2016

Efficient decoder design for error correcting codes

Chenrong Xiong
Lehigh University

EFFICIENT DECODER DESIGN FOR ERROR CORRECTING CODES

BY

CHENRONG XIONG

PRESENTED TO THE GRADUATE AND RESEARCH COMMITTEE

OF LEHIGH UNIVERSITY

IN CANDIDACY FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

LEHIGH UNIVERSITY

MAY 2016

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Date

Prof. Zhiyuan Yan
(Dissertation Advisor)

Accepted Date

Committee Members:

Prof. Zhiyuan Yan
(Committee Chair)

Prof. Meghanad D. Wagh

Prof. Tiffany Jing Li

Prof. Warren J. Gross
McGill University

Dr. Ying Wang
Qualcomm Inc.

Acknowledgements

Though only my name appears on the cover of this dissertation, a great many kind people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience in Lehigh University has been one of the most cherishable things in my life.

First, I would like to express my deepest and most sincere gratitude and respect to my advisor, Professor Zhiyuan Yan, for his knowledge, guidance, mentorship, patience, encouragement, and friendship through my Ph.D career. I has been amazingly fortunate to have such an advisor, who always had time and patience for me and taught me so much in every aspect of research, from topic selection to idea development, from critical thinking to scientific writing, from presentation slide preparation to oral presentation skills. These skills are beneficial not only to my Ph.D study but also to my future career. Besides that, Professor Yan also recommended me for an intership, which provided me lots of precious industrial experience. I cannot have asked for a better advisor than him.

I am also very grateful to my committee members, Prof. Meghanad D. Wagh, Prof. Tiffany Jing Li, Prof. Warren J. Gross and Dr. Ying Wang for generously providing their time and expertise to better my work.

Special thanks are also due to the National Science Foundation (under Grants ECCS-0925890, ECCS-1055877 and ECCS-1509674) for the financial support during my Ph.D career.

I must acknowledge Dr. John Yu and Pinfen Lin. Their academic and industrial experiences inspire and encourage me to pursue my Ph.D. degree and to explore a wider world.

I also would like to thank all teaching and non-teaching staffs of Department of Electrical and Computer Engineering, Lehigh University for their generous assistance, help and support in numerous ways.

I am also thankful to many colleagues and friends, Ning Chen, Xuebin Wu, Feng Shi, Lihua Jiao, Hongmei Xie, Jun Lin, Yang Liu, Xuanxuan LV, Jiangfan Zhang, Ting Xu, Yun Li, Guanghai Ding, Jinhan Zhu, Guibao Xu, Li Zong, Dawei Li, Lei Zhang and Jian Lv for their encouragement, support, help, humor, and friendship. They make my life in Lehigh easier and more enjoyable.

Most importantly, none of this would have been possible without the love of my family. I would like to thank my parents and my brother for their love, their support and their unwavering faith and confidence in me. Many thanks must go to my children, two little lovely gentlemen – Zhuang Zhuang and Du Du. They make me full of energy and happiness. To my wife, Yan, words are not enough to express my gratitude. She has been standing beside me through this journey with her unconditional support, constant inspiration and encouragement, quiet patience and understanding, countless sacrifices and everlasting love.

Finally, I want to thank these people and many others not mentioned here again for their generous help and wish them all the best!

To my parents, my wife, and my children

Contents

List of Tables

List of Figures

Abstract

Error correctiong codes (ECC) are widly used in applications to correct errors in data transmission over unreliable or noisy communication channels. Recently, two kinds of promising codes attracted lots of research interest because they provide excellent error correction performance. One is non-binary LDPC codes, and the other is polar codes. This dissertation focuses on efficient decoding algorithms and decoder design for these two types of codes.

Non-binary low-density parity-check (LDPC) codes have some advantages over their binary counterparts, but unfortunately their decoding complexity is a significant challenge. The iterative hard- and soft-reliability based majority-logic decoding algorithms are attractive for non-binary LDPC codes, since they involve only finite field additions and multiplications as well as integer operations and hence have significantly lower complexity than other algorithms. We propose two improvements to the majority-logic decoding algorithms. Instead of the accumulation of reliability information in the existing majority-logic decoding algorithms, our first improvement is a new reliability information update. The new update not only results in better error performance and fewer iterations on average, but also further reduces computational complexity. Since existing majority-logic decoding algorithms tend to have a high error floor for codes whose parity check matrices have low column weights, our second improvement is a re-selection scheme, which leads to much lower error floors, at the expense of more finite field operations and integer operations, by identifying periodic points, re-selecting intermediate hard decisions, and changing reliability information.

Polar codes are of great interests because they provably achieve the symmetric capacity of discrete memoryless channels with arbitrary input alphabet sizes while having

an explicit construction. Most existing decoding algorithms of polar codes are based on bit-wise hard or soft decisions. We propose symbol-decision successive cancellation (SC) and successive cancellation list (SCL) decoders for polar codes, which use symbol-wise hard or soft decisions for higher throughput or better error performance. Then we propose to use a recursive channel combination to calculate symbol-wise channel transition probabilities, which lead to symbol decisions. Our proposed recursive channel combination has lower complexity than simply combining bit-wise channel transition probabilities. The similarity between our proposed method and Arıkan's channel transformations also helps to share hardware resources between calculating bit- and symbol-wise channel transition probabilities. To reduce the complexity of the list pruning, a two-stage list pruning network is proposed to provide a trade-off between the error performance and the complexity of the symbol-decision SCL decoder. Since memory is a significant part of SCL decoders, we also propose a pre-computation memory-saving technique to reduce memory requirement of an SCL decoder.

To reduce the complexity of the recursive channel combination further, we propose an approximate ML (AML) decoding unit for SCL decoders. In particular, we investigate the distribution of frozen bits of polar codes designed for both the binary erasure and additive white Gaussian noise channels, and take advantage of the distribution to reduce the complexity of the AML decoding unit, improving the throughput-area efficiency of SCL decoders.

Furthermore, to adapt to variable throughput or latency requirements which exist widely in current communication applications, a multi-mode SCL decoder with variable list sizes and parallelism is proposed. If high throughput or small latency is required, the decoder decodes multiple received words in parallel with a small list size. However, if error performance is of higher priority, the multi-mode decoder switches to a serial mode with a bigger list size. Therefore, the multi-mode SCL decoder provides a flexible tradeoff between latency, throughput and error performance at the expense of small overhead.

Chapter 1

Introduction

The only task of telecommunication systems is to maintain a reliable information transmission over unreliable communication channels. Unfortunately, noise is everywhere. Communication channels are not ideal and contain different kinds of noise, thus errors may be introduced during transmission from a transmitter to a receiver. Error correcting codes is used in telecommunication systems to detect or correct such errors and to restore transmitted data. There are a lot of error correcting codes: Hamming codes, Bose-Chaudhuri-Hocquenghem (BCH) codes, Reed-Solomon(RS) codes, convolutional codes, turbo codes, low-density parity check (LDPC) codes, non-binary LDPC codes, polar codes, and so on. Among these codes, LDPC codes, non-binary LDPC and polar codes have attracted lots of research interest recently because they have excellent error correction performance.

1.1 Background and Motivation

1.1.1 Non-binary LDPC codes

Low-density parity-check (LDPC) codes were first developed by Gallager [4] in 1963. They were forgotten until they were rediscovered in the late 1990s by MacKay and Neal [5]. Since then, the academic and industrial communities have focused on binary LDPC codes, because long binary LDPC codes can achieve performance approaching the Shannon limit (see, for example, [6]). Hence binary LDPC codes have been used in various applications, such as digital television, Ethernet, home networking, and Wi-Fi. Efficient decoding algorithms, encoder implementations, and decoder implementations of binary LDPC codes (see, for example, [7–9]) have received significant attentions.

In 1998, the study of Davey and MacKay [10] showed that non-binary LDPC codes over GF(q) ($q > 2$) perform better than their binary counterparts for moderate code lengths. Moreover, non-binary LDPC codes also outperform binary LDPC codes on channels with bursty errors and high-order modulation schemes [11]. These advantages have motivated a steady stream of work on code designs [12, 13], decoding algorithms [10, 11, 14–20], and decoder implementations [21–23] for non-binary LDPC codes. Davey and MacKay [10] first used belief propagation (BP) to decode non-binary LDPC codes. By applying the fast Fourier transform (FFT) of probabilities to the BP algorithm, they also proposed a fast Fourier transform (FFT) based q-ary sum-product algorithm (SPA), called FFT-QSPA [15]. The FFT-QSPA was further improved by Barnault and Declercq [16]. Song and Cruz proposed a logarithm domain FFT-BP algorithm [11]. The Min-Sum algorithm was applied to non-binary LDPC codes by Wymeersch *et al.* [17]. Then Declercq and Fossorier [18] proposed the Extended Min-

Sum (EMS) algorithm by using only a limited number of probabilities in the messages at inputs of check nodes. Savin [19] proposed the Min-Max algorithm.

The advantages of non-binary LDPC codes come at the expense of significantly higher decoding complexity than their binary counterparts. Since complexity of decoding non-binary LDPC codes is a key challenge, the iterative hard- and soft-reliability based majority-logic decoding, referred to as IHRB-MLGD and ISRB-MLGD, respectively, algorithms [20] are particularly attractive. When there is no ambiguity, MLGD is omitted when referring to majority-logic decoding algorithms for brevity. Based on the one-step majority logic decoding, these majority-logic decoding algorithms represent reliability information with finite field elements and integers, and hence involve only finite field additions (FAs) and finite field multiplications (FMs) as well as integer additions (IAs), integer comparisons (ICs), integer multiplications (IMs) and integer divisions (IDs). As a result, they require much lower computational complexities at the expense of moderate error performance degradation. For instance, while the error performance of the ISRB algorithm is within 1 dB of that of FFT-QSPA [16], its complexity is only a small fraction of that of the latter [20]. With a performance loss of 1 dB, the IHRB algorithm has even lower complexity than the ISRB algorithm [20]. Based on the IHRB algorithm, Zhang *et al.* [23] proposed an enhanced IHRB-MLGD (EIHRB) algorithm by introducing the soft-reliability initialization and re-computing the extrinsic information. The EIHRB algorithm has a similar complexity to that of the IHRB algorithm, but its error performance approaches that of the ISRB algorithm. The majority-logic decoding algorithms are particularly effective for LDPC codes constructed based on finite geometries and finite fields [13].

However, in these MLGD algorithms, the reliability information includes all check-to-variable messages of previous iterations. This leads to some performance loss and lower convergence speed. Furthermore, these MLGD algorithms tend to show a high error floor for codes whose parity check matrix have small column weights. In this dissertation, we will tackle these two problems of MLGD algorithms for non-binary LDPC codes.

1.1.2 Polar Codes

Polar codes, a groundbreaking finding by Arıkan [24] in 2009, have ignited a spark of research interest in the fields of communication and coding theory, because they provably achieve the symmetric capacity for both binary-input memoryless channel [24] and nonbinary-input memoryless channels [25]. The second reason polar codes are attractive is their low encoding and decoding complexity. For example, a polar code of length N can be decoded by the successive cancellation (SC) algorithm [24] with complexity $\mathcal{O}(N \log N)$.

Polar codes' capacity-approaching performance is achieved only when the code length is large enough if the SC algorithm is used. For a short or moderate code length, in terms of the error performance, polar codes with the SC algorithm are inferior to Turbo codes or low-density parity-check (LDPC) codes [26, 27]. Thus, a lot of effort has been made to improve the error performance of short polar codes. Systematic polar codes [28] were proposed to reduce the bit error rate (BER) while guaranteeing the same frame error rate (FER) as their non-systematic counterparts. Although a Viterbi algorithm [29], a sphere decoding algorithm [30] and stack sphere decoding algorithm [31] can provide maximum likelihood (ML) decoding of polar codes, they are considered infeasible, especially for long polar codes, due to their much higher complexity than the SC algorithm. Recently, an SC list algorithm for polar codes was proposed in [32] to bridge the performance gap between the SC algorithm and ML algorithms at the cost of complexity $\mathcal{O}(LN \log N)$, where L is the list size. Moreover, the concatenation of polar codes with cyclic redundancy check (CRC) codes was introduced in [26,33]. To decode the CRC-concatenated polar codes, a CRC detector is used in the SCL algorithm to help select the output codeword. The combination of an SCL algorithm and a CRC detector is called CRC-aided SCL (CA-SCL) algorithm. [33] shows that with the CA-SCL algorithm, the error performance of a (2048, 1024) CRC-concatenated polar code is better that of a (2304, 1152) LDPC code, which is used in the WiMax standard [34].

Several architectures have been proposed for the SC and SCL algorithms. Arıkan [24] showed that a fully parallel SC decoder has a latency of $2N-1$ clock cycles. A tree SC decoder and a line SC decoder with complexity $\mathcal{O}(N)$ were proposed in [35]. These two decoders have the same latency as the fully parallel SC decoder. To reduce complexity further, Leroux *et al.* [36] proposed a semi-parallel SC decoder for polar codes by taking advantage of the recursive structure of polar codes to reuse processing resources. Assuming that the number of processing elements (PEs) are P $(P = 2^p \leq N)$, the latency of the semi-parallel SC decoder is $2N + \frac{N}{P}\log_2(\frac{N}{4P})$ clock cycles. A scalable semi-parallel architecture was proposed in [37]. To reduce the latency, a simplified SC (SSC) polar decoder was introduced in [38] and it was further analyzed in [39]. In the SSC polar decoder, a polar code is converted to a binary tree including three types of nodes: rate-one, rate-zero and rate-R nodes. Based on the SSC polar decoder, the ML SSC decoder makes use of the ML algorithm to deal with part of rate-R nodes in [40]. However, the SSC and ML-SSC polar decoders depend on positions of information bits and frozen bits, and are code-specific consequently. In [41], a pre-computation lookahead technique was proposed to reduce the latency of the tree SC decoder by half. For the SCL polar decoder, the semi-parallel architecture was adopted in [42]. In [43] Balatsoukas-Stimming *et al*. proposed an architecture of $L = 4$ to achieve a throughput of 124 Mbps and a latency of 8.25 ms when decoding a (1024, 512) polar code. In [44], Lin and Yan designed an SCL polar decoder with a throughput of 182 Mbps and a latency of 5.63 ms. To reduce the memory requirement, the log-likelihood ratio (LLR) messages are used in [45]. The throughpu t of existing polar decoders is still not high enough for high speed applications.

The aforementioned polar decoders are based on soft or hard bit decisions. That is, they deal with bits serially, leading to low throughput (or long latency). Several previous works attempt to improve the throughput (or latency) by dealing with multiple bits at a time. In [2], the data bits of a polar code are split into several streams, which are decoded simultaneously. This idea of parallel processing is extended in [1], where the SC decoder is transformed into a concatenated decoder, where all the inner SC decoders are carried out in parallel. Yuan and Parhi proposed a multi-bit SCL decoder [3]. But their computation methods for symbol-based transition probabilities have unnecessary high complexity. The first problem we deal with here for polar codes is how to design a low-complexity ML decoding unit to calculation symbol-based transition probabilities.

Many applications, such as modern wireless or wireline communication systems, require variable data rate transmission and have stringent latency requirements. As a potential candidate of FEC technique for future communication systems, a polar decoder supporting variable data rate and variable decoding latency is desired. Unfortunately, existing polar decoders provide only fixed latency and throughput (data rate). In this dissertation, we will introduce a multi-mode area-efficient SCL (MM-SCL) polar decoder with variable list sizes and parallelism. When the error performance is of higher priority, the MM-SCL decoders is under a mode with a large list size to provide a good error performance. However, when the channel is good enough and the error performance with a smaller list size is acceptable, the MM-SCL decoder switches to a mode with this smaller list size to provide a higher throughput and a shorter decoding latency. Therefore, the MM-SCL decoder provides a flexible trade-off between latency, throughput and error performance and can easily adapt to different communication channels and applications.

1.2 Outline of the Dissertation

This dissertation is organized as follows:

In Chapter 2, two improvements to the majority-logic decoding algorithms are presented. Instead of the accumulation of reliability information in the existing majority-logic decoding algorithms, our first improvement is a new reliability information update. The new update not only results in better error performance and fewer iterations on average, but also further reduces computational complexity. Since existing majority-logic decoding algorithms tend to have a high error floor for codes whose parity check matrices have low column weights, our second improvement is a re-selection scheme, which leads to much lower error floors, at the expense of more finite field operations and integer operations, by identifying periodic points, re-selecting intermediate hard decisions, and changing reliability information.

In Chapter 3, symbol-decision successive cancellation (SC) and successive cancellation list (SCL) decoders for polar codes are presented, which use symbol-wise hard or soft decisions for higher throughput or better error performance. First, we propose to use a recursive channel combination to calculate symbol-wise channel transition probabilities, which lead to symbol decisions. Then we show that our proposed recursive channel combination has lower complexity than simply combining bit-wise channel transition probabilities. The similarity between our proposed method and Arıkan's channel transformations also helps to share hardware resources between calculating bit- and symbol-wise channel transition probabilities. Second, a two-stage list pruning network is proposed to provide a trade-off between the error performance and the complexity of the symbol-decision SCL decoder. Third, since memory is a significant part of SCL decoders, we propose a pre-computation memory-saving technique to reduce memory requirement of an SCL decoder. Finally, to evaluate the throughput advantage of our symbol-decision decoders, we design an architecture based on a semi-parallel successive cancellation list decoder. In this architecture, different symbol sizes, sorting implementations, and message scheduling schemes are considered.

In Chapter 4, an approximate ML (AML) decoding unit for SCL decoders is investigated. In particular, the distribution of frozen bits of polar codes designed for both the binary erasure and additive white Gaussian noise channels are investigated. By taking advantage of the distribution, the complexity of the AML decoding unit is reduced further. Moreover, to adapt to variable throughput and latency requirements which widely exist in practical applications, a multi-mode SCL decoder with variable list sizes and parallelisms is discussed. The multi-mode SCL decoder provides a flexible trade-off between throughput, latency and error performance.

Chapter 5 concludes this dissertation and discusses some future work regarding non-binary LDPC codes and polar codes.

Chapter 2

Improved MLGD Algorithms for Non-binary LDPC Codes

The main contributions of this chapter are two improvements to the majority-logic decoding algorithms for non-binary LDPC codes.

- The first improvement is a new reliability information update, instead of the accumulation of reliability information used in existing majority-logic decoding algorithms.
- Since existing majority-logic decoding algorithms tend to have a high error floor for codes whose parity check matrices have small column weights, our second improvement is a re-selection scheme, which lowers error floors at the expense of more finite field operations and integer operations by identifying periodic points, re-selecting intermediate hard decisions, and changing reliability information.

In the ISRB and IHRB algorithm, the reliability information includes all check-to-variable (c-to-v) messages of previous iterations. The new reliability information update proposed in this chapter excludes the c-to-v messages of previous iterations. It not only results in better error performance and fewer iterations on average, but also greatly reduces computational complexities of *all* existing majority-logic decoding algorithms. For instance, when applied to the ISRB majority-logic decoding algorithm, the new reliability information update results in a 0.15 dB coding gain and reduces required number of iterations by 10% at 4.7 dB for a $(16, 16)$-regular $(255, 175)$ cyclic LDPC code over GF(2^8) constructed with the method as describe in [46, Example 4]. Also, at a block error rate (BLER) of 10^{-4}, the coding gain over the EIHRB algorithm is about 0.07 dB. At the SNR of 4.7 dB, the average number of iterations is reduced by about 25%. Furthermore, with the new reliability information update, the improved algorithms require significantly fewer IAs and ICs than the ISRB and EIHRB algorithms. Finally, the existing majority-logic decoding algorithms are based on the accumulation of reliability information, and hence the numerical range of the reliability information increases with iterations. In contrast, the proposed reliability information update results in a fixed numerical range and thus simplifies hardware implementations. Our new reliability update has been presented in part in [47]. By applying both the layered scheduling and our first improvement to the IHRB algorithm, we proposed a layered improved IHRB decoder with a high throughput in [48]. Because the architecture design of non-binary LDPC decoders is beyond the scope of this chapter, we will not discuss the layered improved IHRB decoder henceforth.

In the literature, to analyze the error floor of binary LDPC codes, some notions based on graphical structures have been introduced, such as stopping sets [49], trapping sets [50] and absorbing sets [51]. Unfortunately, trying to lower the error floor based on graphical structures usually incurs very high complexity. Also, some approaches for binary LDPC codes cannot be readily adapted to non-binary ones. For instance, a selective biasing postprocessing algorithm is proposed in [51] to lower the error floors of binary LDPC codes based on the relaxed graphical structure of absorbing sets. However, for non-binary LDPC codes, trapping sets are difficult to identify because they involve not only the graph topology but also values of non-zero entries of parity-check matrices [52]. Moreover, the biasing rule between two elements for binary LDPC codes cannot be applied to non-binary codes directly, because there are more than two elements in a non-binary finite field.

In this chapter, for the majority-logic decoding algorithms, we propose a re-selection scheme based on periodic points to lower the error floors. The re-selection scheme is not a postprocessing algorithm and can be integrated into the regular iteration procedure easily. For instance, for an (837, 726) non-binary quasi-cyclic LDPC code over GF(2^5) constructed with the method in [13] with a column weight of four, the EIHRB algorithm has a BLER floor around 10^{-3}, while the hard-reliability based algorithm with the new reliability information update and the re-selection scheme achieves a BLER floor below 10^{-5}. Although this re-selection scheme requires additional computation, it is used only when existing majority-logic decoding algorithms have a high error floor.

The rest of our chapter is organized as follows. Section 2.1 reviews existing majority-logic decoding algorithms for non-binary LDPC codes. Section 2.2 proposes the two improvements. In Section 2.3.1, the two improvements are applied to existing majority-logic decoding algorithms to illustrate their advantages in error performance and average numbers of iterations. Section 2.3.2 discusses the reduction in the computational complexities due to the two improvements. Some conclusions are given in Section 2.4.

2.1 Existing Majority-Logic Decoding Algorithms

The basic idea of the ISRB and IHRB algorithms [20] is the majority-logic decoding. They calculate reliability information of variable nodes iteratively. Based on the reliability information, by the majority-logic decoding, hard-decisions are made in each iteration and are passed from variable nodes to check nodes. After getting these hard-decisions, check nodes provide votes for variable nodes. Here, a vote for a variable node provided by a check node is the value the variable node should be so that the check node is satisfied, assuming the other variable nodes adjacent to the check node take the values of hard decisions of the previous iteration. Then, based on the reliability information of the previous iteration and these votes, the reliability information of the current iteration is updated. Hence, the reliability information comes from the channel information and votes generated during iterations. The initial hard decisions come from the channel information. The difference between the ISRB and IHRB algorithms is that they use the soft and hard channel information, respectively, for the initial reliability information and vote information coefficients.

The EIHRB algorithm [23] was devised based on the IHRB algorithm by introducing a soft-reliability initialization. However, for vote information coefficients, the EIHRB algorithm still uses the hard channel information. Moreover, the EIHRB algorithm recalculate the extrinsic information to check nodes. Hence, if the soft-reliability information of the received symbol is available, the EIHRB algorithm achieves a better performance than the IHRB algorithm. Therefore, we focus on the EIHRB algorithm and do not consider the IHRB algorithm further.

For the ISRB, IHRB and EIHRB algorithm, the reliability information update is an accumulation operation. To perform these three algorithm correctly, the reliability information must be kept from numerical saturation based on two methods. One is to use a very large numerical range to store the reliability information. The other is to carry out a clipping operation for the reliability information. Furthermore, the accumulated reliability information update conflicts the extrinsic information principle for the iteration message-passing algorithm.

2.2 Two Improvements

A regular LDPC code $\mathcal{C}$ of length N over a finite field GF(2^r) is the null space of an $M \times N$ sparse parity check matrix $\mathbf{H}$ over GF(2^r). $\mathbf{H}$ has constant column and row weights of γ and ρ, respectively. Let $\mathbf{h}_0, \mathbf{h}_1, \cdots, \mathbf{h}_{M-1}$ denote the rows of $\mathbf{H}$, where $\mathbf{h}_i = (h_{i,0}, h_{i,1}, \cdots, h_{i,N-1})$ for $0 \leq i < M$. Let $(a_{l,0}, a_{l,1}, \cdots, a_{l,r-1})$ be the binary representation of $a_l \in$ GF(2^r), for $0 \leq l < 2^r$. Suppose a codeword $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})$ is transmitted. Since $x_i \in$ GF(2^r) can be represented by an r-tuple $(x_{i,0}, x_{i,1}, \cdots, x_{i,r-1})$ over GF(2) for $0 \leq i < N$, an Nr-tuple over GF(2) is transmitted for each codeword. Let $\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})$ represent the received word, and $\mathbf{z} = (z_0, z_1, \cdots, z_{N-1})$ and $\mathbf{q} = (q_0, q_1, \cdots, q_{N-1})$ represent the hard decision and quantization, respectively, of the received word. Let $\mathcal{N}(i) = \{j : h_{i,j} \neq 0, 0 \leq j < N\}$ for $0 \leq i < M$ and $\mathcal{M}(j) = \{i : h_{i,j} \neq 0, 0 \leq i < M\}$ for $0 \leq j < N$. $I_{\max}$ represents the maximal iteration number. Throughout this chapter, the superscript (k) is added to denote variables in the k-th iteration. $\mathbf{s}^{(k)}$ is the syndrome vector corresponding to $\mathbf{z}^{(k)}$. $\sigma_{i,j}^{(k)}$ and $\phi_{i,j}$ are the vote and vote information coefficient, respectively, from check node i to variable node j. $\varphi_{j,l}$, $\psi_{j,l}^{(k)}$, and $R_{j,l}^{(k)}$ are the channel reliability, extrinsic reliability and overall reliability information, respectively, of the j-th received symbol being a_l.

2.2.1 New Reliability Information Update

The reliability information update of the ISRB algorithm can be written as:

$$\begin{aligned} R_{j,l}^{(k+1)} &= R_{j,l}^{(k)} + \psi_{j,l}^{(k)} \\ &= R_{j,l}^{(0)} + \sum_{t=0}^{k} \psi_{j,l}^{(t)} \end{aligned} \tag{2.1}$$

$R_{j,l}^{(k)}$ includes all check-to-variable (c-to-v) messages of previous iterations. This conflicts with the extrinsic information principle.

We propose a new reliability information update to exclude the c-to-v messages of previous iterations. The new reliability information update, shown as Eq. (2.2), uses only the channel information $\varphi_{j,l}$ and $\psi_{j,l}^{(k)}$ of the current iteration to compute the reliability information $R_{j,l}^{(k+1)}$:

$$R_{j,l}^{(k+1)} = \xi_1 \varphi_{j,l} + \xi_2 \psi_{j,l}^{(k)}, \tag{2.2}$$

where ξ_1 and ξ_2 are two parameters to improve the error performance. Consequently the new algorithm, presented in Alg. 1, is called the IISRB algorithm.

To reduce complexity of the IISRB algorithm, we calculate $\xi_1 \varphi_{i,j}$ and $\xi_2 \phi_{i,j}$ in the initialization. This helps to reduce the complexity of each iteration.

By applying the new reliability information update to the EIHRB algorithm, we have

Algorithm 1: IISRB algorithm

```
/* ------Initialization------- */
1  for $j = 0 : (N-1)$ do
2      $z_j^{(0)} = z_j$;
3      for $l = 0 : (2^r - 1)$ do
4          $\varphi'_{j,l} = \sum_{t=0}^{r-1} (1 - 2a_{l,t}) q_{j,t}$;
5          $\varphi_{j,l} = \xi_1 \varphi'_{j,l}$;
6  for $i = 0 : (M-1)$ do
7      for $j \in \mathcal{N}(i)$ do
8          $\phi_{i,j} = \xi_2 \min_{t \in \mathcal{N}(i) \setminus \{j\}} \max_l \varphi'_{t,l}$;
/* -------Iteration-------- */
9  for $k = 0 : I_{\max}$ do
10     $\mathbf{s}^{(k)} = \mathbf{H} \cdot (\mathbf{z}^{(k)})^T$;
11     if $\mathbf{s}^{(k)} == 0$ then return $\mathbf{z}^{(k)}$ else if $k == I_{\max}$ then return *Failure* else
12         for $j = 0 : (N-1)$ do
13             for $l = 0 : (2^r - 1)$ do
14                 $\psi_{j,l}^{(k)} = 0$;
15             for $i \in \mathcal{M}(j)$ do
16                 $\sigma_{i,j}^{(k)} = h_{i,j}^{-1} \sum_{t \in \mathcal{N}(i) \setminus \{j\}} h_{i,t} z_t^{(k)}$;
17                 for $l = 0 : (2^r - 1)$ do
18                     if $\sigma_{i,j}^{(k)} == a_l$ then $\psi_{j,l}^{(k)} = \psi_{j,l}^{(k)} + \phi_{i,j}$
19     for $j = 0 : (N-1)$ do
20         for $l = 0 : (2^r - 1)$ do
21             $R_{j,l}^{(k+1)} = \varphi_{j,l} + \psi_{j,l}^{(k)}$;
22         $z_j^{(k+1)} = \arg_{a_l} \max R_{j,l}^{(k+1)}$;
```

the IEIHRB algorithm. For the sake of conciseness, we only present the initialization of the IEIHRB algorithm in Alg. 2, where c_1, c_2 and c_3 are parameters to improve the error performance, because the iteration part of the IEIHRB algorithm is the same as that of the IISRB algorithm. For the IISRB algorithm, $\phi_{i,j}$ is from the channel information. However, for the IEIHRB algorithm, $\phi_{i,j}$ is a constant.

Algorithm 2: Initialization of the IEIHRB algorithm

```
1 for $j = 0 : (N-1)$ do
2     $z_j^{(0)} = z_j$;
3     for $l = 0 : (2^r - 1)$ do
4         $\varphi_{j,l} = \sum_{t=0}^{r-1}(1 - 2a_{l,t})q_{j,t}$;
5         $R_{j,l}^{(0)} = \max(\lfloor \varphi_{j,l}/c_1 \rfloor + c_2 - \max_l(\lfloor \varphi_{j,l}/c_1 \rfloor), 0)$;
6 for $i = 0 : (M-1)$ do
7     for $j \in \mathcal{N}(i)$ do
8         $\phi_{i,j} = c_3$;
```

2.2.2 Re-selection Scheme

Furthermore, we observe that the error floor of the ISRB algorithm becomes higher, as the column weight of the parity check matrix decreases. The IISRB algorithm suffers the same problem. Unless otherwise specified, the BPSK modulation, the additive white Gaussian noise (AWGN) channel with a single-sided power spectral density N_0, a 6-bit uniform quantization with 64 levels and an interval length $\Delta = 0.0625$, and $I_{\max} = 50$ are used in our following numerical simulations.

In Fig. 2.1, C1 is an (837, 726) LDPC code over GF(2^5) with a column weight of four, C2 an (806, 680) LDPC code over GF(2^5) with a column weight of five, C3 a (775, 634) LDPC code over GF(2^5) with a column weight of six. All three codes are constructed based on Reed–Solomen codes with two information symbols [13]. The error floor of BLER performance becomes lower as the column weight of the parity check matrix increases. Hence, the column weight of the parity check matrix is one key factor for the error floor.

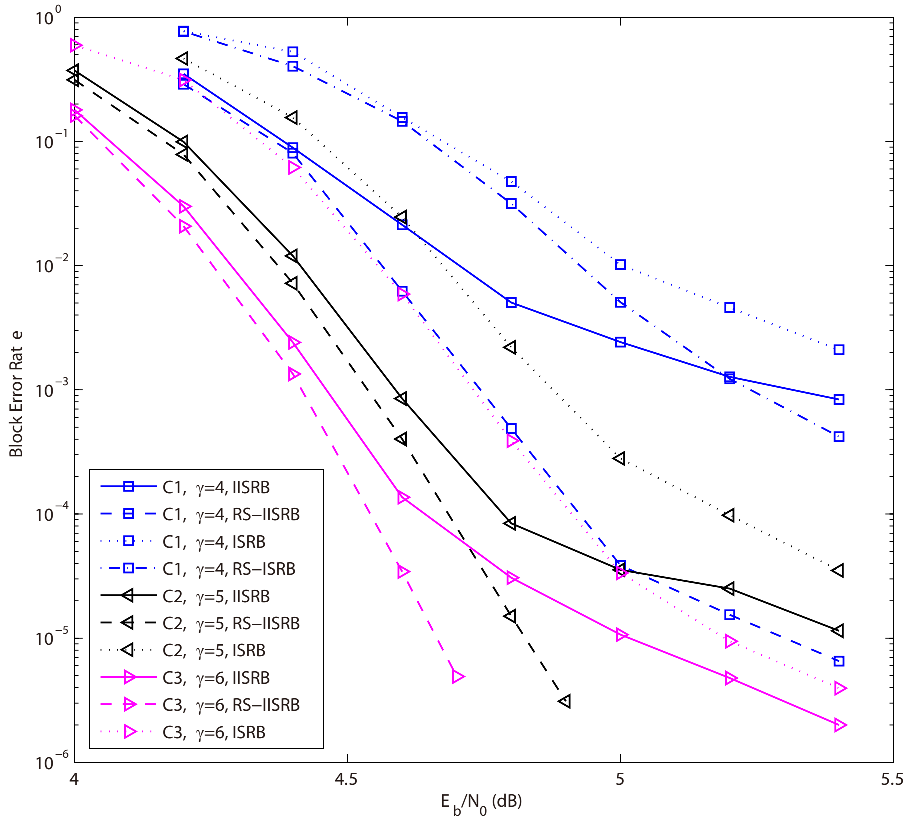


Figure 2.1: BLERs of the soft-reliability based algorithms for different codes with different column weights

We propose a re-selection scheme to address this problem. To simplify the discussion, here we focus on the IISRB algorithm. Our simulation results show that the re-selection scheme also applies to the ISRB, EIHRB and IEIHRB algorithms.

