February 2016
Master's Degree Thesis

Improved Belief Propagation Algorithm and its Hardware Architecture for Short Polar Codes

Graduate School of Chosun University

Department of Information and Communication Engineering

Shajeel Iqbal

Improved Belief Propagation Algorithm and its Hardware Architecture for Short Polar Codes

짧은 극 부호의 개선된 BP 알고리즘 및 하드웨어 구조

February 25, 2016

Graduate School of Chosun University

Department of Information and Communication Engineering

Shajeel Iqbal

Improved Belief Propagation Algorithm and its Hardware Architecture for Short Polar Codes

Advisor: Prof. GoangSeog Choi, Ph.D.

A thesis submitted in partial fulfillment of the requirements for a Master's degree

October 2015

Graduate School of Chosun University

Department of Information and Communication Engineering

Shajeel Iqbal

샤질 이크발의 석사학위논문을 인준함

위원장 조선대학교 교수 신영숙 (인)

위 원 조선대학교 교수 최광석 (인)

위 원 조선대학교 교수 김영식 (인)

2015 년 11 월

조선대학교 대학원

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Improved Belief Propagation Algorithm and its Hardware Architecture for Short Polar Codes

Shajeel Iqbal

Advisor: Prof. GoangSeog Choi, Ph.D.

Department of Information and Communication Engineering

Graduate School of Chosun University

This thesis has its focus on two topics: 1. Improvement of performance of polar codes using belief propagation (BP) decoding method, and 2. its hardware architecture. Polar codes were first introduced by Arikan [1] and are known for theoretically achieving Shannon's capacity for discrete memory-less channels. The encoder and decoder for polar codes can be implemented with a complexity of $O(N \log N)$ [1], where N is the code block-length. This complexity can be achieved by using the successive cancellation (SC) decoding algorithm. However, when compared to the other famous coding schemes, like low-density parity-check (LDPC) codes and turbo codes, the performance of polar codes is disappointing in the finite length regime [2], [3].

Thus in the first part of this thesis, we investigate the finite length performance of polar codes, over the additive white Gaussian noise channel (AWGN), while assuming BP decoding as the decoding method. It is suggested to run the BP

decoder on the parity check matrix of polar codes. In addition to this, since BP is rather well-studied for LDPC codes, there are many proposed approaches to modify BP in order to obtain better BER performance. Therefore, we proposed a modified version of BP employing damped belief propagation algorithm [4]. Here we show that by modifying this algorithm for polar codes we can achieve significant improvement in BER. The BP algorithm is modified by changing the update equation of the variable nodes. The variable nodes are updated by multiplying them with their previous messages. It is found that the parity-check matrix based tanner graph can be used for the decoding of polar codes, and that the performance of short polar codes can be improved by 1-2 dB using the proposed BP decoder, while keeping the complexity of the original BP decoder. Furthermore, we show that the proposed decoder requires about 10-25 iterations less than the standard BP decoder. The second part of this thesis is devoted to the hardware architecture of polar codes, in which we discuss the various hardware architectures available for polar codes. We also provide the details of the hardware architecture based on the proposed BP decoder.

한 글 요 약

짧은 극 부호의 개선된 BP 알고리즘 및 하드웨어 구조

샤질 이크발
지도 교수:최광석
정보 통신공학과
조선대학교 대학원

이 논문은 두 가지 주제에 관해 초점을 가지고 있다. 1. BP 복호 방법을 이용한 극 부호의 성능 개선, 2. 극 부호의 하드웨어 구조이다. 극 부호는 처음 Arikan 에 의해 도입 된 것으로, 개별 메모리 없는 채널에서 Shannon 용량에 이론적으로 근접한다고 알려진다. 극 부호에 대한 부호기와 복호기 복잡도는 O(NlogN)로 구현된다. 여기서, N 은 부호의 블록 길이이다. 이러한 복잡도는 연속 소거(SC) 복호 알고리즘을 사용함으로써 구현 될 수 있습니다.

저밀도 패리티-체크(LDPC)코드, 터보 부호와 같이 다른 유명한 코딩 형태와 비교할 때, 극 후보의 성능은 유한한 길이에서 실망스럽다.

그래서, 본 논문의 첫 번째 부분에서는, 백색 가우시안 잡음 채널(AWGN)상에 극 부호의 유한한 길이의 성능을 조사한다. 여기서, 복호 방식으로 BP 복호를 가정한다.

BP 코드는 LDPC 코드에 비해 잘 연구되어 있고 더 나은 BER 성능을 얻기 위해서는 BP 를 수정하는 많은 제안들이 있습니다. 따라서 BP 논의 감쇠 신뢰도 전달 알고리즘을 이용하여 수정 한 버전을 제안합니다. 다음은 코드에 대한 극성이 알고리즘을 수정하여 우리가 BER 에서 상당한 개선을 있음을 보여 줍니다. BP 알고리즘은 변수 노드의 이전 메시지와 함께 그것들을 변수 노드의 갱신 방정식을 변경하여 수정됩니다다. 이 패리티 검사는 행렬을 기반으로 태너 그래프 극 부호의 복호를 위해 사용될 수 있다는 것을 알 수 있고, 복잡성을 유지하면서 짧은 극 부호의 성능이 본 논문에서 제안하고 있는 BP 디코더를 사용함으로써 1-2db 정도 개선 될 수 있다는 것입니다.

제안하는 복호는 표준 약 10~25 번의 반복을 덜어 줍니다. 이 논문의 두번째 부분은 극 부호에 사용할 수 있는 다양한 하드웨어 구조를 다루며. 저희가 제안한 BP 디코더를 기반으로 하드웨어 구조의 세부사항을 제공합니다.

1. INTRODUCTION

1.1 Motivation

In recent years, digital communications have become very important part of the human lives: mobile phones, computer networks, wireless watches, CD, DVD, HD television broadcasts and many more applications. All these forms of communications transfer the data in the form of digital information which may lead to data corruption due to noise. This accelerated the demand for efficient and reliable data transmission. Retransmission of data is usually costly or sometimes impossible, thus the systems designer must use the error correction codes (ECC) to recover the original information.

In 1948, Claude Shannon [5] showed that the reliable transmission of information is possible over the noisy channel by adding the redundant information to the input data, as long as the information rate is less than the capacity of the channel. Since Shannon's work a lot of efforts have been put to devise efficient encoding and decoding methods to recover from errors in a noisy channel. Although he proved the existence of capacity achieving codes, but he did not introduced any method to construct such codes. In 1960's, Gallager proposed the low-density parity-check (LDPC) codes [6], which were proven to operate very close the Shannon's capacity. However they were impractical at that time, until they were rediscovered in 1990's. Meanwhile, turbo codes were widely used in the digital communication systems, introduced by Berrou, Glavieus and Thitimajshima in 1993 [7].

Both turbo and LDPC codes perform very close to capacity, however, they cannot achieve the channel capacity. In 2008, Arikan in [1] introduced the polar codes which are known for theoretically achieving Shannon's capacity for discrete memory-less channels (DMCs). The encoder and decoder for polar codes can be implemented with a complexity of $O(N \log N)$, where N is the code block-length. This complexity can be achieved by using the successive cancellation (SC) decoding algorithm. However, when compared to the other famous coding schemes, like low-density parity-check (LDPC) codes and turbo codes, the performance of polar codes is disappointing in the finite length regime [2], [3]. Since the invention of polar codes, many researchers have proposed different algorithms to improve their performance. The authors in [8] generalize the SC decoder, proposing the successive-cancellation list (SCL) decoder, showing that the performance of the SCL decoder is close to the maximum-likelihood (ML) decoding. In order to improve the time and space complexity of the SC decoder, successive-cancellation stack (SCS) algorithms were proposed [9]. The successive-cancellation hybrid (SCH) algorithm [10] combines the features of SCL and SC and approaches the performance of ML decoding with acceptable complexity. These SC decoders, when aided with CRC such as CA-SCL and CA-SCS [11], can perform comparable to state-of-the-art LDPC codes. All of the decoders based on the SC decoder have serial architecture and, hence, pose the challenge of achieving high throughput.

However the performance of polar codes at shorter length is poor as shown in Figure 1.1, so Arikan introduced the BP decoding of polar codes [1]. He also

compared the performance of polar codes under BP decoding with Reed-Muller codes [12]. He showed that polar codes perform better under the BP decoder than

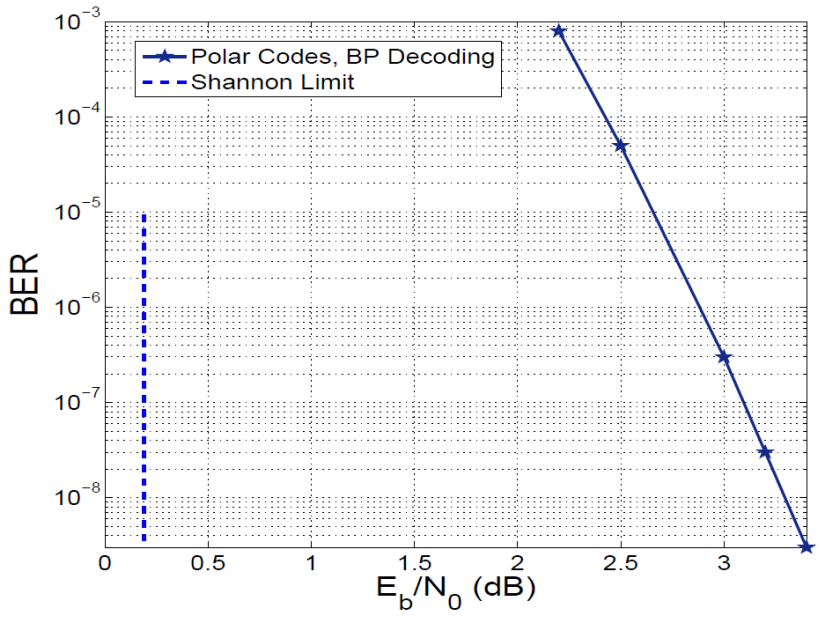


Figure 1.1: BER performance of polar codes under belief propagation (BP) decoding over the binary Gaussian channel, compared to the Shannon limit for error correcting codes. The code-length and code-rate are 2^{13} and ½, respectively. As you can see, polar codes stay far away from the capacity when used in finite block lengths [24].

under the SC decoder for binary erasure channel (BEC). But this is not true for the AWGN channel because of the message passing schedule. Short polar codes were also investigated under the ML [13] decoder to achieve ML performance, but decoding of polar codes using the ML decoder suffers from higher complexity [14]. Motivated by this, in this thesis we will try to improve the performance of short polar codes.

1.1.1 Desired Decoder

A desired decoder is the one which provides good bit error rate but at the same time it should perform very well when it is implemented on the hardware, as the main aim designing a decoder is its efficient implementation on the hardware. While implementing the decoder on the hardware usually power consumption, area utilization and throughput are the main parameters that are usually considered. Thus the following aspects of the decoders must be considered while implementing:

1- ***Complexity:*** The complexity of the decoder is related to the number of computations that are generally performed which includes additions, multiplications and moving/copying operations. Thus if the number of these operations increases then the computational complexity of the decoder will also increase. The increase in computational complexity means increased amount of operations which directly relates to the increased area, power and throughput. Therefore, a desired decoder must have low number of operations while keeping the bit error rate to minimum.

2- ***Memory Requirement***s: While performing the computational operations the data in the decoder need to be saved in the registers or may be in RAM/ROMs. Thus a decoder needs memory as well while performing its operations. In general, increased number of operations means increased requirement of memory, hence, more area, power and throughput. Therefore, a desired decoder must reduce the memory as much as possible.

3- ***Throughput:*** Latency is defined as the number of clock cycles required to process one bit, and it is inversely related to the throughput of the decoder. A

decoder with higher latency requires a higher clock then the one with lower latency to achieve the same throughput. In general, the higher clock rate results in

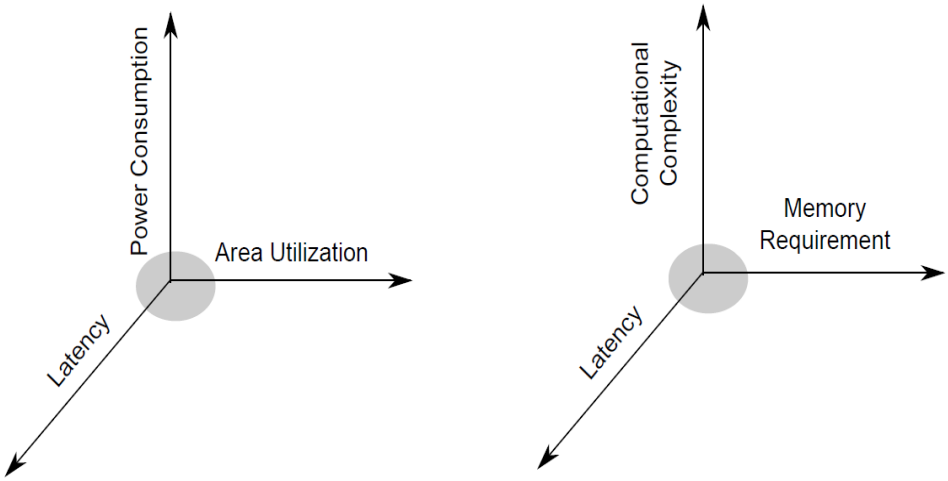


Figure 1.2: A good decoder lies in the region close to the origin in three dimensional spaces of power, area and latency. This region directly corresponds to a region close to the origin in three dimensional spaces of computational complexity, memory requirement and latency [15].

increased power consumption of hardware. Therefore, a desired decoder should minimize its latency as much as possible.

