namely, I would like to thank Shohreh Shaghaghian, Dena Firoozi, Ali Pakniyat, Mohammad Afshari, Mostafa Darabi, Mehdi Roozegar, Jalal Arabneydi, Parisa Moslemi, Dorna Firoozi, Babak Shahmansouri, and Mina Rafienazari.

I would also like to thank my dear friends outside McGill, Mahdi Mirhoseini, Niloufar Afsari, Amirhossein Farshad, Mahya Shariatfar, Mehran Kalantari, Sanaz Mehrzad, Farhad Haghighizadeh, Mohammadreza Arani, Amir Rastpour, Mohsen Ghafouri, Ali Shahrad, Arian Hosseinzadeh, Faraz Falsafi, Mohammad Keymanesh and many others who I did not find the chance to name here. Also, a big shout-out to our football and volleyball teams who made me stay fit [to some extent!].

I have now reached the most important part of this section. It is not possible to thank my parents, Shirin and Hedayat, my sisters, Ziba and Hura, and my brother-in-law, Amin, in words.

Their unconditional love and support was a true guide throughout my life. They simply dedicated their life to my well-being and success without me even noticing how I am progressing in life. Studying abroad was difficult not in the sense of having research challenges, but in the sense of being far from home. I cannot explain how difficult it is for a child to be away from his parents and I am sure it is even more difficult for parents to be away from their child. Thanks Maman and Baba for tolerating me and supporting me.

I would also like to thank my in-laws, Tahereh, Farzad, Farnaz, Navid, Behnaz, and Saber, and my nieces, Niki, and Nila, who made me feel I am not away from home.

Last but not least, I would like to thank my dear wife Golnaz. I cannot even think of being able to finish my studies without her by my side. She gives me the reason to live and love. She showed me that life is not only work so I could advance in my studies by being more efficient. I can always count on her for anything that I need, including the problems I face in research. She simply accepted me the way I am and I am very fortunate to have her in my life. Thanks Golnaz, I love you!

List of Figures

List of Tables

List of Acronyms

3GPP 3rd Generation Partnership Project. 5, 135, 136

5G 5th Generation of Wireless Communications Standard. v, vii, viii, xvii, 2, 4, 5, 8, 10, 12, 14, 111, 113, 122–124, 126, 127, 129, 130, 132, 135, 136, 139, 143, 144

ASIC Application-Specific Integrated Circuit. xix, 108–110

AWGN Additive White Gaussian Noise. xx, 5, 16, 86, 96–98, 123

BAWGN Binary AWGN. xv, xvii, 86, 89, 111, 112

BEC Binary Erasure Channel. 84, 111, 145

BER Bit Error Rate. xiii, xiv, xvi, xvii, 31, 34–37, 54–56, 62, 66–68, 100–104, 123, 124

BMS Binary Memoryless Symmetric. 15, 115

BP Belief Propagation. 144

BPSK Binary Phase-Shift Keying. 16

CMOS Complementary Metal-Oxide-Semiconductor. xviii, xix, 78, 84, 107, 109, 110, 123, 125, 126

CRC Cyclic Redundancy Check. v, xiv–xviii, xxiii, 3, 5, 7, 8, 24, 34, 49–51, 54–56, 62, 65–68, 70, 76–78, 80, 84, 86–90, 92–109, 129, 131–133, 135, 136, 143, 144

CU Control Unit. 123

DCI Downlink Control Information. 135, 136

ECC Error-Correcting Codes. 1, 2

eMBB Enhanced Mobile Broadband. vii, viii, 2, 4, 5, 8, 122, 123, 129, 132, 143

FAR False Alarm Rate. xvii, 136, 137, 139, 140

Fast-SSC Fast Simplified Successive-Cancellation. iv, 3, 19, 21, 39, 101, 140, 141

Fast-SSCL Fast Simplified Successive-Cancellation List. v, xiv, xvii, xviii, 6, 7, 56, 57, 59, 61, 63–67, 69, 71–73, 75–77, 79–81, 129, 132, 133, 140, 141

Fast-SSCL-SPC Fast Simplified Successive-Cancellation List with Single Parity-Check. xiv, xviii, 7, 56, 63–66, 68, 69, 71–77, 79–82, 120, 121, 125, 126

FEC Forward Error Correction. 1

FER Frame Error Rate. xiii–xvii, 8, 31, 34–37, 54–56, 62, 66–68, 87–92, 94–104, 110, 112, 123, 124, 129–132, 136–139

FPGA Field Programmable Gate Array. xviii, 37, 38, 108

FSM Finite State Machine. 122–124

GA Gaussian Approximation. xvi, 95–97, 99, 100

GPSCL Generalized Partitioned Successive-Cancellation List. xv, 7, 83, 84, 89–93, 106, 107, 109, 110, 143

HFS High-Frequency Structure. 52, 78

LCS Low-Complexity Structure. 52, 78

LDPC Low-Density Parity-Check. xv, xvi, 2, 24, 88, 90, 93, 96

List-SD List Sphere Decoding. iv, xiii, xviii, 6, 14, 29–38, 40, 48

LL Log-Likelihood. 4

LLR Log-Likelihood Ratio. v, xvi, xviii, 4, 7, 16–20, 23, 30, 40, 45, 46, 49, 52, 54, 57–60, 62, 63, 67–70, 72–76, 78–80, 91, 96, 97, 101–109, 120–123, 137

LLS Low-Latency Structure. 52, 78, 79

LPSCL Layered Partitioned Successive-Cancellation List. xv–xvii, 7, 84, 91, 93, 94, 106, 107, 109, 110, 120–124, 143

LTE Long-Term Evolution. 5, 135, 136

LUT Look-Up Table. 37, 38, 109

MAP Maximum a Posteriori. v, xvii, 84, 86, 93, 110–112, 145

MDR Missed Detection Rate. xvii, 136–139

ME Memory-Efficient. 102, 103

ML Maximum-Likelihood. vii, viii, 2, 3, 6, 24, 25, 29

MR Matrix Reordered. 35–38

PDCCH Physical Downlink Control Channel. 135, 136

PE Processing Element. xiv, 49, 50, 67, 68, 70, 71, 76, 78, 80, 81, 120, 121, 123, 126

PM Path Metric. vii, 3, 4, 23, 24, 26, 27, 29–31, 37, 40–42, 44–50, 52, 54, 57–61, 63, 69, 70, 72–76, 79, 80, 103, 104, 137

PSCL Partitioned Successive-Cancellation List. v, xv, xvi, xviii, 7, 11, 13, 83–91, 93–95, 97–103, 105–109, 111, 112, 143

Rate-0 Rate Zero. xiv, xvii, 3, 6, 20, 22, 40, 45–47, 50–54, 63, 64, 71, 72, 76, 78–80, 114, 116, 117, 119, 120, 122, 123

Rate-1 Rate One. xvii, 3, 4, 6, 20, 22, 40–44, 50, 52, 54, 56–58, 60–65, 71, 72, 74, 78, 80, 82, 114, 116, 117, 119, 120, 122, 123, 133

Rep Repetition. xvii, 3, 6, 20, 22, 40, 46–48, 50, 52, 54, 63, 64, 71, 72, 76, 78–80, 114, 116, 117, 119, 120, 122, 123

RM Reed-Muller. vi, 11, 145

RNTI Radio Network Temporary Identifier. 135–140

RTL Register-Transfer Level. 78

SC Successive-Cancellation. iv, vii, viii, xiii, xv, xvii, xviii, 2–5, 7–9, 11, 14, 17–25, 30, 32, 34–37, 39, 49, 67, 70, 84–89, 96, 97, 101–103, 105, 106, 113, 114, 119–124, 129, 130, 135, 137, 138, 140, 141, 144, 145

SCA Successive CRC Assignment. xvi, 84, 94–96

SCL Successive-Cancellation List. iv, v, vii, viii, xiii, xv–xviii, 3–8, 10, 12, 13, 23–25, 30, 34–37, 39, 40, 42, 45, 46, 48, 49, 51, 52, 54, 56, 61, 64, 66, 71, 74–79, 81, 83–95, 98, 100–112, 120, 123, 124, 127, 129–132, 135, 137–141, 143–145

SD Sphere Decoding. iv, vii, viii, xiii, 2, 4, 6, 25–27, 29

SNR Signal-to-Noise Ratio. xv–xvii, 2, 83, 86–89, 91, 92, 94–96, 111, 129–133, 138–140

SPC Single Parity-Check. xvii, 3, 4, 6, 20–22, 40, 48–50, 54, 56, 62–65, 71, 73, 74, 78–80, 82, 114, 116, 117, 119, 120, 122, 123

SSCL Simplified Successive-Cancellation List. v, xiv, xv, xvii, xviii, 6, 7, 39–41, 43, 45, 47, 49, 51, 53–57, 62–65, 71–73, 76–80, 129, 132, 133, 140, 141

SSCL-SPC Simplified Successive-Cancellation List with Single Parity-Check. xiv, xv, xviii, 6, 7, 40, 51, 54–56, 62–65, 71–73, 76–82, 126

TSMC Taiwan Semiconductor Manufacturing Company. xviii, xix, 78, 79, 84, 107, 109, 110, 123, 125, 126

UE User Equipment. 5, 135–138

VHDL Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. 78, 107, 109, 123

WiMAX Worldwide Interoperability for Microwave Access. xv, xvi, 88, 90, 96

XOR Exclusive Or. 18, 54, 74, 75, 80

Chapter 1

Introduction

A very fundamental part of a data transmission from one place to another is the communication channel. This channel is basically the medium in which the data is being transferred, e.g. cables, air, etc. Since the medium is not ideal for data transmission, it always contains noise. Therefore, the data that is seen at the receiver may not be the same as the data that was sent by the transmitter rendering the data transmission as unreliable. In order to increase the reliability of the data transfer, error detection and error correction techniques have been employed. A basic representation of a communication scenario is given in Figure 1.1.

Error detection is the process of determining if a received digital data is actually what was sent by the transmitter. A very simple example of such coding scheme is the repetition code which sends multiple copies of blocks of the data across the channel. For example, a block of 0111 can be copied three times and sent as 011101110111. If the receiver receives the data as 010101110111, it can tell that the data contains an error because the first block of data is not the same as the next two. It can also recover the data by choosing the blocks that happened more frequently which in this case is 0111. This operation is called error correction. However, this method of coding is very inefficient. In order to be able to reliably reconstruct the transmitted data, Error-Correcting Codes (ECC) are used.

ECC or Forward Error Correction (FEC) is a method in which redundant data (parity bits) is added to the information (encoding) and the augmented data is sent through the channel. In the receiver side, the receiver reconstructs data by using the parity bits (decoding). FEC has the advantage of not requiring a back-channel to communicate with the transmitter in case an error has occurred. Therefore, it is simpler and can be used in a wide range of communication channels.

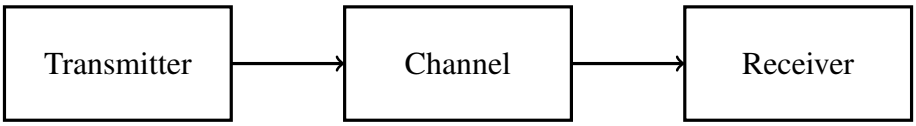


Figure 1.1: Basic communication scheme

However, the question is how many parity bits can be added to the information so that the communication is reliable. This question is answered by Shannon's theorem [1] which states that there is a maximum information rate at which a reliable communication can be established over a channel with a known error probability or Signal-to-Noise Ratio (SNR). This maximum information rate is called the channel capacity.

There has been a great deal of effort in order to reach the channel capacity for high-rate reliable communication. Low-Density Parity-Check (LDPC) [2], [3] and Turbo codes [4] are the two powerful coding techniques which can reach the channel capacity for some specific configurations. Polar codes are the first family of ECC with provable capacity-achieving property and a low-complexity encoding and decoding process [5]. They received an extensive amount of attention in the past few years such that they are selected to be included as a coding scheme for the 5th Generation of Wireless Communications Standard (5G) [6]. Currently, polar codes are selected as the coding scheme for the control link in Enhanced Mobile Broadband (eMBB) channel which requires codes of short lengths. Since their adoption in 5G, the design of fast and low complexity decoders for polar codes which can achieve a good error-correction performance have been an active topic of research in industry and academia.

Short polar codes have been decoded using Maximum-Likelihood (ML) techniques. In [7], the Sphere Decoding (SD) algorithm was used to decode short polar codes and in [8] an optimal path metric was developed for SD. It exploits the lower triangular structure of the polar code generator matrix, leading to roughly $O(N^3)$ complexity for a polar code of length N. While it is able to reach the ML bound, the cubic complexity limits the usage of SD to short codes. Moreover, SD suffers from two main drawbacks. First of all, the time complexity of SD is highly dependent on the channel conditions and therefore is variable. Thus, it is difficult to guarantee the decoder's error-correction performance and throughput. Secondly, SD relies on the selection of a sphere radius, which also depends on the channel characteristics. Choosing a suboptimal radius can lead to strong error-correction performance degradation.

The Successive-Cancellation (SC) decoding is an algorithm that can decode a polar code of

length N with complexity $O(N \log N)$, under which polar codes can achieve the capacity of a memoryless channel. However, there are two main drawbacks associated with SC. Firstly, SC requires the decoding process to advance bit by bit. This results in high latency and low throughput when implemented in hardware [9]. Second, polar codes decoded with SC only achieve the channel capacity when the code length tends toward infinity. For practical polar codes of short to moderate length, SC falls short in providing a reasonable error-correction performance.

The first issue is a result of the serial nature of SC. In order to address this issue, the recursive structure of polar codes construction and the location of information and parity (frozen) bits were utilized in [10], [11] to identify constituent polar codes and to develop Fast Simplified Successive-Cancellation (Fast-SSC) decoding. In particular, Rate Zero (Rate-0) codes with all frozen bits, Rate One (Rate-1) codes with all information bits, Repetition (Rep) codes, and Single Parity-Check (SPC) codes were shown to be capable of being decoded in parallel with low-complexity decoding algorithms. This in turn increased the throughput and reduced the latency significantly. Moreover, the simplifications in [10], [11] did not introduce any error-correction performance degradation with respect to conventional SC.

The second issue stems from the fact that SC is suboptimal with respect to ML decoding. The decoding of each bit is only dependent on the bits already decoded. SC is unable to use the information about the bits that are not decoded yet. In order to address this issue, Successive-Cancellation List (SCL) decoding advances by estimating each bit as either 0 or 1. Therefore, the number of candidate codewords doubles at each bit estimation step. In order to limit the exponential increase in the number of candidates, only L candidate codewords are allowed to survive by employing a Path Metric (PM) [12]. The PMs were sorted and the L best candidates were kept for further processing. SCL reduces the gap between SC and ML and it was shown that when a Cyclic Redundancy Check (CRC) code is concatenated with polar codes, SCL can make polar codes outperform the state-of-the-art codes [13].

The good error-correction performance of SCL comes at the cost of higher latency, lower throughput, and higher area occupation than SC when implemented on hardware [14]. In order to reduce the latency and increase the throughput associated with SCL, a group of M bits were allowed to be decoded together in [15], [16]. [17] proposed a high throughput architecture based on a tree-pruning scheme and further extended it to a multimode decoder in [18]. The throughput increase in [17] is based on code-based parameters which could degrade the error-correction performance significantly. Based on the idea in [11], a fast SCL decoder architecture for software

implementation was proposed in [19] which was able to decode constituent codes in a polar code in parallel. This resulted in fewer number of time-steps to finish the decoding process. However, the SCL decoder in [19] is based on an empirical approach to decode Rate-1 and SPC nodes and cannot guarantee the same error-correction performance as the conventional SCL decoder. Moreover, all the decoders in [17]–[19] require a large sorter to select the surviving candidate codewords. Since the sorter in the hardware implementation of SCL decoders has a long and dominant critical path which is dependent on the number of its inputs [20], increasing the number of PMs results in a longer critical path and a lower operating frequency. Path pruning schemes were also proposed in [21]–[23] to speed up the SCL decoding process. However, these schemes are based on approximations that work for specific code parameters and channel conditions.

It was identified in [20] that using the Log-Likelihood Ratio (LLR) values results in a SCL decoder which is more area-efficient than the conventional SCL decoder with Log-Likelihood (LL) values. The reduction of memory requirements of SCL decoders, that are higher than those of SC, has been addressed in [18], [24], [25]. However, in [18], the design need to be re-evaluated when the code changes. The solution presented in [24] is based on LL messages, which require more memory than its LLR counterparts, and the SD-based technique in [25] suffers from error-correction performance degradation as the code length increases. Effective memory-reduction techniques that do not incur performance loss are needed, especially within the challenging 5G framework.

The construction of polar codes is based on the identification of reliable bit-channels through which information bits are transmitted. The frozen bit-channels carry fix values and the location of the frozen bits and of the information bits is known to the encoder and the decoder. In SC-based decoders, the frozen and information bit sequence can be either stored in a memory, or computed online given the bit-channel relative reliability vector and desired code rate, as proposed in [26]. In fact, the latter approach is significantly more efficient in case of multi-code decoders, and is facilitated by nested reliability vectors as those selected for the 5G eMBB control channel [27]. Therefore, in 5G, the polar encoder and decoder are provided with a vector of bit indices in descending reliability order and an information length K, from which the encoder and the decoder should extract the frozen/information bit sequence. It should be noted that the number of information bits for polar codes in the 5G eMBB control channel can be any value no more than 1706 [28]. Thus, the encoder and the decoder should be able to support a vast range of code rates.

Fast SC-based decoders [11] rely on the identification of the type and the length of constituent codes in a polar code. While the calculation of the frozen/information bit sequence is straight-

forward and can be performed by simply assigning information bits to the first K elements of the reliability vector, the direct calculation of the list of operations for fast SC-based decoders requires complicated controller logic [11]. Therefore, the identification of the type and the length of constituent codes is performed off-line and the decoding order is stored in a dedicated memory as a list of operations [11]. The decoder fetches the list of operations from memory to decode the constituent codes in order one by one. The main drawbacks of the aforementioned fast SC-based decoders are twofold: first, the list of operations requires high memory usage when implemented on hardware. Second, the list of operations is highly dependent on the rate of the polar code and as the rate changes, the list of operations changes too. Therefore, for 5G applications which require the support of multiple rates, multiple lists of operations need to be stored in memory. This in turn increases the hardware implementation overhead and renders fast SC-based decoders not rate-flexible.

Blind decoding, or blind detection, is foreseen by the 3GPP LTE/LTE-Advanced standards to allow the User Equipment (UE) to gather control information related to the downlink shared channel. The UE attempts the decoding of a set of candidates determined by combinations of system parameters, to identify if one of the candidates holds its control information. The scheme used in LTE relies on the concatenation of a CRC with a convolutional code. Blind detection will be present also in 5G: ongoing discussions are considering a substantial reduction of the time frame allocated to blind detection, from 16μs to 4μs. Blind detection must be performed frequently, and given the high number of decoding attempts required in a limited time [29], it can lead to large implementation costs. Since polar codes are selected in 5G eMBB control channel, the blind detection scheme needs to be performed by polar codes.

In this thesis, we tackle the aforementioned issues associated with SC-based decoding algorithms such as SC and SCL. It should be noted that the error-correction performance results in this paper are all based on the Additive White Gaussian Noise (AWGN) channel model. However, 5G requires the error-correction performance to be evaluated also for fading channels [30]. Nevertheless, the algorithms which are proposed in this thesis are mostly hardware-oriented and thus can also be utilised in a fading channel scenario.

1.1 Summary of Contributions

In this thesis, we aim to tackle the issues associated with polar code decoding and their deployment in 5G. Throughout this thesis, the hardware synthesis results were obtained with the help of

Dr. Carlo Condo. The contributions can be summarized as follows:

List Sphere Decoder

We develop a new algorithm to solve the issues regarding the SD algorithm. Similarly to SCL, after each bit estimation, L candidates are allowed to survive: hence, we call the proposed algorithm List Sphere Decoding (List-SD). The time complexity of List-SD is fixed and there is no need to choose a sphere radius value, while the parameter L can be chosen to tune the trade-off between error-correction performance (up to the ML bound) and hardware complexity. We further improve List-SD by exploiting the structure of the generator matrix of polar codes. A novel matrix reordering technique is presented: it changes the order in which bits are decoded in List-SD, and allows significant reduction in complexity without affecting the error-correction performance.

Simplified Successive-Cancellation List Decoder

We propose a SCL speed-up technique that does not rely on any approximations and its error-correction performance is guaranteed to be exactly the same as the conventional SCL decoder. The idea is that Rate-0, Rate-1, and Rep nodes can be decoded more efficiently in SCL decoding. We call the resulting algorithm Simplified Successive-Cancellation List (SSCL). We improve SSCL by proposing a new decoder for SPC codes and call the resulting algorithm Simplified Successive-Cancellation List with Single Parity-Check (SSCL-SPC). We show that the error-correction performance of SSCL-SPC is identical to that of the conventional SCL decoder for list size $L = 2$, and whose error-correction performance loss is negligible (< 0.05 dB) for all other list sizes.

Fast Simplified Successive-Cancellation List Decoder

We show that while SSCL and SSCL-SPC are algorithms that can work with any list size, they fail to address the redundant bit-estimations associated with a specific list size. Therefore, we first prove that there is a specific number of bit-estimations required for decoding the nodes in SSCL and SSCL-SPC for every list size to guarantee the error-correction performance preservation. Any bit-estimation after that number is redundant and any bit-estimation before that number cannot provably preserve the error-correction performance. Since these decoders require fewer number of time-steps than SSCL and SSCL-SPC, we name them Fast Simplified Successive-Cancellation

List (Fast-SSCL) and Fast Simplified Successive-Cancellation List with Single Parity-Check (Fast-SSCL-SPC), respectively. We further show that in practical polar codes, we can achieve similar error-correction performance to SSCL and SSCL-SPC with even fewer number of bit-estimations. Therefore, we can optimize Fast-SSCL and Fast-SSCL-SPC for speed. We propose hardware architectures to implement both new algorithms: implementation results yield the highest throughput in the state-of-the-art with comparable area occupation.

Memory-Efficient Polar Decoder

In order to address the high area occupation of SCL decoders, we propose a Partitioned Successive-Cancellation List (PSCL) decoding algorithm that reduces the memory requirements associated with SCL decoding. More specifically, PSCL decoding performs SCL decoding on partitions of the decoder tree and only one path candidate is transferred from one partition to the next. As a result, memory can be shared between the different partitions of the code, therefore, significantly reducing the overall memory requirements. We further present a Generalized Partitioned Successive-Cancellation List (GPSCL) algorithm that allows more than one candidate to be passed between different partitions. Then, we observe that the SCL decoder has a tree structure in which a specific number of candidate codewords are present at each level of the tree, and we propose a Layered Partitioned Successive-Cancellation List (LPSCL) algorithm that passes different numbers of candidates for different levels. It should be noted that if we pass similar number of candidates at each level of the tree, then LPSCL reduces to GPSCL. Therefore, LPSCL is the generalization of GPSCL in which we are able to fully tune the trade-off between the error-correction performance and the decoder complexity. We also propose two CRC selection schemes which improve the error-correction performance of PSCL. In addition, we propose a set of memory reduction techniques for SC-based decoders, that are orthogonal to the decoder architecture. In particular, aside from partitioning, we present a memory sharing technique that does not introduce any approximation and is independent of the decoder hardware structure, and a study on quantization that allows to reduce the bits necessary to represent the channel LLR values.

Rate-Flexible Fast Polar Decoder

We propose completely rate-flexible fast SC-based decoders by introducing a method to infer the list of operations directly in hardware without the need to store it in memory. We show that the

type and the length of a constituent code in a polar code can be identified with low hardware implementation complexity, by checking only a few bits of the constituent code. We further show that the list of operations adapts with the rate of the code, allowing the resulting fast SC-based decoder to be completely rate-flexible. We design and implement a hardware architecture for the proposed decoder and show that the memory required to store the list of operations can be completely removed, resulting in significantly lower decoder area occupation.

Performance of Polar Codes in 5G

5G is projected to have stringent requirements in terms of power efficiency, error-correction performance, and throughput. We evaluate the performance of polar codes in the 5G framework by studying polar codes of short lengths and different code rates. We show that for a fixed target Frame Error Rate (FER), there is an optimal code rate with which SC and SCL decoders can achieve it with maximum power efficiency. In addition, we study the effect of CRC on the error-correction performance of SCL decoders and show that for a given set of CRC lengths and CRC polynomials, there is an optimal CRC length with which the decoder achieves its best results. We further analyze the speed of polar code decoding by considering state-of-the-art fast SCL decoders available in literature, thus providing a survey of the decoder design space for eMBB, considering error-correction performance, achievable throughput, flexibility and estimated complexity.

Blind Detection with Polar Codes

We propose a blind detection scheme based on polar codes. A first SC decoding stage helps selecting a set of candidates, subsequently decoded with SCL. The scheme is evaluated in terms of error-correction capability, missed detections and false alarms, showing its compliance with the requirements of the standard. The detection speed is analyzed, identifying possible combinations of system parameters to meet the standard current and future timing constraints.

1.2 Related Publications

This doctoral research has resulted in several publications, a list of which is provided here.

Journal Papers

1. **S. A. Hashemi**, C. Condo, M. Mondelli, W. J. Gross, "Rate-Flexible Fast Polar Decoders", In: *IEEE Transactions on Signal Processing*, 2018 [31].

   This paper resolves the rate-flexibility issue associated with fast SC-based decoders for polar codes. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 6.

2. C. Condo, **S. A. Hashemi**, A. Ardakani, F. Ercan, W. J. Gross, "Design and Implementation of a Polar Codes Blind Detection Scheme", In: *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018 [32].

   This paper presents the hardware implementation of the blind detection scheme that we presented in [35]. My contributions to this paper was to help writing the manuscript.

3. **S. A. Hashemi**, M. Mondelli, S. H. Hassani, C. Condo, R. Urbanke, W. J. Gross, "Decoder Partitioning: Towards Practical List Decoding of Polar Codes", In: *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3749-3759, September 2018 [33].

   This paper presents a decoding scheme for polar codes with which the error-correction performance and the implementation complexity trade-off can be fully tuned. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 5.

4. **S. A. Hashemi**, C. Condo, F. Ercan, W. J. Gross, "Memory-Efficient Polar Decoders", In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 604-615, December 2017 [34].

   In this paper, we present several techniques to reduce the memory requirements of SC-based decoders of polar codes. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 5.

5. C. Condo, **S. A. Hashemi**, W. J. Gross, "Blind Detection with Polar Codes", In: *IEEE Communications Letters*, vol. 21, no. 12, pp. 2550-2553, December 2017 [35].

   This paper presents a blind detection scheme with polar codes which meets the requirements of 5G. My contributions to this paper were to help in developing the idea, implement the idea in software and test it, produce the results, and help in writing the manuscript. My contributions to this paper are presented in Chapter 8.

6. **S. A. Hashemi**, C. Condo, W. J. Gross, "Fast and Flexible Successive-Cancellation List Decoders for Polar Codes", In: *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756-5769, November 2017 [36].

   This paper presents a fast decoding algorithm for polar codes which is guaranteed to preserve the error-correction performance with respect to SCL decoding. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 4.

7. **S. A. Hashemi**, C. Condo, W. J. Gross, "A Fast Polar Code List Decoder Architecture Based on Sphere Decoding", In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2368 - 2380, December 2016 [37].

   This paper presents a fast decoding algorithm for polar codes which is guaranteed to preserve the error-correction performance with respect to SCL decoding. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 4.

Conference Papers

1. **S. A. Hashemi**, N. Doan, M. Mondelli, W. J. Gross, "Decoding Reed-Muller and Polar Codes by Successive Factor Graph Permutations", In: *International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, Hong Kong, to appear, 2018 [38].

This paper presents a new low-complexity decoding algorithm for Reed-Muller (RM) and polar codes which improves the error-correction performance of SC-based decoders. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript.

2. N. Doan, **S. A. Hashemi**, M. Mondelli, W. J. Gross, "On the Decoding of Polar Codes on Permuted Factor Graphs", In: *IEEE Global Communications Conference (GLOBECOM)*, Abu Dhabi, UAE, to appear, 2018 [39].

   This paper presents a method to improve the error-correction performance of decoding polar codes by using several factor graph permutations. My contributions to this paper were to help in developing the idea, and in writing the manuscript.

3. N. Doan, **S. A. Hashemi**, W. J. Gross, "Neural Successive Cancellation Decoding of Polar Codes", In: *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Kalamata, Greece, pp. 1-5, June 2018 [40].

   This paper presents an improved SC decoder for polar codes with the use of neural networks. My contribution to this paper was to help in writing the manuscript.

4. F. Ercan, C. Condo, **S. A. Hashemi**, W. J. Gross, "Partitioned Successive-Cancellation Flip Decoding of Polar Codes", In: *IEEE International Conference on Communications (ICC)*, Kansas City, USA, pp. 1-6, May 2018 [41].

   This paper presents an improved SC-Flip decoder for polar codes by using the partitioning technique in [48]. My contributions to this paper was to help in writing the manuscript.

5. **S. A. Hashemi**, M. Mondelli, S. H. Hassani, R. Urbanke, W. J. Gross, "Partitioned List Decoding of Polar Codes: Analysis and Improvement of Finite Length Performance", In: *IEEE Global Communications Conference (GLOBECOM)*, Singapore, pp. 1-7, December 2017 [42].

   This paper proposes several techniques to improve the error-correction performance of the PSCL decoding algorithm. My contributions to this paper were to develop the idea,

implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 5.

6. **S. A. Hashemi**, C. Condo, F. Ercan, W. J. Gross, "On the Performance of Polar Codes for 5G eMBB Control Channel", In: *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Pacific Grove, USA, pp. 1764-1768, October 2017 [43].

   This paper evaluates the performance of polar codes in 5G. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 7.

7. F. Ercan, C. Condo, **S. A. Hashemi**, W. J. Gross, "On Error-Correction Performance and Implementation of Polar Code List Decoders for 5G", In: *Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, USA, pp. 443-449, October 2017 [44].

   This paper presents a study on the performance of SCL decoders for polar codes for 5G. My contribution to this paper was to help in writing the manuscript.

8. C. Condo, **S. A. Hashemi**, W. J. Gross, "Efficient Bit-Channel Reliability Computation for Multi-Mode Polar Code Encoders and Decoders", In: *IEEE International Workshop on Signal Processing Systems (SiPS)*, Lorient, France, pp. 1-6, October 2017 [26].

   This paper presents a method to efficiently construct polar codes. My contribution to this paper was to help in writing the manuscript.

9. **S. A. Hashemi**, C. Condo, W. J. Gross, "Fast Simplified Successive-Cancellation List Decoding of Polar Codes", In: *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, San Francisco, USA, pp: 1-6, March 2017 [45].

   This paper presents a fast decoding algorithm for polar codes which is guaranteed to preserve the error-correction performance with respect to SCL decoding. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 4.

10. **S. A. Hashemi**, C. Condo, W. J. Gross, "Simplified Successive-Cancellation List Decoding of Polar Codes", In: *IEEE International Symposium on Information Theory (ISIT)*, Barcelona, Spain, pp: 815-819, July 2016 [46].

    This paper presents a fast decoding algorithm for polar codes which is guaranteed to preserve the error-correction performance with respect to SCL decoding. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 4.

11. **S. A. Hashemi**, C. Condo, W. J. Gross, "Matrix Reordering for Efficient List Sphere Decoding of Polar Codes", In: *IEEE International Symposium on Circuits and Systems (ISCAS)*, Montréal, Canada, pp: 1730-1733, May 2016 [47].

    This paper presents a technique to improve the error-correction performance of the method presented in [25]. My contributions to this paper were to develop the idea, implement the idea in software and hardware and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 3.

12. **S. A. Hashemi**, A. Balatsoukas-Stimming, P. Giard, C. Thibeault, W. J. Gross, "Partitioned Successive-Cancellation List Decoding of Polar Codes", In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, China, pp: 957-960, March 2016 [48].

    This paper presents the PSCL decoding algorithm for polar codes which significantly reduces the memory requirements of SCL decoding. My contributions to this paper were to develop the idea, implement the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 5.

13. **S. A. Hashemi**, C. Condo, W. J. Gross, "List Sphere Decoding of Polar Codes", In: *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Pacific Grove, USA, pp: 1346-1350, November 2015 [25].

    This paper presents a low-complexity decoding algorithm for polar codes based on sphere decoding. My contributions to this paper were to develop the idea, implement

the idea in software and test it, produce the results, and write the manuscript. My contributions to this paper are presented in Chapter 3.

1.3 Thesis Organization

This thesis is organized in nine chapters. Chapter 2 provides some preliminary material about polar codes and their decoding algorithms. Chapter 3 introduces the List-SD algorithm which is mostly suited for short polar codes. In Chapter 4, fast list decoders for polar codes are introduced and their hardware implementation is presented. Chapter 5 proposes a number of techniques to reduce the memory requirements of SC-based decoders and hardware implementation results are provided to verify the applicability of the proposed techniques. Chapter 6 solves the rate-flexibility issue with fast list decoders and proposes a hardware-friendly method to make fast list decoders completely rate-flexible. The performance of polar codes in 5G is evaluated in Chapter 7. In Chapter 8, polar codes were used in a blind detection scheme which is required by 5G and it is shown that blind detection with polar codes can achieve the requirements of 5G. Finally, the main conclusions of this thesis are presented and some research directions for future work are suggested in Chapter 9.

Chapter 2

Background

2.1 Polar Codes

Polar codes are linear block codes of length $N = 2^n$ constructed by concatenating two polar codes of length $\frac{N}{2}$. Let us consider $\mathbf{u} = \{u_0, u_1, \ldots, u_{N-1}\}$ as the input vector of bits and $\mathbf{x} = \{x_0, x_1, \ldots, x_{N-1}\}$ as the encoded vector of bits. A polar code $\mathcal{P}(N, K)$ with K information bits and rate $R = \frac{K}{N}$ is constructed by finding K most reliable bits in $\mathbf{u}$ and assigning the information bits to them. The remaining $N - K$ bits are fixed to a predetermined value and thus called frozen bits, whose set is $\mathcal{F}$. These bits can be regarded as the code parity-check bits that help the decoder find the likeliest codeword. In addition, since their values are known a priori by the decoder, they do not need to be decoded. Thus, the chance of decoding error for the least reliable bits is reduced as well. This classification is fed into the decoder as a sequence of binary values $\mathbf{s} = \{s_0, s_1, \ldots, s_{N-1}\}$ where

$$s_i = \begin{cases} 0 & \text{if } u_i \in \mathcal{F}, \\ 1 & \text{otherwise.} \end{cases} \tag{2.1}$$

More formally, let W be a Binary Memoryless Symmetric (BMS) channel with input alphabet $\mathcal{X} = \{0, 1\}$ and output alphabet $\mathcal{Y}$, and let $\{W(y \mid x) : x \in \mathcal{X}, y \in \mathcal{Y}\}$ be the transition probabilities. In order to quantify the reliability of the channel W, we use the Bhattacharyya parameter $Z(W) \in [0, 1]$, that is defined as

$$Z(W) = \sum_{y \in \mathcal{Y}} \sqrt{W(y \mid 0) W(y \mid 1)}. \tag{2.2}$$

Hence, the good bit-channels are the ones that have the lowest Bhattacharyya parameter.

For symmetric channels, the values of frozen bits do not affect the error-correction performance of polar codes [5]. Therefore, they are usually set to 0. The reliabilities associated with the bit-channels can be determined either by using the Bhattacharyya parameter [5], or the direct use of probability function [49]. Polar encoding then is represented as a matrix multiplication as

$$\mathbf{x} = \mathbf{u}\mathbf{B}_N\mathbf{G}^{\otimes n}, \tag{2.3}$$

where $\mathbf{B}_N$ is the bit-reversal permutation matrix and $\mathbf{G}^{\otimes n}$ is constructed by calculating the n-th Kronecker product of the polarizing matrix $\mathbf{G} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. Since bit-reversed indexing only has data access advantage in hardware implementation [9], it is used for the decoder architecture design [11], [20] but we drop the permutation matrix $\mathbf{B}_N$ and consider the input bits in natural index order in order to simplify the algorithm notation [5]. Therefore, (2.3) can be rewritten as

$$\mathbf{x} = \mathbf{u}\mathbf{G}^{\otimes n}. \tag{2.4}$$

The encoding process in (2.4) is an n-level polarization scheme. Figure 2.1 shows a polar code encoding example for $\mathcal{P}(8, 4)$ where t denotes the polarization level and the frozen bits set contains $\{u_0, u_1, u_2, u_4\}$. The coded vector $\mathbf{x}$ is then modulated and sent through the channel. Throughout this thesis, we consider Binary Phase-Shift Keying (BPSK) modulation, which maps the coded bits $\mathbf{x} \in \{0, 1\}^N$ to the values $\mathbf{s} \in \{+1, -1\}^N$, and AWGN channel.

The received vector $\mathbf{y} = \{y_0, y_1, \ldots, y_{N-1}\}$ can be represented as

$$\mathbf{y} = \mathbf{s} + \boldsymbol{\nu}, \tag{2.5}$$

where $\boldsymbol{\nu} = \{\nu_0, \nu_1, \ldots, \nu_{N-1}\}$ is the AWGN vector with mean 0 and variance σ^2. For AWGN channel, the vector of LLR values $\boldsymbol{\alpha} = \{\alpha_0, \alpha_1, \ldots, \alpha_{N-1}\}$ associated with the received vector can be calculated as

$$\boldsymbol{\alpha} = \frac{2}{\sigma^2}\mathbf{y}. \tag{2.6}$$

Since the scaling factor in (2.6) does not have an impact on the decoding process, we drop it for simplicity and consider $\boldsymbol{\alpha} = \mathbf{y}$. The vector $\boldsymbol{\alpha}$ is then fed to the decoder for decoding.

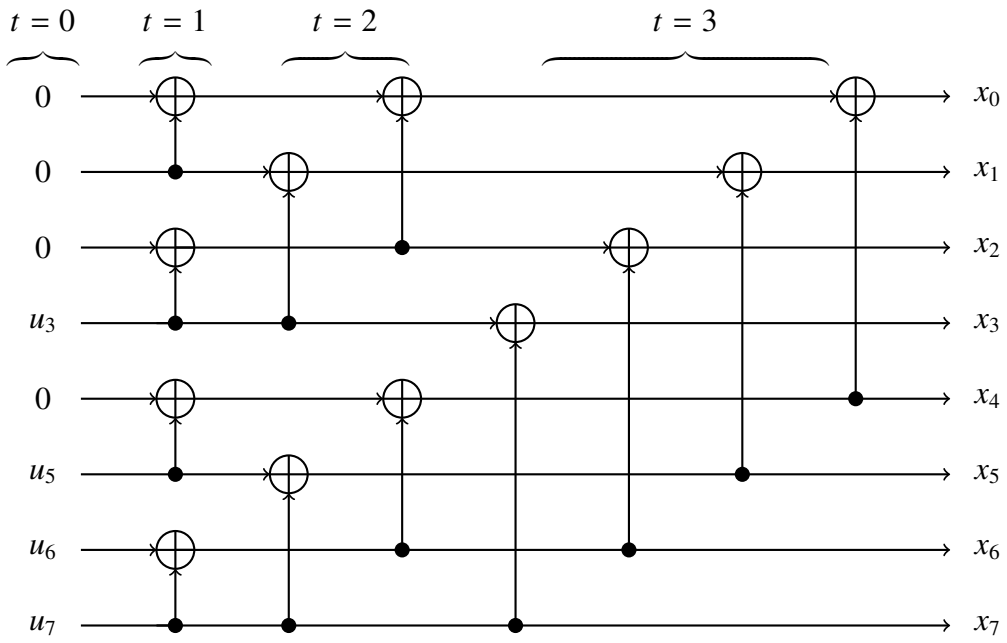


Figure 2.1: Polar code encoding example for $\mathcal{P}(8,4)$ and $\{u_0, u_1, u_2, u_4\} \in \mathcal{F}$.

2.2 Successive-Cancellation Decoding

SC decoding can be represented on a binary tree. An example for $\mathcal{P}(8,4)$ and the frozen bits set $\{u_0, u_1, u_2, u_4\}$ is shown in Figure 2.2. The vertex node at level $t = n$ of the decoding tree is fed with the LLR values received from the channel. Subsequently at each stage t of the decoding tree, the soft LLR values $\boldsymbol{\alpha} = \{\alpha_0, \alpha_1, \ldots, \alpha_{2^t-1}\}$ are passed from a parent node to its child nodes and the hard bit estimates $\boldsymbol{\beta} = \{\beta_0, \beta_1, \ldots, \beta_{2^t-1}\}$ are passed from a child node to its parent node. The elements of the child messages $\boldsymbol{\alpha}^{\mathrm{l}} = \{\alpha^{\mathrm{l}}_0, \alpha^{\mathrm{l}}_1, \ldots, \alpha^{\mathrm{l}}_{2^{t-1}-1}\}$ and $\boldsymbol{\alpha}^{\mathrm{r}} = \{\alpha^{\mathrm{r}}_0, \alpha^{\mathrm{r}}_1, \ldots, \alpha^{\mathrm{r}}_{2^{t-1}-1}\}$, a vector of 2^{t-1} values, are calculated as

$$\alpha^{\mathrm{l}}_i = \ln\left(\frac{1 + \mathrm{e}^{\alpha_i + \alpha_{i+2^{t-1}}}}{\mathrm{e}^{\alpha_i} + \mathrm{e}^{\alpha_{i+2^{t-1}}}}\right), \tag{2.7}$$

$$\alpha^{\mathrm{r}}_i = \alpha_{i+2^{t-1}} + (1 - 2\beta^{\mathrm{l}}_i)\alpha_i, \tag{2.8}$$

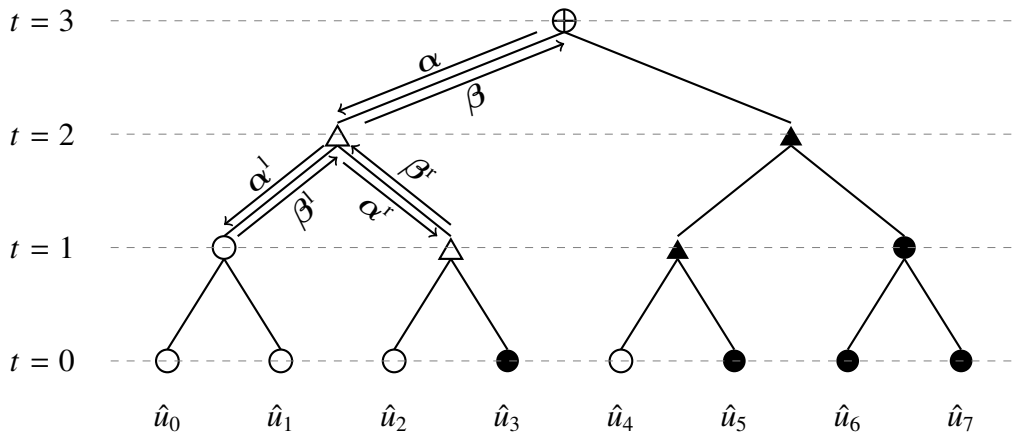


Figure 2.2: SC decoding example for $\mathcal{P}(8,4)$ and $\{u_0, u_1, u_2, u_4\} \in \mathcal{F}$.

while the elements of $\boldsymbol{\beta}$, a vector of 2^t bits, are computed using the child messages $\boldsymbol{\beta}^{\mathrm{l}} = \{\beta_0^{\mathrm{l}}, \beta_1^{\mathrm{l}}, \ldots, \beta_{2^{t-1}-1}^{\mathrm{l}}\}$ and $\boldsymbol{\beta}^{\mathrm{r}} = \{\beta_0^{\mathrm{r}}, \beta_1^{\mathrm{r}}, \ldots, \beta_{2^{t-1}-1}^{\mathrm{r}}\}$ as

$$\beta_i = \begin{cases} \beta_i^{\mathrm{l}} \oplus \beta_i^{\mathrm{r}}, & \text{if} \quad i < 2^{t-1}, \\ \beta_i^{\mathrm{r}}, & \text{otherwise,} \end{cases} \tag{2.9}$$

where $\oplus$ denotes the bitwise XOR and the interval $i < 2^{t-1}$ discriminates between bits considered by the left and the right child. At a leaf node, the i-th bit $\hat{u}_i$ is estimated as

$$\hat{u}_i = \begin{cases} 0, & \text{if } i \in \mathcal{F} \text{ or } \alpha_i^0 \geq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{2.10}$$

where α_i^t denotes the LLR value at stage t.

Due to the data dependencies, each node receives $\boldsymbol{\alpha}$ first, then sends $\boldsymbol{\alpha}^{\mathrm{l}}$, receives $\boldsymbol{\beta}^{\mathrm{l}}$, sends $\boldsymbol{\alpha}^{\mathrm{r}}$, receives $\boldsymbol{\beta}^{\mathrm{r}}$, and finally sends $\boldsymbol{\beta}$, in this order. A hardware-friendly version of (2.7), proposed in [9], can be written as

$$\alpha_i^{\mathrm{l}} = \operatorname{sgn}(\alpha_i) \operatorname{sgn}(\alpha_{i+2^{t-1}}) \min(|\alpha_i|, |\alpha_{i+2^{t-1}}|) \tag{2.11}$$

where

$$\operatorname{sgn}(\alpha_i) = \begin{cases} -1, & \text{if } \alpha_i < 0, \\ 1, & \text{otherwise.} \end{cases} \tag{2.12}$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13
α_0^2						β_0^2	α_4^2						
α_1^2						β_1^2	α_5^2						
α_2^2						β_2^2	α_6^2						
α_3^2						β_3^2	α_7^2						
	α_0^1		β_0^1	α_2^1				α_4^1		β_4^1	α_6^1		
	α_1^1		β_1^1	α_3^1				α_5^1		β_5^1	α_7^1		
		α_0^0	α_1^0		α_2^0	α_3^0			α_4^0	α_5^0		α_6^0	α_7^0
		0	0		0	$\hat{u}_3$			0	$\hat{u}_5$		$\hat{u}_6$	$\hat{u}_7$

Figure 2.3: SC scheduling for $\mathcal{P}(8, 4)$.

In SC decoding, each bit estimation is dependent on the value of all the previous bits. Thus, the conventional SC decoder can complete the decoding process in $2N - 2$ time steps, with the scheduling depicted in Figure 2.3. It can be seen that three sets of memory are required to decode polar codes using SC: the channel LLR memory, the internal LLR memory, and the β memory. Let us consider the channel LLR values are quantized with Q_{α_c} bits and the internal LLR values are quantized with Q_{α_l} bits. Since the each β value is represented with one bit, the total memory requirements for a SC decoder can be calculated as

$$M_{\text{SC}} = NQ_{\alpha_c} + (N - 1)\, Q_{\alpha_l} + N - 1. \tag{2.13}$$

2.3 Fast Simplified Successive-Cancellation Decoding

In SC decoding, the estimation of each bit requires the knowledge of previously estimated bits. The serial nature of this bit by bit estimation process limits the speed of SC decoding algorithm. In order to increase the speed of SC, [10] and [11] observed four different constituent codes: their

decoding can be carried out in a more efficient way than fully exploring the decoding tree.

- *Rate-0 Node*: This node consists of only frozen bits, i.e., $\mathbf{s} = \{0, 0, \ldots, 0\}$.
- *Rate-1 Node*: This node consists of only information bits, i.e., $\mathbf{s} = \{1, 1, \ldots, 1\}$.
- *Rep Node*: This node consists of frozen bits except for the last bit which is an information bit, i.e., $\mathbf{s} = \{0, \ldots, 0, 0, 1\}$.
- *SPC Node*: This node consists of information bits except for the first bit which is a frozen bit, i.e., $\mathbf{s} = \{0, 1, 1, \ldots, 1\}$.

These codes are shown in Figure 2.2. White circles are Rate-0 nodes and black circles are Rate-1 nodes. All leaf nodes at $t = 0$ are either Rate-0 or Rate-1, being either single frozen bits or single information bits. White triangles represent Rep nodes and black triangles represent SPC nodes. Note that SPC and Rep nodes at $t = 1$ are equivalent.