To analyze the error floor, the concept of periodic points is introduced. Given an endomorphism $f : Z \to Z$, a point $\mathbf{z}$ in Z is called a *periodic point with a period of* i if there exists a smallest positive integer i so that $f^{(i)}(\mathbf{z}) = \mathbf{z}$, where $f^{(i)} = f(f^{(i-1)}(\mathbf{z}))$.

An iteration of the IISRB algorithm can be considered a function f. The k-th iteration of the IISRB algorithm is $\mathbf{z}^{(k)} = f(\mathbf{z}^{(k-1)}) = f^{(2)}(\mathbf{z}^{(k-2)}) = \cdots = f^{(k)}(\mathbf{z}^{(0)})$, and if $\mathbf{s}^{(k)} \neq 0$ and $\mathbf{z}^{(k)} = \mathbf{z}^{(k-i)}$ for $0 < i \leq k$, the decoding algorithm results in a periodic point with a period of i. Our algorithm focuses on only the periodic points with a period of up to two for two reasons. First, our simulation results show that the BLERs are caused mainly by periodic points with periods one and two. Second, to identify the existence of a periodic point with a period of greater than two needs more memory to keep track of the hard decisions of the previous iterations.

If the Hamming distance between a periodic point and its corresponding transmitted codeword is less than θ, the periodic point with a period of up to two is called a small-distance periodic point (SDPP). Otherwise, the periodic point with a period of up to two is called a large-distance periodic point (LDPP). The solid curves of Fig. 2.2 show the BLER of the IISRB algorithm and the BLER caused by SDPPs or LDPPs for the (837, 726) code when $\theta = 8$. For low SNRs, the overall BLER is dominated by those caused by LDPPs. The sum of the BLERs due to SDPPs and LDPPs is less than the overall BLER, because periodic points with a period greater than two also cause some BLERs. For high SNRs, the BLER caused by SDPPs dominates the total BLER. A similar trend for the ISRB algorithm was observed as well. Hence, for the ISRB and IISRB algorithms, the error floor is mainly caused by SDPPs.

Consider the hard decision process of line 22 of Alg. 1. If the most likely decision is wrong, the second most likely decision is supposed to be the best choice to be decoded. The smaller the difference between the maximal reliability information and the second

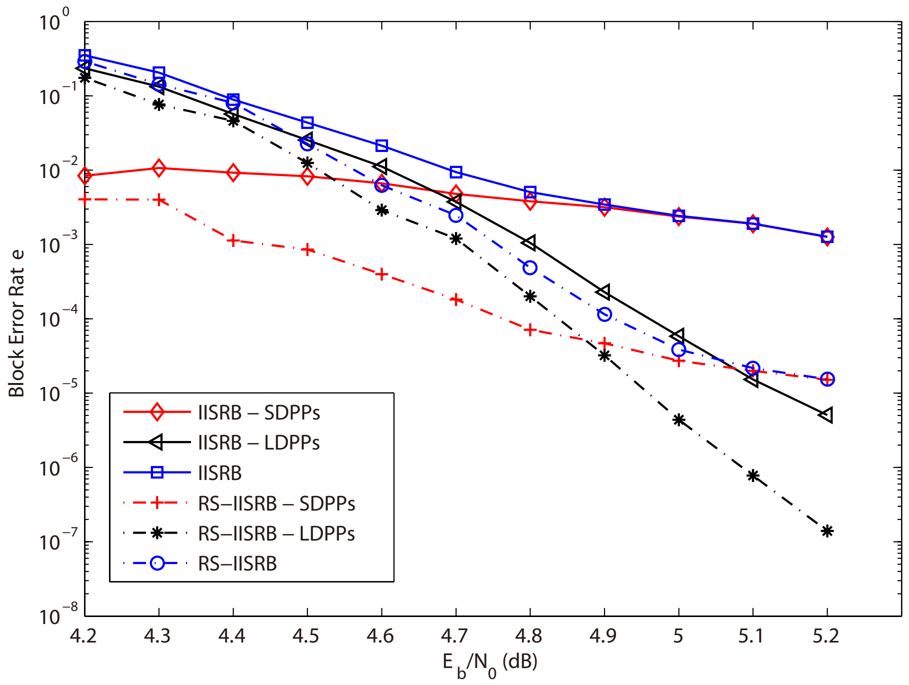


Figure 2.2: BLERs of SDPPs and LDPPs for the IISRB and RS-IISRB algorithm to decode the (837, 726) code

maximal reliability information, the greater the probability that the most likely decision is wrong.

Based on this intuition, when a periodic point is detected, the re-selection scheme tries to help the decoder get away from the periodic point by using the second most likely decision. The re-selection scheme consists of two steps. The first step is to identify the existence of a periodic point when the syndrome vector is a non-zero vector. The second step is to identify positions of erroneous symbols. A set is defined to include variable nodes adjacent to unsatisfied check nodes. This set contains some erroneous symbols. Then, among the variable nodes belonging to the set, the position of a variable node which has the smallest difference between its maximal reliability information and second maximal reliability information can be identified. If there are multiple variable nodes having the smallest difference, the first one is selected. Assume the index of this position is $\mathrm{rs\_n}$. Let $\mathrm{us\_c}_j$ represent the number of unsatisfied check nodes connected with the j-th variable node for $0 \le j < N$. The most likely decision $z^{(k)}_{\mathrm{rs\_n}}$ is replaced by the second most likely decision $\tilde{z}^{(k)}_{\mathrm{rs\_n}}$. Meanwhile, $\varphi_{\mathrm{rs\_n}, z^{(k)}_{\mathrm{rs\_n}}}$ is reduced by a preset offset ζ and $\varphi_{\mathrm{rs\_n}, \tilde{z}^{(k)}_{\mathrm{rs\_n}}}$ is added by the same preset offset. The detailed re-selection scheme is described in Alg. 3. Here, $s^{(k)}_i$ is the i-th value of the syndrome vector $\mathrm{s}^{(k)}$.

Algorithm 3: Re-selection scheme

```
1  for $j = 0 : (N-1)$ do
2      $\tilde{z}^{(k)}_j = \arg_{a_l \in GF(2^r) \setminus \{z^{(k)}_j\}} \max R^{(k)}_{j,l}$;
3  if $(\mathbf{z}^{(k-1)} == \mathbf{z}^{(k)})$ or $(\mathbf{z}^{(k-2)} == \mathbf{z}^{(k)})$ then
4      $\mathrm{dif\_R} = R^{(k)}_{0, z^{(k)}_0}$;
5  for $j = 0 : (N-1)$ do
6      $\mathrm{us\_c}_j = 0$;
7      for $i \in \mathcal{M}(j)$ do
8          if $(s^{(k)}_i > 0)$ then $\mathrm{us\_c}_j + +$
9      if $(\mathrm{us\_c}_j > 0)$ and $((R^{(k)}_{j, z^{(k)}_j} - R^{(k)}_{j, \tilde{z}^{(k)}_j}) < \mathrm{dif\_R})$ then
10         $\mathrm{dif\_R} = (R^{(k)}_{j, z^{(k)}_j} - R^{(k)}_{j, \tilde{z}^{(k)}_j})$;
11         $\mathrm{rs\_n} = j$;
12 
13 $\varphi_{\mathrm{rs\_n}, z^{(k)}_{\mathrm{rs\_n}}} = \varphi_{\mathrm{rs\_n}, z^{(k)}_{\mathrm{rs\_n}}} - \zeta$;
14 $\varphi_{\mathrm{rs\_n}, \tilde{z}^{(k)}_{\mathrm{rs\_n}}} = \varphi_{\mathrm{rs\_n}, \tilde{z}^{(k)}_{\mathrm{rs\_n}}} + \zeta$;
15 $z^{(k)}_{\mathrm{rs\_n}} = \tilde{z}^{(k)}_{\mathrm{rs\_n}}$;
16 for $i \in \mathcal{M}(\mathrm{rs\_n})$ do
17     $s^{(k)}_i = \mathbf{h}_i \cdot (\mathbf{z}^{(k)})^T$;
```

This scheme can be applied to any majority-logic decoding algorithms. For the IISRB algorithm, this scheme is added between lines 11 and 12. Similarly, the re-selection scheme can be inserted at the corresponding position of other algorithms. "RS-" is prefixed in front of the name of the algorithms to show that an algorithm adopts the re-selection scheme. For instance, the ISRB algorithm with the re-selection scheme is called as the RS-ISRB algorithm.

The dash-dot lines of Fig. 2.2 show the BLERs of the RS-IISRB algorithm and those caused by SDPPs and LDPPs. Compared with the IISRB algorithm, the BLER caused by LDPPs descends to 2×10^{-4} from 1.2×10^{-3}, and the BLER caused by SDPPs is reduced to 7×10^{-5} from 4×10^{-3} when SNR is 4.8 dB. Hence, the rs-selection scheme reduces the occurrences of both SDPPs and LDPPs and works better on the former. Even for the RS-IISRB algorithm, SDPPs still are the primary reason for the error floor.

The re-selection scheme helps the decoding algorithm correct some periodic points. It is likely that the decoding algorithm goes out of a periodic point temporarily, and goes back to the same periodic point or results in another periodic point. Therefore, even with the re-selection scheme, the decoding algorithm still encounters the error floor problem. Moreover, the re-selection scheme works better on SDPPs because in general a SDPP involves fewer unsatisfied check nodes than a LDPP.

2.3 Performance Evaluation

2.3.1 Error Performance and Average Numbers of Iterations

Our simulations focus on C1, C2, and C3, whose parity check matrices have small column weights, as well as a (255,175) cyclic LDPC code over GF(2^8) constructed with the method as describe in [46, Example 4], because it has a large column weight of 16.

We first compare the performance of the soft-reliability based algorithms. The ISRB, IISRB and RS-IISRB algorithms are used to decode the (255, 175) code. For the ISRB algorithm, different values of $\lambda = 4l$ for $l = 1, 2, \cdots, 8$, were tried, and $\lambda = 16$ leads to the best performance. For the new reliability information update, different combinations of ξ_1 and ξ_2 were tested. Since they are weighting factors, we fix $\xi_2 = 1$ and try different values for ξ_1. For the (255, 175) code, $\xi_1 = 4, 5, 6, 7, 8, 9, 10$ were tried and ($\xi_1 = 7$, $\xi_2 = 1$) results in the best error performance[1]. The real values from 6.2 to 7 with a step size of 0.2 for ξ_1 and $\xi_2 = 1$ were tested as well. Performance differences between different real value coefficients are very small. Henceforth, integer values are used for ξ_1 and ξ_2 to reduce complexity.

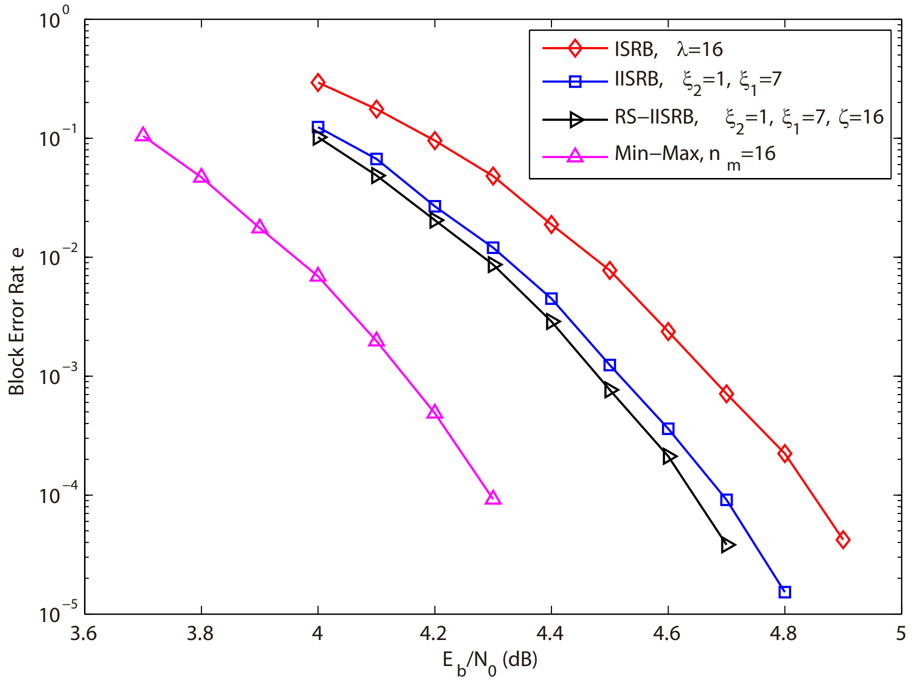


Figure 2.3: BLERs of the ISRB, IISRB and RS-IISRB algorithms for the (255, 175) code

The BLER curves of the ISRB, IISRB, RS-IISRB and Min-Max algorithms for the (255,175) code are shown in Fig. 2.3. The IISRB algorithm has a 0.15 dB coding gain versus the ISRB algorithm in this case. The RS-IISRB algorithm also achieves a slight

[1]The comparison of these parameters is omitted due to the limited space, and can be found in an extended version of this manuscript [53].

improvement compared to the IISRB algorithm and has a performance loss of about 0.4 dB versus the Min-Max algorithm.

If a total of T iterations is used to decode K received words, the average number of iterations per received word is T/K. The average numbers of iterations per received word for the soft-reliability algorithms are compared in Table 2.1, where K is chosen such that at least 100 erroneous decoded words are observed for each SNR. Table 2.1 shows that both the RS-IISRB and IISRB algorithms require fewer iterations than the ISRB algorithm. At 4.7 dB, the average number of iterations of the IISRB algorithm is fewer by 10% than that of the ISRB algorithm. The advantage of the IISRB and RS-IISRB algorithm is even more pronounced for low SNRs.

Table 2.1: Average number of iterations of the Min-Max ($n_m = 16$), ISRB, IISRB and RS-IISRB algorithms for the (255, 175) code

E_b/N_0(dB)	Min-Max [19]	ISRB [20]	IISRB	RS-IISRB
4.0	2.35	18.76	11.58	11.25
4.1	1.91	13.25	8.17	7.85
4.2	1.60	9.10	5.78	5.71
4.3	1.36	6.46	4.46	4.41
4.4	N/A	4.59	3.59	3.59
4.5	N/A	3.68	3.06	3.05
4.6	N/A	3.10	2.71	2.70
4.7	N/A	2.74	2.46	2.46
4.8	N/A	2.50	2.27	2.28

The ISRB, IISRB and RS-IISRB algorithms are also used to decode the (837, 726) code. The best BLER performance of the IISRB algorithm is achieved when $\xi_1 = 4$ and $\xi_2 = 1$. Fig. 2.4 compares the BLERs of the ISRB, IISRB, RS-IISRB and Min-Max algorithms for this code. The IISRB algorithm has a 0.2 dB coding gain versus the ISRB algorithm, but both algorithms show an error floor around 10^{-3}. Compared with these two algorithms, for low SNRs the RS-IISRB algorithm shows a slight improvement, and for high SNRs the RS-IISRB algorithm lowers the error floor to below 10^{-5} and has a performance loss of only 0.6 dB versus the Min-Max algorithm at the BLER of 10^{-3}.

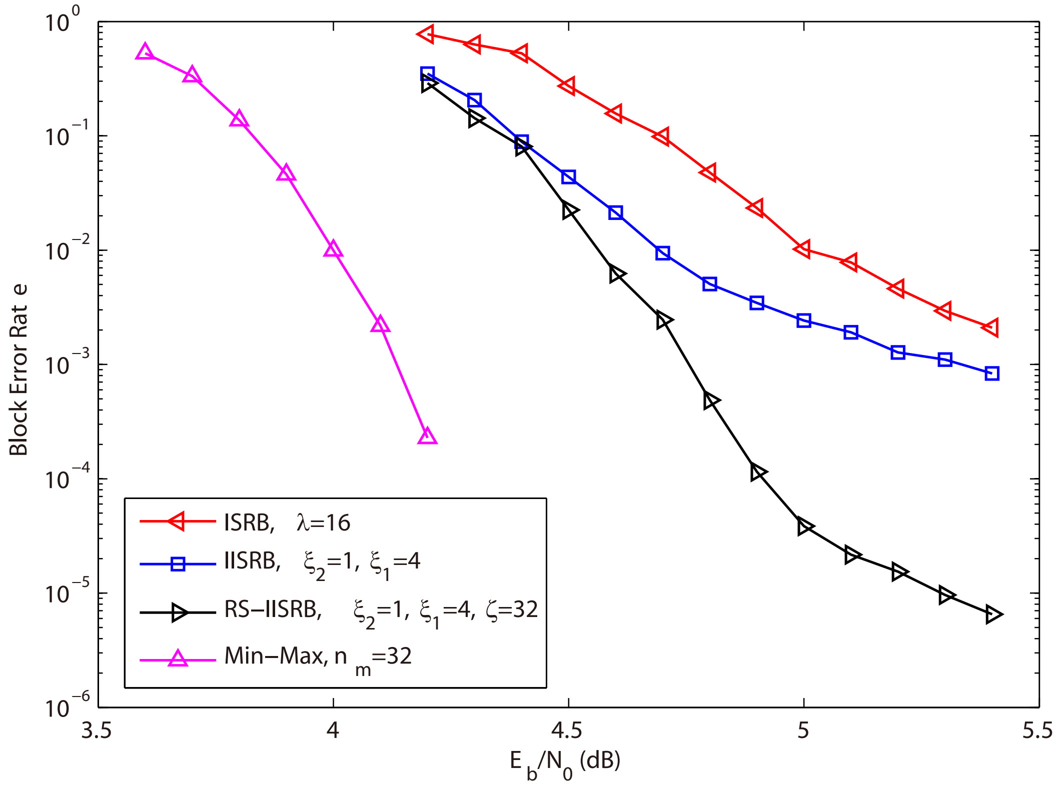


Figure 2.4: BLERs of the ISRB, IISRB and RS-IISRB algorithms for the (837, 726) code

The average numbers of iterations for the (837,726) code with different SNRs are listed in Table 2.2. The average numbers of iterations of the IISRB and RS-IISRB algorithms are fewer by at least 20% than that of the ISRB algorithm. The number of iterations of the RS-IISRB algorithm is slightly fewer than those of the IISRB algorithm because of the re-selection scheme. In addition, we compare the running time of different decoding algorithms (implemented in C) on a DELL Optiplex 755. To decode 10,000 codewords of the (837,726) code over the AWGN channel at the SNR of 5.4 dB, the ISRB, IISRB and RS-IISRB algorithms run 22.22, 19.48 and 19.37 seconds, respectively. The running time is consistent with the comparison based on the average number of iterations.

Table 2.2: Average number of iterations of the ISRB, IISRB and RS-IISRB algorithms for the (837, 726) code

E_b/N_0 (dB)	ISRB [20]	IISRB	RS-IISRB
4.5	22.18	10.58	9.97
4.6	16.52	8.22	7.62
4.7	12.69	6.54	6.30
4.8	9.40	5.49	5.33
4.9	7.44	4.79	4.64
5.0	5.99	4.20	4.12
5.1	5.21	3.80	3.71
5.2	4.53	3.44	3.38
5.3	4.06	3.15	3.10
5.4	3.66	2.90	2.87

For C1, C2 and C3, the BLERs of the RS-IISRB algorithm are shown with the dashed curves in Fig. 2.1. For C1 and C2, $\xi_1 = 4$, $\xi_2 = 1$, $\lambda = 16$, $\zeta = 32$. For C3, $\xi_1 = 5$, $\xi_2 = 1$, $\lambda = 16$, $\zeta = 32$. The RS-IISRB algorithm improves the BLER performance and lowers the error floor for all three codes. In Fig. 2.1, for C1, the simulation result for the RS-ISRB algorithm is shown as well, which does not adopt the new reliability information update but the re-selection scheme. It appears that the re-selection scheme also provides some performance gain. If both improvements are applied, the RS-IISRB algorithm achieves a greater performance gain.

We evaluate the proposed soft-reliability based algorithms over block fading channels, which are widely used in wireless communication systems involving slow time-frequency hopping or multi-carrier modulation using orthogonal frequency division multiplexing technique. We assume that each codeword experiences a block Rayleigh fading channel and that the receiver has perfect channel state information. For the (837,726) code over a block Rayleigh fading channel, the IISRB and RS-IISRB algorithms have a gain of about 0.2 dB over the ISRB algorithm, which is similar to that over the AWGN channel. The proposed improvements reduce the average number of iterations by about 5%[2].

Next, we compare the performances of hard-reliability based MLGD algorithms. The EIHRB-INIT algorithm [23] is a simplified version of the EIHRB algorithm without the recalculation of the extrinsic information. The RS-IEIHRB algorithm is developed by integrating the re-selection scheme describe in Section 2.2.2 into the IEIHRB algorithm.

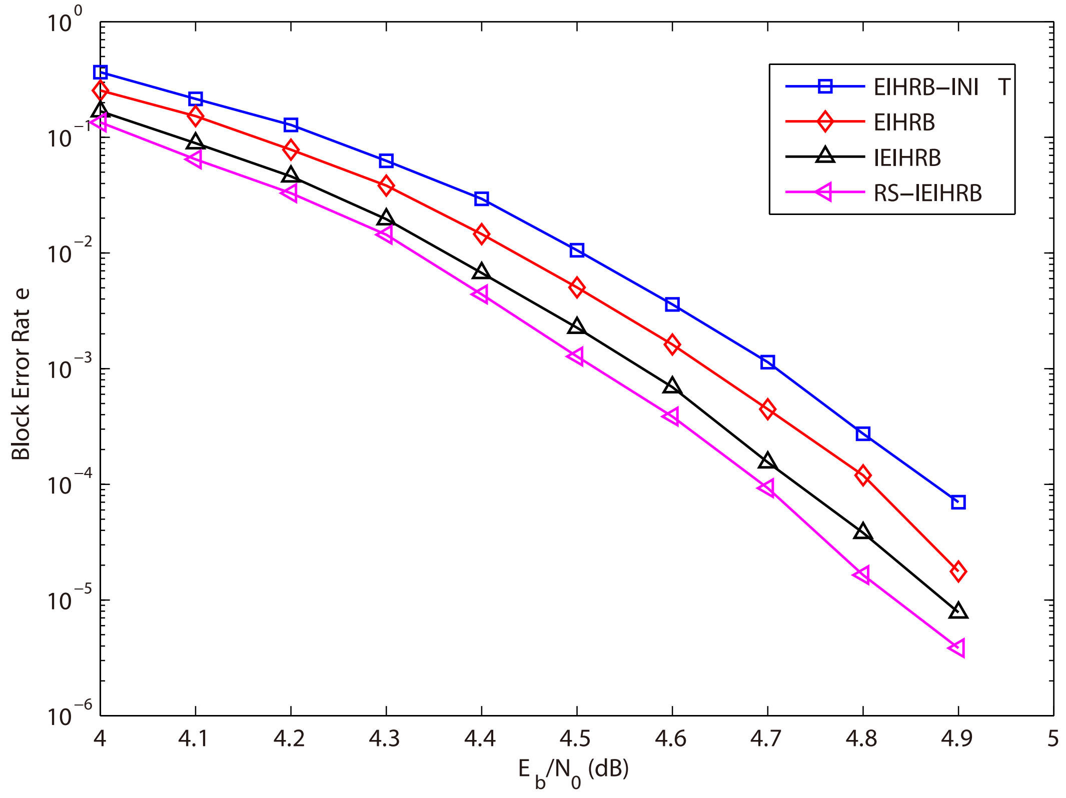


Figure 2.5: BLERs of hard-reliability based algorithms for the (255,175) code

Fig. 2.5 shows the BLERs of different hard-reliability based algorithms for the

[2]More detail can be found in [53].

(255,175) code, and Table 2.3 lists the average numbers of iterations. For the EIHRB-INIT and EIHRB algorithm, $c_1 = 4$ and $c_2 = 15$. For the IEIHRB and RS-IEIHRB algorithm, $c_1 = 1$, $c_2 = 63$, $c_3 = 12$ and $\zeta = 16$. For the (255,175) code the new reliability information update provides about 0.05 dB performance gain, and the re-selection scheme provides another 0.05 dB performance gain. Hence, compared with the EIHRB algorithm, the RS-IEIHRB algorithm has about 0.1 dB performance gain, and the average number of iterations required by the RS-IEIHRB algorithm is reduced by about 30%.

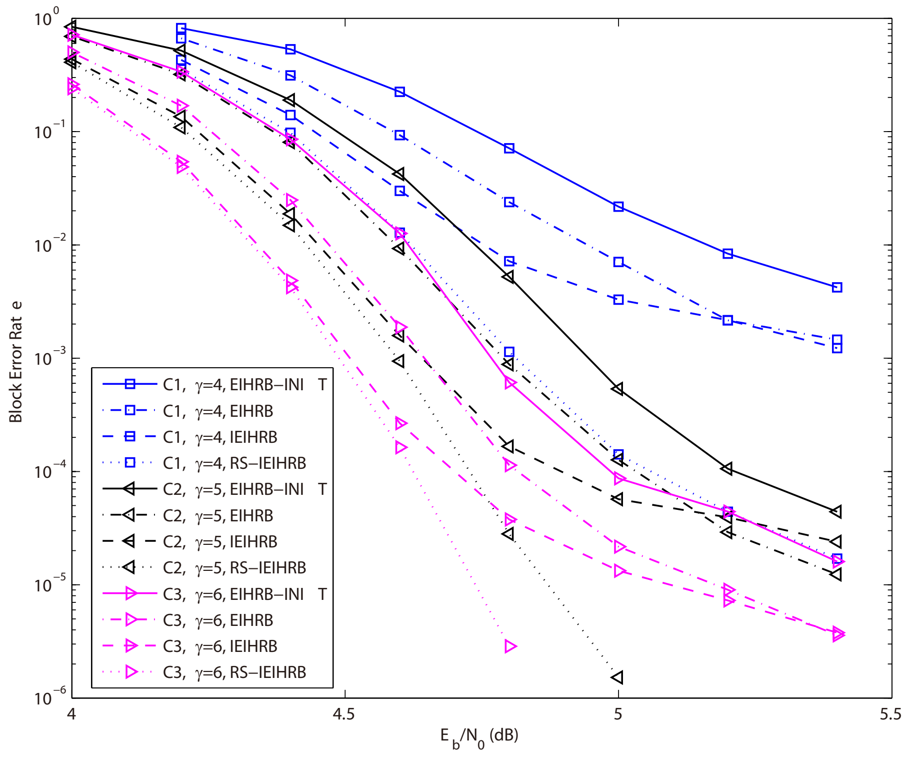


Figure 2.6: BLERs of hard-reliability based algorithms for different non-binary LDPC codes with different column weights

Fig. 2.6 compares the BLERs of hard-reliability based algorithms for different non-binary LDPC codes with different column weights. For C1 and C2, $c_1 = 10$, $c_2 = 63$,

Table 2.3: Average number of iterations of the hard-reliability based algorithms for the (255, 175) code

E_b/N_0 (dB)	EIHRB-INIT [23]	EIHRB [23]	IEIHRB	RS-IEIHRB
4.0	22.28	18.56	12.67	12.09
4.1	16.48	13.62	8.99	8.85
4.2	12.21	9.54	6.42	6.09
4.3	8.57	7.20	4.70	4.62
4.4	6.44	5.57	3.70	3.69
4.5	5.15	4.70	3.10	3.10
4.6	4.29	4.07	2.73	2.72
4.7	3.75	3.65	2.47	2.47
4.8	3.38	3.33	2.28	2.28
4.9	3.09	3.06	2.14	2.14

$c_3 = 2$ and $\zeta = 32$. For C3, $c_1 = 11$, $c_2 = 63$, $c_3 = 2$ and $\zeta = 32$. For the (837,726) code, the EIHRB algorithm also has an error floor of 10^{-3}. For low SNRs, the IEIHRB algorithm outperforms the EIHRB algorithm and the RS-IEIHRB algorithm reduces the error floor to a level of 10^{-5}. In the error floor region, the EIHRB algorithm is better than the IEIHRB algorithm because of the use of $z_{i,t}^{(k)}$ and recalculating the extrinsic information in the latter. The two improvements in Section 2.2 also help to reduce the average number of iterations by at least 20% for the (837,726) code as listed in Table 2.4. For C2 and C3, the new reliability information update provides some performance gains for low SNRs, and the error floors are lowered effectively.

In summary, the two improvements introduced in Section 2.2 apply to both the soft-reliability and hard-reliability based MLGD algorithms. While both improvements improve the error performance and require fewer iterations on average, the re-selection scheme lowers the error floor of codes with low column weights effectively.

Table 2.4: Average number of iterations of the hard-reliability based algorithms algorithm for the (837, 726) code

E_b/N_0 (dB)	EIHRB-INIT [23]	EIHRB [23]	IEIHRB	RS-IEIHRB
4.2	43.01	37.25	28.83	27.49
4.4	32.19	23.09	15.52	14.55
4.6	18.85	12.41	8.29	7.83
4.8	10.57	7.60	5.45	5.22
5.0	6.79	5.49	4.13	4.01
5.2	4.98	4.37	3.36	3.27

2.3.2 Computational Complexity Reduction

We evaluate impacts on the complexity by the two proposed improvements and focus on the soft-reliability based MLGD algorithms first. Assume the quantized input information $q_{j,t}$ has a bit width of ω. Without the clipping operation, for the ISRB algorithm, $R_{j,l}^{(k)}$ needs $\omega + \lceil \log_2((\lambda + I_{\max}\gamma)r) \rceil$ bits and its bit width increases as $I_{\max}$ grows. However, for the IISRB algorithm, $R_{j,l}^{(k)}$ needs only $\omega + \lceil \log_2((\xi_1 + \xi_2\gamma)r) \rceil$ bits and I_{max} has no impact on $R_{j,l}^{(k)}$'s bit width. With the clipping operation, $R_{j,l}^{(k)}$ needs a smaller bit width in the ISRB algorithm. However, $N2^r$ IAs and $N(2^r - 1)$ ICs are needed per iteration to carry out the clipping operation. In contrast, there is no accumulation operation in the IISRB and RS-IISRB algorithms. Thus, saturation is not an issue for the IISRB and RS-IISRB algorithms, and the clipping operation is not needed.

For the IISRB algorithm, let us consider the initialization step first. There are $N2^r$ $\varphi_{j,l}$'s. To compute $\varphi_{j,l}$'s needs $Nr2^r$ IAs. Because $\max_l \varphi_{t,l} = \varphi_{t,z_t}$ and there are $N\gamma$ $\phi_{i,j}$'s, $N\gamma(\rho - 2)$ ICs are needed to calculate $\phi_{i,j}$'s. The calculations of $\xi_1\varphi_{j,l}$'s and $\xi_2\phi_{i,j}$'s need $N2^r$ and $N\gamma\rho$ IMs, respectively. Therefore, the initialization step needs $Nr2^r$ IAs, $N2^r + N\gamma\rho$ IMs and $N\gamma(\rho - 2)$ ICs.