Thus we conclude that, power consumption, area and latency are the three main factors to be considered for the decoder. Figure 1.2 summarizes this section and shows that a practically feasible decoder lies close to the origin.

1.1.2 Desired Encoder

An encoder must provide better error rates. If the encoder itself is weak then the decoder cannot provide the near optimal performance. Thus the main job of the encoder is that it should produce a lower error rate in a channel when decoded by a specific decoder.

1.2 Contributions

In the previous section, we elaborated by different references that the performance of polar codes in the finite regime is not good. While in practical applications we need codes of practical length, thus the polar codes must be studied for their practical use. Furthermore, the proposed decoders must be implemented on the hardware for confirming their use in practical applications. We investigated both the algorithm and hardware of polar codes and will be presented in this thesis.

1.2.1 Improved BP Decoding of Polar Codes

In this thesis, the BP decoding of polar codes is studied. It is suggested to run the BP decoder on the parity check matrix of polar codes. In addition to this, since BP is rather well-studied for LDPC codes, there are many proposed approaches to modify BP in order to obtain better BER performance. Therefore, we proposed a modified version of BP employing damped belief propagation algorithm. The algorithm was studied in [4] and was shown to achieve good BER performance. Here we show that by modifying this algorithm for polar codes we can achieve significant improvement in BER. The BP algorithm is modified by changing the update equation of the variable nodes. The variable nodes are updated by multiplying them with their previous messages. It is found that the parity-check-based Tanner graph can be used for the decoding of polar codes, and that the performance of short polar codes can be improved significantly by using the proposed BP decoder, while keeping the complexity almost the same.

1.2.2 Hardware Architecture of Polar Codes

In order to address the practical use of proposed BP algorithm for polar codes, we discussed the hardware architecture for the proposed BP decoder. Successive cancellation (SC) decoder is the most popular decoder for the polar codes, thus first we discuss the hardware architecture of SC decoder and its performance. Then we discuss the various variants of SC decoders and their performances. Later we discuss the soft decoders for polar codes especially the hardware architecture of belief propagation decoder and compare its performance with the SC decoder. In the end we discuss in detail the hardware architecture of the proposed decoder.

1.3 Outline

The remainder of this thesis is divided as follows: chapter 2 presents a theoretical description of polar codes, introduces the notation used throughout this thesis, and provides a survey of relevant literature. In chapter 3, we discuss the standard belief propagation algorithm in detail and the proposed modified belief propagation as well. Chapter 4 describes the various hardware architectures for polar codes including soft and hard output polar decoders. The hardware architecture of proposed BP algorithm is also discussed in this chapter. We will conclude this thesis in chapter 5.

2. THEORETICAL BACKGROUND

This chapter provides the necessary background required for the rest of the thesis. We begin by providing the brief history of channel codes, and then we discussed some of the basic channel models. Next we give an introduction to block codes along with description of their two important types, namely the single parity-check and repetition codes. After that is an introduction of polar codes, their factor graphs construction and their construction for binary erasure and additive white Gaussian noise channels. Later, we discuss the original successive cancellation decoder and its different variants available. In the end, we present a brief overview if the stat-of-the-art decoders available for polar codes.

2.1 Coding Theory: A brief history

Claude Shannon in his paper [5] “A Mathematical Theory of Communication”, 1948, wrote that “the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at one point”. According to Shannon, message refers to or is correlated according to some system with certain physical or conceptual entities. However the solution of this fundamental communication problem is theoretical. The ideology of this problem is related to the transmitting or receiving the message signal. In conventional procedure, we should encode the selected message by adding some redundant information, such that even if the transmitted encoded message is

corrupted by noise, there will be sufficient redundancy in it to recover the original message. Regarding these statement two questions arises: 1. How much redundancy is required? This is related to quantitative question. Shannon showed the answer regarding the quantitative question and proved that for reliable transmission of data, there is a certain limit to the information rate over a noisy channel. According to Shannon's theorem, for a communication channel C, the channel capacity $cap(c) \in [0,1]$ and the information rate R are related as $R < cap(C)$ for reliable communication. That means there exists a reliable coding scheme for information rate R. 2. What kind of redundancy is the best choice? This is related to qualitative question. These are two interesting questions. At the receiving end, the original message recovery depends on the amount of redundancy. Therefore, how many redundant bits are required to recover the transmitted messages is an important question. Alternatively, it makes sense that what is the optimum use if the communication resources at this disposal, e.g., of channel bandwidth. Each and every coding scheme assigns a value know as information rate R that means what portion of the transmitted signal is useful. The qualitative question is seeking for actual coding schemes, which should not only optimally use the communication resources, but also be equipped with the set of encoding and decoding algorithms, which can be performed practically and efficiently. For this reason, the aim of the code designer is to apply the coding scheme in such a way so that the maximum information rate may be achieved with a vanishing probability of decoding error.

Over the last few decades coding theory has evolved tremendously. Researchers from the various fields of mathematics, computer sciences and engineering have been conducting research on it, posing and answering problems of both theories and practical implementation. But still the race for efficient coding scheme is on-going. Among many researchers, Berrou, Glavieux and Thitimajshima [7], introduced in 1993 a novel and apparently revolutionary error-control coding technique, which was called as turbo coding. This coding technique consists of a parallel concatenation of two binary convolutional codes, decoded by an iterative decoding algorithm. The bit error rate (BER) performance of these codes was excellent and was considered as the first practical codes to achieve the channel capacity. Turbo codes play the vital role in the field of error correction coding. It is employed in mobile communications, satellite communications and in many wireless networks standards.

Following this stimulating discovery, 3 years later in 1996 Mackay and Neal [16, 17] rediscovered the LDPC codes which were first introduced by Gallager in 1962. These codes can be constructed by generating a sparse parity-check matrix H which contains relatively less number of ‘1s’ then ‘0s’. Gallager also presented an iterative method of decoding these codes, which was capable of achieving excellent performance. Long LDPC codes with iterative decoding based on belief propagation (BP) have been shown to achieve an error performance only a fraction of decibel away from Shannon limit. LDPC codes have some advantages over turbo codes: 1. they do not need a long interleave for good performance; 2. they have better block error rate (BLER) performance; and 3. their decoding is not

trellis based. Soon after the rediscovery of LDOC codes, it has been realized that the iterative decoder of LDPC codes is a belief propagation decoder. Eventually, it has also been shown that decoding of turbo codes is another representation of belief propagation algorithm [18].

Until now, many turbo and LDPC codes have been proposed which empirically achieve rates close to capacity for various channels. However, none of these codes are proven to achieve capacity for channels other than the BEC. In this thesis, we discuss polar codes. Polar codes, recently introduced by Arıkan [1], are a family of codes that provably achieve the capacity of symmetric channels with ″low encoding and decoding complexity″. This settles the long standing open problem of achieving capacity with low complexity.

2.2 Fundamentals of Channel Coding

Channel coding plays a vital role for the reliable information transfer between a sender and a receiver of a communication system. The signal carrying information bits is propagated through a medium called channel, due to the response if this channel the signal carrying information bits is seriously affected by the noise. Thus at the receiver end, we receive corrupted bits which need to be corrected. So successful correction of errors depend upon the channel response. Therefore, a coding scheme like channel coding scheme is necessary to remove the unwanted data from the information bits.

2.2.1 Channel Models

A typical transmission system consists of source encoder/decoder and channel encoder/decoder with a noisy channel as a medium of transferring the data as shown in Figure 2.1. Here we are mainly concerned with the channel encoder and decoder. This Figure introduces notation and concepts used $\hat{u}$ throughout this thesis. Under this model, a user receives an input vector $u_1^N = (u_1, u_2, ..., u_N)$, where N is the code length, from the source encoder. The channel encoder then adds redundancy to this vector $\hat{u}_1^N$ and generates a codeword $x_1^N = (x_1, x_2, ..., x_N)$. This codeword x_1^N is then transmitted through a noisy channel which introduces noise to it.

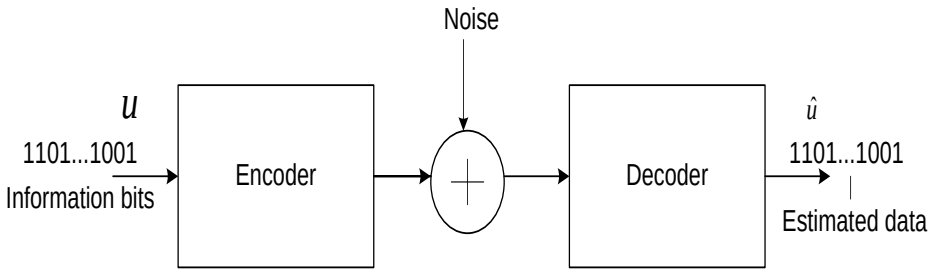


Figure 2.1: The basic communication scenario with encoder and decoder, and a communication channel.

The decoder then observes a sequence of corrupted symbols, i.e., a received vector $y_1^N = (y_1, y_2, ..., y_N)$ and estimates the information vector $\hat{u}_1^N$ based on y_1^N . The ratio $R = k / N$ is referred to as the code rate; it reflects the amount of redundancy

added to the data stream by the channel encoder. Vectors u, x, y can be realized as random variables, U *on* U^k , X *on* X^n and Y *on* Y^N , respectively. Similarly, each u_i, x_i *and* y_i is a realization of scalar random variables U_i, X_i *and* Y_i respectively. In addition, we assume that each of them is independent and identically distributed (i.i.d) according to the probability density function $P_U(u)$, $P_X(x)$ *and* $P_Y(y)$ respectively. The relationship between Y and Z is modeled by a conditional probability density function $P_{Z|y}(Z|Y)$. The meaning of communication channel is to specify its probability density function. Figure. 2.2 shows the three communication channels names as symmetric channel, erasure channel and AWGN channel [19].

2.2.1.1 Binary Discrete Memory-less Symmetric Channels (BDMS)

Throughout this thesis we assume that the channel models are binary –input discrete memory-less channels (BDMS). Figure 4 shows the model of binary-input discrete memory-less channel. In case of memory-less channels, for any input vector $u_1^N = (u_1, u_2, ..., u_N)$ there is an output vector $y_1^N = (y_{1,} y_2, ..., y_N)$, from the alphabet $y_i \in Y$. These channels have a binary codeword symbol alphabet Y represented as $F_2 = \{0, 1\}$ or as {-1, +1}. These channels have no memory which means that the output these channels at any time instant depends on the input at that time instant, i.e. $P_{z|y}(\mathrm{z}|y) = \prod_{j=1}^{n} P_{z_j|y_j}(\mathrm{z}_j|y_j)$. The channel is symmetric because the input is symmetric. The maximum amount of information

per symbol that can be conveyed about the codeword X from the received word Y in the case memory-less channel C, is referred to as the channel capacity [18]: $Cap(C) = \sup_{P_Y(y)} I(Y;Z)$. Where sup is supreme function and I(X;Y) denotes mutual information between the random variables X and Y. According to Shannon's theorem for reliable transmission the value of code rate R satisfies the condition R<cap (C) [19].

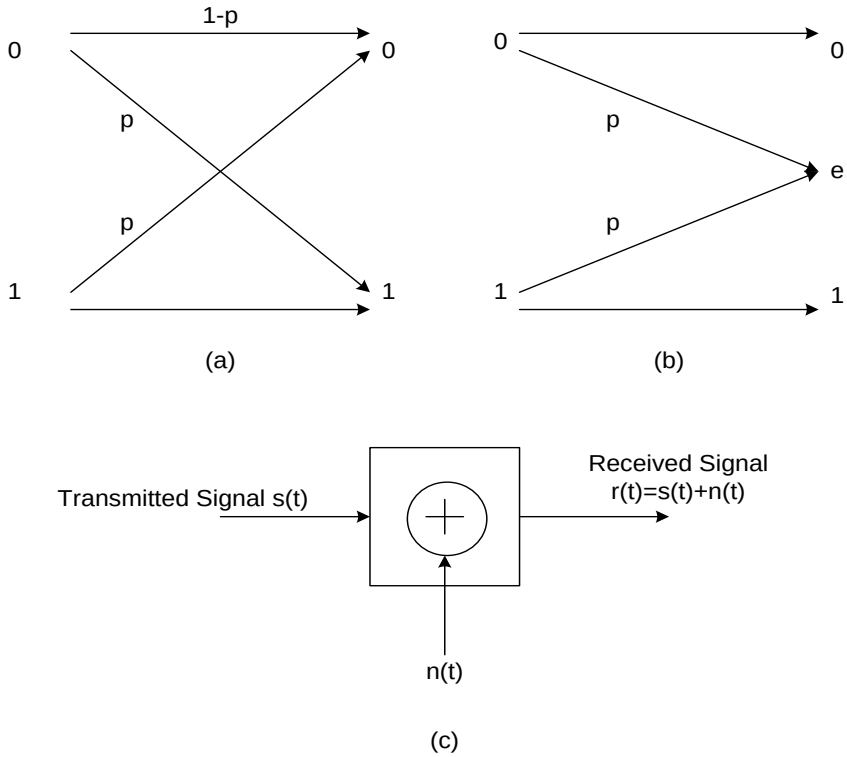


Figure 2.2: Three communication channels. (a) Memory-less symmetric channel. (b) Binary erasure channel. (c) AWGN channel [19].