Since Rate-0 nodes correspond to frozen bits, they can be estimated as

$$\beta_i = 0. \tag{2.14}$$

Rate-1 nodes correspond to all-information bit vectors. A direct hard estimate based on the LLR values at a Rate-1 node can be carried out as

$$\beta_i = \frac{1}{2}\left(1 - \operatorname{sgn}\left(\alpha_i\right)\right). \tag{2.15}$$

Rep nodes have one information bit at the leaf branches and the bit estimate of this information bit is the bit estimate of all the bits in a Rep node. The LLR value of the information bit in a Rep node can be calculated as $\sum_{i=0}^{2^t-1} \alpha_i$. Therefore, the bit estimates at a Rep node can be found as

$$\beta_i = \frac{1}{2}\left(1 - \operatorname{sgn}\left(\sum_{i=0}^{2^t-1} \alpha_i\right)\right). \tag{2.16}$$

SPC nodes have the even-parity constraint. In SC decoding, this constraint is imposed by checking the hard estimate of the least reliable bit. If this bit does not satisfy the even-parity constraint, then

it is flipped. The least reliable bit in an SPC node is found as

$$i_{\min} = \underset{0 \le i < 2^t}{\arg\min}(|\alpha_i|), \tag{2.17}$$

and the parity of it is derived as

$$\gamma = \bigoplus_{i=0}^{2^t-1} \left(\frac{1}{2} \left(1 - \operatorname{sgn}(\alpha_i)\right) \right). \tag{2.18}$$

Finally, the bit estimate in an SPC node is calculated as

$$\beta_i = \begin{cases} \left(\frac{1}{2} \left(1 - \operatorname{sgn}(\alpha_i)\right) \right) \oplus \gamma, & \text{if} \quad i = i_{\min}, \\ \frac{1}{2} \left(1 - \operatorname{sgn}(\alpha_i)\right), & \text{otherwise.} \end{cases} \tag{2.19}$$

The advantage of Fast-SSC is that the decoder is optimal, i.e. it performs exactly as the original SC decoder. Since parts of the code can be decoded in parallel, Fast-SSC can decode a received vector with a much higher speed than SC.

Recently, five new special nodes are observed in [50] and efficient decoders that can be used in SC decoding were designed for them. These nodes are:

- *Type-I Node*: This node consists of frozen bits except for the last two bits which are information bits, i.e., $\mathbf{s} = \{0, \ldots, 0, 1, 1\}$.
- *Type-II Node*: This node consists of frozen bits except for the last three bits which are information bits, i.e., $\mathbf{s} = \{0, \ldots, 0, 1, 1, 1\}$.
- *Type-III Node*: This node consists of information bits except for the first two bits which are frozen bits, i.e., $\mathbf{s} = \{0, 0, 1, \ldots, 1\}$.
- *Type-IV Node*: This node consists of information bits except for the first three bits which are frozen bits, i.e., $\mathbf{s} = \{0, 0, 0, 1, \ldots, 1\}$.
- *Type-V Node*: This node consists of frozen bits except for the bits $N-5$, $N-3$, $N-2$, and $N-1$ which are information bits, i.e., $\mathbf{s} = \{0, \ldots, 0, 1, 0, 1, 1, 1\}$.

The pruned decoding tree for the same example as in Figure 2.2 is shown in Figure 2.4. If the new nodes are not taken into account, $\mathcal{P}(8, 4)$ can be decoded in four time steps by traversing the

Table 2.1: Different operations that are supported in SC-based decoding algorithms.

Operation	Description	Decoder
F_t	Calculate $\boldsymbol{\alpha}^{\ell}$ at level t.	SC-based
G_t	Calculate $\boldsymbol{\alpha}^{\mathrm{r}}$ at level t.	SC-based
Rate-0_t	Decode Rate-0 node of length 2^t.	Fast SC-based
Rate-1_t	Decode Rate-1 node of length 2^t.	Fast SC-based
Rep_t	Decode Rep node of length 2^t.	Fast SC-based
SPC_t	Decode SPC node of length 2^t.	Fast SC-based
Type-I_t	Decode Type-I node of length 2^t.	Fast SC-based
Type-II_t	Decode Type-II node of length 2^t.	Fast SC-based
Type-III_t	Decode Type-III node of length 2^t.	Fast SC-based
Type-IV_t	Decode Type-IV node of length 2^t.	Fast SC-based
Type-V_t	Decode Type-V node of length 2^t.	Fast SC-based

tree for one level and decode the resulting Rep and SPC nodes. The resulting list of operations for the decoder would be $\{F_2, \mathrm{Rep}_2, G_2, \mathrm{SPC}_2\}$, where Rep_t and SPC_t represent the decoding of Rep and SPC nodes of length 2^t, respectively. However, by considering the new nodes, the decoder can immediately decode the received vector by decoding the Type-V node. The corresponding list of operations would be $\{\text{Type-V}_3\}$, where Type-V_t represents the decoding of Type-V nodes of length 2^t. The operations which are performed in fast SC-based decoders are summarized in Table 2.1. Note that F_t and G_t operations are common between conventional SC-based and fast SC-based decoding algorithms. In the hardware implementation of fast SC-based decoders, this list of operations is stored in memory and is fed into the decoder to perform decoding [11], [36], [37].

Let us consider the example in Figure 2.4. If the rate of the code changes from $1/2$ to $5/8$, the list of operations also changes as shown in Figure 2.5. Without using the new nodes, the list of operations becomes $\{F_2, \mathrm{Rep}_2, G_2, \text{Rate-}1_2\}$, and by considering the new nodes it becomes $\{\text{Type-IV}_3\}$. Therefore, as the rate changes, the list of operations changes. The resulting decoder is therefore not rate-flexible. For applications that support codes with multiple rates, for each rate, the list of operations has to be stored in memory to make the decoder flexible. However, this results in high memory usage when implemented on hardware.

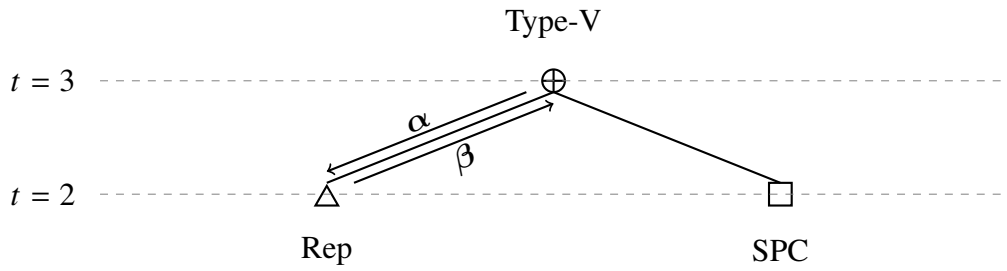


Figure 2.4: Fast SC-based decoding on a binary tree for $\mathcal{P}(8,4)$ and $\mathbf{s} = \{0,0,0,1,0,1,1,1\}$.

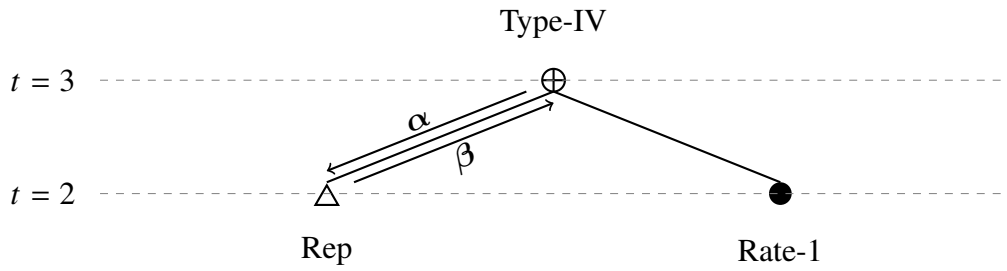


Figure 2.5: Fast SC-based decoding on a binary tree for $\mathcal{P}(8,5)$ and $\mathbf{s} = \{0,0,0,1,1,1,1,1\}$.

2.4 Successive-Cancellation List Decoding

To improve the error-correction performance of SC for codes with moderate lengths, the SCL decoding algorithm estimates a bit with both its possible values 0 and 1. In order to control the exponential increase in the complexity of this algorithm, a set of L codeword candidates is memorized at all times and every new bit estimate produces $2L$ new candidates, half of which must be discarded. To this purpose, a PM is associated to each codeword candidate (path) and updated at every new estimation: it can be considered a cost function, and the L paths with the lowest PMs are allowed to survive. In the LLR-based formulation of SCL [20], the PM can be computed as

$$\mathrm{PM}_{i_l} = \sum_{j=0}^{i} \ln\left(1 + \mathrm{e}^{-(1-2\hat{u}_{j_l})\alpha_{j_l}}\right), \tag{2.20}$$

where l is the path index, α_{j_l} is the LLR value of u_j at path l, and $\hat{u}_{j_l}$ is the estimate of bit j at path l.

From (2.20), we can also see that PM for a polar code with $N = 2^t$ can be computed as the sum of its two constituent codes with $N = 2^{t-1}$. However, it must be noted that due to the sequential

nature of SCL, to compute the PM at bit i, it is necessary to have traversed the whole decoding tree at the left of leaf node i. A hardware-friendly version of (2.20), as proposed in [20], can be expressed as

$$\begin{aligned} &\mathrm{PM}_{-1_l} = 0, \\ &\mathrm{PM}_{i_l} = \begin{cases} \mathrm{PM}_{i-1_l} + |\alpha_{i_l}|, & \text{if } \hat{u}_{i_l} \neq \frac{1}{2}\left(1 - \operatorname{sgn}\left(\alpha_{i_l}\right)\right), \\ \mathrm{PM}_{i-1_l}, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.21}$$

which can be rewritten as

$$\mathrm{PM}_{i_l} = \frac{1}{2} \sum_{j=0}^{i} \operatorname{sgn}(\alpha_{j_l})\alpha_{j_l} - (1 - 2\hat{u}_{j_l})\alpha_{j_l}. \tag{2.22}$$

SCL decoding process can be illustrated by an example as in Figure 2.6 for $\mathcal{P}(4, 3)$ and $L = 2$. In this example, u_0 is the least reliable bit and is frozen to 0, while information bits are carried by the more reliable bits u_1, u_2 and u_3. SCL starts by performing SC decoding until it reaches bit $\hat{u}_0$. Since it is a frozen bit, it is estimated as 0. It then continues the SC decoding process until it reaches bit $\hat{u}_1$. It estimates $\hat{u}_1$ as either 0 or 1 and calculates the corresponding PMs. It continues by performing SC until it reaches bit $\hat{u}_2$. $\hat{u}_2$ is estimated as either 0 or 1 and based on the resulting PMs, the two best candidates out of the resulting four are kept and the two worst are discarded. Surviving candidates are shown as black squares, while solid gray squares are estimated but then discarded. A same approach is carried out for $\hat{u}_3$ and the candidate with the minimum PM is selected as $\hat{\mathbf{u}}$.

The error-correction performance of SCL decoding is bounded by the ML performance. It was observed in [12] that although the correct estimate might not be the candidate with the best PM, the correct candidate usually falls in the list of candidates which survive the decoding process. To determine the correct candidate, a CRC was concatenated to the code to help SCL find the correct candidate. The use of CRC improved the performance of SCL to the extent that the CRC-aided SCL can outperform state of the art codes such as LDPC codes [12].

The memory requirements of SCL (CRC-aided SCL) decoding are higher than that of the SC decoding since L paths of the candidate codewords need to be stored. In addition, the PM for each path also needs to be stored. Let us consider Q_{PM} bits are used to store the PMs. The memory

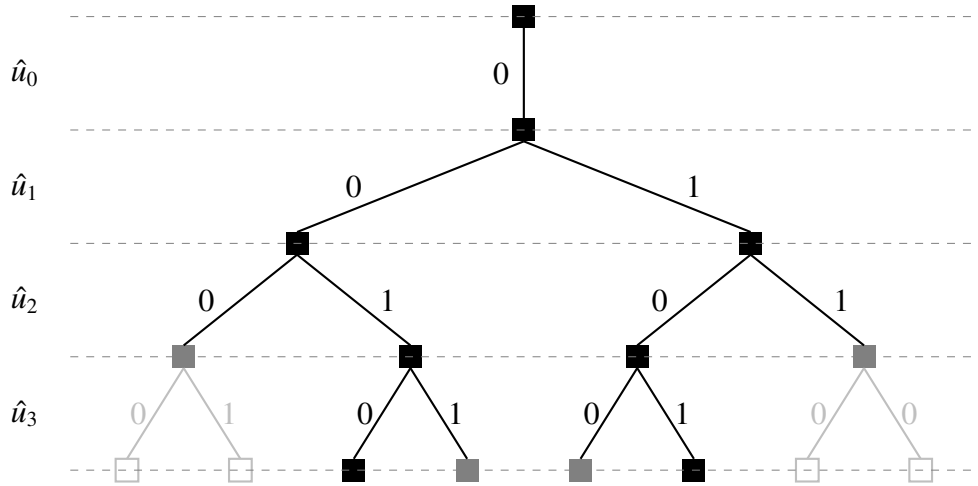


Figure 2.6: Binary tree representation of SCL on $\mathcal{P}(4,3)$, with $L = 2$ and $u_0 \in \mathcal{F}$. Surviving candidates are shown as black squares. Visited candidates are solid gray squares. White squares are not visited.

requirement of a SCL decoder can be written as

$$M_{\text{SCL}} = NQ_{\alpha_C} + L\,(N-1)\,Q_{\alpha_I} + LQ_{\text{PM}} + L\,(2N-1)\,. \tag{2.23}$$

It should be noted that in SCL decoding, LN bits need to be stored for the final codeword candidates as opposed to SC decoding.

2.5 Sphere Decoding

The SD algorithm tries to obtain the ML estimate of the transmitted signal by considering only the candidates inside a N-dimensional sphere of radius r and center $\mathbf{y}$. The ML estimate can be found by solving the minimization problem in

$$\hat{\mathbf{u}} = \arg\min_{\mathbf{s}} ||\mathbf{y} - \mathbf{s}||^2 = \arg\min_{\mathbf{u}} ||\mathbf{y} - (\mathbf{1} - 2\mathbf{u}\mathbf{G}^{\otimes n})||^2, \tag{2.24}$$

where $\mathbf{1}$ is the all-ones vector of length N.

SD exploits the the lower triangular structure of $\mathbf{G}^{\otimes n}$ to determine the order of bit estimation. Thus, the algorithm starts with the estimation of u_{N-1}, considering both 0 and 1 cases. Then, it compares the Euclidean distance between y_{N-1} and s_{N-1} with the radius r for both cases. The

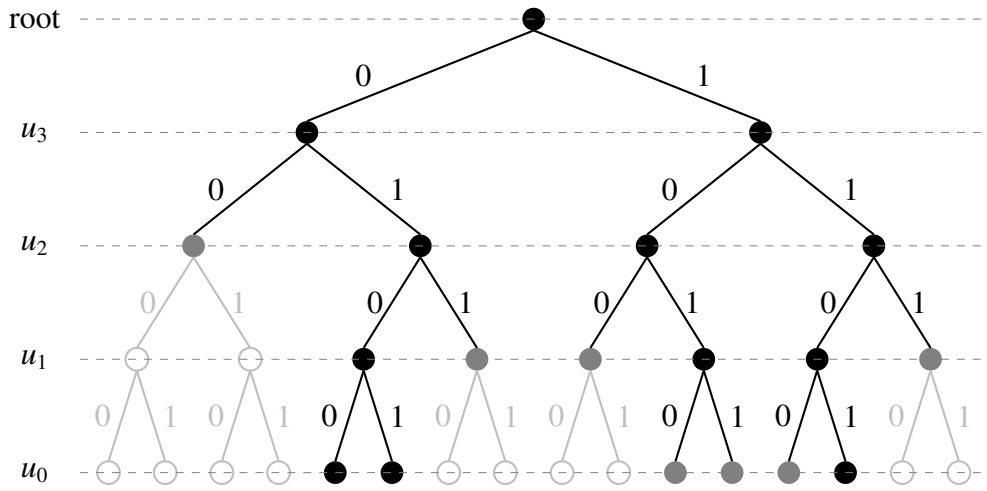


Figure 2.7: Binary tree representation of SD on $\mathcal{P}(4,4)$. Surviving candidates are shown as black circles. Visited candidates are solid gray circles. White circles are not visited.

paths of all the candidates that reside inside the sphere are allowed to survive and are used in the estimation of next bit. Otherwise, paths are discarded and never revisited. Figure 2.7 shows how SD works on a binary tree. The left branches are due to 0 bit estimations and the right branches to 1 bit estimations.

The sphere radius does not remain constant throughout the decoding process. The first candidate is found by considering the initial selection of r. After the first candidate is found, r is updated to the euclidean distance of the first candidate from $\mathbf{y}$. The second candidate is then found based on the new r, and again updating its value. This process is repeated until the candidate with the minimum euclidean distance is found. (2.25) expresses the PM used to select the candidates in terms of euclidean distance, where g_{kj} is the element in the k-th row and j-th column of $\mathbf{G}^{\otimes n}$.

$$\mathrm{PM}_i = \sum_{j=i}^{N-1} |y_j - s_j|^2 = \sum_{j=i}^{N-1} |y_j - (1 - 2\sum_{k=j}^{N-1} g_{kj} u_k)|^2. \tag{2.25}$$

An optimum version of this PM is introduced in [8] by taking the probability of bit estimations into account. However, it involves very complex calculations. A hardware friendly approximation

of the optimum PM can be written as

$$\mathrm{PM}_i = \sum_{j=i}^{N-1}(y_j s_j - |y_j|) = \sum_{j=i}^{N-1}(y_j(1 - 2\sum_{k=j}^{N-1} g_{kj}u_k) - |y_j|). \tag{2.26}$$

Chapter 3

List Sphere Decoding

SD is able to find the ML estimate of the transmitted data. However, it comes at the cost of variable time complexity, and consequently unpredictable decoding latency (except for the upper bound of visiting all the paths). Moreover, the performance of SD is highly dependent on the choice of the radius r: with a large radius, many candidates survive and the decoding latency increases. On the other hand, with a small r, SD may never be able to find the ML estimate. To overcome these issues, List-SD is proposed, forfeiting the radius and fixing the algorithm time complexity.

3.1 List Sphere Decoding Algorithm

Let us consider the estimation of each coded bit $\hat{x}_i$ based solely on the received vector as

$$\hat{x}_i = \begin{cases} 0 & \text{if } y_i \geq 0 \\ 1 & \text{if } y_i < 0 \end{cases}. \tag{3.1}$$

Moreover, let us call u'_i the information bit estimated during the decoding process, and x'_i the corresponding coded bit.

The lower triangular structure of $\mathbf{G}^{\otimes n}$ leads to the transmitted bit x_{N-1} being equal to u_{N-1}. Therefore, if x'_{N-1} and $\hat{x}_{N-1}$ do not match, the corresponding path is penalized. Once u'_{N-1} has been estimated, the algorithm moves to u'_{N-2}, calculates the corresponding coded bit x'_{N-2} and compares it with $\hat{x}_{N-2}$, updating the PM accordingly. The process is repeated until u'_0 has been reached. At

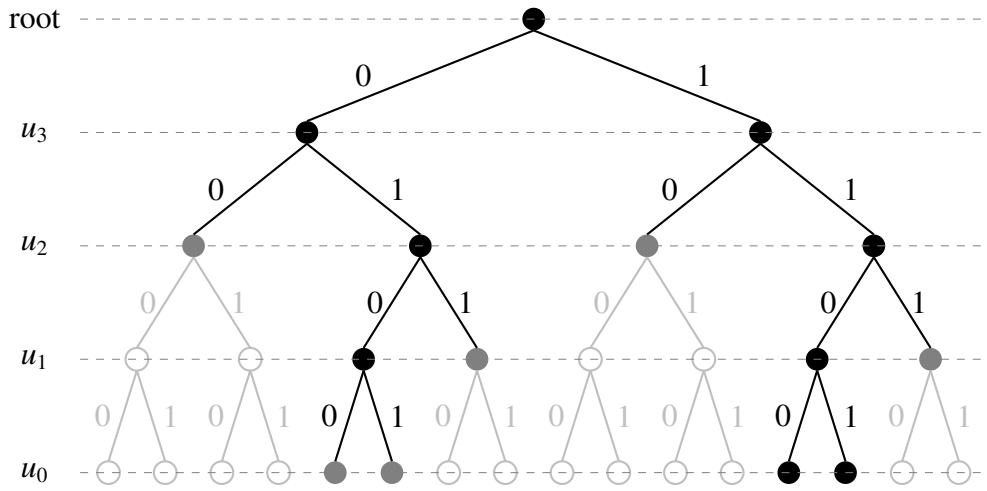


Figure 3.1: List-SD on a binary tree for $\mathcal{P}(4, 4)$ and with list size $L = 2$.

each step m and for each path l, the PM is updated in accordance with

$$\begin{aligned} \mathrm{PM}_{0_l} &= 0 \\ \mathrm{PM}_{m_l} &= \begin{cases} \mathrm{PM}_{m-1_l} + |y_{N-m}| & \text{if } x'_{N-m} \neq \hat{x}_{N-m} \\ \mathrm{PM}_{m-1_l} & \text{if } x'_{N-m} = \hat{x}_{N-m} \end{cases}. \end{aligned} \tag{3.2}$$

The proposed List-SD algorithm has a fixed time complexity. Every time a new bit is estimated, the number of considered paths doubles. Their number is however maintained constant by keeping only the L paths with the lower PMs, and discarding the remaining L. No radius is involved. Figure 3.1 shows List-SD with list size $L = 2$. In the end, the path with the minimum PM is chosen as the final candidate. It should be noted that the PM update must be performed on both information and frozen bits, therefore, the List-SD algorithm takes N time steps to finish. The List-SD algorithm is summarized in Algorithm 1.

List-SD has potentially lower memory requirements than SC and SCL. While SC-based decoding algorithms need the whole received vector to start the decoding process, List-SD only needs one LLR at a time. Thus, supposing to serially receive $\mathbf{y}$ from y_{N-1} to y_0, there is no need to allocate memory for channel LLR values. The surviving candidates and their associated PMs do need to be

Input: $\mathbf{y}$
Output: $\mathbf{u}'$
Initialize $\mathrm{PM}_{0_l} = 0$ for all $0 \leq l < 2L$
Initialize $\hat{x}_i$ for all $0 \leq i < N$ with (3.1)
for $m \leftarrow 1$ ***to*** N **do**
  for $l \leftarrow 0$ ***to*** $L - 1$ **do**
    $u'^{l}_{N-m} = 0$
    $x'^{l}_{N-m} = \sum_{k=N-m}^{N-1} g_{k(N-m)} u'^{l}_{k}$ Modulo-2
    if $x'^{l}_{N-m} \neq \hat{x}_{N-m}$ **then**
      $\mathrm{PM}_{m_l} = \mathrm{PM}_{m-1_l} + |y_{N-m}|$
    else
      if $u'^{l}_{N-m} \notin \mathcal{F}$ **then**
        $\mathrm{PM}_{m_{l+L}} = \mathrm{PM}_{m-1_{l+L}} + |y_{N-m}|$
        $u'^{l}_{N-m} = 1$
      else
        $\mathrm{PM}_{m_{l+L}} = \mathrm{PM}_{m_l}$
      end
    end
  end
  Sort(PM_{m_0},. . .,$\mathrm{PM}_{m_{2L-1}}$) and store L smallest in PM_{m_0},. . .,$\mathrm{PM}_{m_{L-1}}$.
end
Result: $\mathbf{u}'^{N-1}_{0}$ with the smallest PM.

Algorithm 1: The List-SD algorithm

stored between decoding steps, resulting in the total memory requirements represented by

$$M_{\text{List-SD}} = NL + LQ_{\text{PM}}. \tag{3.3}$$

3.2 Matrix Reordering for List-SD

The error-correction performance and complexity of List-SD can be tweaked through its list size L. Enhanced FER and Bit Error Rate (BER) require larger L values and thus more memory and logic, while a small L leads to lower hardware complexity and worse error-correction performance. In this section, we propose a method to improve this trade-off by reducing the complexity of List-SD without degrading its performance.

Frozen bits in the PM calculation play an important role in the identification of the best paths [20]. Their location is chosen according to the reliability of each bit-channel, and this is in turn

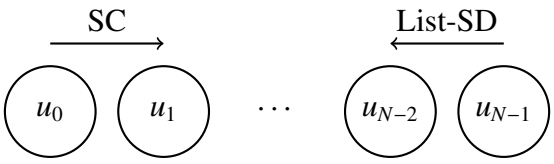


Figure 3.2: Decoding direction for SC and List-SD.

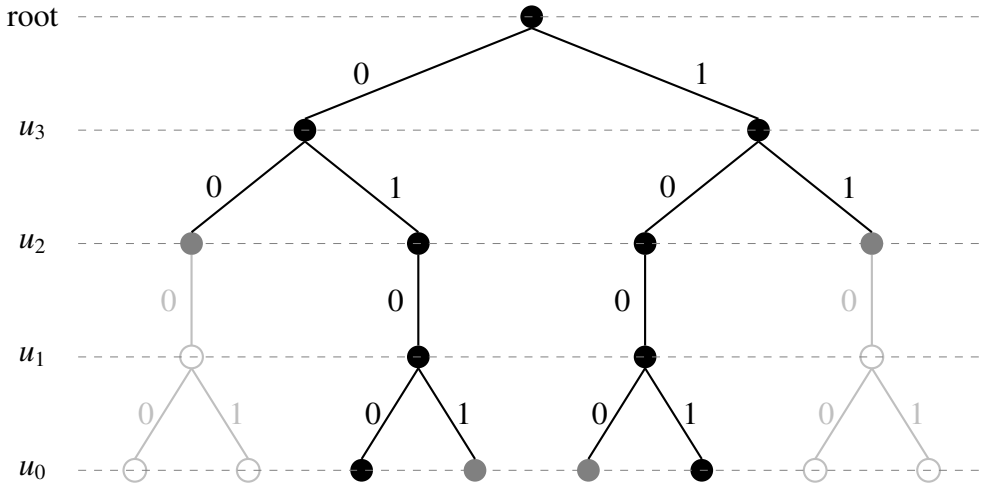


Figure 3.3: Binary tree representation of List-SD on $\mathcal{P}(4, 3)$, with $L = 2$ and frozen u_1. Surviving candidates are shown as black circles. Visited candidates are solid gray circles. White circles are not visited.

dependent on the decoding algorithm. Polar codes are constructed targeting the SC algorithm. The majority of frozen bits is in fact located in the leftmost part of the code, i.e. among the left columns of the $\mathbf{G}^{\otimes n}$ matrix, that represent the first bits to be estimated. This is because bit i is dependent on the correct estimation of bits 0 to $i - 1$. On the contrary, List-SD begins the decoding process from the rightmost column of the code. Staring from bit $N - 1$, List-SD calculates the probability of error bit by bit until it reaches bit 0, and each bit i is dependent on the correct estimate of bits $i + 1$ to $N - 1$. Therefore, to be maximally useful, frozen bits should be located in the rightmost part of the code starting from location $N - 1$. This is illustrated in Figure 3.2.

We thus propose a matrix reordering technique that allows to reach the left part of the code faster, so that List-SD can exploit the beneficial positioning of frozen bits. When u_{N-1} is estimated, bits $u_{N-(2^k+1)}$, where $0 \leq k < n$, depend only on u_{N-1}. We then choose the leftmost bit among them, i.e. $u_{N-(2^{n-1}+1)}$, to be estimated next. The next step will estimate the leftmost bit among those

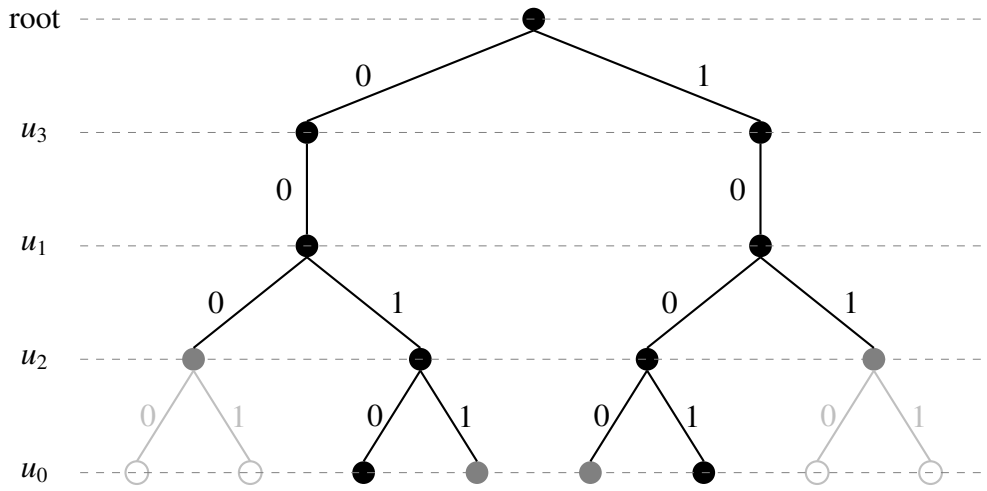


Figure 3.4: Binary tree representation of List-SD on a reordered matrix for $\mathcal{P}(4,3)$, for $L = 2$ and frozen bit u_1.

depending only on those already estimated, i.e. $u_{N-(2^{n-2}+1)}$. This process is repeated until all bits have been decoded. This modified decoding order can be represented as a reordered matrix $\mathbf{G}_r^{\otimes n}$. For example, $\mathbf{G}^{\otimes 3}$ can be represented as

$$\mathbf{G}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3.4}$$

The bit decoding order according to the proposed technique would be $\{7, 3, 5, 1, 6, 2, 4, 0\}$, that can

be translated as the matrix with reordered columns as

$$\mathbf{G}_r^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3.5}$$

List-SD then estimates bits on the reordered matrix from the rightmost to the leftmost column. Let us consider List-SD of $\mathcal{P}(4,3)$ with u_1 being the frozen bit. Figure 3.3 shows the binary tree representation of List-SD while Figure 3.4 portrays the same example, but with a reordered matrix. It can be noticed how the frozen bit u_1 is decoded before u_2.

It is worth mentioning that even though we do not discuss CRC-aided SCL decoding, the structure of both List-SD and its matrix reordered version are, just like SCL, suitable for the inclusion of CRC to help the selection of the best path.

3.3 Simulation Results

In this section, the proposed List-SD algorithm is used to decode short polar codes. Figures 3.5-3.7 shows a comparison between the FER and BER performance of the proposed List-SD against SC and SCL algorithms. In this figure, List-SD(L) and SCL(L) denote the List-SD and SCL algorithms with list size L. Figure 3.5 plots the results for $\mathcal{P}(8,5)$ code: it can be seen that List-SD(4) and List-SD(8) respectively match and outperform both SC and SCL(2). Figure 3.6 depicts FER and BER curves obtained with $\mathcal{P}(16,12)$ code. While List-SD(8) is very close in performance with both SC and SCL(2), List-SD(16) is required to match their error-correction performance. Finally, the performance of $\mathcal{P}(64,57)$ is illustrated in Figure 3.7, where both List-SD(16) and List-SD(32) outperform as good as or better than SC and SCL(2).

The effect of the proposed matrix reordering technique on the error-correction performance of List-SD is also evaluated in Figures 3.5-3.7. We can see both the improved error-correction performance in case of equal complexity (same L) and the reduced L with which FER and BER can be

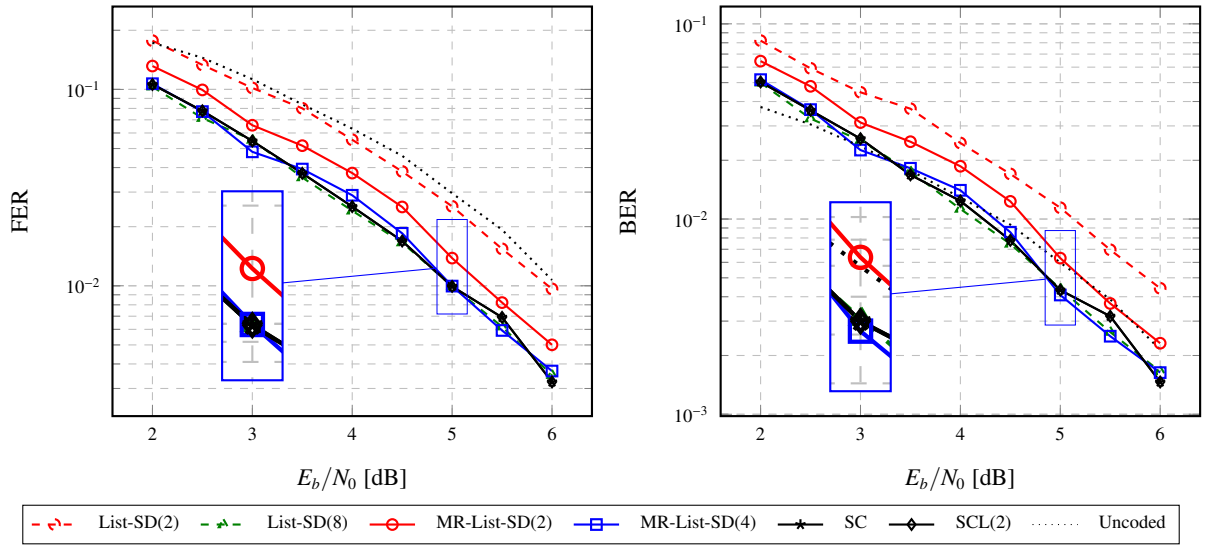


Figure 3.5: FER and BER performance results for List-SD of $\mathcal{P}(8, 5)$.

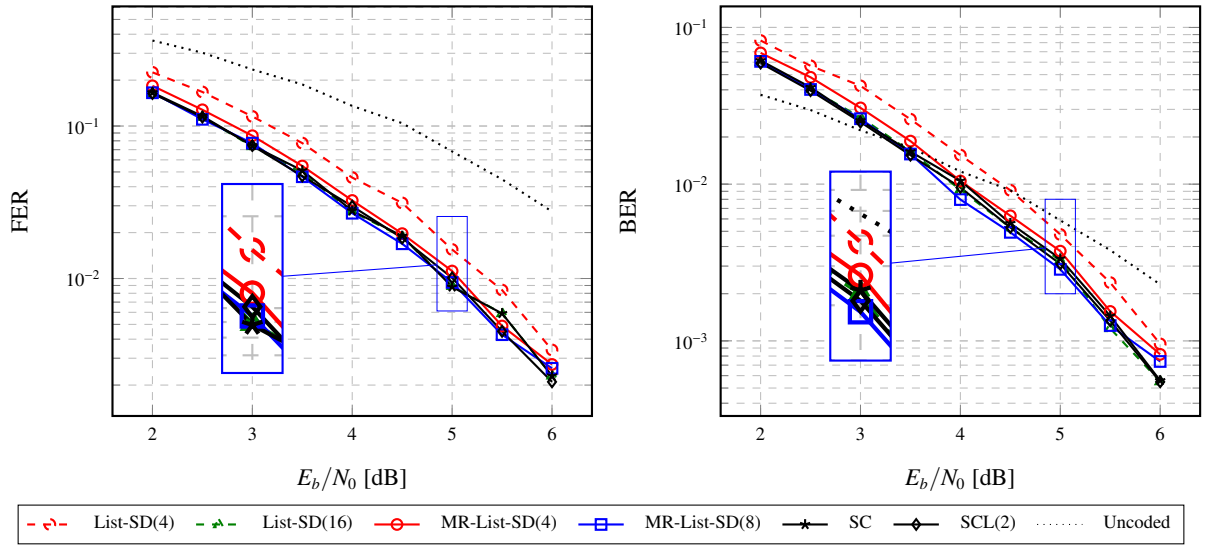


Figure 3.6: FER and BER performance results for List-SD of $\mathcal{P}(16, 12)$.

matched. In these figures, MR-List-SD(L) refers to the Matrix Reordered version. It can be seen that for $\mathcal{P}(8, 5)$, MR-List-SD(2) improves the error-correction performance of List-SD(2) of approximately 0.75 dB on both FER and BER, while MR-List-SD(4) performs as well as List-SD(8). For $\mathcal{P}(16, 12)$, the improvement of MR-List-SD(4) over its non-reordered version is approximately 0.4 dB, and MR-List-SD(8) is sufficient to match the performance of List-SD(16). Finally for

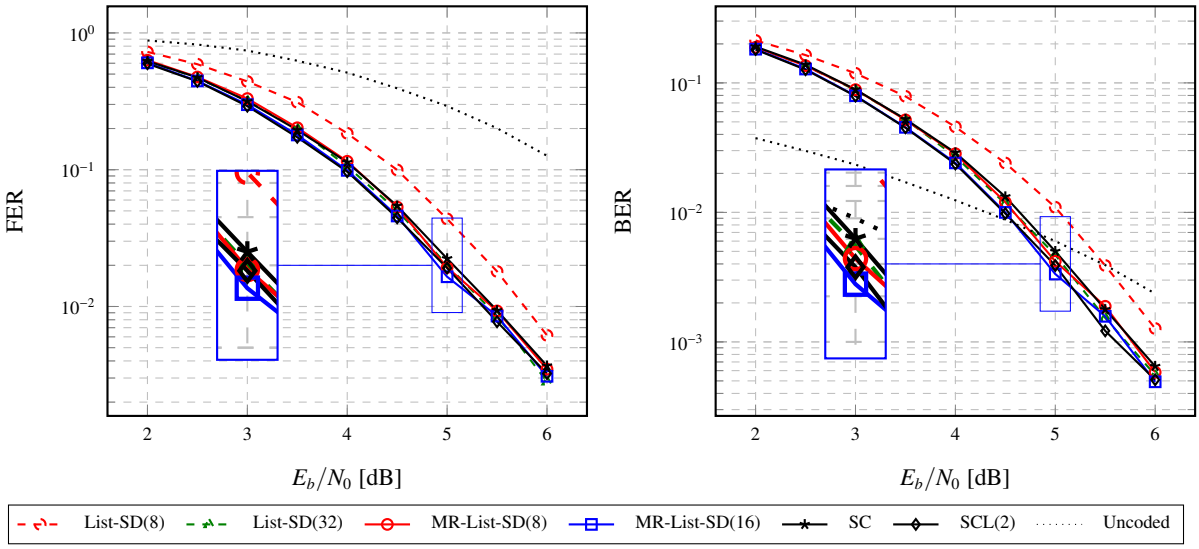


Figure 3.7: FER and BER performance results for List-SD of $\mathcal{P}(64, 57)$.

Table 3.1: Memory requirements for $\mathcal{P}(8, 5)$

Algorithm	SC	SCL(2)	List-SD(2)	List-SD(4)	List-SD(8)
Memory Bits	97	178	32	64	128

$\mathcal{P}(64, 57)$, MR-List-SD(8) improves FER and BER performance of List-SD(8) of about 0.5 dB on average, and MR-List-SD(16) performs slightly better than List-SD(32).

The memory requirements in accordance with (2.13), (2.23), and (3.3) for SC, SCL, and List-SD under the same conditions as the curves in Figures 3.5-3.7 are summarized in Tables 3.1-3.3. In all considered cases, $Q_{\alpha_C} = Q_{\alpha_I} = 6$ and $Q_{\text{PM}} = 8$. As with SCL, the choice of L in List-SD imposes a trade-off between error-correction performance and memory usage. However, for short polar codes, List-SD is able to match the FER and BER of SC and SCL(2) with a lower number of memory bits. This is particularly evident in Table 3.1, where even with the largest considered L, List-SD requires less memory bits than SCL(2), while the intermediate L value leads to less memory bits than SC. In case of Table 3.2, List-SD(8) both outperforms and requires less memory bits than SC, while the memory requirements for List-SD(16) are slightly higher than SCL(2). With the largest considered code, and thus with List-SD requiring larger L values, the memory requirements of the proposed algorithm are higher than SC and SCL(2), as shown in Table 3.3.

Table 3.2: Memory requirements for $\mathcal{P}(16, 12)$

Algorithm	SC	SCL(2)	List-SD(4)	List-SD(8)	List-SD(16)
Memory Bits	201	354	96	192	384

Table 3.3: Memory requirements for $\mathcal{P}(64, 57)$

Algorithm	SC	SCL(2)	List-SD(8)	List-SD(16)	List-SD(32)
Memory Bits	825	1410	576	1152	2304

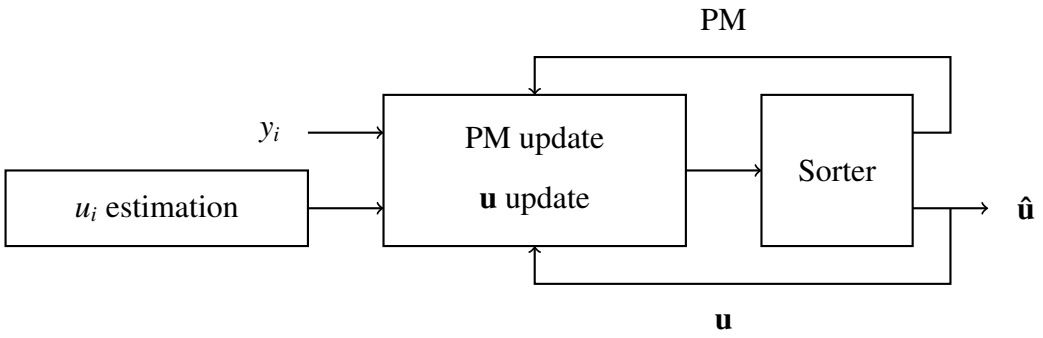


Figure 3.8: Top-level architecture of List-SD

3.4 Implementation Results

A simple architecture implementing the reordered List-SD algorithm has been synthesized on a Xilinx Virtex-7 XC7V2000T Field Programmable Gate Array (FPGA). Figure 3.8 shows the block diagram of the architecture implementing List-SD. The **u** estimation block forks the current paths, and the new PMs are generated following (3.2). These are joined to the L surviving paths from the previous stage and sent to the sorter block, that implements a bitonic sorting algorithm. The L surviving paths are fed back to the PM and **u** update block and stored in registers, ready for the next stage.

Table 3.4 provides the Look-Up Table (LUT) and register synthesis results for four different polar codes and for two list sizes that result in matching BER and FER. Since the matrix reordering does not change the architecture, results are valid for both List-SD and MR-List-SD. For example, while MR-List-SD(4) matches the error-correction performance of List-SD(8) for $\mathcal{P}(8, 5)$, it requires only about 32% of the LUTs and 50% of the registers. For $\mathcal{P}(16, 12)$, MR-List-SD(8) requires

Table 3.4: Synthesis results for List-SD on Xilinx Virtex-7 FPGA.

Polar Code	List Size L	LUTs	Registers
$\mathcal{P}(8,5)$	2	405	32
	4	1446	64
	8	4462	128
$\mathcal{P}(16,12)$	4	1550	96
	8	5387	192
	16	19597	384
$\mathcal{P}(32,26)$	8	6949	320
	16	22660	640
$\mathcal{P}(64,57)$	8	9506	576
	16	31340	1152
	32	91778	2304

27% of the LUTs and 50% of the registers needed by List-SD(16). For $\mathcal{P}(64,57)$, MR-List-SD(16) requires 34% of the LUTs and 50% of the registers of List-SD(32) with the same error-correction performance.

Chapter 4

Fast List Decoding

The SCL decoding algorithm for polar codes shows superior error-correction performance in comparison with SC decoding. However, this error-correction performance improvement comes at the cost of higher decoding latency and lower throughput. A direct application of Fast-SSC decoding algorithm on SCL decoding is not straightforward due to the fact that in SCL, there are multiple SC decoders which are connected through a sorting network. The fast SCL decoders in state of the art are either not an exact representation of SCL decoding or they require a large sorting network to provide a high throughput. Therefore, they are either adding approximations which negatively impact the error-correction performance of the decoder, or they add to the hardware implementation complexity of the underlying decoder. In this chapter, we propose two methods which first identify the redundant calculations in the SCL decoding and then remove these redundant calculations to achieve a low latency and high throughput SCL decoder without incurring any error-correction performance loss. We further propose a number of optimization techniques which allow for negligible error-correction performance loss while providing a higher throughput and lower latency.

4.1 Simplified Successive-Cancellation List Decoding

The time complexity of SCL decoding algorithm can be represented by the number of time steps required to decode a codeword [20]. Supposing that all the parallelizable instructions are performed in one clock cycle, the time required to decode a codeword can be calculated as

$$T_{\mathrm{SCL}} = 2N + K - 2. \tag{4.1}$$

Using (4.1), Rate-1 nodes can be decoded in $3N-2$ time-steps, Rate-0 nodes can be decoded in $2N-2$ time-steps, Rep nodes can be decoded in $2N-1$ time-steps, and SPC nodes can be decoded in $3N-3$ time-steps.

In order to reduce the time complexity of SCL decoding, [19] proposed optimal simplified PM calculations for the hardware-friendly formulations of Rate-0 and Rep nodes. In this section, we propose a SSCL decoding algorithm which uses the similarities between List-SD and SCL decoding algorithms for Rate-1 nodes, also proposing optimal simplified PM calculations for them. We also extend the optimal PM calculations for the exact formulations of Rate-0 and Rep nodes. The simplified PMs rely on the LLR values at the top of the tree identified by their respective node: thus, it is not necessary to traverse the tree to correctly compute them. For all three nodes, we prove that the proposed calculations are exactly equivalent to those proposed in [20]. We further propose a modified version of List-SD to design a low-complexity decoder for SPC nodes and we show that the error-correction performance degradation due to using this decoder is negligible. We call the decoder which has simplifications for SPC nodes as SSCL-SPC. For a more readable notation, we drop the path index l and we introduce $\eta_i = 1 - 2\beta_i$.

Rate-1 Nodes

The following theorems prove that List-SD is equivalent to SCL for a Rate-1 node, both for the exact and the hardware-friendly formulations. Furthermore, for a Rate-1 node in List-SD, the bit estimation can be performed directly at the top of the tree and when all the bits are estimated, the surviving candidates at the top of the tree can be passed through a polar code encoder to get the final candidates at the bottom of the tree. Since there is no constraint on the estimation of the bits, at each time-step in the decoding of a Rate-1 node, one bit needs to selected for estimation. Therefore, while SCL requires $3N-2$ time-steps to decode Rate-1 nodes, our simplified decoder requires the following number of time-steps:

$$T_{\text{Rate-1}} = \binom{N}{1} = N. \tag{4.2}$$

Theorem 1. *The PM for a Rate-1 node of length 2^t can be calculated as:*

$$\text{PM}_{2^t-1} = \sum_{i=0}^{2^t-1} \ln\left(1 + \mathrm{e}^{-\eta_i \alpha_i}\right), \tag{4.3}$$

where α_i and η_i are relative to the top of the Rate-1 node tree.