We now analyze the complexity per iteration of the IISRB algorithm. Each iteration needs $M\rho$ FMs and $M(\rho-1)$ FAs to calculate the syndrome $\mathrm{s}^{(k)}$. Line 16 in Alg. 1 can be reformulated as:

$$\sigma_{i,j}^{(k)} = h_{i,j}^{-1} s_i^{(k)} + z_j^{(k)} \tag{2.3}$$

Hence, $N\gamma$ FAs and $N\gamma$ FMs are needed to calculate $\sigma_{i,j}^{(k)}$'s. Assume there are $u_j^{(k)}$ $(0 < u_j^{(k)} \leq \gamma)$ different values among $\sigma_{i,j}^{(k)}$'s for each j, then $u_j^{(k)}$ $R_{j,l}^{(k+1)}$'s need to be updated. To compute $\psi_{j,l}^{(k)}$'s and $R_{j,l}^{(k+1)}$'s, $\gamma - u_j^{(k)}$ and $u_j^{(k)}$ IAs are needed, respectively, for each j. For $z_j^{(k+1)}$, $R_{j,z_j^{(k+1)}}^{(k+1)}$ must be one of $R_{j,z_j}^{(k+1)}$ and those $R_{j,l}^{(k+1)}$ updated in the k-th iteration. To make the hard decisions, $N\gamma$ ICs are needed at most. Hence, in the worst case, each iteration of the IISRB algorithm requires $2N\gamma$ FMs, $2N\gamma - M$ FAs, $N\gamma$ IAs and $N\gamma$ ICs $(M\rho = N\gamma)$. Compared with the ISRB algorithm, the IISRB algorithm saves $N2^r$ IAs and $N(2^{r+1} - 2 - \gamma)$ ICs for each iteration, while requiring the same numbers of FAs and FMs. This saving is significant if 2^r is large.

We now calculate the computational complexity overhead due to the re-selection scheme. $\tilde{\mathbf{z}} = (\tilde{z}_0, \tilde{z}_1, \cdots, \tilde{z}_{N-1})$ represents the second most likely decision of the received word $\mathbf{y}$. To acquire $\tilde{z}_j$ in the initialization step, $r-1$ ICs are needed for each j, because r-bit representations of $\tilde{z}_j$ and z_j differ by one bit and there are r elements over GF(2^r) satisfying this constraint. Hence, the initialization step of the RS-IISRB algorithm needs $N(r-1)$ ICs more than that of the IISRB algorithm. For each iteration, the second maximum among $R_{j,l}^{(k)}$'s must be one of $R_{j,\tilde{z}_j}^{(k)}$, $R_{j,z_j}^{(k)}$ and those $R_{j,l}^{(k)}$'s updated. It needs at most $N(\gamma+1)$ ICs per iteration. Line 3 of Alg. 3 needs $2N$ ICs to identify the existence of a periodic point. $N\gamma$ IAs and $N\gamma$ ICs are needed to calculate us_c$_j$. The calculation of dif_R needs $2N$ ICs and N IAs. $\varphi_{\mathrm{rs\_n}, z_{\mathrm{rs\_n}}^{(k)}}$ and $\varphi_{\mathrm{rs\_n}, \tilde{z}_{\mathrm{rs\_n}}^{(k)}}$ need two IAs. After the re-selection scheme, there are γ syndromes to be recalculated so that Eq. (2.3) can be applied, requiring 2γ FAs and γ FMs. Therefore, the RS-IISRB algorithm needs 2γ FAs, γ FMs, $N\gamma + N + 2$ IAs and $5N + 2N\gamma$ ICs per iteration more than the IISRB algorithm.

Table 2.5: Computational complexities in initialization for various decoding algorithms, where $V = N2^r$ and $W = N\gamma$

	Operations need by initialization				
Algorithm	IA	IM	IC	ID	Floor
ISRB [20]	rV	V	$MN(2^r-1)(3\rho-6)$	0	0
IISRB	rV	$V+W\rho$	$W(\rho-2)$	0	0
RS-IISRB	rV	$V+W\rho$	$N(\gamma(\rho-2)+(r-1))$	0	0
EIHRB [23]	$V(r+2)$	0	V	V	V
IEIHRB	$V(r+2)$	0	V	V	V
RS-IEIHRB	$V(r+2)$	0	$V+N(r-1)$	V	V

Table 2.6: Computational complexities per iteration for various decoding algorithms, where $V = N2^r$ and $W = N\gamma$

	Operations need by per iteration			
Algorithm	FA	FM	IA	IC
ISRB [20]	$2W-M$	$2W$	$W+V$	$2V-2N$
IISRB	$2W-M$	$2W$	W	W
RS-IISRB	$2W-M+2\gamma$	$2W+\gamma$	$2W+N+2$	$5N+3W$
EIHRB [23]	$3W-2M$	$3W$	$2W+V$	$2V-2N+W$
IEIHRB	$2W-M$	$2W$	W	W
RS-IEIHRB	$2W-M+2\gamma$	$2W+\gamma$	$2W+N+2$	$5N+3W$

Complexities of the hard-reliability based algorithms can be analyzed similarly. The IEIHRB algorithm has the same computational complexity per iteration as the IISRB algorithm, because they have the same iteration procedure. For the same reason, the RS-IEIHRB algorithm has the same computational complexity per iteration as the RS-IISRB algorithm.

Tables 2.5 and 2.6 compare computational complexities of various decoding algorithms needed by initialization and per iteration, respectively. For the initialization step, the numbers of IMs of the IISRB and RS-IISRB algorithms are greater than those of the ISRB algorithm because the calculation of $\xi_2\phi_{i,j}$ is done in initialization to reduce computational complexities of iterations. This is a good trade-off for computational complexity. The number of ICs needed by the initialization step of the ISRB algorithm provided in [20, Section III-A] is significantly greater than those of the other algorithms. This is because in [20], $\phi_{i,j}$'s are calculated for every i and j, and $\max_l \varphi_{t,l}$'s are re-calculated for each $\phi_{i,j}$. For each iteration, the numbers of integer operations required by the ISRB and EIHRB algorithms scale with 2^r, the order of the finite field. With the new reliability information update, the numbers of integer operations are reduced greatly and are now independent of 2^r. The re-selection scheme incurs some additional complexity, but complexities of the RS-IISRB and RS-IEIHRB algorithms are still lower than those of the ISRB and EIHRB algorithms, respectively.

Tables 2.7 and 2.8 list the numbers of various operations for initialization and each iteration, respectively, needed by various decoding algorithms for the (255,175) code. For initialization, the ISRB algorithm needs significantly more ICs than the other algorithms. When the order of the finite field is higher, our improved algorithms reduce the numbers of IAs and ICs for each iteration significantly.

From a perspective of the computational complexity, the IISRB and IEIHRB algorithms are the best. The re-selection scheme needs more finite field operations and integer operations. All the improved algorithms are simpler than the ISRB and EIHRB algorithms.

We also consider the memory overhead required by the two improvements. Our first improvement—the new reliability information update—does not need any extra memory. The second improvement—the re-selection scheme—needs to store $\mathbf{z}^{(k-1)}$

and $\mathbf{z}^{(k-2)}$ and hence requires $2Nr$ extra memory bits. Hence, the re-selection scheme increases the memory requirement slightly, but it does lower the error floor.

Table 2.7: Computational complexities of the initialization step for various decoding algorithms to decode the (255,175) code

Algorithm	IA	IM	IC	ID	Floor
ISRB [20]	522240	65280	696417750	0	0
IISRB	522240	130560	57120	0	0
RS-IISRB	522240	130560	58905	0	0
EIHRB [23]	522750	0	65280	65280	65280
IEIHRB	522750	0	65280	65280	65280
RS-IEIHRB	522750	0	67065	65280	65280

Table 2.8: Computational complexities required per iteration for various decoding algorithm to decode the (255,175) code

Algorithm	FA	FM	IA	IC
ISRB [20]	7905	8160	69360	130050
IISRB	7905	8160	4080	4080
RS-IISRB	7937	8176	8417	13515
EIHRB [23]	11730	12240	73440	134130
IEIHRB	7905	8160	4080	4080
RS-IEIHRB	7937	8176	8417	13515

2.4 Summary

In this chapter, we propose two improvements to the soft- and hard-reliability based MLGD algorithms for non-binary LDPC codes. The first improvement—the new reliability information update—helps the reliability-based MLGD algorithms achieve better BLERs, require fewer iterations, and have lower complexities. The second improvement—the re-selection scheme—results in a better error performance, fewer iterations on average, and a lower error floor. Although the re-selection scheme needs additional complexity, the MLGD algorithms with the re-selection scheme still require lower computational complexities than the existing MLGD algorithms.

Chapter 3

Symbol-Decision Polar Decoder

In this chapter, we address the throughput/latency issue of polar decoders by proposing symbol-decision SC and SCL decoders, which are based on symbol-wise hard or soft decisions. Since each symbol consists of M bits, when $M > 1$ the symbol-decision decoders achieve higher throughput as well as better error performance. The proposed symbol-decision decoders are natural generalization of their bit-wise counterparts, and reduce to existing bit-wise decoders when the symbol size is one bit. The main contributions of this chapter are:

- We propose a novel recursive channel combination to calculate the symbol-wise channel transition probabilities, which enable symbol decisions in SC and SCL algorithms, based on the bit-wise channel transformation [24]. The proposed recursive channel combination has lower complexity than simply combining bit-wise channel transition probabilities. The similarity between Arıkan's recursive channel transformation and our symbol-based recursive channel combination also helps to share hardware resources to calculate the bit- and symbol-wise channel transition probabilities.

- An M-bit symbol-decision SCL decoder needs to select the L most reliable candidates out of a list consisting of $2^M L$ list candidates. To perform this pruning function, we propose a two-stage list pruning network, which provides a trade-off between performance and complexity.

- By adopting the pre-computation technique [54], We develop a pre-computation memory-saving (PCMS) technique to reduce the memory requirement of the SCL decoder. Specifically, the channel information memory can be eliminated when using the PCMS technique. Moreover, this technique also helps to improve throughput slightly.

- To evaluate the throughput of symbol-decision SCL decoders, we propose an area efficient architecture for symbol-decision SCL decoders[1]. In our architecture, to save the area, adders in processing units are reused to calculate the symbol-wise channel transition probability. We propose two scheduling schemes for sharing hardware resources. We also propose two list pruning networks for designs with different symbol sizes.

- We design two-, four-, and eight-bit symbol-decision SCL decoders for a (1024, 480) CRC32-concatenated polar code with a list size of four. Synthesis results show that in terms of area efficiency, our symbol-decision SCL decoder outperforms all existing state-of-the-art SCL decoders in [3, 43–45]. For example, the area efficiency of our four-bit symbol-decision SCL decoder is 259.2 Mb/s/mm^2,

[1]We focus on the SCL decoder because the SC decoder can be considered as an SCL decoder with a list size of one.

which is 1.51 times as big as that of [45]. Our implementation results also demonstrate that the symbol-decision SCL decoder can provide a range of trade-offs between area, throughput, and area efficiency.

Our symbol-decision decoding algorithms assume that the underlying channel has a binary input, and our symbol-based channel combination is virtual and introduced for decoding only. Hence, our work is different from those assuming a q-ary ($q > 2$) channel (see, for example, [55]).

The decoding schedule (bit sequence) of our symbol-decision decoding algorithms is actually the same as those in [1–3], but our symbol-decision decoding algorithms are different from multi-bit decoding algorithms in [1–3] in two aspects. First, our symbol-based recursive channel combination is different from how symbol-wise transition probabilities are derived in [1–3]. Second, the symbol-decision perspective allows us to prove that the symbol-decision algorithms have better frame error rates (FERs) than their bit-decision counterparts [56] for SC algorithms, while only simulation results are provided in [2, 3] and error performance is not investigated in [1]. There are additional differences between our decoding algorithms/architectures and those in [1–3]. For instance, all the bits within a symbol are estimated jointly in our symbol-decision SC algorithm, whereas some bits are sometimes decoded independently for the decoder with parallelism two in [2]. Also, while our symbol-decision decoding is introduced on the algorithmic level, the multi-bit decoder is introduced on the level of decoding operations [3]. Finally, for our symbol-decision SCL decoder architectures, we use the semi-parallel architecture because it is more area efficient than the tree architecture and the line architecture in [35], on which the multi-bit decoder architecture in [3] is based. No hardware implementation was presented in [1, 2].

The proposed PCMS technique is useful for our proposed symbol-decision SCL decoder as well as prior bit-decision and multi-bit SCL decoders regardless of the message representation. Zhang *et al*. [41] also used the pre-computation technique to improve the throughput of the SC decoder with the cost of increased area. In contrast, our proposed PCMS technique aims to reduce the memory requirement of the SCL decoder, while improving the throughput slightly.

The hardware implementations in this chapter are important in verifying the throughput/latency advantage of symbol-decision decoders. Some prior works claim a throughput improvement proportional to the size of the symbol (in bits) (see, for example, [2]). Such claim usually assumes constant clock cycle while growing the symbol size. This assumption is unrealistic, as shown by our implementations and discussion in Section 3.4.

The rest of our chapter is organized as follows. Section 3.1 briefly reviews polar codes and existing decoding algorithms for polar codes. In Section 3.2, the symbol-based recursive channel combination is proposed to calculate the symbol-wise channel transition probability. Moreover, to simplify the selection of the list candidates, a two-stage list pruning network is proposed. In Section 3.3, we introduce a method to reduce memory requirement of list decoders of polar codes by the pre-computation technique. In Section 3.4, we present the architecture for symbol-decision SCL decoders. Two scheduling schemes for hardware sharing are discussed. We also propose two list pruning networks for different designs: a folded sorting implementation and a tree sorting implementation. A discussion on the latency of our architecture and synthesis results for our implementations are provided in this section as well. Finally, we draw some conclusions in Section 3.5.

3.1 Polar Codes and Existing Decoding Algorithms

3.1.1 Preliminaries

We follow the notation for vectors in [24], namely $u_a^b = (u_a, u_{a+1}, \cdots, u_{b-1}, u_b)$ when $a \leq b$; if $a > b$, u_a^b is regarded as void. $u_{a,o}^b$ and $u_{a,e}^b$ denote the subvectors of u_a^b with odd and even indices, respectively.

Let $W : \mathcal{X} \to \mathcal{Y}$ represent a generic discrete memoryless channel with binary input alphabet $\mathcal{X}$, arbitrary output alphabet $\mathcal{Y}$, and transition probabilities $W(y|x)$, $y \in \mathcal{Y}$, $x \in \{0, 1\}$. Let $W_N^{(j)}$ denote a set of N coordinate channels:

$$W_N^{(j)} : \mathcal{X} \to \mathcal{Y}^N \times \mathcal{X}^{j-1}, 0 < j \leq N$$

with transition probabilities $W_N^{(j)}(y_1^N, x_1^{j-1}|x_j)$, where (y_1^N, x_1^{j-1}) and x_j denote the output and input of $W_N^{(j)}$, respectively.

3.1.2 Polar Codes

Polar codes are linear block codes, and their block lengths are restricted to powers of two, denoted by $N = 2^n$ for $n \geq 2$. Assume $\mathbf{u} = u_1^N = (u_1, u_2, \cdots, u_N)$ is the data bit sequence. Let $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. The corresponding encoded bit sequence $\mathbf{x} = x_1^N = (x_1, x_2, \cdots, x_N)$ is generated by

$$\mathbf{x} = \mathbf{u} B_N F^{\otimes n}, \tag{3.1}$$

where B_N is the $N \times N$ bit-reversal permutation matrix and $F^{\otimes n}$ denotes the n-th Kronecker power of F [24].

For any index set $\mathcal{A} \subseteq \{1, 2, \cdots, N\}$, $\mathbf{u}_{\mathcal{A}} = (u_i : 0 < i \leq N, i \in \mathcal{A})$ is the sub-sequence of $\mathbf{u}$ restricted to $\mathcal{A}$. For an (N, K) polar code, the data bit sequence is grouped into two parts: a K-element part $\mathbf{u}_{\mathcal{A}}$ which carries information bits, and $\mathbf{u}_{\mathcal{A}^c}$ whose elements are predefined frozen bits, where $\mathcal{A}^c$ is the complement of $\mathcal{A}$. For convenience, frozen bits are set to zero.

3.1.3 SC Algorithm for Polar Codes

Given a transmitted codeword $\mathbf{x}$ and the corresponding received word $\mathbf{y}$, the SC algorithm for an (N, K) polar code estimates the encoding bit sequence $\mathbf{u}$ successively as shown in Alg. 4. Here, $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_N)$ represents the estimated value for $\mathbf{u}$.

Algorithm 4: SC Decoding Algorithm [24]

```
1 for $j = 1 : N$ do
2     if $j \in \mathcal{A}^c$ then $\hat{u}_j = 0$ else
3         if $\frac{W_N^{(j)}(\mathbf{y}, \hat{u}_1^{j-1}|u_j=1)}{W_N^{(j)}(\mathbf{y}, \hat{u}_1^{j-1}|u_j=0)} \geq 1$ then $\hat{u}_j = 1$ else $\hat{u}_j = 0$
4
```

To calculate $W_N^{(j)}(\mathbf{y}, \hat{u}_1^{j-1}|u_j)$, Arıkan's recursive channel transformation [24] is applied. A pair of binary channels $W_{2\Lambda}^{(2i-1)}$ and $W_{2\Lambda}^{(2i)}$ are obtained by a single-step transformation of two independent copies of a binary input channel $W_{\Lambda}^{(i)} : (W_{\Lambda}^{(i)}, W_{\Lambda}^{(i)}) \mapsto (W_{2\Lambda}^{(2i-1)}, W_{2\Lambda}^{(2i)})$. The channel transition probabilities of $W_{2\Lambda}^{(2i-1)}$ and $W_{2\Lambda}^{(2i)}$ are given by

$$\begin{aligned}
&W_{2\Lambda}^{(2i-1)}(y_1^{2\Lambda}, u_1^{2i-2}|u_{2i-1}) \\
&\quad = \frac{1}{2}\sum_{u_{2i}}\Big[W_{\Lambda}^{(i)}(y_1^{\Lambda}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus u_{2i}) \\
&\qquad\qquad \cdot W_{\Lambda}^{(i)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{2i-2}|u_{2i})\Big],
\end{aligned} \tag{3.2}$$

and

$$\begin{aligned}
&W_{2\Lambda}^{(2i)}(y_1^{2\Lambda}, u_1^{2i-1}|u_{2i}) \\
&\quad = \frac{1}{2}W_{\Lambda}^{(i)}(y_1^{\Lambda}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus u_{2i}) \\
&\qquad\qquad \cdot W_{\Lambda}^{(i)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{2i-2}|u_{2i}),
\end{aligned} \tag{3.3}$$

where $0 < i \leq \Lambda = 2^{\lambda} < N$ and $0 \leq \lambda < n$.

Expressed in log-likelihood (LL), Eqs. (3.2) and (3.3) can be approximated as [26]:

$$\begin{aligned}
&\mathrm{LL}_{2\Lambda}^{(2i-1)}(y_1^{2\Lambda}, u_1^{2i-2}|u_{2i-1}) \\
&\quad \approx \max\Bigg\{\Big[\mathrm{LL}_{\Lambda}^{(i)}(y_1^{\Lambda}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 0) \\
&\qquad\qquad + \mathrm{LL}_{\Lambda}^{(i)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{2i-2}|0)\Big], \\
&\qquad\quad \Big[\mathrm{LL}_{\Lambda}^{(i)}(y_1^{\Lambda}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus 1) \\
&\qquad\qquad + \mathrm{LL}_{\Lambda}^{(i)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{2i-2}|1)\Big]\Bigg\} - \log 2,
\end{aligned} \tag{3.4}$$

$$\begin{aligned}
&\mathrm{LL}_{2\Lambda}^{(2i)}(y_1^{\Lambda}, u_1^{2i-1}|u_{2i}) \\
&\quad = \mathrm{LL}_{\Lambda}^{(i)}(y_1^{\Lambda}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus u_{2i}) \\
&\qquad\qquad + \mathrm{LL}_{\Lambda}^{(i)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{2i-2}|u_{2i}) - \log 2.
\end{aligned} \tag{3.5}$$

To simplify the calculation, the constants in Eqs. (3.4) and (3.5) can be discarded since this global offset for all LLs does not affect the decoding decision.

3.1.4 SCL and CA-SCL Algorithms for Polar Codes

Algorithm 5: SCL Decoding Algorithm [32]

```
$\alpha = 1$;
for $j = 1 : N$ do
    if $j \in \mathcal{A}^c$ then
        for $i = 1 : \alpha$ do
            $(\mathcal{L}_i)_j = 0$;
    else if $2\alpha \leq L$ then
        for $i = 1 : \alpha$ do
            $(\mathcal{L}_i)_1^j =$conc$((\mathcal{L}_i)_1^{j-1}, 0)$;
            $(\mathcal{L}_{i+\alpha})_1^j =$conc$((\mathcal{L}_i)_1^{j-1}, 1)$;
        $\alpha = 2\alpha$;
    else
        for $i = 1 : L$ do
            $\mathsf{S}[i].\mathsf{P} = W_N^{(j)}(\mathbf{y}, (\mathcal{L}_i)_1^{j-1}|0)$;
            $\mathsf{S}[i].\mathsf{L} = (\mathcal{L}_i)_1^{j-1}$;
            $\mathsf{S}[i].\mathsf{U} = 0$;
            $\mathsf{S}[i+L].\mathsf{P} = W_N^{(j)}(\mathbf{y}, (\mathcal{L}_i)_1^{j-1}|1)$;
            $\mathsf{S}[i+L].\mathsf{L} = (\mathcal{L}_i)_1^{j-1}$;
            $\mathsf{S}[i+L].\mathsf{U} = 1$;
        sortPDecrement($\mathsf{S}$);
        for $i = 1 : L$ do
            $(\mathcal{L}_i)_1^j =$conc$(\mathsf{S}[i].\mathrm{L}, \mathsf{S}[i].\mathrm{U})$;
        $\alpha = L$;
$\hat{\mathbf{u}} = \mathbf{L}_1$;
```

Instead of making a hard decision for each information bit of $\mathbf{u}$ in the SC algorithm, the SCL algorithm creates two paths in which the information bit is assumed to be 0 and 1, respectively. If the number of paths is greater than the list size L, the L most reliable paths are selected. At the end of the decoding procedure, the most reliable path is chosen as $\hat{\mathbf{u}}$. The SCL algorithm is formally described in Alg. 5. Without loss of generality, we assume L to be a power of two, i.e. $L = 2^l$. We use $\mathbf{L}_i = ((\mathcal{L}_i)_1, (\mathcal{L}_i)_1, \cdots, (\mathcal{L}_i)_N)$ to represent the i-th list vector, where $0 < i \leq L$. S is a structure type array with size $2L$. Each element of S has three members: P, L, and U. The function `sortPDecrement` sorts the array S by decreasing order of P. `c=conc(a,b)` attaches a bit sequence `b` at the end of a bit sequence `a`, and the length of the output bit sequence `c` is the sum of lengths of `a` and `b`.

The CA-SCL algorithm is used for the CRC-concatenated polar codes. The difference between CA-SCL [33] and SCL algorithms is how to make the final decision for $\hat{\mathbf{u}}$. If there is at least one path satisfying the CRC constraint, the most reliable CRC-valid path is chosen for $\hat{\mathbf{u}}$. Otherwise, the decision rule of the SCL algorithm is used for the CA-SCL algorithm.

3.1.5 Multi-bit SC and SCL Algorithms for Polar Codes

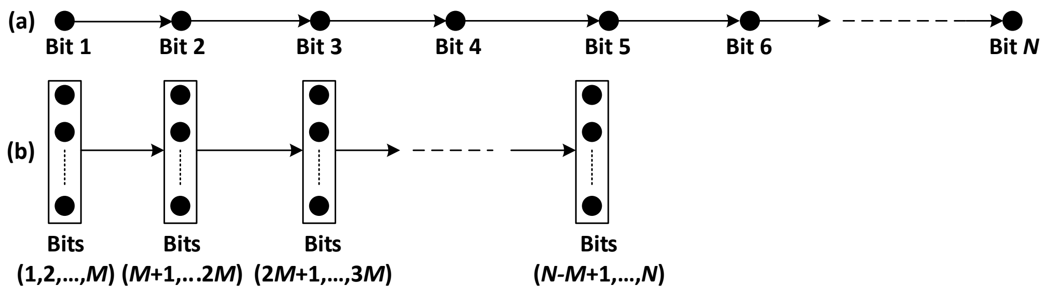


Figure 3.1: (a) bit-decision vs. (b) M-bit decision

The SC and SCL algorithms described above make hard or soft decision for only one bit at a time, and their decoding schedule is shown in Fig. 3.1(a). We call them bit-decision decoding algorithms. Multi-bit SC and SCL decoders [1–3] make hard or soft decision for M bits instead of only one bit at a time, as shown in Fig. 3.1(b). If $M = N$, these decoding algorithms are exactly maximum-likelihood sequence decoding algorithms.

3.2 M-bit Symbol-Decision Decoding Algorithms for Polar Codes

3.2.1 Symbol-Decision SC and SCL Algorithms

Here, we propose symbol-decision SC and SCL algorithms, which also have the schedule in Fig. 3.1(b). However, we treat each group of M bits in Fig. 3.1(b) as an M-bit symbol. Let $\mathcal{Z}$ represent the alphabet of all M-bit symbols. The symbol-decision algorithms deal with the virtual channels

$$\mathcal{W}_N^{(j)} : \mathcal{Z} \to \mathcal{Y}^N \times \mathcal{Z}^{j-1}, 0 < j \leq \frac{N}{M}$$

with transition probabilities $\mathcal{W}_N^{(j)}(y_1^N, z_1^{j-1}|z_j)$, where (y_1^N, z_1^{j-1}) and $z_j = (u_{jM-M+1}, \cdots, u_{jM})$ denote the output and input of $\mathcal{W}_N^{(j)}$, respectively.

3.2.2 Symbol-Based Recursive Channel Combination

When the channel input alphabet is $\mathcal{Z}$, a recursive channel transformation similar to (3.2) and (3.3) can be used to derive $\mathcal{W}_N^{(j)}$ (see [55, (2) and (3)]). We focus on the case where the channel input is binary.

In [1–3], the channel transition probability of a group of bits is obtained by combining the bit-wise channel transition probability, referred to as direct-mapping calculation henceforth. Let $W_{N,M}^{(j)}$ denote a set of $\frac{N}{M}$ coordinate channels:

$$W_{N,M}^{(j)} : \mathcal{X}^M \rightarrow \mathcal{Y}^N \times \mathcal{X}^{jM-M}, 0 < j \leq \frac{N}{M}$$

with transition probabilities $W_{N,M}^{(j)}(y_1^N, x_1^{jM-M}|x_{jM-M+1}^{jM})$, where (y_1^N, x_1^{jM-M}) and x_{jM-M+1}^{jM} denote the output and input of $W_{N,M}^{(j)}$, respectively. If $M = 1$, $W_{N,1}^{(j)} = W_N^{(j)}$. Let

$$u_{iM-M+1}^{iM} = (w_i, w_{i+\frac{N}{M}}, \cdots, w_{i+N-\frac{N}{M}})B_M F^{\otimes m}, 1 \leq i \leq \frac{N}{M}.$$

In [1–3], to perform the multi-bit decision, the calculation of channel transition probability $W_{N,M}^{(i)}(\mathbf{y}, \hat{u}_1^{iM-M}|u_{iM-M+1}^{iM})$ is based on a product of bit-wise channel transition probabilities:

$$\begin{aligned} &W_{N,M}^{(i)}(\mathbf{y}, \hat{u}_1^{iM-M}|u_{iM-M+1}^{iM}) = \\ &\prod_{j=0}^{M-1} W_{\frac{N}{M}}^{(i)}(y_{j\frac{N}{M}+1}^{(j+1)\frac{N}{M}}, \hat{w}_{1+j\frac{N}{M}}^{(i-1)+j\frac{N}{M}}|w_{i+j\frac{N}{M}}), \end{aligned} \tag{3.6}$$

where $W_{\frac{N}{M}}^{(i)}(y_{j\frac{N}{M}+1}^{(j+1)\frac{N}{M}}, \hat{w}_{1+j\frac{N}{M}}^{(i-1)+j\frac{N}{M}}|w_{i+j\frac{N}{M}})$ is calculated by Arıkan's recursive channel transformations.

Actually, $\mathcal{W}_N^{(j)}$ is exactly equivalent to $W_{N,M}^{(j)}$ if we consider $\mathcal{X}^M$ as the binary vector representation of $\mathcal{Z}$. Therefore, the symbol-decision SC/SCL algorithms have the same schedule as the multi-bit SC/SCL algorithms in [1–3]. However, our symbol-decision SC/SCL algorithms have a different approach, called symbol-based recursive channel combination (as described in Proposition 3.2.1), to compute channel transition probabilities $W_{N,M}^{(j)}(\mathbf{y}, \hat{u}_1^{jM-M}|u_{jM-M+1}^{jM})$, which is our main focus.

Proposition 3.2.1. Assume that all bits of $\mathbf{u}$ are independent and each bit has an equal probability of being a 0 or 1. Given $0 < m \leq n$, $N = 2^n$, $M = 2^m$, for any $1 \leq \phi \leq m$, $0 \leq \lambda < n$, $\Lambda = 2^\lambda$, $\Phi = 2^\phi$, and $0 \leq i < \frac{2\Lambda}{\Phi}$, we say that a Φ-bit channel $W_{2\Lambda,\Phi}^{(i+1)}$ is obtained by a single-step combination of two independent copies of a $\frac{\Phi}{2}$-bit channel $W_{\Lambda,\Phi/2}^{(i+1)}$ and write

$$(W_{\Lambda,\Phi/2}^{(i+1)}, W_{\Lambda,\Phi/2}^{(i+1)}) \mapsto W_{2\Lambda,\Phi}^{(i+1)}, \tag{3.7}$$

where channel transition probabilities satisfy,

$$\begin{aligned} &W_{2\Lambda,\Phi}^{(i+1)}(y_1^{2\Lambda}, u_1^{i\Phi}|u_{i\Phi+1}^{i\Phi+\Phi}) = \\ &\quad W_{\Lambda,\Phi/2}^{(i+1)}(y_1^{\Lambda}, u_{1,o}^{i\Phi} \oplus u_{1,e}^{i\Phi}|u_{i\Phi+1,o}^{i\Phi+\Phi} \oplus u_{i\Phi+1,e}^{i\Phi+\Phi}) \\ &\quad \cdot W_{\Lambda,\Phi/2}^{(i+1)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{i\Phi}|u_{i\Phi+1,e}^{i\Phi+\Phi}). \end{aligned} \tag{3.8}$$

Proof of Proposition 3.2.1. According to the definition of conditional probability $\Pr(B|A) = \frac{\Pr(AB)}{\Pr(A)}$,

$$\begin{aligned} &W_{2\Lambda,\Phi}^{(i+1)}(y_1^{2\Lambda}, u_1^{i\Phi}|u_{i\Phi+1}^{i\Phi+\Phi}) \\ &\qquad = \frac{W_{2\Lambda,1}^{(i\Phi+\Phi)}(y_1^{2\Lambda}, u_1^{i\Phi+\Phi-1}|u_{i\Phi+\Phi})}{\Pr(u_{i\Phi+1}^{i\Phi+\Phi-1}|u_{i\Phi+\Phi})}. \end{aligned} \tag{3.9}$$

Because all bits of $\mathbf{u}$ are independent and each bit has an equal probability of being a 0 or 1,

$$\Pr(u_{i\Phi+1}^{i\Phi+\Phi-1}|u_{i\Phi+\Phi}) = \Pr(u_{i\Phi+1}^{i\Phi+\Phi-1}) = 2^{-(\Phi-1)}.$$

Therefore,

$$\begin{aligned} &W_{2\Lambda,\Phi}^{(i+1)}(y_1^{2\Lambda}, u_1^{i\Phi}|u_{i\Phi+1}^{i\Phi+\Phi}) \\ &\qquad = 2^{(\Phi-1)}W_{2\Lambda,1}^{(i\Phi+\Phi)}(y_1^{2\Lambda}, u_1^{i\Phi+\Phi-1}|u_{i\Phi+\Phi}). \end{aligned} \tag{3.10}$$

According to (3.3),

$$\begin{aligned} &W_{2\Lambda,1}^{(i\Phi+\Phi)}(y_1^{2\Lambda}, u_1^{i\Phi+\Phi-1}|u_{i\Phi+\Phi}) \\ &\quad = \frac{1}{2}W_{\Lambda,1}^{(\frac{i\Phi+\Phi}{2})}(y_1^{\Lambda}, u_{1,o}^{i\Phi+\Phi-2} \oplus u_{1,e}^{i\Phi+\Phi-2}|u_{i\Phi+\Phi-1} \oplus u_{i\Phi+\Phi}) \\ &\quad \cdot W_{\Lambda,1}^{(\frac{i\Phi+\Phi}{2})}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{i\Phi+\Phi-2}|u_{i\Phi+\Phi}). \end{aligned} \tag{3.11}$$

Similarly, we have

$$\begin{aligned} &W_{\Lambda,1}^{(\frac{i\Phi+\Phi}{2})}(y_1^{\Lambda}, u_{1,o}^{i\Phi+\Phi-2} \oplus u_{1,e}^{i\Phi+\Phi-2}|u_{i\Phi+\Phi-1} \oplus u_{i\Phi+\Phi}) \\ &\quad = 2^{-(\frac{\Phi}{2}-1)}W_{\Lambda,\Phi/2}^{(i+1)}(y_1^{\Lambda}, u_{1,o}^{i\Phi} \oplus u_{1,e}^{i\Phi}|u_{i\Phi+1,o}^{i\Phi+\Phi} \oplus u_{i\Phi+1,e}^{i\Phi+\Phi}), \end{aligned} \tag{3.12}$$

and

$$\begin{aligned} &W_{\Lambda,1}^{(\frac{i\Phi+\Phi}{2})}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{i\Phi+\Phi-2}|u_{i\Phi+\Phi}) \\ &\qquad = 2^{-(\frac{\Phi}{2}-1)}W_{\Lambda,\Phi/2}^{(i+1)}(y_{\Lambda+1}^{2\Lambda}, u_{1,e}^{i\Phi}|u_{i\Phi+1,e}^{i\Phi+\Phi}). \end{aligned} \tag{3.13}$$

Then, by equations (3.10) to (3.13), equation (3.8) is obtained. □

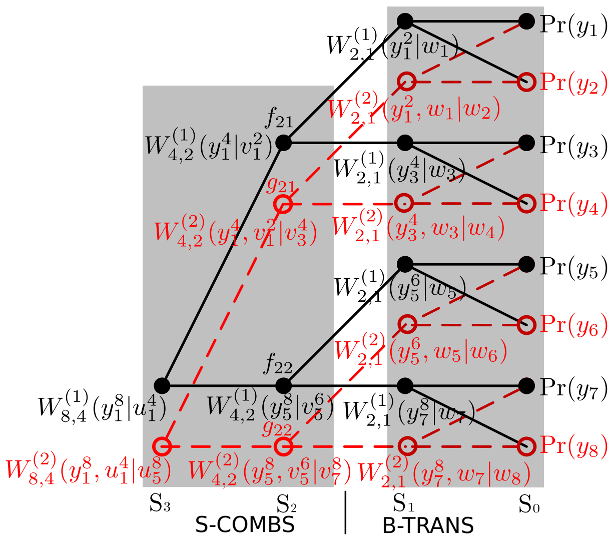
We observe that the channel combination in (3.8) is different from that in (3.2) and (3.3) and their symbol-wise counterpart in [55, (2) and (3)]. In (3.2), (3.3), and [55, (2) and (3)], the size of the symbol is fixed. In contrast, in (3.8) the symbol size (in bits) is doubled during each recursion call.