2.2.1.2 Binary Erasure Channel (BEC)

The binary erasure channel (BEC) is the simplest non-trivial channel model imaginable. It was introduced by Elias as a toy example in 1954. The emergence of the internet promoted the erasure channel into the class of "real world" channels. Indeed, erasure channels can be used to model data networks, where packets either arrive correctly or are lost due to buffer overflows or excessive delays [19].

Erasure channels model situations where information may be lost but is never corrupted. The BEC captures erasure in the simplest form: single bits are transmitted and either received correctly or known to be lost. The decoding problem is to find the values of the bits given the locations of the erasures and the non-erased part of the codeword. Figure 2.2(b) depicts BEC(e). Time, indexed by t, is discrete the transmitter and receiver are synchronized. The channel input at time t, denoted by X_t , is binary: $X_t \in \{0,1\}$. The corresponding output Y_t takes on values in the alphabet {0, 1, *}, where * indicates an erasure. Each transmitted bit is either erased with probability e, or received correctly: $Y_t \in \{X_t, *\}$ and $P\{Y_t = *\} = e$. Erasure occurs for each t independently. For this reason we say that channel is memory-less. The capacity of BEC(e) is $C_{BEC}(e) = 1 - e$ bits per channel use [19].

2.2.1.3 Binary Additive White Gaussian Noise Channel (BAWGNC)

A basic and generally accepted model for thermal noise in communication channels is the assumption that, 1. the noise is additive, i.e., the received signal equals the transmit signal plus some noise, where the noise is statistically independent of the signal, 2. the noise is white, i.e., the power spectral density is flat, 3. The noise samples have a Gaussian distribution. Figure 2.2(c) depicts the binary additive white noise Gaussian noise channel with noise variance σ^2 . We denote it by BAWGN(σ). The input X_t is an element of {+1, -1} and the output Y_t is real-valued. More precisely, $Y_t = X_t + Z_t$, where Z_t is a Gaussian random variable with zero mean and variance σ^2 . The random variables { Z_t } are independent. We can characterize the channel by E_n / σ^2 , the ratio of the energy per transmitted bit E_N to the noise energy σ^2 called as signal-to-noise ratio. For our setting we have $E_N = 1$ since $X_t \in \{-1, +1\}$. Alternatively, the measure E_b / N_o is often quoted. Here, E_b is the energy per transmitted information bit, $E_b = E_N / r$, where r is the rate, and $N_o = 2\sigma^2$ is the so called double-sided power spectral density. We therefore have $E_b / n_o = E_N / \left(2r\sigma^2\right)$. The signal-to-noise ratio is sometimes quoted as dB, this means as $10log10\left(E_N / \sigma^2\right)$ [19].

2.3 An Introduction to Block Codes

The basic purpose of a communication system is to reproduce a message from one time instant to another. Block coding of information is organized so that the message to be retransmitted, basically presented in binary format, is grouped into blocks of k bits, which are called the message bits, constituting a set of 2^k possible messages. The encoder takes each block of k bits, and converts it into longer block of $n > k$ bits, called the coded bits or the bits of the codeword. In this procedure there are $(n-k)$ bits that the encoder adds to the message word, which are usually called redundant bits or parity-check bits as shown in Figure 2.3.

The final step is the decoding process which involves the application of the decoding procedure, and then the discarding of the redundancy bits, since they do not contain any message information. These types of codes are called block codes and are denoted by $C_b(n,k)$. The rate of the code, $R_c = k/n$, is a measure of the level of redundancy applied in a given code, being the ratio of the given coded bits hat represent information or message bits.

Two important block codes are single parity-check codes and Reed-Muller codes. Both of these codes are the building blocks of more powerful codes such as LDPC, polar codes. We briefly explain single parity-check and repetition codes in the following section.

2.3.1 Single Parity-Check Codes

Single parity-check codes are $(N, N-1, 1-1/N)$ codes that append a single extra bit to the message. The extra bit is called the parity bit, as its value is the binary XOR of all the message bits. Suppose we want to transmit a binary message

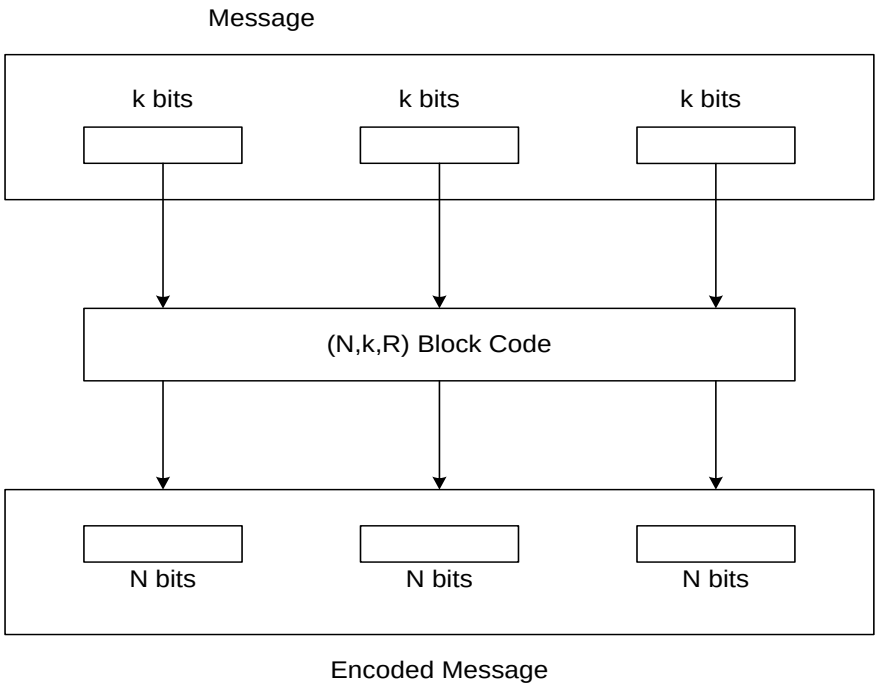


Figure 2.3: In a block code, the message is divided into block of k bits. Each block is then independently encoded to a block of N coded bits using an encoder [20].

$\mathbf{m} = [\,1\ 0\,]$ using a $(3,\ 2,\ 2/3)$ single parity-check code. We first compute the parity-check bit $1 = 1 \oplus 0$ and append it to the message forming the codeword $\mathbf{c} = [\,1\ 0\ 1\,]$, where the bit in the box is the parity bit.

Now, suppose we transmit codeword $\mathbf{c} = [\,1\ 0\ 1\,]$ on a binary-input discrete memory-less channel with conditional probability distribution $p(y \mid c)$. Let the channel observation be vector $\mathbf{y} = [\,y_0\ y_1\ y_2\,]$. The task of the decoder is to take this observation y and decide what is transmitted. The decoder's decision depends on which parameter it optimizes, and one such parameter is the probability of error $Pr(\hat{c}_i \neq c_i)$, where c_i is the decoder's decision for the $i-th$ coded bit. A common optimization metric is error probability, which can be lower bounded as follows [15]:

$$\begin{aligned} Pr(\hat{c}_i \neq c_i) &= 1 - Pr(c_i = \hat{c}_i) \\ &= 1 - \sum_{y} p(c_i = \hat{c}_i, \mathbf{y}) \\ &= 1 - \sum_{y} p(c_i = \hat{c}_i \mid \mathbf{y}) p(\mathbf{y}) \\ &= 1 - \sum_{y} p(\mathbf{y}) \max_{\hat{c}_i \in \{0,1\}} p(c_i = \hat{c}_i, \mathbf{y}) \end{aligned} \tag{2.1}$$

Equation (2.1) shows that the decoder that minimizes the error probability is the one that maximizes the probability $p(c_i \mid \mathbf{y})$ for all c_i. We call this probability the *a posterior probability* (APP), the decoder the *maximum a posterior* (MAP)

decoder, and the decision the MAP decision. The MAP decision $\hat{c}_i$ is given by [15]:

$$\begin{aligned}\hat{c}_i(\mathbf{y}) &= \arg\max_{c_i} p(c_i|\mathbf{y}) \\ &= \arg\max_{c_i} \frac{p(\mathbf{y}|c_i)\,p(c_i)}{p(\mathbf{y})} \qquad \text{(Bayes' Rule)} \quad (2.2) \\ &= \arg\max_{c_i} p(\mathbf{y}|c_i)\,p(c_i)\end{aligned}$$

where in the last step, we omitted $p(\mathbf{y})$, as it does not depend on c_i and does not take part in the optimization process. The last step of the decoder is to use these probabilities to make a decision about $i-th$ bit, according to [15]:

$$p(c_i = 0\,|\,\mathbf{y}) \underset{1}{\overset{0}{\lessgtr}} p(c_i = 1\,|\,\mathbf{y}).$$

or

$$p(\mathbf{y}|c_i = 0)\Pr(c_i = 0) \underset{1}{\overset{0}{\lessgtr}} p(\mathbf{y}1)\Pr(c_i = 1)$$

In simple words, it means that decide the bit is zero if

$$p(\mathbf{y}|c_i = 0)\Pr(c_i = 0) > p(\mathbf{y}1)\Pr(c_i = 1)$$

and one otherwise. We can further simplify it by converting it into a ratio and taking natural logarithm of both sides as follows [15]:

$$\frac{p(\mathbf{y}|c_i = 0)\Pr(c_i = 0)}{p(\mathbf{y}1)\Pr(c_i = 1)} \underset{1}{\overset{0}{\lessgtr}} 1$$

$$\log\left(\frac{p(\mathbf{y}|c_i = 0)\Pr(c_i = 0)}{p(\mathbf{y}1)\Pr(c_i = 1)}\right) \underset{1}{\overset{0}{\lessgtr}} 0$$

We can use trigonometric identities to reduce it to

$$\log\left(\frac{p(c_2=0\,|\,\mathbf{y})}{p(c_2=1\,|\,\mathbf{y})}\right)=\underbrace{2\tanh^{-1}\left(\tanh\left(\frac{L_0^{(i)}}{2}\right)\tanh\left(\frac{L_1^{(i)}}{2}\right)\right)}_{Extrinsic\ LLR}+\underbrace{L_2^{(i)}}_{Intrinsic\ LLR}$$
$$=L_2^{(e)}+L_2^{(i)}$$

where

$$L_j^{(i)}=\log\left(\frac{p(y_j\,|c_j=0)}{p(y_j\,|c_j=0)}\right)$$

Extrinsic LLRs provide the reliability information generated from code constraints, whereas intrinsic LLRs provide the reliability information directly gleaned from channel observations. To make a decision about bit c_2 , the decoder adds these two LLRs to compute the so-called full LLR and declares the bit zero if the sum is positive and one otherwise.

In general, for a $(N, N-1, 1-1/N)$ code the extrinsic LLR and full LLR of *kth* bit are given by [15]:

$$L_k^{(e)}=2\tanh^{-1}\left(\prod_{j\neq k}\tanh\left(\frac{L_j^{(i)}}{2}\right)\right)$$

and

$$L_k^{(full)}=L_k^{(e)}+L_k^{(i)}$$

respectively.

Figure 2.4 shows that the whole decision process as message passing along the edges on a graph. The rectangle represents a parity-check node, whereas the circles represent variable nodes. The parity-check node represents the constraint that all its inputs should sum to zero. The figure shows that to compute the extrinsic LLR for

the parity bit, we send intrinsic LLRs observed from the channel on the edges corresponding to variable nodes c_0 to c_{N-2} . The parity-check node takes all these messages from different variable nodes and computes the outbound message $L_{N-1}^{(e)}$. The same process can be repeated for all other bits to compute their extrinsic LLRs and eventually their full LLRs for detection.

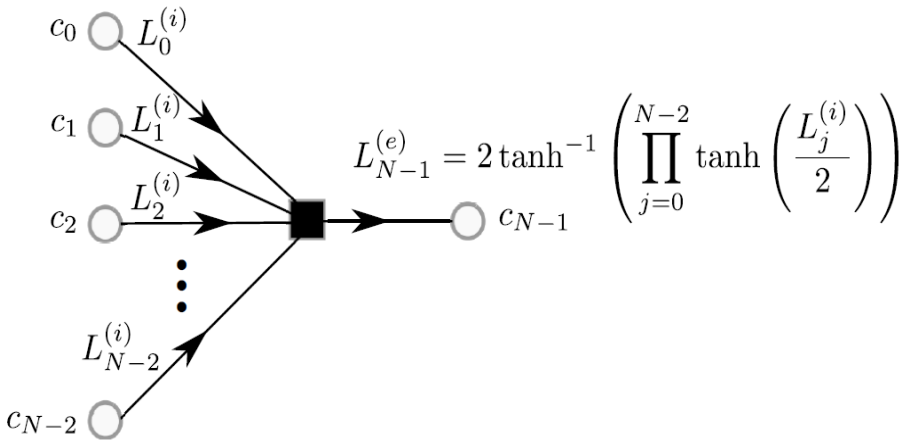


Figure 2.4: The parity-check node takes all the input messages in the form of intrinsic LLRs and produces the outbound message [15].