Proof. We prove Theorem 1 by induction. We first show that for $\mathcal{P}(2,2)$, i.e. a Rate-1 polar code of length $N = 2$, the theorem is correct. For any polar code with $N = 2$ we have

$$\alpha_0^{\mathrm{l}} = \ln\left(\frac{1 + \mathrm{e}^{\alpha_0+\alpha_1}}{\mathrm{e}^{\alpha_0} + \mathrm{e}^{\alpha_1}}\right), \tag{4.4}$$

$$\alpha_0^{\mathrm{r}} = \alpha_1 + \eta_0^{\mathrm{l}}\alpha_0. \tag{4.5}$$

For a Rate-1 code with $N = 2$, from (2.9) we can see that:

$$\eta_0^{\mathrm{l}} = \eta_0\eta_1,$$
$$\eta_0^{\mathrm{r}} = \eta_1,$$

where η_0^{l} and η_0^{r} are obtained traversing the whole decoding tree, that in our case consists of stages $t = 1$ and $t = 0$. Therefore, (4.5) can be re-written as

$$\alpha_0^{\mathrm{r}} = \alpha_1 + \eta_0\eta_1\alpha_0. \tag{4.6}$$

Starting from (2.20) and then substituting (4.4) and (4.5), the PM associated with $\mathcal{P}(2,2)$ is

$$\begin{aligned}
\mathrm{PM}_1 &= \ln\left(1 + \mathrm{e}^{-\eta_0^{\mathrm{l}}\alpha_0^{\mathrm{l}}}\right) + \ln\left(1 + \mathrm{e}^{-\eta_0^{\mathrm{r}}\alpha_0^{\mathrm{r}}}\right) = \ln\left(1 + \mathrm{e}^{-\eta_0\eta_1 \ln\left(\frac{1+\mathrm{e}^{\alpha_0+\alpha_1}}{\mathrm{e}^{\alpha_0}+\mathrm{e}^{\alpha_1}}\right)}\right) + \ln\left(1 + \mathrm{e}^{-\eta_1(\alpha_1+\eta_0\eta_1\alpha_0)}\right) \\
&= \ln\left(1+\left(\frac{\mathrm{e}^{\alpha_0} + \mathrm{e}^{\alpha_1}}{1+\mathrm{e}^{\alpha_0+\alpha_1}}\right)^{\eta_0\eta_1}\right) + \ln\left(1 + \mathrm{e}^{-\eta_1\alpha_1-\eta_0\alpha_0}\right) \\
&= \begin{cases}
\ln\left(1 + \mathrm{e}^{-\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{-\alpha_1}\right), & \text{when } \begin{cases}\eta_0 = 1\\ \eta_1 = 1\end{cases} \\
\ln\left(1 + \mathrm{e}^{-\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{\alpha_1}\right), & \text{when } \begin{cases}\eta_0 = 1\\ \eta_1 = -1\end{cases} \\
\ln\left(1 + \mathrm{e}^{\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{-\alpha_1}\right), & \text{when } \begin{cases}\eta_0 = -1\\ \eta_1 = 1\end{cases} \\
\ln\left(1 + \mathrm{e}^{\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{\alpha_1}\right), & \text{when } \begin{cases}\eta_0 = -1\\ \eta_1 = -1\end{cases}
\end{cases}
\end{aligned}$$

$$= \ln\left(1 + \mathrm{e}^{-\eta_0\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{-\eta_1\alpha_1}\right), \tag{4.7}$$

which is equal to (4.3), and thus proves the theorem for $N = 2$. Now suppose the theorem stands for a Rate-1 polar code of length 2^{t-1}. For $\mathcal{P}(2^t, 2^t)$ we can calculate α_i^{l} and α_i^{r} with (2.7) and (2.8) respectively. Since α_i^{l} and α_i^{r} are associated with two Rate-1 polar codes of length 2^{t-1}, for which we have supposed Theorem 1 stands, their respective PMs can be calculated as

$$\mathrm{PM}^{\mathrm{l}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1 + \mathrm{e}^{-\eta_i^{\mathrm{l}}\alpha_i^{\mathrm{l}}}\right), \tag{4.8}$$

$$\mathrm{PM}^{\mathrm{r}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1 + \mathrm{e}^{-\eta_i^{\mathrm{r}}\alpha_i^{\mathrm{r}}}\right). \tag{4.9}$$

Recall that the PM at a node is the summation of the PMs calculated at the child nodes: we can consequently write

$$\mathrm{PM}_{2^t-1} = \mathrm{PM}^{\mathrm{l}}_{2^{t-1}-1} + \mathrm{PM}^{\mathrm{r}}_{2^{t-1}-1}. \tag{4.10}$$

From Figure 4.1 we can see the recursive construction of polar codes. For example, u_0 and u_1 are coded into a polar code with $N = 2$, consisting of β_0 and β_1. At the same time, β_0 and β_2 are coded into another polar code with $N = 2$, consisting of x_0 and x_2. Thus, we are allowed to pair bit i and bit $i + 2^{t-1}$ at any node at stage t in the decoding tree, and consider them a polar code with $N = 2$. In our case, we pair bit i of (4.8) and (4.9), and identify 2^{t-1} Rate-1 polar codes of length two. Consequently, we can express (4.10) as the summation of 2^{t-1} instances of (4.7):

$$\mathrm{PM}_{2^t-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1 + \mathrm{e}^{-\eta_i\alpha_i}\right) + \ln\left(1 + \mathrm{e}^{-\eta_{i+2^{t-1}}\alpha_{i+2^{t-1}}}\right) = \sum_{i=0}^{2^t-1} \ln\left(1 + \mathrm{e}^{-\eta_i\alpha_i}\right),$$

which is equal to (4.3). This means that if Theorem 1 is valid for $N = 2^{t-1}$, then it stands for $N = 2^t$ as well. Since we have proven it for $N = 2$, Theorem 1 is valid for all N. □

Theorem 2. *In the hardware-friendly formulation of SCL algorithm, the PM for a Rate-1 node of length 2^t can be calculated as:*

$$\mathrm{PM}_{2^t-1} = \frac{1}{2} \sum_{i=0}^{2^t-1} \mathrm{sgn}\left(\alpha_i\right) \alpha_i - \eta_i\alpha_i, \tag{4.11}$$

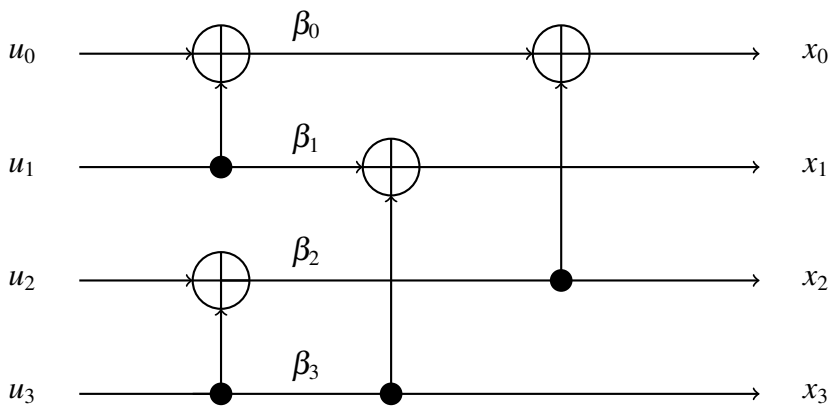


Figure 4.1: Polar encoding example.

where α_i and η_i are relative to the top of the Rate-1 node tree.

Proof. (4.11) is equivalent to

$$\mathrm{PM}_i = \begin{cases} \mathrm{PM}_{i-1} + |\alpha_i|, & \text{if } \eta_i \neq \operatorname{sgn}(\alpha_i), \\ \mathrm{PM}_{i-1}, & \text{otherwise,} \end{cases}$$

which is an expression similar to that of (2.21). As with Theorem 1, we prove Theorem 2 by induction, starting from $\mathcal{P}(2,2)$. According to (2.11) and (2.8), for every polar code with $N = 2$ we have

$$\alpha_0^{\mathrm{l}} = \operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)\min(|\alpha_0|, |\alpha_1|), \tag{4.12}$$

$$\alpha_0^{\mathrm{r}} = \alpha_1 + \eta_0^{\mathrm{l}}\alpha_0. \tag{4.13}$$

where α_0^{l} and α_0^{r} are obtained traversing the whole decoding tree, that in our case consists of stages $t = 1$ and $t = 0$. For a Rate-1 node with $N = 2$:

$$\operatorname{sgn}\left(\alpha_0^{\mathrm{l}}\right) = \operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1),$$

$$\operatorname{sgn}(\alpha_0^{\mathrm{r}}) = \operatorname{sgn}(\alpha_1 + \eta_0\eta_1\alpha_0) = \begin{cases} \operatorname{sgn}(\alpha_1), & \text{if } |\alpha_0| \leq |\alpha_1|, \\ \eta_0\eta_1 \operatorname{sgn}(\alpha_0), & \text{otherwise.} \end{cases}$$

From (2.22), substituting (4.12) and (4.13), the PM associated with $\mathcal{P}(2,2)$ would be computed as

$$\begin{aligned}\mathrm{PM}_1 =& \frac{1}{2}\left(\operatorname{sgn}\left(\alpha_0^{\mathrm{l}}\right)\alpha_0^{\mathrm{l}} - \eta_0^{\mathrm{l}}\alpha_0^{\mathrm{l}} + \operatorname{sgn}\left(\alpha_0^{\mathrm{r}}\right)\alpha_0^{\mathrm{r}} - \eta_0^{\mathrm{r}}\alpha_0^{\mathrm{r}}\right)\\ =& \frac{1}{2}(\operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)\operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)\min(|\alpha_0|,|\alpha_1|) - \eta_0\eta_1\operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)\min(|\alpha_0|,|\alpha_1|)\\ &+ \operatorname{sgn}(\alpha_1+\eta_0\eta_1\alpha_0)(\alpha_1+\eta_0\eta_1\alpha_0) - \eta_1(\alpha_1+\eta_0\eta_1\alpha_0)).\end{aligned}$$

Now we break the problem into two cases:
When $|\alpha_0| \le |\alpha_1|$, recalling that $|\alpha| = \operatorname{sgn}(\alpha)\alpha$, we have

$$\begin{aligned}\mathrm{PM}_1 =& \frac{1}{2}(|\alpha_0| - \eta_0\eta_1\operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)|\alpha_0| + \operatorname{sgn}(\alpha_1)(\alpha_1+\eta_0\eta_1\alpha_0) - \eta_1(\alpha_1+\eta_0\eta_1\alpha_0))\\ =& \frac{1}{2}(\operatorname{sgn}(\alpha_0)\alpha_0 - \eta_0\alpha_0 + \operatorname{sgn}(\alpha_1)\alpha_1 - \eta_1\alpha_1).\end{aligned} \tag{4.14}$$

When $|\alpha_0| > |\alpha_1|$:

$$\begin{aligned}\mathrm{PM}_1 =& \frac{1}{2}(|\alpha_1| - \eta_0\eta_1\operatorname{sgn}(\alpha_0)\operatorname{sgn}(\alpha_1)|\alpha_1| + \eta_0\eta_1\operatorname{sgn}(\alpha_0)(\alpha_1+\eta_0\eta_1\alpha_0) - \eta_1(\alpha_1+\eta_0\eta_1\alpha_0))\\ =& \frac{1}{2}(\operatorname{sgn}(\alpha_0)\alpha_0 - \eta_0\alpha_0 + \operatorname{sgn}(\alpha_1)\alpha_1 - \eta_1\alpha_1).\end{aligned} \tag{4.15}$$

(4.14) and (4.15) are identical, and they are equal to (4.11), but they have been derived from (2.22), so the theorem is proven for $N = 2$. Now suppose the theorem stands for a Rate-1 polar code of length $N = 2^{t-1}$. For $\mathcal{P}(2^t, 2^t)$ we can calculate α_i^{l} and α_i^{r} with (2.11) and (2.8) respectively. Since α_i^{l} and α_i^{r} are two Rate-1 polar codes of length 2^{t-1}, the theorem stands for them and the PM can be calculated as

$$\mathrm{PM}^{\mathrm{l}}_{2^{t-1}-1} = \frac{1}{2}\sum_{i=0}^{2^{t-1}-1}\operatorname{sgn}\left(\alpha_i^{\mathrm{l}}\right)\alpha_i^{\mathrm{l}} - \eta_i^{\mathrm{l}}\alpha_i^{\mathrm{l}}, \tag{4.16}$$

$$\mathrm{PM}^{\mathrm{r}}_{2^{t-1}-1} = \frac{1}{2}\sum_{i=0}^{2^{t-1}-1}\operatorname{sgn}\left(\alpha_i^{\mathrm{r}}\right)\alpha_i^{\mathrm{r}} - \eta_i^{\mathrm{r}}\alpha_i^{\mathrm{r}}. \tag{4.17}$$

As with Theorem 1, we exploit the recursive construction of polar codes, that allows us to identify polar codes with $N = 2$ pairing bit i and bit $i + 2^{t-1}$ in a node. Joining (4.10) and (4.15)-(4.17) we

get

$$\mathrm{PM}_{2^t-1} = \frac{1}{2}\sum_{i=0}^{2^t-1} \mathrm{sgn}\left(\alpha_i\right)\alpha_i - \eta_i\alpha_i. \tag{4.18}$$

which is equal to (4.11). Theorem 2 is then valid for $N = 2^t$ if it stands for $N = 2^{t-1}$: since we have proven it for $N = 2$, it is valid for every N. □

Rate-0 Nodes

The simplified formulation for Rate-0 nodes proposed in this section can be completed in at most $\log_2 N$ time-steps, as opposed to the $2N - 2$ required by SCL, since simplified Rate-0 nodes only need to add at most N numbers together.

Theorem 3. *The PM for a Rate-0 node of length 2^t can be calculated as:*

$$\mathrm{PM}_{2^t-1} = \sum_{i=0}^{2^t-1} \ln\left(1 + \mathrm{e}^{-\alpha_i}\right), \tag{4.19}$$

where α_i is the LLR value at the top of the Rate-0 node tree.

Proof. Since β_i is initialized as 0 for frozen bits, for $\mathcal{P}(2, 0)$ which is a Rate-0 code with $N = 2$ we have:

$$\eta_0^{\mathrm{l}} = \eta_0^{\mathrm{r}} = 1.$$

Therefore, (4.5) can be re-written as

$$\alpha_0^{\mathrm{r}} = \alpha_1 + \alpha_0. \tag{4.20}$$

Starting from (2.20) and then substituting (4.4) and (4.20), the PM associated with $\mathcal{P}(2, 0)$ is

$$\begin{aligned}\mathrm{PM}_1 &= \ln\left(1 + \mathrm{e}^{-\eta_0^{\mathrm{l}}\alpha_0^{\mathrm{l}}}\right) + \ln\left(1 + \mathrm{e}^{-\eta_0^{\mathrm{r}}\alpha_0^{\mathrm{r}}}\right) = \ln\left(1 + \mathrm{e}^{-\ln\left(\frac{1+\mathrm{e}^{\alpha_0+\alpha_1}}{\mathrm{e}^{\alpha_0}+\mathrm{e}^{\alpha_1}}\right)}\right) + \ln\left(1 + \mathrm{e}^{-(\alpha_1+\alpha_0)}\right) \\ &= \ln\left(1 + \mathrm{e}^{-\alpha_0}\right) + \ln\left(1 + \mathrm{e}^{-\alpha_1}\right),\end{aligned} \tag{4.21}$$

which is equal to (4.19) and proves the theorem for $N = 2$. For $\mathcal{P}(2^t, 0)$ we can calculate α_i^{l} and α_i^{r} with (2.7) and (2.8) respectively. Supposing the theorem stands for a Rate-0 polar code with $N = 2^{t-1}$, the PMs of α_i^{l} and α_i^{r} which are two Rate-0 polar codes of length $N = 2^{t-1}$ can be

calculated as

$$\mathrm{PM}^{\mathrm{l}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1 + \mathrm{e}^{-\alpha^{\mathrm{l}}_i}\right), \tag{4.22}$$

$$\mathrm{PM}^{\mathrm{r}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1 + \mathrm{e}^{-\alpha^{\mathrm{r}}_i}\right). \tag{4.23}$$

Identifying $\mathcal{P}(2, 0)$ by pairing bits as in Theorem 1, and using (4.10) and (4.21)-(4.23), for a Rate-0 polar code of length $N = 2^t$ we get

$$\mathrm{PM}_{2^t-1} = \sum_{i=0}^{2^t-1} \ln\left(1 + \mathrm{e}^{-\alpha_i}\right),$$

which proves Theorem 3 is valid for $N = 2^t$ supposing it stands for $N = 2^{t-1}$. Having proven it for $N = 2$, Theorem 3 is valid for all N. □

Theorem 4. *In the hardware-friendly formulation of SCL algorithm, the PM for a Rate-0 node of length 2^t can be calculated as:*

$$\mathrm{PM}_{2^t-1} = \frac{1}{2} \sum_{i=0}^{2^t-1} \mathrm{sgn}\left(\alpha_i\right) \alpha_i - \alpha_i, \tag{4.24}$$

where α_i is the LLR value at the top of the Rate-0 node tree.

Proof. (4.24) has been proposed in [19]. The proof follows that of Theorem 3 and therefore it is not reported in this chapter. □

Rep Nodes

The simplified Rep node decoding can be completed in at most $1 + \log_2 N$ time-steps instead of $2N - 1$, since simplified Rep nodes only need to add at most N numbers together and choose the best L out of the resulting $2L$ candidates.

Theorem 5. *The PM for a Rep node of length 2^t can be calculated as:*

$$\mathrm{PM}_{2^t-1} = \sum_{i=0}^{2^t-1} \ln\left(1 + \mathrm{e}^{-\eta_{2^t-1}\alpha_i}\right), \tag{4.25}$$

where α_i is relative to the top of the Rep node tree and η_{2^t-1} is relative to the information bit in the Rep node tree.

Proof. For $\mathcal{P}(2,1)$ which is a Rep code of length $N = 2$ we have:

$$\eta_0^{\mathrm{l}} = \eta_0\eta_1 = 1,$$
$$\eta_0^{\mathrm{r}} = \eta_1 = \eta_0.$$

Therefore, (4.5) can be re-written as

$$\alpha_0^{\mathrm{r}} = \alpha_1 + \alpha_0. \tag{4.26}$$

From (2.20) and by substituting (4.4) and (4.26), the PM associated with $\mathcal{P}(2,1)$ is

$$\begin{aligned}\mathrm{PM}_1 &= \ln\left(1+\mathrm{e}^{-\eta_0^{\mathrm{l}}\alpha_0^{\mathrm{l}}}\right) + \ln\left(1+\mathrm{e}^{-\eta_0^{\mathrm{r}}\alpha_0^{\mathrm{r}}}\right) = \ln\left(1+\mathrm{e}^{-\ln\left(\frac{1+\mathrm{e}^{\alpha_0+\alpha_1}}{\mathrm{e}^{\alpha_0}+\mathrm{e}^{\alpha_1}}\right)}\right) + \ln\left(1+\mathrm{e}^{-\eta_1(\alpha_1+\alpha_0)}\right)\\ &= \begin{cases}\ln\left(1+\mathrm{e}^{-\alpha_0}\right)+\ln\left(1+\mathrm{e}^{-\alpha_1}\right), & \text{when } \eta_1 = 1\\ \ln\left(1+\mathrm{e}^{\alpha_0}\right)+\ln\left(1+\mathrm{e}^{\alpha_1}\right), & \text{when } \eta_1 = -1\end{cases}\\ &= \ln\left(1+\mathrm{e}^{-\eta_1\alpha_0}\right)+\ln\left(1+\mathrm{e}^{-\eta_1\alpha_1}\right),\end{aligned} \tag{4.27}$$

which is equivalent to (4.25) and proves the theorem for $N = 2$. For a Rep code of length 2^t we can calculate α_i^{l} and α_i^{r} with (2.7) and (2.8) respectively. Assuming the theorem stands for a Rep polar code with $N = 2^{t-1}$ and using Theorem 3, α_i^{l} represents a Rate-0 node and α_i^{r} represents a Rep node of length $N = 2^{t-1}$. Their PMs are computed as

$$\mathrm{PM}^{\mathrm{l}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1+\mathrm{e}^{-\alpha_i^{\mathrm{l}}}\right), \tag{4.28}$$

$$\mathrm{PM}^{\mathrm{r}}_{2^{t-1}-1} = \sum_{i=0}^{2^{t-1}-1} \ln\left(1+\mathrm{e}^{-\eta_{2^t-1}\alpha_i^{\mathrm{r}}}\right). \tag{4.29}$$

Again by pairing bits, we can identify $\mathcal{P}(2,1)$. Joining (4.10) and (4.27)-(4.29), for a Rep node of length $N = 2^t$ we obtain

$$\mathrm{PM}_{2^t-1} = \sum_{i=0}^{2^t-1} \ln\left(1+\mathrm{e}^{-\eta_{2^t-1}\alpha_i}\right),$$

which, as in the previous theorems, by induction proves Theorem 5 for all N. □

Theorem 6. *In the hardware-friendly formulation of SCL algorithm, the PM for a Rep node of length 2^t can be calculated as:*

$$\mathrm{PM}_{2^t-1} = \frac{1}{2}\sum_{i=0}^{2^t-1} \mathrm{sgn}\left(\alpha_i\right)\alpha_i - \eta_{2^t-1}\alpha_i, \tag{4.30}$$

where α_i is relative to the top of the Rep node tree and η_{2^t-1} is relative to the information bit in the Rep node tree.

Proof. (4.30) has been proposed in [19]. The proof follows that of Theorem 5 and therefore it is not reported in this chapter. □

SPC Nodes

Since List-SD goes from right to left of the tree as opposed to SCL, the even-parity constraint which is imposed at the leftmost bit is decoded last in List-SD and that introduces significant error-correction performance loss. A lossless approach with List-SD would be to estimate the bits at the top of the tree in groups of two, while considering the even-parity constraint satisfied by the other bits. For example, if $\beta_i\beta_j = 00$ satisfies the even-parity constraint of a code $\mathcal{B}$ of length N, then neither $\beta_i\beta_j = 01$ nor $\beta_i\beta_j = 10$ will satisfy the constraint for the same code while $\beta_i\beta_j = 11$ will satisfy the constraint. This approach will create $2L$ candidates from which L best need to be selected. However, since at each time-step two bits have to be selected for estimation, the number of time-steps required to decode SPC nodes with guarantee of no error-correction performance loss will be

$$T_{\mathrm{SPC}} = \binom{N}{2} = \frac{N(N-1)}{2}, \tag{4.31}$$

which is quadratic with N and it is more complex than the time required by SCL which is linear with N. With this approach for SPC nodes, List-SD only becomes less complex than SCL when the following stands

$$\frac{N(N-1)}{2} < 3N - 3, \tag{4.32}$$

which results in $1 < N < 6$. Therefore, only SPC nodes of length 2 or 4 can be decoded with List-SD with fewer time-steps than SCL without any error-correction performance degradation.

To alleviate this increase in time complexity, we propose a technique to decode the frozen bit in the first stage of the decoding process. We also approximate the PM calculations in an SPC

node in the hardware-friendly formulation by only finding the LLR value of the least reliable bit and using the LLR values at the top of the polar code tree in the SCL decoding algorithm for the rest of the bits. This will reduce the time requirements of decoding an SPC node from $\frac{N(N-1)}{2}$ to at most $N + \log_2 N - 1$ which is also less than the time requirements of SCL which is $3N - 3$.

To satisfy the even-parity constraint, we first find γ for each path based on (2.18) which can take at most $\log_2 N$ time-steps. We then initialize the PM with

$$\mathrm{PM}_0 = \begin{cases} \mathrm{PM}_{-1} + |\alpha_{i_{\min}}|, & \text{if } \gamma = 1, \\ \mathrm{PM}_{-1}, & \text{otherwise,} \end{cases} \tag{4.33}$$

where PM_{-1} is the PM carried over from the previous stage (if any). In this way, the least reliable bit which corresponds to the even-parity constraint is decoded first. For bits other than the least reliable bit, the PM is updated as

$$\mathrm{PM}_i = \begin{cases} \mathrm{PM}_{i-1} + |\alpha_i| + (1-2\gamma)|\alpha_{i_{\min}}|, & \text{if } \eta_i \neq \operatorname{sgn}(\alpha_i), \\ \mathrm{PM}_{i-1}, & \text{otherwise,} \end{cases} \tag{4.34}$$

which takes $N - 1$ time-steps. Finally, when all the bits are estimated, the least reliable bit is set to preserve the even-parity constraint as

$$\beta_{i_{\min}} = \bigoplus_{\substack{i=0 \\ i \neq i_{\min}}}^{2^t - 1} \beta_i. \tag{4.35}$$

It should be noted that this approximation becomes exact for $L = 2$ and for $L > 2$ there is almost no error-correction performance loss as will be discussed in Section 4.1.1.

4.1.1 Decoder Architecture

The architecture of the decoder is portrayed in Figure 4.2. We based our design on the CRC-aided SCL decoder architecture presented in [20]: the SCL part of the architecture remains virtually identical. Input channel LLRs are stored in a single memory, while each of the L decoders requires a dedicated memory for its PMs, internal LLRs, current path estimation, and hard estimates β. A set of P_e Processing Element (PE)s is allocated in each of the L SC decoders. A PE implements

Table 4.1: Number of Special Nodes for Different Rates of a Polar Code of Length $N = 1024$ and CRC of Length 16. The Code is Optimized for $E_b/N_0 = 2$ dB.

Rate	Node	Length								
		2	4	8	16	32	64	128	256	512
$\frac{1}{4}$	Rate-1	1	2	1	2	1	0	0	0	0
	Rate-0	1	1	2	1	1	2	0	0	0
	Rep	0	18	7	5	3	0	0	1	0
	SPC	0	16	9	3	0	0	0	0	0
$\frac{1}{2}$	Rate-1	2	5	5	3	2	0	0	0	0
	Rate-0	2	4	3	3	2	0	0	0	0
	Rep	0	13	6	3	0	1	1	0	0
	SPC	0	12	6	4	0	1	1	0	0
$\frac{3}{4}$	Rate-1	3	2	3	3	3	1	0	0	0
	Rate-0	3	2	3	2	0	0	0	0	0
	Rep	0	12	5	1	0	1	0	0	0
	SPC	0	11	4	3	1	0	1	1	0

both (2.11) and (2.8), and its output is selected depending on the current decoding phase. PMs are computed and sorted through a non-pruned radix-$2L$ sorter [20].

The identification of special nodes is performed offline, by exploring the decoding tree in its entirety. As an example, Table 4.1 shows the number of each special node for a polar code of length $N = 1024$ and rates $1/4$, $1/2$ and $3/4$. It is worth mentioning that as the rate increases, the number of Rate-1 and SPC nodes of longer lengths increases and the number of Rate-0 and Rep nodes of longer lengths decreases.

This information allows to know the type of each node, its size and its order within the decoding schedule. It is then passed as an input to the decoder controller unit, and it is considered available during the whole decoding process. Storage of the schedule is not considered in the proposed architecture, as it is strongly dependent on the structure and number of the supported codes. Note, however, that code rate and frozen bit positions do not impact on the decoder architecture and do not need to be known at design time. This is because a maximum special node size is decided at design time, usually equal to the number of PEs: all special nodes whose size is smaller or equal to

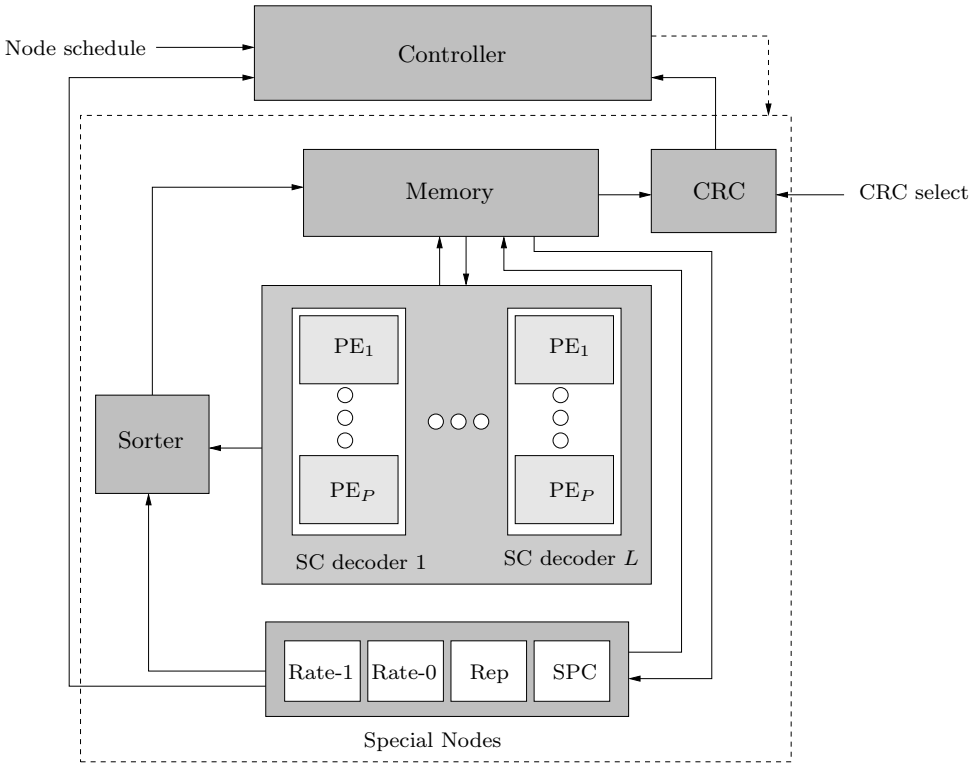


Figure 4.2: SSCL (SSCL-SPC) decoder architecture.

that can be handled, while larger ones are divided into smaller nodes. For example, if the maximum special node size is 128, a Rate-0 node of size 256 will be considered as two Rate-0 nodes of size 128. The proposed architecture allows to select, at run time, the number and position of frozen and information bits, along with special nodes. These are handled by the special nodes unit.

The decoding process follows the standard SCL and the architecture described in [20] until the schedule foresees a special node. At this point, the hardware implementing the SCL is bypassed by the special node unit (Figure 4.2). This unit is composed of four sub-modules, each taking care of the computations necessary for each special node. Moreover, this unit enforces bit estimation and path duplication at special nodes: in the standard SCL scheduling, these operations would normally be applied only when the currently scheduled node is a leaf node. Finally, it overrides the control module address generator to accommodate the change of schedule.

Each CRC unit implements four parallel CRC calculations based on polynomials of different lengths (4, 8, 16 and 32 bits). An input chooses the polynomial that should be used for the selection of the best among the L surviving candidates.

Rate-1 Node

Rate-1 nodes are constituted of $N_{\text{Rate-1}}$ information bits. The estimation of each of them is equivalent to an SCL estimation step, in which paths are duplicated and split for $N_{\text{Rate-1}}$ times. This means that no inherent degree of parallelism can be exploited, and that the speed-up comes only from the fact that we do not need to traverse the whole polar code tree to do it. Thus, the Rate-1 node in the special node unit enforces a series of pipelined bit estimations and path splitting operations, preventing the tree from being traversed further. At every step, each PM calculation follows (4.11), requiring a single LLR value α_i and making the computation straightforward.

Rate-0 Node

All $N_{\text{Rate-0}}$ bits that constitute a Rate-0 node are frozen. Consequently, the Rate-0 module reliably sets to 0 all the $N_{\text{Rate-0}}$ bits. At the same time, it updates each active path with the newly estimated bits, without the need for any path duplication operation. This step can be completed in as little as one clock cycle, depending on the structure of the memory where the paths are stored and how many bits can be accessed concurrently. The PM in Rate-0 nodes is calculated as in (4.24). It requires the summation of up to $N_{\text{Rate-0}}$ values: three different options are given at design time, and they are shown in Figure 4.3. The first possible structure is a Low-Latency Structure (LLS) with a fully parallel adder tree, as shown in Figure 4.3a: the PM calculation lasts a single clock cycle. The same structure can be pipelined as well (Figure 4.3b) to have a High-Frequency Structure (HFS), thus requiring $\log_2(N_{\text{Rate-0}})$ clock cycles to be completed but potentially allowing for higher frequencies. Finally, a Low-Complexity Structure (LCS) with a serial architecture is also considered in Figure 4.3c, with a single adder and a latency of $N_{\text{Rate-0}}$ clock cycles.

Rep Node

Of the N_{Rep} bits of a Rep node, $N_{\text{Rep}} - 1$ are frozen and one is an information bit. As with the Rate-0 node, all frozen bits are set to 0 by the dedicated special node module without path duplication. However, the information bit must be estimated: consequently, the paths are duplicated a single time. Like the Rate-0 node, depending on the number of bits that can be accessed at the same time in the path memory, this operation can be completed in a single clock cycle. PM is computed as described in (4.30) for Rep nodes. Up to N_{Rep} values need to be summed together: the same possible architectural choices portrayed in Figure 4.3 for Rate-0 can be applied to Rep nodes and

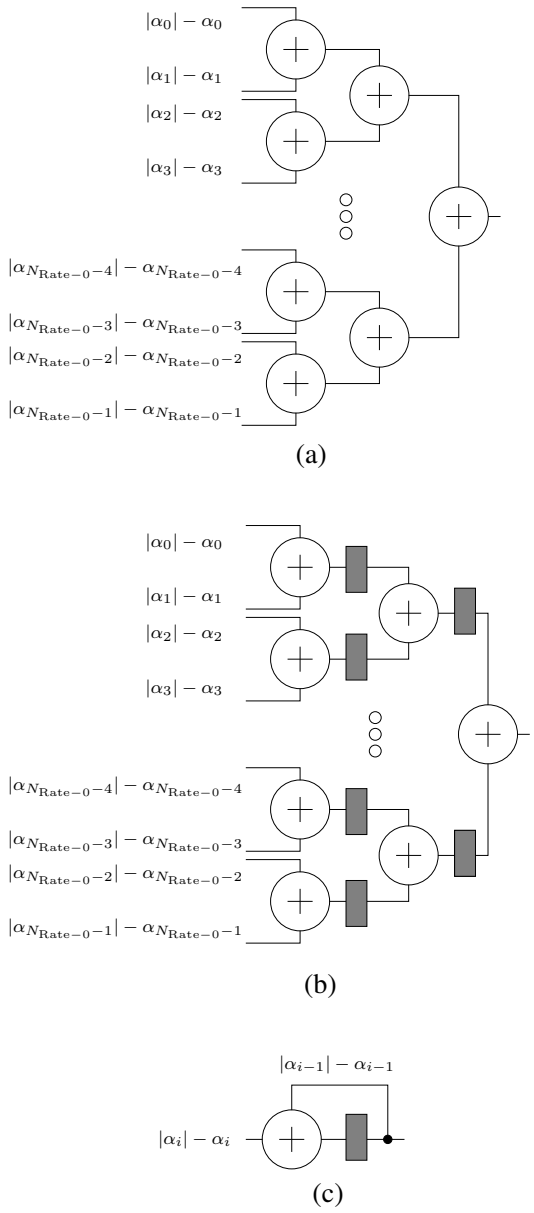


Figure 4.3: Design-time architectural choices for Rate-0 nodes.

are given at design time.

SPC Node

Since the SPC node has $N_{\text{SPC}} - 1$ information bits, the special node module splits the paths as many times. PM calculation for SPC nodes follows (4.33)-(4.34). The path and PM updates are performed concurrently, one bit at a time as the paths are duplicated, sorted and selected: they cannot make use of any degree of parallelism. However, different node structures have been considered for the calculation of γ and $|\alpha_{i_{\min}}|$. The lowest latency solution computes γ in one clock cycle over N_{SPC} bits cycle through an XOR tree, and concurrently finds $|\alpha_{i_{\min}}|$ through a tree of comparators. The second structure pipelines both XOR and comparator trees: $\log_2(N_{\text{SPC}})$ clock cycles are then needed, while with a single XOR and a single comparator $N_{\text{SPC}} - 1$ cycles are necessary.

Error-Correction Performance

As we have detailed in this section, the introduction of Rate-1, Rate-0 and Rep nodes has no impact on the error-correction performance of SCL. However, the equations implemented by the SPC node are approximated for $L > 2$, and might introduce degradation. Figure 4.4 plots the FER and BER for a polar code with $N = 2048$ and $R = \frac{1}{2}$, decoded with SSCL and SSCL-SPC and a 32-bit CRC. In this figure, SSCL(L) (SSCL-SPC(L)) represents SSCL (SSCL-SPC) decoding with list size L. Error-correction performance is shown for list sizes ranging between 2 and 32. As expected from the theory, SSCL and SSCL-SPC curves are superimposed for $L = 2$: nevertheless, the degradation brought by the approximated SPC is negligible in all cases (< 0.1 dB at FER of 10^{-5}). Similar observations can be made in case of the smaller $\mathcal{P}(1024, 512)$ code, whose FER and BER are shown in Figure 4.5 for 16-bit CRC. It can be seen that the SPC approximations have more impact as the list size increases.

The impact of quantization of LLRs and PMs on the error-correction performance of both SSCL and SSCL-SPC has been analyzed through extensive simulations. We have investigated the trio of values $Q(Q_\alpha, Q_{\text{PM}}, Q_f)$, where Q_α represents the number of LLR bits, Q_{PM} is the number of bits of each PM and Q_f is the number of fractional bits for Q_α and Q_{PM}. For code $\mathcal{P}(1024, 512)$, the best performing combination of quantization values is $Q(6, 8, 2)$. Figure 4.6 plots FER and BER curves with such quantization values for both SSCL and SSCL-SPC, along with SCL floating point results. It can be seen that, with the careful choice of quantization values, the degradation with respect to floating point is negligible for all list sizes.

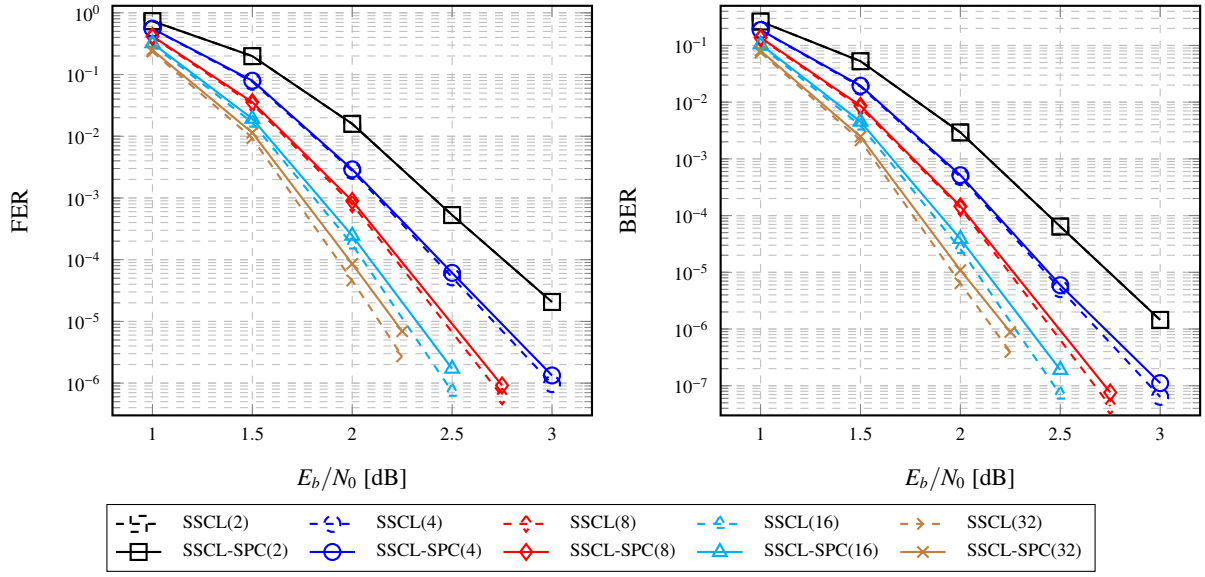


Figure 4.4: FER and BER performance comparison between CRC-aided SSCL and SSCL-SPC decoding of $\mathcal{P}(2048, 1024)$. The code is optimized for $E_b/N_0 = 2$ dB and the CRC length is 32.

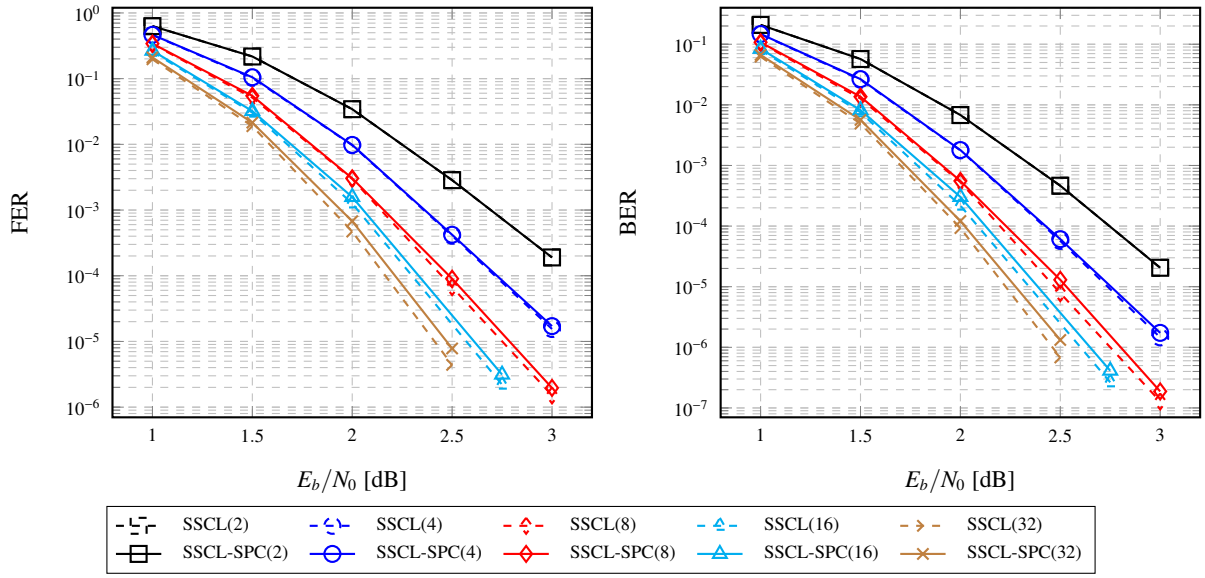


Figure 4.5: FER and BER performance comparison between CRC-aided SSCL and SSCL-SPC decoding of $\mathcal{P}(1024, 512)$. The code is optimized for $E_b/N_0 = 2$ dB and the CRC length is 16.

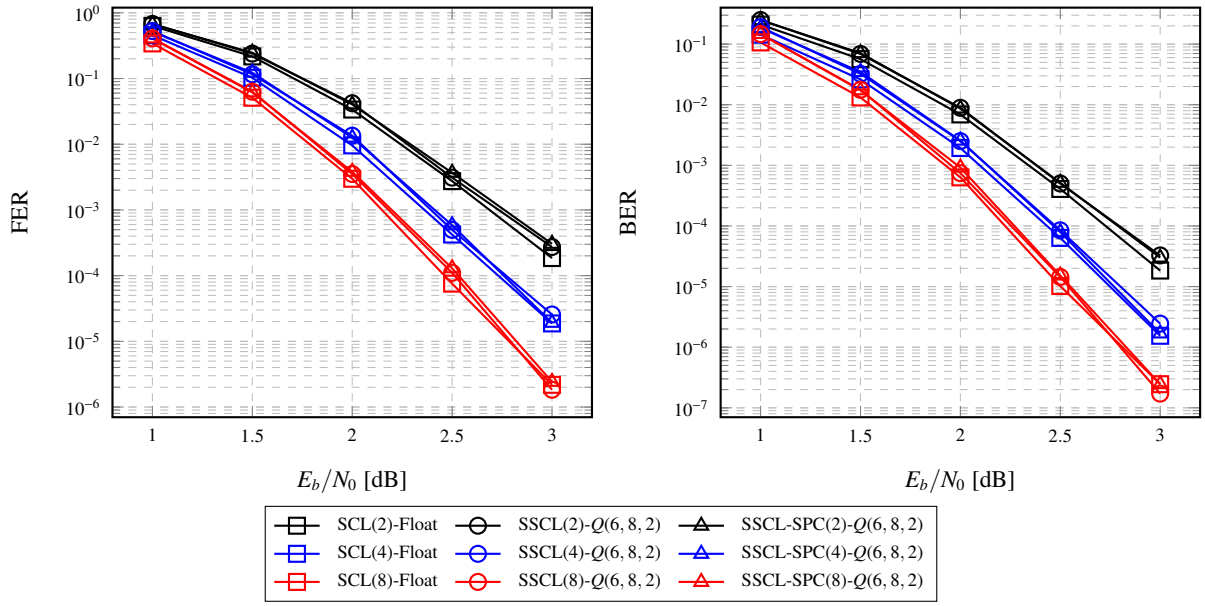


Figure 4.6: Effect of quantization on the FER and BER of $\mathcal{P}(1024, 512)$ using CRC-aided SSCL and SSCL-SPC decoding with CRC length of 16. The code is optimized for $E_b/N_0 = 2$ dB.

4.2 Fast Simplified Successive-Cancellation List Decoding

In this section, we propose a fast decoding approach for Rate-1 nodes and use it to develop Fast-SSCL. We further propose a fast decoding approach for SPC nodes in SSCL-SPC and use it to develop Fast-SSCL-SPC. To this end, we provide the exact number of path forks in Rate-1 and SPC nodes to guarantee error-correction performance preservation. Any path splitting after that number is redundant and any path splitting less than that number cannot guarantee the error-correction performance preservation. We further show that in practical applications, this number can be reduced with almost no error-correction performance loss. We use this phenomenon to optimize Fast-SSCL and Fast-SSCL-SPC for speed.

Guaranteed Error-Correction Performance Preservation

The fast Rate-1 and SPC decoders can be summarized by the following theorems.

Theorem 7. *In SSCL decoding with list size L, the number of path splitting in a Rate-1 node of*

length 2^t required to get the exact same results as the conventional SSCL decoder is

$$\min\left(L-1, 2^t\right). \tag{4.36}$$

The proposed technique results in $T_{\text{Fast-SSCL}_{\text{Rate-1}}}(2^t, 2^t) = \min\left(L-1, 2^t\right)$ which improves the required number of time steps to decode Rate-1 nodes when $L-1 < 2^t$. Every bit after the $L-1$-th can be obtained through hard decision on the LLR as

$$\beta_{i_l} = \begin{cases} 0, & \text{if } \alpha_{i_l} \geq 0, \\ 1, & \text{otherwise,} \end{cases} \tag{4.37}$$

without the need for path splitting. On the other hand, in case $\min\left(L-1, 2^t\right) = 2^t$, all bits of the node need to be estimated and the decoding automatically reverts to the process in SSCL decoding. The proof of the theorem is nevertheless valid for both $L-1 < 2^t$ and $L-1 \geq 2^t$.

In order to prove Theorem 7, we first introduce the following lemma.

Lemma 1. *For two positive real numbers a and b where $a < b$, the following holds:*

$$\ln\left(1+\mathrm{e}^{-a}\right) + \ln\left(1+\mathrm{e}^{b}\right) > \ln\left(1+\mathrm{e}^{a}\right) + \ln\left(1+\mathrm{e}^{-b}\right) \tag{4.38}$$

Proof. We prove

$$\ln\left(1+\mathrm{e}^{-a}\right) + \ln\left(1+\mathrm{e}^{b}\right) - \ln\left(1+\mathrm{e}^{a}\right) - \ln\left(1+\mathrm{e}^{-b}\right) > 0.$$

We can write

$$\begin{aligned} &\ln\left(1+\mathrm{e}^{-a}\right) + \ln\left(1+\mathrm{e}^{b}\right) - \ln\left(1+\mathrm{e}^{a}\right) - \ln\left(1+\mathrm{e}^{-b}\right) = \ln\left(\frac{1+\mathrm{e}^{-a}}{1+\mathrm{e}^{a}}\right) + \ln\left(\frac{1+\mathrm{e}^{b}}{1+\mathrm{e}^{-b}}\right) = \\ &\ln\left(\mathrm{e}^{-a}\frac{1+\mathrm{e}^{a}}{1+\mathrm{e}^{a}}\right) + \ln\left(\mathrm{e}^{b}\frac{1+\mathrm{e}^{-b}}{1+\mathrm{e}^{-b}}\right) = \ln\left(\mathrm{e}^{-a}\right) + \ln\left(\mathrm{e}^{b}\right) = b-a > 0, \end{aligned} \tag{4.39}$$

which proves the lemma. □

We now prove Theorem 7.

Proof. Let us consider the PMs associated with the L surviving paths at bit estimation step i as $\mathrm{PM}_i = \{\mathrm{PM}_{i_0}, \ldots, \mathrm{PM}_{i_{L-1}}\}$ and the LLR values associated with the Rate-1 node at path l as $\boldsymbol{\alpha}_l =$

$\{\alpha_{0_l}, \alpha_{1_l}, \ldots, \alpha_{2^t-1_l}\}$. For the purpose of this proof, let us also consider the vectors PM_i and $\boldsymbol{\alpha}_l$ sorted as follows:

$$\begin{aligned} &\mathrm{PM}_{i_l} \leq \mathrm{PM}_{i_{l+1}}, && 0 \leq l < L-1, \\ &|\alpha_{i_l}| \leq |\alpha_{i+1_l}|, && 0 \leq i < 2^t - 1. \end{aligned}$$

At each estimation step i, the corresponding bit is estimated as either 0 or 1, and the PMs are updated as

$$\mathrm{PM}_{i_l} = \sum_{j=0}^{i} \ln\left(1 + \mathrm{e}^{-\eta_{j_l}\alpha_{j_l}}\right), \tag{4.40}$$

which is a monotonic and non-decreasing function of i. At any given estimation step i within the Rate-1 node, the least reliable LLR among those still to be estimated is α_{i_l}.

We now prove the theorem by contradiction. Let us suppose that step $L-1$ splits path l into two surviving paths. The corresponding PMs will be

$$\mathrm{PM}_{L-1_p} = \sum_{j=0}^{L-2} \ln\left(1 + \mathrm{e}^{-\eta_{j_l}\alpha_{j_l}}\right) + \ln\left(1 + \mathrm{e}^{-|\alpha_{L-1_l}|}\right), \tag{4.41}$$

$$\mathrm{PM}_{L-1_q} = \sum_{j=0}^{L-2} \ln\left(1 + \mathrm{e}^{-\eta_{j_l}\alpha_{j_l}}\right) + \ln\left(1 + \mathrm{e}^{|\alpha_{L-1_l}|}\right), \tag{4.42}$$

where $0 \leq p < q < L$. We now show that there are at least L bit estimation sequences that result in PMs which are less than PM_{L-1_q}. To this end, we demonstrate that there are L paths originated from path l with smaller PMs than PM_{L-1_q} that are generated before estimating bit $L-1$.