Compared with the direct-mapping calculation in [1–3], our recursive channel combination has lower complexity. We will illustrate this from the perspective of a message flow graph (MFG). Similar to the SC algorithm, with the help of the symbol-based recursive channel combination, an M-bit symbol-decision SC algorithm can be represented by an MFG as well, where a channel transition probability is referred to as a *message* for the sake of convenience. This MFG is referred to as SR-MFG. If the code length of a polar code is N, the SR-MFG can be divided into $(n + 1)$ stages $(\mathrm{S}_0, \mathrm{S}_1, \cdots, \mathrm{S}_n)$ from the right to the left: one initial stage S_0 and n calculation stages. For the SC algorithm, all calculation stages carry out Arıkan's recursive channel transformations. However, for the M-bit symbol-decision SC algorithm, in the left-most m calculation stages $(\mathrm{S}_n, \cdots, \mathrm{S}_{n-m+1})$, called S-COMBS stages, symbol-based channel combinations are carried out. For the remaining $(n - m)$ calculation stages $(\mathrm{S}_{n-m}, \cdots, \mathrm{S}_1)$, called B-TRANS stages, Arıkan's recursive channel transformations are performed. The S-COMBS stages use outputs of B-TRANS stages to calculate symbol-wise messages.

We refer to the MFG in [1–3] as the DM-MFG, which also consists of two parts: B-TRANS and DM-CAL. The B-TRANS part of the DM-MFG is the same as that of the SR-MFG. However, there is only one stage in the DM-CAL part, and it performs the direct-mapping calculation.

For example, as shown in Fig. 3.2, the SR-MFG of a four-bit symbol-decision SC algorithm for a polar code with $N = 8$ has four stages. Messages of the initial stage (S_0) come from the channel directly. Messages of the first stage (S_1) are calculated with Arıkan's transformations. Messages of the second and third stages (S2 and S3) are calculated with (3.8). Stages in the left gray box are the S-COMBS stages. Stages in the right gray box are the B-TRANS stages. Fig. 3.3 shows the DM-MFG when the direct-mapping calculation is used to calculate symbol-wise channel transition probabilities $W_{8,4}^{(1)}(y_1^8|u_1^4)$ and $W_{8,4}^{(2)}(y_1^8, u_1^4|u_5^8)$. Here,

$$
\begin{aligned}
&v_1^4 = u_{1,o}^8 \oplus u_{1,e}^8, \quad v_5^8 = u_{1,e}^8,\\
&w_1 = v_1 \oplus v_2 = u_1 \oplus u_2 \oplus u_3 \oplus u_4,\\
&w_2 = v_3 \oplus v_4 = u_5 \oplus u_6 \oplus u_7 \oplus u_8,\\
&w_3 = v_2 = u_3 \oplus u_4,\\
&w_4 = v_4 = u_7 \oplus u_8,\\
&w_5 = v_5 \oplus v_6 = u_2 \oplus u_4,\\
&w_6 = v_7 \oplus v_8 = u_6 \oplus u_8,\\
&w_7 = v_6 = u_4,\\
&w_8 = v_8 = u_8.
\end{aligned}
$$



$\Pr(y_1)$
$W_{2,1}^{(1)}(y_1^2|w_1)$
$\Pr(y_2)$
$W_{2,1}^{(2)}(y_1^2, w_1|w_2)$
f_{21}
$W_{4,2}^{(1)}(y_1^4|v_1^2)$
$\Pr(y_3)$
$W_{2,1}^{(1)}(y_3^4|w_3)$
g_{21}
$\Pr(y_4)$
$W_{4,2}^{(2)}(y_1^4, v_1^2|v_3^4)$
$W_{2,1}^{(2)}(y_3^4, w_3|w_4)$
$\Pr(y_5)$
$W_{2,1}^{(1)}(y_5^6|w_5)$
$\Pr(y_6)$
$W_{2,1}^{(2)}(y_5^6, w_5|w_6)$
f_{22}
$\Pr(y_7)$
$W_{8,4}^{(1)}(y_1^8|u_1^4)$
$W_{4,2}^{(1)}(y_5^8|v_5^6)$
$W_{2,1}^{(1)}(y_7^8|w_7)$
g_{22}
$\Pr(y_8)$
$W_{8,4}^{(2)}(y_1^8, u_1^4|u_5^8)$
$W_{4,2}^{(2)}(y_5^8, v_5^6|v_7^8)$
$W_{2,1}^{(2)}(y_7^8, w_7|w_8)$
S₃
S₂
S₁
S₀
S-COMBS
B-TRANS

Figure 3.2: The message flow graph of a four-bit symbol-decision SC algorithm for a polar code with code length eight by using the proposed symbol-based recursive channel combination.

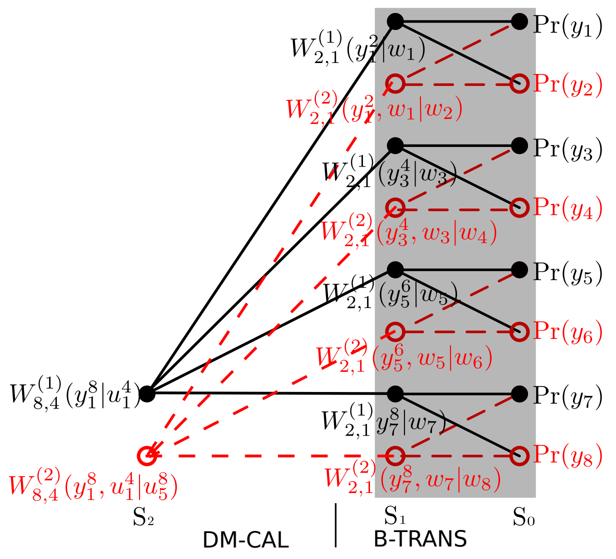

$W_{2,1}^{(1)}(y_1^2|w_1)$
$W_{2,1}^{(2)}(y_1^2, w_1|w_2)$
$W_{2,1}^{(1)}(y_3^4|w_3)$
$W_{2,1}^{(2)}(y_3^4, w_3|w_4)$
$W_{2,1}^{(1)}(y_5^6|w_5)$
$W_{2,1}^{(2)}(y_5^6, w_5|w_6)$
$W_{2,1}^{(1)}y_7^8|w_7)$
$W_{2,1}^{(2)}(y_7^8, w_7|w_8)$
$W_{8,4}^{(1)}(y_1^8|u_1^4)$
$W_{8,4}^{(2)}(y_1^8, u_1^4|u_5^8)$
$\Pr(y_1)$
$\Pr(y_2)$
$\Pr(y_3)$
$\Pr(y_4)$
$\Pr(y_5)$
$\Pr(y_6)$
$\Pr(y_7)$
$\Pr(y_8)$
S₂
S₁
S₀
DM-CAL
B-TRANS

Figure 3.3: The message flow graph of a four-bit multi-bit SC algorithm for a polar code with code length eight by using direct-mapping calculation [1–3].

The direct-mapping calculation in (3.6) needs $(M-1)$ additions. Therefore, a total of $2^{|\mathcal{AM}_j|}(M-1)$ additions are needed to calculate all LL-based symbol-wise channel transition probabilities for u_{jM+1}^{jM+M}. Let us consider the recursive symbol-based channel combination now. The S-COMBS stages of the SR-MFG are indexed as 1 to m from left to right. There are $2^{n-i}(0 < i \leq m)$ nodes in the i-th S-COMBS stage and each node contains 2^{M+i-n} messages. One addition is needed to compute each LL message according to (3.8). Hence, the number of additions needed by the S-COMBS stages to calculate $W_{N,M}^{(j)}(\mathbf{y}, \hat{u}_1^{jM-M}|u_{jM-M+1}^{jM})$ is $\sum_{i=1}^{m-1} 2^i 2^{\frac{M}{2^i}} + 2^{|\mathcal{AM}_j|}$. When we perform the hardware implementation, the worst case - that all bits of a symbol are information bits - should be considered. Therefore, the recursive symbol-based channel combination can be taken advantage of to reduce complexity of calculating the symbol-wise channel transition probability.

For the example shown in Fig. 3.2, (3.6) needs $2^4(4-1) = 48$ additions to calculate

$\log(W_{8,4}^{(1)}(y_1^8|u_1^4))$. With the symbol-based channel combination, 4, 4 and 16 additions are needed to calculate $\log(W_{4,2}^{(1)}(y_1^4|v_1^2))$, $\log(W_{4,2}^{(1)}(v_5^8|v_5^6))$ and $\log(W_{8,4}^{(1)}(y_1^8|u_1^4))$, respectively. Therefore, our method needs only 24 additions, which is only a half of those needed by (3.6). Table 3.1 lists the numbers of additions needed by our recursive method and the direct-mapping calculation [1–3] when all M bits of a symbol are information bits. When $M = 8$, additions needed by our proposed method are 17% of that needed by the direct-mapping calculation.

Table 3.1: The numbers of additions to calculate $W_{N,M}^{(j+1)}(\mathbf{y}, \hat{u}_1^{jM}|u_{jM+1}^{jM+M})$ when the $(j+1)$-th M-bit symbol has no frozen bit.

	Proposed method	Direct-mapping calculation [1–3]
$M = 2$	4	4
$M = 4$	24	48
$M = 8$	304	1792

The other advantage of the proposed method to calculate the symbol-wise channel transition probability is that it reveals the similarity between Arıkan's recursive channel transformations and the symbol-based recursive channel combination. We will take advantage of this similarity to reuse adders and to save area when computing the bit- and symbol-wise channel transition probability in our proposed architecture. In [3], additional dedicated adders are used to calculate the symbol-wise channel transition probability, which is not area efficient.

In terms of the error performance, the symbol-decision SC algorithm is not worse than the bit-decision SC algorithm [56]. Fig. 3.4 shows the BERs and FERs of symbol-decision SC algorithms for a (1024, 512) polar codes. SDSC-i denotes the i-bit symbol-decision SC algorithm. When $M = 2$ and 4, the FER performance is the same as that of the bit-decision SC algorithm. When $M = 8$, the FER performance is slightly better.

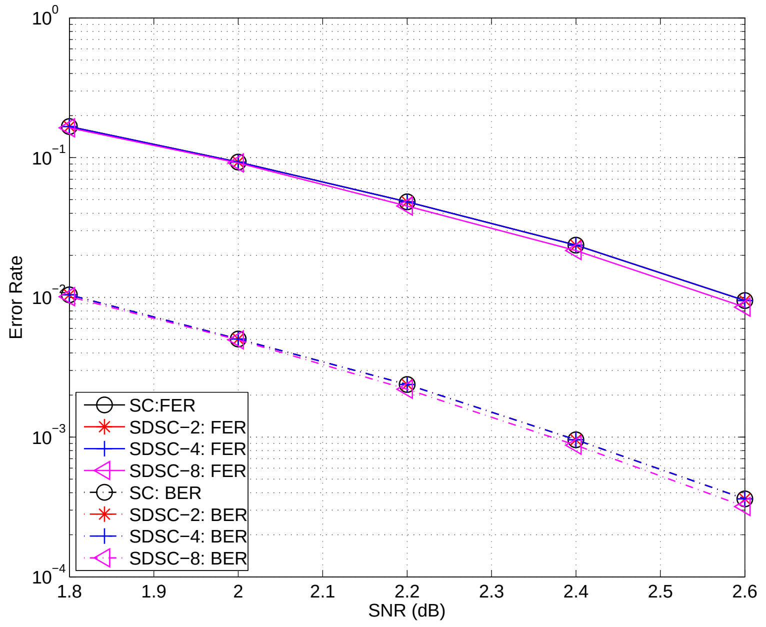


Figure 3.4: Error rates of symbol-decision SC algorithms for a (1024, 512) polar code.

3.2.3 Generalized Symbol-Decision SCL Algorithm

The symbol-based recursive channel combination is also applicable for the SCL algorithm. The symbol-decision SCL algorithm is more complex than the SCL algorithm, since the path expansion coefficient is not a constant any more. In the bit-decision SCL algorithm, the path expansion coefficient is two for each information bit. For the M-bit symbol-decision SCL algorithm, the path expansion coefficient is $2^{|\mathcal{AM}_j|}$, which depends on the number of information bits in the j-th symbol. The M-bit symbol-decision SCL algorithm is formally described in Alg. 6. Without any ambiguity, $\mathbf{0}$ represents a zero vector, whose bit-width is determined by its left-hand operator. The function $\texttt{dec2bin}(d, b)$ converts a decimal number d to a b-bit binary vector. (3.8) is used to calculate the symbol-wise channel transition probability corresponding to each list, i.e. $W_{N,M}^{(j+1)}(\mathbf{y}, (\mathcal{L}_i)_1^{jM} | u_{jM+1}^{jM+M})$.

Algorithm 6: Symbol-Decision SCL Algorithm

```
$\alpha = 1$;
for $j = 1 : \frac{N}{M}$ do
    $\beta = 2^{|\mathcal{AM}_j|}$;
    if $\beta == 1$ then
        for $i = 1 : \alpha$ do
            $(\mathcal{L}_i)_{jM-M+1}^{jM} = \mathbf{0}$;
    else if $\alpha\beta \leq L$ then
        $u_{\mathcal{AM}_j^c} = \mathbf{0}$;
        for $k = 0 : \beta - 1$ do
            $u_{\mathcal{AM}_j}$ =dec2bin($k, |\mathcal{AM}_j|$);
            for $i = 1 : \alpha$ do
                $t = i + k\alpha$;
                $(\mathcal{L}_t)_1^{jM}$ =conc($(\mathcal{L}_i)_1^{jM-M}, u_{jM-M+1}^{jM}$);
        $\alpha = \alpha\beta$;
    else
        $u_{\mathcal{AM}_j^c} = \mathbf{0}$;
        for $k = 0 : \beta - 1$ do
            $u_{\mathcal{AM}_j}$ =dec2bin($k, |\mathcal{AM}_j|$);
            for $i = 1 : L$ do
                $t = i + kL$;
                $\mathsf{S}[t].\mathsf{P} = W_{N,M}^{(j)}(\mathbf{y}, (\mathcal{L}_i)_1^{jM-M} | u_{jM-M+1}^{jM})$;
                $\mathsf{S}[t].\mathsf{L} = (\mathcal{L}_i)_1^{jM-M}$;
                $\mathsf{S}[t].\mathsf{U} = u_{jM-M+1}^{jM}$;
        sortPDecrement($\mathsf{S}$);
        for $i = 1 : L$ do
            $(\mathcal{L}_i)_1^{jM}$ =conc($\mathsf{S}[i]$.L, $\mathsf{S}[i]$.U);
        $\alpha = L$;
```

Fig. 3.5 shows the BERs and FERs of symbol-decision SCL algorithms for a (1024, 480) CRC32-concatenated polar code with $L = 4$ where the generator polynomial of the CRC32 is 0x1EDC6F41. This CRC32 is also used in all the CRC-concatenated polar codes henceforth. SDSCL-i denotes the i-bit symbol-decision SCL algorithm. The performances of the symbol-decision SCL algorithms with different symbol sizes are almost the same.

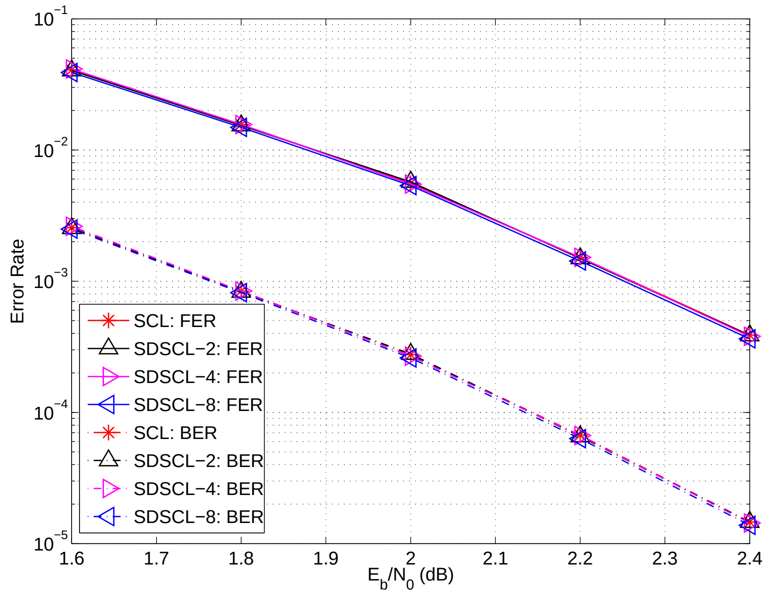


Figure 3.5: Error rates of symbol-decision SCL algorithms for a (1024, 480) CRC32-concatenated polar code with $L = 4$.

3.2.4 Two-Stage List Pruning Network for the Symbol-Decision SCL Algorithm

For the M-bit symbol-decision SCL algorithm, the maximum path expansion coefficient is 2^M, i.e. each existing path generates 2^M paths. Therefore, in the worst-case scenario, the L most reliable paths should be selected out of $2^M L$ paths. To facilitate this sorting network, we propose a two-stage list pruning network. In the first stage, the q most reliable paths are selected from up to 2^M paths that come from the expansion of each existing path. Therefore, there are qL paths left. In the second stage, the L most reliable paths are selected from the qL paths generated by the first stage. The message flow of a two-stage list pruning network is illustrated in Fig. 3.6.

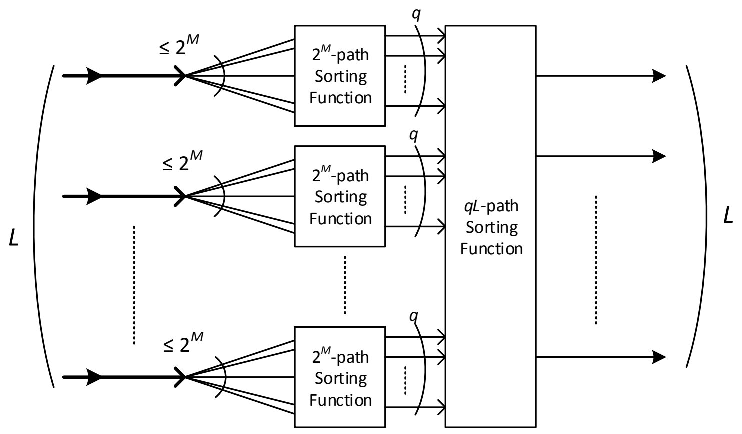


Figure 3.6: Message flow for a two-stage list pruning network.

If $q \geq L$, the L paths found by the two-stage list pruning network are exactly the L most reliable paths among the $2^M L$ paths. When $q < L$, the probability that the L paths found by the two-stage list pruning network is exactly the L most reliable paths among the $2^M L$ paths decreases. This may cause some performance loss. On the other hand, a smaller q leads to a two-stage list pruning network with lower complexity.

Fig. 3.7 shows how different values of q affect the error performance of an SDSCL-8 algorithm with $L = 4$ for a (1024, 480) CRC32-concatenated polar code. When $q = 2$, the SDSCL-8 algorithm shows an FER performance loss of about 0.25 dB at an FER level of 10^{-3}. As shown in Fig. 3.8, for a (2048,1401) CRC32-concatenated polar code, the two stage list-pruning network of $q = 4$ helps to reduce the complexity of the SDSCL-4 decoder without observed performance loss when $L = 8$. When $q = 2$ and

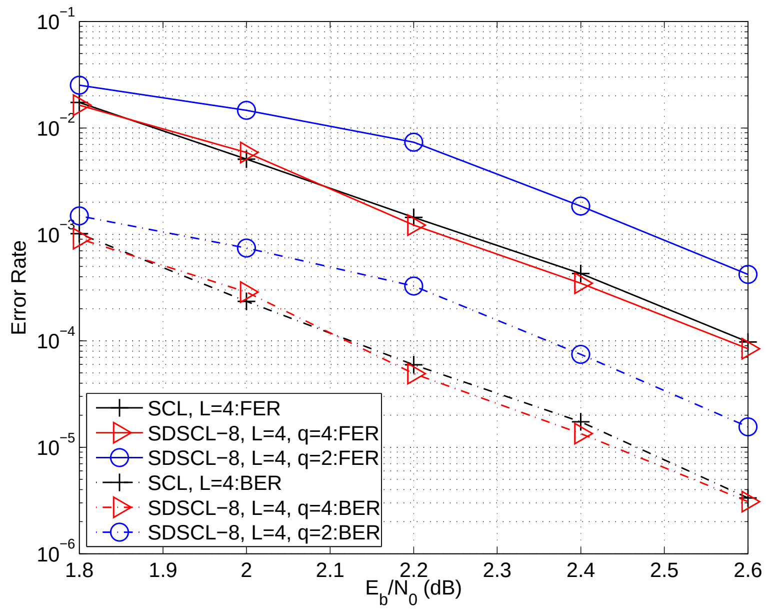


Figure 3.7: Error rates of the SDSCL-8 decoder for a (1024, 480) CRC32-concatenated polar code with $L = 4$.

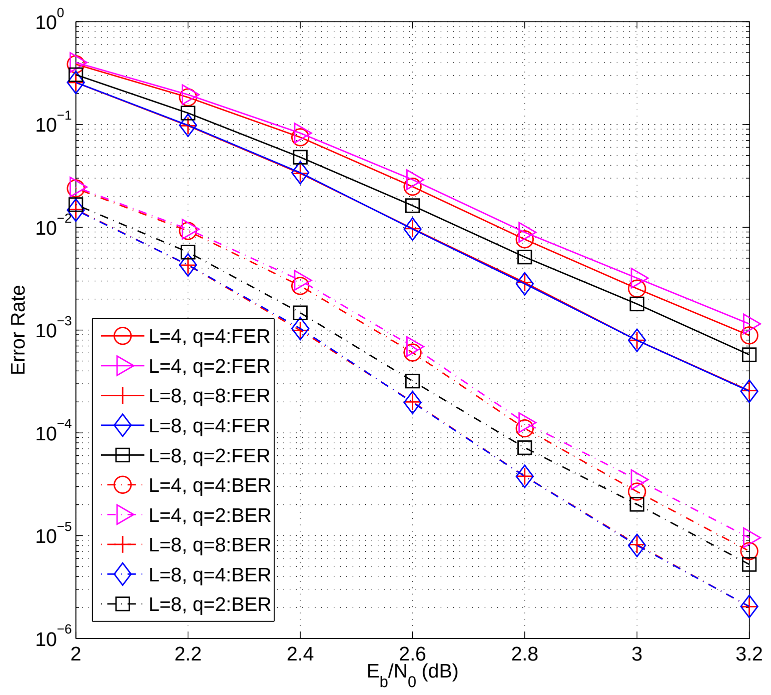


Figure 3.8: Error rates of the SDSCL-4 algorithm for a (2048, 1401) CRC32-concatenated polar code with $L = 4$ and $L = 8$.

$L = 8$, the SDSCL-4 decoder has a performance degradation of about 0.1 dB at an FER level of 10^{-3}, compared with the SDSCL-4 decoder with $q = 8$ and $L = 8$. If $L = 4$, the error performance due to $q = 2$ is negligible.

Therefore, the two-stage list pruning network uses a parameter q to introduce different trade-offs between error performance and complexity.

3.3 Pre-Computation Memory-Saving Technique

Pre-computation technique was first proposed in [54] and can be used to improve processing rate when the number of possible outputs is finite. In [41], the pre-computation technique is used to improve the throughput of the line SC decoder at the expense of increased area. Here, our main purpose is to use the pre-computation technique to reduce the memory required by list decoders because the memory of an SCL decoder to store the channel transition probability becomes a big challenge as the list size and code length increase. Henceforth, this memory saving technique is called the pre-computation memory-saving (PCMS) technique. It is worth noting that this memory-saving technique is *independent* of the decoder architecture and the message representation of SCL decoders.

Let us take the MFG shown in Fig. 3.2 as an example. For stages S_0 and S_1, the numbers of pairs of LLs stored by the list decoder with list size L are 8 and $4L$, respectively. Actually, the outgoing message $W_{2,1}^{(1)}(y_1^2|w_1)$ of the top black solid node in S_1 can only be either $W_{2,1}^{(1)}(y_1^2|0)$ or $W_{2,1}^{(1)}(y_1^2|1)$. The outgoing message $W_{2,1}^{(2)}(y_1^2, w_1|w_2)$ of the top red hollow node in S_1 can only be one of $W_{2,1}^{(2)}(y_1^2, 0|0)$, $W_{2,1}^{(2)}(y_1^2, 0|1)$, $W_{2,1}^{(2)}(y_1^2, 1|0)$, and $W_{2,1}^{(2)}(y_1^2, 1|1)$. Hence, no matter what the list size is, the total number of possible values of outgoing messages of S_1 is $2 \times 4 + 4 \times 4 = 24$. These 24 values provide all the information we need for calculations of further stages. With knowledge of these 24 values, channel LLs are not needed any more.

Generally speaking, the PCMS technique takes advantage of the relationship between messages of S_0 (channel LLs) and outgoing messages of S_1. By storing only all possible outgoing messages of S_1, the PCMS technique helps list decoders save memory.

Let us evaluate the memory saving of the PCMS technique, assuming LL representation is used for the channel transition probability. Without the PCMS technique, a list decoder for a polar code with code length N and list size L stores $(N-2)L + N$ LL pairs. Each pair contains two messages, which are associated with the conditional bit being zero or one. The total number of bits used for LL storage is

$$\begin{aligned} B_{\text{LL}} &= 2\Big(NQ_{ch} + L\sum_{i=1}^{\log N - 1} 2^i(Q_{ch} + \log N - i)\Big) \\ &= 2(L+1)NQ_{ch} + 4L(N - \log N - Q_{ch} - 1), \end{aligned} \tag{3.14}$$

where Q_{ch} denotes the number of bits used for the quantization of channel LLs.

With the PCMS technique, the total number of LL pairs needed by a list decoder is $\frac{N}{2}L + \frac{3}{2}N$. The total number of bits needed for LL storage is:

$$\begin{aligned}B_{\text{PCMS}} =& 2(\frac{N}{2}+N)(Q_{ch}+1)+\\ & 2L\sum_{i=1}^{\log N-2}2^i(Q_{ch}+\log N-i)\\ =& 3N(Q_{ch}+1)+LN(Q_{ch}+3)\\ & -4L(\log N+Q_{ch}+1)\\ =& B_{\text{LL}}-N(LQ_{ch}+L-Q_{ch}-3).\end{aligned} \tag{3.15}$$

Therefore, when LL representation is used for messages, the PCMS technique saves $N(LQ_{ch}+L-Q_{ch}-3)$ bits of memory. The saving is approximately linear with both N and L. Consider a polar code with $N = 1024$, a list decoder with $L = 4$ and $Q_{ch} = 4$. Without the PCMS technique, $B_{\text{LL}} = 57104$. With the PCMS technique, $B_{\text{PCMS}} = 43792$. The PCMS technique helps to save 13312 bits of memory, which is 23% of B_{LL}.

The other advantage of the PCMS technique is that it improves the throughput slightly because the messages of S_1 are already in the memory and do not need to be calculated from channel messages. For example, for a bit-decision semi-parallel SCL decoder with the list size of L, if the code length is N and the number of processing units is P, the latency saving due to the PCMS technique is $\frac{NL}{P}$ clock cycles.

3.4 Implementation of Symbol-Decision SCL Decoders

3.4.1 Architecture of Symbol-Decision SCL Decoders

We propose an architecture of an M-bit symbol-decision SCL decoder shown in Fig. 3.9. It consists of M MPU blocks ($\mathrm{MPU}_0, \mathrm{MPU}_1, \cdots, \mathrm{MPU}_{M-1}$), a list pruning network (LPN), a mask bit generator (MBG), a message-screening block (MSNG), a control block (CNTL), an output-list generator (OLG) and a CRC checker (CRCC).

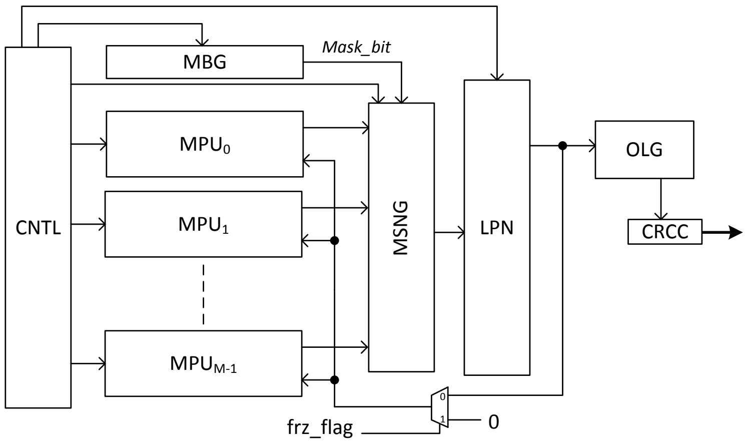


Figure 3.9: Top architecture for an M-bit symbol-decision SCL decoder.

An MPU block calculates messages for B-TRANS and S-COMBS messages and updates the partial-sum network by adopting blocks of the SCL decoder in [44]. The additions of S-COMBS stages are carried out by reusing the same hardware resource used to calculate messages of the B-TRANS stages to reduce area. Compared with the SCL decoder in [44], the MPU has neither path pruning unit nor the CRC checker. The other improvement for the MPU is that the PCMS technique is used here. The architecture of an MPU is shown in Fig. 3.10. Channel messages are not needed any more due to the PCMS technique. L-MEM stores messages corresponding to stages of the MFG. For stage S_1, MSEL selects messages from L-MEM based on partial sum values and/or the type of calculation nodes. PUs are processing units to calculate LL messages. PSUs are used to update partial-sums. ISel selects messages from LMEM or OSel module for the crossbar (CB) module which chooses messages for PUs. OSel outputs messages to L-MEM for intermediate stages and output symbol-wise messages to MSNG.

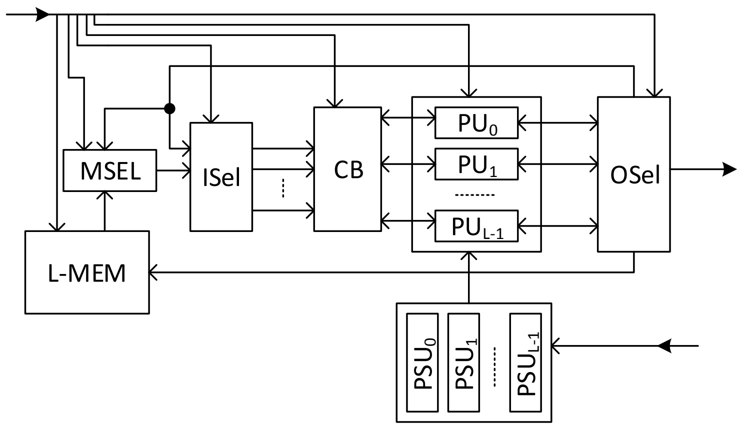


Figure 3.10: Architecture of an MPU.