2.3.2 Repetition Codes

An $(N,1,1/N)$ repetition code repeats every message bit N times so that each message bit m_0 gets mapped to the codeword $c=[m_0,m_1,\ldots,m_{N-1}]$.

Suppose we want to transmit a zero bit using a $(N,1,1/N)$ repetition code. This code takes the message bit and transmits it N times producing the following equality constraint between all these transmitted bits $c_0=c_1=\ldots,=c_{N-1}$.

Following a similar analysis as in Section 2.3.1, the MAP decision for *kth* coded bit is given by [15]

$$\log\left(\frac{p(c_j=0|\mathbf{y})}{p(c_j=1|\mathbf{y})}\right) = \underbrace{\sum_{j\neq k} L_j^{(i)}}_{\textit{Extrinsic LLR}} + \underbrace{L_k^{(i)}}_{\textit{Intrinsic LLR}} \qquad (2.3)$$
$$= L_k^{(e)} + L_k^{(i)}$$

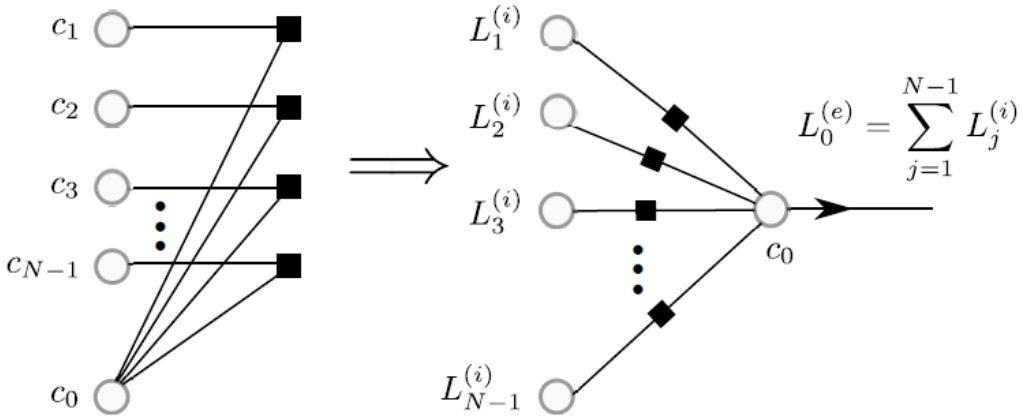


Figure 2.5: A $(N,1,1/N)$ repetition code repeats the message bit N times giving rise to $N-1$ trivial parity check nodes with two edges as shown on the left. The right hand side image is the rearranged version of the left hand side image [15].

Figure 2.5 shows that the decoding process for a repetition code can also be viewed as message passing on a graph like parity-check codes. The circle represents the variable node that puts the constraint that the bits corresponding to all the edges connected to it should be equal to each other. The left-hand image shows the trivial parity-check constraints of two edges. This two-edge parity-check is equivalent to an equality-check constraint meaning that the bits corresponding to

the edges connected to it should be equal. The right-hand image is a rearranged form of the left-hand image and resembles the parity-check code graph.

In Figure 2.5, we detect bit c_0 as all transmitted bits are equal, and detecting this bit is enough to decode the message. To detect c_0, we send all the intrinsic LLRs for $c_1, c_2, ..., c_{N-1}$ to the variable node of c_0. The variable node sums all the incoming messages and produces the extrinsic LLR $L_0^{(e)}$ that we add to intrinsic LLR $L_0^{(i)}$ calculating the full LLR. The sign of this full LLR decides about the message bit. It is easy to see that the full LLR for all the bits is equal to $\sum_{j=0}^{N-1} L_j^{(i)}$, highlighting once again that no matter which bit we decode, the resulting decision for the message bit stays the same. However, note that extrinsic LLRs corresponding to different bits will generally be different.

2.4 An Introduction to Polar Codes

In the previous section, we discussed block codes and their two important examples i.e., single parity-check codes and repetition codes. This section builds on this discussion and introduces polar codes that are complex combination of the basic parity-check and repetition codes.

As we have seen in the previous section that the block codes take a block of k bits, encode it to a block of N bits using a generator matrix $\mathbf{G}$. The difference is only in the construction of generator matrix. Same is the case with the polar codes, in polar codes the generator matrix is generated according the Reed-Muller codes rule.

Consider a (N,k,R) polar code of length N , dimension k and rate $R = k/N$, explained by Arikan in [1]. The source binary vector $u_1^N = (u_1, u_2, ..., u_N)$ can be used to generate the codeword $x_1^N = (x_1, x_2, ..., x_N)$ such that [1]

$$x_1^N = u_1^N \mathbf{B}_N \mathbf{G}_N \qquad (2.4)$$

where $\mathbf{B}_N$ is the $N \times N$ bit-reversal permutation matrix [1] ensuring that

$$\mathbf{x} = \mathbf{yB} \Rightarrow x_{b_{n-1}, b_{n-2}, ..., b_0} = y_{b_0, b1, ..., b_{n-1}},$$

where $b_1, ..., b_n \in \{0,1\}$ represent the binary expansion of the index of an element in row vectors $\mathbf{x}$ and $\mathbf{y}$. For example, $x_1 = x_{0,0,1}$ for $n = 3$. And $\mathbf{G}_N$ can be constructed using $\mathbf{G}_N = \mathbf{G}_2^{\otimes n}$, where $\otimes$ denotes the Kronecker product and $n = \log_2 N$, and [1]

$$\mathbf{G}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

By removing $N-K$ rows with indices from the set $I^c \subset \{0, 1, \ldots, N-1\}$ from G_N , we can obtain a $k \times N$ matrix $\mathbf{G}$. The codeword $\mathbf{x}$ is given by

$$\mathbf{x} = m\mathbf{G}$$

An alternative to this is to take the message vector $m = [m_0\ m_1, ..., m_{(k-1)}]$ of length k and form another vector $u = [u_0, u_1, \ldots, u_{(N-1)}]$ such that m appears in u on the index set $I \subset \{0, 1, \ldots, N-1\}$ and zero bits appear on I^c . In this case,

$\mathbf{x} = m\mathbf{G}$. In literature, the set I is usually referred to as the set of '*free indices*' and the complement I^c as the set of '*frozen indices'*. The construction of polar codes is equivalent to constructing I .

Example 2.1. Suppose we want to generate a $(8,4,0.5)$ polar code with frozen indices $I^c = \{0,1,\ 2,4\}$. Taking 3rd Kronecker product (because $log_2^N = 3$) of F_2 , we get [1]

$$F_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \textit{ with}$$

$$\mathrm{B}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.5)$$

Therefore, G_8 matrix is given by

$$G_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Removing four rows corresponding to the set of frozen indices gives us a 4 × 8 generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The encoder for example 2.1 is shown in Figure 2.6. Note that the encoder corresponds to G_8 and not F_8. Since, B_8 is only a permutation matrix, the same encoder diagram can be used to represent F_8 by permuting the positions of $\mathbf{u}$.

Reed-Muller codes are similar to polar codes in the sense that we can construct both the codes using the same F_N or G_N matrix. The difference is in the choice of rows to be removed. We can construct a (r, m) Reed-Muller code by keeping all the rows in F_N with Hamming weight greater than or equal to 2^{m-r} [21]. In other words, in Reed-Muller codes the rows with low Hamming weights are removed. In polar codes, a channel-specific rule determines the rows to be removed.

The polar code in example 2.1 with generator matrix **G** is also a (1, 3) Reed-Muller code [21]. Note that **G** is obtained by removing all the rows with Hamming weight less than $2^{3-1}=4$ in G_8. Example 2.1 is a special case in which the choice of the rows to be removed happens to be the same for Reed-Muller and polar codes, and in general, both codes will be different.

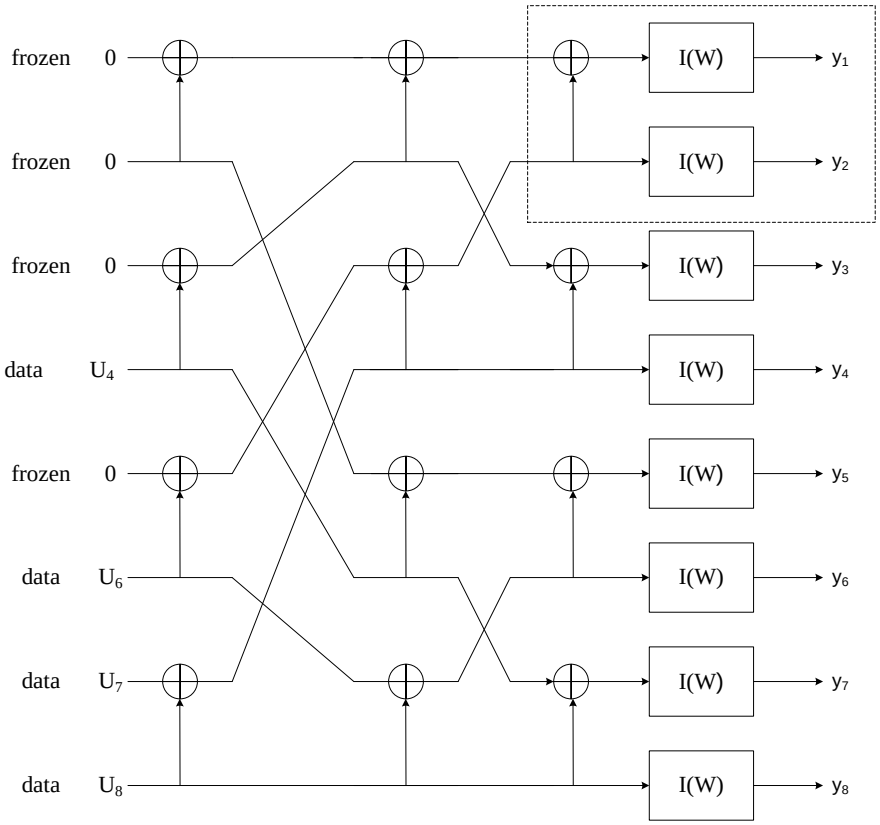


Figure 2.6: Polar code example $W = BEC(1/2)\ N = 8\ and\ rate = 0.5$. The dotted box shows the basic building block of polar codes [1].

2.4.1 Factor Graph of Polar Codes

A factor graph is a graphical way of expressing channel codes. This graphical representation is especially beneficial when the relationship to be represented can be factored into relationships between subsets of these variables. The factor graph for polar codes can be understood with the help of parity-check and repetition codes.

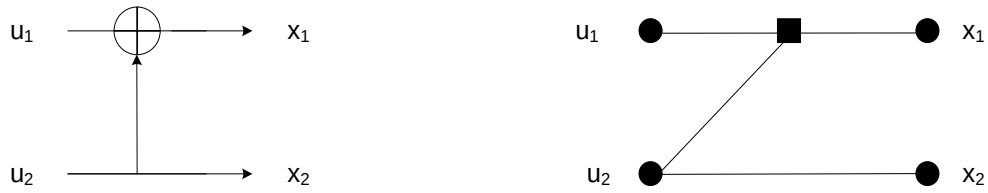


Figure 2.7: The basic structure in the encoder of polar code represents codes of length two. This basic encoder corresponds to a parity-check and an equality-check constraint [1].

Consider the case $N = 2$. Given (X_1, Y_1) and (X_2, Y_2), we define $U_1, U_2 \in \{0,1\}$ through the mapping (see Figure 2.7)

$$U_1 = X_1 + X_2 \quad and \quad U_2 = X_2 \tag{2.6}$$

where '+' denotes modulo-2 addition. Notice that the correspondence between U_1, U_2 and X_1, X_2 is one-to-one. This is in fact the simplest instance of polar source coding, with code length 2 and rate ½. Two bits u_0 and u_1 add to produce the first codeword bit x_0, whereas the second codeword bit x_1 is simply u_1. As discussed in section 2.4, we can decode u_0 using the parity-check node, whereas a

variable node can represent the equality constraint of x_1 and u_1 as discussed in section 2.4. Therefore, we can convert the constraints between message bits and codeword bits using a parity-check and variable node, as shown in figure 2.7.

We can repeat the process of converting the basic encoding structure of length two to two parity-check codes on the entire encoder to build constraints between message bits and codeword bits of polar code of arbitrary length. This collection of constraints gives rise to a factor graph, as shown in figure 2.6.

In this figure, the left-most circles represent the encoded information bits $u_1^N = (u_1, u_2, ..., u_N)$, the right most circles represent the codeword bits $x_1^N = (x_1, x_2, ..., x_N)$, while CN_{ij} and VN_{ij} represent check nodes and variable nodes, respectively. In between, the process of polar encoding is applied for $n = \log_2 N$ stages.