Let us consider the lowest possible value that PM_{L-1_q} can assume:

$$\mathrm{PM}_{L-1_q} = \sum_{j=0}^{L-2} \ln\left(1 + \mathrm{e}^{-|\alpha_{j_l}|}\right) + \ln\left(1 + \mathrm{e}^{|\alpha_{L-1_l}|}\right), \tag{4.43}$$

which represents the case where the bits estimated in steps $0 \leq j \leq L-2$ match the hard decision of their corresponding LLR values, and the $L-1$-th does not. Let us consider the bit sequences differing from path q, with the bit not matching the LLR hard decision at step w, where $0 \leq w \leq$

$L-2$, while the $L-1$-th matches. The corresponding PM would be

$$\mathrm{PM}_{L-1_v} = \sum_{\substack{j=0 \\ j\neq w}}^{L-2} \ln\left(1+\mathrm{e}^{-|\alpha_{j_l}|}\right) + \ln\left(1+\mathrm{e}^{|\alpha_{w_l}|}\right) + \ln\left(1+\mathrm{e}^{-|\alpha_{L-1_l}|}\right). \tag{4.44}$$

Rewriting (4.43) as

$$\mathrm{PM}_{L-1_q} = \sum_{\substack{j=0 \\ j\neq w}}^{L-2} \ln\left(1+\mathrm{e}^{-|\alpha_{j_l}|}\right) + \ln\left(1+\mathrm{e}^{-|\alpha_{w_l}|}\right) + \ln\left(1+\mathrm{e}^{|\alpha_{L-1_l}|}\right), \tag{4.45}$$

and using the fact that $|\alpha_{L-1_l}| > |\alpha_{w_l}|$, we can use the result in Lemma 1 to conclude

$$\mathrm{PM}_{L-1_q} > \mathrm{PM}_{L-1_v}, \tag{4.46}$$

which in turn results in $q > v$. Since w can assume $L-1$ values, and taking in account the bit sequence represented by path p where all the bits agree with their corresponding LLR hard decision, there are at least L bit sequences which result in a smaller PM than PM_{L-1_q}. Therefore, $q \geq L$ which contradicts the assumption that $q < L$ and confirms that path q will be discarded. In other words, this proves that paths that consider bits not matching the LLR hard decision after the $L-1$-th step will always be discarded: it is thus useless to split paths after the $L-1$-th. Theorem 7 is consequently proven. □

The proposed theorem remains valid also for the hardware-friendly formulation that can be written as

$$\mathrm{PM}_{i_l} = \begin{cases} \mathrm{PM}_{i-1_l} + |\alpha_{i_l}|, & \text{if } \eta_{i_l} \neq \mathrm{sgn}(\alpha_{i_l}), \\ \mathrm{PM}_{i-1_l}, & \text{otherwise.} \end{cases} \tag{4.47}$$

Proof. At each step i, depending on the value of $|\alpha_{i_l}|$, two cases arise.

A $|\alpha_{i_l}| \geq \mathrm{PM}_{i-1_{L-1}} - \mathrm{PM}_{i-1_l}$

From (4.47), we can see that the modulus of the least reliable bit $|\alpha_{i_l}|$ is the minimum quantity that can be added to the PM in case $\eta_{i_l} \neq \mathrm{sgn}(\alpha_{i_l})$. If this quantity is greater than the difference between the currently considered path metric PM_{i-1_l} and the largest surviving path metric $\mathrm{PM}_{i-1_{L-1}}$, every estimation that sees $\eta_{i_l} \neq \mathrm{sgn}(\alpha_{i_l})$ will lead to $\mathrm{PM}_{i-1_l} + |\alpha_{i_l}| \geq \mathrm{PM}_{i-1_{L-1}}$

and thus to a discarded path. Consequently, for all the remaining estimations in the Rate-1 node, paths need not to be duplicated, and bits are estimated as $\eta_{i_l} = \text{sgn}(\alpha_{i_l})$.

B $|\alpha_{i_l}| < \text{PM}_{i-1_{L-1}} - \text{PM}_{i-1_l}$

Let us consider positions p and q in the ordering of PMs such that

$$\begin{aligned} \text{PM}_{i_p} &= \text{PM}_{i-1_l}, \\ \text{PM}_{i_q} &= \text{PM}_{i-1_l} + |\alpha_{i_l}|, \end{aligned}$$

where $l \leq p < q < L$. In this case, both bit estimates for the least reliable bit have to be taken into account since their corresponding paths will be ordered among the first L. In turn, the path in position $L-1$ at step $i-1$ is moved to position L at step i and thus discarded. The following estimation step $i+1$ must be evaluated independently, to see if it falls in case **A** or **B**.

As soon as case **A** is encountered in path l, that path does not need to undergo any subsequent splitting, and the remaining β_{i_l} can be obtained through LLR hard decision of (4.37). While case **B** requires continued path splitting, this can occur a limited amount of times before case **A** is encountered. The maximum amount of consecutive case **B** occurrences can be identified by the following worst case analysis.

1. Case **B** occurs at $i = 0$ and $l = 0$.

2. Considering that PM_{-1_0} is the PM at $l = 0$ before the first bit of the Rate-1 node is estimated, if $p = 0$ and $q = 1$ then

$$\begin{aligned} \text{PM}_{0_0} &= \text{PM}_{-1_0} \\ \text{PM}_{0_1} &= \text{PM}_{-1_0} + |\alpha_{0_0}| \\ \text{PM}_{0_2} &= \text{PM}_{-1_1} \\ &\vdots \end{aligned}$$

3. Case **B** occurs at $i = 1$ and $l = 0$.

4. Since $|\alpha_{0_0}| \leq |\alpha_{1_0}|$, $q > 1$. For $p = 0$ and $q = 2$,

$$\begin{aligned}
\mathrm{PM}_{1_0} &= \mathrm{PM}_{-1_0} \\
\mathrm{PM}_{1_1} &= \mathrm{PM}_{-1_0} + |\alpha_{0_0}| \\
\mathrm{PM}_{1_2} &= \mathrm{PM}_{-1_0} + |\alpha_{1_0}| \\
&\vdots
\end{aligned}$$

5. Since at every step $|\alpha_{i-1_0}| \leq |\alpha_{i_0}|$, then $q > i$. If at every case **B** step $p = 0$ and $q = i + 1$, a total of $L - 1$ consecutive case **B** are possible, after which $q > L - 1$, resulting in case **A**. At $i = L - 2$, the L surviving PMs after $L - 1$ consecutive case **B** are the following:

$$\begin{aligned}
\mathrm{PM}_{L-2_0} &= \mathrm{PM}_{-1_0} \\
\mathrm{PM}_{L-2_1} &= \mathrm{PM}_{-1_0} + |\alpha_{0_0}| \\
\mathrm{PM}_{L-2_2} &= \mathrm{PM}_{-1_0} + |\alpha_{1_0}| \\
&\vdots \\
\mathrm{PM}_{L-2_{L-1}} &= \mathrm{PM}_{-1_0} + |\alpha_{L-2_0}|.
\end{aligned}$$

Much like the case considered in the proof for Theorem 7, the above analysis shows that at most $L - 1$ bit estimations are required to guarantee the exact same results as the conventional SCL. Thus, the theorem is valid also with (4.47). □

The result of Theorem 7 provides an exact number of path forks in Rate-1 nodes for each list size in SCL decoding in order to guarantee error-correction performance preservation. The Rate-1 node decoder of [19] empirically states that two path forks are required to preserve the error-correction performance. The following remarks are the direct results of Theorem 7.

Remark 1. *The Rate-1 node decoder of [19] for $L = 2$ is redundant.*

Theorem 7 states that for a Rate-1 node of length 2^t when $L = 2$, the number of path splitting is $\min(L - 1, 2^t) = 1$. Therefore, there is no need to split the path after the least reliable bit is estimated. [19] for $L = 2$ is thus redundant.

Remark 2. *The Rate-1 node decoder of [19] falls short in preserving the error-correction performance for higher rates and larger list sizes.*

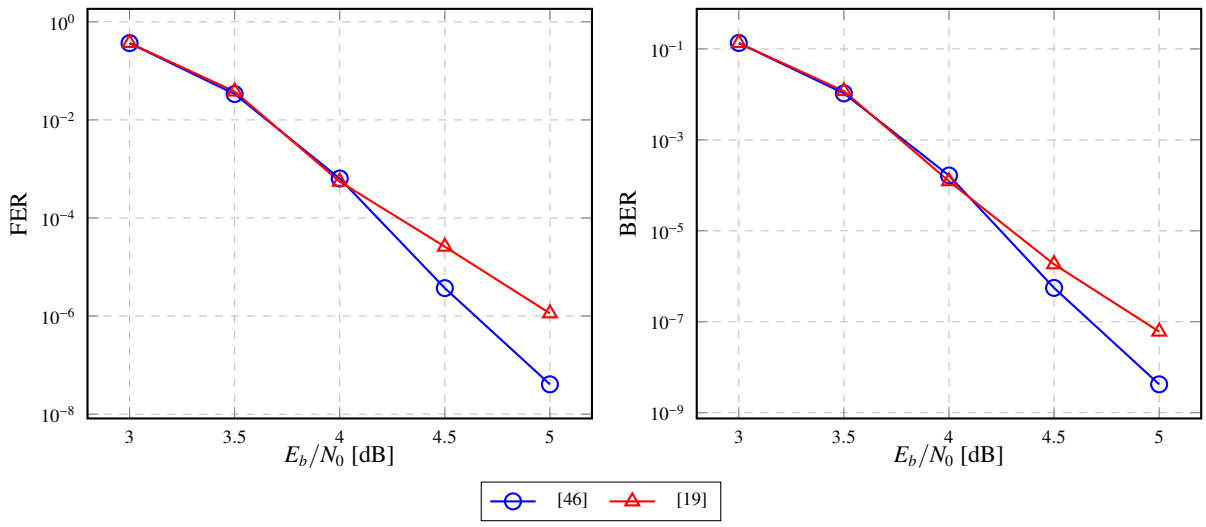


Figure 4.7: FER and BER performance comparison of SSCL [46] and the empirical method of [19] for $\mathcal{P}(1024, 860)$ when $L = 128$. The CRC length is 32.

For codes of higher rates, the number of Rate-1 nodes of larger length increases [37]. Therefore, when the list size is also large, $\min(L-1, 2^t) \gg 2$. The gap between the empirical method of [19] and the result of Theorem 7 can introduce significant error-correction performance loss. Figure 4.7 provides the FER and BER of decoding a $\mathcal{P}(1024, 860)$ code with SSCL of [46] and the empirical method of [19] when the list size is 128. It can be seen that the error-correction performance loss reaches 0.25 dB at FER of 10^{-5}. In the next section, we show that the number of path forks can be tuned for each list size to find a good trade-off between the error-correction performance and the speed of decoding.

Theorem 8. *In SSCL-SPC decoding with list size L, the number of path forks in a SPC node of length 2^t required to get the exact same results as the conventional SSCL-SPC decoder is*

$$\min\left(L, 2^t\right). \tag{4.48}$$

Following the time step calculation of SSCL-SPC, the proposed technique in Theorem 8 results in $T_{\text{Fast-SSCL-SPC}_{\text{SPC}}}(2^t, 2^t - 1) = \min\left(L, 2^t\right) + 1$ which improves the required number of time steps to decode SPC nodes when $L < 2^t$. Every bit after the L-th can be obtained through hard decision on the LLR as in (4.37) without the need for path splitting. In case $\min\left(L, 2^t\right) = 2^t$, the paths need to be split for all bits of the node and the decoding automatically reverts to the process described in

[37]. The proof of the theorem is nevertheless valid for both $L < 2^t$ and $L \geq 2^t$.

Proof. In order to prove Theorem 8, we note that the first step is to initialize the PMs based on (4.33). Therefore, the least reliable bit needs to be estimated first. For the bits other than the least reliable bit, the PMs are updated based on (4.34). However, the term $(1 - 2\gamma)|\alpha_{i_{\min}}|$ is constant for all the bit estimations in the same path. Therefore, we can define a new set of $2^t - 1$ LLR values as

$$\alpha_{i_m} = \alpha_i + \operatorname{sgn}(\alpha_i)(1 - 2\gamma)|\alpha_{i_{\min}}|, \tag{4.49}$$

for $i \neq i_{\min}$ and $0 \leq i_m < 2^t - 1$, which results in

$$|\alpha_{i_m}| = |\alpha_i| + (1 - 2\gamma)|\alpha_{i_{\min}}|. \tag{4.50}$$

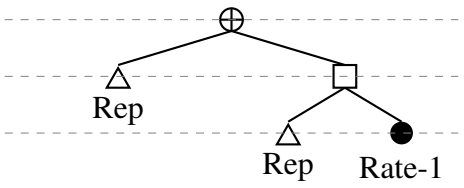
The problem is now reduced to a Rate-1 node of length $2^t - 1$ which, with the result of Theorem 7, can be decoded by considering only $\min(L - 1, 2^t - 1)$ path splitting. Adding the bit estimation for $i_{\min}$, SPC nodes can be decoded by splitting paths $\min(L, 2^t)$ times while guaranteeing the same results as in SSCL-SPC. Theorem 8 is consequently proven. □

The effectiveness of hard decision decoding after the $\min(L - 1, 2^t)$-th bit in Rate-1 nodes and the $\min(L, 2^t)$-th bit in SPC nodes is due to the fact that the bits with high absolute LLR values are more reliable and less likely to incur path splitting. However, whether path splitting must occur or not depends on the list size L. The proposed Rate-1 node decoder is used in Fast-SSCL and Fast-SSCL-SPC algorithms and the proposed SPC node decoder is used in Fast-SSCL-SPC, while the decoders for Rate-0 and Rep nodes remain similar to those used in SSCL [46] such that

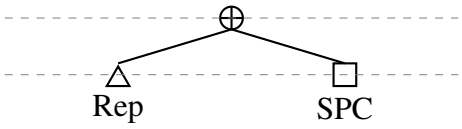
$$T_{\text{Fast-SSCL}_{\text{Rate-0}}}(2^t, 0) = T_{\text{Fast-SSCL-SPC}_{\text{Rate-0}}}(2^t, 0) = 1, \tag{4.51}$$

$$T_{\text{Fast-SSCL}_{\text{Rep}}}(2^t, 1) = T_{\text{Fast-SSCL-SPC}_{\text{Rep}}}(2^t, 1) = 2. \tag{4.52}$$

It should be noted that the number of path forks is directly related to the number of time steps required in the decoding process [20]. Therefore, when $L < 2^t$, the time step requirement of SPC nodes based on Theorem 8 is two time steps more than the time step requirement of Rate-1 nodes as in Theorem 7. However, if SPC nodes are not taken into account as in Fast-SSCL decoding, the polar code tree needs to be traversed to find Rep nodes and Rate-1 nodes as shown in Figure 4.8.



(a) SSCL (Fast-SSCL)



(b) SSCL-SPC (Fast-SSCL-SPC)

Figure 4.8: (a) SSCL (Fast-SSCL), and (b) SSCL-SPC (Fast-SSCL-SPC) decoding tree for $\mathcal{P}(8,4)$ and $\{u_0, u_1, u_2, u_4\} \in \mathcal{F}$.

Table 4.2: Time-Step Requirements of Decoding Different Nodes of Length 2^t with List Size L.

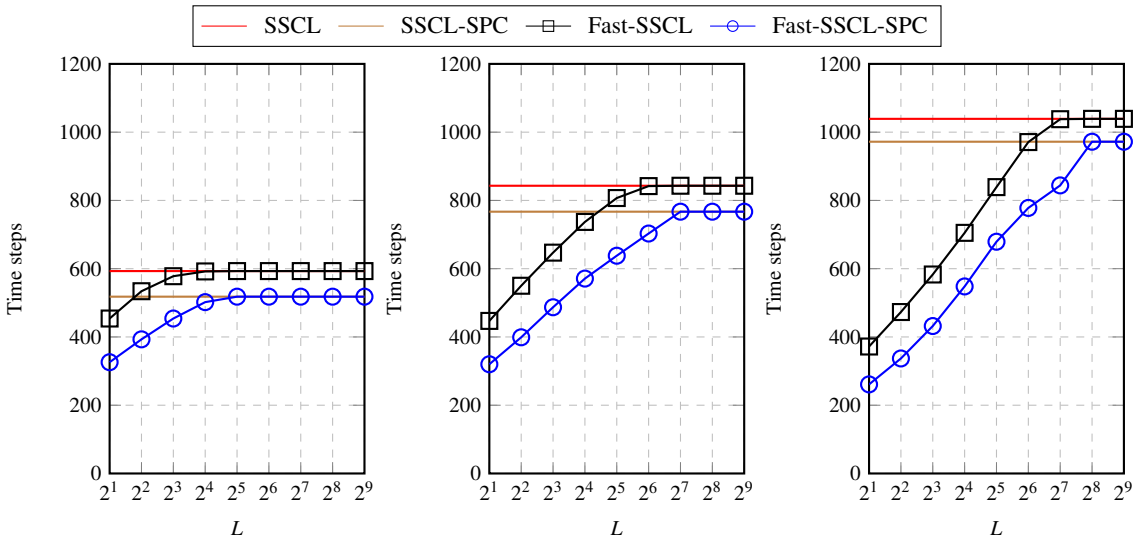
Algorithm	Rate-0	Rep	Rate-1	SPC
SCL	$2^{t+1}-2$	$2^{t+1}-1$	$3\times 2^t-2$	$3\times 2^t-3$
SSCL	1	2	2^t	2^t+2t-2
SSCL-SPC	1	2	2^t	2^t+1
Fast-SSCL	1	2	$\min(L-1,2^t)$	$2t+\sum_{i=1}^{t-1}\min(L-1,2^{t-i})$
Fast-SSCL-SPC	1	2	$\min(L-1,2^t)$	$\min(L,2^t)+1$

For a SPC node of length 2^t, this will result in additional time step requirements as

$$T_{\text{Fast-SSCL}_{\text{SPC}}}(2^t, 2^t-1) = 2t-2+T_{\text{Fast-SSCL}_{\text{Rep}}}(2,1)+\sum_{i=1}^{t-1} T_{\text{Fast-SSCL}_{\text{Rate-1}}}(2^i, 2^i).$$

For example, for a SPC node of length 64, Fast-SSCL with $L=4$ results in time-step requirement of $T_{\text{Fast-SSCL}_{\text{SPC}}}(64,63)=26$, while Fast-SSCL-SPC with $L=4$ results in time-step requirement of $T_{\text{Fast-SSCL-SPC}_{\text{SPC}}}(64,63)=5$. Table 4.2 summarizes the number of time steps required to decode each node with different decoding algorithms.

In practical polar codes, there are many instances where $L-1<2^t$ for Rate-1 nodes and using the Fast-SSCL algorithm can significantly reduce the number of required decoding time steps with respect to SSCL. Similarly, there are many instances where $L<2^t$ for SPC nodes and using the Fast-SSCL-SPC algorithm can significantly reduce the number of required decoding time steps with respect to SSCL-SPC. Figure 4.9 shows the savings in time step requirements of a polar code with three different rates. It should be noted that as the rate increases, the number of Rate-1 and SPC nodes increases. This consequently results in more savings by going from SSCL (SSCL-SPC) to Fast-SSCL (Fast-SSCL-SPC).



(a) $\mathcal{P}(1024, 256)$ (b) $\mathcal{P}(1024, 512)$ (c) $\mathcal{P}(1024, 768)$

Figure 4.9: Time-step requirements of SSCL, SSCL-SPC, Fast-SSCL, and Fast-SSCL-SPC decoding of (a) $\mathcal{P}(1024, 256)$, (b) $\mathcal{P}(1024, 512)$, and (c) $\mathcal{P}(1024, 768)$.

Speed Optimization

The analysis in Section 4.2 provides exact reformulations of SSCL and SSCL-SPC decoders without introducing any error-correction performance loss. However, in practical polar codes, there are fewer required path forks for Fast-SSCL and Fast-SSCL-SPC in order to match the error-correction performance of SSCL and SSCL-SPC, respectively.

Without loss of generality, let us consider $L - 1 < 2^t$ for Rate-1 nodes and $L < 2^t$ for SPC nodes such that Fast-SSCL and Fast-SSCL-SPC result in higher decoding speeds than SSCL and SSCL-SPC, respectively. Let us now consider $S_{\text{Rate-1}}$ be the number of path forks in a Rate-1 node of length 2^t, and S_{SPC} be the number of path forks in a SPC node of length 2^t where $S_{\text{Rate-1}} \leq L - 1$ and $S_{\text{SPC}} \leq L$. It should be noted that $S_{\text{Rate-1}} = L - 1$ and $S_{\text{SPC}} = L$ result in optimal number of path forks as presented in Theorem 7 and Theorem 8, respectively. The smaller the values of $S_{\text{Rate-1}}$ and S_{SPC}, the faster the decoders of Fast-SSCL and Fast-SSCL-SPC. Similar to (4.36) and (4.48), the new number of required path forks for Rate-1 and SPC nodes can be stated as $\min(S_{\text{Rate-1}}, 2^t)$ and $\min(S_{\text{SPC}}, 2^t)$, respectively.

The definition of the parameters $S_{\text{Rate-1}}$ and S_{SPC} provides a trade-off between error-correction performance and speed of Fast-SSCL and Fast-SSCL-SPC. Let us consider CRC-aided Fast-SSCL

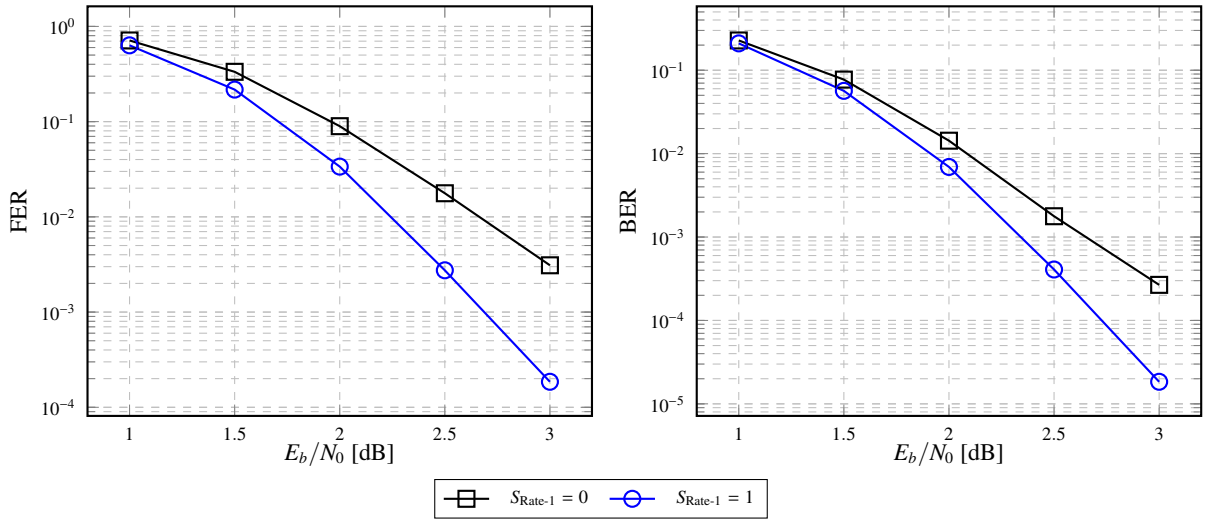


Figure 4.10: FER and BER performance comparison of Fast-SSCL decoding of $\mathcal{P}(1024, 512)$ for $L = 2$ and different values of $S_{\text{Rate-1}}$. The CRC length is 16.

decoding of $\mathcal{P}(1024, 512)$ with CRC length 16. Figure 4.10 shows that for $L = 2$, choosing $S_{\text{Rate-1}} = 0$ results in significant FER and BER error-correction performance degradation. Therefore, when $L = 2$, the optimal value of $S_{\text{Rate-1}} = 1$ is used for Fast-SSCL. The optimal value of $S_{\text{Rate-1}}$ for $L = 4$ is 3. However, as shown in Figure 4.11, $S_{\text{Rate-1}} = 1$ results in almost the same FER and BER performance as the optimal value of $S_{\text{Rate-1}} = 3$. For $L = 8$, the selection of $S_{\text{Rate-1}} = 1$ results in ~0.1 dB of error-correction performance degradation at FER $= 10^{-5}$ as shown in Figure 4.12. However, selecting $S_{\text{Rate-1}} = 2$ removes the error-correction performance gap to the optimal value of $S_{\text{Rate-1}} = 7$. In the case of CRC-aided Fast-SSCL-SPC decoding of $\mathcal{P}(1024, 512)$ with 16 bits of CRC, selecting $S_{\text{Rate-1}} = 1$ and $S_{\text{SPC}} = 3$ for $L = 4$ results in almost the same FER and BER performance as the optimal values of $S_{\text{Rate-1}} = 3$ and $S_{\text{SPC}} = 4$ as shown in Figure 4.13. As illustrated in Figure 4.14 for $L = 8$, the selection of $S_{\text{Rate-1}} = 2$ and $S_{\text{SPC}} = 4$ provides similar FER and BER performance as the optimal values of $S_{\text{Rate-1}} = 7$ and $S_{\text{SPC}} = 8$.

4.2.1 Decoder Architecture

To evaluate the impact of the proposed techniques on a practical case, a SCL-based polar code decoder architecture implementing Fast-SSCL and Fast-SSCL-SPC has been designed. Its basic structure is inspired to the decoders presented in [14], [37], and it is portrayed in Figure 4.15. The decoding flow follows the one portrayed in Section 2.4 for a list size L. This means that

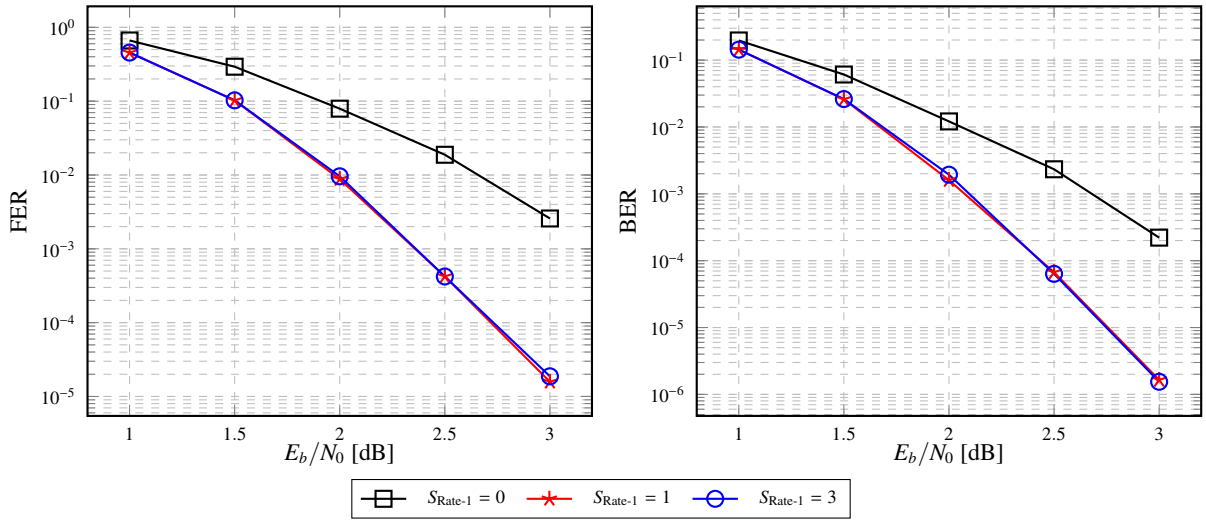


Figure 4.11: FER and BER performance comparison of Fast-SSCL decoding of $\mathcal{P}(1024, 512)$ for $L = 4$ and different values of $S_{\text{Rate-1}}$. The CRC length is 16.

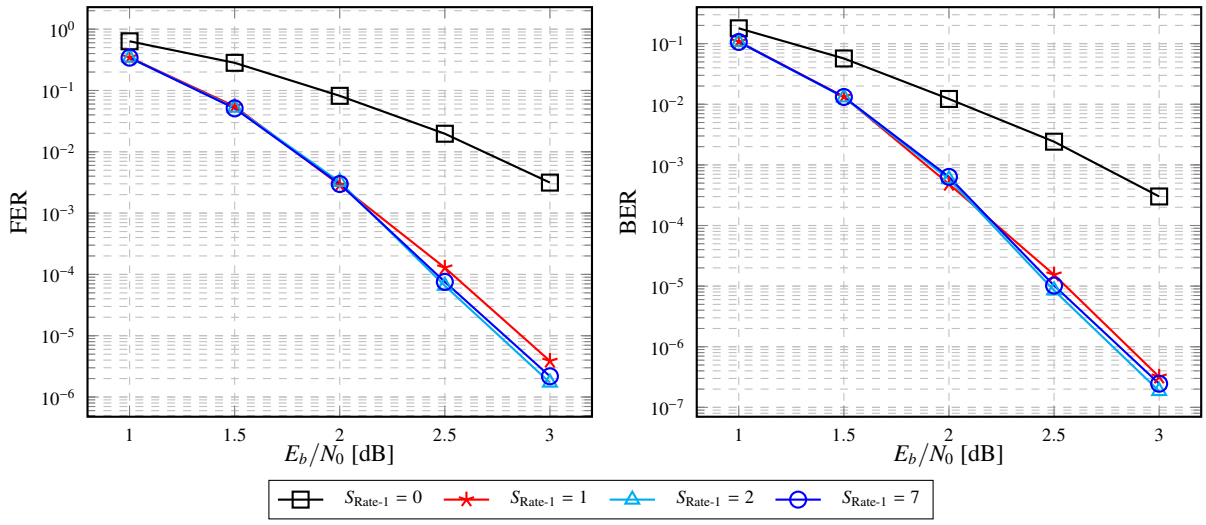


Figure 4.12: FER and BER performance comparison of Fast-SSCL decoding of $\mathcal{P}(1024, 512)$ for $L = 8$ and different values of $S_{\text{Rate-1}}$. The CRC length is 16.

the majority of the datapath and of the memory are replicated L times, and work concurrently on different candidate codewords and the associated LLR values.

Starting from the tree root, the tree is descended by recursively computing (2.11) and (2.8) on left and right branches respectively at each tree stage t, with a left-first rule. The computations are performed by L sets of P_e PEs, where each set can be considered a standalone SC decoder, and P_e

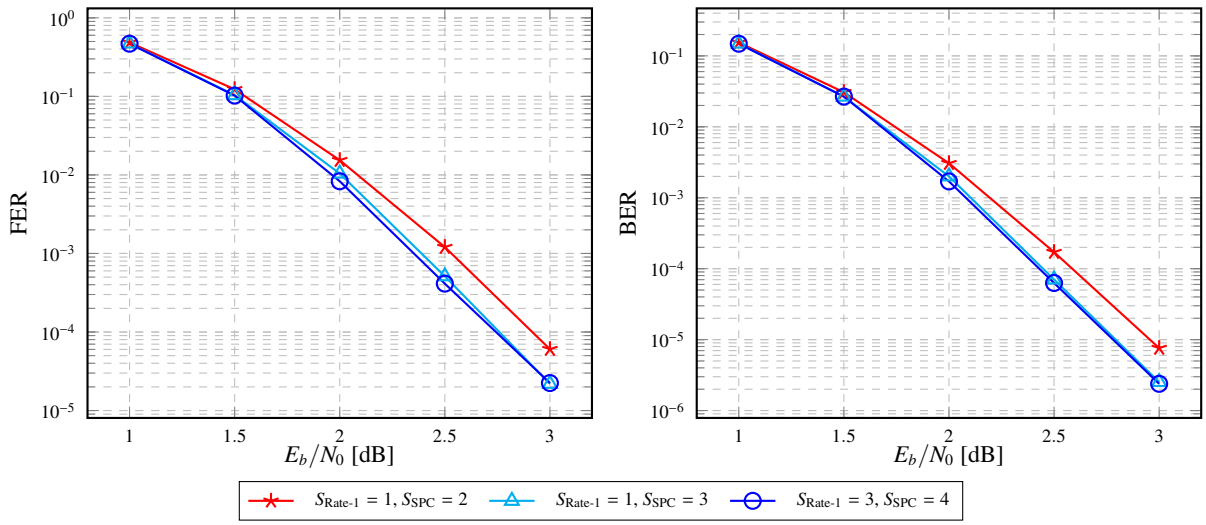


Figure 4.13: FER and BER performance comparison of Fast-SSCL-SPC decoding of $\mathcal{P}(1024, 512)$ for $L = 4$ and different values of $S_{\text{Rate-1}}$ and S_{SPC}. The CRC length is 16.

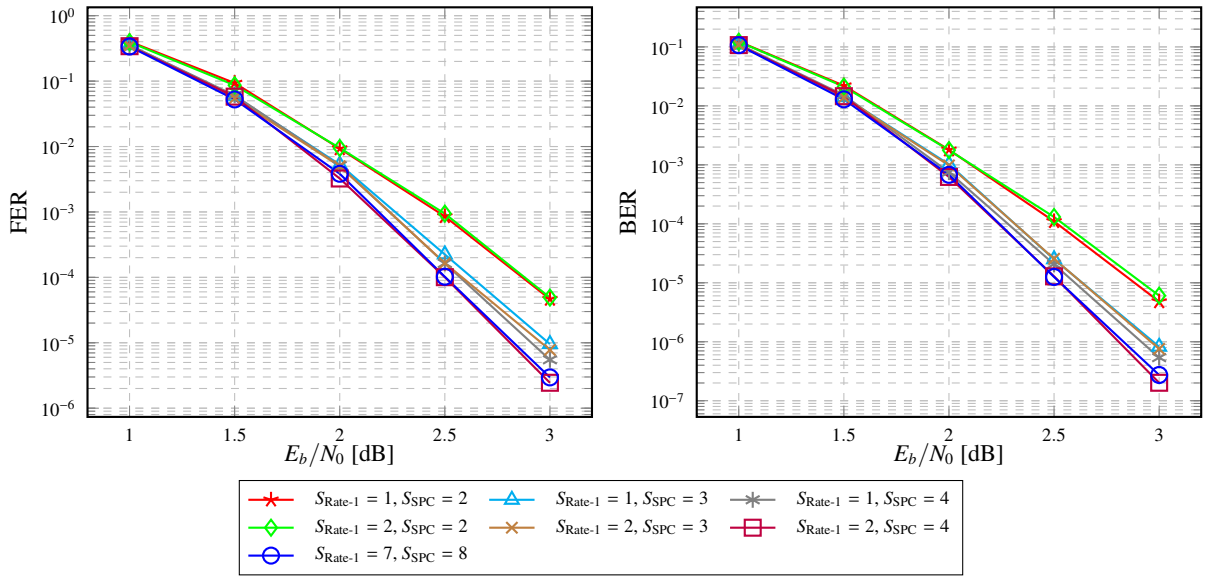


Figure 4.14: FER and BER performance comparison of Fast-SSCL-SPC decoding of $\mathcal{P}(1024, 512)$ for $L = 8$ and different values of $S_{\text{Rate-1}}$ and S_{SPC}. The CRC length is 16.

is a power of 2. In case $2^t > 2P$, (2.11) and (2.8) require $2^t/(2P)$ time steps to be completed, while otherwise needing a single time step. The updated LLR values are stored in dedicated memories.

The internal structure of PEs is shown in Figure 4.16. Each PE receives as input two LLR

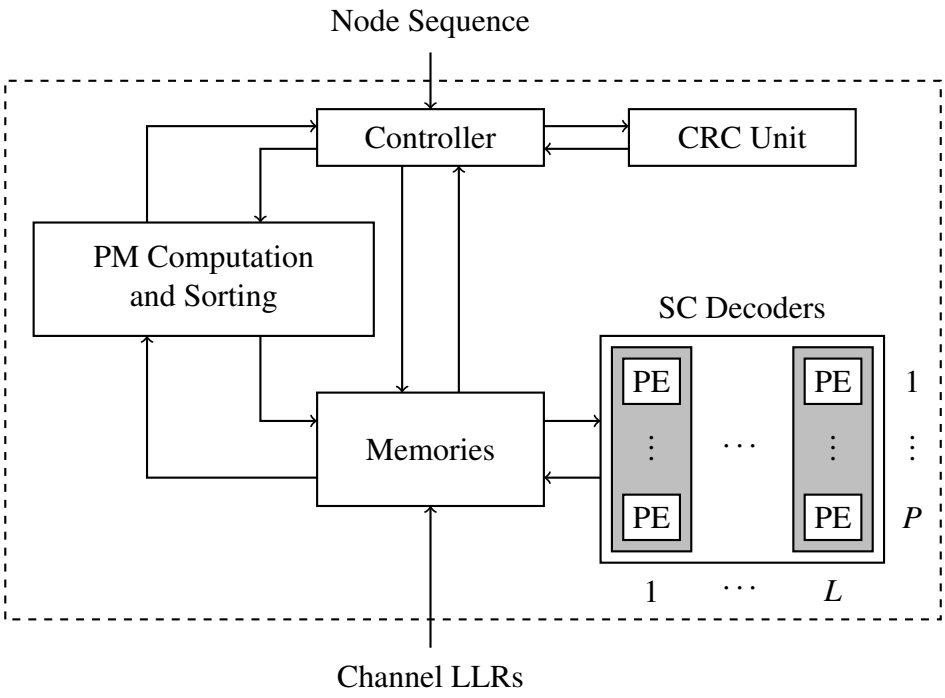


Figure 4.15: Fast-SSCL (Fast-SSCL-SPC) decoder architecture.

values, outputting one. The computations for both (2.11) and (2.8) are performed concurrently, and the output is selected according to i_t, that represents the t-th bit of the index i, where $0 \leq i < N$. The index i is represented with $t_{\max} = \log_2 N$ bits, and identifies the next leaf node to be estimated, and can be composed by observing the path from the root node to the leaf node. From stage $t_{\max}$ down to 0, for every left branch we set the corresponding bit of i to 0, and to 1 for every right branch.

When a leaf node is reached, the controller checks Node Sequence, identifying the leaf node as an information bit or a frozen bit. In case of a frozen bit, the paths are not split, and the bit is estimated only as 0. All the L path memories are updated with the same bit value, as are the LLR memories and the β memories. On the other hand, in case of an information bit, both 0 and 1 are considered. The paths are duplicated and the PMs are calculated for the $2L$ candidates according to (2.22). They are subsequently filtered through the sorter module, designed for minimum latency. Every PM is compared to every other in parallel: dedicated control logic uses the resulting signals to return the values of the PMs of the surviving paths and the newly estimated bits they are associated with. The latter are used to update the LLR memories, the β memories and the path memories,

while also being sent to the CRC calculation module to update the remainder.

All memories in the decoder are implemented as registers: this allows the LLR and β values to be read, updated by the PEs, and written back in a single clock cycle. At the same time, the paths are either updated, or split and updated (depending on the constituent code), and the new PMs computed. In the following clock cycle, in case the paths were split, the PMs are sorted, paths are discarded and the CRC value updated. In case paths were not split, the PMs are not sorted, and the CRC update occurs in parallel with the following operation.

Memory Structure

The decoding flow described above relies on a number of memories that are shown in Figure 4.17. The channel memory stores the N LLR values received from the channel at the beginning of the decoding process. Each LLR value is quantized with Q_α bits, and represented with sign and magnitude. The high and low stage memories store the intermediate α computed in (2.11) and (2.8). The high stage memory is used to store LLR values related to stages with nodes of size greater than P_e. The number of PEs determines the number of concurrent (2.11) or (2.8) that can be performed: for a node in stage t, where $2^t > 2P$, a total of $2^t/(2P)$ time steps are needed to descend to the lower tree level. The depth of the high stage memory is thus $\sum_{j=\log_2 P+1}^{t_{\max}-1} 2^j/P = N/P - 2$, while its width is $Q_\alpha \times P$. On the other hand, the low stage memory stores the LLR values for stages where $2^t \leq 2P$: the width of this memory is Q_α, while its depth is defined as $\sum_{j=0}^{\log_2 P-1} P/2^j = 2P-2$. Both high and low stage memory words are reused by nodes belonging to the same stage t, since once a subtree has been completely decoded, its LLR values are not needed anymore. While high and low stage memories are different for each path, the channel LLR values are shared among the L datapaths. Table 4.3 summarizes the memory read and write accesses for the aforementioned LLR memories. When $2^t = 2P$, $2P$ LLR values are read from the high stage memory, and the P_e resulting LLR values are written in the low stage memory. The channel memory is read at $t_{\max}$ only.

Each of the L candidate codewords is stored in one of the N-bit path memories, updated after every bit estimation. The β memories hold the β values for each stage from 0 to $t_{\max} - 1$, for a total of $N - 1$ bits each. Each time a bit is estimated, all the β values it contributes to are concurrently updated. When the decoding of the left half of the SC decoding tree has been completed, the β memories are reused for the right half. Finally, the PM memories store the L PM values computed in (2.22).

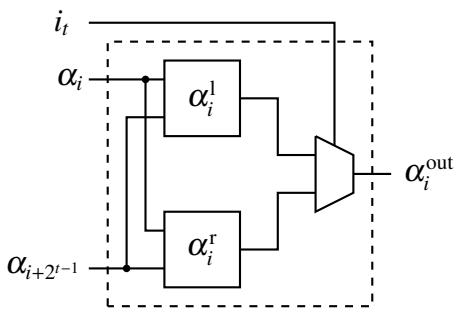


Figure 4.16: PE architecture.

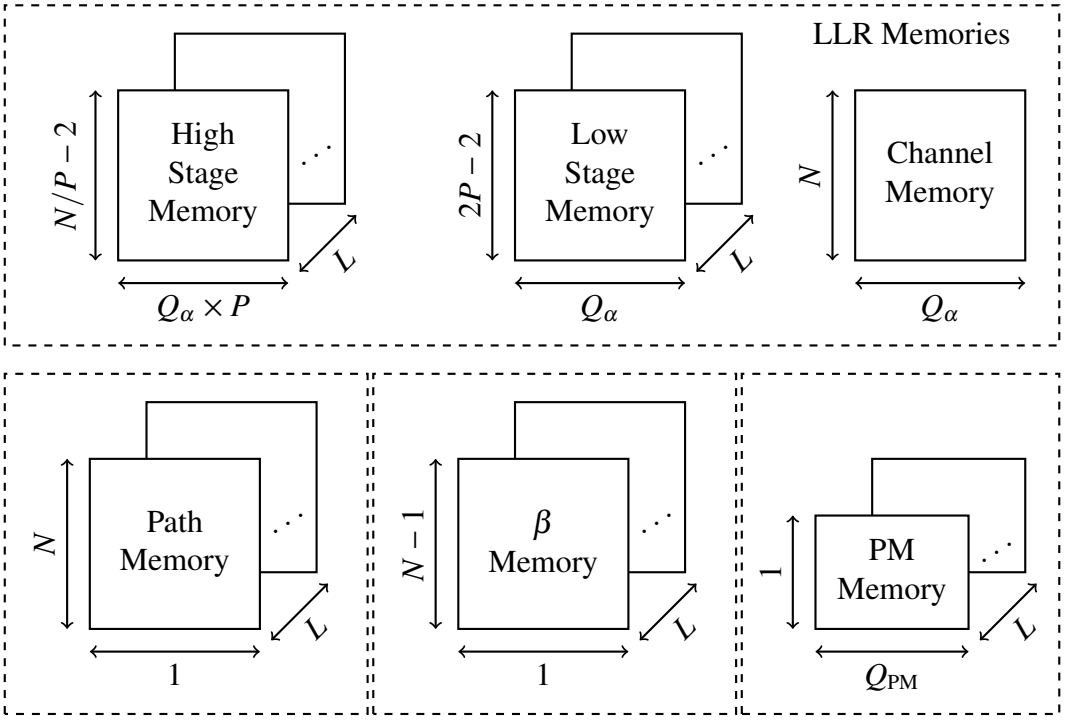


Figure 4.17: Memory architecture.

Special Nodes

The decoding flow and memory structure described before implement the standard SCL decoding algorithm. The SSCL, SSCL-SPC and the proposed Fast-SSCL and Fast-SSCL-SPC algorithms demand modifications in the datapath to accommodate the simplified computations for Rate-0, Rate-1, Rep and SPC nodes.

As with standard SCL, the pattern of frozen and information bits is known a priori given a

Table 4.3: LLR Memory Access.

Stage	READ	WRITE
$t = t_{\max}$	Channel	High Stage
$\log_2 P + 1 < t < t_{\max}$	High Stage	High Stage
$t = \log_2 P + 1$	High Stage	Low Stage
$t < \log_2 P + 1$	Low Stage	Low Stage

polar code structure, the same can be said for special nodes. In the modified architecture, the Node Sequence input in the controller (see Figure 4.15) is not limited to the frozen/information bit pattern, but it includes the type of encountered nodes, their size and the tree stage in which they are encountered. Table 4.4 summarizes the content of Node Sequence depending on the type of node for SSCL and SSCL-SPC, while in case of Fast-SSCL and Fast-SSCL-SPC Node Sequence is detailed in Table 4.5. The node stage allows the decoder to stop the tree exploration at the right level, and the node type identifies the operations to be performed. Each of the four node types is represented with one or more decoding phases, each of which involves a certain number of codeword bits, identified by the node size parameter. Finally, the frozen bit parameter identifies a bit or set of bits as frozen or not. To limit the decoder complexity, the maximum node stage for special nodes is limited to $t = \log_2 P$, thus the maximum node size is P_e. If the code structure identifies special nodes with node size larger than P_e, they are considered as composed by a set of P_e-size special nodes.

- Rate-0 nodes are identified in the Node Sequence with a single decoding phase. No path splitting occurs, and all the 2^t node bits are set to 0. The PM update requires a single time step, as discussed in [37].

- Rate-1 nodes are composed of a single phase in both SSCL and SSCL-SPC, in which paths are split 2^t times. In case of Fast-SSCL and Fast-SSCL-SPC, each Rate-1 is divided into two phases. The first takes care of the $\min(S_{\text{Rate-1}}, 2^t)$ path forks, requiring as many time steps, while the second sets the remaining $2^t - \min(S_{\text{Rate-1}}, 2^t)$ bits according to (4.37) and updates the PM according to (4.47). This second phase takes a single time step.

- Rep nodes are identified by two phases in the Node Sequence, the first of which takes care of the $2^t - 1$ frozen bits similarly as Rate-0 nodes do, and the second estimates the single

Table 4.4: Node Sequence Input Information for SSCL and SSCL-SPC.

Node Type	Node Stage	Node Size	Frozen
RATE0	t	2^t	1
RATE1	t	2^t	0
REP1	t	$2^t - 1$	1
REP2	t	1	0
DESCEND	Next Node	Next Node	Next Node
LEAF	0	1	$0/1$
SPC1	t	1	1
SPC2	t	$2^t - 1$	0
SPC3	t	2^t	0

Table 4.5: Node Sequence Input Information for Fast-SSCL and Fast-SSCL-SPC.

Node Type	Node Stage	Node Size	Frozen
RATE0	t	2^t	1
RATE1-1	t	$\min(S_{\text{Rate-1}}, 2^t)$	0
RATE1-2	t	$2^t - \min(S_{\text{Rate-1}}, 2^t)$	0
REP1	t	$2^t - 1$	1
REP2	t	1	0
DESCEND	Next Node	Next Node	Next Node
LEAF	0	1	$0/1$
SPC1	t	1	1
SPC2-1	t	$\min(S_{\text{SPC}}, 2^t)$	0
SPC2-2	t	$2^t - \min(S_{\text{SPC}}, 2^t) - 1$	0
SPC3	t	2^t	0

information bit. Each of these two phases lasts a single time step.

- SPC nodes are split in three phases in the original SSCL-SPC formulation. The first phase takes care of the frozen bit, and computes both (2.17) and (2.18), initializing the PM as (4.33) in a time step. The extraction of the least reliable bit in (2.17) is performed through a comparison tree that carries over both the index and the value of the LLR.

The second phase estimates the $2^t - 1$ information bits, splitting the path as many times in as many time steps. During this phase, each time a bit is estimated, it is XORed with the previous β values: this operation is useful to compute (4.35). The update of $\beta_{i_{\min}}$ is finally performed in the third phase, that takes a single time step. Moving to Fast-SSCL-SPC, the second SPC phase is split in two, similarly to what happens to the Rate-1 node.

- Descend is a non-existing node type that is inserted for one clock cycle in Node Sequence for control purposes after every special node. The node size and stage associated with this label are those of the following node. The Descend node type is used by the controller module.
- Leaf nodes identify all nodes that can be found at $t = 0$, for which the standard SCL algorithm applies.

The decoding of special nodes requires a few major changes in the decoder architecture.