We take the MFG of Fig. 3.2 as an example to illustrate the function of MSEL. For node f_{21} of path l, $\{W_{2,1}^{(1)}(y_1^2|0), W_{2,1}^{(1)}(y_1^2|1)\}$ and $\{W_{2,1}^{(1)}(y_3^4|0), W_{2,1}^{(1)}(y_3^4|1)\}$ are selected from LMEM by MSEL and output to Isel. For node g_{21} of path l, $\{W_{2,1}^{(2)}(y_1^2, w_{1l}|0),$ $W_{2,1}^{(2)}(y_1^2, w_{1l}|1)\}$ and $\{W_{2,1}^{(2)}(y_3^4, w_{3l}|0), W_{2,1}^{(2)}(y_3^4, w_{3l}|1)\}$ are selected from LMEM. Here, w_{1l} and w_{3l} are the partial sums for w_1 and w_3, respectively, belonging to path l. The detailed information of other blocks in Fig. 3.10 can be found in [44] and will not be discussed in this chapter.

The message-passing scheme in MFG of a polar code is in a serial way, which means that the calculation of a stage depends on the output of its previous stage. The PUs in [44] carry out only the B-TRANS additions. On the other hand, the S-COMBS stages need only additions and a processing unit has four adders. Therefore, in order to save hardware resources, the adders in the processing units are reused to calculate symbol-wise channel transition probabilities, after these processing units finish calculations for the B-TRANS stages. In other words, additions of both the B-TRANS and S-COMBS stages are folded onto the same adders in the processing units. As shown in Fig. 3.11, $c[0]$ and $c[1]$ are outputs for the B-TRANS stages; $d[0], d[1], d[2]$, and $d[3]$ are outputs for the S-COMBS stages.

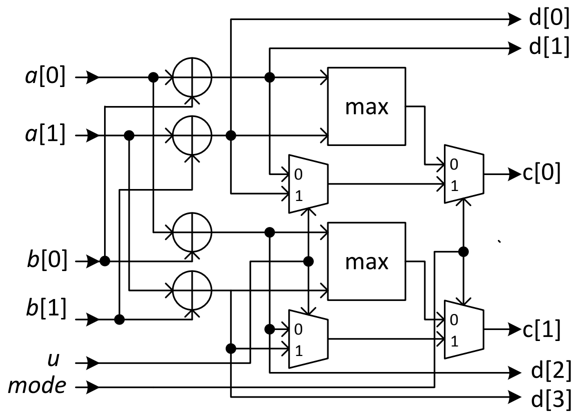


Figure 3.11: Architecture of a processing unit.

MBG provides a mask bit for each path. If there are f $(f \geq 0)$ frozen bits in an M-bit symbol, the number of expanded paths is 2^{M-f} for this symbol. For hardware implementations, we need to consider the worst case and all messages corresponding to 2^M possible paths are calculated. Each path is associated with a mask bit. When some paths are not needed, due to frozen bits, they are turned off by mask bits. Fig. 3.12 shows how to generate the mask bit for path i, where $i = (i_1, i_2, \cdots, i_M) \in \{1, 0\}^M$ $(0 \leq i < 2^M - 1)$ and $\mathbf{b}_j = (b_{j,1}, b_{j,2}, \cdots, b_{j,M})$ is a frozen bit indication vector for u_{jM+1}^{jM+M}. If u_{jM+t} is a frozen bit, $b_{j,t} = 1$. Otherwise, $b_{j,t} = 0$. If $\mathbf{b}_j$ is an all-one vector, all bits of u_{jM+1}^{jM+M} are frozen bits, called an M-bit frozen symbol. If *Mask_bit*$_i$ is 1, u_{jM+1}^{jM+M} cannot be i and the message corresponding to $u_{jM+1}^{jM+M} = i$ is set to 0 in MSNG.

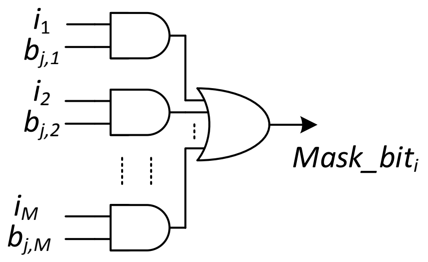


Figure 3.12: Architecture for generating a mask bit.

LPN receives $2^M L$ messages from MSNG, finds the most reliable L paths, and feeds decision results back to the MPUs. Here, we use two different sorting implementations – a folded sorting implementation and a tree sorting implementation – for different designs. The basic unit for these two implementations is a bitonic sorter [57] , which outputs the L max values out of $2L$ inputs and is referred to as BS_L. The folded sorting implementation needs 2^{M-1} BS_Ls $(\mathrm{BS\_L}_0, \mathrm{BS\_L}_1, \cdots, \mathrm{BS\_L}_{2^{M-1}-1})$. The outputs of the $\mathrm{BS\_L}_{2i}$ and the $\mathrm{BS\_L}_{2i+1} (0 \leq i < 2^{M-2})$ are connected with inputs of $\mathrm{BS\_L}_i$ through registers and multiplexers. For the tree sorting implementation with $2^M L$ inputs, $2^M - 1$ BS_Ls are needed. The tree sorting implementation can be divided into M layers. For $0 \leq i < M$, there are 2^i BS_Ls in the i-th layer. Inputs of the BS_Ls of the i-th layer are connected with outputs of the BS_Ls of the $(i+1)$-th layer. Fig. 3.13 and 3.14 show examples of the folded and tree sorting implementations, respectively, for $2^M = 8$.

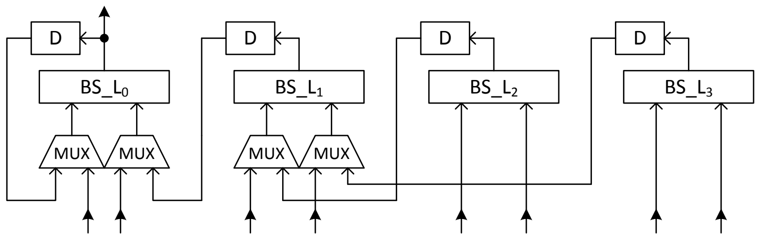


Figure 3.13: Architecture for the folded sorting implementation when $2^M = 8$.

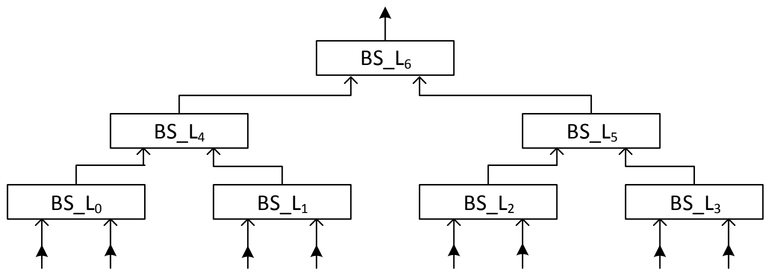


Figure 3.14: Architecture for the tree sorting implementation when $2^M = 8$.

The folded sorting implementation has a smaller area than the tree sorting implementation. However, pipelining can be applied to the tree sorting implementation by inserting registers between layers to improve the throughput of the tree sorting implementation.

For the two-stage list pruning network proposed in Sec. 3.2.4, either the folded sorting implementation or the tree sorting implementation can be used for the 2^M-to-q sorting function and the qL-to-L sorting function.

CNTL provides control signals to the decoder. More details about CNTL will be provided later. The signal frz_flag is an indicator. It is one only when a frozen symbol appears. In this case, all MPUs use zero to update the partial-sums instead of outputs of LPN. Therefore, LPN, MSNG, and calculations for the S-COMBS stages are bypassed. OLG stores the output paths. CRCC checks whether a path satisfies the CRC constraint.

3.4.2 Message Scheduling and Latency Analysis

To improve area efficiency, different scheduling schemes are needed for different number of PUs. To reuse the adders of the processing units, additions of the S-COMBS stages in the MFG must be scheduled properly. Assuming that the number of processing units is P, the total number of adders provided by processing units is $4P$. If $2^M L \leq 4P$, we use a serial scheduling, which means that there is no overlap for the processing units and LPN in terms of the operation time, as shown in Fig. 3.15.

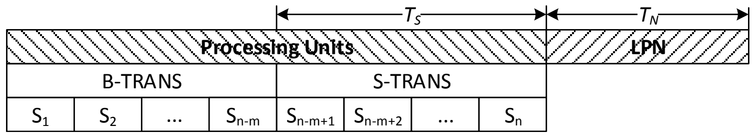


Figure 3.15: Serial scheduling (in clock cycles).

Suppose each addition takes one clock cycle. Then each S-COMBS stage takes one clock cycle to compute messages. Therefore, it takes m clock cycles for the S-COMBS stages to output messages to LPN. To save area, the folded sorting implementation is applied for the serial scheduling.

When $2^M L > 4P \geq 2^{M/2} L$, there are not enough adders to calculate all $2^M L$ messages of stage S_n in one clock cycle, but all $2^{M/2^{n-i}} L$ messages of stage S_i $(n+m-1 \leq i \leq n-1)$ can be calculated in one clock cycle. Without increasing the number of adders, $\frac{2^M L}{4P}$ cycles are needed. In each cycle, $4P$ messages are calculated. To reduce the latency, the overlapping scheduling shown in Fig. 3.16 is used. In clock cycle c_0, the first $4P$ messages come out. In clock cycle c_1, LPN starts. Therefore, MPUs and LPN are working simultaneously for $\frac{2^M L}{4P} - 1$ clock cycles. Here, LPN works in a pipelined way. Hence, the tree sorting implementation is deployed for the overlapping scheduling and a BS_L is connected at the end of the tree sorting implementation in a way shown in Fig. 3.17, where the number on a line represents the number of messages transmitted through the line.

The latency of an M-bit symbol-decision SCL decoder consists of the latency for calculating messages of the B-TRANS stages, the latency for calculating messages of

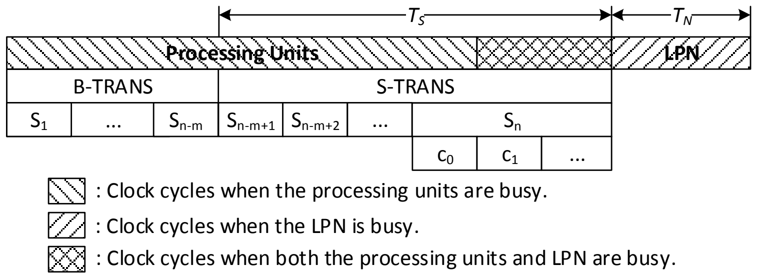

T_S
T_N
Processing Units
LPN
B-TRANS
S-TRANS
S_1
...
S_{n-m}
S_{n-m+1}
S_{n-m+2}
...
S_n
c_0
c_1
...
: Clock cycles when the processing units are busy.
: Clock cycles when the LPN is busy.
: Clock cycles when both the processing units and LPN are busy.

Figure 3.16: Overlapping scheduling (in clock cycles).

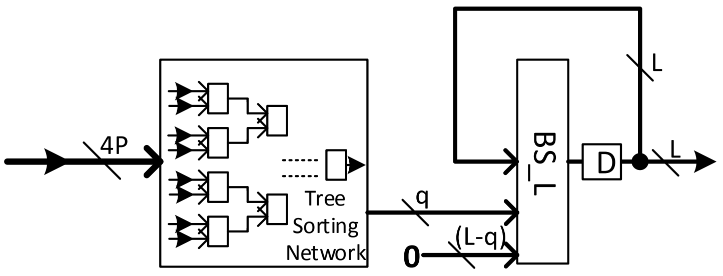


Figure 3.17: A pipelined tree sorting implementation for the overlapping scheduling.

the S-COMBS stages, and the latency of the list pruning network:

- T_B represents the overall number of clock cycles for the calculations of the B-TRANS stages. It is equivalent to the latency of a bit-decision SCL decoder with code length $\frac{N}{M}$ and $\frac{P}{M}$ processing units:

$$T_B = 2\frac{N}{M} + \frac{NL/M}{P/M}\log_2(\frac{NL/M}{4P/M}) - \frac{NL/M}{P/M},$$

  where the third term, $-\frac{NL/M}{P/M}$, is the latency saving by using the PCMS technique.

- T_S represents the number of clock cycles for the calculations of the S-COMBS stages per symbol. If $2^M L \leq 4P$, the number of clock cycles used to calculate messages for the S-COMBS stages is $T_S = m$. When $2^M L > 4P \geq 2^{M/2}L$, $T_S = m - 1 + \lceil \frac{2^M L}{4P} \rceil$. More generally, $T_S \leq \sum_{i=1}^{m} \lceil \frac{2^{2^i} L}{4P} \rceil$.

- T_N represents the number of extra clock cycles per symbol needed by LPN to finish the list pruning after all messages of stage S_n are calculated. T_N is determined by the detailed implementation.

Here, T_S and T_N are needed by only symbols including information bit(s). For frozen symbols, LPN and the calculation of the S-COMBS stages are bypassed. Hence, the latency of the symbol-decision SCL decoder is:

$$\begin{aligned} T(M) &= (1-\gamma)\frac{N}{M}(T_S+T_N)+T_B \\ &= (1-\gamma)\frac{N}{M}(T_S+T_N)+2\frac{N}{M}+\frac{NL}{P}\log_2(\frac{NL}{8P}), \end{aligned} \tag{3.16}$$

where γ is a ratio of the number of frozen symbols to $\frac{N}{M}$.

Table 3.2 shows latencies (in clock cycles) of different decoders to decode a (1024, 480) CRC32-concatenated polar code with 64 processing units and $L = 4$. We assume a BS_L needs one clock cycle to find the four maximum values out of eight values. For $M = 2$ and $M = 4$, a folded sorting implementation and the serial scheduling are used. For $M = 8$, a pipelined tree sorting implementation and the overlapped scheduling are applied. For $M = 8$ and $q = 2$, the basic unit in the tree sorting implementation is to find the two maximum values out of eight values, which needs one clock cycles. Therefore, $T_N = 4$ when $M = 8$ and $q = 2$.

It is claimed in [2] that the M-bit SDSCL decoder could have M times faster decoding speed than the bit-decision SCL decoder. In practice, this is unrealistic as shown by Table 3.2. Let us review (3.16) again. For a fair comparison, suppose the MPUs of the M-bit SDSCL decoder has the same architecture as the conventional SCL decoder. Then a conventional SCL decoder with the PCMS technique has a latency of

Table 3.2: Latencies of different decoders for a (1024, 480) CRC32-concatenated polar code with 64 processing units and $L = 4$.

Decoder	γ	T_S	T_N	q	Latency (# of cycles)
SDSCL-2	0.445	1	2	4	2069
SDSCL-4	0.395	2	4	4	1634
SDSCL-8	0.344	6	7	4	1540
SDSCL-8	0.344	6	4	2	1288

$T(1) = 2N + \frac{NL}{P}\log_2\frac{NL}{8P}$. The decoding speed gain of the M-bit SDSCL decoder is

$$\begin{aligned}\frac{T(1)}{T(M)} &= \frac{2N + \frac{NL}{P}\log_2\frac{NL}{8P}}{(1-\gamma)\frac{N}{M}(T_S+T_N) + 2\frac{N}{M} + \frac{NL}{P}\log_2(\frac{NL}{8P})} \\ &= M - \frac{(1-\gamma)N(T_S+T_N) + (M-1)\frac{NL}{P}\log_2\frac{NL}{8P}}{(1-\gamma)\frac{N}{M}(T_S+T_N) + 2\frac{N}{M} + \frac{NL}{P}\log_2\frac{NL}{8P}}\end{aligned} \tag{3.17}$$

To be exactly M $(M > 1)$ times faster, $(1-\gamma)\frac{N}{M}(T_S+T_N) + (M-1)\frac{NL}{P}\log_2\frac{NL}{8P}$ needs to be small. For $NL > 8P$, $\frac{T(1)}{T(M)} < M$, because $T_S > 0$ and $T_N > 0$. For $NL = 8P$, $T_S = T_N = 0$ should be satisfied, which means that the calculation of the symbol-wise channel transition probability and the list pruning procedure do ***NOT*** take any clock cycle. This is impractical, since T_S and T_N cannot be zero in a practical design. If $NL < 8P$ and $P \leq NL$, to achieve M times faster, $(T_R + T_N) = \frac{(M-1)L\log_2\frac{8P}{NL}}{(1-\gamma)P} < \frac{5(M-1)}{(1-\gamma)N}$. Usually, $(1-\gamma)N >> 5(M-1)$. Therefore, the statement about the decoding speed gain in [2] is too idealistic to be achieved in practice because practical implementations need extra cycles to calculate symbol-wise channel transition probabilities and to perform the list pruning function.

3.4.3 Control Logic

The control part CNTL is similar to that of [36] because a semi-parallel architecture is used in our decoders. Some necessary modifications are made for the symbol-decision decoder.

Basically, CNTL mainly consists of finite state machines and counters to generate control signals for other blocks. It provides the information of frozen bits and accomplishes another three main jobs.

- It controls MPU blocks to calculates messages for B-TRANS messages.
- It controls MPU blocks to calculates messages for S-TRANS messages.
- It controls the list pruning network to generate L list candidates.

The first job is almost the same as the control logic used in [36] except for additional control signals needed by the PCMS technique. Based on different message scheduling schemes, different controllers are used to accomplish the second and third jobs.

There are several counters in CNTL, such as the current decoded symbol index counter, the current stage index counter, the current message index counter, and so on.

Main finite state machines (FSMs) for different scheduling schemes are shown in Fig. 3.18.

For the serial scheduling, the decoder is enabled by signal "start" and goes to state "B-TRANS" to calculate B-TRANS messages. If the symbol is a frozen symbol, the decoder stays in state "B-TRANS" and the current decoded symbol index increases by

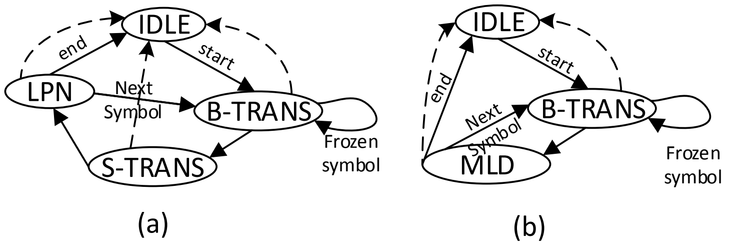


Figure 3.18: Finite state machines for different scheduling schemes, (a) serial scheduling (b) overlapping scheduling.

1. Otherwise, the decoder goes to states "S-TRANS" and "LPN" sequentially. After L candidates are founded, the decoder goes back to state "B-TRANS" and starts to deal with the next symbol. Meanwhile, the current decoded symbol index counter goes up by 1. After all symbols are decoded, the decoder goes back to state "IDLE" and waits for the next "start" to decode the next codeword. There are three dash arrows in Fig. 3.18 indicating that whenever a reset signal is received by the decoder, the decoder goes back to state "IDLE" immediately.

For the overlapping scheduling, the FSM has only three states. The decoder calculates S-TRANS messages and perform list pruning function in state "MLD". More counters are needed when the decoder adopts the overlapping scheduling. For examples: a counter is needed to generate a signal indicating when LPN starts to work.

Actually, the control logic is a very small part for polar decoders. The ratio of the area of the control logic over the entire decoder area is less than 0.5%.

3.4.4 Synthesis results

To implement the proposed symbol-decision SCL decoder, we consider only $M = 2, 4$ and 8. For $M \geq 16$, it is impractical to build list pruning networks. For example, for the worst case of $M = 16$ that all the bits of a symbol are information bits, there are $2^{16}L = 65536L$ paths. Even if $L = 1$, to find the maximum value among 65536 values still needs a huge amount of hardware resources and leads to a large latency.

Table 3.3: Synthesis results of proposed decoders with $L = 4$ for the aforementioned (1024, 480) CRC32-concatenated polar code.

	Our Proposed Architectures					
Algorithm	Symbol-decision SCL					
M	2	4	8 ($q = 4$)		8 ($q = 2$)	
Message Type	LL					
Clock Rate (MHz)	500					
Latency (us)	4.14	3.27	3.08	3.21**	2.58	2.70**
Throughput (Mbps)	247	313	332	319**	398	379**
Area (mm^2)	1.126	1.209	1.669	1.782**	1.403	1.519**
Area eff. (Mb/s/mm^2)	219.4	259.2	199.2	179.1**	283.3	249.3**

** The design is without the PCMS technique.

Table 3.4: Synthesis results of existing decoders with $L = 4$ for the aforementioned (1024, 480) CRC32-concatenated polar code.

	[3]				[44]	[43]‡	[43]†	[45]
Algorithm	Multi-bit SCL				Bit-decision SCL			
M	2		4		N/A			
Message Type	LL							LLR
Clock Rate (MHz)	525	379*	400	289*	500	694	314	794
Latency (us)	3.89	5.39*	2.56	3.53*	5.63	4.06	8.25	3.34
Throughput (Mbps)	262	189*	401	289*	182	252	124	307
Area (mm^2)	1.98	3.79*	2.14	4.10*	1.099	2.197	3.53	1.78
Area eff. (Mb/s/mm^2)	132.3	49.9*	187.3	70.6*	165.6	114.7	35.1	172

† The synthesis result in [43] is based on a UMC 90nm CMOS technology.
‡ The synthesis result is provided by the authors of [43] based on a TSMC 90nm CMOS technology.
* Original synthesis results in [3] are based on an ST 65nm CMOS technology. For a fair comparison, synthesis results scaled to a 90nm technology are used in the comparison.

Our implementations assume $L = 4$ and 64 processing units. LL messages are used in our designs. Channel LL messages are quantized with 4 bits. A (1024, 480) CRC32-concatenated polar code is used. The synthesis tool is Cadence RTL compiler. The process technology is TSMC 90nm CMOS technology. Our proposed architectures are compared with state-of-the-art SCL architectures, in [3,43–45], of both bit-decision and multi-bit algorithms. The synthesis results in [45] and [44] are also based on a TSMC 90nm CMOS technology. The original synthesis results of [43] and [3] are based on a UMC 90nm and ST 65nm CMOS technologies, respectively.

The synthesis results shown in Tables 3.3 and 3.4, demonstrate that our symbol-decision SCL polar decoders have higher area efficiencies than the SCL decoders in [43], [44], [3], and [45]. The SCL decoders in [3,43,45] have higher clock rates than our designs, because they use registers as storage units. However, in our designs, register files are used.

The SDSCL-8 decoders provide a higher throughput and a smaller latency than the SDSCL-2 and SDSCL-4 decoders, and occupy larger areas. However the improvements on the throughput and latency are not linear in the symbol size.

Compared with the bit-decision SCL decoder in [44], the increase of areas of symbol-decision SCL decoders is mainly due to sorting networks because the adders of processing units are reused to calculate both bit- and symbol-wise channel transition probabilities.

Parameter q provides trade-offs between complexity and performance. For the SDSCL-2 and SDSCL-4 decocers, $q = 4$. For the SDSCL-4 decoder, because the sorting network of the SDSCL-4 decoder is only 0.073 mm^2, there is no need to shrink q further. For the SDSCL-8 decoders, both $q = 4$ and $q = 2$ are tested. When $q = 4$, the area of the sorting network is 0.454 mm^2. However, when $q = 2$, the sorting network occupies 0.196 mm^2 which is less than a half of that of $q = 4$. A smaller q does help the SDSCL-8 decoder achieve a higher throughput, a smaller latency, a smaller area, and a higher area efficiency, but it also introduces an FER performance loss of 0.25 dB to the SDSCL-8 decoder at an FER level of 10^{-3} as shown in Fig. 3.7.

Moreover, synthesis results for SDSCL-8 decoders without the PCMS technique are also provided. The PCMS technique helps the SDSCL-8 decoders gain an area saving of about 0.12 mm^2.

LL messages are used in our designs presented herein. If LLR messages are used in our decoder architecture as in [45], our symbol-decision SCL decoders will have better area efficiencies than our current designs, because the memory requirement for LLR messages are smaller than that for LL messages [45].

3.5 Summary

In this chapter, we use the symbol-based recursive channel combination to calculate the symbol-wise channel transition probability. We show that based on the LL representation of the transition probability, this recursive procedure needs fewer additions than the method used in [1–3]. Furthermore, a two-stage list pruning network is proposed to simplify the L-path finding problem. We use the PCMS technique to reduce the memory requirement for list decoders. By applying the PCMS technique, we design an efficient architecture for symbol-decision SCL decoders. Specifically, we introduce two scheduling schemes to perform the hardware sharing. A folded sorting implementation and tree sorting implementation are also discussed. We also implement symbol-decision SCL polar decoders for two-bit, four-bit and eight-bit, respectively, with a list size of four. Our synthesis results show that symbol-decision SCL polar decoders outperform existing SCL polar decoders in terms of the area efficiency. Our proposed methods and architecture provide a range of trade-offs between area, throughput and area efficiency.

Chapter 4

Multi-Mode Area-Efficient SCL Polar Decoder

Maximum likelihood (ML) decoding algorithms [29–31] can be used to decode polar codes, but their complexity can be prohibitively high. Compared with ML decoding algorithms, the successive cancellation (SC) decoding algorithm [24] has a lower complexity at the cost of sub-optimal performance. To improve the performance of the SC algorithm, the SC list (SCL) decoding algorithm [33] and the CRC-aided SCL (CA-SCL) algorithm [26, 33] were proposed. A key drawback of the SC, SCL and CA-SCL algorithms is their long decoding latency and low decoding throughput, as these algorithms deal with only one bit at a time.

To reduce decoding latency and improve throughput of an SC polar decoder, several algorithms [1, 2, 40, 58] were proposed to deal with several bits at a time instead of only one bit by using ML decoding units, which calculate symbol-wise channel transition probabilities and make hard decisions for several bits at a time. Based on the SC algorithm, the parallel SC [2], hybrid ML-SC [1], ML simplified SC (ML-SSC) [40] and fast ML-SSC [58] algorithms were proposed. The basic difference of ML decoding units between these four algorithms is that hybrid ML-SC [1] and fast ML-SSC [58] take advantage of the distribution of frozen bits to reduce complexity, but neither parallel SC [2] nor ML-SSC [40] algorithms do so.

ML decoding units in [3, 59–63] are also used to improve throughput of SCL-based decoders and to reduce decoding latency. Instead of making hard decisions in SC-based algorithms, an ML decoding unit for SCL-based algorithms calculates symbol-wise channel transition probabilities and performs path expansion and pruning. None of these SCL-based algorithms takes advantage of the distribution of frozen bits to reduce complexity of ML decoding units. Therefore, ML decoding units in these SCL-based algorithms have high complexities. For example, when the list size is four and the symbol size is eight, the ML decoding unit accounts for 27% of the overall decoder area in [60]. In [62], when the code length is 1024, the area of an ML decoding unit takes up as much as 62% of the overall decoder area.

In this chapter, we first propose a low-complexity approximate ML (AML) decoding unit by utilizing the distribution of frozen bits of polar codes and then propose a multi-mode SCL (MM-SCL) polar decoder to support variable throughput and latency. Our main contributions are:

- The divide-and-conquer method in [1] is applied in the probability domain to simplify the ML unit for SC-based algorithms. By extending this idea, a divide-and-conquer AML decoding unit for SCL-based algorithms is proposed by considering the distribution of frozen bits. It has greatly smaller computational complexity than those of existing ML decoding units for SCL-based algorithms, and has negligible performance loss when properly configured.

- The distribution of frozen bits of polar codes is analyzed. We show that there are only a small number of frozen-location patterns for polar codes constructed by a method proposed by Arıkan in [64] and a method in [65].

- Since only a small number of frozen-location patterns exist in polar codes, the divide-and-conquer AML decoding unit for SCL-based algorithms is simplified further. A low-complexity hardware implementation for the simplified divide-and-conquer AML decoding unit, the LC-AML decoding unit, is proposed. Synthesis results show that by taking advantage of a small number of frozen-location patterns, our CA-SCL decoder with the LC-AML unit has a better throughput-area efficiency than existing SCL decoders, while working for all channel conditions.

- An MM-SCL polar decoder is also proposed. This decoder supports SCL algorithms with different list sizes and parallelism. When a high throughput or small latency is needed, the MM-SCL decoder decodes multiple received words in parallel with a small list size. If error performance is of higher priority, the MM-SCL decoder switches to a mode with a greater list size. Therefore, the MM-SCL polar decoder provides a flexible tradeoff between latency, throughput and performance at the expense of small overhead.

Our proposed divide-and-conquer AML decoding unit for SCL-based algorithm is a nontrivial extension of the method for SC-based algorithm in [1]. However, by investigating the distribution of frozen bits of polar codes, we reduce the complexity of the ML decoding unit further. Existing ML decoding units for SCL decoders [3,59–62] perform list pruning after all the symbol-wise channel transition probabilities are calculated, whereas the proposed LC-AML decoding unit sorts intermediate results generated by the recursive channel combination method [59], leading to a reduced number of symbol-wise channel transition probabilities dealt with by list pruning. Hence, the proposed LC-AML decoding unit has a much smaller complexity. The performance degradation due to the proposed LC-AML is the same as that in [59, 60]. Although the ML decoding unit in [3] has no performance degradation, its complexity grows quickly as the list size and symbol size increase. The performance degradation of the ML decoding unit in [61, 62] depends on its design parameters.

Many applications, such as modern wireless or wireline communication system, require variable data rate transmission and have stringent latency requirements. As a potential candidate of FEC technique for future communication systems, a polar decoder supporting variable data rate and variable decoding latency is desired. Unfortunately, existing polar decoders provide only fixed latency and throughput (data rate). To the best of our knowledge, the proposed MM-SCL decoder is the *first* polar decoder with variable throughput and decoding latency given a polar code.

The rest of this chapter is organized as follows. In Section 4.1, construction methods for polar codes and existing ML decoding units are reviewed. In Section 4.2, the divide-and-conquer method is first applied to the ML unit of SC-based algorithms in the probability domain. Then, the divide-and-conquer AML decoding unit for SCL-based algorithms is proposed, and its computational complexity is also analyzed. In Section 4.3, frozen-location patterns for polar codes are investigated. In Section 4.4, a hardware design of the LC-AML unit is proposed, and an area-efficient CA-SCL decoder with the LC-AML unit is implemented as well. The hardware implementation and synthesis results for the area-efficient SCL decoder are also presented in this section. In Section 4.5, the MM-SCL decoder, its hardware implementation and synthesis results are presented. Finally, some conclusions are provided in Section 4.6.

4.1 Preliminaries

4.1.1 Construction Methods of Polar Codes

An essential problem for constructing a polar code is to determine the locations of frozen bits (elements of $\mathcal{A}^c$). For the BEC with an erasure probability ϵ $(0 < \epsilon < 1)$, assuming $z_{0,1} = \epsilon$, the following recursions [64] are used to construct an (N, K) polar code, where $N = 2^n$ and $0 < K < N$:

$$z_{i,2j-1} = 2z_{i-1,j} - z_{i-1,j}^2, \tag{4.1}$$

$$z_{i,2j} = z_{i-1,j}^2, \tag{4.2}$$

where $1 \leq i \leq n$. Then $\mathcal{A}^c$ is chosen such that $\sum_{j \in \mathcal{A}^c} z_{n,j}$ is maximal and $|\mathcal{A}^c| = N - K$.

For the AWGN channel and a given noise variance σ^2, let $z_{0,1} = \frac{2}{\sigma^2}$, the following recursive method [65] based on Gaussian approximation is used for $1 \leq i \leq n$:

$$z_{i,2j-1} = \tau^{-1}\Big(1 - \big(1 - \tau(z_{i-1,j})\big)^2\Big), \tag{4.3}$$

$$z_{i,2j} = 2z_{i-1,j}, \tag{4.4}$$

where

$$\tau(x) = \begin{cases} e^{-0.4527x^{0.86}+0.0218}, & 0 < x < 10, \\ \sqrt{\frac{\pi}{x}} e^{-\frac{x}{4}} (1 - \frac{10}{7x}) & x \geq 10. \end{cases} \tag{4.5}$$

In this case, $\mathcal{A}^c$ is chosen such that $\sum_{j \in \mathcal{A}^c} z_{n,j}$ is minimal and $|\mathcal{A}^c| = N - K$.