2.4.2 Construction of Polar Codes

The basic principle of the construction of polar codes is to find a set I such that the block error probability of the polar code is minimum when decoded using the SC decoder [1]. Arikan provided an upper bound on the block error probability of polar codes under the SC decoder [1] as follows:

$$P_e \leq \sum_{i \in I} P_e(i) \qquad (2.7)$$

where $P_e(i)$ is the error probability in detecting u_i given the perfect knowledge of all the previous bits u_1 to u_N and no information about u_{i+1} to u_{N-1} – the bits yet to

be detected. Therefore, the task of constructing I boils down to efficiently estimating $P_e(i)$ and finding a set I that minimizes the right-hand side of (2.12). The estimation of $P_e(i)$ is specific to the underlying channel, and therefore, we discuss the construction method for two famous channels; namely, the binary erasure channel (BEC) and the added white Gaussian noise (AWGN) channel.

2.4.2.1 Construction for the BEC

Suppose we want to construct a polar code for a BEC with erasure probability $\in$. Arikan showed in [1] that we can calculate $P_e(i) = W_N^{(i)}$ using the following recursions:

$$\begin{aligned} I(W_N^{(2i-1)}) &= I(W_{N/2}^{(i)})^2, \\ I(W_N^{(2i)}) &= 2I(W_{N/2}^{(i)})^2 - I(W_{N/2}^{(i)})^2 \end{aligned} \tag{2.8}$$

where $\in = 0.5$. The last step of the code construction is to choose a set I such that $\sum_{i \in I} P_e(i)$ is minimum.

2.4.2.2 Construction for the AWGN Channel

In this thesis, we choose the denisty evolution-based construction [22] for the construction of AWGN channel, because of its accuracy and implementation ease. In this method we asssume that the transmitter sends all-zeros codeword. In the case of all-zeros codeword transmission, it is easy to show that the log-likelihoods (LLRs) $2r_i / \sigma^2$ are distributed according to a Gaussian distribution with mean

$2/\sigma^2$ and variance $4/\sigma^2$, i.e., $2r_i/\sigma^2 \sim N(2/\sigma^2,\ 4/\sigma^2)$. Since the decoder uses the same equations to compute all the LLRs, all these LLRs follow the same distribution $N(m(\phi), 2m(\phi))$, where $m(\phi)$ is the mean of this distribution. In this notation, the LLRs corresponding to message bits u_i have mean $m_n(i)$, and the probability of error $P_e(i)$ (the probability that the LLR is negative) is given by [15]:

$$P_e(i) = Q\left(\sqrt{\frac{m_n(i)}{2}}\right) \qquad (2.9)$$

Trifonov and Semenov showed in [23] that $m_n(i)$ in (2.9) can be approximated using the initial value of $m_0(0) = 2/\sigma^2$ in the following recursions:

$$m_k(2j) = g(m_{k-1}(j)),$$
$$m_k(2j+1) = 2m_{k-1}(j),$$

Where

$$g(x) = h^{-1}(1-(1-h(x))^2),$$
$$h(x) = \begin{cases} e^{-0.4527x^{0.86}+0.0128} & x > 10 \\ \sqrt{\frac{\pi}{x}}e^{\frac{-x}{4}}[1-\frac{10}{7x}] & otherwise. \end{cases}$$

Once we have $P_e(i)$ for all $i \in \{0,\ 1,\ \ldots,\ N-1\}$, we can choose a set I that minimizes $\sum_{i\in I} P_e(i)$ in (2.7).

2.4.3 Successive-Cancellation Decoding

A Successive cancellation (SC) is the canonical algorithm for decoding polar codes and is the one used when proving their capacity-achieving performance in [1].

The SC algorithm estimates bits sequentially using either the predetermined values for frozen bits, or using the received channel information y and the previously estimated bits $\hat{u}_1^N$ according to the following rule for an information bit u_i [1]

$$\hat{u}_i = \begin{cases} u_i & if\ \ i \in I^C \\ h_i(y_1^N, \hat{u}_1^{i-1}) & if\ \ i \in I \end{cases} \qquad (2.10)$$

where

$$h_i(y_1^N, \hat{u}_1^{i-1}) = \begin{cases} 0 & if\ \dfrac{W_N^i(y_1^N, \hat{u}_1^{i-1} \mid 0)}{W_N^i(y_1^N, \hat{u}_1^{i-1} \mid 1)} \geq 1 \\ 1 & otherwise \end{cases}$$

A particular instance for $N = 8$ is shown in Figure 2.6, with BEC channel showing Bhattacharyya parameters for $\in = 0.5$ and 0.5 rate. We may visualize the decoder as consisting of N decision elements (DEs), one for each source element u_i ; the DEs are activated in the order $1\ to\ N$. If $i \in I^c$, the element u_i is known; so, the ith DE, when its turn comes, simply sets $\hat{u}_i = u_i$ and sends this result to all succeeding DEs. If $i \in I$, the ith DE waits until it has received the previous decisions $\hat{u}_1^N$, and upon receiving them, computes the likelihood ratio (LR) as given in equation 2.10, which is then sent to all succeeding DEs. This is a single-pass algorithm, with no revision of estimates. The complexity of this algorithm is $O(N \log N)$ [1].

2.5 More Powerful Decoding Algorithms

Although polar codes achieve the capacity asymptotically, their performance with SC decoding is unsatisfactory when the code has a finite length. Several alternative decoding schemes have been proposed to improve the finite-length performance of polar codes, such as successive cancellation list/stack (SCL/SCS) and BP decoding algorithms.

2.5.1 Improved SC Decoding

We can use a unified framework, called compact-stage code tree [24], to describe SC decoding and its improved algorithms, such as SCL/SCS decoding. Figure 2.8 gives an example of SC decoding over a compact-stage code tree, which consists of eight levels, with each level related to a piece of trellis which is circled by a color-dashed box.

In this code tree, except for the leaf nodes and the frozen nodes, each node has two descendants and the corresponding branches are labeled with 0 and 1, respectively. A decoding path consists of a branch sequence from the root to one of the nodes and the corresponding reliability can be measured using a posteriori probability (APP). In figure 2.8 the number written next to each node provides the APP metric of the decoding path from the root to that node. The black circles represent the nodes that are visited (the APP metrics of which are calculated), and the gray ones are those that are not visited in the search process.

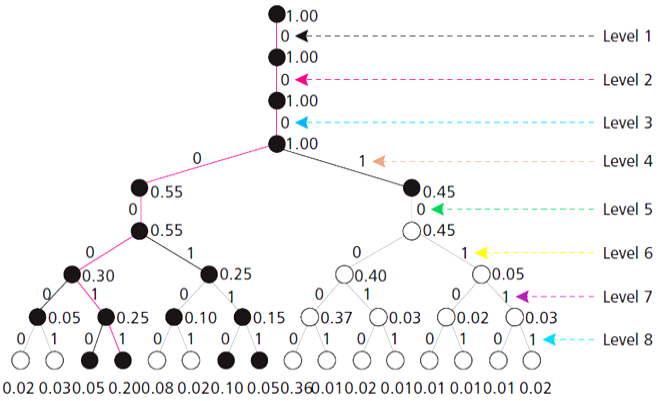
2.5.1.1 SCL and SCS

The SC decoding of polar codes can be regarded as a greedy search algorithm over the compact-stage code tree. Between the two branches associated with an information bit at a certain level, only the one with the larger probability is selected for further processing. Whenever a bit is wrongly determined, correcting it in the future decoding procedure becomes impossible. The red bold branches in Figure 2.8(a) illustrate an SC decoding path "00000011." Obviously this decoding path is not optimal due to the level by level decision strategy.

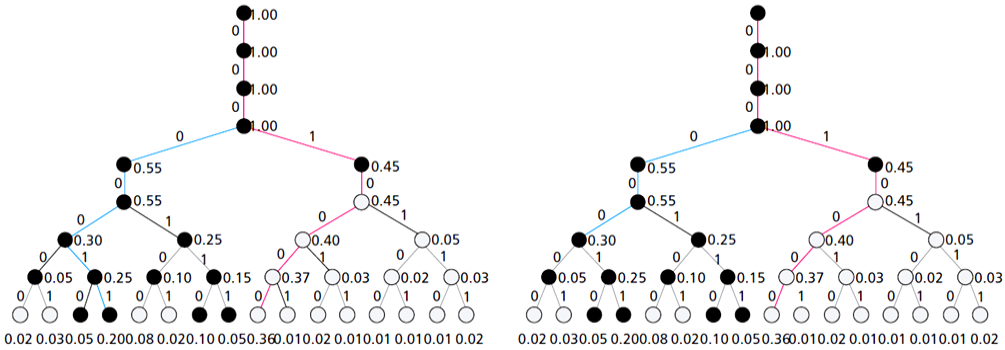
As an enhanced version of SC, the SCL decoder proposed in [11] searches the code tree level by level, in much the same manner as does the SC. However, unlike SC where only one path is reserved after processing at each level, the SCL can be regarded as a breadth-first search algorithm and allows a maximum of L candidate paths to be further explored. At each level related to an information bit, SCL doubles the number of candidates by appending a bit 0 or 1 to each of the candidate paths. It then selects a maximum of L ones with the largest metrics and stores them in a list. Figure 2.8(b) shows an example of SCL with list size $L = 2$. In each level, two candidate paths (illustrated by blue and red bold edges) are reserved, and the most reliable path "000100000" is found.

The SCS decoder proposed in [9] uses an ordered stack to store the candidate paths and tries to find the optimal estimation by searching along the best candidate in the stack. Whenever the top path in the stack that has the largest path metric reaches a leaf node, the decoding process stops and outputs this path. Unlike the candidate paths in the list of SCL, which always have the same length, the candidates in the

stack of SCS have different length. Figure 2.8(c) gives a simple example of SCS decoding. Compared with the SCL decoding, SCS can also find the same optimal path but the two candidate paths have different lengths so the number of path extensions can be reduced.



(a)



(b) (c)

Figure 2.8: SC, SCL and SCS over compact-stage code tree. (a) SC decoding over compact-stage code tree; (b) SCL decoding process L=2; (c) SCS decoding process [21].

As for the implementation aspect, a spaceefficient structure is suggested in [11] to implement the SC decoder, and the time and space complexities are $O(N\log N)$ and $O(N)$, respectively. A direct implementation of the SCL decoder will require $O(LN2)$ computations. In [11] a so-called “lazy copy” technique based on the memory sharing structure among the candidate paths is introduced to reduce the redundant copy operations. Therefore the SCL decoder can be implemented with time complexity $O(LN\log N)$. Similar to that for the SC and SCL decoders, these features can also be applied in the implementation of the SCS decoders. The actual computations of SCS, supposing the depth of stack D is large enough, become much fewer than that of SCL when working in the moderate or high signal-to-noise ratio (SNR) regime.

2.5.1.2 Hybrid Decoding

Combining the principles of SCL and SCS, a new decoding algorithm called successive cancellation hybrid (SCH) is proposed in [10]. This proposed algorithm can provide a flexible configuration when the time and space complexities are limited. Under proper configurations, all three improved SC decoding algorithms, such as SCL, SCS, and SCH, can approach the performance of ML decoding but with acceptable complexity. With the help of the proposed pruning technique, the time and space complexities of these decoders can be significantly reduced and be made very close to those of the SC decoder in the high SNR regime.

2.5.1.3 CRC-Aided Decoding

To further improve the performance of polar codes, CRC (cyclic redundancy check)-aided SCL/SCS decoding schemes, such as CA- *SCL/CA-SCS*, have been proposed in [11]. In these schemes the SCL/SCS decoder outputs the candidate paths into a CRC detector, and the check results are utilized to detect the correct codeword. To lower the time complexity of SCL decoding brought by a large list size, an adaptive CRC-aided SCL decoder (aCA-SCL) is proposed in [25] by progressively enlarging the list size. Under these CRC-aided decoding schemes, the performance of polar codes is substantially improved and outperforms that under ML decoding.

2.5.2 Belief Propagation Decoding

Based on the factor graph representation which is equivalent to the trellis of polar code shown in Figure 2.6, the BP decoder of polar codes is first introduced by Arikan in [1]. Rather than exchanging the hard messages, soft messages are transferred between the check and variable nodes in the BP decoding. Thus the BP decoder can considerably outperform the SC decoder. However, the message-passing schedule in BP plays an important role for channels other than BECs due to the redundant loops on the factor graph, and the optimal schedule is hard to determine. Besides, it is difficult to improve the parallelism of the BP decoder. Therefore, a practical BP decoder suffers a higher implementation complexity and offers a lower throughput than an SC decoder.

2.5.3 ML or MAP Decoding

Theoretically the optimal decoding algorithms of polar codes are ML or MAP decoders, which can be implemented via Viterbi and BCJR algorithms on the trellis, respectively. But these well known algorithms are too complex to be practical for a medium or long code length, so they are only regarded as a reference of performance comparison for other decoding algorithms of polar codes.

Table 1: Comparison of different decoding schemes for polar codes [24].