- *Path Memory*: each path memory is an array of N registers, granting concurrent access to all bits with a 1-bit granularity. In SCL, the path update is based on the combination of a write enable signal, the codeword bit index i that acts as a memory address, and the value of the estimated bit after the PMs have been sorted and the surviving paths identified. Figure 4.18 shows the path memory access architecture for Fast-SSCL-SPC. Unlike SCL, the path memory is not always updated with the estimated bit $\hat{u}$. Thus, the SCL datapath is bypassed according to the node type. When Node Sequence identifies RATE0, REP1 and SPC1 nodes that consider frozen bits, the path memory is updated with 0 values. The estimated bit $\hat{u}$ is chosen as input for RATE1-1, REP2, SPC2-2 and LEAF nodes, where the path is split. RATE1-2 and SPC2-2 nodes estimate the bits through hard decision on the LLR values, while in the SPC3 case the update considers the result of (4.35). At the same time. whenever the estimated bits are more than one, the corresponding bits in the path memory must be concurrently updated. Thus, the address becomes a range of addresses for RATE0, RATE1-2, REP1 and SPC2-2.
- β *Memory*: the update of this memory depends on the value of the estimated bit. In order to limit the latency cost of these computations, concurrently to the estimation of $\hat{u}$, the updated values of all the bits of the β memory are computed assuming both $\hat{u} = 0$ and $\hat{u} = 1$. The actual value of $\hat{u}$ is used as a selection signal to decide on the two alternatives. The β memory in SCL, unlike the path memory, already foresees the concurrent update of multiple entries

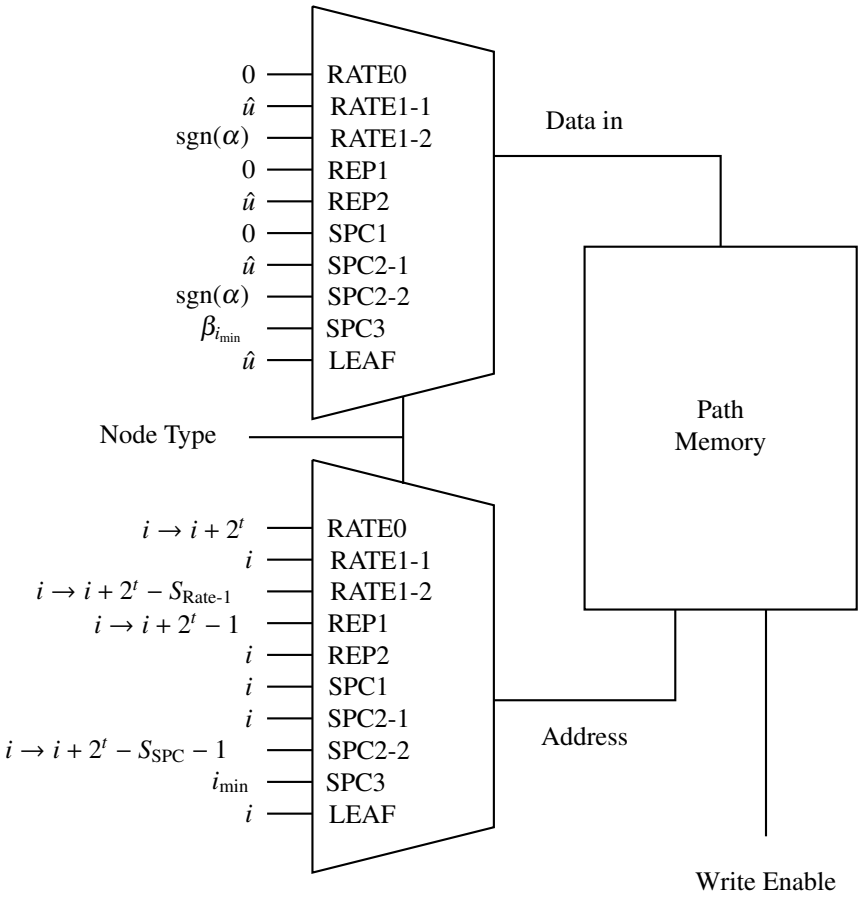


Figure 4.18: Path memory access architecture for Fast-SSCL-SPC.

that are selected based on the bit index i. Given an estimated leaf node, the β values of all the stages that it affects are updated: in fact, since as shown in (2.9) the update of β values is at most a series of XORs, it is possible to distribute this operation in time. The same can be said of multi-bit (2.9) updates. To implement Fast-SSCL-SPC, the β update selection logic must be modified to foresee the special nodes, similar to that portrayed in Figure 4.18 for the path memory. For RATE0, REP1, and SPC1, the $\hat{u} = 0$ update is always selected. RATE1-1, REP2, SPC2-1 and LEAF nodes maintain the standard SCL selection based on the actual value of $\hat{u}$. The update for SPC3 case is based on $\beta_{i_{\min}}$. For RATE1-2 and SPC1-2, the selection is based on the XORed sign bits of the LLR values read from the memory.

- *PM Calculation*: this operation is performed, in the original SCL architecture and for leaf

nodes in general according to (2.22). The paths and associated PMs are split and sorted every time an information bit is estimated, while PMs are updated without sorting when frozen bits are encountered. While the sorting architecture remains the same, the implementation of the proposed algorithm requires a different PM update structure for each special node. Unlike with leaf nodes, the LLR values needed for the PM update in special nodes are not the output of PEs, and are read directly from the LLR memories. Additional bypass logic is thus needed. For RATE0 and REP1, (4.24) and (4.30) require a summation over up to P_e values, while SPC1 nodes need to perform the minimum α search (2.17): these operations are tackled through adder and comparator trees. RATE1-1, REP2 and SPC2-1 PM updates are handled similarly to the leaf node case, since a single bit at a time is being estimated. RATE1-2, SPC2-2 and SPC3 do not require any PM to be updated.

- *CRC Calculation*: the standard SCL architecture foresees the estimation of a single bit at a time. Thus, the CRC is computed sequentially. However, Rate-0 and Rep nodes in SSCL and SSCL-SPC estimate up to P_e and $P_e - 1$ bits concurrently. Thus, for the CRC operation not to become a latency bottleneck, the CRC calculation must be parallelized by updating the remainder. Following the idea presented in [51], it is possible to allow for variable input sizes with a high degree of resource sharing and limited overall complexity. The circuit is further simplified by the fact that both Rate-0 and Rep nodes guarantee that the estimated bit values are all 0. Figure 4.19 shows the modified CRC calculation module in case $P_e = 64$, where N_{CRC} represents the number of concurrently estimated bits: the estimated bit can be different from 0 only in case of leaf nodes and $t = 1$ Rep nodes, for which a single bit is estimated in any case.

  The Fast-SSCL and Fast-SSCL-SPC architectures follow the same idea, but require additional logic. RATE1-2 and SPC2-2 nodes introduce new degrees of parallelism, as up to $P_e - S_{\text{Rate-1}}$ and $P_e - S_{\text{SPC}}$ bits are updated at the same time. Moreover, it is not possible to assume that these bits are 0 as with RATE0 and REP1. The value of the estimated bit must be taken into account, leading to increased complexity.

- *Controller*: this module in the SCL architecture is tasked with the generation of memory write enables, the update of the codeword bit index i and the stage tracker t, along with the LLR memory selection signals according to Table 4.3 and path enable and duplication signals. It implements a finite state machine that identifies the status of the decoding process.

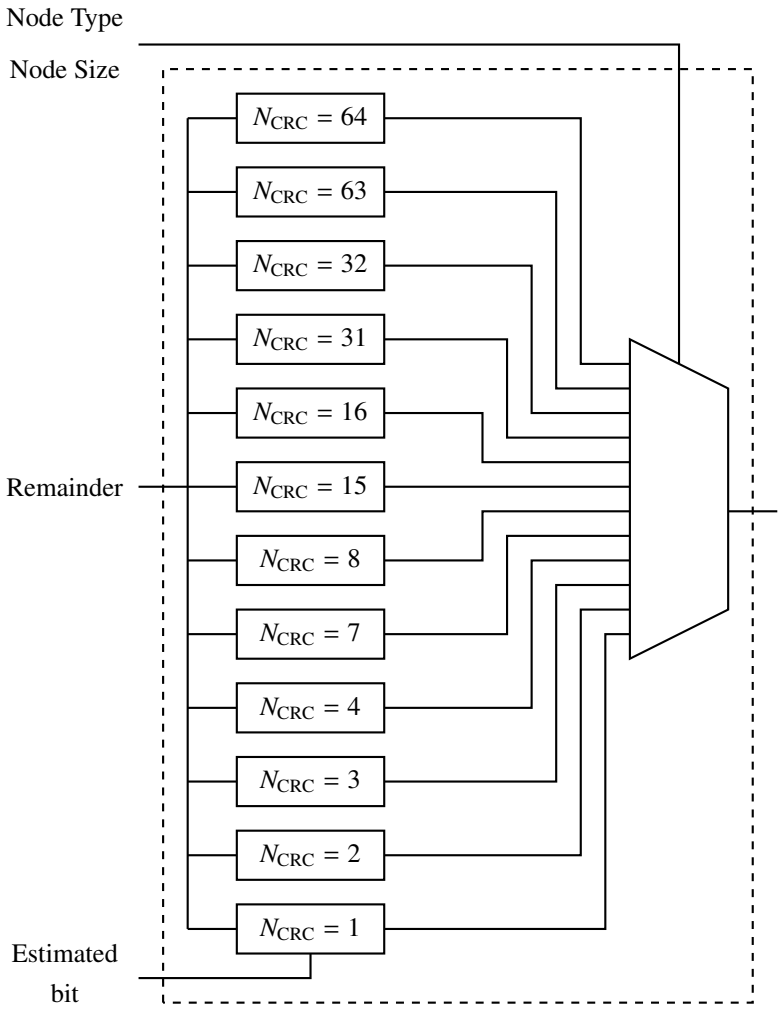


Figure 4.19: CRC architecture for SSCL and SSCL-SPC.

The introduction of special nodes demands that most of the control signal generation logic is modified. Of particular importance is the fact that, in the SCL architecture, the update of i is bound to having reached a leaf node, i.e. $t = 0$. In Fast-SSCL-SPC, it is instead linked to t being equal to the special node stage. The index i is moreover incremented of the amount of bits estimated in a single time step, depending on the type of node. Memory write enables are also bound to having reached the special node stage, and not only to $t = 0$.

Table 4.6: TSMC 65 nm Implementation Results for $N = 1024$, $R = 1/2$, $L = 2$, $P_e = 64$.

Implementation	Rate-1	Rate-0			Rep			SPC			Area [mm^2]	f [MHz]	T/p [Mb/s]
		LLS	HFS	LCS	LLS	HFS	LCS	LLS	HFS	LCS			
SCL	✗		✗			✗			✗		0.599	1031	389
SSCL	✓	✓			✓						0.643	1031	1108
			✓			✓			✗		0.650	1031	1028
				✓			✓				0.636	1031	642
SSCL-SPC	✓	✓			✓			✓			0.684	1031	1229
			✓			✓			✓		0.694	1031	1088
				✓			✓			✓	0.681	1031	737

Hardware Implementation

The decoder architecture described in this section has been coded in VHDL and synthesized in TSMC 65 nm CMOS technology with Cadence RTL Compiler. Mentor Graphics ModelSim has been used for verification. All reported area occupation values include both cell area and net area.

Table 4.6 details the design choices of the different architectures that have been synthesized. The decoder labelled as **SCL** implements the LLR-based CRC-aided SCL decoding algorithm, targeting a rate $1/2$ code with $N = 1024$, $L = 2$ and making use of 64 PEs for each concurrent path. The quantization $Q(6, 8, 2)$ is used. All memories have been synthesized with flip-flops.

The second type of implementation, **SSCL**, uses the same choice of design parameters as **SCL**, while also taking advantage of the SSCL nodes, i.e. Rate-1, Rate-0 and Rep nodes. Finally, the **SSCL-SPC** implementations add the SPC node to the **SSCL** architecture.

The **SCL** architecture yields an area occupation of 0.599 mm^2, reaching a maximum frequency of 1031 MHz. The achievable coded throughput is of 389 Mb/s.

To limit the hardware overhead with a negligible sacrifice in the achieved speed-up, all **SSCL** and **SSCL-SPC** implementations have a special node parallelism equal to the number of PEs, i.e. 64 in our case. This choice adds complexity to the control unit but simplifies the memory accesses. As shown in Table 4.6, the **SSCL** architecture has been implemented with three different Rate-0 and Rep node structures, a LLS, a HFS, and a LCS. It is worth noting how none of the special node structures impacts the achievable frequency. Consequently, the LLS allows for the highest achievable throughput, and is superior to the HFS, having a lower area occupation as well. As expected, the LCS yields the smallest area occupation, but the considerable throughput degradation makes it not worthy of consideration. The increment in throughput brought by the implementation

Table 4.7: TSMC 65 nm Implementation Results for $\mathcal{P}(1024, 512)$ and $P_e = 64$.

Implementation	L	$S_{\text{Rate-1}}$	S_{SPC}	Area [mm^2]	Frequency [MHz]	Throughput [Mb/s]
SCL	2	✗	✗	0.599	1031	389
	4	✗	✗	0.998	961	363
	8	✗	✗	2.686	722	272
SSCL	2	✗	✗	0.643	1031	1108
	4	✗	✗	1.192	961	1033
	8	✗	✗	2.958	722	776
SSCL-SPC	2	✗	✗	0.684	1031	1229
	4	✗	✗	1.223	961	1146
	8	✗	✗	3.110	722	861
Fast-SSCL	2	1	✗	0.871	885	1579
	4	1	✗	1.536	840	1499
	4	3	✗	1.511	840	1446
	8	2	✗	3.622	722	1053
	8	7	✗	3.588	722	827
Fast-SSCL-SPC	2	1	2	1.048	885	1861
	4	1	3	1.822	840	1608
	4	3	4	1.797	840	1338
	8	2	4	3.975	722	1198
	8	7	8	3.902	722	959

of SSCL ranges between 1.65× and 2.85×, with a cost in area occupation between 6.2% and 8.5%.

Similar results can be observed in case of **SSCL-SPC**. The different design choices within the architecture of Rate-0, Rep, and SPC nodes lead the LLS choice to dominate the others in terms of throughput. The highest achievable throughput is of 1229 Mb/s with an area occupation of 0.684 mm^2. With respect to **SCL**, they correspond to a +14.2% increment in area occupation, along with a 3.16× increment in throughput.

Implementation results for different decoders in this work are provided in Table 4.7 for the design choice taken from Table 4.6. Each decoder has been synthesized with three list sizes ($L = 2, 4, 8$), while the Fast-SSCL and Fast-SSCL-SPC architectures have been synthesized for considering different combinations of $S_{\text{Rate-1}}$ and S_{SPC}. Quantization values are the same used in [37], i.e. 6 bits for LLR values and 8 bits for PMs, with two fractional bits each. All memory elements have been implemented through registers and the area results include both net area and cell area. The reported throughput is coded.

All Fast-SSCL and Fast-SSCL-SPC, regardless of the value of $S_{\text{Rate-1}}$ and S_{SPC}, show a sub-

stantial increase in area occupation with respect to SSCL and SSCL-SPC. The main contributing factors to the additional area overhead are three:

- In SSCL and SSCL-SPC, the CRC computation needs to be parallelized, since in Rep and Rate-0 nodes multiple bits are updated at the same time. However, the bit value is known at design time, since they are frozen bits. This, along with the fact that 0 is neutral in the XOR operations required by CRC calculation, limits the required additional area overhead. On the contrary, in Fast-SSCL and Fast-SSCL-SPC, Rate-1 and SPC nodes update multiple bits within the same time step (SPC2-2 and RATE1-2 stages). In these cases, however, they are information bits, whose values cannot be known at design time: the resulting parallel CRC tree is substantially wider and deeper than the ones for Rate-0 and Rep nodes. Moreover, with increasing number of CRC trees, the selection logic becomes more cumbersome.

- A similar situation is encountered for the β memory update signal. As described in the previous section, the β memory update values are computed assuming both estimated values, and the actual value of $\hat{u}$ is used as a selection signal. In SSCL and SSCL-SPC the multiple-bit update does not constitute a problem since all the estimated bits are 0 and the β memory content does not need to be changed. On the contrary, in Fast-SSCL and Fast-SSCL-SPC, the value of the estimated information bits might change the content of the β memory. Moreover, since β is computed as (2.9), the update of β bits depends on previous bits as well as the newly estimated ones. Thus, an XOR tree is necessary to compute the right selection signal for every information bit estimated in SPC2-2 and RATE1-2 stages.

- The aforementioned modifications considerably lengthen the system critical path. In case of large code length, small list size, or large P_e, the critical path starts in the controller module, in particular in the high stage memory addressing logic, goes through the multiplexing structure that routes LLR values to the PEs, and ends after the PM update. In case of large list sizes or short code length, the critical path passes through the PM sorting and path selection logic, and through the parallel CRC computation. Thus, pipeline registers have been inserted to lower the impact of critical path, at the cost of additional area occupation.

Fast-SSCL and Fast-SSCL-SPC implementations show consistent throughput improvements with respect to previously proposed architectures. The gain is lower than what is shown to be theoretically achievable in Figure 4.9. This is due to the aforementioned pipeline stages, that increase the number of steps needed to complete the decoding of component codes.

Table 4.8: Comparison with State-of-the-Art Decoders.

	Fast-SSCL-SPC			[15]	[16]†	[17]†		[18]†	SSCL-SPC		
L	2	4	8	4	4	2	4	4	2	4	8
P_e	64	64	64	64	64	64	64	256	64	64	64
Area [mm^2]	1.048	1.822	3.975	0.62	0.73	1.03	2.00	0.99	0.68	1.22	3.11
Frequency [MHz]	885	840	722	498	692	586	558	566	1031	961	722
Throughput [Mb/s]	1861	1608	1198	935	551	1844	1578	1515	1229	1146	861
Latency [µs]	0.55	0.64	0.85	1.10	1.86	0.57	0.66	0.69	0.83	0.89	1.19
Area Efficiency [Mb/s/mm^2]	1776	883	301	1508	755	1790	789	1530	1807	939	277

† The results are originally based on TSMC 90 nm technology and are scaled to TSMC 65 nm technology.

Comparison with Previous Works

The Fast-SSCL-SPC hardware implementation of $\mathcal{P}(1024, 512)$ with $P_e = 64$, presented in this work, is compared with the state-of-the-art architectures in [15]–[18] and the results are provided in Table 4.8. The architectures presented in [16]–[18] were synthesized based on 90 nm technology: for a fair comparison, their results have been converted to 65 nm technology using a factor of $90/65$ for the frequency and a factor of $(65/90)^2$ for the area. The synthesis results in [15] were carried out in 65 nm technology but reported in 90 nm technology. Therefore, a reverse conversion was applied to convert the results back to 65 nm technology.

The architecture in this chapter shows 72% higher throughput and 42% lower latency with respect to the multibit decision SCL decoder architecture of [15] for $L = 4$. However, the area occupation of [15] is smaller, leading to a higher area efficiency than the design in this chapter.

The symbol-decision SCL decoder architecture of [16] shows lower area occupation than the design in this chapter for $L = 4$ but it comes at the cost of lower throughput and higher latency. Our decoder architecture achieves 192% higher throughput and 66% lower latency than [16] which resulted in 17% higher area efficiency.

The high throughput SCL decoder architecture of [17] for $L = 2$ requires lower area occupation than our design but it comes at the expense of lower throughput and higher latency. Moreover, the design in [17] relies on parameters that need to be tuned for each code, and it is shown in [17] that a change of code can result in more than 0.2 dB error-correction performance loss. For $L = 4$, our decoder not only achieves higher throughput and lower latency than [17], but also it occupies a smaller area. This in turn yields a 12% increase in the area efficiency in comparison with [17].

The multimode SCL decoder in [18] relies on a higher number of PEs than our design: nevert-

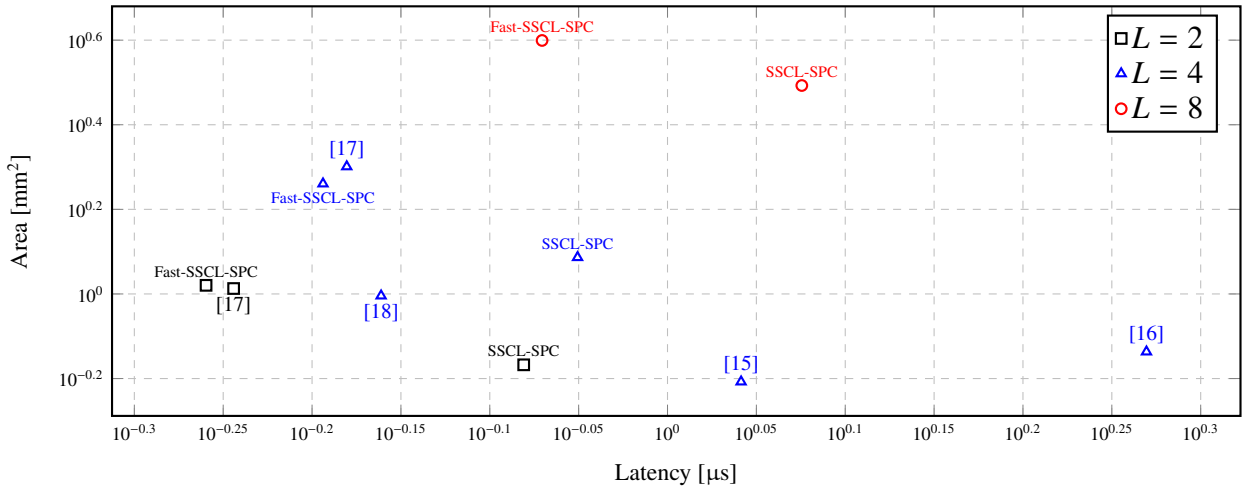


Figure 4.20: Comparison with state-of-the-art decoders.

heless, it yields lower throughput and higher latency than the architecture proposed in this chapter for $L = 4$. It should be noted that [18] is based on the design presented in [17], whose code-specific parameters may lead to substantial error-correction performance degradation. On the contrary, the design in this chapter is targeted for speed and flexibility and can be used to decode any polar code of any length.

Compared to SSCL-SPC, that has the same degree of flexibility of Fast-SSCL-SPC, this decoder achieves 51% higher throughput and 34% lower latency for $L = 2$, and 40% higher throughput and 28% lower latency for $L = 4$. However, the higher area occupation of Fast-SSCL-SPC yields lower area efficiencies than SSCL-SPC for $L = \{2, 4\}$. For $L = 8$, Fast-SSCL-SPC has 39% higher throughput and 29% lower latency than SSCL-SPC, which results in 9% increase in area efficiency. The reason is that for $L = 8$, the sorter is quite large and falls on the critical path. Consequently, the maximum achievable frequency for Fast-SSCL-SPC is limited by the sorter and not by Rate-1 and SPC nodes as opposed to the $L = \{2, 4\}$ case. This results in the same maximum achievable frequency for both designs, hence, higher throughput and area efficiency.

Figure 4.20 plots the area occupation against the decoding latency for all the decoders considered in Table 4.8. For each value of L, Fast-SSCL-SPC has the shortest latency, shown by their leftmost position on the graph.

Chapter 5

Memory-Efficient List Decoding

Although SCL decoding significantly improves the error-correction performance of polar codes, it comes at the cost of higher area occupation when implemented in hardware. In particular, it was shown in [20] that the high area requirement of SCL decoding is mostly dominated by its memory usage. In this chapter, a PSCL decoding algorithm is proposed in order to reduce the memory requirements associated with SCL decoding. More specifically, PSCL decoding performs SCL decoding on partitions of the decoder tree and only one path candidate is transferred from one partition to the next. As a result, memory can be shared between the different partitions of the code, therefore, significantly reducing the overall memory requirements. It is shown that the memory requirement of the PSCL decoder can be significantly reduced as the number of partitions increases. However, as the number of partitions increases, the error-correction performance of PSCL decoding degrades in comparison with SCL decoding at the same list size. Therefore, in this chapter, we present several improvements on the original PSCL scheme. These improvements are practical, in the sense that they boost the error-correction performance of the code.

We propose a modified code construction which leads to a remarkable gain in the finite-length performance. The idea is to design the polar code for a "better" channel, namely for a channel with a larger SNR compared to the channel that is used for transmission. Furthermore, the performance improvement comes at no additional cost in terms of memory, latency, or operational complexity, since we are simply constructing a polar code for a better channel and do not alter any of the encoding or decoding procedures.

In order to bridge the performance gap between SCL and PSCL, we first present a GPSCL algorithm that allows more than one candidate to be passed between different partitions. Then, we

observe that the SCL decoder has a tree structure in which a specific number of candidate codewords are present at each level of the tree, and we propose aLPSCL algorithm that passes different numbers of candidates for different levels. It should be noted that if we pass similar number of candidates at each level of the tree, then LPSCL reduces to GPSCL. Therefore, LPSCL is the generalization of GPSCL in which we are able to fully tune the trade-off between the error-correction performance and the decoder complexity. Furthermore, we implement the various decoders on hardware with TSMC 65 nm CMOS technology. We show that, by using the proposed algorithms, the area occupation of the original SCL decoder can be significantly reduced, while keeping the same error-correction performance.

We further propose a Successive CRC Assignment (SCA) strategy to select the number of CRC bits. Our approach is based on the successive minimization of the error probability for each partition. As such, this selection strategy is optimal in the sense that it minimizes the total error probability of the PSCL decoding algorithm.

We finally present a lower bound on the size of the list that ensures optimal Maximum a Posteriori (MAP) performance. The proof holds for the special case of the Binary Erasure Channel (BEC), but numerical simulations suggest that the claim holds for the transmission over general channels. The bound exploits the fact that the information bits tend to cluster at the end of the successive decoding process.

5.1 Partitioned Successive-Cancellation List (PSCL) Decoding

PSCL decoding allows to significantly reduce the memory requirements associated with SCL decoders. The idea is to break the decoding tree into two parts, i.e. the top and the bottom of the tree: the latter is composed of P sub-trees (partitions), and each partition is decoded with a CRC-aided SCL decoder. Instead of using the CRC to find the correct codeword at the end of CRC-aided SCL decoding process, PSCL uses the CRC to find a single correct codeword for each partition. It then passes this candidate to the top of the tree, where standard SC decoding is performed, and until the next partition is encountered. Therefore, it is not necessary to store L full trees as in SCL, but only L copies of the part of the tree contained in the partitions. Moreover, memory can be shared among the P partitions, which results in significant savings as P increases. Figures 5.1 and 5.2 show the PSCL decoding tree when $P = 2$ and $P = 4$, respectively. As described in [5], a polar code of length N can be seen as the concatenation of two polar codes of length $N/2$: thus, each partition is

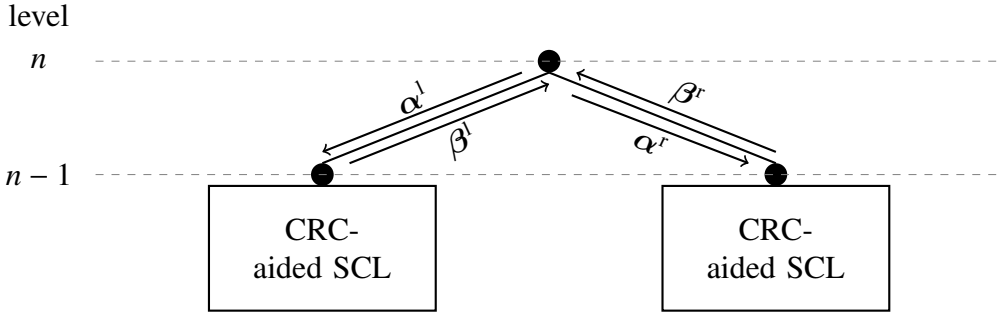


Figure 5.1: PSCL tree structure for $P = 2$.

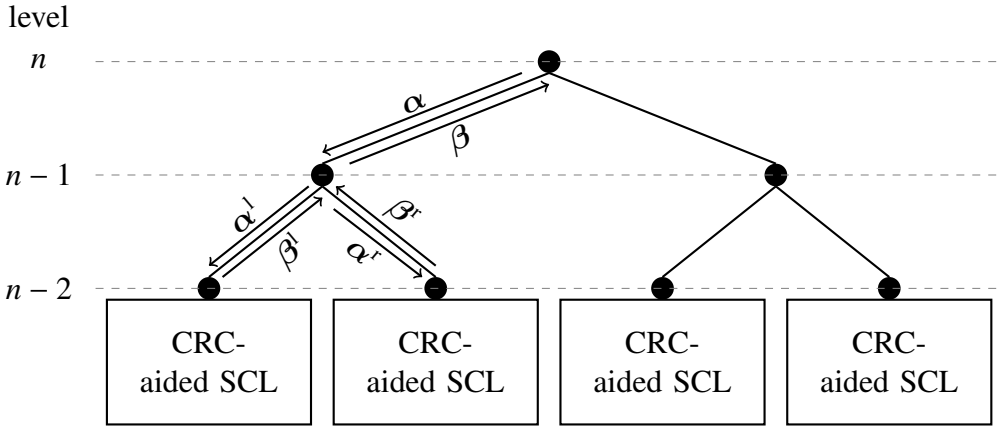


Figure 5.2: PSCL tree structure for $P = 4$.

a polar code of length N/P.

The memory requirement of PSCL is bound between those of SC ($P = N$) and SCL ($P = 1$) decoders as

$$M_{\text{PSCL}} = NQ_{\alpha_C} + \left(\sum_{k=1}^{\log_2 P} \frac{N}{2^k} + L\left(\frac{N}{P} - 1\right)\right) Q_{\alpha_I} + LQ_{\text{PM}} + \sum_{k=1}^{\log_2 P} \frac{N}{2^k} + L\left(\frac{2N}{P} - 1\right), \tag{5.1}$$

where $2 \leq P < N$. It is worth noting that as the number of partitions increases, the memory usage decreases exponentially toward the SC bound as depicted in Figure 5.3 for a code of length 1024. However, in the conventional PSCL algorithm, this memory saving is obtained at the cost of error-correction performance degradation as shown in Figure 5.4.

There are two main reasons associated with this performance degradation. First, PSCL uses SC decoding at the top of the polar code tree which can cause error-correction performance de-

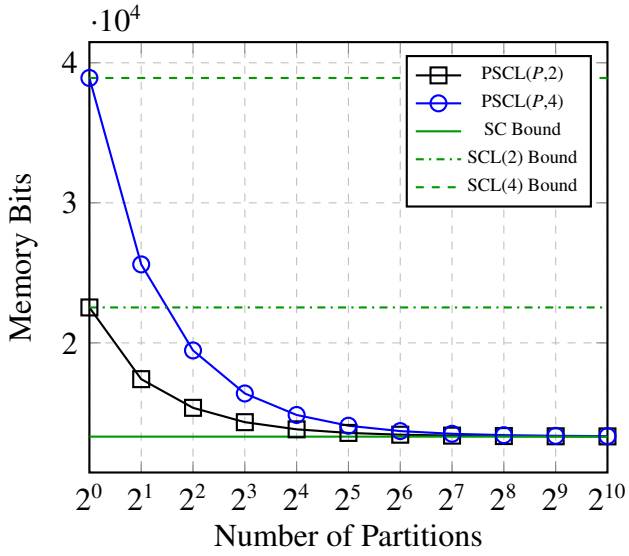


Figure 5.3: PSCL memory requirements for a polar code of length 1024 when $L = \{2, 4\}$. PSCL(L,P) represents PSCL decoding with list size L and with P partition, and SCL(L) represents SCL decoding with list size L.

gradation with respect to SCL. Second, the uniform distribution of CRCs between partitions in the conventional PSCL may result in significant deterioration in error-correction performance. We will propose methods to tackle these issues in the following sections.

5.1.1 Modified Code Construction

In this section, we describe a modified code construction that allows to obtain a gain in the error-correction performance at no additional complexity cost.

Denote by $W = \text{BAWGN}(\text{SNR}^*)$ the Binary AWGN (BAWGN) channel with SNR equal to SNR^* and let $\mathcal{P}_\kappa(N, K)$ be the polar code of length N and rate K/N designed for the transmission over $W_\kappa = \text{BAWGN}(\text{SNR}^*/\kappa)$. In words, when $\kappa = 1$, $\mathcal{P}_\kappa(N, K)$ is the polar code designed for W and, as κ goes from 1 to 0, $\mathcal{P}_\kappa(N, K)$ is a polar code designed for better and better channels.

Consider the transmission of the family of polar codes $\{\mathcal{P}_\kappa(N, K) : \kappa \in [0, 1]\}$ over the channel W. Note that the transmission channel is fixed, while the codes of the family are designed for different channels as κ varies. Empirically, one observes that the error probability under MAP decoding decreases as κ goes from 1 to 0. However, the error probability under SC decoding increases as κ goes from 1 to 0. The latter is due to the fact that the frozen positions of a polar code

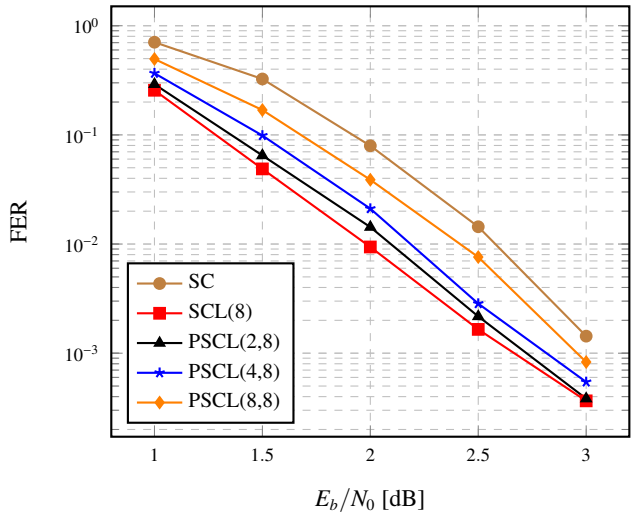


Figure 5.4: FER comparison between SC, SCL(8), and PSCL(P, 8) for $P \in \{2, 4, 8\}$ when no CRC is used. The polar code is $\mathcal{P}(1024, 512)$ and it is optimized for SNR = 2 dB.

are chosen in order to minimize its error probability under SC decoding. Starting from these two observations, in [52] a trade-off between complexity and performance is developed by considering low-complexity decoders (e.g., SCL). The trade-off comes from the fact that, as the decoder becomes more complex (e.g., the list size increases), the best performance are achieved for smaller values of κ.

This principle can also be applied to PSCL decoding, as shown in the numerical simulations of Figs. 5.5–5.7. In particular, different curves of Figure 5.5 refer to different codes (i.e., codes designed for different SNRs). The x-axis corresponds to SNR of the transmission channel, and y-axis labels the correspondent FER. Note that, at $E_b/N_0 = 3$ dB, a polar code constructed for SNR = 5 dB significantly outperforms the polar code optimized for SNR = 3 dB (which is the SNR of the channel over which the transmission takes place).

Figure 5.6 plots the error-correction performance of the code as a function of the design SNR. The transmission channel is fixed and it has $E_b/N_0 = 3$ dB. Different curves correspond to different decoding algorithms. For SC decoding, the best error-correction performance is achieved for a code constructed for SNR = 3 dB. Again, this is to be expected, since the polar code is constructed in order to minimize the error probability under SC decoding. For SCL(8), the best error-correction performance is achieved when the code is constructed for SNR = 5 dB. For PSCL(2,8), the optimal design SNR is 4.5 dB, and for PSCL(4,8), it is 4 dB. Note that, as the number of partitions increa-

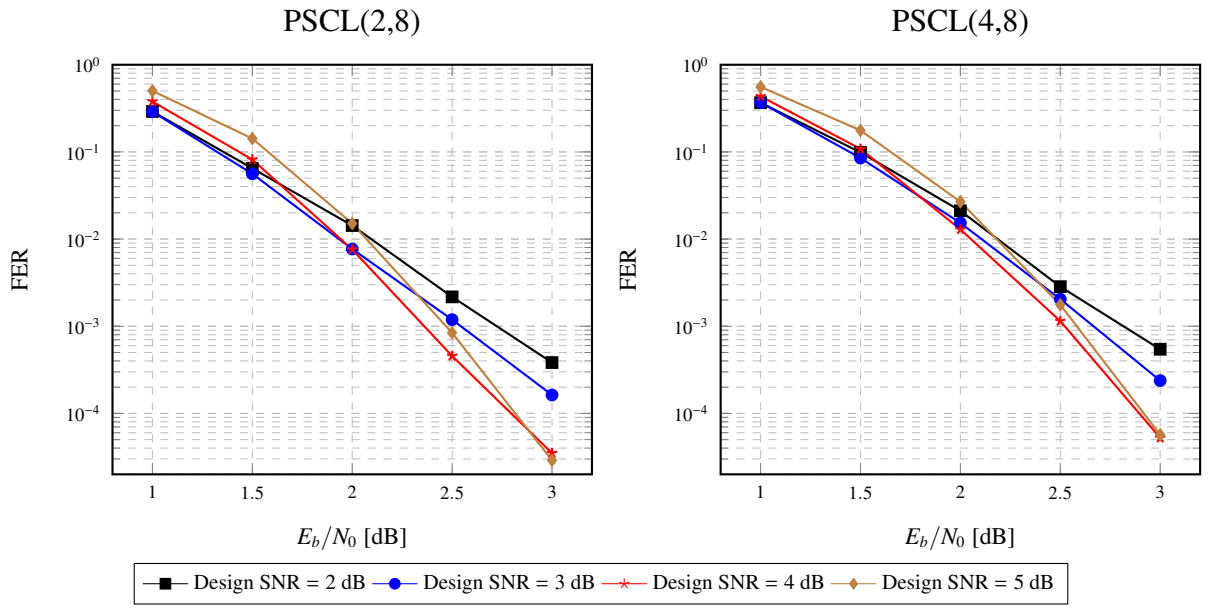


Figure 5.5: FER for the transmission of $\mathcal{P}(1024, 512)$ under PSCL decoding with $L = 8$ and $P \in \{2, 4\}$. Different curves correspond to different design SNR values and no CRC is used.

ses, the optimal design SNR for PSCL decoding moves from that for SCL decoding to that for SC decoding. This is due to the fact that, as the number of partitions increases, the error-correction performance of PSCL moves from that of SCL to that of SC (see also Figure 5.4).

Fig. 5.7 summarizes the gains in the FER performance guaranteed by the modified code construction. It also compares the FER performance of polar codes with that of an LDPC code of length 1152 and rate $1/2$ which is used in the WiMAX standard. The dashed curves refer to polar codes in which the design SNR is equal to the SNR of the transmission channel, i.e., polar codes constructed in the standard way. The continuous curves refer to polar codes in which the design SNR is chosen in order to minimize the FER. For example, for the PSCL(2,8) curve, when the channel SNR is 2 dB the design SNR is 3.5 dB, and when the channel SNR is 3 dB the design SNR is 4.5 dB. Both PSCL(2,8) and PSCL(4,8) exhibit gains of almost 0.5 dB at a target FER of 10^{-3}. Note that these gains come "for free", since changing the design SNR does not affect the computational complexity, the latency or the memory requirement.

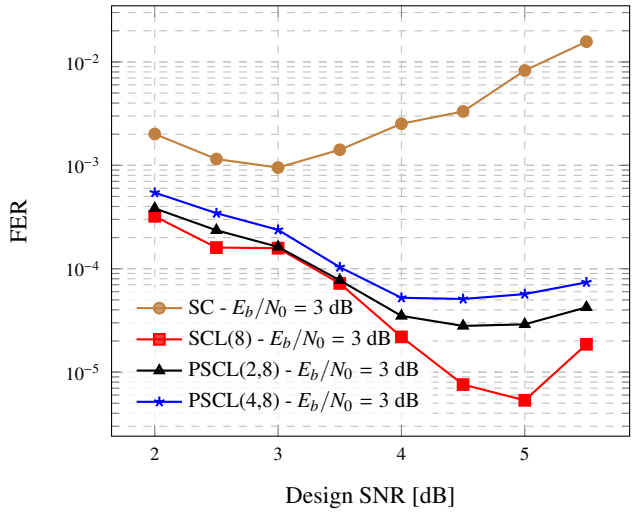


Figure 5.6: FER for PSCL(2,8), PSCL(4,8), SCL(8), and SC decoding of $\mathcal{P}(1024, 512)$ as a function of the design SNR. Different curves correspond to different decoding algorithms and no CRC is used. Note that the transmission takes place over a BAWGN with SNR = 3 dB.

5.1.2 Bridging the Performance Gap between PSCL and SCL

Generalized Partitioned Successive-Cancellation List (GPSCL)

The fundamental performance gap between PSCL and SCL decoding is the result of passing a single candidate codeword between partitions. To address this issue, we propose a GPSCL decoding algorithm which can improve the error-correction performance of PSCL to the extent that it is comparable with SCL, while requiring a lower decoder complexity. To this end, we allow for S candidates to be passed between the partitions, where $S \geq 1$. This is illustrated in Fig. 5.8 in which $P = 4$ and two candidate codewords are allowed to pass between the four partitions ($S = 2$).

Let us denote by GPSCL(P,L,S) the GPSCL decoder with P partitions and list size L, where S candidate codewords are allowed to pass between the partitions. It should be noted that GPSCL(P,L,1) is equivalent to PSCL(P,L), and GPSCL(P,L,L) is equivalent to SCL(L). Fig. 5.9 shows the effect of GPSCL decoding of $\mathcal{P}(1024, 512)$ with design SNR = 5 dB and $L = 8$. Clearly, as the number of partitions increases, we need to pass more candidates between the partitions to keep the FER performance of GPSCL close to that of SCL. Furthermore, in order to achieve the performance of the SCL decoder, it suffices to have $S = 2$ when $P \in \{2, 4\}$, and $S = 4$ when $P = 8$.

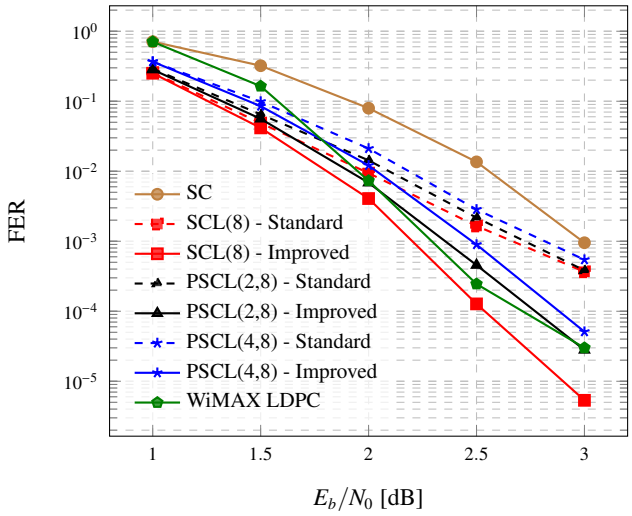


Figure 5.7: FER performance comparison between the standard and the modified construction of the code $\mathcal{P}(1024, 512)$. Different curves correspond to different decoding algorithms and no CRC is used. Note that gains of 0.5 dB are obtained at a target FER of 10^{-3} for PSCL(2,8) and PSCL(4,8). The FER performance of the WiMAX LDPC code of length 1152 and rate $1/2$ is also plotted for comparison.

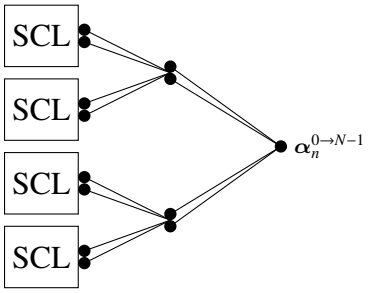


Figure 5.8: GPSCL tree structure for $P = 4$ and when two candidate codewords are allowed to pass between the partitions ($S = 2$).

For $S < L$[1], the memory requirement of the GPSCL decoder is less than that of the SCL decoder. The number of memory bits for GPSCL decoding can be calculated as

$$M_{\text{GPSCL}} = NQ_{\alpha_C} + \left(S \sum_{k=1}^{\log_2 P} \frac{N}{2^k} + L\left(\frac{N}{P} - 1\right) \right) Q_{\alpha_I} + L \cdot Q_{\text{PM}} + S \sum_{k=1}^{\log_2 P} \frac{N}{2^k} + L\left(\frac{2N}{P} - 1\right), \tag{5.2}$$

[1]Recall that, when $S = L$, GPSCL reduces to SCL.

where $2 \leq P < N$. It should be noted that S copies of the top $\log_2 P$ levels of the tree have to be stored, whereas for the bottom part, L copies are needed. The savings in memory bits guaranteed by the GPSCL decoder are summarized in Fig. 5.10 for $\mathcal{P}(1024, 512)$ with $L = 8$, $P \in \{2, 4, 8\}$, $Q_\alpha = 6$ and $Q_{\text{PM}} = 8$. Note that the memory requirement of GPSCL(P,8,S) changes from that of PSCL(P,8) to SCL(8) as S increases.

Layered Partitioned Successive-Cancellation List (LPSCL)

The error-correction performance of GPSCL(P,L,S) can be tuned by changing P and S. For GPSCL decoding with P partitions, S candidates are passed to the top $\log_2 P$ levels of the decoder tree, while L candidates are passed to the remaining bottoms levels. Let us consider again the example in Fig. 5.9. For $P \in \{2, 4\}$, it suffices to have $S = 2$ in order to match the performance of SCL(8), while for $P = 8$, we need $S = 4$. This means that we need to pass two candidates for levels $t = \{1, 2\}$ and four candidates for $t = 3$. Motivated by this observation, we propose a LPSCL decoder that at each level of the decoding tree passes a specific number of candidate codewords.

Denote by $\mathbf{s} = \{s_{n-1}, s_{n-2}, \ldots, s_{n-\log_2 P}\}$ the set of candidates present at each level of the tree, where s_t is the number of candidates at level t.[2] Let us also denote by LPSCL(P,L,$\mathbf{s}$), the LPSCL decoder with list size L, P partitions, and set of candidates $\mathbf{s}$. Fig. 5.11 shows the LPSCL decoding tree when $P = 4$ and $\mathbf{s} = \{2, 4\}$. It should be noted that LPSCL(P,L,$\{S, S, \ldots, S\}$) is equivalent to GPSCL(P,L,S), LPSCL(P,L,$\{1, 1, \ldots, 1\}$) is equivalent to PSCL(P,L), and LPSCL(P,L,$\{L, L, \ldots, L\}$) is equivalent to SCL(L).

Fig. 5.12 shows the FER performance of LPSCL decoding when the design SNR = 5 dB and $L = 8$ (same conditions as in Fig. 5.9) for different values of $\mathbf{s}$. Since different number of candidate codewords are stored at each layer of the decoding tree, a trade-off can be achieved between the memory requirements and the FER performance. Furthermore, it suffices to have $\mathbf{s} = \{2, 2, 4\}$, in order to achieve roughly the same FER performance as SCL(8).

The number of memory bits for LPSCL decoding can be calculated as

$$M_{\text{LPSCL}} = NQ_{\alpha_C} + \left(\sum_{k=1}^{\log_2 P} s_{n-k} \frac{N}{2^k} + L\left(\frac{N}{P} - 1\right) \right) Q_{\alpha_l} + L \cdot Q_{\text{PM}} + \sum_{k=1}^{\log_2 P} s_{n-k} \frac{N}{2^k} + L\left(\frac{2N}{P} - 1\right), \quad (5.3)$$

where $2 \leq P < N$.

[2]Clearly, only one copy of the channel LLR values needs to be stored for $t = n$.