4.1.2 Existing ML Decoding Units for Polar Decoders

When $\mathbf{x} = \mathbf{u} B_N F^{\otimes n}$ is transmitted, suppose the received word is $\mathbf{y} = y_1^N$ and the symbol size is $M = 2^m$. A symbol-decision [59] ML decoding unit first calculates symbol-wise channel transition probabilities, $\Pr(\mathbf{y}, \hat{u}_1^{jM} | u_{jM+1}^{jM+M})$ $(0 \leq j < \frac{N}{M})$, then makes a symbol-wise ML decision for SC-based decoders or chooses the L most reliable paths for SCL-based decoders. Here, $\hat{u}_1^{jM}$ is the previously estimated bits.

There are three methods to calculate the symbol-wise channel transition probabilities, and all of them do not take advantage of the distribution of frozen bits. The first [2,3,40,61] is based on an M-element product of bit-wise channel transition probabilities, called directing mapping method (DMM):

$$\begin{aligned} &\Pr(\mathbf{y}, \hat{u}_1^{iM-M} | u_{iM-M+1}^{iM}) = \\ &\prod_{j=0}^{M-1} \Pr(y_{j\frac{N}{M}+1}^{(j+1)\frac{N}{M}}, \hat{w}_{1+j\frac{N}{M}}^{(i-1)+j\frac{N}{M}} | w_{i+j\frac{N}{M}}), \end{aligned} \tag{4.6}$$

where $u_{iM-M+1}^{iM} = (w_i, w_{i+\frac{N}{M}}, \cdots, w_{i+N-\frac{N}{M}}) B_M F^{\otimes m}$ for $1 \leq i \leq \frac{N}{M}$, and $\hat{w}_{1+j\frac{N}{M}}^{(i-1)+j\frac{N}{M}}$ is the previously estimated bit vector of $w_{1+j\frac{N}{M}}^{(i-1)+j\frac{N}{M}}$.

The second [59], called as the recursive channel combination (RCC) method, is based on a product of symbol-wise channel transition probabilities recursively,

$$\begin{aligned}\Pr(y_1^{2\Lambda}, \hat{u}_1^{i\Phi}|u_{i\Phi+1}^{i\Phi+\Phi}) =&\Pr(y_1^{\Lambda}, \hat{u}_{1,o}^{i\Phi} \oplus \hat{u}_{1,e}^{i\Phi}|u_{i\Phi+1,o}^{i\Phi+\Phi} \oplus u_{i\Phi+1,e}^{i\Phi+\Phi}) \\ &\cdot \Pr(y_{\Lambda+1}^{2\Lambda}, \hat{u}_{1,e}^{i\Phi}|u_{i\Phi+1,e}^{i\Phi+\Phi}),\end{aligned} \tag{4.7}$$

where $1 \leq \phi \leq m$, $0 \leq \lambda < n$, $\Lambda = 2^{\lambda}$, and $\Phi = 2^{\phi}$.

The third [62] is a hybrid method by applying the DMM first and then the RCC method, referred to as the DRH method.

Based on the distribution of frozen bits, some data symbols in [58] are considered as some special constituent codes, such as repetition codes and single-parity-check nodes. Different methods were proposed to deal with different constituent codes.

Furthermore, an ML decoding unit in [1] with the divide-and-conquer method was proposed for SC algorithms based on an empirical assumption [1]:

Assumption 4.1.1. For a well designed polar code, there is no such case that u_{2i-1} is an information bit and u_{2i} is a frozen bit, for any $1 \leq i \leq \frac{N}{2}$.

Based on this assumption and the divide-and-conquer method, a simplified ML unit was proposed in [1]. Moreover, a recursive way of the divide-and-conquer method was proposed in [1], but it is not suitable for hardware implementation since it is for a large symbol size, which has a very high complexity for hardware implementation.

How to take advantage of frozen-location patterns to reduce complexity of ML decoding units has been discussed in [58] and [1] for SC-based algorithms, but it has not been investigated yet for SCL-based algorithms.

4.2 Divide-and-Conquer AML Decoding Unit

The simplified ML unit in [1] is based on the Euclidean distance since an AWGN channel is assumed. Here, we first apply the divide-and-conquer method in the probability domain and reformulate the ML unit for SC-based algorithms. This simplified ML unit in the probability domain is slightly more general than that in [1], because it is applicable to both AWGN channels and other channels. We then extend the simplified ML unit in the probability domain to SCL-based algorithms.

4.2.1 Reformulation of the Divide-and-Conquer ML Unit for SC-based Algorithms [1] in the Probability Domain

For the ease of discussion, a string vector $\mathbb{S}_a^b =$'$\mathcal{S}_a \cdots \mathcal{S}_b$' (for $1 \leq a \leq b \leq N$) is introduced to represent a frozen-location pattern of symbol u_a^b. If u_j $(a \leq j \leq b)$ is an information bit, S_j is denoted as '$\mathcal{D}$'. Otherwise, S_j as '$\mathcal{F}$'. Consider a toy example of a four-bit symbol u_1^4. Assuming u_1 and u_3 are frozen bits, and u_2 and u_4 are information bits. Then the frozen-location pattern $\mathbb{S}_1^4$ of u_1^4 is '$\mathcal{F}\mathcal{D}\mathcal{F}\mathcal{D}$'. Obviously, for all M-bit symbols, there are 2^M frozen-location patterns.

Based on Assumption 1, $u_{jM+1}^{jM+M} (0 \leq j < \frac{N}{M})$ can be divided into $\frac{M}{2}$ pairs, $u_{jM+2i-1}$ and u_{jM+2i} for $1 \leq i \leq \frac{M}{2}$. In theory, any pair of $u_{jM+2i-1}$ and u_{jM+2i} have four possibilities. '$\mathcal{F}\mathcal{F}$' is trivial. Under Assumption 1, '$\mathcal{D}\mathcal{F}$' is not possible. Hence, in [1], only two remaining possibilities are considered: '$\mathcal{F}\mathcal{D}$' and '$\mathcal{D}\mathcal{D}$'. Let $\Omega_{01}^{(j)}$ represent the index set of i that $u_{jM+2i-1}^{jM+2i}$ is '$\mathcal{F}\mathcal{D}$'. $\Omega_{11}^{(j)}$ represents the index set of i that $u_{jM+2i-1}^{jM+2i}$ is '$\mathcal{D}\mathcal{D}$'.

For SC-based algorithms, the maximum of $2^{|\Omega_{01}^{(j)}|+2|\Omega_{11}^{(j)}|}$ values of $T(u_{jM+1}^{jM+M}) \triangleq \Pr(\mathbf{y}, \hat{u}_1^{jM}|u_{jM+1}^{jM+M})$ needs to be found. Based on the RCC method [59], $T(u_{jM+1}^{jM+M}) = T_1(v_{\frac{jM}{2}+1}^{\frac{jM+M}{2}}) \times T_2(u_{jM+1,e}^{jM+M})$, where $v_1^{\frac{N}{2}} \triangleq u_{1,o}^N \oplus u_{1,e}^N$, $T_1(v_{\frac{jM}{2}+1}^{\frac{jM+M}{2}}) \triangleq \Pr(y_1^{\frac{N}{2}}, \hat{v}_1^{\frac{jM}{2}}|v_{\frac{jM}{2}+1}^{\frac{jM+M}{2}})$, and $T_2(u_{jM+1,e}^{jM+M}) \triangleq \Pr(y_{\frac{N}{2}+1}^N, \hat{u}_{1,e}^{jM}|u_{jM+1,e}^{jM+M})$. The possible values of $T(u_{jM+1}^{jM+M})$ can be divided into $2^{|\Omega_{01}^{(j)}|}$ groups based on $\Omega_{01}^{(j)}$, each with $2^{2|\Omega_{11}^{(j)}|}$ values. In each group, for $i \in \Omega_{11}^{(j)}$, since $v_{\frac{jM}{2}+i}$ and u_{jM+2i} are independent, $\max(T) = \max(T_1)\max(T_2)$. Then, the maximum of $2^{|\Omega_{01}^{(j)}|}$ values generated in the previous step is found. Therefore, if $\Omega_{01}^{(j)} = \varnothing$,

$$\max_{u_{jM+1}^{jM+M}}(T) = \max_{\substack{u_{jM+2i-1}^{jM+2i} \\ i\in\Omega_{11}^{(j)}}}(T_1) \times \max_{\substack{u_{jM+2i} \\ i\in\Omega_{11}^{(j)}}}(T_2); \tag{4.8}$$

otherwise,

$$\max_{u_{jM+1}^{jM+M}}(T) = \max_{\substack{u_{jM+2i} \\ i\in\Omega_{01}^{(j)}}}\left(\max_{\substack{u_{jM+2i-1}^{jM+2i} \\ i\in\Omega_{11}^{(j)}}}(T_1) \times \max_{\substack{u_{jM+2i} \\ i\in\Omega_{11}^{(j)}}}(T_2)\right). \tag{4.9}$$

Under Assumption 1, considering Eq. (4.8), if $\Omega_{01}^{(j)} = \varnothing$, the maximal value of T is just a product of the maximal value of T_1 and the maximal value of T_2. For example, Fig. 4.1(b) shows an example of frozen-location pattern '$\mathcal{DDDD}$' which has $\Omega_{01}^{(0)} = \varnothing$, when $M = 4$ and $N = 4$. If $\Omega_{01}^{(j)}$ is not empty, for any $i \in \Omega_{01}^{(j)}$, $v_{\frac{jM}{2}+i} = u_{jM+2i}$.

$$\Pr(y_1^4|u_1^4) = \Pr(y_1^2|v_1,v_2) \times \Pr(y_3^4|u_2,u_4) \text{ where } v_1 = u_1 \oplus u_2 \text{ and } v_2 = u_3 \oplus u_4$$

(a)

$$\mathcal{DDDD}\text{:} \quad \max\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \vdots \\ \Pr(y_1^4|1110), \\ \Pr(y_1^4|1111) \end{pmatrix} = \max\begin{pmatrix} \Pr(y_1^2|00), \\ \Pr(y_1^2|01), \\ \Pr(y_1^2|10), \\ \Pr(y_1^2|11) \end{pmatrix} \times \max\begin{pmatrix} \Pr(y_3^4|00), \\ \Pr(y_3^4|01), \\ \Pr(y_3^4|10), \\ \Pr(y_3^4|11) \end{pmatrix}$$

(b)

$\mathcal{FDDD}$:

$$\max\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \vdots \\ \Pr(y_1^4|0111) \end{pmatrix} = \max\begin{pmatrix} \max\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \Pr(y_1^4|0010), \\ \Pr(y_1^4|0011) \end{pmatrix}, \\ \max\begin{pmatrix} \Pr(y_1^4|0100), \\ \Pr(y_1^4|0101), \\ \Pr(y_1^4|0110), \\ \Pr(y_1^4|0111) \end{pmatrix}, \end{pmatrix} = \max\begin{pmatrix} \max\begin{pmatrix} \Pr(y_1^2|00), \\ \Pr(y_1^2|01) \end{pmatrix} \times \max\begin{pmatrix} \Pr(y_3^4|00), \\ \Pr(y_3^4|01) \end{pmatrix}, \\ \max\begin{pmatrix} \Pr(y_1^2|10), \\ \Pr(y_1^2|11) \end{pmatrix} \times \max\begin{pmatrix} \Pr(y_3^4|10), \\ \Pr(y_3^4|11) \end{pmatrix} \end{pmatrix}$$

(c)

For SC

$$\mathcal{DDDD}\text{:} \quad \max{}_2\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \vdots \\ \Pr(y_1^4|1110), \\ \Pr(y_1^4|1111) \end{pmatrix} = \max{}_2\left(\max{}_2\begin{pmatrix} \Pr(y_1^2|00), \\ \Pr(y_1^2|01), \\ \Pr(y_1^2|10), \\ \Pr(y_1^2|11) \end{pmatrix} \boxtimes \max{}_2\begin{pmatrix} \Pr(y_3^4|00), \\ \Pr(y_3^4|01), \\ \Pr(y_3^4|10), \\ \Pr(y_3^4|11) \end{pmatrix}\right)$$

(d)

$\mathcal{FDDD}$:

$$\max{}_2\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \vdots \\ \Pr(y_1^4|0111) \end{pmatrix} = \max{}_2\begin{pmatrix} \max{}_2\begin{pmatrix} \Pr(y_1^4|0000), \\ \Pr(y_1^4|0001), \\ \Pr(y_1^4|0010), \\ \Pr(y_1^4|0011) \end{pmatrix}, \\ \max{}_2\begin{pmatrix} \Pr(y_1^4|0100), \\ \Pr(y_1^4|0101), \\ \Pr(y_1^4|0110), \\ \Pr(y_1^4|0111) \end{pmatrix}, \end{pmatrix} = \max{}_2\begin{pmatrix} \max{}_2\begin{pmatrix} \Pr(y_1^2|00), \\ \Pr(y_1^2|01) \end{pmatrix} \boxtimes \max{}_2\begin{pmatrix} \Pr(y_3^4|00), \\ \Pr(y_3^4|01) \end{pmatrix}, \\ \max{}_2\begin{pmatrix} \Pr(y_1^2|10), \\ \Pr(y_1^2|11) \end{pmatrix} \boxtimes \max{}_2\begin{pmatrix} \Pr(y_3^4|10), \\ \Pr(y_3^4|11) \end{pmatrix} \end{pmatrix}$$

(e)

For SCL

Figure 4.1: Examples for calculating $\max()$ and $\max_2()$ functions when $M = 4$, $N = 4$, $q = 2$, and $j = 0$, (a) the calculation rule for $\Pr(y_1^4|u_1^4)$, (b) the calculation of $\max(\Pr(y_1^4|u_1^4))$ in [1] when the frozen-location pattern is '$\mathcal{DDDD}$' , (c) the calculation of $\max(\Pr(y_1^4|u_1^4))$ in [1] when the frozen-location pattern is '$\mathcal{FDDD}$', (d) the calculation of $\max_2(\Pr(\mathrm{y}_1^4|\mathrm{u}_1^4))$ when the frozen-location pattern is '$\mathcal{DDDD}$' , and (e) the calculation of $\max_2(\Pr(\mathrm{y}_1^4|\mathrm{u}_1^4))$ when the frozen-location pattern is '$\mathcal{FDDD}$'.

4.2.2 Divide-and-Conquer AML Decoding Unit for SCL-based Algorithms

Extending the idea in Eqs. (4.8) and (4.9), we propose a divide-and-conquer AML decoding method for SCL-based algorithms under Assumption 1. For SC-based algorithms, only the maximal value of $\Pr(\mathbf{y}, \hat{u}_1^{jM} | u_{jM+1}^{jM+M})$ is needed. In contrast, for SCL-based algorithms with list size L, the L maximal values of $\Pr(\mathbf{y}, \hat{u}_1^{jM} | u_{jM+1}^{jM+M})$ are needed. A simple understanding for our method is that the $\max(\Pr(\rho))$ function is replaced by a function finding the L maximal values of $\Pr(\rho)$, denoted as $[\Pr(\rho_1), \cdots,$ $\Pr(\rho_L)] = \max_L(\Pr(\rho))$. $\max_L(\Pr(\rho)) \boxtimes \max_L(\Pr(\psi))$ generates L^2 values of $\Pr(\rho_i) \times \Pr(\psi_j)$ for $1 \leq i, j \leq L$.

The path expansion-and-pruning procedure of SCL-based algorithms is divided into two stages. In the first stage, the q most reliable paths are selected for each list by calculating and comparing path metrics. In the second stage, the L most reliable paths among the qL survival paths generated in the first stage. This two stage approach was proposed in our prior work [59], and the novelty herein is that we use the divide-and-conquer method to reduce the complexity of the first stage. The second stage has been described in [59], and we omit its discussions.

Assuming $|\Omega_{01}^{(j)}| = \beta_j$, $|\Omega_{11}^{(j)}| = \gamma_j$. When $\beta_j \geq 1$, let $\Omega_{01}^{(j)} = \{i_1^{(j)}, i_2^{(j)}, \cdots, i_{\beta_j}^{(j)}\}$ $(i_1^{(j)} < i_2^{(j)} < \cdots < i_{\beta_j}^{(j)})$. The first stage includes:

Step 0: the RCC method [59] is applied to calculate both T_1 and T_2.

Step 1: Given any β_j-bit binary vector $\mathcal{B}^{(j)} = (u_{jM+2i_1^{(j)}}, u_{jM+2i_2^{(j)}}, \ldots, u_{jM+2i_{\beta_j}^{(j)}})$, there are 2^{γ_j} possible values for both $v_{\frac{jM}{2}+1}^{\frac{jM+M}{2}}$ and $u_{jM+1,e}^{jM+M}$. We find the $\min(q, 2^{\gamma_j})$

maximal values of 2^{γ_j} values of T_1, and the $\min(q, 2^{\gamma_j})$ maximal values of 2^{γ_j} values of T_2.

Step 2: For $\mathcal{B}^{(j)}$, there are $\left(\min(q, 2^{\gamma_j})\right)^2$ values of T, which is a product of values of T_1 and T_2 generated in Step 1.

Step 3: The q maximal values are selected from $\left(\min(q, 2^{\gamma_j})\right)^2 2^{\beta_j}$ values of T generated by Step 2 because there are 2^{β_j} possible values for $\mathcal{B}^{(j)}$.

If $\Omega_{01}^{(j)} = \varnothing$ and $\beta_j = 0$, we still use the aforementioned four steps to find the q most reliable paths for each list except that $\mathcal{B}^{(j)}$ is considered as a void binary vector which is the only value for $\mathcal{B}^{(j)}$ when $\beta_j = 0$.

Fig. 4.1(d) and 4.1(e) show two examples for frozen-location patterns '$\mathcal{DDDD}$' and '$\mathcal{FDDD}$', respectively when $M = 4$, $N = 4$, and $q = 2$. After these four steps are carried out for each list, there are qL values of T left, which are sorted to choose the L maximal values in the second stage.

The proposed divide-and-conquer AML decoding unit has a lower computational complexity. It reduces the number of symbol-wise channel transition probabilities dealt by the list pruning function by sorting the intermediate calculation results generated by the RCC method [59], whereas the DMM, RCC, and DRH methods perform list pruning function after all the symbol-wise channel transition probabilities are calculated. For example, in Fig. 4.1(d), the DMM, RCC, and DRH methods perform $\max_2(\Pr(y_1^4|u_1^4))$ after all 16 values of $\Pr(y_1^4|u_1^4)$ are calculated. The proposed divide-and-conquer method performs $\max_2(\Pr(y_1^2|v_1^2))$ and $\max_2(\Pr(y_3^4|u_2, u_4))$ first. Then it finds the two maximal values out of four elements generated by $\max_2(\Pr(y_1^2|v_1^2)) \boxtimes \max_2(\Pr(y_3^4|u_2, u_4))$. The output of the proposed AML decoding unit is the same as those of other ML decoding units if they have the same input.

Given an M-bit symbol u_{jM+1}^{jM+M}, $\Omega_{01}^{(j)}$, $\Omega_{11}^{(j)}$, $\Pr(y_1^{\frac{N}{2}}, \hat{v}_1^{\frac{jM}{2}} | v_{\frac{jM}{2}+1}^{\frac{jM+M}{2}})$, and $\Pr(y_{\frac{N}{2}+1}^N, \hat{u}_{1,e}^{jM} | u_{jM+1,e}^{jM+M})$, the first stage using the divide-and-conquer decoding unit needs two 2^{γ_j}-to-$\big(\min(q, 2^{\gamma_j})\big)$ sorts, one $\big(\min(q, 2^{\gamma_j})\big)^2 2^{\beta_j}$-to-$q$ sort, and $\big(\min(q, 2^{\gamma_j})\big)^2 2^{\beta_j}$ multiplications per list, whereas the ML decoding unit in [59] needs $2^{\beta_j+2\gamma_j}$ multiplications and a $2^{\beta_j+2\gamma_j}$-to-q sort per list. By examining all possible values of β_j and γ_j, we can find the worst-case computational complexity.

We demonstrate the advantage of the proposed divide-and-conquer AML unit in computational complexity as opposed to other ML decoding units with an example of $M = 8$ and $q = 4$. Henceforth, we only discuss the computational complexity per list to accomplish the job of the first stage of the proposed method. Table 4.1 lists worse-case computational complexities of different methods, and shows that the proposed method has the smallest computational complexity when 81 eight-bit frozen-location patterns under Assumption 1 need to be dealt with.

Table 4.1: Worst-case computational complexity of different methods when $M = 8$ and $q = 4$.

Method	Computational Complexity
RCC [60] ‡	304 multiplications, a 256-to-4 sort
DMM [40] ‡	1792 multiplications, a 256-to-4 sort
DRH [62] ‡	784 multiplications, a 256-to-4 sort
Divide-and-Conquer AML ‡	112 multiplications, a 64-to-4 sort and two 16-to-4 sorts
Divide-and-Conquer AML †	80 multiplications, a 32-to-4 sort and two 16-to-4 sorts
LC-AML ⋆	80 multiplications, a 32-to-4 sort and four 8-to-4 sorts

‡ All 81 eight-bit patterns under Assumption 1 are dealt with.
† Only nine eight-bit patterns in Sec. 4.3.1 are dealt with.
⋆ Only six eight-bit patterns in Sec. 4.4 are dealt with.

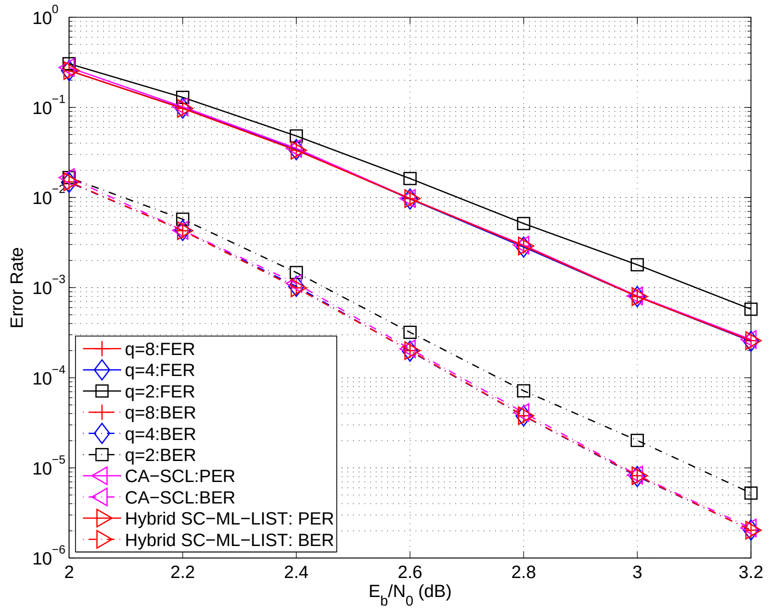


Figure 4.2: Frame and bit error rates of CA-SCL decoders with different qs for a (2048,1433) polar codes with a 32-bit CRC over the AWGN channel when adapting the LC-AML decoding unit.

Our proposed method has the same performance degradation as in [60]. If $q \geq L$, our method does not introduce any performance degradation for SCL-based algorithms. If $q < L$, the performance degradation depends on q and L, and is usually negligible when q and L are small. Fig. 4.2 shows the frame and bit error rates of a (2048, 1433) polar code with a 32-bit CRC of CA-SCL decoders with the LC-AML decoding unit when $M = 8$ and $L = 8$. When $q = 8$, the proposed algorithm has no performance loss compared with the CA-SCL decoder in [33] and the hybrid SC-ML-LIST algorithm in [1]. When $q = 4$, the performance loss is negligible compared with that of $q = 8$. However, $q = 2$ leads to a performance loss of about 0.1 dB at a frame error rate (FER) of 10^{-3}.

4.3 Frozen-Location Patterns for Polar Codes

Considering the hardware implementation for the divide-and-conquer AML unit, a uniform hardware design for all frozen-location patterns is preferred rather than different dedicated designs for various frozen-location patterns. For $M = 8$ and $M = 16$, there are 81 and 6561 possible frozen-location patterns satisfying Assumption 1, respectively. Actually, some of them may never exist in a polar code. Therefore, we want to know the exact number of frozen-location patterns in a polar code, since the number of frozen-location patterns impacts the complexity of the divide-and-conquer AML decoding unit for SCL-based algorithms: the more frozen-location patterns, the higher complexity the divide-and-conquer AML decoding unit has.

4.3.1 Polar Codes for the BEC

For polar codes constructed for the BEC with an erasure probability ϵ $(0 < \epsilon < 1)$, Eqs. (4.1) and (4.2) are used in [64]. In order to examine frozen-location patterns in these polar codes, we have following results regarding the ordering of $z_{i,j}$ for $i \geq 1$ and $1 \leq j \leq 2^i$. This ordering determines possible frozen-location patterns in a polar code.

Proposition 4.3.1. Assuming $z_{0,1} = \epsilon \in (0, 1)$, given any $i \geq 1$ and $1 \leq j \leq 2^i$, $z_{i,j}$ is calculated by Eqs. (4.1) or (4.2). We have

(a) $0 < z_{i,j} < 1$ for $i \geq 1$ and $1 \leq j \leq 2^i$,

(b) $z_{i,2j-1} > z_{i,2j}$ for $i \geq 1$ and $1 \leq j \leq 2^{i-1}$,

(c) $z_{i,4j-3} > z_{i,4j-2} > z_{i,4j-1} > z_{i,4j}$ for $i \geq 2$ and $1 \leq j \leq 2^{i-2}$,

(d) $z_{i,8j-7} > z_{i,8j-6} > z_{i,8j-5} > z_{i,8j-3} > z_{i,8j-4} > z_{i,8j-2} > z_{i,8j-1} > z_{i,8j}$ for $i \geq 3$ and $1 \leq j \leq 2^{i-3}$.

Proof of Proposition 4.3.1. (a) First, $0 < z_{1,1} = 2\epsilon - \epsilon^2 = 1 - (1-\epsilon)^2 < 1$. Second, $0 < z_{1,2} = \epsilon^2 < 1$. Then, by induction, for $i \geq 1$ and $1 \leq j \leq 2^i$, $0 < z_{i,j} < 1$ is satisfied.

(b) For any $i \geq 1$ and $1 \leq j \leq 2^i$, $z_{i,2j-1} - z_{i,2j} = 2z_{i-1,j} - z_{i-1,j}^2 - z_{i-1,j}^2 = 2z_{i-1,j}(1 - z_{i-1,j})$. By Proposition 4.3.1(a), $z_{i,2j-1} - z_{i,2j} > 0 \Rightarrow z_{i,2j-1} > z_{i,2j}$.

(c) By Proposition 4.3.1(b), $z_{i,4j-3} > z_{i,4j-2}$ and $z_{i,4j-1} > z_{i,4j}$. $z_{i,4j-2} - z_{i,4j-1} = 2z_{i-2,j}^2(1 - z_{i-2,j})^2$. By Proposition 4.3.1(a), $z_{i,4j-2} - z_{i,4j-1} > 0 \Rightarrow z_{i,4j-2} > z_{i,4j-1}$.

Therefore, $z_{i,4j-3} > z_{i,4j-2} > z_{i,4j-1} > z_{i,4j}$.

(d) By Proposition 4.3.1(c), $z_{i,8j-7} > z_{i,8j-6} > z_{i,8j-5} > z_{i,8j-4}$ and $z_{i,8j-3} > z_{i,8j-2} > z_{i,8j-1} > z_{i,8j}$. We also have $z_{i,8j-5} > z_{i,8j-3}$ and $z_{i,8j-4} > z_{i,8j-2}$ because $z_{i,4j-2} > z_{i,4j-1}$.

Now let us compare $z_{i,8j-4}$ and $z_{i,8j-3}$,

$$\begin{aligned} z_{i,8j-4} - z_{i,8j-3} &= -2z_{i-3,j}^2(1 - z_{i-3,j})^2 \\ &\quad \times (2 + 4z_{i-3,j} - 5z_{i-3,j}^2 + 2z_{i-3,j}^3 - z_{i-3,j}^4). \end{aligned}$$

By Proposition 4.3.1(a), $z_{i,8j-4} < z_{i,8j-3}$.

Therefore, $z_{i,8j-7} > z_{i,8j-6} > z_{i,8j-5} > z_{i,8j-3} > z_{i,8j-4} > z_{i,8j-2} > z_{i,8j-1} > z_{i,8j}$.

□

Now, let us explain how the ordering of $z_{n,j}$ determines 2^m-bit ($1 \le m \le 3$) frozen-location patterns in an (N, K) polar code over the BEC. First, to choose elements of $\mathcal{A}^c$ for an (N, K) polar code over the BEC, $\mathcal{A}^c$ is chosen such that $\sum_{j\in\mathcal{A}^c} z_{n,j}$ is maximal and $|\mathcal{A}^c| = N - K$, where $N = 2^n$. Then, if there are k_j frozen bits in a symbol $u_{2^m j+1}^{2^m(j+1)}$ $(0 \le j < \frac{N}{2^m})$, a set $\mathcal{A}_j^c$ consisting of indexes of these k_j frozen bits must be chosen such that $\sum_{t\in\mathcal{A}_j^c} z_{n,t}$ is maximal while $|A_j^c| = k_j$. For example, assuming there are four frozen bits in u_1^8 in a (16, 12) polar code, by Proposition 4.3.1(d), $z_{4,1} > z_{4,2} > z_{4,3} > z_{4,5} > z_{4,4} > z_{4,6} > z_{4,7} > z_{4,8}$. Hence, u_1, u_2, u_3, and u_5 will be frozen bits and the frozen-location pattern for u_1^8 will be '$\mathcal{FFFDFDDD}$'.

Therefore, for polar codes constructed by the method in [64], by Proposition 4.3.1(b), there are three two-bit frozen-location patterns: '$\mathcal{DD}$', '$\mathcal{FD}$', and '$\mathcal{FF}$'. We note that the implication of Proposition 4.3.1(b) is the counterpart over the BEC of Assumption 1 in [1]. By Proposition 4.3.1(c), there are five four-bit frozen-location patterns: '$\mathcal{DDDD}$', '$\mathcal{FDDD}$', '$\mathcal{FFDD}$', '$\mathcal{FFFD}$', and '$\mathcal{FFFF}$'. By Proposition 4.3.1(d), there are nine eight-bit frozen-location patterns: '$\mathcal{DDDDDDDD}$', '$\mathcal{FDDDDDDD}$', '$\mathcal{FFDDDDDD}$', '$\mathcal{FFFDDDDD}$', '$\mathcal{FFFDFDDD}$', '$\mathcal{FFFFFDDD}$', '$\mathcal{FFFFFFDD}$', '$\mathcal{FFFFFFFD}$', and '$\mathcal{FFFFFFFF}$'.

For a larger symbol size, it is hard to get the ordering of $z_{i,j}$ by an analytical method. A numerical method can be used. For example, the symbol size is 16. By Proposition 4.3.1(d), we have $z_{4,1} > z_{4,2} > z_{4,3} > z_{4,5} > z_{4,4} > z_{4,6} > z_{4,7} > z_{4,8}$ and $z_{4,9} > z_{4,10} > z_{4,11} > z_{4,13} > z_{4,12} > z_{4,14} > z_{4,15} > z_{4,16}$. We also have $z_{4,5} > z_{4,9} > z_{4,7} > z_{4,11}$ and $z_{4,4} > z_{4,6} > z_{4,10} > z_{4,8} > z_{4,12}$. For $0 < z_{0,1} = \epsilon < 1$,

$$z_{4,10} - z_{4,7} = 2\epsilon^4(\epsilon - 1)^4 \left[\epsilon^3(\epsilon^2 - \epsilon + 24)(\epsilon - 1)^3 - 8)\right] < 0.$$

Moreover, for $0 < z_{0,1} = \epsilon < 1$, $z_{4,4} - z_{4,9} = 2\epsilon^2(\epsilon-1)^4(\epsilon^{10} - 4\epsilon^9 + 34\epsilon^8 - 116\epsilon^7 + 237\epsilon^6 - 375\epsilon^5 + 420\epsilon^4 - 280\epsilon^3 + 102\epsilon^2 - 16\epsilon - 4) < 0$ and $z_{4,8} - z_{4,13} = 2\epsilon^4(\epsilon-1)^2(\epsilon^{10} - 6\epsilon^9 + 43\epsilon^8 - 132\epsilon^7 + 251\epsilon^6 - 262\epsilon^5 + 121\epsilon^4 - 8\epsilon^3 - 6\epsilon^2 - 4\epsilon - 2) < 0$. These two inequalities are verified numerically.

Because of the recursive calculation of $z_{i,j}$, for $i \geq 4$ and $1 \leq j \leq 2^{i-4}$, we have

$$\begin{aligned} z_{i,16j-15} &> z_{i,16j-14} > z_{i,16j-13} > z_{i,16j-11} \\ &> z_{i,16j-7} > z_{i,16j-12} > z_{i,16j-10} > z_{i,16j-9} \\ &> z_{i,16j-6} > z_{i,16j-5} > z_{i,16j-3} > z_{i,16j-8} \\ &> z_{i,16j-4} > z_{i,16j-2} > z_{i,16j-1} > z_{i,16j}. \end{aligned}$$

Thus, there are only 17 frozen-location patterns for 16-bit symbols.