Decoding			
	Algorithms	**Complexity**	**Performance**
Polar Codes	SC	$O(NlogN)$ low	Suboptimal
	SCL	$O(LNlogN)$ mdeium	Approach ML
	BP	$O(I_{\max} N \log N)$ high	Suboptimal
	CA-SCL	$O(LNlogN)$ medium	Outperform ML

The table 2-1 gives the complexity and performance analysis of polar codes for different decoding algorithms. We see that the complexity of SC decoder is low but at the same time the performance is not good. While for SCL and CA-SCL the complexity is medium but the performance is remarkable i.e., it approcahes ML, and in the case of CA-SCL it outperforms the ML performance. However, BP decoding lies in midway with the highest complexity but with the suboptimal

performance. Hence, polar codes with CA-SCL/CA-SCS can achieve better tradeoff between performance and complexity than the others. In conclusion we find that polar codes have evident benefits on the theoretic performance, construction methods, and encoding/decoding schemes.

3. IMPROVED BELIEF PROPAGATION DECODING OF POLAR CODES

In this chapter, we propose a modified version of belief propagation algorithm based on damped BP decoder which outperforms the original BP decoder by 1-2dB while having the same complexity as the original BP decoder.

We begin by the introduction of standard BP decoder. We also discuss the update rules for variable and check nodes. Then we give a comprehensive BP decoder for the polar codes. Later we discuss the proposed BP algorithm and parity-check matrix-based BP decoding of polar codes. In the end, we give the simulation results we show that using the proposed technique we achieve 1-2dB gain in the performance of polar codes for finite lengths. We also give the complexity analysis and the number of iterations which shows that the complexity is same as the standard BP algorithm but the number if iterations required for decoding the correct codeword are greatly reduced.

3.1 Belief Propagation Algorithm

Linear block and LDPC codes can iteratively be decoded on the factor graph by Belief Propagation Algorithm (BP) (it is also known as Sum-Product (Message Passing or Max-Product (Min-Sum) Algorithm). As it is known that a factor graph represents a factorization of the global code constraint

$$\mathbf{H} \otimes x$$

into the local code constraints which are represented by the connection between variable and check nodes. These nodes perform local decoding operations and exchange the messages along the edges of the factor graph. It can be construed that the extrinsic message is a soft-value for a symbol when the direct observation of the symbol is not considered in the computation (local decoding operation) of this specific value.

The belief propagation algorithm is an iterative decoding technique. So, in the first iteration, the incoming messages received from the channel at the variable nodes are directly passed along the edges to the neighboring check nodes because there are no incoming messages (extrinsic) from the check nodes in the first iteration. The check nodes perform local decoding operations to compute outgoing messages (extrinsic) depending on the incoming messages received from the neighboring variable nodes. Thereafter, these new outgoing messages are sent back along the edges to the neighboring variable nodes. The meaning of one complete iteration can be comprehended that the one outgoing message (extrinsic) has passed in both directions along every edge. One iteration is illustrated in the Figure 3.1 for the (7; 4; 3) Hamming code by showing the direction of the message in each direction along every edge. The variable-to-check (μ_{vc}) and check-to-variable (μ_{cv}) are extrinsic messages which are also shown in the same figure 3.1 [23].

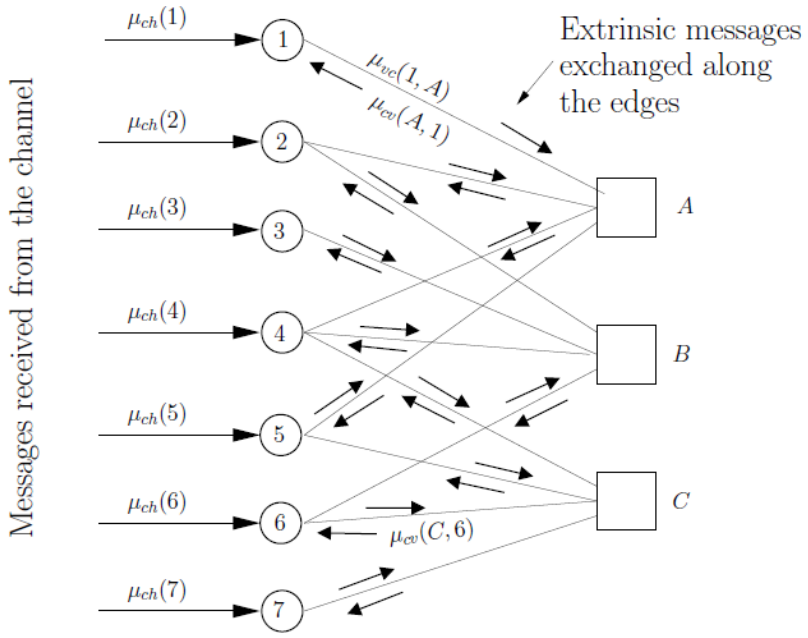


Figure 3.1: To illustrate one complete iteration in a factor graph for (7,4,3) hamming code. The messages μ_{vc} and μ_{cv} for instance are extrinsic. μ_{ch} are the messages coming from the channel [26].

After every one complete iteration, it will be checked whether a valid codeword is found or not. If the estimated code symbols form a valid codeword such that

$$\mathbf{H} \otimes \hat{x} = 0$$

(where, $\hat{x}$ is an estimated codeword.)

then the iteration will be terminated otherwise it will continue. After the first complete iteration, the variable nodes will perform the local decoding operations in the same way to compute the outgoing messages (extrinsic) from the incoming messages received from both the channel and the neighboring check nodes. In this

way, the iterations will continue to update the extrinsic messages unless the valid codeword is found or some stopping criterion is fulfilled.

3.1.1 Belief Propagation Algorithm and Node Operations

Considering any factor graph in general, the message passing algorithm is listed below in order to give an overview of this algorithm. The extrinsic messages which are computed by the local decoding operations at the variable nodes are denoted as μ_{vc} which means message from variable $\rightarrow$ check while at the check nodes are denoted as μ_{cv} which means message from check variable.

1. The initial message coming from the channel at variable node n is denoted as $\mu_{ch}(n)$.

2. The extrinsic message from the variable to check node is

$$\mu_{vc}(n,m) = fct_v(\mu_{ch}(n), \mu_{cv}(m',n)) \qquad (3.1)$$

Where

n = variable node

$m \in M(n)$: check nodes which are neighbor of the variable node n.

$m' \in M(n)/m$: check nodes except m which are neighbor of the varibale node n.

The new or updated extrinsic message $\mu_{vc}(n,m)$ which is computed by the local decoding operation or function fct_v , will be sent to the check node m. Therefore,

the incoming extrinsic message $\mu_{cv}(m,n)$ from the check node m is not considered for updating the message $\mu_{vc}(n,m)$.

3. The extrinsic message from the check to variable node is

$$\mu_{cv}(m,n) = fct_c(\mu_{vc}(n',m)) \qquad (3.2)$$

where, fct_c is the local decoding operation at a check node and $n' \in N(m)/n$: variable nodes except n which are the neighbor of the check node m.

4. The final message that is computed at the variable node n in order to estimate the code symbol.

$$\mu_v(n) = fct_v(\mu_{ch}(n), \mu_{cv}(m,n)) \qquad (3.3)$$

5. The estimation of a code symbol X_n can be done by hard decision

$$\hat{x}_n = \begin{cases} 0 & if\ \Pr(X_n = 0 | \mu_v(n)) \geq \Pr(X_n = 1 | \mu_v(n)) \\ 1 & else \end{cases} \qquad (3.4)$$

6. If these symbol-wise estimated code symbols are stacked to form vector $\hat{x}$ of length N, then it can be checked whether $\hat{x}$ the is a valid codeword by

$$\mathbf{H} \otimes \hat{x} = 0 \qquad (3.5)$$

7. If the above equation (4.5) is satisfied or the current number of iteration is equal to some defined maximum number of iterations then stop the iteration otherwise repeat the algorithm from step 2 to step 7.

3.1.2 Example of Node Operations

Considering the factor graph of (7,4,3) Hamming code which is shown in Figure 3.1, the above steps of the algorithm is shown.

At the variable node 2:

The initial message at the variable node 2 is $\mu_{ch}(2)$.

In general, the incoming (extrinsic) messages at the node 2:

$$\mu_{ch}(2),\ \mu_{cv}(A,2),\ \mu_{cv}(B,2)$$

In general, the outgoing (extrinsic) message from the node 2:

$$\mu_{vc}(2,B)\ =\ fct_{v}(\mu_{ch}(2),\mu_{cv}(A,2))$$

So, the message $\mu_{cv}(B,2)$ is excluded for the computation of $\mu_{vc}(2,B)$.

At the check node B:

The incoming (extrinsic) messages at the check node B:

$$\mu_{vc}(2,B),\ \mu_{vc}(3,B),\ \mu_{vc}(4,B),\ \mu_{vc}(6,B)$$

The outgoing (extrinsic) message from the check node B:

$$\mu_{cv}(B,2),\ \mu_{cv}(B,3),\ \mu_{cv}(B,4),\ \mu_{cv}(B,\ 6)$$

the local decoding operation at the check node B to compute the extrinsic (outgoing) message say $\mu_{cv}(B,2)$:

$$\mu_{cv}(B,2)\ =\ fct_{c}\mu_{vc}(3,B),\ \mu_{vc}(4,B),\ \mu_{vc}(6,B)$$

It can be noticed that the message $\mu_{vc}(2,B)$ is excluded for the computation of $\mu_{cv}(B,2)$.

It shall be noted that these messages μ can be in terms of either probabilities or log likelihood ratios LLR.

3.2 Belief Propagation Decoder for Polar Codes

The factor graph description of figure 2.6 enables the message passing based BP algorithm to be used for polar codes [12], [27]. One advantage of this decoder is that it provides soft outputs for the coded bits. The pitfall, however, is that it requires a large number of iterations over the factor graph of the polar code resulting in a larger processing and memory requirement. Additionally, a large number of iterations hinders high-throughput (low-latency) implementations. Therefore, this high processing, memory and latency requirement of the BP decoder makes it practically infeasible for many applications. We briefly describe the operation of the decoder here.

The decoder works locally on protographs of length two by computing extrinsic LLRs corresponding to all the four nodes of the protograph [15]. Figure 3.11 explains the LLR updates in a single protograph, which is the factor graph of a polar code of length two. The decoder first updates $L_1(0,0)$ using $L_0(0,\ 0)$ and $L_0(0,\ 1)$ from L and $B_1(1,\ 0)$ from B, and then updates $B_1(1,\ 0)$ using the same $L_0(0,\ 0)$ and $L_0(0,\ 1)$ from L but $B_1(0,\ 0)$ from B, completing the left-to-right update on the protograph. The decoder then starts to update the LLRs from right to left by updating $B_0(0,\ 0)$ and $B_0(0,\ 1)$. The four steps shown in Figure 3.11

complete the updates on this single protograph. The LLRs in red represent the LLRs that are used to update the LLRs in green.

The BP decoder updates messages on the top-right protograph of the factor graph and then moves down updating all the protograph at $\lambda = n$. Once the BP decoder has updated all the protographs at depth $\lambda = n$, it moves from right to left updating all the protographs at depth $0 \le \lambda < n$ in a similar fashion until it reaches the bottom-left corner of the factor graph, completing the first iteration of the BP decoder. The decoder repeats this process for a fixed number of iterations and at the end of last iteration, produces the message estimate.

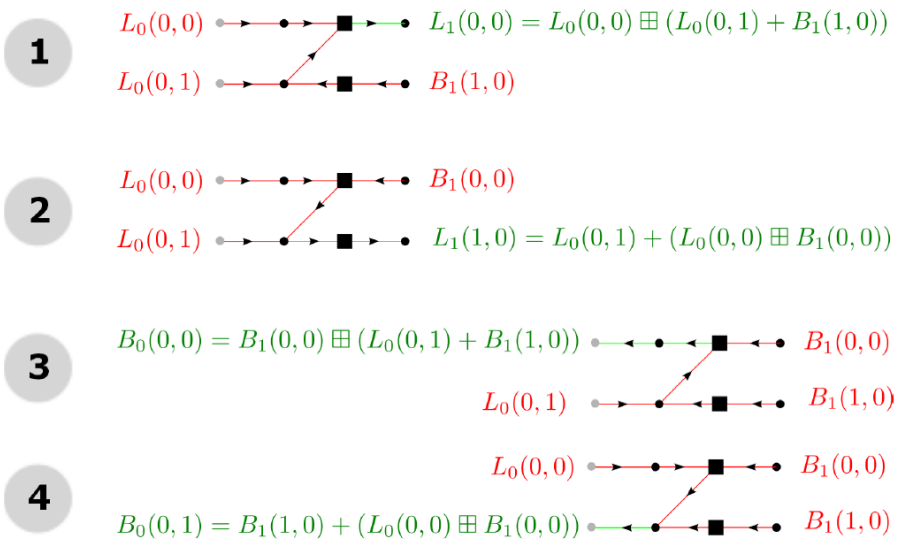


Figure 3.2: The BP decoder updates LLRs on a protogrpah-by-protograph basis. In a single protograph, it updates four LLrs with two LLRs in both L and B [15].