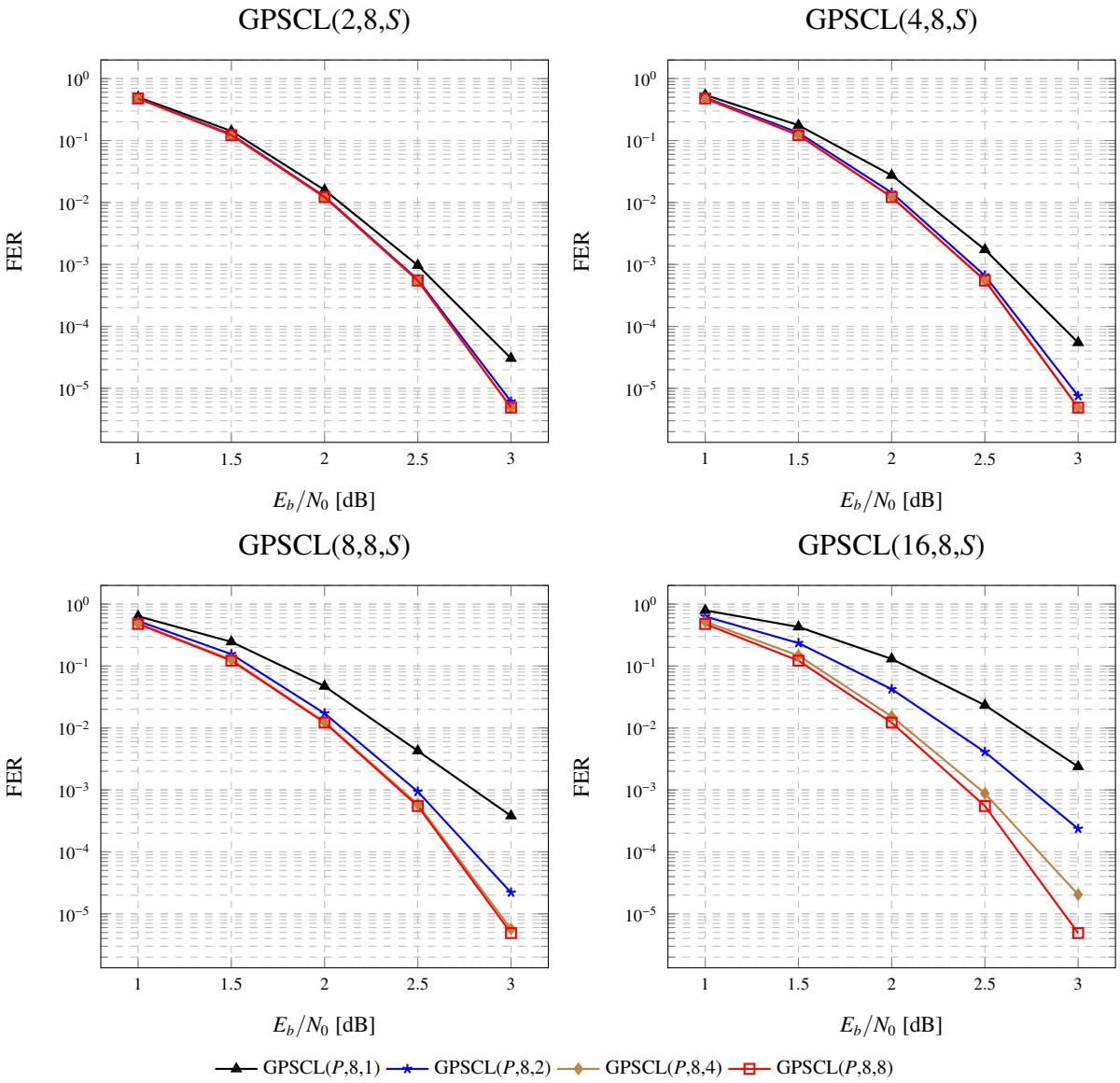


Figure 5.9: GPSCL decoding of $\mathcal{P}(1024, 512)$ with $L = 8$, $P \in \{2, 4, 8, 16\}$, and $S \in \{1, 2, 4, 8\}$, when design SNR = 5 dB. Note that as P increases, more candidates need to be passed between the partitions to maintain the FER performance close to SCL(8).

5.1.3 CRC Selection Scheme

In this section, we discuss the use of CRC bits and we present a strategy to choose the length of the CRC that minimizes the FER.

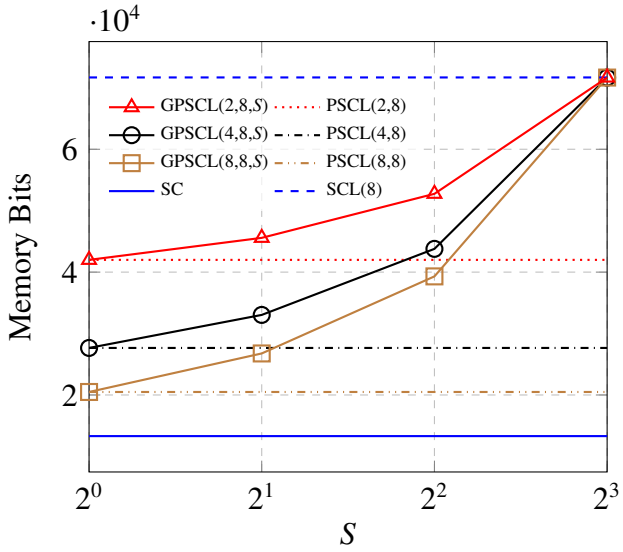


Figure 5.10: Memory bits required by GPSCL as a function of the number of passed candidates S with $L = 8$ and $P \in \{2, 4, 8\}$. Note that as S increases, the memory requirement of GPSCL changes from that of PSCL to that of SCL.

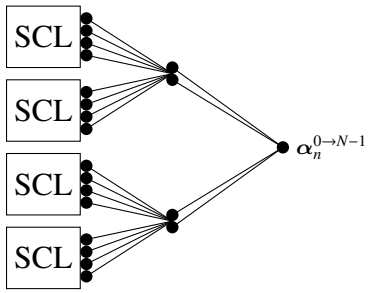


Figure 5.11: LPSCL tree structure for $P = 4$ and $\mathbf{s} = \{2, 4\}$.

The error-correction performance under SCL decoding is lower bounded by that under MAP decoding. However, in scenarios of interest in practical applications, even the performance of the MAP decoder is not satisfactory compared to state-of-the-art coding schemes, such as LDPC codes. In order to address this issue, it was shown in [12] that, by adding a CRC, the error-correction performance under SCL decoding significantly outperforms that under MAP decoding with no CRC, and it is comparable to that of state-of-the-art LDPC codes.

Denote by C the length of the CRC. In order to send K bits of information for N channel uses, we need to use a code of block length N and rate $(K + C)/N$. More specifically, let us focus on the PSCL decoder with P partitions. Consider the vectors $\mathbf{c} = \{c_0, c_1, \ldots, c_{P-1}\}$ and

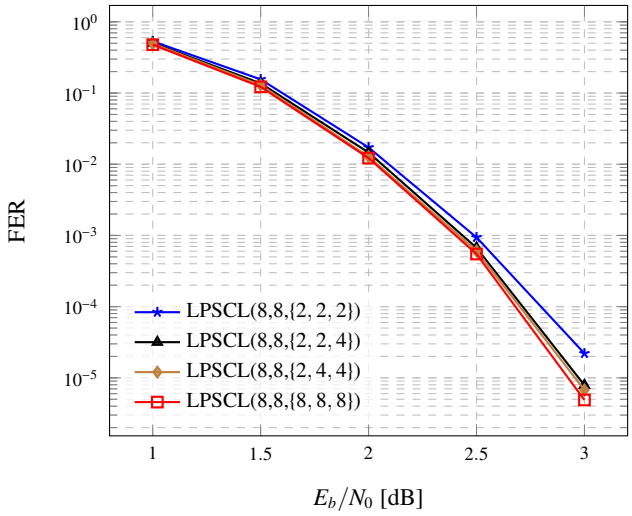


Figure 5.12: LPSCL decoding of $\mathcal{P}(1024, 512)$ with $L = 8$ and $P = 8$, when design SNR = 5 dB. Note that selecting $\mathbf{s} = \{2, 2, 4\}$ provides almost the same FER performance as SCL(8).

$\mathbf{k} = \{k_0, k_1, \ldots, k_{P-1}\}$, where c_p is the length of the CRC concatenated to the p-th partition and k_p represents the number of information bits associated to the p-th partition. Clearly,

$$\sum_{p=0}^{P-1} c_p = C, \qquad \sum_{p=0}^{P-1} k_p = K. \tag{5.4}$$

Then, in the p-th partition, we select the most reliable $k_p + c_p$ bits and use the first k_p to store the information bits and the last c_p to store the CRC bits.

On the one hand, if c_p is too small, then some of the incorrect paths might pass the CRC and one of these incorrect paths might be transmitted to the next partition, thus causing an error. On the other hand, if c_p is too large, then the positions chosen to store the CRC might not be sufficiently reliable. Hence, there is a trade-off in the choice of the vector $\mathbf{c}$.

We propose a *SCA* strategy that can be described as follows. For $p \in \{0, \ldots, P-1\}$, we evaluate numerically the FER of the p-th partition as a function of c_p, assuming that the previous partitions were decoded correctly. Then, we choose the value of c_p that minimizes the FER. In this way, the FER of the PSCL algorithm is minimized.

Figure 5.13 considers the transmission of the polar code $\mathcal{P}(1024, 512)$ over a channel with SNR = 2 dB and represent the error-correction performance of the P partitions of the PSCL decoder as a function of the length of the CRC for $P = 2$ and $P = 4$. The design SNR of the

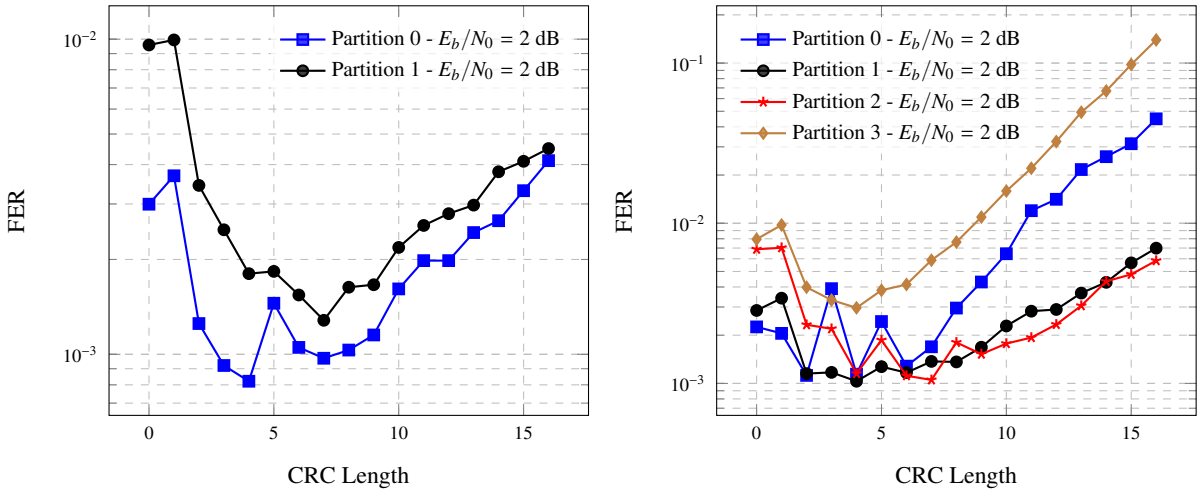


Figure 5.13: Effect of the CRC length on the FER for PSCL(2,8) decoding (left) and PSCL(4,8) decoding (right) of $\mathcal{P}(1024, 512)$. Different curves correspond to different partitions. The SNR of the transmission channel and the design SNR of the code are equal to 2 dB.

code is equal to the SNR of the transmission channel. For PSCL(2,8), partition 0 and partition 1 achieve the best error-correction performance with a CRC of length 4 and 7 respectively. Therefore, we conclude that PSCL(2,8) has optimal performance when a CRC of length 4 is concatenated to the first partition and a CRC of length 7 is concatenated to the second. Similarly, for PSCL(4,8), partitions 0, 1, 2, and 3 achieve the best error-correction performance with CRCs of length 2, 4, 7, and 4, respectively. Therefore, PSCL(4,8) has optimal performance with $\mathbf{c} = \{2, 4, 7, 4\}$.

Figure 5.14 summarizes the gains in the error-correction performance guaranteed by the SCA strategy. In particular, we consider the SCL(8), PSCL(2,8), and PSCL(4,8) decoders and we compare the SCA scheme with the CRC lengths chosen in [48]. All the algorithms exhibit gains of about $1/4$ dB at the target FER of 10^{-3}.

Although effective, there are two main issues associated with the above SCA approach. The first issue is that the identification of the optimal CRC length for each partition is performed by finding the error-correction performance of each partition in a serial manner which requires the knowledge of the correct codeword for previously decoded partitions. The second issue stems from the fact that SCA finds the optimal CRC lengths of partitions without having any constraint on them. This is in contrast with the conventional PSCL approach, in which in order to keep the effective rate of polar codes constant, the sum of CRC lengths for partitions is kept at C.

In order to tackle the above issues, we use Gaussian Approximation (GA) to find the error-

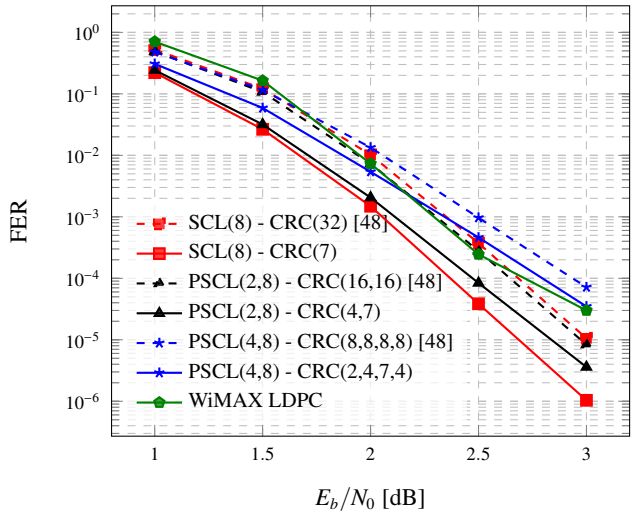


Figure 5.14: FER performance comparison for the transmission of $\mathcal{P}(1024, 512)$ between the proposed SCA scheme (solid) and the CRC lengths chosen in [48] (dashed). Different curves correspond to different decoding algorithms. In the SCA scheme, the lengths of the CRCs are optimized separately for each value of the SNR of the transmission channel. Note that gains of about $1/4$ dB are obtained at a target FER of 10^{-3} for all the decoding algorithms. The FER performance of the WiMAX LDPC code of length 1152 and rate $1/2$ is also plotted for comparison.

correction performance of each partition in parallel, without any recourse to the information from other partitions. After finding the error-correction performance of each partition for a given E_b/N_0 and CRC length, we impose a constraint on the sum of CRC lengths of partitions to find the optimal value of CRC length for each partition. We further show that this constraint can be modified to only select the CRC lengths from a set of practical ones.

GA was first used to reduce the complexity of polar code construction for AWGN channel in [53]. In this setting, the channel LLR values, α_i^n, have a normal distribution $\mathcal{N}(\frac{2}{\sigma^2}, \frac{4}{\sigma^2})$, where σ represents the standard deviation of the AWGN channel, and the transmission of the all-zero codeword is considered. The intermediate LLR values at stage t of the SC decoding tree can be approximated as having a normal distribution $\mathcal{N}(\mu_i^t, 2\mu_i^t)$. The value of μ_i^t can be calculated recursively as

$$\mu_i^t = \phi^{-1}\left(1 - \left(1 - \phi\left(\mu_i^{t+1}\right)\right)^2\right), \tag{5.5}$$

$$\mu_{i+2^t}^t = 2\mu_{i+2^t}^{t+1}, \tag{5.6}$$

where

$$\phi(\mu) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi\mu}} \int_{-\infty}^{\infty} \tanh \frac{u}{2} \mathrm{e}^{-\frac{(u-\mu)^2}{4\mu}} \, du, & \mu > 0, \\ 1, & \mu = 0. \end{cases} \tag{5.7}$$

It should be noted that in (5.5) and (5.6), $\mu_i^{t+1} = \mu_{i+2^t}^{t+1}$. GA determines the channel seen by each partition, provided that all the sub-channels are approximated as AWGN channels. We can therefore obtain the error-correction performance of each partition independently. GA approximates the intermediate LLR values calculated by SC decoding by assuming that the previous bits are decoded correctly. Therefore, the FER of polar codes under PSCL decoding can be calculated as the sum of the FER performance of the partitions simulated with GA-obtained AWGN channels.

The GA-obtained channels allow to find the optimal CRC length for each partition. Let us consider $\mathbf{c} = \{c_0, c_1, \ldots, c_{P-1}\}$, representing the set of CRC lengths for the PSCL partitions. In the conventional PSCL algorithm, the CRC length for partition p is selected as

$$c_p = \frac{C}{P} \text{ bits.} \tag{5.8}$$

While simple, this method can introduce significant error-correction performance loss as the number of partitions increases [48]. Let us consider an AWGN channel with a certain E_b/N_0. We can calculate the standard deviation of this channel as

$$\sigma = \sqrt{\frac{10^{-\frac{E_b/N_0}{10}}}{2R}}. \tag{5.9}$$

Therefore,

$$\mu_i^n = 4R10^{\frac{E_b/N_0}{10}}. \tag{5.10}$$

We can now use (5.5) and (5.6) to determine the channels that are seen by the two partitions in stage $n-1$. The E_b/N_0 values of the channels seen by each partition can be calculated as

$$E_b/N_{0_0} = -10 \log_{10}\left(\frac{4R_0}{\mu_i^{n-1}}\right), \tag{5.11}$$

$$E_b/N_{0_1} = -10 \log_{10}\left(\frac{4R_1}{\mu_{i+N/2}^{n-1}}\right), \tag{5.12}$$

Table 5.1: Rate of polar codes seen in PSCL decoding of $\mathcal{P}(1024, 512)$ for $P = 2$ and $P = 4$.

P	Rate			
1	512/1024			
2	135/512		377/512	
4	19/256	116/256	141/256	236/256

Table 5.2: E_b/N_0 values of the channels seen in PSCL decoding of $\mathcal{P}(1024, 512)$ for $P = 2$ and $P = 4$.

P	E_b/N_0 [dB]			
1	3			
2	3.3217		4.3293	
4	5.2449	3.9805	4.1832	6.3636

where E_b/N_{0_p} represents the E_b/N_0 value of the channel seen by partition p, and R_p is the rate of the polar code in partition p. A recursive application of (5.9)-(5.12) results in the channels seen by partitions at a lower stage.

As an example, let us assume PSCL decoding of $\mathcal{P}(1024, 512)$ with an AWGN channel with $E_b/N_0 = 3$ dB, while the code is optimized for $E_b/N_0 = 2$ dB. Table 5.1 shows the rate of the polar codes in the various partitions, and Table 5.2 summarizes the E_b/N_0 values of the channels seen by the partitions.

It is now possible to find the FER of each partition based on the E_b/N_0 values of Table 5.2 while considering different values of CRC length for each partition. Figure 5.15 shows the effect of CRC length on the FER of partitions when the channel seen by $\mathcal{P}(1024, 512)$ has $E_b/N_0 = 3$ dB for $P = 2$ and $P = 4$. In this figure, SCL decoding with $L = 2$ was used to derive the FER of each partition. Figure 5.16 is plotted similar to Figure 5.15, but it uses SCL with $L = 4$ instead. It can be seen that there is a specific c_p which leads to an optimal error-correction performance for partition p for every L and when $E_b/N_0 = 3$ dB. However, we have the constraint that

$$\sum_{p=0}^{P-1} c_p = C. \tag{5.13}$$

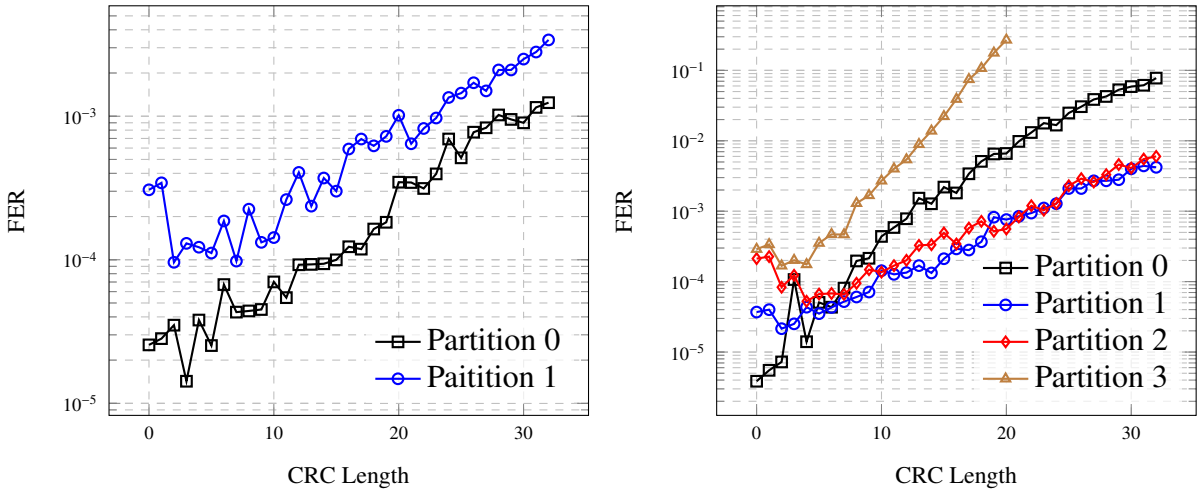


Figure 5.15: Effect of CRC length on the FER of each partition for $\mathcal{P}(1024, 512)$ when $P = 2$ (left) and $P = 4$ (right) with $L = 2$. The FER of the two partitions were derived independently using GA for $E_b/N_0 = 3$ dB.

Input: $L, P, C, E_b/N_0$
Output: **c**
Find E_b/N_0 of the P partitions using (5.9)-(5.12)
for $j \leftarrow 0$ ***to*** $P-1$ **do**
  for $c_j \leftarrow 0$ ***to*** C **do**
    Find $\text{FER}_j(c_j)$ for given L
  end
end
Solve (5.14)
Result: c_j for $0 \leq j < P$.

Algorithm 2: Determining CRC length for each partition.

Therefore, we have to solve the following optimization problem

$$\underset{\sum_{p=0}^{P-1} c_p = C}{\arg\min} \sum_{p=0}^{P-1} \text{FER}_p(c_p), \tag{5.14}$$

where $\text{FER}_p(c_p)$ is the FER of partition p when the CRC length is c_p. The process for selecting a good CRC length for partitions is summarized in Algorithm 2.

Algorithm 2 causes the CRC lengths of each partition to be any value between 0 and C. In practical applications, we are usually constrained to have a set of standard CRC lengths. Let us

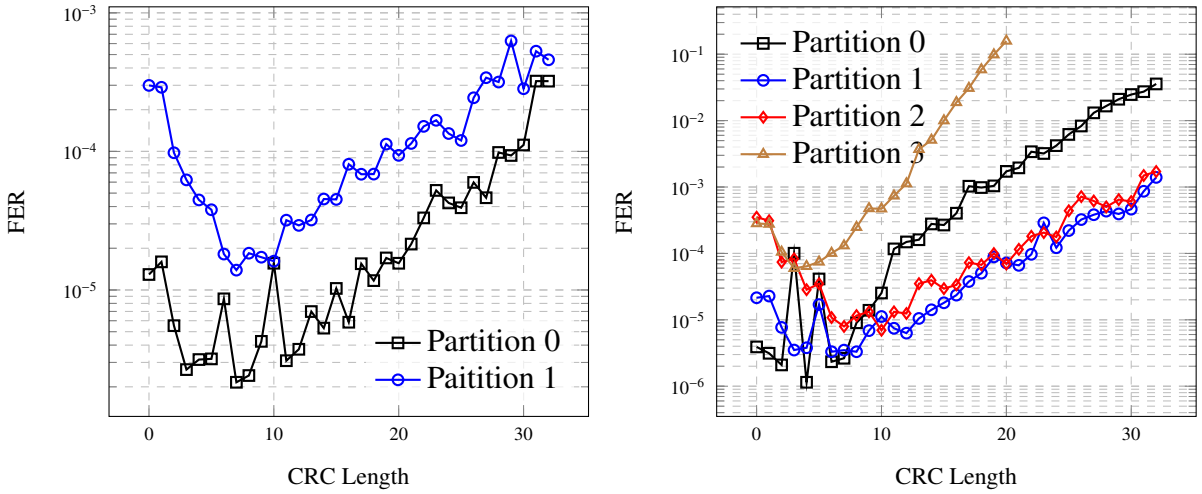


Figure 5.16: Effect of CRC length on the FER of each partition for $\mathcal{P}(1024, 512)$ when $P = 2$ (left) and $P = 4$ (right) with $L = 4$. The FER of the two partitions were derived independently using GA for $E_b/N_0 = 3$ dB.

consider we are constrained to have CRC lengths which are multiples of four. Therefore, (5.14) can be rewritten as

$$\underset{\substack{\sum_{p=0}^{P-1} c_p = C \\ \text{mod}(c_p,4)=0}}{\arg\min} \sum_{p=0}^{P-1} \text{FER}_p(c_p), \tag{5.15}$$

where mod is the modulus operation. In this chapter, we solve (5.14) and (5.15) with an exhaustive search on the sum of FER values of partitions for different CRC lengths.

Figure 5.17 and Figure 5.18 show the FER and BER curves for CRC-aided SCL and PSCL with $L = 2, 4$ and $P = 2, 4$. The CRC-aided SCL decoders are labelled as SCL(L)-CRC(C), which represents CRC-aided SCL decoding with list size L and using a CRC of size C. The PSCL decoders are labelled as PSCL(P,L)-CRC(c_0,c_1,. . .,c_{P-1}), where c_p is the length of the CRC assigned to partition p. They compare the error-correction performance obtained with CRC lengths calculation of Algorithm 2, the additional constraint of (5.15), and the conventional CRC lengths calculation of (5.8). It should be noted that for $L = 2$, Algorithm 2 results in PSCL(2,2)-CRC(19,13) and PSCL(4,2)-CRC(6,14,8,4), while adding the constraint in (5.15) results in PSCL(2,2)-CRC(16,16) and PSCL(4,2)-CRC(4,12,12,4). For $L = 4$, Algorithm 2 results in PSCL(2,4)-CRC(20,12) and PSCL(4,4)-CRC(7,12,10,3), while adding the constraint in (5.15) results in PSCL(2,4)-CRC(20,12) and PSCL(4,4)-CRC(4,12,12,4). It is possible to see that the error-correction performance loss due

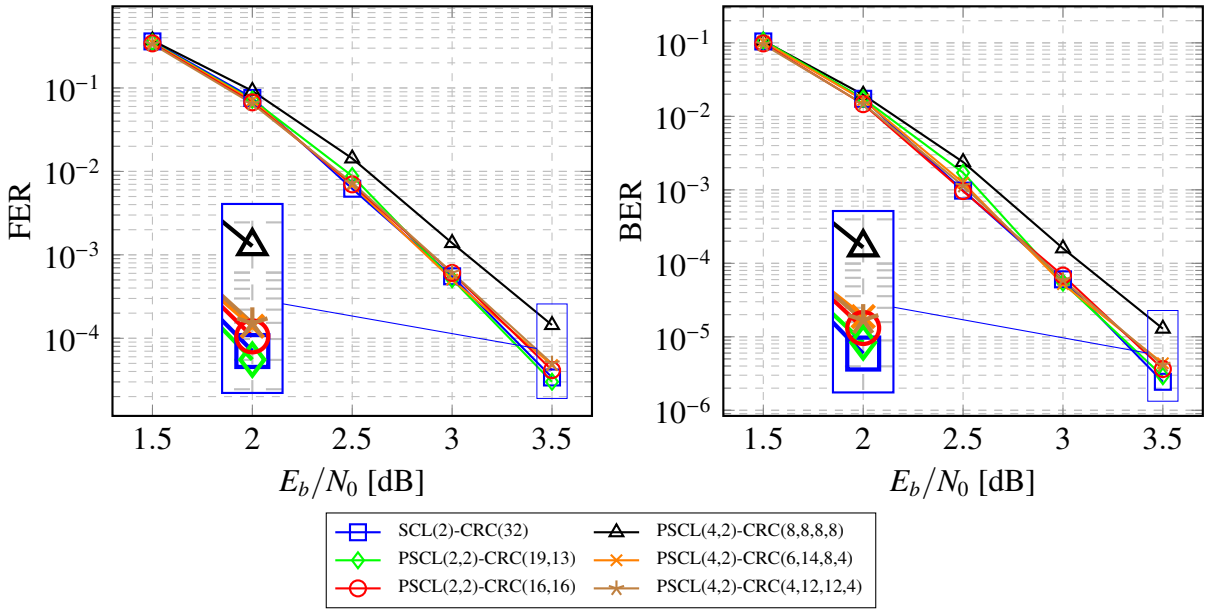


Figure 5.17: FER and BER performance for PSCL decoding of $\mathcal{P}(1024, 512)$ when $L = 2$. The code is optimized for $E_b/N_0 = 2$ dB.

to (5.15) is negligible for both $L = 2$ and $L = 4$. While these curves consider a code constructed for $E_b/N_0 = 2$ dB, similar results have been observed for any E_b/N_0 value.

5.2 LLR-β Memory Sharing

This section presents a memory reduction technique that can be applied to decoders implementing an SC-based algorithm, like SC, Fast-SSC [11], and SCL (CRC-aided SCL). It does not imply any particular decoder hardware structure, since its basic idea is derived from the order with which calculations need to be performed according to SC. The SC decoding process follows a specific operation schedule. This scheduling allows for substantial memory reduction for a SC decoder if the memory is shared between the LLR memory and the β memory. Let us consider the example in Figure 2.3. The vector of LLR values $\{\alpha_0^2, \alpha_1^2, \alpha_2^2, \alpha_3^2\}$ is used to calculate the vectors $\{\alpha_0^1, \alpha_1^1\}$ and $\{\alpha_2^1, \alpha_3^1\}$. The vector of β values $\{\beta_0^2, \beta_1^2, \beta_2^2, \beta_3^2\}$ is only created after both vectors $\{\beta_0^1, \beta_1^1\}$ and $\{\beta_2^1, \beta_3^1\}$ are created. Since the vector $\{\beta_0^2, \beta_1^2, \beta_2^2, \beta_3^2\}$ is no longer needed, the vector $\{\beta_0^2, \beta_1^2, \beta_2^2, \beta_3^2\}$ can use the same memory allocated for $\{\alpha_0^2, \alpha_1^2, \alpha_2^2, \alpha_3^2\}$. This will result in memory sharing between LLR and β memories and subsequently results in memory saving. Since β

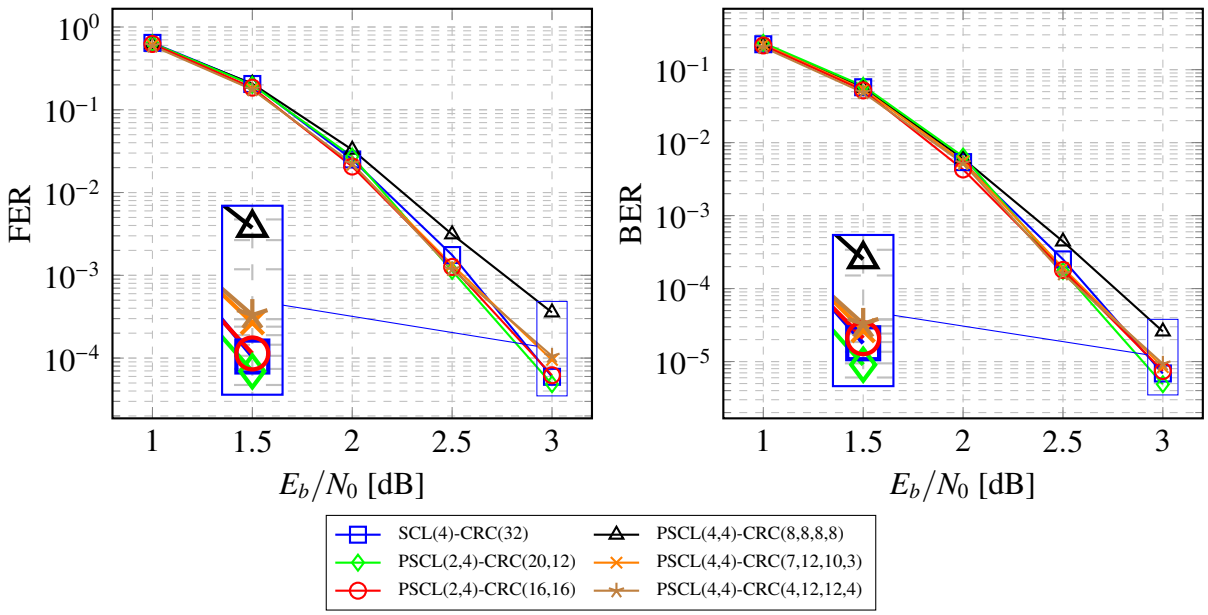


Figure 5.18: FER and BER performance for PSCL decoding of $\mathcal{P}(1024, 512)$ when $L = 4$. The code is optimized for $E_b/N_0 = 2$ dB.

values are represented with one bit and LLR values with Q_{α_l} bits, we can store the β values in the sign bit of the LLR values. The resulting memory requirement for the Memory-Efficient (ME) SC decoder is

$$M_{\mathrm{SC_{ME}}} = NQ_{\alpha_C} + (N-1)\,Q_{\alpha_l}, \tag{5.16}$$

which has $N-1$ fewer memory bits than (2.13).

SCL decoders follow the same schedule as SC decoders and thus the LLR and β memory sharing can be applied to them. Following the same reasoning for SC decoders, the memory requirement of the ME SCL decoder can be calculated as

$$M_{\mathrm{SCL_{ME}}} = NQ_{\alpha_C} + L\,(N-1)\,Q_{\alpha_l} + LQ_{\mathrm{PM}} + LN. \tag{5.17}$$

It can be seen that the LN memory bits required to store the final candidate codewords are present in the ME SCL decoder and ME SCL requires $(N-1)L$ fewer memory bits than a conventional SCL decoder.

The PSCL decoder uses the CRC-aided SCL decoder to decode the partitions and uses SC decoding rules to pass the candidate codewords from one partition to another. Therefore, the LLR-

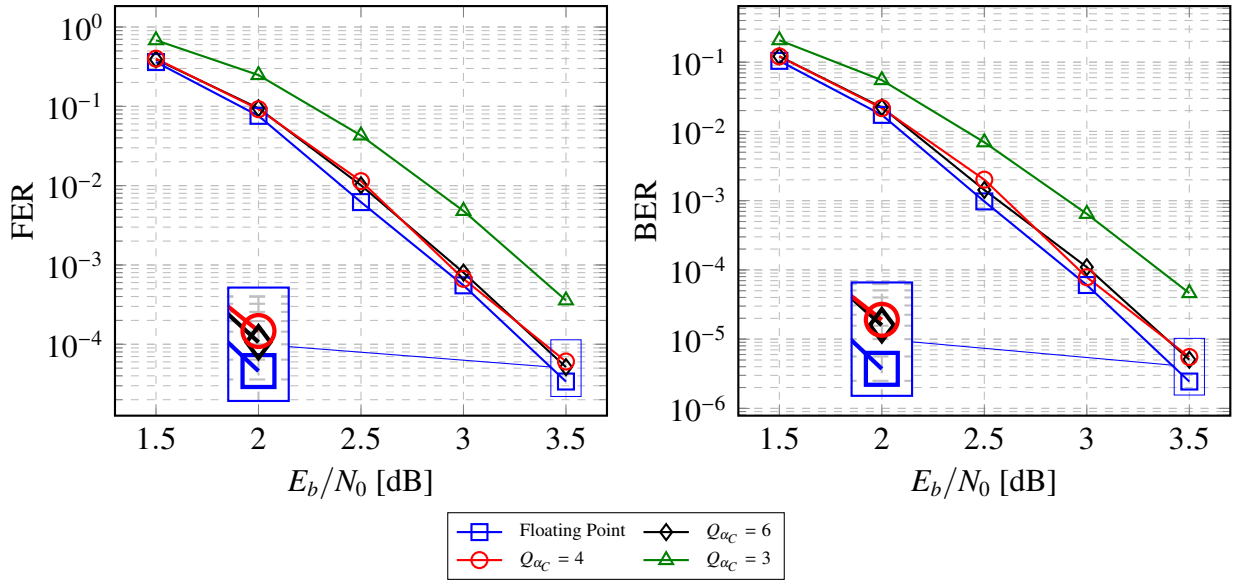


Figure 5.19: Effect of Q_{α_C} on FER and BER performance for CRC-aided SCL decoding of $\mathcal{P}(1024, 512)$ when $L = 2$, $Q_{\alpha_l} = 6$, and $Q_{\text{PM}} = 8$. The code is optimized for $E_b/N_0 = 2$ dB and the CRC length is 32.

β memory sharing can be applied as well, to the CRC-aided SCL and the SC decoders both. The resulting memory requirement for the ME PSCL decoder is

$$M_{\text{PSCL}_{\text{ME}}} = NQ_{\alpha_C} + \left(\sum_{k=1}^{\log_2 P} \frac{N}{2^k} + L\left(\frac{N}{P} - 1\right)\right) Q_{\alpha_l} + LQ_{\text{PM}} + L\frac{N}{P}. \tag{5.18}$$

The proposed technique can be applied to all SC-based decoding algorithms. No approximation is used, and it incurs no error-correction performance degradation.

5.3 Quantization Reduction for Channel LLR Values

The channel LLR memory requires the storage of N LLR values received from the channel. It was observed in [20] that quantizing channel LLR values with the same number of bits as the internal LLR values would not introduce significant error-correction performance loss in comparison with its floating point counterpart. For a code of length $N = 1024$, the channel LLR and the internal LLR values were quantized with 6 bits. The PM requires more bits than both channel and internal LLR values for minimal error-correction performance degradation due to quantization. Therefore,

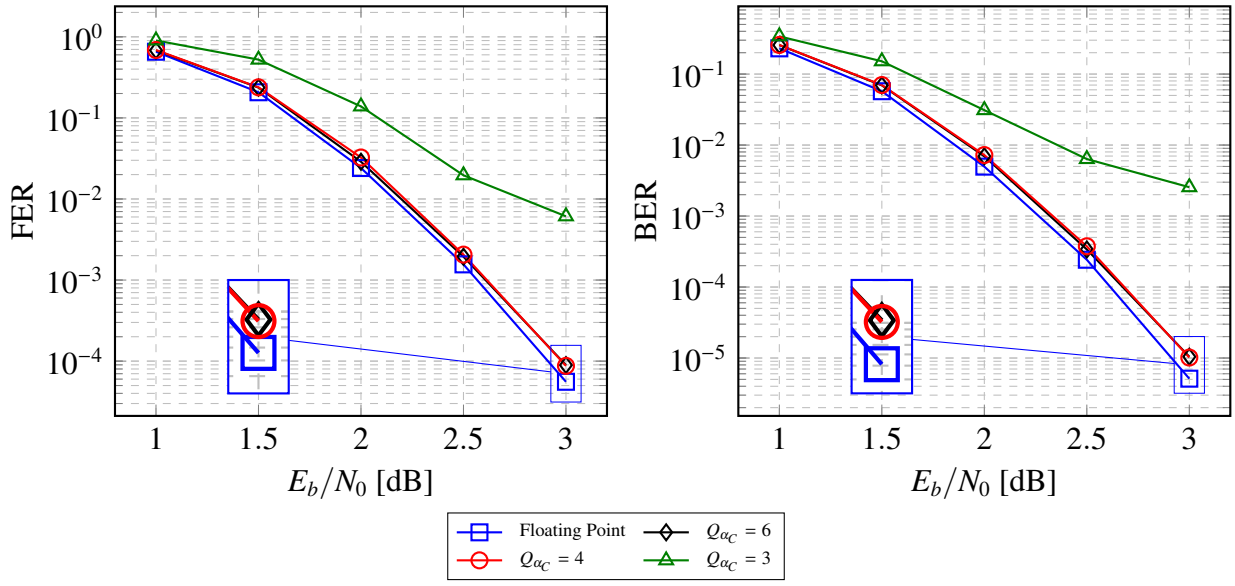


Figure 5.20: Effect of Q_{α_C} on FER and BER performance for CRC-aided SCL decoding of $\mathcal{P}(1024, 512)$ when $L = 4$, $Q_{\alpha_l} = 6$, and $Q_{\text{PM}} = 8$. The code is optimized for $E_b/N_0 = 2$ dB and the CRC length is 32.

8 bits were assigned to PM values. The same quantization scheme was adopted in the LLR-based SCL decoders of [37], [48]. In [17], [18], it was observed that the channel LLR values require fewer number of quantization bits and for a polar code of length $N = 1024$, the channel LLR values were quantized with 5 bits. As can be seen in Figure 5.19 and Figure 5.20, our simulations for $\mathcal{P}(1024, 512)$ show that setting $Q_{\alpha_C} = 4$ is adequate to keep the error-correction performance of CRC-aided SCL decoder close to the floating point version of channel LLR values for $L = 2$ and $L = 4$, for BER and FER values useful for wireless communications.

5.4 Hardware Implementation

CRC-aided SCL decoder

As a proof of concept, we considered as a starting point the CRC-aided SCL decoder architecture described in [20], sized for a polar code with $N = 1024$, and able to decode any code rate. We modified it to implement both the channel LLR quantization reduction and LLR-β memory sharing.

Implementation of the LLR quantization reduction is straightforward. The channel LLR me-

mory width is reduced to fit the new quantization: since LLR values are represented with sign and magnitude, the magnitude of values read from memory is zero-padded to fit the internal quantization Q_{α_I}.

The LLR-β memory sharing requires a few modifications to the hardware architecture in [20]: it does not change the scheduling of operations, that remain as shown in Figure 2.3. The decoder relies on P_e processing elements that can perform (2.11) and (2.8) operations in parallel. Each processing element receives as inputs two LLR and one β values. Thus, the β memory in standard CRC-aided SCL requires to be read with a parallelism of P_e values, and the LLR memory with a parallelism of $2P_e$ values. If the β memory is used to store LLR signs as well, then it must allow for additional $2P_e$ values to be read concurrently: since the β memory in [20] is composed of registers only, due to its irregular update pattern, this modification can be achieved with dedicated multiplexers, as shown in Figure 5.21. The update structure of the β memory already allows for multiple irregular updates: in the standard CRC-aided SCL decoder structure, whenever $t = 0$ and a bit is estimated, all the β values at all stages that are influenced by that bit are concurrently updated. The P_e concurrent LLR signs can thus be easily stored in the β memory by modifying the write enable generation logic and allowing for updates also when $t \neq 0$.

The additional memory addressing logic is instantiated in parallel to the standard one, and does not lie on the system critical path. Thus, the implementation of this technique does not decrease the achievable frequency, and since it does not add any step to the decoding process, the decoder latency and throughput remain unchanged. Moreover, control signals are already present in the standard SCL decoder architecture: thus, no additional logic is required to create them. The schedule described in Section 5.2 assures that newly computed β values overwrite obsolete LLR signs, and updated LLR signs overwrite β values that are no longer needed.

PSCL decoder

To implement PSCL, the CRC-aided SCL decoder described in the previous section have to undergo some architectural modification. Both LLR and β memories have to be reduced to fit the size of the partition on which CRC-aided SCL decoding is performed. A secondary partial memory is instantiated to fit the part of the decoding tree that is handled by SC decoding, along with the related addressing and enabling logic.

On top of the PSCL decoder, we also applied the architectural modifications necessary to implement the LLR-β memory sharing and channel LLR quantization reduction. These modifications

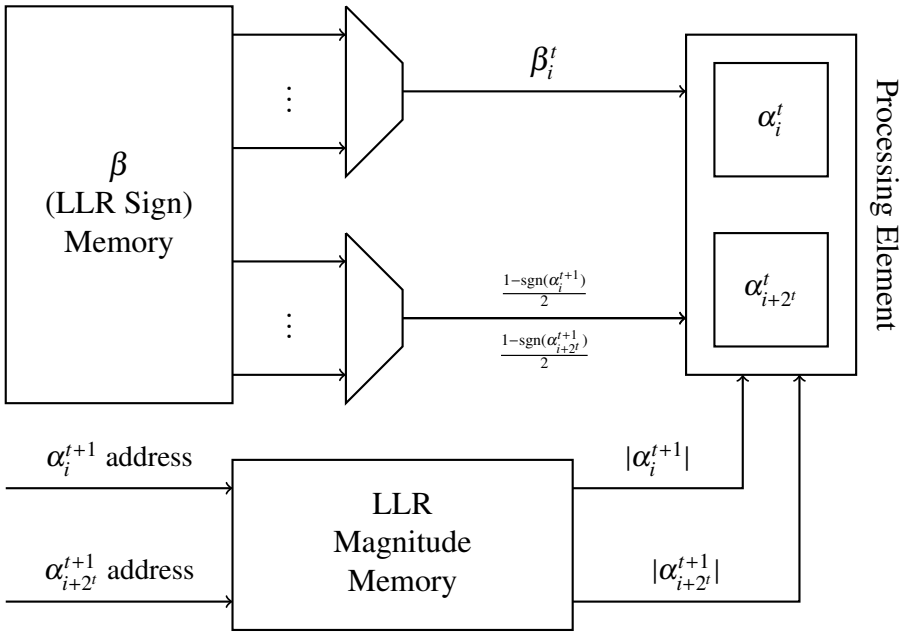


Figure 5.21: LLR-β memory sharing architecture.

are analogous to those for CRC-aided SCL. Applying the channel LLR quantization reduction remains straightforward, and the LLR sign and β addressing logic is made simpler by the smaller LLR and β memories.

GPSCL and LPSCL decoders

In order to design hardware architectures for the GPSCL and LPSCL decoders, the PSCL architecture that implements the original PSCL algorithm [48], has been taken as a starting point. This architecture relies on a set of L processing elements, and on L dedicated memories for LLR values, β values and estimated bits. For the GPSCL decoder, the parallelism of the partial LLR and β memories that allowed the SC algorithm to be applied at the upper stages of the decoding tree has been increased from 1 to S, along with the related addressing logic. The LPSCL decoder sees the same concept applied not only to a single additional layer, but to $\log_2 P$. The number of instantiated partial memories is equal to the number of different list sizes considered by the $\log_2 P$ upper stages.

We have added the two memory reduction techniques for SCL decoders along with PSCL: a quantization reduction for channel LLR values, and a memory sharing method that allows the β va-

Table 5.3: Synthesis area results with 65 nm TSMC CMOS technology for CRC-aided SCL and PSCL decoding of $\mathcal{P}(1024, 512)$. The target frequency is 800 MHz and $P_e = 64$.

Algorithm	No Memory Reduction		LLR-β Sharing		LLR-β Sharing + Channel LLR Quantization	
	Total [mm^2]	Memory [mm^2]	Total [mm^2]	Memory [mm^2]	Total [mm^2]	Memory [mm^2]
SCL(2)-CRC(32)	0.5756	0.2774	0.4470	0.1934	0.4243	0.1868
SCL(4)-CRC(32)	0.9888	0.4749	0.9002	0.3400	0.8768	0.3040
PSCL(2,2)-CRC(16,16)	0.4736	0.2097	0.3942	0.1844	0.3670	0.1785
PSCL(4,2)-CRC(8,8,8,8)	0.4348	0.1843	0.3695	0.1792	0.3400	0.1669
PSCL(4,2)-CRC(4,12,12,4)	0.4531	0.1866	0.3662	0.1793	0.3318	0.1569
PSCL(2,4)-CRC(20,12)	0.7548	0.3578	0.7411	0.3094	0.7273	0.2839
PSCL(2,4)-CRC(16,16)	0.7551	0.3524	0.7370	0.2999	0.7298	0.2842
PSCL(4,4)-CRC(8,8,8,8)	0.6998	0.2948	0.6786	0.2756	0.6503	0.2565
PSCL(4,4)-CRC(4,12,12,4)	0.7000	0.2947	0.6829	0.2741	0.6565	0.2581

lues to be stored in the same locations as the LLR sign bit. These two techniques are implemented in the SCL, GPSCL and LPSCL decoders as well.

5.4.1 Results

The CRC-aided SCL and PSCL architectures have been described in VHDL and synthesized in 65 nm TSMC CMOS technology. Table 5.3 reports the synthesis results for architectures sized for $\mathcal{P}(1024, 512)$, targeting a frequency of 800 MHz, with all memory elements implemented with registers and with $P_e = 64$. Two CRC-aided SCL architectures are considered, for $L = 2$ and $L = 4$, both with a CRC of length 32 bits. PSCL architectures are considered for $L = 2$ and $L = 4$ as well, taking in account different numbers of partitions P and different CRC lengths and distributions. All presented CRC-aided SCL and PSCL decoders rely on the same decoding flow and have been synthesized for the same target frequency of 800 MHz: they all yield a throughput of 301 Mb/s.

The first set of results (second and third column) details the total area occupation and the memory area occupation for the standard architectures, where no additional memory reduction technique is applied. Both channel and internal LLR values are quantized with 6 bits in all considered designs. We have designed and implemented our own decoders to be able to modify the architectures, implementing the memory reduction techniques. This allows us to correctly evaluate benefits and costs of the proposed methods. Nevertheless, the designed CRC-aided SCL decoders yield an area occupation very similar to that of state-of-the-art decoders when scaled to the same technology node. The work in [18] occupies 0.99 mm^2 for $L = 4$, while the area occupation of [17] is 1.03 mm^2 and 2.00 mm^2, respectively: our baseline CRC-aided SCL decoder has an area of 0.58 mm^2 for $L = 2$ and 0.99 mm^2 for $L = 4$. The SCL decoder in [15] occupies an area of

0.62 mm^2 for $L = 4$, and the SCL decoder described in [16] has an area of 0.73 mm^2 for $L = 4$. Most decoders in literature are single-code decoders built for high-throughput: on the other hand, our baseline CRC-aided SCL decoder is inspired to the high degree of flexibility of the decoder in [20], that can decode any code rate. The PSCL decoder implementations are direct modifications of the aforementioned CRC-aided SCL decoders. It is possible to see that PSCL decoders yield substantially lower total and memory area occupation with respect to the CRC-aided SCL decoders with the same list size L, with total area saving reaching 25% for $L = 2$ and 29% for $L = 4$. The memory reduction is more substantial as P increases.