It is not meaningful to consider the symbol size greater than 16, because this will incur very high complexity for hardware implementations.

4.3.2 Polar Codes for the AWGN Channel

For the construction method introduced in [65] for the AWGN channel, it is difficult to analyze the relationship between $z_{3,i}$'s for $1 \leq i \leq 8$ based on Eqs. (4.3) and (4.5). Instead, we examine eight polar codes constructed with the method in [65], which have code lengths from 2^{10} to 2^{13} and code rates of 0.5 and 0.8 to identify eight-bit frozen-location patterns. By examining all eight-bit symbols of these polar codes, we found that in these codes there are only nine eight-bit frozen-location patterns, which are the same as those for polar codes constructed for the BEC, listed in Sec. 4.3.1. Our observation is consistent with Assumption 1 in [1].

4.3.3 Computational Complexity of the Divide-and-Conquer AML Decoding Unit

When it needs to deal with only the frozen-location patterns mentioned in Sections 4.3.1 and 4.3.2, the divide-and-conquer AML decoding unit has a smaller complexity. If $M = 8$ and $q = 4$, it needs 80 multiplications, two 16-to-4 sorts, and a 32-to-4 sort, as listed in Table 4.1. It saves 32 multiplications, a 32-to-4 sort and a 8-to-4 sort compared with the divide-and-conquer AML decoding unit which deals with all 81 frozen-location patterns following Assumption 1, since a 64-to-4 sort consists of two 32-to-4 sorts and a 8-to-4 sort.

If $M = 16$ and $q = 4$, to deal with all $3^8 = 6561$ 16-bit frozen-location patterns satisfying Assumption 1, the first stage of the proposed ML decoding unit needs 1632 multiplications, two 256-to-4 sorts, and a 1024-to-4 sort. However, to deal with 17 16-bit frozen-location patterns discussed in Section 4.3.1, the simplified divide-and-conquer AML decoding unit needs 736 multiplications, two 256-to-4 sorts, and a 128-to-4 sort.

4.4 Low-Complexity AML Decoding Unit

For convenience, we implement the proposed divide-and-conquer AML decoding unit, assuming $M = 8$ henceforth. Our implementation can be readily extended to other values of M. To further reduce complexity and latency, we do not use the divide-and-conquer method to deal with patterns '$\mathcal{DDDDDDDD}$', '$\mathcal{FFFFFFFD}$', and '$\mathcal{FFFFFFFF}$', which will be described in Sec. 4.4.2. Then the divide-and-conquer AML decoding unit can be simplified further by dealing with only the remaining six eight-bit frozen-location patterns. This simplified divide-and-conquer AML decoding unit, referred to as the LC-AML decoding unit, needs 80 multiplications, four 8-to-4 sorts, and a 32-to-4 sort. It saves two 8-to-4 sorts compared with the divide-and-conquer AML decoding unit dealing with nine patterns, since a 16-to-4 sort consists of three 8-to-4 sorts. This also leads to a shorter critical path in our design than the divide-and-conquer AML decoding unit.

4.4.1 Hardware Design for the LC-AML Decoding Unit

SCL-based polar decoders in the literature can be divided into two categories: the log-likelihood (LL) based decoders [42,44,60] and the log-likelihood-ratio (LLR) based decoders [45, 62]. Although our proposed algorithm in Sec. 4.2 is described in the probability domain, it can be easily adapted for both the LL-based decoder and the LLR-based decoder. We focus on the LLR-based polar decoder, because in general the LLR-based decoder has a better throughput-area efficiency than the LL-based decoder.

First, we adapt the proposed LC-AML decoding unit to the LLR-based SCL decoder. Given path metrics $\text{PM}_k^{(t)}$ of L list survivors and assuming u_t is the last bit processed by the decoder, where $1 \leq k \leq L$, $1 \leq t \leq N$, and t is a multiple of M. Suppose $\alpha_{j,l}$ $(0 \leq j < M)$ represents the LLR of $\Pr(y_{j\frac{N}{M}+1}^{(j+1)\frac{N}{M}}, \hat{w}_{1+j\frac{N}{M}}^{\frac{t}{M}+j\frac{N}{M}} | w_{\frac{t}{M}+1+j\frac{N}{M}})$ corresponding to the list l. The path metric $\mathcal{PM}_{k,p}^{(t+M)}$ of the p-th expanded path from the k-th list survivor corresponding to $u_{t+1}^{t+M} = p$ $(0 \leq p < 2^M)$ is $\mathcal{PM}_{k,p}^{(t+M)} = \text{PM}_k^{(t)} + \sum_{j=0}^{M-1} m_j|\alpha_{j,l}|$, where $m_j = 0$ if $w_{\frac{t}{M}+1+j\frac{N}{M}} = \frac{1}{2}(1 - \text{sign}(\alpha_{j,l}))$ [45]. Otherwise, $m_j = 1$. Then our goal is to calculate $\mathcal{PM}_{k,p}^{(t+M)}$ and to select the L minimum values of $\mathcal{PM}_{k,p}^{(t+M)}$.

Fig. 4.3 shows the top architecture of our low-complexity implementation for the LC-AML decoding unit. MLD_S1 calculates path metrics and selects the q minimum values for each list. FrzInfVec is an M-bit frozen bit indication vector $(f_1, f_2, \cdots, f_M)$ for u_{t+1}^{t+M}. For $1 \leq j \leq M$, if u_{t+j} is a frozen bit, $f_j = 1$; otherwise, $f_j = 0$. LLRInV_l is the vector $(\alpha_{0,l}, \alpha_{1,l}, \cdots, \alpha_{M-1,l})$ for $1 \leq l \leq L$.

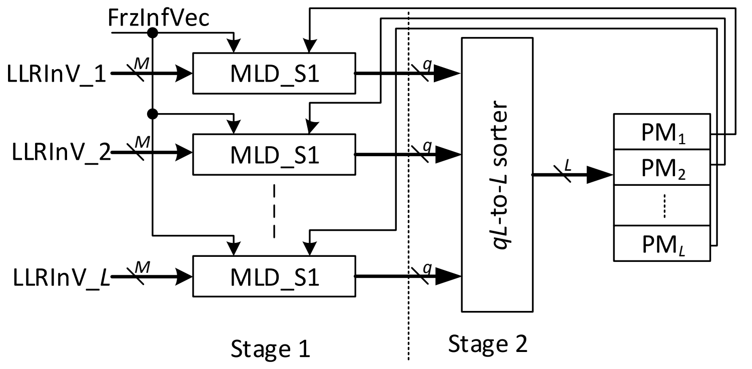


Figure 4.3: Top architecture of the proposed LC-AML decoding unit.

Fig. 4.4(a) shows the design for MLD_S1_$q4$ when $M = 8$ and $q = 4$. Here, we focus on the data path for calculating path metrics. The circuitry to generate symbol values associated with path metrics is simple and consists of XORs, and therefore is omitted. The data paths corresponding to different steps aforementioned in Section 4.2 are labeled as well.

In Step 0, two RCC blocks, shown in Fig. 4.4(b), are used. LLR a_i $(16 \leq i \leq 31)$ associated with $\Pr(y_{\frac{N}{2}+1}^{N}, \hat{u}_{1,e}^{8j}|u_{8j+1,e}^{8j+8} = (i-16)_2)$ is calculated by the right RCC block. LLR a_i $(0 \leq i \leq 15)$ associated with $\Pr(y_1^{\frac{N}{2}}, \hat{v}_1^{4j}|v_{4j+1}^{4j+4} = (i)_2)$ is calculated by the left RCC block. Here, $(i)_2$ represents the binary string of interger i. 16-ADDER contains

16 adders to calculate path metrics, shown in Fig. 4.4(c).

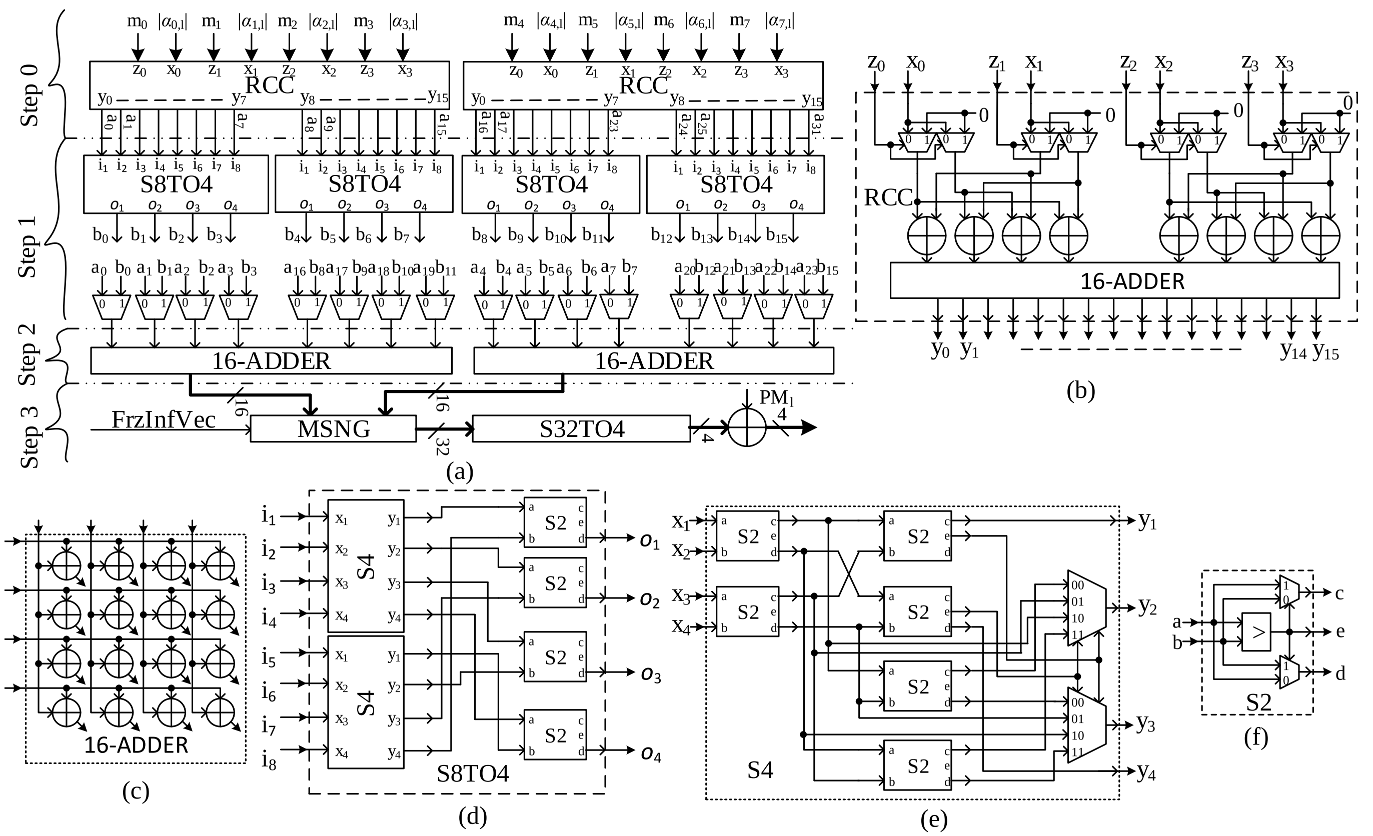


Figure 4.4: Design of MLD_S1_$q4$ when $M = 8$ and $q = 4$. (a) top architecture of MLD_S1_$q4$, (b) diagram of RCC, (c) diagram of 16-ADDER, (d) diagram of S8TO4, (e) diagram of S4, (f) diagram of S2.

In Step 1, for different frozen-location patterns, path metrics go through different data paths selected by 16 2-to-1 multiplexers. Their control words are 1 if frozen-location patterns are $\mathcal{FDDDDDDD}$, $\mathcal{FFDDDDDD}$; otherwise they are 0.

In Step 2, results from Step 1 are combined to calculate $\sum_{j=0}^{7} m_j|\alpha_{j,l}|$.

In Step 3, there are 32 path metrics going through a 32-to-4 sorter. However, for some frozen-location patterns, the number of valid symbol values is less than 32 because the number of frozen bits can be larger than three. Therefore, path metrics associated with those invalid symbol values need to be set to the maximal positive value as well so that the 4 minimum path metrics belong to valid symbol values. MSNG accomplishes this job with FrzInfVec, which contains the frozen-location pattern information.

Different sorters used in our design are shown in Fig. 4.4(d), 5(e) and 5(f). S8TO4 finds the minimum four values of eight values. S4 sorts the four inputs and outputs them in decreasing order and has a shorter critical path of two comparators and one 4-to-1 multiplexer than a four-input bitonic sorter [57], which has a critical path of three comparators. S32To4 consists of seven S8TO4 units in a binary tree structure.

Although MLD_S1_$q4$ is designed for six eight-bit frozen-location patterns, other frozen-location patterns also can be dealt with by MLD_S1_$q4$, such as all frozen-location patterns satisfying the following two conditions. First, the frozen-location pattern has at least three '$\mathcal{F}$'s. Second, two frozen bits are located at the first two bits of the data symbol.

4.4.2 Area-Efficient SCL Decoder

To examine the advantage of our proposed design, we incorporate MLD_S1_$q4$ into CA-SCL polar decoders with list size $L = 4$. Architecture-wise, our decoder, referred to as the AE-SCL decoder, is almost the same as the architecture of the tree based reduced latency SCL polar decoder in [62], which performs the CA-SCL decoding algorithm on a binary tree representation of a polar code, except that our AE-SCL decoder uses the LC-AML decoding unit instead of the DRH ML decoding unit used in [62].

Leaf nodes of the decoding tree for our decoder are divided into four categories:

1. Rate-0 node: its frozen-location pattern contains only '$\mathcal{F}$', i.e., the node contains only frozen bits.
2. Rate-1 node: its frozen-location pattern contains only '$\mathcal{D}$', i.e., the node contains only information bits.
3. Repetition node [58]: its frozen-location pattern is either '$\mathcal{FFFFFFFF}$_$\mathcal{FFFFFFFD}$' or '$\mathcal{FFFFFFFD}$'.
4. Rate-R-2 node: its frozen-location pattern is one of the six eight-bit frozen-location patterns.

Rate-0 and rate-1 nodes are decoded with the same methods as in [62]. The main difference between our proposed decoder here and the tree based reduced latency SCL polar decoder is how to deal with repetition nodes and rate-R-2 nods. For repetition nodes, a binary tree of adders is used to calculate LLRs in order to reduce the decoding latency [58]. Rate-R-2 nodes are dealt with by MLD_S1_$q4$, which reduces the area of AE-SCL decoders.

4.4.3 Synthesis Results

AE-SCL decoders with $L = 4$ are implemented for three polar codes: a (1024, 512) code, an (8192,4096) code, and a (32768, 29504) code. The first two codes are constructed for the BEC with $\epsilon = 0.5$ and the third code is for the AWGN channel with a noise variance of 0.1757. These three codes are with a 32-bit cyclic redundancy check whose generator polynomial is 0x1EDC6F41. The number of processing units of decoders for $N = 1024$ is 256. For the other two codes, the decoder has 512 processing units. Five-bit channel LLRs are used. The synthesis tool is Cadence RTL compiler. The process technology is TSMC 90nm CMOS technology. Here, four stages of pipeline registers are used in the LC-AML decoding unit. Areas of different ML decoding units for the (1024, 512) polar codes are listed in Table 4.2. The area of our proposed LC-AML decoding unit is only one fourth of that of the ML decoding unit in [62]. By taking into account fewer patterns, the area of the LC-AML decoding unit is 67% of that of the Divide-and-Conquer AML design which deals with all 81 eight-bit frozen-location patterns following Assumption 1.

Table 4.2: Areas of different ML decoding units for the (1024,512) polar code.

	LC-AML†	Divide-and-Conquer AML‡	[62]
area (mm^2)	0.456	0.673	2.298

† The LC-AML design targets the six eight-bit frozen-location patterns.
‡ The Divide-and-Conquer AML design targets all 81 eight-bit frozen-location patterns following Assumption 1.

The synthesis results of three entire decoders (AE-SCLs) are also listed in Tables 4.4, 4.5, and 4.6, respectively. Here, NIT means the net information throughput. Compared with decoders in [62], the SCL decoder architecture with the best throughput-area efficiency to our knowledge, the AE-SCL decoders have smaller areas because the proposed

LC-AML decoding unit is applied. The LC-AML decoding unit has a slightly larger decoding latency than that in [62], because the proposed LC-AML decoding unit deals with only eight-bit frozen-location patterns, whereas the ML decoding unit in [62] can deal with some 16-bit frozen-location patterns. Since the extra decoding cycles needed by AE-SCL decoders are a very small fraction of the entire decoding cycles, the proposed AE-SCL decoders still achieve better throughput-area efficiency than decoders in [62]. For example, for the (1024, 512) polar code, the throughput-area efficiency of the AE-SCL decoder is 1.93 times of that of the decoder in [62]. As the code length increases, the advantage of throughput-area efficiency is less because the ML decoding unit occupies a smaller fraction of the entire decoder if the code is longer. Compared with symbol-decision SCL decoders in [3, 45, 60], the advantage of our decoders on the throughput-area efficiency is more significant. The throughput-area efficiency of the AE-SCL decoder is 3.32, 8.25, and 3.17 times of that of decoders in [3, 45, 60], respectively, for the (1024, 512) polar code.

4.5 Multi-Mode SCL Decoder

All existing SCL polar decoders in the literature provide fixed throughput and decoding latency given a polar code. These SCL decoders cannot adapt to variable communication channels and applications. In order to adapt to different throughput and latency requirements, we propose a multi-mode SCL (MM-SCL) decoder with n_d decoding paths, which can decode P received words with list size L in parallel, where $1 \leq P, L \leq n_d$ and $n_d \geq P \times L$. For simplicity, we use the number of the received words decoded simultaneously as the mode index and call this mode-P. This multi-mode feature requires the decoder to perform SCL decoding algorithms with different list sizes (the SC decoding algorithm is a special case of the SCL decoding algorithm with list size $L = 1$).

4.5.1 Architecture Description

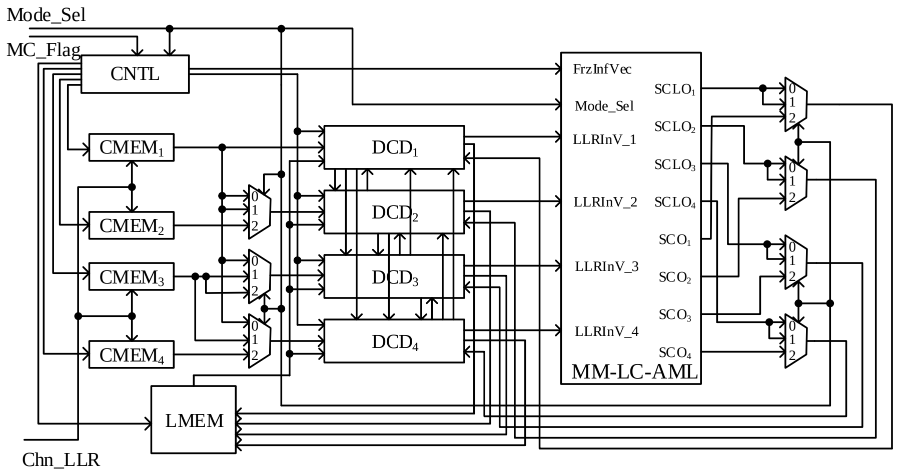


Figure 4.5: Top architecture of the MM-SCL decoder when $n_d = 4$.

Assuming $n_d = 4$, the top architecture of the MM-SCL decoder is shown in Fig. 4.5. It has four blocks of channel memory, CMEM$_i$ $(1 \leq i \leq 4)$, to store four received words from Chn_LLR since the decoder under mode-4 deals with four received words simultaneously. Block DCD$_i$ $(1 \leq i \leq 4)$ contains processing units to calculate LLRs, and partial-sum units to update partial-sum for each list. The intermediate LLRs calculated by DCD$_i$ are stored in LMEM. Designs for processing units, partial-sum units and the interface between processing units and LMEM adopt blocks of the reduced-latency tree-based SCL decoder in [62].

The control block (CNTL), designed based on the instruction RAM based methodology in [58], includes two parts shown in Fig. 4.6. The first part is the control ROM

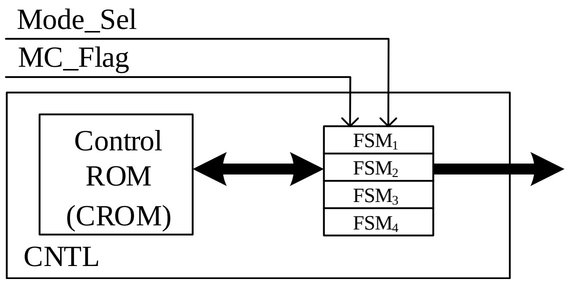


Figure 4.6: Diagram of CNTL.

(CROM) which has $\mathcal{N}$ words. Here, $\mathcal{N}$ is the number of the leaf nodes of the decoding tree [62] which is determined by the frozen-bit distribution of the polar code and parameters of the decoder. For a different polar code, the CROM needs to be re-programed. Each word of the CROM corresponds to a leaf node and contains the following information:

- the layer index, the node type and the size of the current leaf node,
- the frozen-location pattern of the current leaf node,
- the indices of log-likelihood ratio (LLR) vectors which will be updated for the current leaf node,
- the indices of partial sums which will be udpated for the current leaf node,
- the number of clock cycles needed for the current leaf node.

The other part is four finite state machines (FSMs). Each of them is associated with a decoding path and reads a word from the CROM and generates control signals for the proposed decoder, such as read/write addresses and enable signals for memory blocks, and control signals for decoding paths.

We focus on the additional logic to support multi-mode features. Mode_Sel is a two-bit control word to select the decoding mode of the MM-SCL decoder: 00, 01, and 10 for mode-1, mode-2, and mode-4, respectively. MC_Flag indicates that a mode change happens within a decoding process: at the beginning of a decoding process, MC_Flag is reset to 0; when a mode change happens within a decoding process, MC_Flag is set to 1 until the end of the decoding process.

Under mode-1, all four DCD$_i$ $(1 < i \leq 4)$ access channel information of CMEM$_1$ and perform a list decoding algorithm with $L = 4$. Under mode-2, our MM-SCL decoder simultaneously decodes two received words with $L = 2$. DCD$_1$ and DCD$_2$ are used to decode the received word located in CMEM$_1$; DCD$_3$ and DCD$_4$ use the channel information from CMEM$_3$. Under mode-4, four received words are simultaneously decoded with $L = 1$: the received word in CMEM$_i$ is decoded by DCD$_i$ for $1 \leq i \leq 4$.

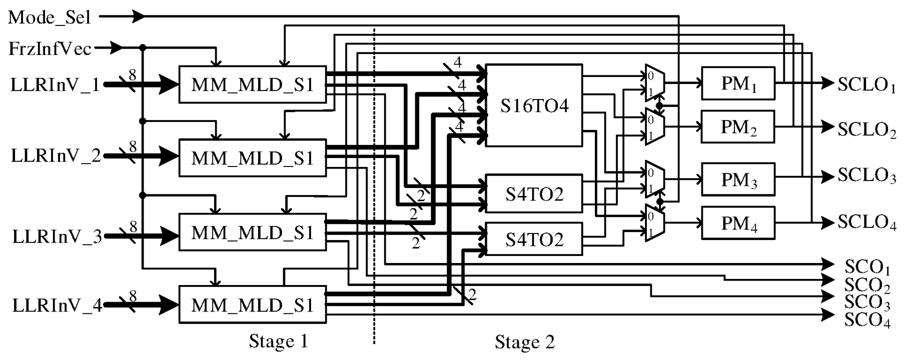


Figure 4.7: Top architecture of MM-LC-AML for the MM-SCL decoder.

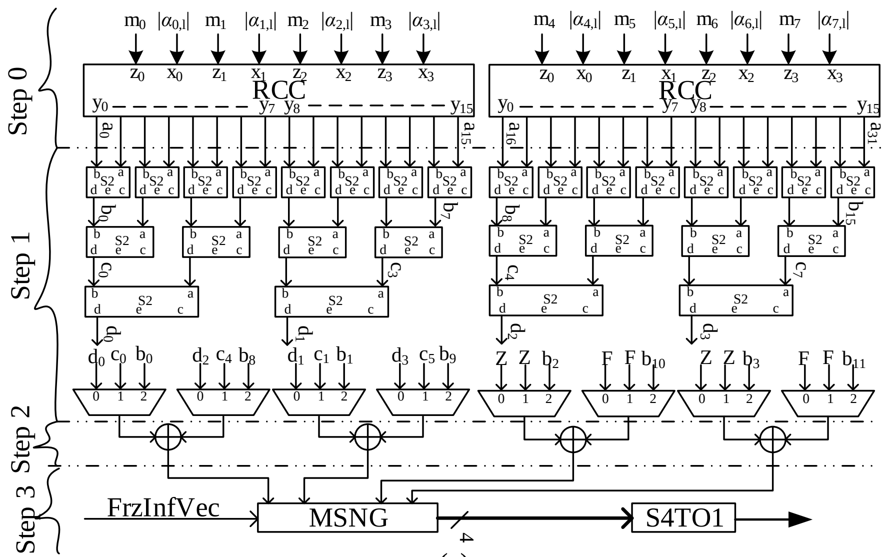
Block MM-LC-AML performs the LC-AML decoding function for different types of leaf nodes and is supposed to output the most reliable list candidate for mode-4, the two most reliable list candidates for mode-2, and the four most reliable list candidates for mode-1. The architecture in Fig. 4.3 is for a fixed list size only. Here, we propose an MM-LC-AML unit (we take $n_d = 4$ and $M = 8$ as an example) shown in Fig. 4.7

to support the multi-mode features. Under mode-1, all of SCLO_1, SCLO_2, SCLO_3, and SCLO_4 are used to decode a received word. Under mode-2, SCLO_1 and SCLO_2 are used for one of two received words; SCLO_3 and SCLO_4 are used for the other received word. Under mode-4, each of $\text{SCO}_i(1 \leq i \leq 4)$ is used by an individual received word.

MM_MLD_S1 performs the same function as MLD_S1_$q4$ in Fig. 4.4, except that MM_MLD_S1 supports multiple modes. This can be accomplished by simply adding an S4 block between S32TO4 and the adder at the bottom right of Fig. 4.4(a). This implementation leads to a slightly longer critical path due to the extra block in the data path and hence a larger decoding latency. To address this issue, we redesign MLD_S1 for mode-4 and mode-2, respectively, called MLD_S1_$q1$ and MLD_S1_$q2$, shown in Fig. 4.8(a) and 4.8(b). Symbol values for 'Z' and 'F' are four-bit vectors '0000' and '1111', respectively. Hence, the symbol value calculated from 'Z' and 'F' is '11111111', which is guaranteed to be an invalid symbol value for our designs.

If mode-2 is used, the control words for patterns '$\mathcal{FFDDDDDD}$', '$\mathcal{FDDDDDDD}$', and '$\mathcal{FFFDDDDD}$' are 0, 1, and 2, respectively; for the remaining patterns, the control words are 3. If mode-4 is used, the control words for patterns '$\mathcal{FFDDDDDD}$', '$\mathcal{FDDDDDDD}$', and '$\mathcal{FFFDDDDD}$' are 0, 0, and 1, respectively; for the remaining patterns, the control words are 2.

Actually, MLD_S1_$q4$, MLD_S1_$q2$ and MLD_S1_$q1$ are integrated together instead of three individual blocks in MM_MLD_S1, since they have the same circuitry for Step 0. Furthermore, sorting units of the top row of Step 1 in these three designs can also be reused because S8TO4 contains several S4 blocks and S2 blocks. The hardware sharing reduces the additional area for supporting multiple modes and improves throughput-area efficiency without increasing the critical path delay.


(a)

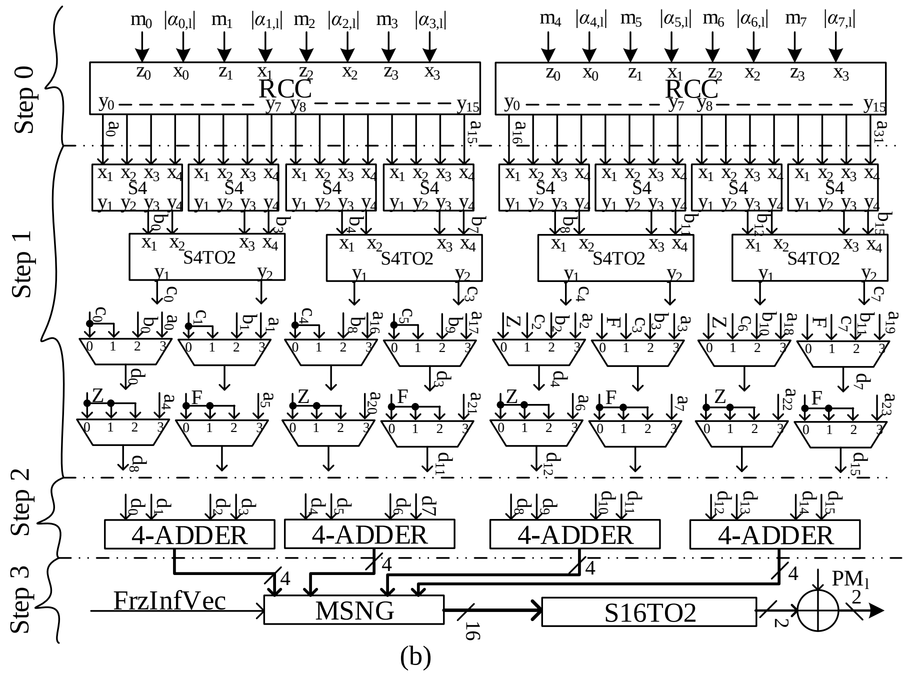

(b)

Figure 4.8: (a) Design of MLD_S1_$q1$ for $q = 1$, (b) design of MLD_S1_$q2$ for $q = 2$.

Compared with the AE-SCL decoder in Section 4.4.2, to support multiple modes the MM-SCL decoder needs additional hardware, including the additional three blocks of channel memories and the additional circuitry for mode-2 and mode-4 in MM_MLD_S1.

FERs of the SC and CA-SCL algorithms and different modes for the MM-SCL decoder to decode all three codes are shown in Fig. 4.9. These figures show that, the smaller the mode index, the greater the list size and the better the FER. The CA-SCL-i algorithm is the CA-SCL decoding algorithm in [33] with list size i. The performance differences between our decoder and prior decoding algorithms with the same list size (mode-1 vs. CA-SCL-4, mode-2 vs. CA-SCL-2, mode-4 vs. SC) are very small.

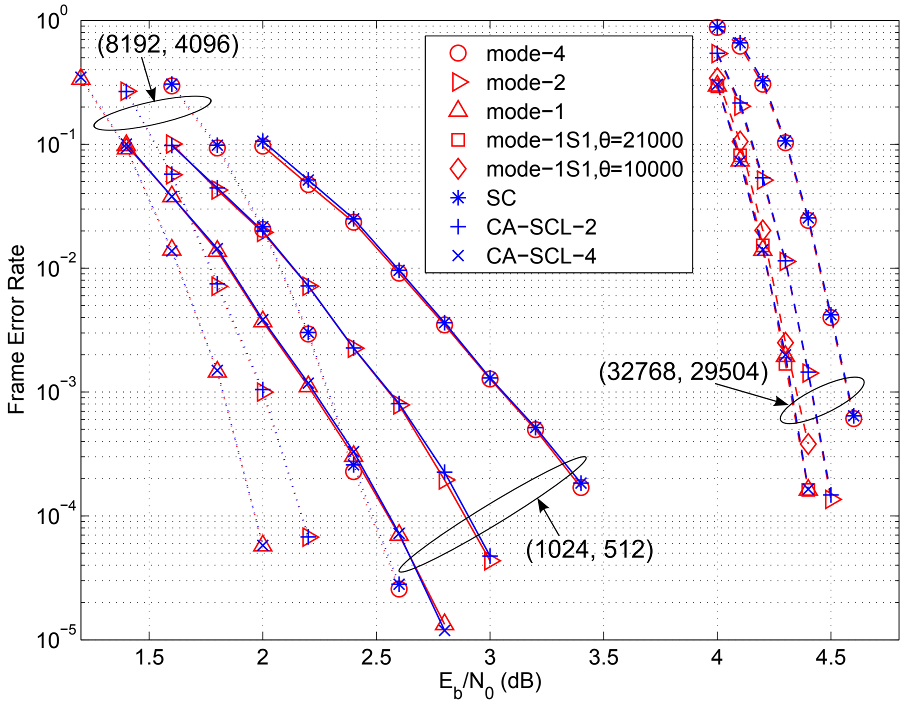


Figure 4.9: Frame error rates of of the SC and CA-SCL algorithms and different modes for the MM-SCL decoder.