3.3 Proposed BP Decoding of Polar Codes

We will now breifly review the notations that we will use for the polar codes and explain the proposed method for polar codes. In the BP algorithm [28], a factor graph is considered, containing variable nodes VN_i and check nodes CN_j, as shown in Figure 3.12. The channel probabilities for the possible values of y_i are connected to variable nodes VN_i, and are fed directly to the BP decoder, as shown in Figure 3.12. Each variable node is connected to one or more check nodes according to the parity check equations, and, in turn, check nodes are also connected to one or more variable nodes according to the parity check equations. For each iteration of the BP algorithm, the variable nodes pass their beliefs to the check nodes. After processing those beliefs, check nodes send their own beliefs to the variable nodes. This process will continue until all of the parity check equations are satisfied, i.e., $\mathbf{H}.\hat{x}_1^N = 0$.

Let $\mathbf{H}$ be a $N \times M$ parity check matrix of polar codes, consisting of N variable nodes VN_i and M check nodes CN_j. For integer i and j with $1 \le i \le N$, $1 \le j \le M$, consider the extrinsic probabilities, $PC_{ij}(0)$ and $PC_{ij}(1)$, and a-priori probabilities, $PV_{ij}(0)$ and $PV_{ij}(1)$. Here $PC_{ij}(0)$ and $PC_{ij}(1)$ are the messages sent by the check node CN_j to the variable node VN_i, and $PV_{ij}(0)$, $PV_{ij}(1)$ are the messages sent by the variable node VN_i to the check node CN_j. The probabilities

are scaled so that $PV_{ij}(0)+PV_{ij}(1)=0$ and $PC_{ij}(0)+PC_{ij}(1)=1$. The general procedure for belief propagation can be explained as follows:

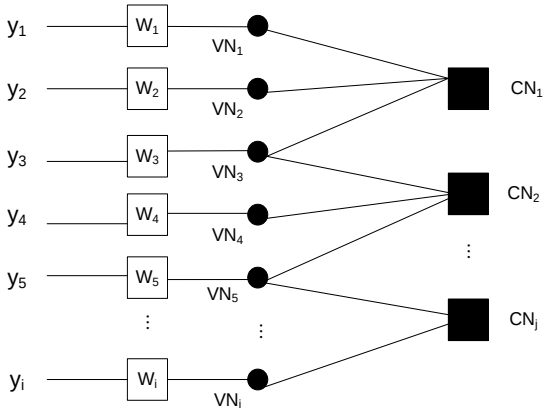


Figure 3.3: General view of a factor graph with i variable nodes and j check nodes using the coding model [29].

Step 1: Initialization

When the transmitted codeword, x_n, is received at the receiver, the decoder will first compute the channel probabilities as [30]

$$P_{ch,i}=1/(1+e^{2y_i/\sigma^2}) \qquad (3.6)$$

where y_i is the received vector corrupted by noise, and $\sigma^2=N_o/2$ is the variance of the AWGN channel. These channel probabilities are then assigned to the corresponding variable nodes.

Step 2: Sending messages from check nodes to variable nodes

Upon receiving messages from the variable nodes, the check nodes calculate their messages and send them back to the variable nodes. Figure 3.13 depicts the general flow of messages from any check node, CN_j, to variable node, VN_i. Each check node, CN_j connected with a variable node, VN_i calculates their messages as [30]

$$PC_{ij}(\alpha) = \beta . \sum_{j'} \prod_{j'} PV_{ij}(\alpha) \qquad (j' = 1, 2, ..., j \text{ and } j' \neq j) \qquad (3.7)$$

where $\alpha \in \{0,1\}$, and β is the scaling constant such that $PC_{ij}(0) + PC_{ij}(1) = 1$.

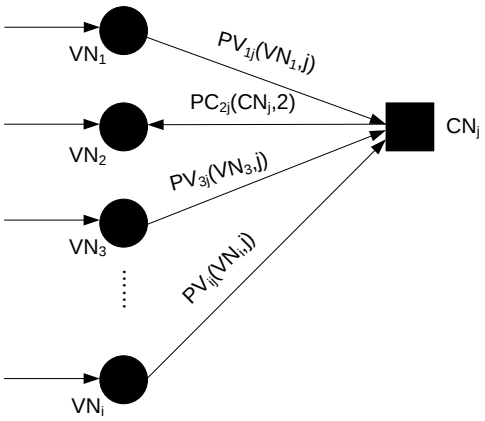


Figure 3.4: Flow of messages to and from a check node [29].

Step 3: Sending messages from variable nodes to check nodes

After calculating their messages, the check nodes send their messages to the variable nodes, as shown in Figure 3.14. The variable nodes then update their messages based on the messages from check nodes using [30]

$$PV_{ij}(\alpha) = \beta'.P_{ch,i}\prod_{i'} PV_{ij}(\alpha) \qquad (i' = 1,2,...,i \text{ and } i' \neq i) \qquad (3.8)$$

where $\alpha \in \{0,1\}$,and β' is the scaling constant such that $PV_{ij}(0) + PV_{ij}(1) = 1$.

Step 4: Decoding

After each iteration, the codeword will be estimated as [30]:

$$\hat{x}_n = \begin{cases} 0 & if\ PV_{ij}(0) > PV_{ij}(1) \\ 1 & otherwise \end{cases} 、 \qquad (3.9)$$

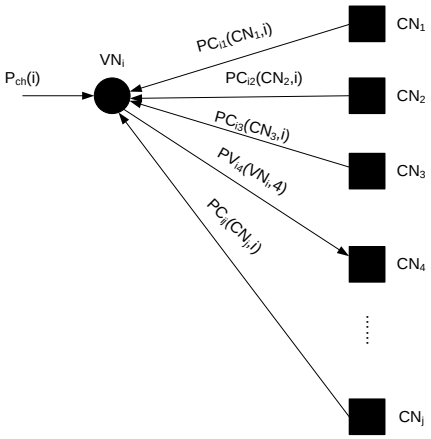


Figure 3.5: Flow of messages to and from a variable node [29].

After this is estimated, the decoder will check if $\mathbf{H}.\hat{x}_n = 0$. If this equation holds true, then the correct codeword has been decoded [30]. Otherwise, it will repeat all the steps until the correct codeword is decoded or until a maximum number of iteration is reached.

3.3.1 Parity-Check Matrix-Based BP Decoding of Polar Codes

The BP decoder for polar codes requires $2N\left(log_2\left(N\right)+1\right)$ real values to store, whereas full parallel implementation of the BP decoder for LDPC codes requires storage of $N\left(d_v+1\right)$ real values. Hence, to reduce the memory required for BP decoder we propose here to use the parity check based BP decoding of polar codes as compared to the factor graph BP decoding of polar codes we studied in section 3.3. According to [31], the parity-check matrix $\mathbf{H}$ for polar codes of length $N=2^2$ and rate $R=1/N$, can be given as

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

and the factor graph representation for this code is given in Figure 3.15a [31]. This factor graph can also be represented as the Tanner graph shown in Figure 3.15b [31]. But according to $\mathbf{H}$ **,** its equivalent factor graph can be represented as in Figure 3.15c. This leads us to our proposition.

Proposition :

If $\mathbf{H}$ is the parity-check matrix of polar codes, which can be represented as a Tanner graph (as in figure 3.15c), then according to [19, 29] we can run the BP decoding for polar codes on this Tanner graph.

Therefore we conclude that we can use the parity-check matrix $\mathbf{H}$ of polar codes for decoding under belief propagation algorithm.

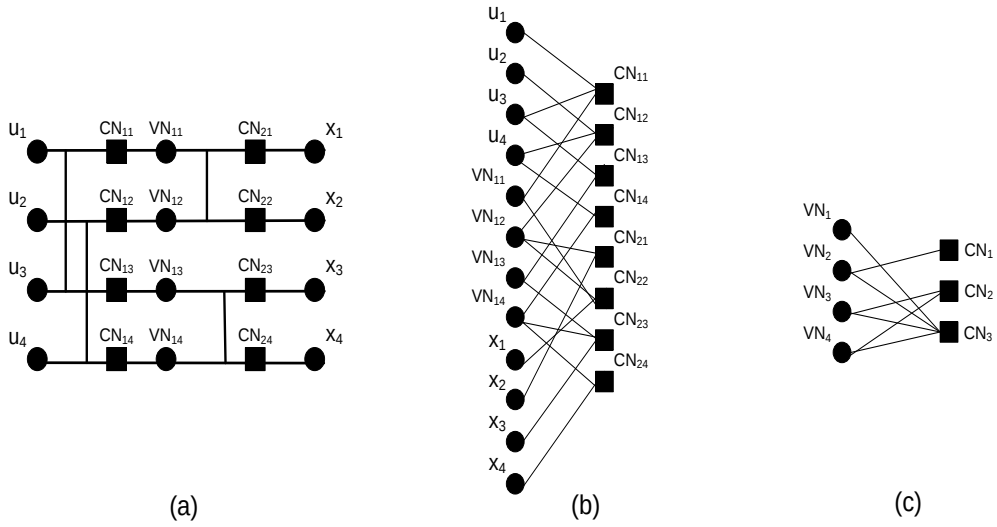


Figure 3.6: (a) Factor graph representation of polar codes for $N = 2^2$ [31]. (b) Equivalent representation of polar codes for $N = 2^2$ in the form of a Tanner graph based on (a) [31]. (c) Equivalent representation of polar codes for $N = 2^2$ and rate $R = 1/N$ in the form of a Tanner graph based on $\mathbf{H}$.

3.3.2 Proposed Method

It is evident from [2] and [3] that the performance of polar codes of finite length is poor. Motivated by this, we investigated the polar codes and the BP decoder. In [14], it is noted that the poor performance of polar codes is due to the stopping sets in the factor graph of the polar codes. And if all the variable nodes in the stopping sets are erased then none of them can be decoded in the belief propagation [14]. Also, if any of the variable nodes is incorrectly decoded then all of them will be incorrectly decoded. As a result, the probability of decoding successfully is closely related to the probability of decoding variable nodes correctly. Hence, we proposed

to modify the variable update equations in the belief propagation algorithm for better BER performance of polar codes.

Since LDPC uses belief propagation decoding, there have been various methods proposed to improve the BER performance of belief propagation in LDPC codes. One of the schemes is to use the damped belief propagation proposed in [4]. In [4], we see that the current messages are updated by adding the weighted average of previous and current messages, which results in the better BER performance for LDPC codes. Instead of adding the weighted average of previous and current messages to update current messages, we proposed to multiply the previous messages with the new messages for the update of variable nodes. This improves the BER performance of polar codes under belief propagation decoding and it is confirmed by the simulation results in the next section.

In the previous section, we have seen that the variable nodes are multiplied by the channel probabilities when they are updated (see equation 3.15). Due to the very small degrees of check nodes and variable nodes in the case of polar codes [14], the belief that is passed between the factor graphs is distorted over multiple iterations. Moreover, if only one bit is incorrectly decoded, it will pass the wrong belief to all of the other nodes, and the correct codeword cannot be decoded. We propose to multiply the variable nodes by their previous messages instead of by their channel probabilities, in order to increase the reliability of propagated messages. This is like a feedback path in which the previous signal is fed to the present signal, as shown in Figure 3.16, so that the present signal does not go out-of-bounds. The first iteration in the proposed BP decoder will remain same as that

of the original BP decoder, while the subsequent iterations will be different; i.e., in the next iteration, the variable-node update equations will be multiplied by their previous messages. The BP algorithm explained in Section 3 can now be described for polar codes as follows.

Step 1: Initialization

The first step remains the same: the transmitted codeword x_i is received at the receiver, and the decoder computes the channel probabilities as given in equation 3.6.

Step 2: Sending messages from check nodes to variable nodes

The second step also remains the same. Upon receiving the messages from variable nodes, the check nodes calculate their messages and send them back to the variable nodes. Each check node, CN_j connected with variable node, VN_i calculates their messages as given in equation 3.7.

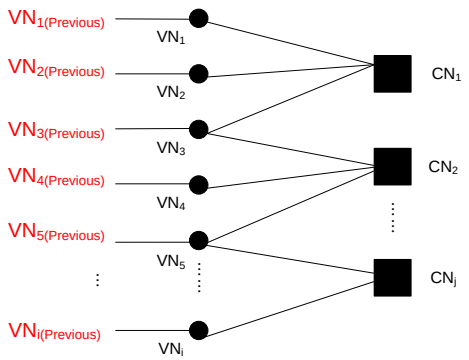


Figure 3.7: General view of the coding model according to the proposed method, where previous values of variable nodes are fed to the present values of variable nodes.

Step 3: Sending messages from variable nodes to check nodes

The variable nodes then update their messages based on the messages from check nodes using equation 3.8. For the first iteration, the formula for updating the variable nodes will remain same as given in above in equation 3.8. For the next iterations, the previous message, $PV_{ij(Previous)}$ of the variable node will be multiplied by the current message, PV_{ij} as given in equation 3.10. The new diagram of variable nodes is shown in figure 3.17, where the previous message of the variable node, $PV_{ij(Previous)}$ is multiplied by the variable node when calculating the current message.

$$PV_{ij}(\alpha) = \beta'.\ PV_{ij(previous)}(\alpha)\prod_{i'} PV_{ij}(\alpha) \qquad (i' = 1,2,...,i \text{ and } i' \neq i) \tag{3.10}$$

where $\alpha \in \{0,1\}$,and β' is the scaling constant such that $PV_{ij}(0) + PV_{ij}(1) = 1$.