The effects on the area occupation brought by the LLR-β sharing memory reduction technique detailed in Section 5.2 are reported in the fourth and fifth column of Table 5.3. Applied to the CRC-aided SCL decoder, it allows for 22% total area reduction when $L = 2$, and 9% when $L = 4$. The LLR-β memory sharing technique is able to reduce the PSCL decoder memory by 12% in case of PSCL(2,2) and by 22% for PSCL(4,2), accounting for a combined total area saving of 32% and 36% reduction with respect to CRC-aided SCL. The LLR-β area saving contribution for PSCL(2,4) and PSCL(4,4) accounts for around 3% of the total. Unlike PSCL, whose gain with respect to CRC-aided SCL increases with higher values of L, the impact of LLR-β memory sharing decreases as L increases. This is due to the total memory requirements of CRC-aided SCL decoders, that rise proportionally to the list size L.

The sixth and seventh columns of Table 5.3 report the area occupation for the different decoders, after the conjunct application of LLR-β memory sharing and reduction of channel LLR quantization. Their implementation results in 26% and 11% total area reduction for CRC-aided SCL with $L = 2$ and $L = 4$ respectively. The area reduction brought by these two techniques can reach 24% and 27% in case of PSCL(2,2) and PSCL(4,2), while it settles around 3% and 7% for PSCL(2,4) and PSCL(4,4). The combined contribution of PSCL, LLR-β memory sharing and channel LLR quantization reduction leads to 42% and 34% total area saving with respect to SCL(2)-CRC(32) and SCL(4)-CRC(32), with memory area reduction peaking at 46% for PSCL(4,4)-CRC(4,12,12,4).

The total area reduction brought by each of the proposed memory reduction techniques with respect to the CRC-aided SCL benchmark decoders is summarized in Table 5.4, for different values of P and L. The results we provided target ASIC implementations, where memory plays a major role not only in terms of area occupation, but also in power consumption and energy efficiency. In case of FPGA implementation, the proposed memory reduction techniques will lead to

Table 5.4: Total area reduction in comparison with CRC-aided SCL considering different memory reduction techniques.

Memory Reduction Technique	P	L	Total Area Reduction
PSCL	2	2	18%
	2	4	24%
	4	2	21%
	4	4	29%
PSCL + LLR-β Memory Sharing	2	2	32%
	2	4	25%
	4	2	36%
	4	4	31%
PSCL + LLR-β Memory Sharing + Channel LLR Quantization	2	2	36%
	2	4	26%
	4	2	42%
	4	4	34%

gains in resource utilization proportional to those shown in Table 5.4. However, both LUTs and registers contribute relatively less to the system power consumption than in the ASIC case. Thus, the proposed methods will lead to a lower energy efficiency improvement.

The SCL, GPSCL and LPSCL decoders have been described in VHDL and synthesized in TSMC 65 nm CMOS technology. Table 5.5 reports the area occupation and memory cost results for four configurations of the GPSCL decoder, two for the LPSCL decoder, and the baseline SCL decoder, obtained with a target frequency of 700 MHz. The decoders employ the following quantization scheme: 4 bits are assigned to channel LLR values, and 6 bits to the internal LLR values, two of which are assigned to the fractional part. Path metrics are represented with 8 bits. All memories in the decoder have been implemented with registers: thus, the reported number of memory bits include pipeline stages and temporary values storage as well. The number of processing elements is 64 for all the designs, and the code length is set to $N = 1024$. All the implemented decoders employ $L = 8$.

The presented results show that both GPSCL and LPSCL decoders guarantee a substantial area occupation reduction with respect to the standard SCL decoder. In particular, the area of GPSCL decoders ranges from 57.2% to 65.5% of that of SCL(8). LPSCL decoders allow for additional

Table 5.5: ASIC implementation results for TSMC 65 nm CMOS technology for a polar code of length $N = 1024$ and with target frequency $f = 700$ MHz.

Decoder	Total Area [mm^2]	Memory [bits]	Area Saving vs SCL(8)	E_b/N_0 [dB] FER = 10^{-4}
SCL(8)	2.728	70361	–	2.6803
GPSCL(2,8,2)	1.788	64090	34.5%	2.6920
GPSCL(4,8,2)	1.609	58394	41.1%	2.7108
GPSCL(4,8,4)	1.703	65050	37.6%	2.6818
GPSCL(8,8,4)	1.563	62458	42.8%	2.6908
LPSCL(8,8,$\{2,2,4\}$)	1.288	49755	52.8%	2.7141
LPSCL(8,8,$\{2,4,4\}$)	1.345	56792	50.7%	2.7003

complexity saving, with the area occupation being as low as 47.2% of the baseline decoder. It is possible to see how the number of partitions has the biggest impact on the decoder area: GPSCL decoders with a higher P can be implemented with a lower complexity, regardless of the value of S. For LPSCL decoders, the majority of the complexity gain is brought by the value of s_{n-1}: as shown by the results relative to LPSCL(8,8,$\{2,4,4\}$), $s_{n-1} = 2$ accounts for 13.9% less area with respect to GPSCL(8,8,4). Decreasing s_{n-2} to 2 as well guarantees an additional 3.7% area reduction (see LPSCL(8,8,$\{2,2,4\}$) results).

The required E_b/N_0 value to achieve FER = 10^{-4} is also presented in Table 5.5. It can be seen how all the decoders require E_b/N_0 values very close (< 0.04 dB) to that of SCL(8) to achieve FER = 10^{-4}. In a nutshell, we obtain almost the same FER performance as the original SCL decoder with up to 52.8% area occupation reduction.

5.5 List Size Requirement for MAP Decoding

In this section, we present a simple upper bound on the list size of the SCL decoder that guarantees the same error-correction performance as the MAP decoder.

Recall that MAP decoding of a code of length N with K information bits requires finding the most reliable codeword out of the 2^K possible codewords. Note also that SCL decoding with list size 2^K provides a list of 2^K codewords from which the most reliable one is selected. Therefore,

SCL(2^K) is equivalent to MAP decoding. For polar codes, in order for SCL to be equivalent to MAP, Theorem 9 asserts that the list size can be chosen to be much smaller than 2^K. As we will argue shortly, the result of Theorem 9 can be particularly useful when low-rate polar codes are deployed.

Theorem 9. *Consider the transmission of $\mathcal{P}(N, K)$ over a BEC and let E denote the number of information bits located after the* last *frozen bit. Then, SCL decoding with $L = 2^{K-E}$ is equivalent to MAP decoding.*

Proof. Recall that for the case of the transmission over a BEC, the synthetic channels seen by the input bits are also BECs. Hence, the SCL decoder doubles the number of paths every time that one of the information bits cannot be decoded and all these paths are equally likely. Furthermore, when the SCL decoder encounters a frozen position, it tries to decode such a position and, if this is possible, it cancels all the paths that do not agree with the value of this position.

Run the SCL decoder until the last frozen bit. As there are at most $K - E$ information bits until this position, there are at most 2^{K-E} equally likely paths. Therefore, the SCL decoder will not exceed its list size. Assume now that the MAP decoder does not fail. This means that there is only one remaining path at the end of the decoding process. Since there are no more frozen bits, no more path cancellations can occur. Therefore, only one path must be available at this point and no new paths will be created while decoding the last E positions. As a result, the SCL decoder also succeeds. □

While we have proved Theorem 9 for the BEC, numerical experiments suggest that the result holds for other important channels such as the BAWGN.

Let us illustrate the importance of this result through the analysis of a practical setting. Polar codes have been selected for the transmission over the control channel in 5G. Such a task requires codes of short lengths ($\leq$ 1024) with rates as low as $1/12$ [54]. Consider a polar code of length 128 and rate $1/12$, which is optimized for SNR = 2 dB. This code has 10 information bits and 6 of them are located after the last frozen bit. Therefore, SCL(16) can provide the same result as the MAP decoder. This is illustrated in Fig. 5.22 when the transmission takes place over the BAWGN channel.

Recall that PSCL results in polar codes of length N/P that are decoded with SCL. Consider the decoding of a polar code of length 1024 and rate $1/12$ with PSCL with $P = 2$. The code is constructed for SNR = 2 dB. The first partition contains 7 information bits and 3 of them are

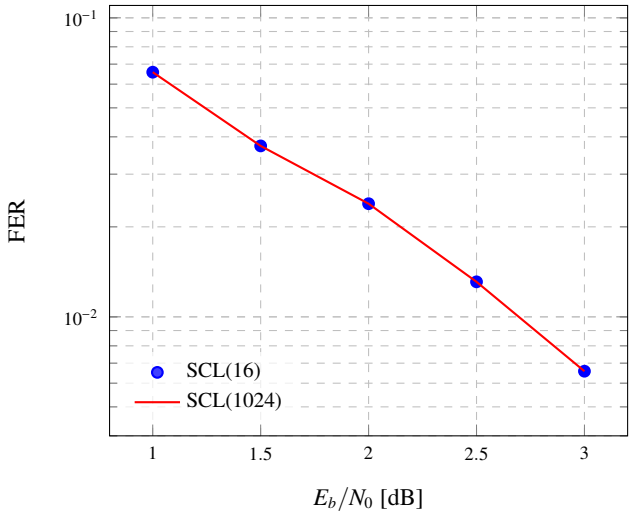


Figure 5.22: FER performance comparison between SCL(1024) and SCL(16) for the transmission of $\mathcal{P}(128, 10)$ on a BAWGN channel. Note that, since there are 10 information bits in the code, SCL(1024) is equivalent to MAP decoding. It can be seen that, since 6 of the information bits are located after the last frozen bit, SCL(16) results in the same FER performance as the MAP decoder.

located after the last frozen bit. Therefore, SCL(16) results in the same error performance as a MAP decoder over that partition. Suppose now that we decode the same code with PSCL with $P = 4$. Then, the first partition contains only frozen bits; the second partition contains 7 information bits and 3 are located after the last frozen bit; and the third partition contains 9 information bits and 3 of them are located after the last frozen bit. Therefore, for the second and the third partitions, SCL(16) and SCL(64) are respectively equivalent to the MAP decoder. In conclusion, while running PSCL decoding, it is possible to perform MAP decoding on some of the partitions with practical values of the list size.

Chapter 6

Rate-Flexible Fast Polar Decoding

Fast SC-based decoders rely on the identification of the type and the length of constituent codes in a polar code. While the calculation of the frozen/information bit sequence is straightforward and can be performed by simply assigning information bits to the first K elements of the reliability vector, the direct calculation of the list of operations for fast SC-based decoders requires complicated controller logic [11]. Therefore, the identification of the type and the length of constituent codes is performed off-line and the decoding order is stored in a dedicated memory as a list of operations [11], [36], [37]. The decoder fetches the list of operations from memory to decode the constituent codes in order one by one. The main drawbacks of the aforementioned fast SC-based decoders are twofold: first, the list of operations requires high memory usage when implemented on hardware. Second, the list of operations is highly dependent on the rate of the polar code and as the rate changes, the list of operations changes too. Therefore, for 5G applications which require the support of multiple rates, multiple lists of operations need to be stored in memory. This in turn increases the hardware implementation overhead and renders fast SC-based decoders not rate-flexible.

In this chapter, we propose completely rate-flexible fast SC-based decoders by introducing a method to infer the list of operations directly in hardware without the need to store it in memory. We show that the type and the length of a constituent code in a polar code can be identified with low hardware implementation complexity, by checking only a few bits of the constituent code. We further show that the list of operations adapts with the rate of the code, allowing the resulting fast SC-based decoder to be completely rate-flexible. We design and implement a hardware architecture for the proposed decoder and show that the memory required to store the list of operations can be

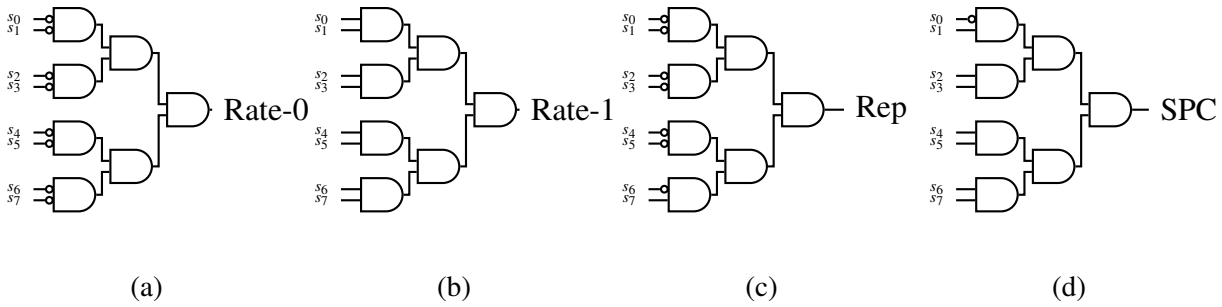


Figure 6.1: Determination of node types for fast SC-based decoding in a node of length 8. (a) Rate-0 node, (b) Rate-1 node, (c) Rep node, (d) SPC node.

completely removed, resulting in significantly lower decoder area occupation.

6.1 Rate-Flexible Decoder

The high memory usage of storing the list of operations can be mitigated by generating the list of operations on hardware as the decoding proceeds. A rudimentary approach would be to check the pattern of information and frozen bits in $\mathbf{s}$ for every encountered node. This is shown in Figure 6.1 for determining Rate-0, Rate-1, Rep, and SPC nodes of length 8. The problem with this approach is that for nodes of large length, there is a high hardware complexity overhead in determining the node types. Moreover, the module that generates the list of operations should account for the largest possible node which is the root node in the decoding tree with size N. This results in a large critical path which limits the operating frequency.

In order to tackle the above issue, the idea is to exploit the inherent order in the Bhattacharyya parameters of the bit-channels. Let W_i and W_j be the bit-channels corresponding to u_i and u_j, and let $\mathbf{b}^i$ and $\mathbf{b}^j$ be the binary expansions of the integers i and j. In [55], [56] a partial order between the polarized bit-channels was introduced. In particular, it was proven that W_i is stochastically degraded with respect to W_j, i.e., $W_i \prec W_j$, when one of the following two properties hold:

- *Addition Property*: There exists $k \in \{0, 1, \ldots, n-1\}$ such that

$$\begin{cases} b^i_m = b^j_m, & \text{if} \quad m \neq k, \\ b^i_k = 0, \\ b^j_k = 1. \end{cases} \tag{6.1}$$

- *Left-Swap Property*: There exist $k, l \in \{0, 1, \ldots, n-1\}$ such that $k < l$ and

$$\begin{cases} b_m^i = b_m^j, & \text{if} \quad m \neq k, m \neq l, \\ b_k^i = b_l^j = 0, \\ b_l^i = b_k^j = 1. \end{cases} \tag{6.2}$$

Recall that, if $W_i \prec W_j$, then all the reliability measures of W_i are worse than those of W_j, i.e., W_i has smaller mutual information, larger Bhattacharyya parameter, and larger error probability. Consequently, if u_j belongs to the frozen set, then also u_i belongs to the frozen set. Furthermore, if u_i belongs to the information set, then also u_j belongs to the information set. By using the two properties above, it was shown in [57] that it suffices to compute the reliability of a sublinear fraction of channels in order to identify the frozen and the information sets.

Another option to find an ordering between the Bhattacharyya parameters of the bit-channels can be described as follows. Consider the transmission over a Binary Memoryless Symmetric (BMS) channel W with Bhattacharyya parameter $Z(W)$ and define the synthetic channels W^0 and W^1 as

$$\begin{aligned} W^0(y_1, y_2 \mid x_1) &= \sum_{x_2} \frac{1}{2} W(y_1 \mid x_1 \oplus x_2) W(y_2 \mid x_2), \\ W^1(y_1, y_2, x_1 \mid x_2) &= \frac{1}{2} W(y_1 \mid x_1 \oplus x_2) W(y_2 \mid x_2). \end{aligned} \tag{6.3}$$

Then, the following inequalities between $Z(W^0)$, $Z(W^1)$ and $Z(W)$ hold

$$\begin{aligned} Z(W)\sqrt{2 - Z(W)^2} \leq Z(W^0) \leq 2Z(W) - Z(W)^2, \\ Z(W^1) = Z(W)^2, \end{aligned} \tag{6.4}$$

which follow from Proposition 5 of [5] and from Exercise 4.62 of [58]. Furthermore, the bit-channel W_i corresponding to u_i is given by the recursive formula below:

$$W_i = (((W^{b_{n-1}^i})^{b_{n-2}^i})^{\cdots})^{b_0^i}. \tag{6.5}$$

In what follows, we will denote by Z_i be the Bhattacharyya parameter of W_i.

At this point, we are ready to state and prove the first result of this chapter, which concerns the

identification of Rate-0, Rate-1, Rep, and SPC nodes.

Theorem 10. *Consider a polar code of length $N = 2^n$. Then, the following properties hold:*

1. *If $s_{N-1} = 0$, then the polar code represents a Rate-0 node.*
2. *If $s_0 = 1$, then the polar code represents a Rate-1 node.*
3. *If $s_{N-1} = 1$ and $s_{N-2} = 0$, then the polar code represents a Rep node.*
4. *If $s_0 = 0$ and $s_1 = 1$, then the polar code represents an SPC node.*

Proof.

1. Note that $\mathbf{b}^{N-1} = \{1, \ldots, 1\}$. By using the addition property (6.1), we obtain that $W_i \prec W_{N-1}$ for any $i \in \{0, 1, \ldots, N-2\}$. Hence, as $s_{N-1} = 0$, $s_i = 0$ for any $i \in \{0, 1, \ldots, N-2\}$. This means that the polar code consists of only frozen bits, i.e., it is a Rate-0 node.

2. Note that $\mathbf{b}^0 = \{0, \ldots, 0\}$. By using the addition property (6.1), we obtain that $W_0 \prec W_i$ for any $i \in \{1, 2, \ldots, N-1\}$. Hence, as $s_0 = 1$, $s_i = 1$ for any $i \in \{1, 2, \ldots, N-1\}$. This means that the polar code consists of only information bits, i.e., it is a Rate-1 node.

3. Note that $\mathbf{b}^{N-2} = \{1, \ldots, 1, 0\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_i \prec W_{N-2}$ for any $i \in \{0, 1, \ldots, N-3\}$. Hence, as $s_{N-2} = 0$, $s_i = 0$ for any $i \in \{0, 1, \ldots, N-3\}$. As $s_{N-1} = 1$, the polar code consists of frozen bits except for the last bit which is an information bit, i.e., it is a Rep node.

4. Note that $\mathbf{b}^1 = \{0, \ldots, 0, 1\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_1 \prec W_i$ for any $i \in \{2, 3, \ldots, N-1\}$. Hence, as $s_1 = 1$, $s_i = 1$ for any $i \in \{2, 3, \ldots, N-1\}$. As $s_0 = 0$, the polar code consists of information bits except for the first bit which is a frozen bit, i.e., it is an SPC node.

□

An immediate consequence of Theorem 10 is that, by checking only one bit, we can find out if a constituent node is either a Rate-0 or a Rate-1 node. Furthermore, by checking only two bits, we can find out if a constituent node is either a Rep or an SPC node. This observation significantly reduces the hardware complexity associated with the on-line node identification. In addition, the proposed approach is independent of the node length, making it suitable for codes of any length and rate. Figure 6.2 shows the circuit required to generate the list of operations on-line for a polar

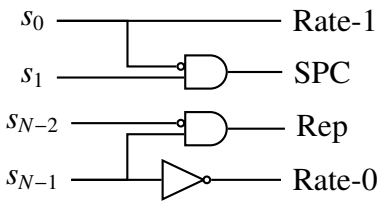


Figure 6.2: Efficient generation of the list of operations on hardware.

code of length N. It can be seen that the circuit consists of only three NOT gates and two AND gates.

Let us now state and prove the second result of this chapter, which concerns the identification of Type-I, Type-II, Type-III, Type-IV, and Type-V nodes.

Theorem 11. *Consider a polar code of length $N = 2^n$. Then, the following properties hold:*

1. *If $s_{N-1} = 1$, $s_{N-2} = 1$, and $s_{N-3} = 0$, then the polar code represents a Type-I node.*
2. *If $s_{N-1} = 1$, $s_{N-2} = 1$, $s_{N-3} = 1$, and $s_{N-5} = 0$, then the polar code represents a Type-II node.*
3. *If $s_0 = 0$, $s_1 = 0$, and $s_2 = 1$, then the polar code represents a Type-III node.*
4. *If $s_0 = 0$, $s_1 = 0$, $s_2 = 0$, and $s_4 = 1$, then the polar code represents a Type-IV node.*
5. *If $s_{N-1} = 1$, $s_{N-2} = 1$, $s_{N-3} = 1$, $s_{N-4} = 0$, $s_{N-5} = 1$, and $s_{N-9} = 0$, then the polar code represents a Type-V node.*

Proof. 1. Note that $\mathbf{b}^{N-3} = \{1, \ldots, 1, 0, 1\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_i \prec W_{N-3}$ for any $i \in \{0, 1, \ldots, N-4\}$. Hence, as $s_{N-3} = 0$, $s_i = 0$ for any $i \in \{0, 1, \ldots, N-4\}$. As $s_{N-1} = 1$ and $s_{N-2} = 1$, the polar code consists of frozen bits except for the last two bits which are information bits, i.e., it is a Type-I node.

2. Note that $\mathbf{b}^{N-5} = \{1, \ldots, 1, 0, 1, 1\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_i \prec W_{N-5}$ for any $i \in \{0, 1, \ldots, N-6\}$. Hence, as $s_{N-5} = 0$, $s_i = 0$ for any $i \in \{0, 1, \ldots, N-6\}$. Furthermore, note that $\mathbf{b}^{N-4} = \{1, \ldots, 1, 1, 0, 0\}$. Let W be the transmission channel and let z be the Bhattacharyya parameter of the channel defined as

$$(((W^{\overbrace{1)^1)^{\cdots})^1}^{n-3 \text{ times}}}.$$

Then, by using (6.4), we have that

$$Z_{N-5} \leq (2z - z^2)^4,$$
$$Z_{N-4} \geq z^2 \sqrt{2 - z^4} \sqrt{2 - z^4(2 - z^4)}.$$

It is easy to check that, for any $z \in [0, 1]$,

$$(2z - z^2)^4 \leq z^2 \sqrt{2 - z^4} \sqrt{2 - z^4(2 - z^4)}, \tag{6.6}$$

which implies that

$$Z_{N-5} \leq Z_{N-4}.$$

Consequently, as $s_{N-5} = 0$, $s_{N-4} = 0$. As a result, since $s_{N-1} = 1$, $s_{N-2} = 1$ and $s_{N-3} = 1$, the polar code consists of frozen bits except for the last three bits which are information bits, i.e., it is a Type-II node.

3. Note that $\mathbf{b}^2 = \{0, \ldots, 0, 1, 0\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_2 \prec W_i$ for any $i \in \{3, 4, \ldots, N-1\}$. Hence, as $s_2 = 1$, $s_i = 1$ for any $i \in \{3, 4, \ldots, N-1\}$. As $s_0 = 0$ and $s_1 = 0$, the polar code consists of information bits except for the first two bits which are frozen bits, i.e., it is a Type-III node.

4. Note that $\mathbf{b}^4 = \{0, \ldots, 0, 1, 0, 0\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_4 \prec W_i$ for any $i \in \{5, 6, \ldots, N-1\}$. Hence, as $s_4 = 1$, $s_i = 1$ for any $i \in \{5, 6, \ldots, N-1\}$. Furthermore, note that $\mathbf{b}^3 = \{0, \ldots, 0, 0, 1, 1\}$. Let W be the transmission channel and let z be the Bhattacharyya parameter of the channel defined as

$$(((W^{\overbrace{0)^0)^{\cdots})^0}^{n-3 \text{ times}}}.$$

Then, by using (6.4), we have that

$$Z_3 \leq (2z - z^2)^4,$$
$$Z_4 \geq z^2 \sqrt{2 - z^4} \sqrt{2 - z^4(2 - z^4)}.$$

Since (6.6) holds for any $z \in [0, 1]$, we obtain that

$$Z_3 \leq Z_4.$$

Consequently, as $s_4 = 1$, $s_3 = 1$. As a result, since $s_0 = 0$, $s_1 = 0$ and $s_2 = 0$, the polar code consists of information bits except for the first three bits which are frozen bits, i.e., it is a Type-IV node.

5. Note that $\mathbf{b}^{N-9} = \{1, \ldots, 1, 0, 1, 1\}$. By using the addition property (6.1) and the left-swap property (6.2), we obtain that $W_i \prec W_{N-9}$ for any $i \in \{0, 1, \ldots, N-10\}$. Hence, as $s_{N-9} = 0$, $s_i = 0$ for any $i \in \{0, 1, \ldots, N-10\}$. By using again the left-swap property (6.2), we obtain that $W_{N-6} \prec W_{N-4}$ and $W_{N-7} \prec W_{N-4}$. By using again the addition property (6.1), we obtain that $W_{N-8} \prec W_{N-4}$. Hence, as $s_{N-4} = 0$, $s_i = 0$ for any $i \in \{N-6, N-7, N-8\}$. As a result, since $s_{N-1} = 1$, $s_{N-2} = 1$, $s_{N-3} = 1$, and $s_{N-5} = 1$, the polar code is a Type-V node.

□

The proofs for the identification of Rate-0, Rep, SPC, Rate-1, Type-I, Type-III, and Type-V nodes are based on stochastic degradation arguments. Consequently, these proofs are general and do not depend on the fact that the frozen bits are determined according to the value of the Bhattacharyya parameter.

On the contrary, the proofs for Type-II and Type-IV nodes use the inequalities (6.4) which are valid for Bhattacharyya parameters. However, let us point out that the strategy of the proof (use extremes of information combining bounds such as (6.4) in order to compare the reliability of specific channels) is general. In order to prove a similar statement for different reliability measures, one would need to find bounds of the form (6.4) for the desired reliability measure (e.g., mutual information, error probability). Let us further clarify that the proofs for Type-II and Type-IV nodes provide an ordering between the Bhattacharyya parameter of bit-channels. As such, they do not depend on the particular technique used to compute those Bhattacharyya parameters (Gaussian approximation [53], beta-expansion [59], Monte Carlo simulation [5], etc.). Let us also note that the Bhattacharyya parameter represents the typical performance metric employed for code construction [49], [60], [61].

It is also worth mentioning that since every node in the SC-based decoding tree represents a polar code constructed for a different channel [5], the results in this section are valid for all the nodes in any polar code of any length.

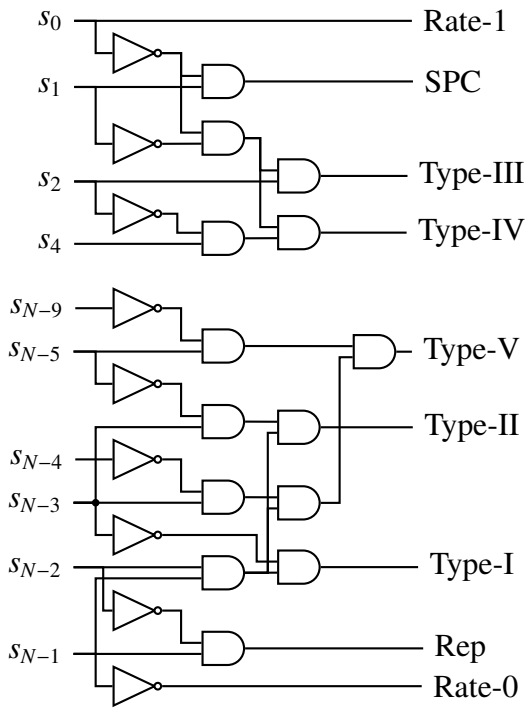


Figure 6.3: Efficient generation of the list of operations on hardware considering new nodes.

6.2 Decoder Architecture

As a proof of concept, a decoder architecture implementing the proposed technique has been designed. It implements the LPSCL decoding and the Fast-SSCL-SPC algorithm, along with the memory-reduction techniques proposed in Chapter 5. The LPSCL decoder decreases the memory requirements of standard SCL decoding by dividing the SC tree in different partitions; the bottom part of the SC tree belonging to each partition is decoded with SCL with a list size $L_{\max}$. When information needs to be passed between partitions, i.e. at the top stages of the tree, only $L_t < L_{\max}$ candidate codewords are passed, with L_t decreasing progressively as the stage t increases. The Fast-SSCL-SPC algorithm is applied to the lower stages of the tree, where $L_{\max}$ candidates are considered.

The proposed decoder is based on a semi-parallel SCL architecture, where $L_{\max}$ sets of P_e PEs are instantiated in parallel. Each set works on a different candidate codeword and relies on a dedicated memory to store the internal LLR values relative to all stages of the SC decoding tree. LLRs are quantized with Q_{LLR} bits, and represented with sign and magnitude. Each stage

of the SC decoding tree requires the storage of 2^{t-1} LLRs: however, given the limited number of PEs instantiated, the LLR memory is split in high stage and low stage memories. The high stage memory stores LLRs of stages with nodes of size greater than P_e: at stage t, where $2^t > 2P_e$, a total of $2^t/(2P_e)$ time steps are needed to descend to the lower tree level. The depth of the high stage memory is $\sum_{j=\log_2 P_e+1}^{t_{\max}-1} 2^j/P_e = N/P_e - 2$, while it is $Q_{\text{LLR}} \times P_e$ wide. The low stage memory stores LLRs for stages where $2^t \leq 2P_e$, and it is Q_{LLR} bits wide, while its depth is $\sum_{j=0}^{\log_2 P_e-1} P_e/2^j = 2P_e-2$. High and low stage memory words are rewritten when a node belonging to the same stage t is traversed. L different instantiations of both high and low stage memories are required. L separate memories store the hard bit estimates for all the tree stages as well, updating them every time that a bit is estimated. Path metrics, that identify the likelihood of a candidate codeword (or path) to be correct, are incremented every time a bit is estimated differently from the sign of the LLR associated to it. They are sorted before and after the estimation of an information bit, in order to identify the $L_{\max}$ surviving paths out of the $2L_{\max}$ created.

This baseline architecture has been modified to implement the LPSCL decoder. The bottom stages of the SC decoding tree are left unchanged, and decoded with a list size $L_{\max}$. Given the partitioning factor P, the top $\log_2 P$ stages rely on a smaller list size L_t, with $\log_2(N/P) < t \leq \log_2 N$, and $L_t \geq L_{t+1}$. Consequently, only L_t LLR memories are instantiated in the upper stages, reducing the LLR memory requirements for each upper stage of a factor $\frac{L_{\max}-L_t}{L_{\max}}$. Depending on the number of instantiated PEs and on the partitioning factor, the high and/or low stage memories might need to be separated into different memory structures, each part belonging to a different layer of LPSCL and thus instantiated a different number of times, depending on L_t. Since the number of surviving paths is reduced from $L_{\max}$ to L_t when ascending the tree above stage $\log_2(N/P)$, the path metrics need to be sorted not only before and after the estimation of an information bit, but also when $i \mod (N/P) = 0$, where i is the index of the codeword bit that needs to be estimated, and mod represents the modulo operation. This allows the most reliable paths, their LLR values, and their hard bit estimates to be transferred between partitions.

The implementation of the Fast-SSCL-SPC algorithm requires more substantial modifications. As detailed in [36], the hard bit estimate memory and path memories are updated according to different values depending on the node type, along with path metrics. This requires different parallel instantiations of the path metric computation logic, and more complex routing and selection logic are necessary to update memories, since multiple concurrent values need to be updated and propagated through the hard bit estimates memory structure. A sorter module for LLR values is

needed in Rate-1 and SPC nodes, to identify the order with which bits are estimated: the disruption of the sequential bit estimation order that SC is based on leads to additional complexity in memory updates and control logic.

In [36], the proposed decoder architecture relied on an off-line compiler to obtain the sequence of special nodes and their size. These informations differ for every code supported by the decoder, and need to be stored in a memory. Note that the frozen and information bit sequence, treated as an input, can be either stored in a memory, as supposed by most decoder architectures in literature, or computed online given the bit-channel relative reliability vector and the desired code rate, as proposed in [26]. This approach is significantly more efficient in case of multi-code decoders, and is facilitated by nested reliability vectors as those selected for the 5G eMBB control channel [27]. The control unit of the proposed architecture implements the proposed special node online identification, based on the sequence of frozen and information bits. Figure 6.2 shows the simple logic needed to identify the considered special nodes. Since the inputs s_i include a wide variety of bits, and given the low complexity of the node identification circuit, the structure is instantiated at every tree stage t, separately at every partition identified by LPSCL, to reduce the amount of multiplexing needed at the inputs and the possible increase in the system critical path.

The logic pictured in Figure 6.2 is inserted within a Finite State Machine (FSM) in the decoder control unit to identify the correct decoding phase, through two main control signals, `NodeType` and `NodeSize`. A maximum `NodeSize` value for each `NodeType` is selected at design time, to limit the additional complexity and critical path degradation.

- While the general node type can be identified easily through the proposed identification, different decoding phases are foreseen within each special node. Thus, `NodeType` foresees subtypes in the special node. While the Rate-0 node is a standalone node type, the Rate-1 node is divided into three subtypes: one phase is assigned to the fetching and sorting of the LLR values, a second to the estimation of the bits associated to the least reliable LLR values, and the third for the hard-decision on the remainder of the bits. The Rep node is divided in two subtypes, one for the frozen bits and one for the information bit. Finally, SPC nodes foresee four subtypes: one for the concurrent fetching and sorting of LLR values and frozen bit selection, one for the bit estimations, one for the hard decision on the remaining bits, and one for the parity correction. The `NodeType` signal is thus influenced not only by the result of the logic in Figure 6.2, but also by the number of estimated bits within the special node, the stage t, and the current `NodeType` subtype.

- The control unit identifies the size of the special node `NodeSize` as 2^t, given the current SC decoding tree stage t. This information is used to update the index i of the codeword bit to be estimated. The index i is usually updated once a leaf node has been reached and the corresponding bit estimated, but during the decoding of special nodes, it is kept fixed pointing at the first bit of the node. Once the decoding is terminated, the index is updated as $i +$ `NodeSize`.

6.3 Hardware Implementation Results

The proposed decoder architecture has been described in VHDL and synthesized in TSMC 65 nm CMOS technology. Two versions of the decoder have been implemented: one considering the proposed special node identification technique, and one based on the offline identification and storage used in [36]. Both decoders target a code length $N = 1024$, rely on a partitioning factor $P = 4$, and make use of 64 parallel PEs. The bottom part of the SC tree is decoded with a list size $L_{\max} = 4$, while for the upper stages $L_{10} = L_9 = 2$. Figure 6.4 shows the FER and BER performance of the LPSCL decoder used in this chapter in comparison with SCL decoding with $L = 4$, when the transmission takes place over AWGN channel. The curves in Figure 6.4 are provided for the code rates of $\{\frac{1}{12}, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$, which are considered by the eMBB control channel in the 5G standard [54]. It can be seen that LPSCL decoding incurs negligible FER and BER performance loss with respect to SCL for all considered rates. It should be noted that the introduction of the proposed technique to infer the list of operations on the fly do not change the FER or BER performance of the decoder in comparison with the same memory-based decoder.

The channel LLR values are quantized with 4 bits and internal LLR values with 6 bits, with 2 bits assigned to the fractional part, while path metrics are quantized with 8 bits. The maximum node size is set to 16 for Rate-0 and Rep nodes, and to 64 for Rate-1 and SPC nodes. Table 6.1 reports the area occupation and achievable frequency for the proposed decoder, and for the decoder based on the offline identification technique, labelled as memory-based decoder. The two decoders differ in their implementation of the Control Unit (CU): its area occupation A_{CU} in the proposed decoder is 24% less than that of the memory-based decoder. This is due to the fact that the information computed offline in the memory-based case, i.e. the equivalent of the `NodeType` signal, needs to be inserted in an FSM analogous to that used by the control unit of the proposed decoder. This FSM handles the node subtypes and the internal counters that determine when a special

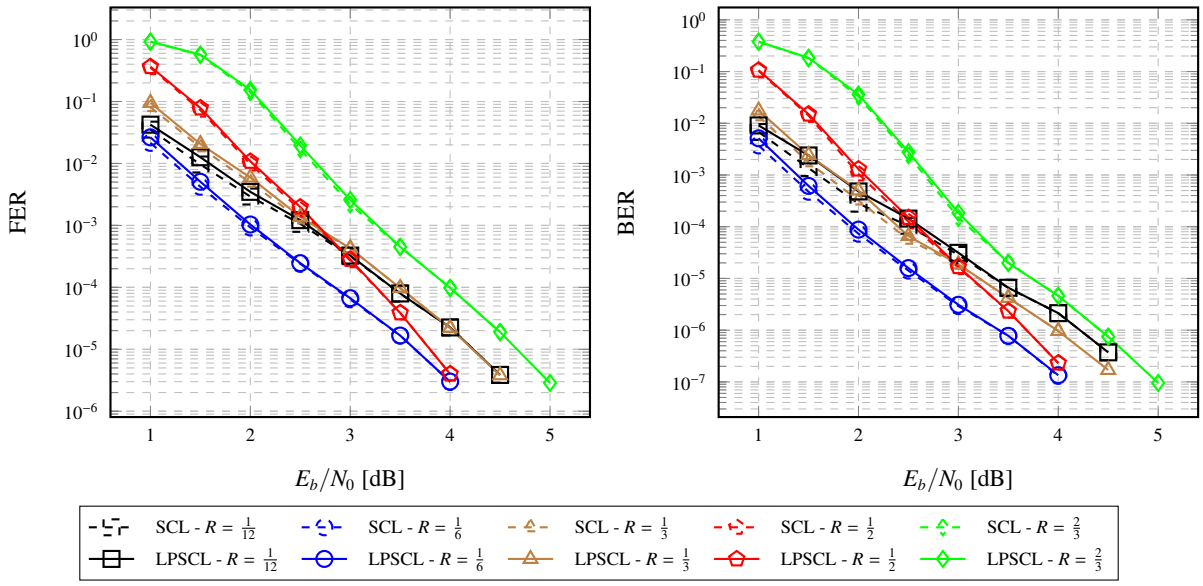


Figure 6.4: FER and BER performance comparison of decoding the 5G polar code of length $N = 1024$ and $R \in \{\frac{1}{12}, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$, using LPSCL decoding with $L_{\max} = 4$ and $L_{10} = L_9 = 2$, and SCL decoding with $L = 4$.

node decoding is terminated. Moreover, the memory-based case needs an additional information, `NodeStage`, to identify at which SC tree stage the special node is encountered: the `NodeSize` information is derived from that. The `NodeStage` signal is inserted in its own FSM, that adds substantial complexity to the control unit, resulting in a larger A_{CU}. While the contribution of A_{CU} to the total decoder area occupation A is relatively small, with $A = 1.410$ mm^2 and $A = 1.454$ mm^2 for the proposed and the memory-based decoders respectively, the `NodeStage` FSM influences signals in the `NodeSize` and `NodeType` FSM, lengthening the critical path. In particular, the state of `NodeStage` is combined to the `NodeType` and `NodeSize` to determine the current and future node subtypes. This leads to a lower achievable frequency f in the memory-based case. Table 6.1 reports the coded throughput T and area efficiency $A_{\text{eff}} = \frac{T}{A}$, for the code rates of $\{\frac{1}{12}, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$. It can be seen that the higher frequency and smaller area occupation brought by the proposed node identification method lead to higher T and substantially higher A_{eff}.

It is worth mentioning that the goal of this work is to propose a low-complexity approach to generate the list of operations for fast SC-based decoders directly on hardware, therefore, allowing for the implementation of a fast and rate-flexible SC-based decoder. Our implementation results show that by using the proposed method, there is no area occupation overhead or throughput loss

Table 6.1: TSMC CMOS 65 nm synthesis results, for $N = 1024$, $P = 4$, $L_{\max} = 4$, and $L_{10} = L_9 = 2$.

			Proposed	Memory-based
A_{CU}	[μm^2]		35881	47025
A	[mm^2]		1.410	1.454
f	[MHz]		955	926
T	[Mb/s]	$R = 1/12$	2885	2797
		$R = 1/6$	2228	2160
		$R = 1/3$	1389	1347
		$R = 1/2$	1223	1186
		$R = 2/3$	1052	1020
A_{eff}	[Mb/s/mm^2]	$R = 1/12$	2046	1924
		$R = 1/6$	1580	1486
		$R = 1/3$	985	926
		$R = 1/2$	867	816
		$R = 2/3$	746	702
Mem_{ext}	[bits]	$R = 1/12$	–	1072
		$R = 1/6$	–	1432
		$R = 1/3$	–	1496
		$R = 1/2$	–	1520
		$R = 2/3$	–	1304

in comparison with the memory-based decoders, while having a completely rate-flexible decoder.

The main advantage of the proposed approach is that given the design code length, any code with the same N can be decoded using the Fast-SSCL-SPC algorithm without foreknowledge of the information/frozen bit sequence, regardless of rate and target E_b/N_0. On the contrary, the memory-based decoder needs to store the `NodeType` and `NodeStage` information for each considered code in an external memory of Mem_{ext} bits. Considering $N = 1024$, the `NodeType` and `NodeStage` signals require a 4-bit representation. The last rows of Table 6.1 report the external memory requirements Mem_{ext} for each considered code rate: while the proposed identification method does not require any external memory, the memory-based decoder requires thousands of memory bits

Table 6.2: Comparison with state-of-the-art decoders.

	This work	[36]	[37]	[15]	[16]†	[17]†
A [mm^2]	1.410	1.797 (2.514)	1.22 (1.937)	0.62	0.73	2.00
f [MHz]	955	840	961	498	692	558
T [Mb/s]	1223	1338	1146	935	551	1578
Latency [µs]	0.84	0.77	0.89	1.10	1.86	0.66
A_{eff} [Mb/s/mm^2]	867	744 (532)	939 (592)	1508	755	789

†The results are originally based on TSMC 90 nm technology and are scaled to TSMC 65 nm technology.

for each code rate, with a total of 6824 bits for the rates considered in 5G. Implemented in TSMC 65 nm CMOS technology with registers, the area occupation of such external memory would be 0.717 mm^2.

Table 6.2 compares the proposed decoder to other architectures in the state of the art which use 64 parallel PEs. Results are reported for $\mathcal{P}(1024, 512)$ and $L = 4$. The architectures presented in [36] and [37] are based on the Fast-SSCL-SPC and SSCL-SPC algorithms, respectively: it is possible to add the cost of the external memory directly to their area occupation, and evaluate its impact on the area efficiency. These modified results are reported within parentheses. It can be seen that the external memory increases A by 40% in [36] and by 59% in [37]: the proposed special node identification technique is thus able to substantially limit the area occupation and increase the area efficiency in both architectures. The architecture presented in this work has higher A_{eff} and lower A than both [37] and [36]. Different design choices in terms of concurrent operations in the special nodes lead to a slightly lower T than [36], together with a substantially lower A and higher A_{eff}.

The architectures presented in [15]–[17] do not rely on a special-node-based decoding algorithm: thus, the throughput benefits and complexity saving of the proposed node identification technique cannot be directly evaluated. Moreover, the synthesis results of [15] were reported in 90 nm technology, but they were carried out in 65 nm technology. Therefore, a factor of $90/65$ was used to convert the frequency, and a factor of $(65/90)^2$ was used to convert the area of the decoder from 90 nm to 65 nm technology in [15]. The same conversion factors were used to convert to 65 nm technology the synthesis results in [16], [17], which were synthesized with a 90 nm node.

Our work shows 31% higher throughput and 31% lower latency with respect to the multibit decision SCL decoder architecture of [15], while the smaller area occupation of [15] leads to a higher A_{eff}. The decoder in [16] shows lower area occupation than our work. However, the architecture proposed in this work achieves 122% higher throughput and 55% lower latency, leading to 15% higher area efficiency. The high throughput SCL decoder architecture of [17] achieves higher throughput and lower latency than this work, at the cost of 42% higher area occupation and 9% lower A_{eff}. Moreover, [17] relies on tunable parameters that can lead to more than 0.2 dB error-correction performance loss. These parameters also reduce the flexibility of the decoder, since for each code rate, a different set of parameters need to be used. However, the decoder proposed in this chapter is designed to guarantee rate-flexibility, making it suitable for 5G applications.

Chapter 7

Performance of Polar Codes in 5G

In this chapter, we analyze the performance of polar codes in the 5G eMBB control channel framework. We first show that there is a specific rate at which polar codes can maximize the power efficiency of a channel at a target FER. We then show the effect of CRC on the error-correction performance of polar codes under SCL decoding for short codes. Finally, we show the effect of CRC on the speed of decoding under SSCL and Fast-SSCL decoders which were introduced in Chapter 4.

7.1 Power Efficiency

In the design of wireless communication systems, the transmitter power is limited. Therefore, power-efficient transmission is required to be able to achieve the specific FER requirements. A measure of power efficiency is the ratio of energy per information bit to the noise power per unit bandwidth, E_b/N_0 [62]. The relation between E_b/N_0 and the SNR is expressed as

$$E_b/N_0 = \text{SNR} - 10\log_{10}(2R). \tag{7.1}$$

It can be seen that power efficiency is dependent on the rate of the code. Therefore, there is a specific code rate with which maximum power efficiency is achieved.

To find such a code rate for polar codes under SC and SCL decoding schemes, we first fix a target FER of 10^{-4} and design polar codes of lengths $N \in \{32, 64, 128, 256, 512\}$ optimized for SNR $= 2$ dB. We then find the E_b/N_0 value which results in the target FER for the rates

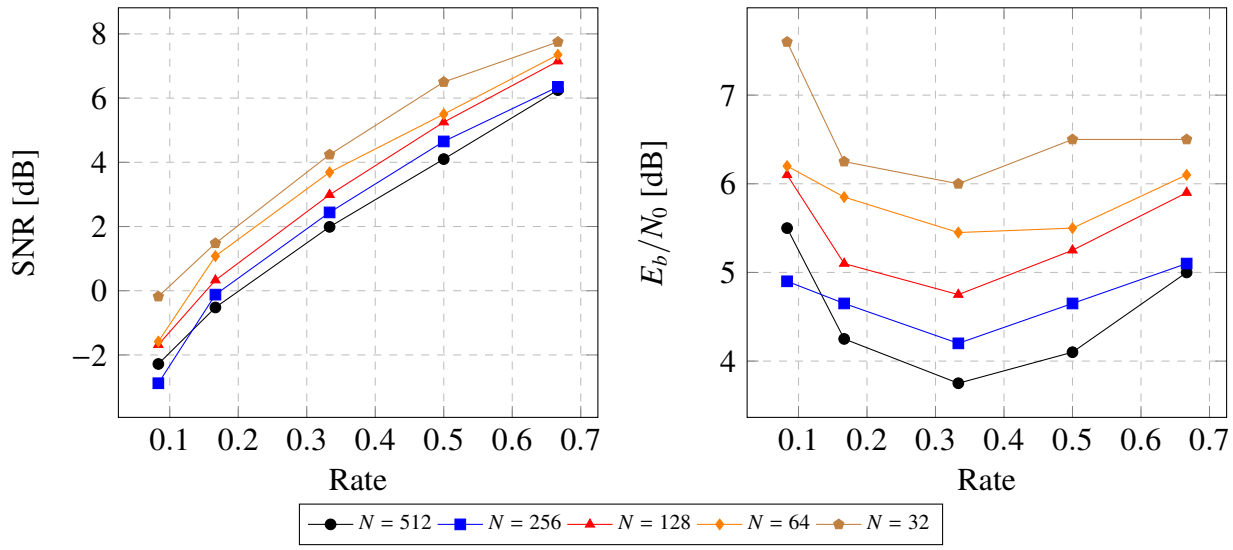


Figure 7.1: SNR and E_b/N_0 requirements for different rates of SC decoding of polar codes at FER $= 10^{-4}$ when the code is optimized for SNR = 2 dB.