4.5.2 Simplified Modes and Mode Changes

The three modes described above provide three possible throughputs and latency. To provide a wider range of throughput and latency, we consider simplified modes and mode changes by changing Mode_Sel and MC_Flag on-the-fly.

Simplified modes are motivated by the computational complexity and latency caused by the expansion-and-pruning process when it involves L paths for a list size L. The idea for simplified modes is to make a switch at some point so that while all L paths are kept, the expansion-and-pruning process is carried out among every L_s ($L_s|L$) paths. The computational complexity and latency of the expansion-and-pruning process are reduced since $L_s < L$, while the performance degradation is negligible when an appropriate switching point θ is used. Let us consider an L_s-simplified mode-P (referred to as mode-$P\text{S}L_s$) with a switching point θ. The first θ bits, u_1^θ, are decoded with list size $\frac{n_d}{P}$. Then each list is divided into $\frac{n_d}{PL_s}$ groups, each of which has L_s survivors. For the remaining bits, the expansion-and-pruning process happens only among the L_s survivors of each group. For example, for a (32768, 29504) code, under mode-1S1 with $\theta = 21000$, u_1^{21000} are decoded with $L = 4$. Then four survivors are divided into four groups and each group has one survivor, i.e., $L_s = 1$. For the remaining 11768 bits, u_{21001}^{32768}, the expansion-and-pruning process happens only for one survivor. In this case, there is no observed performance loss compared with mode-1, while the decoding latency of 6530 cycles is slightly shorter than that of mode-1, 6718 cycles. To reduce the decoding latency further, a smaller switching point can be used at the expense of small performance loss. With $\theta = 10000$, mode-1S1 has a decoding latency of 6206 cycles and has a performance loss of about 0.03 dB compared with mode-1 as shown in Fig. 4.9, but still has a better performance and a slightly shorter decoding latency than mode-2.

Hence, simplified modes provide a different tradeoff between latency (throughput) and performance.

Under simplified modes, the number of simultaneously decoded received words is not changed within the decoding process. To support a wider range of throughput, it is also possible to perform a mode change, i.e., the number of received words simultaneously decoded can be changed within a decoding process. Here, we use mode-PCP' with θ to represent that in the decoding process of u_1^θ, P received words $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_P)$ are decoded simultaneously with list size $\frac{n_d}{P}$; in the decoding process of $u_{\theta+1}^N$, P' ($P' > P$) received words $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{P'})$ are decoded simultaneously with list size $\frac{n_d}{P'}$. More specifically, for each of $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_P$, u_1^θ are decoded with list size $\frac{n_d}{P}$, then the $\frac{n_d}{P'}$ most reliable survivors are kept, and $u_{\theta+1}^N$ are decoded with list size $\frac{n_d}{P'}$. After the switching point, only $\frac{n_d P}{P'}$ decoding paths are used for $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_P$, and the remaining $\frac{n_d(P'-P)}{P'}$ decoding paths are used for $\mathbf{y}_{P+1}, \mathbf{y}_{P+2}, \cdots, \mathbf{y}_{P'}$. For example, let us consider mode-1C4 with four decoding paths and $\theta = 10000$ for a (32768, 29504) polar code. For the first 10000 bits of $\mathbf{y}_1$, the SCL algorithm with $L = 4$ is used and four survivors are generated at this point. Then, the most reliable survivor is selected by comparing the path metrics of these four survivors. Based on the knowledge of this most reliable survivor associated with u_1^{10000} of $\mathbf{y}_1$, the remaining 22768 bits of $\mathbf{y}_1$ are decoded with the SC algorithm by one decoding path DCD$_1$. Meanwhile, after the switching point, three received words ($\mathbf{y}_2$, $\mathbf{y}_3$, and $\mathbf{y}_4$) are fed into the other three decoding paths (DCD$_2$, DCD$_3$, and DCD$_4$) to be decoded with the SC algorithm. Therefore, mode changes provide a wider range of throughput than simplified modes.

The control signals Mode_Sel and MC_Flag are used to facilitate simplified modes and mode changes. When Mode_Sel changes the mode index from P to P', it signifies either a simplified mode-P with $L_s = \frac{n_d}{P'}$ if MC_Flag = 0 or a mode change mode-PCP' if MC_Flag = 1.

Figure 4.10 illustrates decoding schedules for different modes and their control words. In terms of the throughput, mode-4 $>$ mode-1C4 $>$ mode-1S1 $>$ mode-1. The range of the throughput of mode-1S1 is from the throughput of mode-1 up to a quarter of that of mode-4. The range of the throughput of mode-1C4 is wider than that of mode-1S1, from the throughput of mode-1 to that of mode-4.

Therefore, simplified modes and mode changes provide a way for the MM-SCL decoder to reduce decoding latency further somewhat without noticeable performance loss and improves throughput-area efficiency further. It can also be used when decoding needs to finish as soon as possible due to external reasons such as buffer overflow.

In terms of the control of the decoding process, FSMs of different decoding paths of simplified modes are synchronous on the decoding tree because all the decoding paths are working on the same part of the decoding tree. However, these FSMs are not synchronous on the decoding tree when a mode change happens within a decoding process. Therefore, if the feature of mode changes within a decoding process is not needed, only one FSM for all decoding paths in CNTL is enough, but the hardware saving is very small because the area of the control circuitry is a very small fraction of that of the entire decoder. For example, the area of FSMs of the MM-SCL decoder for the (1024, 512) code is 0.011 mm^2, 0.47% of the area of the whole decoder.

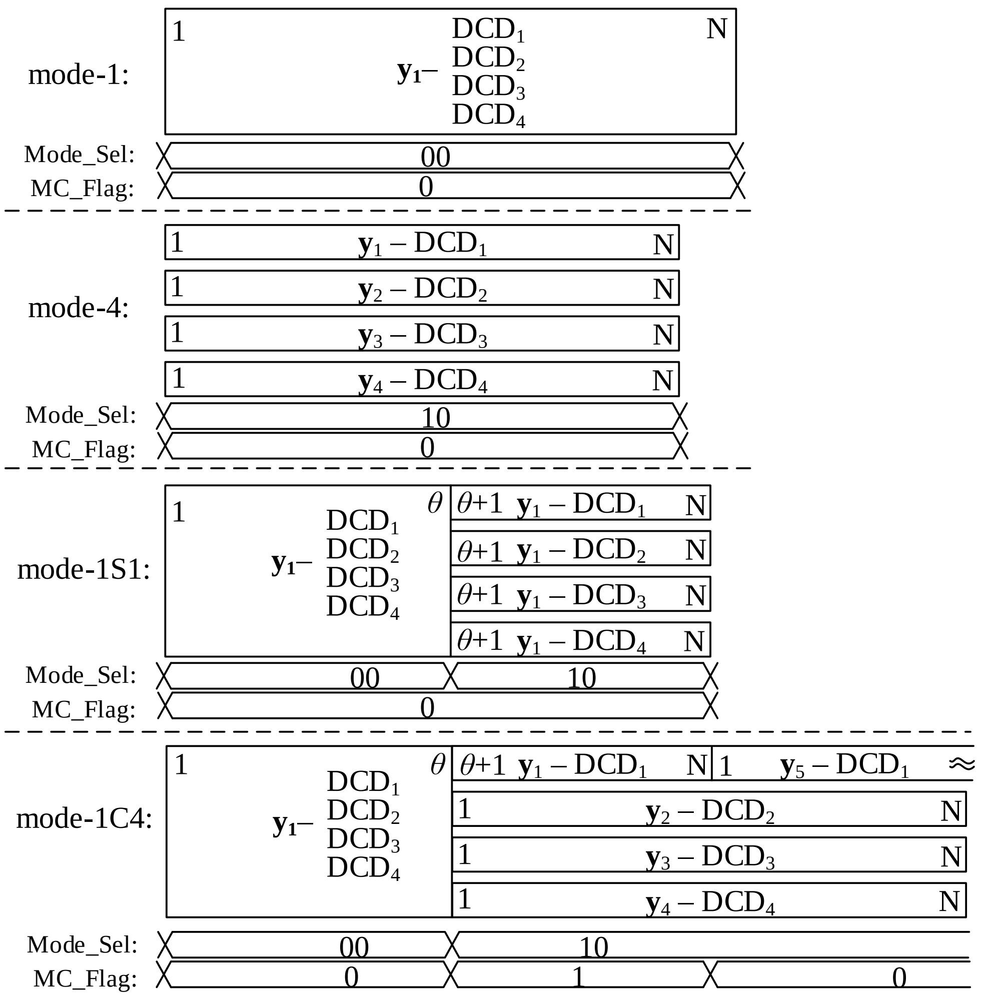

mode-1:
1
$\mathbf{y}_1-$ DCD$_1$ DCD$_2$ DCD$_3$ DCD$_4$
N
Mode_Sel: 00
MC_Flag: 0
mode-4:
1 $\mathbf{y}_1 -$ DCD$_1$ N
1 $\mathbf{y}_2 -$ DCD$_2$ N
1 $\mathbf{y}_3 -$ DCD$_3$ N
1 $\mathbf{y}_4 -$ DCD$_4$ N
Mode_Sel: 10
MC_Flag: 0
mode-1S1:
1 $\mathbf{y}_1-$ DCD$_1$ DCD$_2$ DCD$_3$ DCD$_4$ θ
θ+1 $\mathbf{y}_1 -$ DCD$_1$ N
θ+1 $\mathbf{y}_1 -$ DCD$_2$ N
θ+1 $\mathbf{y}_1 -$ DCD$_3$ N
θ+1 $\mathbf{y}_1 -$ DCD$_4$ N
Mode_Sel: 00 10
MC_Flag: 0
mode-1C4:
1 $\mathbf{y}_1-$ DCD$_1$ DCD$_2$ DCD$_3$ DCD$_4$ θ
θ+1 $\mathbf{y}_1 -$ DCD$_1$ N 1 $\mathbf{y}_5 -$ DCD$_1$ ≈
1 $\mathbf{y}_2 -$ DCD$_2$ N
1 $\mathbf{y}_3 -$ DCD$_3$ N
1 $\mathbf{y}_4 -$ DCD$_4$ N
Mode_Sel: 00 10
MC_Flag: 0 1 0

Figure 4.10: Decoding schedules and control words for different modes.

4.5.3 Synthesis Results

The MM-SCL decoder are implemented for the aforementioned three codes. For the (1024, 512) code, the areas of the channel memory and the ML decoding unit are listed in Table 4.3. It shows that the increased area of the MM-SCL decoder over the AE-SCL decoder is dominated by the area of the three additional blocks of channel memory. Due to hardware sharing, the increased area of the ML decoding unit is small.

Synthesis results of MM-SCL decoders for different polar codes are listed in Tables 4.4, 4.5 and 4.6. The decoding latency of mode-2 is smaller than that of mode-1 and the decoder has the smallest decoding latency under mode-4. This is because MLD_S1_$q2$ and MLD_S1_$q1$ have shorter data paths. Therefore, in MM-LC-AML, three stages and two stages of pipeline registers are used by the circuitry for mode-2 and mode-4, respectively. Mode-1S1 can have a smaller latency than mode-1 and mode-2.

Compared with the AE-SCL decoder, the MM-SCL decoder under mode-1 has a smaller throughput-area efficiency due to the additional circuitry for supporting multiple modes. However, the MM-SCL decoder provides multiple choices of output throughput and decoding latency, which is more suitable for variable communication channels and applications.

Table 4.3: Areas (in mm^2) of the channel memory and the ML decoding unit for MM-SCL and AE-SCL decoders when $N = 1024$ and r=0.5.

	MM-SCL	AE-SCL	Difference
Area of Channel Memory	0.484	0.121	0.363
Area of ML Decoding Unit	0.513	0.456	0.057

Table 4.4: Synthesizing results for different decoders when $N = 1024$ and r=0.5.

Decoder	AE-SCL	mode-1	mode-2	mode-4	[62]	[60]	[45]	[3]		[36]	
List Size	4	4	2	1	4					1	
Area (mm^2)	1.89	2.32			3.83	1.70	1.78	2.14	4.10*	0.31	0.59**
Clock Rate (MHz)	409	409			403	500	794	400	289*	500	361**
# of Decoding Cycles	391	391	357	304	371	1540	2649	1022		2080	
Latency (µs)	0.96	0.96	0.87	0.74	0.92	3.08	3.34	2.56	3.54*	4.16	5.76**
NIT (Mbps)	547	547	1208	2887	570	155	154	200	144*	123	89**
Area Eff. (Mbps/mm^2)	289	236	521	1244	149	91	87	93	35*	397	149**

* Original synthesis results in [3] are based on an ST 65nm CMOS technology. For a fair comparison, synthesis results scaled to a 90nm technology are used in the comparison.
** Original synthesis results in [36] are based on a TSMC 65nm CMOS technology. For a fair comparison, synthesis results scaled to a 90nm technology are used in the comparison.

Table 4.5: Synthesizing results for different decoders when $N = 8192$ and r=0.5.

Decoder	AE-SCL	mode-1	mode-2	mode-4	[62]	[60][‡]	[45][†]
List Size	4	4	2	1	4		
Area (mm^2)	4.49	5.52			6.46	7.32	12.73
Clock Rate (MHz)	398	398			398	434	794
# of Decoding Cycles	2542	2542	2323	1975	2367	11700	20736
Latency (μs)	6.39	6.39	5.84	4.96	5.95	26.96	26.12
NIT (Mbps)	670	670	1473	3503	723	150	156
Area Eff. ($Mbps/mm^2$)	149	121	267	635	112	20	12

‡ The decoder architecture in [60] has been re-synthesized with the TSMC 90nm CMOS technology.
† These results for the decoder in [45] are estimated conservatively.

Table 4.6: Synthesizing results for different decoders when N=32768 and r=0.9.

Decoder	AE-SCL	mode-1	mode-1S1		mode-2	mode-4	[62]	[60][‡]	[45][†]
List Size	4	4	4[⁎]	4[⋆]	2	1	4		
Area (mm^2)	9.97	11.94					11.89	15.8	50.41
Clock Rate (MHz)	350	350					359	389	794
# of Decoding Cycles	6718	6718	6530	6206	6300	5368	6492	65813	96576
Latency (μs)	19.19	19.19	18.66	17.73	18	15.34	18.08	169.19	121.63
NIT (Mbps)	1662	1662	1714	1811	3564	8499	1772	165	242
Area Eff. ($Mbps/mm^2$)	167	139	144	152	298	712	149	10	5

‡ The decoder architecture in [60] has been re-synthesized with the TSMC 90nm CMOS technology.
† These results for the decoder in [45] are estimated conservatively.
⁎ The switching point of mode-1S1 is 21000.
⋆ The switching point of mode-1S1 is 10000.

Compared with the decoder in [62], for $N = 1024$ and $N = 8192$, the MM-SCL decoder has a smaller area and a better throughput-area efficiency. For $N = 32768$, the area of the MM-SCL decoder is bigger than that of the decoder in [62] because the additional circuitry to support multiple modes is larger than the saving due to the low-complexity ML decoding unit in the MM-SCL decoder. For the (1024, 512) code, under mode-1, mode-2, and mode-4, the MM-SCL decoder provides area-efficiencies of 1.58, 3.5, and 8.36 times of throughput-area efficiency of the decoder in [62], respectively.

Compared with decoders in [3, 45, 60], the advantage in throughput-area efficiency of the MM-SCL decoder is more significant. This advantage comes from two aspects. The first is that the tree-based low-latency SCL architecture in [62] is adopted for the MM-SCL decoder. This helps to reduce the decoding latency. The second is due to the low-complexity AML decoding unit. For $N = 1024$, the MM-SCL decoder under mode-1 provides an throughput-area efficiency of 2.73, 6.71, and 2.59 times of area efficiencies of SCL decoders in [3,45,60], respectively. When mode-4 is used, the ratios of the throughput-area efficiency of the MM-SCL decoder over those of SCL decoders in [3, 45, 60] and that of the semi-parallel SC decoder in [36] are 14.38, 35.43, 13.65, and 8.32, respectively.

For $N = 32768$, decoding latencies and throughputs respect to different switching points of mode-1S1 are also provided. A smaller switching point leads to a smaller latency. When the switching point is 10000, the latency of mode-1S1 is even smaller than that of mode-2. Compared with mode-1, improvements on throughput and latency are about 8%.

4.6 Summary

In this chapter, the divide-and-conquer method is applied to SC-based algorithms in the probability domain. By extending this idea, a divide-and-conquer AML decoding unit for SCL-based polar decoder is proposed. By examining frozen-location patterns of polar codes, an efficient hardware design for a simplified divide-and-conquer AML decoding unit is developed. To adapt to different throughput and latency requirements, the MM-SCL polar decoder is proposed in this chapter. Synthesis results show that our implementations for our MM-SCL decoder and SCL decoder with the LC-AML unit achieve better area efficiencies than existing SCL polar decoders.

Chapter 5

Conclusion and future work

5.1 Conclusion

Error correcting codes provide a powerful tool to recover transmitted data from a noisy channel. Two of the most powerful codes, non-binary LDPC codes and polar codes have received lots of research interests because of their excellent error correcting performance. In this dissertation, we investigate how to improve the error performance of MLDG algorithms for non-binary LDPC codes. We also discuss how to reduce complexity of polar decoders and how to design polar decoders for variable throughput and latency requirements which widely exist in applications. Simulation results and complexity analysis are provided as well to demonstrate advantages of our proposed methods and algorithms. Hardware implementation results are also presented. We briefly summarize our main contribution as follows.

In Chapter 2, two improvements to the soft- and hard-reliability based MLGD algorithms for non-binary LDPC codes are proposed. The one is the new reliability information update, that helps the reliability-based MLGD algorithms achieve better error performance, smaller average iteration numbers, and lower complexities. The other is the re-selection scheme, which is intentionally used to lower the error floor, and also results in a better error performance, fewer iterations on average. Though additional computational resources are needed by the re-selection scheme, the MLGD algorithms with the re-selection scheme still have lower computational complexities than the existing MLGD algorithms.

In Chapter 3, the symbol-based recursive channel combination are proposed to calculate the symbol-wise channel transition probability. It shows that based on the LL representation of the transition probability, this recursive procedure needs fewer additions than methods used in existing parallel polar decoders and polar decoders with multi-bit decisions. To simplify the path pruning problem of SCL polar decoders, a two-stage list pruning network is proposed. Meanwhile, the PCMS technique is used to reduce the memory requirement for list decoders. With these proposed ideas, an efficient architecture for symbol-decision SCL decoders is designed. Specifically, two scheduling schemes are introduced to perform the hardware sharing for different configurations. A folded sorting implementation and tree sorting implementation are also investigated. Furthermore, symbol-decision SCL polar decoders for two-bit, four-bit and eight-bit, respectively, are implemented with a list size of four. Our proposed methods and architecture provide a range of trade-offs between area, throughput and area efficiency for SCL polar decoders.

In Chapter 4, we first propose a method to reduce the complexity of a local ML decoding unit. We discuss the divide-and-conquer ML decoding technique of SC-based algorithms in the probability domain. Then we extend this idea to SCL-based polar decoder and design a low-complexity divide-and-conquer AML decoding unit for SCL-based polar decoder to reduce the computational complexity of a local ML decoding unit. Frozen-location patterns of polar codes constructed for binary erasure channels and AWGN channels are investigated as well. Our research shows that only a small number of frozen-location patterns exist in polar codes constructed for a binary erasure channel or an AWGN channel. This enables us to develop an efficient hardware design for a simplified divide-and-conquer AML decoding unit with less complexity. Furthermore, the MM-SCL polar decoder is discussed in this chapter to adapt to different throughput and latency requirements which widely exist in current communication applications. This decoder supports SCL algorithms with different list sizes and parallelism. It decodes multiple received words in parallel with a small list size if a high throughput or small latency is required. When a better error performance is desired, the MM-SCL decoder switches to a mode with a greater list size. Implementation results demonstrate advantages of our proposed ideas and architecture of the MM-SCL polar decoder.

5.2 Future work

Non-binary LDPC codes and polar codes are two of the most attractive topics in the fields of error correcting codes currently. There are lots of potential research directions for this two codes, such as code constructions, an exploration of error floor, efficient architecture designs for both encoders and decoders. Based on our research results, the following potential future follow-up work may be carried out,

- In Chapter 2, we have discussed improved MLGD algorithms with a soft-information initialization. Our idea can be extended to MLGD algorithms with a hard-information initialization, because there are some applications, such as hard disk and data storage applications, which provide only hard decisions from their communication channel. Therefore, it is reasonable and meaningful to extend our ideas to such kind of applications and to improve and to optimize our methods further for hard-information channels. Furthermore, the hardware implementation for proposed improved MLGD algorithms with the re-selection scheme are worthy to be investigated as well.

- The capacity-achieving performance of polar codes is only achieved when the code length is infinite. Furthermore, the upper bound under successive cancellation decoding in [66] is satisfied when the code length is large enough. But for a real application, a large code length might not be satisfied. Therefore, naturally, the question is that how good the error performance of polar codes is under SC or SCL decoding when the code length is moderate or short. Currently, it takes a very long time to emulate the block error performance as low as the level of 10^{-12} by using a software polar decoder. Hence, how to build an area-efficient high-throughput polar decoder hardware emulator based on our research results would be an interesting and meaningful research topic.

- The proposed methods and decoder architectures in Chapters 3 and 4 are meant for polar codes in general and not tied with any particular application. How to apply polar codes to real applications is a practical problem we need to solve in the near future. Based on the research results of polar codes in the past ten years, polar codes have at least the same error performance compared to Turbo codes and LDPC codes which are used widely by current telecommunication standards.

However, Turbo codes and LDPC codes have error floor issues such that it is a big challenge to apply these two families of codes to some applications with stringent error performance requirements for a very low bit error rate region, such as fiber-optical communication ($10^{-10} \sim 10^{-12}$) and data storage application ($10^{-12} \sim 10^{-15}$). Fortunately, the research in [67] shows that polar codes have a large girth of 12 and show superior error floor performance compared to Turbo codes and LDPC codes. Hence, polar codes are a better candidate of FEC technology than turbo codes and LDPC codes for the fiber-optical communication and the data storage application regarding the error performance. How to apply polar codes to the fiber-optical communication and the data storage application should be a promising research topic.

Bibliography

[1] B. Li, H. Shen, D. Tse, and W. Tong, "Low-latency polar codes via hybrid decoding," in *Proceedings of 2014 8th International Symposium on Turbo Codes and Iterative Information Processing*, Aug. 2014, pp. 223–227.

[2] B. Li, H. Shen, and D. Tse, "Parallel decoders of polar codes," arXiv:1309.1026, Sep. 2013. [Online]. Available: http://arxiv.org/abs/1309.1026

[3] B. Yuan and K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2268–2280, Oct. 2015.

[4] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[5] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[6] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[7] A. Cohen and K. Parhi, “A low-complexity hybrid LDPC code encoder for IEEE 802.3an (10GBase-T) ethernet,” *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 4085–4094, Oct. 2009.

[8] Q. Huang, J. Kang, L. Zhang, S. Lin, and K. Abdel-Ghaffar, “Two reliability-based iterative majority-logic decoding algorithms for LDPC codes,” *IEEE Transactions on Communications*, vol. 57, no. 12, pp. 3597–3606, Dec. 2009.

[9] J. Kang, Q. Huang, S. Lin, and K. Abdel-Ghaffar, “An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes,” *IEEE Transactions on Communications*, vol. 59, no. 1, pp. 64–73, Jan. 2011.

[10] M. C. Davey and D. MacKay, “Low-density parity check codes over GF(q),” *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, Jun. 1998.

[11] H. Song and J. R. Cruz, “Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording,” *IEEE Transactions on Magnetics*, vol. 39, no. 2, pp. 1081–1087, Mar. 2003.

[12] X. Jiang and M. H. Lee, “Large girth non-binary LDPC codes based on finite fields and euclidean geometries,” *IEEE Signal Processing Letters*, vol. 16, no. 6, pp. 521–524, Jun. 2009.

[13] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, “Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions,” *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1652–1662, Jun. 2009.

[14] W. Tang, J. Huang, L. Wang, and S. Zhou, “Nonbinary LDPC decoding by min-sum with adaptive message control,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2011, pp. 3164–3167.

[15] D. J. MacKay and M. C. Davey, “Evaluation of Gallager codes of short block length and high rate application,” in *Proceedings of IMA workshop on Codes, Systems and Graphical Models*, 2000, pp. 113–130.

[16] L. Barnault and D. Declercq, “Fast decoding algorithm for LDPC over GF(2^q),” in *Proceedings of IEEE Information Theory Workshop*, 2003, pp. 70–73.

[17] H. Wymeersch, H. Steendam, and M. Moeneclaey, “Log-domain decoding of LDPC codes over GF(q),” in *Proceedings of IEEE International Conference on Communications*, vol. 2, Jun. 2004, pp. 772–776.

[18] D. Declercq and M. Fossorier, “Decoding algorithms for nonbinary LDPC codes over GF(q),” *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 633–643, Apr. 2007.

[19] V. Savin, “Min-max decoding for non binary LDPC codes,” in *Proceedings of IEEE International Symposium on Information Theory*, Jul. 2008, pp. 960–964.

[20] C.-Y. Chen, Q. Huang, C.-C. Chao, and S. Lin, “Two low-complexity reliability-based message-passing algorithms for decoding non-binary LDPC codes,” *IEEE Transactions on Communications*, vol. 58, no. 11, pp. 3140–3147, Nov. 2010.

[21] K. Kasai and K. Sakaniwa, “Fourier domain decoding algorithm of non-binary LDPC codes for parallel implementation,” in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2011, pp. 3128–3131.

[22] F. Cai and X. Zhang, “Efficient check node processing architectures for non-binary LDPC decoding using power representation,” in *Proceedings of IEEE Workshop on Signal Processing Systems*, Oct. 2012, pp. 137–142.

[23] X. Zhang, F. Cai, and S. Lin, “Low-complexity reliability-based message-passing decoder architectures for non-binary LDPC,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 1938–1950, Nov. 2012.

[24] 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, July 2009.

[25] E. Sasoglu, I. Telatar, and E. Arıkan, “Polarization for arbitrary discrete memoryless channels,” in *Proceedings of IEEE Information Theory Workshop*, Oct 2009, pp. 144–148.

[26] K. Niu and K. Chen, “CRC-aided decoding of polar codes,” *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, October 2012.

[27] A. Eslami and H. Pishro-Nik, “A practical approach to polar codes,” in *Proceedings of IEEE International Symposium on Information Theory*, July 2011, pp. 16–20.

[28] E. Arıkan, “Systematic polar coding,” *IEEE Communications Letters*, vol. 15, no. 8, pp. 860–862, August 2011.

[29] E. Arıkan, H. Kim, G. Markarian, U. Ozgur, and E. Poyraz, “Performance of short polar codes under ML decoding,” in *Proceedings of ICT Mobile Summit Conference*, 2009.

[30] S. Kahraman and M. Celebi, “Code based efficient maximum-likelihood decoding of short polar codes,” in *Proceedings of IEEE International Symposium on Information Theory*, July 2012, pp. 1967–1971.

[31] 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, February 2014.

[32] I. Tal and A. Vardy, "List decoding of polar codes," in *Proceedings of IEEE International Symposium on Information Theory*, July 2011, pp. 1–5.

[33] ——, "List decoding of polar codes," *IEEE Transactions on Information Theory,*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[34] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1*, IEEE Std. 802.16e-2005, Mar. 2006.

[35] C. Leroux, I. Tal, A. Vardy, and W. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2011, pp. 1665–1668.

[36] C. Leroux, A. Raymond, G. Sarkis, and W. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, Jan 2013.

[37] A. Raymond and W. Gross, "A scalable successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 62, no. 20, pp. 5339–5347, Oct. 2014.

[38] A. Alamdar-Yazdi and F. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.

[39] C. Zhang and K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 2, pp. 115–119, Feb. 2014.

[40] G. Sarkis and W. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, vol. 17, no. 4, pp. 725–728, Apr. 2013.

[41] C. Zhang and K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2429–2441, May 2013.

[42] J. Lin and Z. Yan, "Efficient list decoder architecture for polar codes," in *Proceedings of IEEE International Symposium on Circuits and Systems*, June 2014, pp. 1022–1025.

[43] A. Balatsoukas-Stimming, A. Raymond, W. 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.

[44] J. Lin and Z. Yan, "An efficient list decoder architecture for polar codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2015, accepted and to appear, available on IEEE Explore, DOI: 10.1109/TVLSI.2014.2378992.

[45] 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.

[46] L. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. A. S. Abdel-Ghaffar, "Construction of nonbinary cyclic, quasi-cyclic and regular LDPC codes: A finite geometry

approach," *IEEE Transactions on Communications*, vol. 56, no. 3, pp. 378–387, Mar. 2008.

[47] C. Xiong and Z. Yan, "Improved iterative soft-reliability-based majority-logic decoding algorithm for non-binary low-density parity-check codes," in *Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers*, Nov. 2011, pp. 894–898.

[48] ——, "Low-complexity layered iterative hard-reliability-based majority-logic decoder for non-binary quasi-cyclic LDPC codes," in *IEEE International Symposium on Circuits and Systems*, May 2013, pp. 1348–1351.

[49] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[50] T. Richardson, "Error floors of LDPC codes," in *Proceedings of Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 2003, pp. 1426–1435.

[51] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright, "Lowering LDPC error floors by postprocessing," in *Proceedings of IEEE Global Telecommunications Conference*, 2008, pp. 1–6.

[52] A. Poloni, S. Valle, and S. Vincenti, "NB-LDPC: Absorbing set and importance sampling," in *Proceedings of 7th International Symposium on Turbo Codes and Iterative Information Processing*, Aug. 2012, pp. 101–105.

[53] C. Xiong and Z. Yan, "Improved iterative hard- and soft-reliability based majority-logic decoding algorithms for non-binary low-density parity-check codes," arXiv:1408.3357, Aug. 2014. [Online]. Available: http://arxiv.org/abs/1408.3357

[54] K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 7, pp. 745–754, July 1991.

[55] W. Park and A. Barg, "Polar codes for q-ary channels, $q = 2^r$," *IEEE Transactions on Information Theory*, vol. 59, no. 2, pp. 955–969, February 2013.

[56] C. Xiong, J. Lin, and Z. Yan, "Error performance analysis of the symbol-decision SC polar decoder," arxiv:1501.01706, 2015. [Online]. Available: http://arxiv.org/abs/1501.01706

[57] K. E. Batcher, "Sorting networks and their applications," in *AFIPS Proceeding of the Spring Joint Computer Conference*, 1968, pp. 307–314.

[58] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014.

[59] C. Xiong, J. Lin, and Z. Yan, "Symbol-based successive cancellation list decoder for polar codes," in *Proceedings of IEEE Workshop on Signal Processing Systems*, Belfast, UK, October 2014, pp. 198–203.

[60] ——, "Symbol-decision successive cancellation list decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 64, no. 3, pp. 675–687, Feb. 2016.

[61] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *Proceedings of IEEE Workshop on Signal Processing*

Systems (SiPS 2014), Belfast, UK, October 2014, pp. 56–61. [Online]. Available: http://arxiv.org/abs/1405.4819

[62] ——, “A high throughput list decoder architecture for polar codes,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, accepted and to appear, available on IEEE Explore, DOI: 10.1109/TVLSI.2015.2499777.

[63] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, “Increasing the speed of polar list decoders,” in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–6.

[64] E. Arıkan, “A performance comparison of polar codes and Reed-Muller codes,” *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, June 2008.

[65] D. Wu, Y. Li, and Y. Sun, “Construction and block error rate analysis of polar codes over AWGN channel based on Gaussian approximation,” *IEEE Communications Letters*, vol. 18, no. 7, pp. 1099–1102, July 2014.

[66] E. Arıkan and I. Telatar, “On the rate of channel polarization,” in *Proceedings of IEEE International Symposium on Information Theory*, June 2009, pp. 1493–1495.

[67] A. Eslami and H. Pishro-Nik, “On finite-length performance of polar codes: Stopping sets, error floor, and concatenated design,” *IEEE Transactions on Communications*, vol. 61, no. 3, pp. 919–929, March 2013.

Vita

Chenrong Xiong received the B.E. degree in electrical engineering and the M.E. degree in microelectronics from Tsinghua University, Beijing, China, in 2001 and 2004, respectively.

Since 2010, he has been working towards the Ph.D. degree in the Department of Electrical and Computer Engineering at Lehigh University, Bethlehem, PA under supervision of Prof. Zhiyuan Yan.

From 2004 to 2010, he was with Beijing Embedded System Key Lab, Beijing, working on VLSI architecture design for digital signal processing and communication systems. His research interests include channel coding technology and VLSI architecture design for digital signal processing and communication systems.