$R \in \{\frac{1}{12}, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$. Figure 7.1 shows the E_b/N_0 requirements when polar codes are decoded with SC decoding and Figure 7.2 shows the E_b/N_0 requirements when polar codes are decoded with SCL decoding of list size $L = 8$. It can be seen that while the SNR requirements decrease as the code rate decreases, the E_b/N_0 requirements do not follow the same behaviour. In fact, as the rate decreases, the number of information bits per codeword decreases. If the rate is too low, it results in lower power efficiency since the power allocated to the codeword is used to transmit fewer information bits. Therefore, there is an optimal rate with which the power efficiency is maximized. In SC decoding and for all the code lengths, $R = \frac{1}{3}$ results in the best power efficiency. However, for SCL(8) decoding, the best power efficiency is achieved at a lower rate of $R = \frac{1}{6}$ for all code lengths except $N = 64$. For $N = 64$, the best power efficiency is achieved at $R = \frac{1}{2}$. However, in general we can conclude that as the list size increases[1], the rate at which the power efficiency is maximized decreases.

[1]SC can be considered as SCL with $L = 1$.

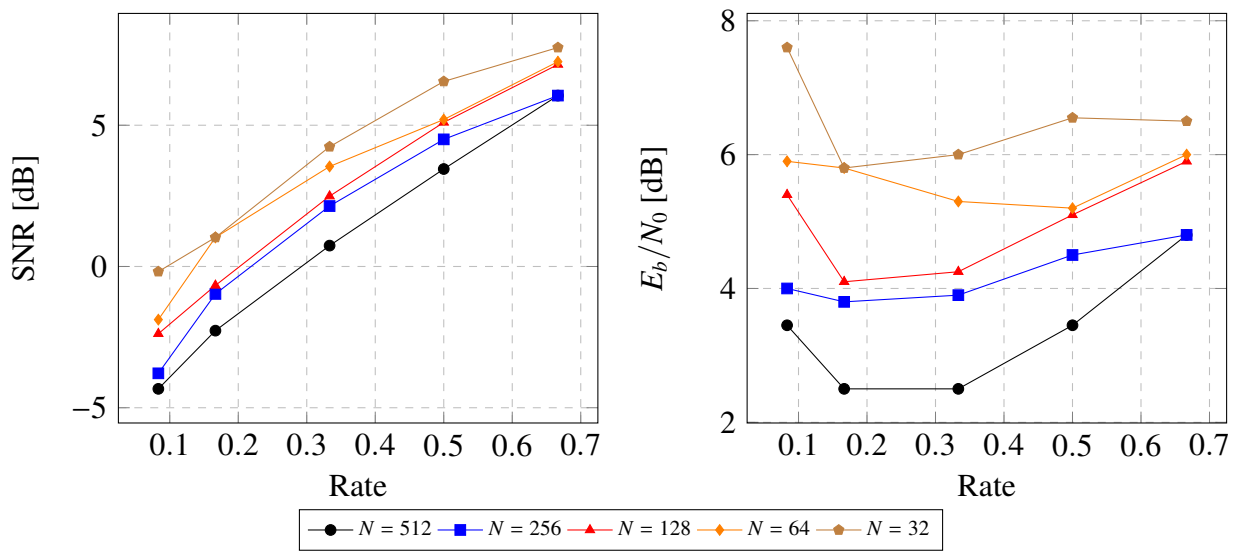


Figure 7.2: SNR and E_b/N_0 requirements for different rates of SCL(8) decoding of polar codes at FER = 10^{-4} when the code is optimized for SNR = 2 dB.

7.2 Effect of CRC on Error-Correction Performance

CRC plays an important role in finding the correct candidate in SCL decoding of polar codes and care needs to be taken to select a CRC length that results in the best error-correction performance. A small CRC may result in incorrect paths to pass the CRC. A large CRC may result in the selection of bit-channels that are not sufficiently reliable to carry CRC bits. To analyze the effect of CRC on error-correction performance of polar codes under SCL decoding, we use list size $L = 8$ for the SCL decoder and fix CRC polynomials to the ones in [63]. For each code length and code rate, we use the E_b/N_0 values obtained in Section 7.1 that result in FER = 10^{-4} when no CRC is used. We then progressively increase the length of the CRC C from 0 (no CRC) to 32 and obtain the resulting FER.

Figure 7.3 shows the effect of CRC length on the error-correction performance of polar codes of length $N = 512$ and 32 for the rates $R \in \{\frac{1}{12}, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$. We can see that for each code rate, there exists an optimal CRC length for which the FER is the lowest. For very low code rates, the optimal CRC length is zero, but as the rate increases, so does the CRC length. For example, for $N = 512$ and $R = \frac{2}{3}$, a CRC of length $C = 16$ improves the FER to 5×10^{-8}. When the code is very short, the gain obtained by adding a CRC is smaller than longer codes. This is due to the fact that, adding more CRC bits increases the effective rate more rapidly in short codeword lengths. For example,

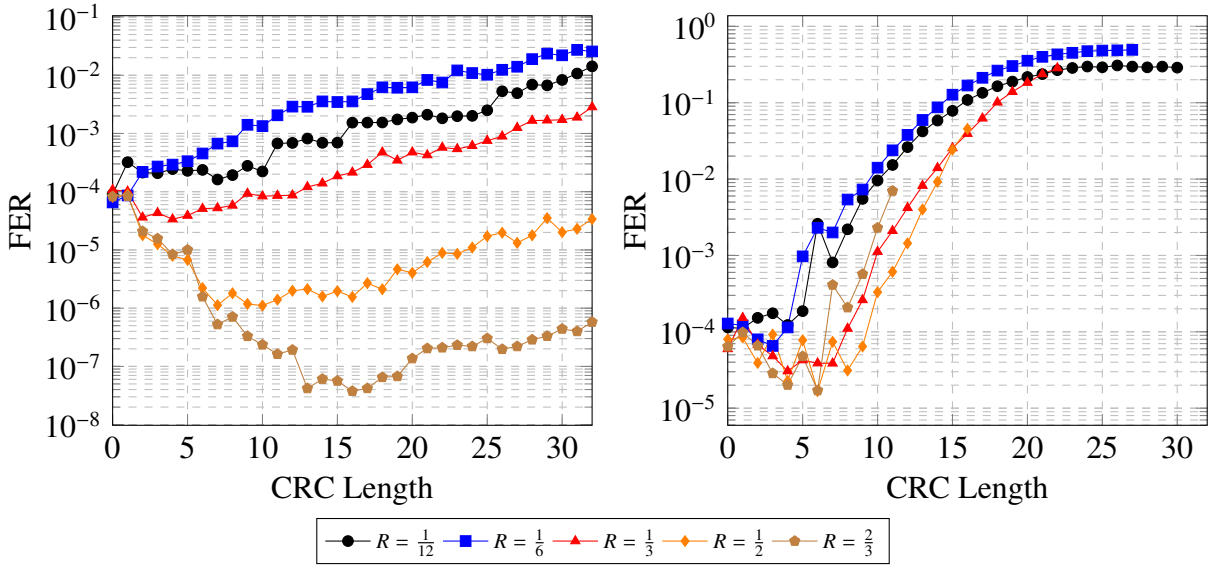


Figure 7.3: The effect of CRC length on the FER performance of polar codes of length 512 (left) and 32 (right) with different rates when decoded with SCL(8). The FER target without using CRC is 10^{-4} and the code is optimized for SNR = 2 dB.

for $N = 32$ and $R = \frac{2}{3}$, CRC of length $C = 6$ can only improve the FER to 1.7×10^{-5}.

7.3 Effect of CRC on Decoding Speed

Adding a CRC of length C to a polar code of length N with K information bits changes the effective rate of the code which is seen by the decoder to $\frac{K+C}{N}$. This in turn changes the pattern of frozen and information bits. Therefore, the number and length of special nodes in the decoding tree of polar codes changes.

We analyze the state-of-the-art flexible and high-speed SCL decoders of [37] and [36] and show what the achievable latency is when short polar codes are used in the eMBB control channel. We measure the latency by counting the number of time steps required to complete the decoding process as discussed in [36], [37]. It should be noted that the throughput of SCL decoding is inversely proportional to its latency so a smaller latency results in a higher throughput.

Figure 7.4 shows the effect of CRC length on the decoding speed of SSCL and Fast-SSCL decoding for polar codes of length 512 and 32. It can be seen that Fast-SSCL provides significant latency improvement with respect to SSCL for higher rates. This is expected since a polar code

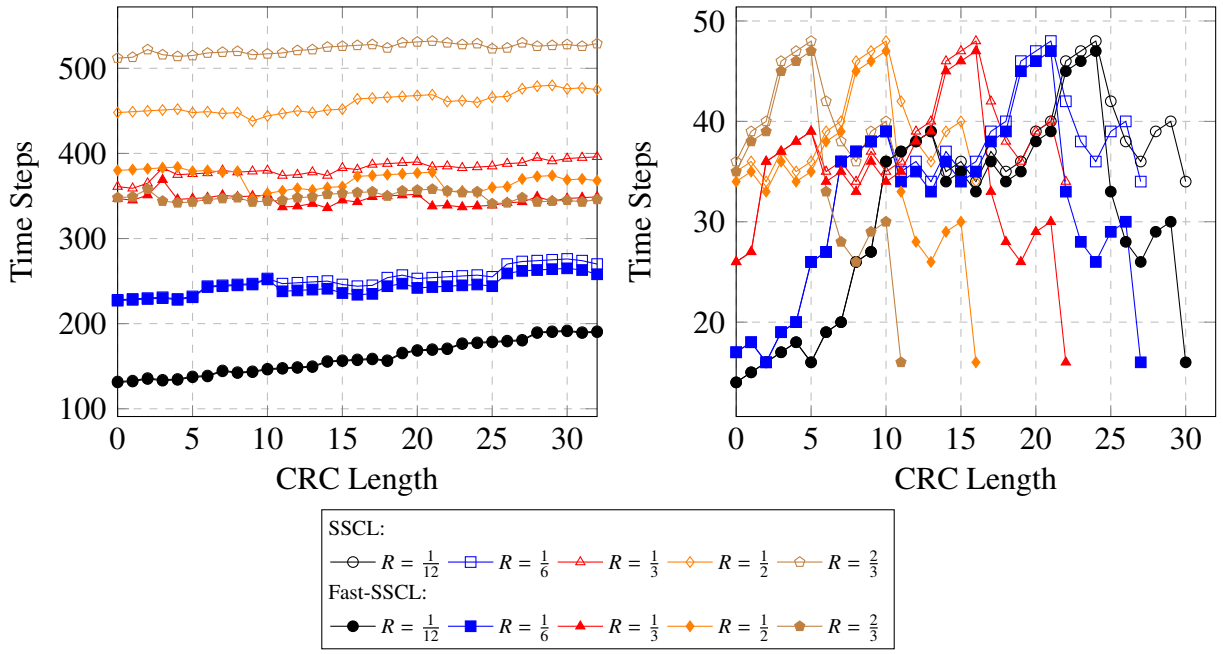


Figure 7.4: The effect of CRC length on the time step requirements of polar codes of length 512 (left) and 32 (right) with different rates when decoded with SSCL and Fast-SSCL with $L = 8$. The code is optimized for SNR = 2 dB.

of high rate has more Rate-1 nodes that can be decoded faster with Fast-SSCL. This latency improvement is more significant when a longer CRC is used, since adding CRC bits increases the effective rate of polar codes and consequently increases the number of Rate-1 nodes. The improvement is more noticeable for a longer code of length 512. For the case of $N = 32$, the latency improvement caused by employing Fast-SSCL decoding is only significant when the CRC length is high. Therefore, for polar codes of low rate and short CRC length, SSCL can be used since its latency is almost the same as that of Fast-SSCL but with a lower hardware implementation complexity. For polar codes of high rate and long CRC length, Fast-SSCL provides a high-speed alternative.

Chapter 8

Blind Detection with Polar Codes

Blind detection requires the receiver of a set of bits to identify if said bits compose a codeword of a particular channel code. In 3GPP LTE/LTE-Advanced standards blind detection is used by the UE to receive control information related to the downlink shared channel. The UE attempts the decoding of a set of candidates, to identify if one of the candidates holds its control information. Blind detection is required in 5G as well: ongoing discussions are considering a substantial reduction of the time frame allocated to blind detection, from 16μs to 4μs. Blind detection must be performed frequently, and involves a high number of decoding attempts in a limited time; it can thus lead to large implementation costs and high energy consumption. Blind detection solutions for codes adopted in existing standards can be found in [64]–[66].

In this chapter, we propose a blind detection scheme with polar codes and show that the scheme fits within 3GPP LTE-Advanced and 5G requirements. It is based on a two-step scheme: a first SC decoding phase helps selecting a set of candidates, which are subsequently decoded with SCL.

8.1 Blind Detection

The Physical Downlink Control Channel (PDCCH) is used in 3GPP LTE/LTE-Advanced to transmit the Downlink Control Information (DCI) related to the downlink shared channel. The DCI carries information regarding the channel resource allocation, transport format and hybrid automatic repeat request, and allows the UE to receive, demodulate and decode. A CRC is attached to the DCI payload before transmission, and masked according to the Radio Network Temporary Identifier (RNTI) of the UE to which the transmission is directed, or according to one of the system-wide

RNTIs. Finally, the DCI is encoded with a convolutional code. The UE is not aware of the format with which the DCI has been transmitted: it thus has to explore a combination of PDCCH locations, PDCCH formats, and DCI formats in two search spaces, and attempt decoding to identify useful DCIs. This process is called blind decoding, or blind detection. Blind detection solutions for widely adopted codes are present in literature [64], [65]. Blind detection will be present also in the 5G: ongoing discussions are considering a substantial reduction of the time frame allocated to blind detection, from 16μs to 4μs. Blind detection must be performed frequently, and given the high number of decoding attempts required in a limited time [29], it can lead to large implementation costs and energy consumption. For each PDCCH candidate in the search space, the UE performs channel decoding, and demasks the CRC with its UE RNTI. If no error is found in the CRC, the DCI is considered as carrying the UE control information.

Blind detection is used by the UE in the PDCCH in the 3GPP LTE standard to scan a set of candidate locations and decode them according to its RNTI, identifying if a transmission is targeting it. Based on LTE standard R8 [29], the performance specifications for the blind detection process are the following:

- The DCI of PDCCH is from 8 to 57 bits plus 16-bit CRC, masked by 16-bit RNTI.
- 44 candidate locations between two search spaces.
- Code length could be between 72 and 576 bits.
- Information length (including 16-bit CRC) could be between 24 and 73 bits.
- Target FER is 10^{-2}.
- False-alarm scenarios: in Type-1 the UE RNTI is not transmitted but detected, in Type-2, the UE RNTI is transmitted but another one is detected. The target False Alarm Rate (FAR) is $< 10^{-4}$.
- Missed detection occurs when UE RNTI is transmitted but not detected. The Missed Detection Rate (MDR) is close to FER curve.
- The available time frame for blind detection is 16μs.

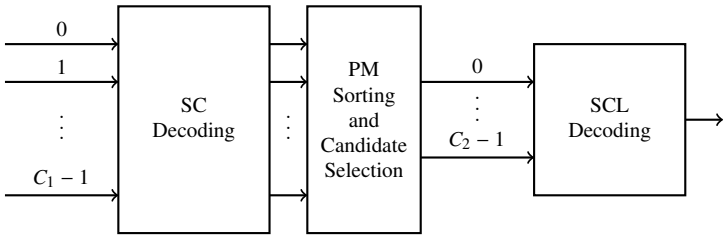


Figure 8.1: Blind detection with polar codes scheme.

8.2 Proposed Blind Detection Scheme

We propose the use of polar codes in a blind detection framework, and provide a novel two-phase blind detection scheme, shown in Fig. 8.1, where some of the frozen bit positions are used to transmit the RNTI. In the first phase, C_1 candidates are received concurrently: in our case, $C_1 = 44$. The C_1 candidates are decoded with the simple SC algorithm. A metric is obtained for each candidate, equivalent to the LLR of the last decoded bit: thanks to the serial nature of SC decoding, the LLR of the last bit can be interpreted as a reliability measure on the decoding process. The metrics are then sorted, to help the selection of the best candidates to forward to the second phase. In the second phase, C_2 candidates are selected to be decoded with the SCL decoding algorithm. SCL has a better error-correction performance, but a higher implementation complexity than SC. The C_2 candidates are chosen as all S candidates whose RNTI, after the first phase, matches the one assigned to the UE. If $S > C_2$, the ones with the highest metrics are selected. If $S < C_2$, the $C_2 - S$ candidates with the smallest metrics are selected. The candidates with large metrics have higher probability to be correctly decoded: if their RNTI does not match the one assigned to the UE, it is probably a different one. On the other hand, candidates with small metrics have a higher chance of being incorrectly decoded, and a transmission to the UE might be hiding among them. After the second phase, if one of the C_2 candidates matches the UE RNTI, it is selected, otherwise no selection is attempted.

8.2.1 Simulation Results

Simulations were performed to evaluate the FER, MDR, and FAR of the proposed scheme under a variety of parameters. For polar codes, block lengths $N = \{128, 256, 512\}$ and information lengths $K = \{8, 16, 32, 57\}$ have been considered. The number of RNTI bits is set to 16 and the position of the RNTI bits has been selected according to two operation modes. In RNTI Mode 1 (RM1), the

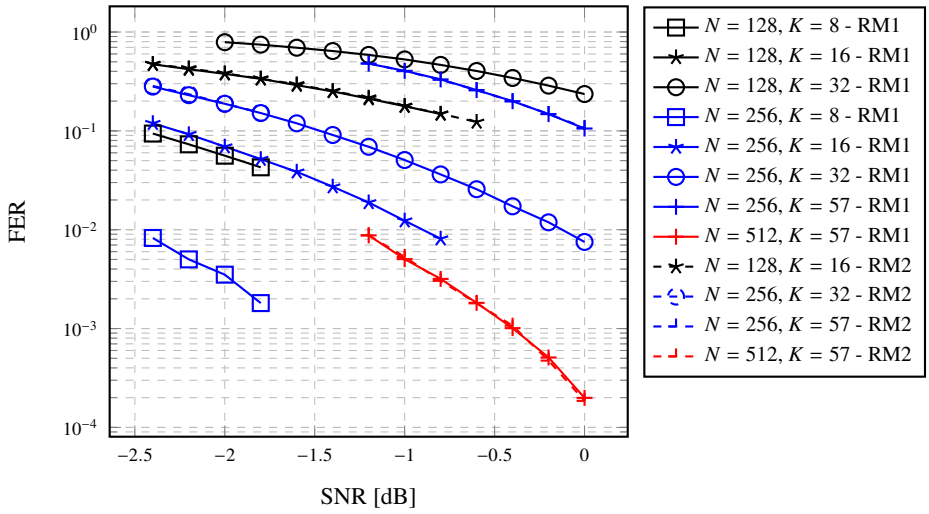


Figure 8.2: FER curves after the first phase with SC decoding.

RNTI bits are the most reliable after the K information bits. In RNTI Mode 2 (RM2), the RNTI bits are the most reliable, while the K information bits are the most reliable after the RNTI bits. The number of candidates passed to the second phase has been selected as $C_2 = \{4, 5, 6, 7\}$, and list sizes for SCL decoding have been considered as $L = \{2, 4, 8\}$. Figure 8.2 depicts the FER of the simulated codes after the first phase with SC decoding: the difference between RM1 and RM2 is generally negligible.

Figure 8.3 depicts the MDR after the second phase, where MDR is defined as the number of missed detections over the number of transmissions in which the UE RNTI was sent. MDR simulations consider $C_1/2$ candidates of length N_1, and $C_1/2$ candidates of length N_2, with an information length of $K_1 = K_2 = K$ bits. The UE RNTI is randomly transmitted through one of the C_1 codes. The curves consider the extreme values of the C_2 and L simulation space, i.e $C_2 = 4$, $L = 2$, and $C_2 = 7$, $L = 8$. Performance of the intermediate values sits in between the portrayed ones. Increasing C_2 and L leads to better MDR, regardless of the code lengths and rates. More specifically, increasing C_2 rises the probability of having, among the C_2 candidates in the second phase, the one whose RNTI matches the UE RNTI, and a larger L improves the error-correction performance of the SCL algorithm. RM2 has a substantial advantage over RM1 when MDR is high, and grants slight improvements at lower MDR. In general, the MDR curve is shown to be substantially lower than the FER curve of the least reliable of the two codes.

The false alarm curves shown in Figure 8.4 report the combination of Type-1 and Type-2 errors.

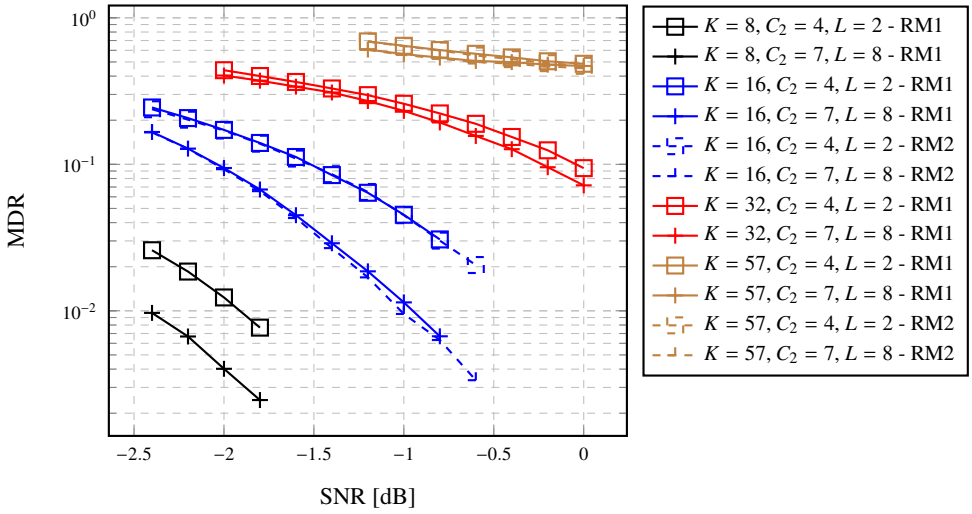


Figure 8.3: MDR after the second phase, for transmissions including $C_1/2$ cases of $N_1 = 128$, and $C_1/2$ cases of $N_2 = 256$.

The results have been obtained with the RNTIs of the C_1 candidates assuming random values over the full 16-bit dynamic. It can be seen that all curves yield FAR $< 10^{-4}$.

It is possible to use SCL during the first phase of the proposed blind detection scheme, to improve performance at the expense of implementation complexity. For example, for $C_2 = 4$, we used SCL with $L = 2$ in the first phase and SCL with $L = 8$ in the second and observed ≈ 0.8 dB improvement in FER, while the MDR was halved and the FAR remained unaffected.

A fair comparison with the state of the art is not possible, since in [67], the only other work addressing blind detection based on polar codes, no RNTI is considered, and MDR/FAR results are given within a different scenario, i.e. on the ability to detect if a frame is encoded with a particular polar code or not. Nevertheless, it is possible to observe how in [67] the FAR increases as the MDR decreases: on the contrary, the proposed scheme allows to decrease both at the same time, thus avoiding performance limitations that could make it unappealing for 5G standard applications.

8.2.2 Detection Speed

We analyze the duration of the blind detection process based on polar codes, according to the system parameters. The number of time steps required to complete the different phases can be

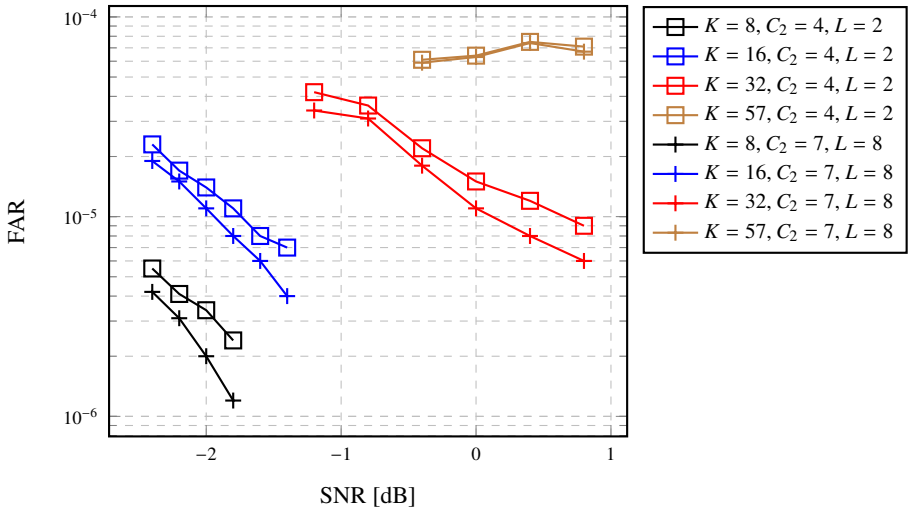


Figure 8.4: FAR after the second phase with RM1, for transmissions including $C_1/2$ cases of $N_1 = 128$ and $C_1/2$ cases of $N_2 = 256$.

computed as:

$$T_{\text{bd}} = \left\lceil \frac{C_1}{N_{\text{SC}}} \right\rceil \left(\frac{T_{\text{SC}}^1}{2} + \frac{T_{\text{SC}}^2}{2} \right) + T_{\text{sort}} + \left\lceil \frac{C_2}{N_{\text{SCL}}} \right\rceil T_{\text{SCL}}, \tag{8.1}$$

where N_{SC} and N_{SCL} are the number of parallel SC and SCL decoders, and T_{SC}^1 and T_{SC}^2 are the SC decoding latencies for codes of length N_1 and N_2, respectively. T_{SCL} is the decoding latency of an SCL decoder, while T_{sort} is the number of time steps required to obtain the C_2 candidates out of the C_1 candidate locations through sorting. The worst case for T_{SC} and T_{SCL} occurs when the standard SC and SCL algorithms are applied. In the SC case the decoding latency is expressed as $T_{\text{SC}}^i = 2N_i - 2$, and in the SCL case as:

$$T_{\text{SCL}} = \max(2N_1 + K_1, 2N_2 + K_2) + RNTI_b - 2,$$

where $RNTI_b$ represents the number of bits assigned to the RNTI. In our case $C_1 = 44$ and $RNTI_b = 16$, and we estimate $T_{\text{sort}} = C_2$, whose contribution to the latency is minimal. The worst case sees $N_1 = 512$, $N_2 = 256$, $K_1 = K_2 = 57$. The 4μs mark is achieved with $f = 800$ MHz, when $N_{\text{SC}} = 22$ and $N_{\text{SCL}} = C_2$.

Considering the Fast-SSC, SSCL, and Fast-SSCL algorithms allows to exploit particular patterns of frozen and information bits to reduce the decoding latency and thus the complexity nee-

Table 8.1: Time Steps Requirements

	Decoding Algorithm				
	SC	Fast-SSC	SCL	SSCL	Fast-SSCL $L = 2$
$\mathcal{P}(128, 32)$	254	49	302	112	86
$\mathcal{P}(128, 57)$	254	52	327	134	84
$\mathcal{P}(256, 32)$	510	109	558	163	149
$\mathcal{P}(256, 57)$	510	127	583	226	203
$\mathcal{P}(512, 32)$	1022	85	1070	140	124
$\mathcal{P}(512, 57)$	1022	91	1095	193	163

Table 8.2: Parameters needed to meet the 4μs target

Algorithm	f [MHz]	N_{SC}	N_{SCL}	Latency [μs]
SC + SCL	800	22	C_2	3.9
Fast-SSC + SSCL	300	11	$C_2/2$	3.8
	500	5	$C_2/2$	3.5
	700	3	$C_2/2$	4.0
Fast-SSC + Fast-SSCL, $L = 2$	300	11	$C_2/2$	3.7
	500	5	$C_2/2$	3.4
	700	3	$C_2/2$	3.9

ded to reach the 4μs target. In our case, the number of decoding time steps for some example codes with different decoding algorithms is detailed in Table 8.1. The worst case occurs for $N_1 = N_2 = 256$, $K_1 = 57$, $K_2 = 32$. Results are valid for RM1, RM2, and RM3. Table 8.2 reports combinations of parameters that satisfy the 4μs target. The faster decoding process of Fast-SSC, SSCL, and Fast-SSCL allows to reduce the resources needed to meet the 4μs target with respect to standard SC and SCL, at the cost of higher implementation complexity [45].

Chapter 9

Conclusion

As a new coding scheme which made their way to the upcoming 5G, polar codes require efficient decoding algorithms in order to be able to meet stringent 5G requirements. In this thesis, we addressed the issues associated with polar code decoding algorithms. We first proposed a list sphere decoding algorithm which is suitable for polar codes of short lengths. This is particularly interesting because polar codes are selected for the control link of the eMBB channel of 5G which requires codes of short lengths. We then addressed the fundamental issue with SCL decoders in which the decoder proceeds bit by bit. We identified the redundant calculations in SCL decoding algorithm and showed how to remove these redundant calculations in order to speed up the SCL decoders without incurring any error-correction performance loss. In addition, we proposed several optimization methods to increase the speed of SCL decoders even more by allowing negligible error-correction performance degradation. We further addressed the high memory consumption of SCL decoders by proposing a PSCL decoder. We showed that while the PSCL decoder can reduce the memory requirements of SCL, it may incur error-correction performance loss. Therefore, we proposed two new decoding algorithms, namely, GPSCL and LPSCL which bridged the error-correction performance gap between PSCL and SCL. We provided a CRC selection scheme to boost the error-correction performance of PSCL when it is aided by CRC. We introduced memory reduction techniques which is orthogonal to the underlying decoder and do not incur any error-correction performance loss. A rate-flexible fast polar decoder is proposed to solve the flexibility issue associated with fast polar decoders. Moreover, the performance of polar codes was evaluated in 5G framework. Finally, we demonstrated the application of polar codes in a blind detection scheme and showed that 5G requirements can be met when polar codes are used in blind detection.

9.1 Suggestions for Future Work

In this thesis, we presented several techniques to improve the throughput and reduce the area occupation associated with polar code SCL decoders. This will allow SCL decoders to be deployed in 5G applications which require high throughput, low latency, and small area occupation when implemented on hardware. However, there are still some questions which are not fully addressed. Here is a list of suggestions for future work.

9.1.1 Power and Energy Efficient Decoders

5G foresees applications which require low power consumption with high energy efficiency. This is particularly useful in mobile devices to have long battery life while they are connected to the network. In fact, error correcting codes are one of the power hungry modules in the baseband signal processing unit that a mobile device has [68]. Therefore, it is crucial to design and implement error-correcting codes which are power and energy efficient. In case of polar codes, most of the attention was paid to increasing the speed of decoders. While there are a few works that discuss power consumption of polar codes [44], research needs to be carried out to design polar code decoders which are optimized for low power and energy consumption.

9.1.2 Investigating Other Decoding Algorithms

SC decoding was the first algorithm with which polar codes could achieve the capacity of a channel. SCL showed superior error-correction performance than SC for codes of finite length. Although we addressed the issues associated with SC-based decoders in this thesis, there are other decoding algorithms which are used to decode polar codes and that have shown great potential for future implementation. Belief Propagation (BP) decoding is a high-throughput decoding algorithm which unlike SC-based decoding algorithms, decodes the bits in parallel. However, its error-correction performance is far from satisfactory when applied to decode polar codes [69], [70]. Therefore, research needs to be conducted to improve the error-correction performance of BP decoders so that they can achieve that of SCL decoders. One method to improve the error-correction performance of BP decoders is to use redundant representations of the polar code factor graph [71], [72] and run BP decoding on several factor graphs when a CRC is used to help find the correct codeword [39], [73], [74]. Although this idea is useful, it is still in its early stages and the error-correction performance of it is away from that of SCL. A combination of BP and SC decoding was also

proposed in [75] but its error-correction performance improvement comes at the cost of lower throughput than SC. Thus it needs to be optimized for high throughput applications.

9.1.3 Reed-Muller (RM) Codes

RM codes [76], [77] are similar to polar codes in the sense that the generator matrices of both codes are constructed by selecting rows from a Hadamard matrix. The row selection of polar codes minimizes the error probability under SC decoding, while the row selection of RM codes maximizes the minimum distance. As a result, polar codes outperform RM codes under SC decoding and RM codes outperform polar codes under MAP decoding. Recently, it was shown that RM codes achieve the capacity of a BEC under MAP decoding [78]. Since MAP decoding is practically intractable, sub-optimal decoding algorithms such as SC [79] and SCL [80] are used to decode RM codes. However, SC decoding provides a poor error-correction performance when used to decode RM codes and SCL decoding requires a large list size to achieve a desirable error rate. The advantage of RM codes over polar codes is that unlike polar codes, their construction is independent of the channel on which the transmission takes place. Therefore, a low-complexity decoder which can provide an error-correction performance close to the MAP decoder can be of great interest since it paves the way for RM codes to replace polar codes. A first step towards this goal is presented in [38].

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication", *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[2] R. Gallager, "Low-density parity-check codes", *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962, ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057683.

[3] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes", *Electronics Letters*, vol. 32, no. 18, pp. 1645–, Aug. 1996, ISSN: 0013-5194. DOI: 10.1049/el:19961141.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1", in *IEEE International Conference on Communications (ICC)*, vol. 2, May 1993, 1064–1070 vol.2. DOI: 10.1109/ICC.1993.397441.

[5] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels", *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009, ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2021379.

[6] 3GPP, *Final report of 3GPP TSG RAN WG1 #87 v1.0.0*, http://www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_87/Report/Final_Minutes_report_RAN1%2387_v100.zip, Reno, USA, Nov. 2016.

[7] S. Kahraman and M. Çelebi, "Code based efficient maximum-likelihood decoding of short polar codes", in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2012, pp. 1967–1971. DOI: 10.1109/ISIT.2012.6283643.

[8] K. Niu, K. Chen, and J. Lin, "Low-complexity sphere decoding of polar codes based on optimum path metric", *IEEE Communications Letters*, vol. 18, no. 2, pp. 332–335, Feb. 2014, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2014.010214.131826.

[9] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes", *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, Jan. 2013, ISSN: 1053-587X. DOI: 10.1109/TSP.2012.2223693.

[10] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes", *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2011.101811.111480.

[11] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014, ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140514.

[12] I. Tal and A. Vardy, "List decoding of polar codes", *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015, ISSN: 0018-9448. DOI: 10.1109/TIT.2015.2410251.

[13] G. Liva, L. Gaudio, T. Ninacs, and T. Jerkovits, "Code design for short blocks: A survey", *ArXiv e-prints*, Oct. 2016. arXiv: 1610.00873 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1610.00873.

[14] 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*, vol. 61, no. 8, pp. 609–613, Aug. 2014, ISSN: 1549-7747. DOI: 10.1109/TCSII.2014.2327336.

[15] B. Yuan and K. K. Parhi, "LLR-based successive-cancellation list decoder for polar codes with multibit decision", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 1, pp. 21–25, Jan. 2017, ISSN: 1549-7747. DOI: 10.1109/TCSII.2016.2546904.

[16] C. Xiong, J. Lin, and Z. Yan, "Symbol-decision successive cancellation list decoder for polar codes", *IEEE Transactions on Signal Processing*, vol. 64, no. 3, pp. 675–687, Feb. 2016, ISSN: 1053-587X. DOI: 10.1109/TSP.2015.2486750.

[17] J. Lin, C. Xiong, and Z. Yan, "A high throughput list decoder architecture for polar codes", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2378–2391, Jun. 2016, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2015.2499777.

[18] C. Xiong, J. Lin, and Z. Yan, "A multimode area-efficient SCL polar decoder", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 12, pp. 3499–3512, Dec. 2016, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2016.2557806.

[19] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes", *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 318–328, Feb. 2016, ISSN: 0733-8716. DOI: 10.1109/JSAC.2015.2504299.

[20] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes", *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, Oct. 2015, ISSN: 1053-587X. DOI: 10.1109/TSP.2015.2439211.

[21] Y. Fan, C. Xia, J. Chen, C. Y. Tsui, J. Jin, H. Shen, and B. Li, "A low-latency list successive-cancellation decoding implementation for polar codes", *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 303–317, Feb. 2016, ISSN: 0733-8716. DOI: 10.1109/JSAC.2015.2504318.

[22] B. Li, H. Shen, and K. Chen, "A decision-aided parallel SC-List decoder for polar codes", *ArXiv e-prints*, Jun. 2015. arXiv: 1506.02955 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1506.02955.

[23] C. Xia, J. Chen, Y. Fan, C. y. Tsui, J. Jin, H. Shen, and B. Li, "A high-throughput architecture of list successive cancellation polar codes decoder with large list size", *IEEE Transactions on Signal Processing*, vol. 66, no. 14, pp. 3859–3874, Jul. 2018, ISSN: 1053-587X. DOI: 10.1109/TSP.2018.2838554.

[24] J. Lin and Z. Yan, "An efficient list decoder architecture for polar codes", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2508–2518, Nov. 2015, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2014.2378992.

[25] S. A. Hashemi, C. Condo, and W. J. Gross, "List sphere decoding of polar codes", in *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Nov. 2015, pp. 1346–1350. DOI: 10.1109/ACSSC.2015.7421362.

[26] C. Condo, S. A. Hashemi, and W. J. Gross, "Efficient bit-channel reliability computation for multi-mode polar code encoders and decoders", in *IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2017, pp. 1–6. DOI: 10.1109/SiPS.2017.8109987.

[27] 3GPP, *Summary of email discussion [NRAH2-11] polar code sequence*, http://www.3gpp.org/ftp/tsg_ran/wg1_rl1/TSGR1_90/Docs/R1-1712174.zip, Prague, Czech Republic, Aug. 2017.

[28] 3GPP, *Multiplexing and channel coding*, http://www.3gpp.org/ftp/Specs/archive/38_series/38.212/38212-f11.zip, Apr. 2018.

[29] 3GPP, "Physical layer procedures", *3GPP TS 36.213 V.8.2.0*, 2008.

[30] 3GPP, *Polar code performance in fading channel*, http://www.3gpp.org/ftp/TSG_RAN/WG1_RL1/TSGR1_AH/NR_AH_1706/Docs/R1-1711752.zip, Qingdao, China, Jun. 2017.

[31] S. A. Hashemi, C. Condo, M. Mondelli, and W. J. Gross, "Rate-flexible fast polar decoders", *IEEE Transactions on Signal Processing*, 2018, submitted.

[32] C. Condo, S. A. Hashemi, A. Ardakani, F. Ercan, and W. J. Gross, "Design and implementation of a polar codes blind detection scheme", *ArXiv e-prints*, Jan. 2018. arXiv: 1801.01820 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1801.01820.

[33] S. A. Hashemi, M. Mondelli, S. H. Hassani, C. Condo, R. L. Urbanke, and W. J. Gross, "Decoder partitioning: Towards practical list decoding of polar codes", *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3749–3759, Sep. 2018, ISSN: 0090-6778. DOI: 10.1109/TCOMM.2018.2832207.

[34] S. A. Hashemi, C. Condo, F. Ercan, and W. J. Gross, "Memory-efficient polar decoders", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 604–615, Dec. 2017, ISSN: 2156-3357. DOI: 10.1109/JETCAS.2017.2764421.

[35] C. Condo, S. A. Hashemi, and W. J. Gross, "Blind detection with polar codes", *IEEE Communications Letters*, vol. 21, no. 12, pp. 2550–2553, Dec. 2017, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2017.2748940.

[36] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive-cancellation list decoders for polar codes", *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756–5769, Nov. 2017, ISSN: 1053-587X. DOI: 10.1109/TSP.2017.2740204.

[37] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2368–2380, Dec. 2016, ISSN: 1549-8328. DOI: 10.1109/TCSI.2016.2619324.

[38] S. A. Hashemi, N. Doan, M. Mondelli, and W. J. Gross, "Decoding Reed-Muller and polar codes by successive factor graph permutations", *ArXiv e-prints*, Jul. 2018. arXiv: 1807.03912 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1807.03912.

[39] N. Doan, S. A. Hashemi, M. Mondelli, and W. J. Gross, "On the decoding of polar codes on permuted factor graphs", *ArXiv e-prints*, Jun. 2018. arXiv: 1806.11195 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1806.11195.

[40] N. Doan, S. A. Hashemi, M. Mondelli, and W. J. Gross, "Neural successive cancellation decoding of polar codes", in *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, to appear, Jun. 2018.

[41] F. Ercan, C. Condo, S. A. Hashemi, and W. J. Gross, "Partitioned successive-cancellation flip decoding of polar codes", in *IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422464.

[42] S. A. Hashemi, M. Mondelli, S. H. Hassani, R. Urbanke, and W. J. Gross, "Partitioned list decoding of polar codes: Analysis and improvement of finite length performance", in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2017, pp. 1–7. DOI: 10.1109/GLOCOM.2017.8254940.

[43] S. A. Hashemi, C. Condo, F. Ercan, and W. J. Gross, "On the performance of polar codes for 5G eMBB control channel", in *Asilomar Conference on Signals, Systems, and Computers (ACSSC)*, Oct. 2017, pp. 1764–1768. DOI: 10.1109/ACSSC.2017.8335664.

[44] F. Ercan, C. Condo, S. A. Hashemi, and W. J. Gross, "On error-correction performance and implementation of polar code list decoders for 5G", in *Annual Allerton Conference on Communications, Control, and Computing (Allerton)*, Oct. 2017, pp. 443–449. DOI: 10.1109/ALLERTON.2017.8262771.

[45] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast simplified successive-cancellation list decoding of polar codes", in *IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2017, pp. 1–6. DOI: 10.1109/WCNCW.2017.7919044.

[46] S. A. Hashemi, C. Condo, and W. J. Gross, "Simplified successive-cancellation list decoding of polar codes", in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016, pp. 815–819. DOI: 10.1109/ISIT.2016.7541412.

[47] S. A. Hashemi, C. Condo, and W. J. Gross, "Matrix reordering for efficient list sphere decoding of polar codes", in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 1730–1733. DOI: 10.1109/ISCAS.2016.7538902.

[48] 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, pp. 957–960. DOI: 10.1109/ICASSP.2016.7471817.

[49] I. Tal and A. Vardy, "How to construct polar codes", *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013, ISSN: 0018-9448. DOI: 10.1109/TIT.2013.2272694.

[50] M. Hanif and M. Ardakani, "Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes", *IEEE Communications Letters*, vol. 21, no. 11, pp. 2360–2363, Nov. 2017, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2017.2740305.

[51] C. Condo, M. Martina, G. Piccinini, and G. Masera, "Variable parallelism cyclic redundancy check circuit for 3GPP-LTE/LTE-Advanced", *IEEE Signal Processing Letters*, vol. 21, no. 11, pp. 1380–1384, Nov. 2014, ISSN: 1070-9908. DOI: 10.1109/LSP.2014.2334393.

[52] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "From polar to reed-muller codes: A technique to improve the finite-length performance", *IEEE Transactions on Communications*, vol. 62, no. 9, pp. 3084–3091, Sep. 2014, ISSN: 0090-6778. DOI: 10.1109/TCOMM.2014.2345069.

[53] P. Trifonov, "Efficient design and decoding of polar codes", *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3221–3227, Nov. 2012, ISSN: 0090-6778. DOI: 10.1109/TCOMM.2012.081512.110872.

[54] 3GPP, *Evaluation on channel coding candidates for eMBB control channel*, 3GPP TSG RAN WG1 #87, R1-1611109, Reno, USA, Nov. 2016.

[55] C. Schürch, "A partial order for the synthesized channels of a polar code", in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016, pp. 220–224. DOI: 10.1109/ISIT.2016.7541293.

[56] M. Bardet, V. Dragoi, A. Otmani, and J. Tillich, "Algebraic properties of polar codes from a new polynomial formalism", in *IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016, pp. 230–234. DOI: 10.1109/ISIT.2016.7541295.

[57] M. Mondelli, S. H. Hassani, and R. Urbanke, "Construction of polar codes with sublinear complexity", in *IEEE International Symposium on Information Theory (ISIT)*, Jun. 2017, pp. 1853–1857. DOI: 10.1109/ISIT.2017.8006850.

[58] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[59] G. He, J. C. Belfiore, I. Land, G. Yang, X. Liu, Y. Chen, R. Li, J. Wang, Y. Ge, R. Zhang, and W. Tong, "Beta-expansion: A theoretical framework for fast and recursive construction of polar codes", in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254146.

[60] J. Guo, M. Qin, A. G. i Fàbregas, and P. H. Siegel, "Enhanced belief propagation decoding of polar codes through concatenation", in *IEEE International Symposium on Information Theory (ISIT)*, Jun. 2014, pp. 2987–2991. DOI: 10.1109/ISIT.2014.6875382.

[61] H. Vangala, E. Viterbo, and Y. Hong, "A comparative study of polar code constructions for the AWGN channel", *ArXiv e-prints*, Jan. 2015. arXiv: 1501.02473 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1501.02473.

[62] M. C. Gursoy, H. V. Poor, and S. Verdu, "The capacity and power efficiency of OOFSK signaling over wideband fading channels", in *IEEE Global Communications Conference (GLOBECOM)*, vol. 1, Nov. 2004, pp. 441–445. DOI: 10.1109/GLOCOM.2004.1377986.

[63] P. Koopman, *Best CRC polynomials*, https://users.ece.cmu.edu/~koopman/crc/, Accessed: 2017-11-23.

[64] R. Moosavi and E. G. Larsson, "A fast scheme for blind identification of channel codes", in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2011, pp. 1–5. DOI: 10.1109/GLOCOM.2011.6133507.

[65] T. Xia and H. C. Wu, "Novel blind identification of LDPC codes using average LLR of syndrome a posteriori probability", *IEEE Transactions on Signal Processing*, vol. 62, no. 3, pp. 632–640, Feb. 2014, ISSN: 1053-587X. DOI: 10.1109/TSP.2013.2293975.

[66] J. Zhou, Z. Huang, C. Liu, S. Su, and Y. Zhang, "Information-dispersion-entropy-based blind recognition of binary bch codes in soft decision situations", *Entropy*, vol. 15, no. 5, pp. 1705–1725, 2013, ISSN: 1099-4300. DOI: 10.3390/e15051705. [Online]. Available: http://www.mdpi.com/1099-4300/15/5/1705.

[67] P. Giard, A. Balatsoukas-Stimming, and A. Burg, "Blind detection of polar codes", in *IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2017, pp. 1–6. DOI: 10.1109/SiPS.2017.8109977.

[68] S. Chouhan, R. Bose, and M. Balakrishnan, "Integrated energy analysis of error correcting codes and modulation for energy efficient wireless sensor nodes", *IEEE Transactions on Wireless Communications*, vol. 8, no. 10, pp. 5348–5355, Oct. 2009, ISSN: 1536-1276. DOI: 10.1109/TWC.2009.090279.

[69] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders", *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6496–6506, Dec. 2014, ISSN: 1053-587X. DOI: 10.1109/TSP.2014.2366712.

[70] B. Yuan and K. K. Parhi, "Architecture optimizations for BP polar decoders", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2013, pp. 2654–2658. DOI: 10.1109/ICASSP.2013.6638137.

[71] S. B. Korada, "Polar codes for channel and source coding", PhD thesis, ISC, Lausanne, 2009, p. 181.

[72] N. Hussami, S. B. Korada, and R. Urbanke, "Performance of polar codes for channel and source coding", in *IEEE International Symposium on Information Theory (ISIT)*, Jun. 2009, pp. 1488–1492. DOI: 10.1109/ISIT.2009.5205860.

[73] A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Belief propagation decoding of polar codes on permuted factor graphs", in *IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 1–6. DOI: 10.1109/WCNC.2018.8377158.

[74] S. Cammerer, M. Ebada, A. Elkelesh, and S. ten Brink, "Sparse graphs for belief propagation decoding of polar codes", *ArXiv e-prints*, Dec. 2017. arXiv: 1712.08538 [cs.IT]. [Online]. Available: https://arxiv.org/abs/1712.08538.

[75] U. U. Fayyaz and J. R. Barry, "Low-complexity soft-output decoding of polar codes", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 958–966, May 2014, ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140515.

[76] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection", *Transactions of the IRE Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, Sep. 1954, ISSN: 2168-1740. DOI: 10.1109/IREPGELC.1954.6499441.

[77] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme", *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, Sep. 1954, ISSN: 2168-2690. DOI: 10.1109/TIT.1954.1057465.

[78] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşoğlu, and R. L. Urbanke, "Reed-Muller codes achieve capacity on erasure channels", *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4298–4316, Jul. 2017, ISSN: 0018-9448. DOI: 10.1109/TIT.2017.2673829.

[79] I. Dumer, "Recursive decoding and its performance for low-rate Reed-Muller codes", *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 811–823, May 2004, ISSN: 0018-9448. DOI: 10.1109/TIT.2004.826632.

[80] I. Dumer and K. Shabunov, "Soft-decision decoding of Reed-Muller codes: Recursive lists", *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1260–1266, Mar. 2006, ISSN: 0018-9448. DOI: 10.1109/TIT.2005.864443.