Step 4: Decoding

After each iteration, the codeword will be estimated according to the equation 6. After this is estimated, the decoder will check if $\mathbf{H}.\hat{x}_n = 0$. If this equation holds true, then the correct codeword has been decoded. Otherwise, it will repeat all the steps until the correct codeword is decoded or until a maximum number of iteration is reached.

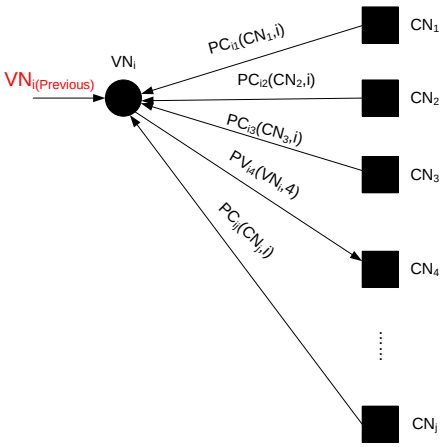


Figure 3.8: Diagram of a variable node in the proposed decoder.

For instance, consider that the variable node VN_3 connected with check nodes CN_1 and CN_2 , as shown in figure 3.17, calculated $\left(PV_{ij}(0)=0.53, PV_{ij}(1)=0.47\right)$ as its previous message and $\left(PV_{ij}(0)=0.297, PV_{ij}(1)=0.204\right)$ as its current message before multiplying with previous message. After multiplying and normalizing the current message with the previous message, the result is $\left(PV_{ij}(0)=0.621, PV_{ij}(1)=0.379\right)$. This shows that the reliability of the message is increased. This analytical result shows that the performance of the BP decoder will be improved with the new method, and this is proved with simulation results in the next section.

3.4 Simulation Results

In this section, we show that the complexity of the proposed decoder is not increased as compared to the original BP decoder. We will also show that the number of iterations taken by the proposed BP decoder is less than as compared to the original BP decoder. We further compare the error-rate performance of the polar codes designed using our proposed method with the polar codes designed for the AWGN channel.

3.4.1 Complexity Analysis

The complexity of each iteration for the proposed decoder is $O(N \log N)$ which is equal to the complexity of the original BP decoder. It can be proved by considering equations 3.8 and 3.10, where equation 3.10 gives the update equation for the variable node using the proposed method. It can be seen that equation 3.10 is multiplied by the previous value of the variable node, i.e., $PV_{ij(Previous)}$ while equation 3.18 used by the original BP decoder is multiplied by the channel probabilities, i.e., $P_{ch,i}$. Thus, we see that both the equations are multiplied by their corresponding probabilities, implying that the number of multiplications is not increased, thereby showing that the complexity of the algorithm is not increased.

In figure 3.18 we show the number of iterations taken by the standard and proposed BP decoders. From the figure it can be seen that the number of iterations taken by the proposed BP decoder is less as compared to the standard BP decoder by 10-25 iterations for different values of SNR.

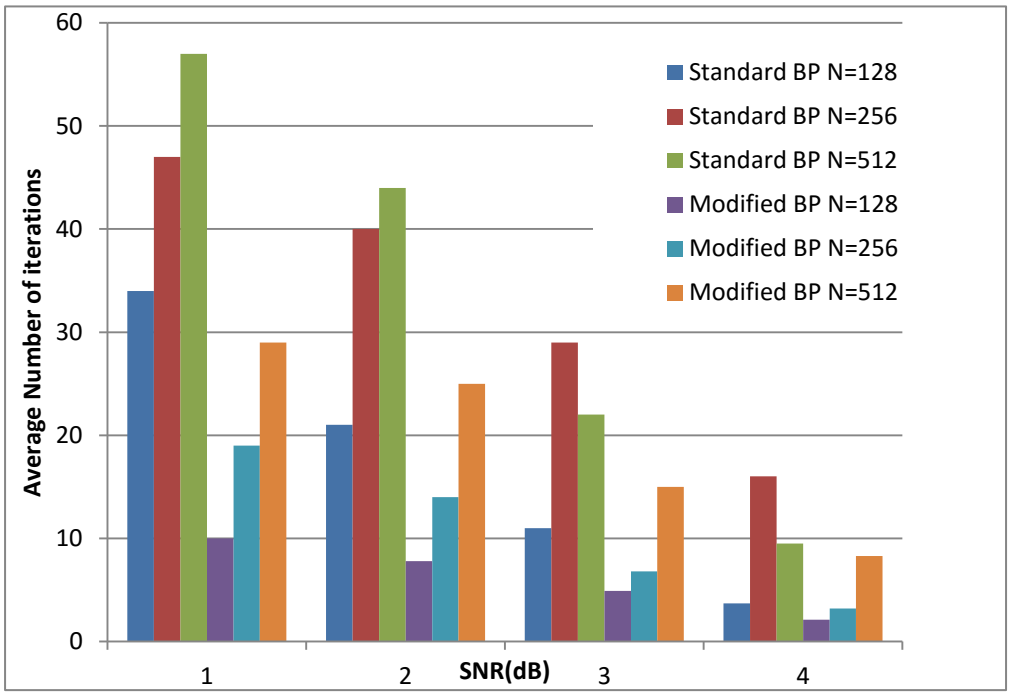


Figure 3.9: Comparison of average number of iterations taken by standard and proposed BP decoders for different values of N .

3.4.2 Performance

Figure 3.19 demonstrates the improved performance of our proposed algorithm in the AWGN channel as compared to the standard BP algorithm. We have simulated the proposed and standard BP decoder for a polar code of block lengths N= 128, 256, 512 and 1024 and code rate is 0.5. The polarized channels carrying the information bits are selected according to the density evolution method given in [26]. We set up Monte Carlo simulation with BPSK modulation. We have simulated a maximum of 10^6 codewords to calculate error rates on all SNR values,

terminating the simulation if 100 or more codewords are found in error. In all simulations, a maximum of 60 BP iterations are used.

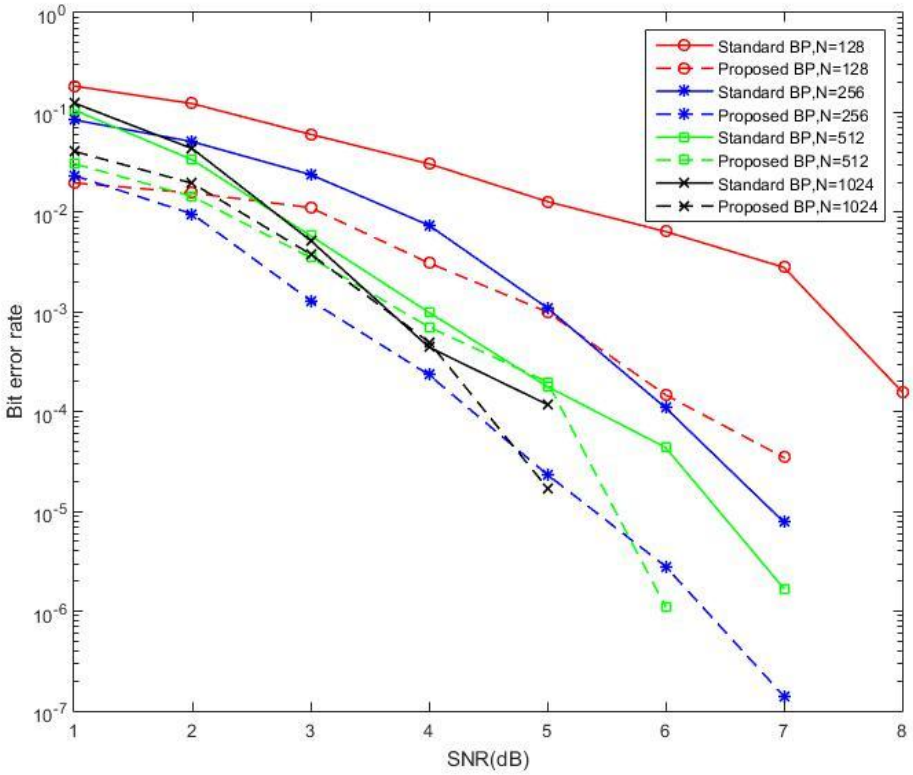


Figure 3.10: Comparison of BER between the standard and modified BP algorithms with a 0.5 code rate.

In the figure 3.19, bit error rates of standard BP and modified BP decoders with different code length, $N = 128, 256, 512, 1024$, are given. It can be seen from the figure 3.19 that we can achieve a 2dB of gain at BER of 10^{-2} for $N = 128$ and $N = 256$. At BER of 10^{-3} we see SNR advantage of approximately 2.5 dB for $N = 128$ and $N = 256$, which shows that the SNR advantage increases as the value

of SNR increases. The performance for $N = 512$ and $N = 1024$ at small values of SNR is same as the standard BP algorithm. However, at higher values of SNR, the performance of proposed algorithm is still better; e.g. at BER of 10^{-4} we observed SNR advantages of approximately 0.6 dB. Hence, it can be concluded that the performance of our proposed BP decoder is better than the original BP decoder.

3.5 Summary

In this chapter we studied the BER performance of short polar codes under belief propagation decoding. We analyzed the factor graph of polar codes and showed that the BP algorithm can be run on the Tanner graph constructed by the parity check matrix of polar codes. Furthermore, motivated by the poor performance of short polar codes, we proposed a novel technique for improving the performance of such codes under BP decoding. The variable nodes in the Tanner graph of polar codes are multiplied by their previous messages, which improve the reliability of propagated messages in the Tanner graph. Simulation results show that the proposed method achieves a gain of 1-2 dB at higher SNR values over the original BP decoder. It is also shown that the proposed method does not increase the complexity of decoding; the complexity is the same as that of the original BP decoder. Furthermore, we have shown that the number of iterations taken by the proposed BP decoder is less by 10-25 iterations as compared to the standard BP decoder.

4. HARDWARE ARCHITECTURES OF POLAR CODES

In this chapter we will discuss the available hardware architectures of polar codes. We begin by discussing the successive cancellation decoder and its different variants. Then we will discuss the Belief propagation decoder followed by the comparison of available hardware architectures. In the end we will discuss the hardware of proposed polar BP decoder.

4.1 Successive-Cancellation Decoder Implementation

A direct implementation of the SC decoding algorithm uses $N/2$ processing elements (PEs) [32]. However, it was observed in [28] that the $N/2$ PEs were only used simultaneously once in the decoding, leading to low utilization of the hardware resources. That work shows that, a decoder implementing $P = 64$, instead of $N/2$, processing elements result in <10% throughput reduction for codes of length $N \leq 2^{20}$.

The semi-parallel SC (SP-SC) decoder of [33] consists of three major parts: the processing elements, the partial-sum update logic, and the memory, which will be briefly described in this section.

4.1.1 Processing Elements

A PE is the core functional unit of the SP-SC decoder and can perform the following *f* and *g* functions:

$$\lambda_{u_0} = f(\lambda_{v_0}, \lambda_{v_1}) = \text{sign}(\lambda_{v_0})\,\text{sign}(\lambda_{v_1})\min\left(\left|\lambda_{v_0}\right|, \left|\lambda_{v_1}\right|\right)$$

And

$$\lambda_{u_1} = g(\lambda_{v_0}, \lambda_{v_1}, \hat{u}_0) = \begin{cases} \lambda_{v_0} + \lambda_{v_1} & when\ \hat{u}_0 = 0, \\ -\lambda_{v_0} + \lambda_{v_1} & when\ \hat{u}_0 = 1 \end{cases}$$

where λ_{v_0} and λ_{v_1} correspond to inputs and are replaced with *y* in the last stage of the recursive decoding. To reduce the implementation complexity, [28] presents a merged-PE in which hardware resources are shared between the *f* and *g* functions. No contention arises from applying this optimization as a PE is never required to perform *f* and *g* simultaneously. Since the *f* function is more complex than *g*, the decoder uses sign-magnitude representation of LLR values which simplifies the magnitude comparisons. An array of *P* processing elements is implemented and is fed up to 2*P* LLR values every cycle. In addition, it has access to up to *P* partial-sum values when *g* is performed. A maximum of *P* LLR values can be produced every cycle.

4.1.2 Partial-Sum Update Logic

Partial sums are the results of combining bit estimates in various stages of the decoder and are used as inputs to the *g* function. Each estimated bit is involved in multiple partial sums that are updated once the bit estimate is available. This is

accomplished by storing the partial sums in independent registers with appropriate enable logic.

4.1.3 Memory

There are two major memory groups in the SP-SC decoder: the LLR memory and the partial-sum memory. The PEs read and writes LLR value simultaneously; therefore, the LLR memory must be able to provide the new data when write-while-reading condition occurs. This is achieved using a *P*-LLR bypass buffer composed flip-flops whose contents are provided in the case of a write-while-reading; while a random access memory (RAM) is used for the remaining value to save area. The partial-sum memory is implemented using registers.

4.2 Simplified Successive-Cancellation Decoding

The sequential nature of SC decoding and the large iteration count of BP decoder have limited the throughput of polar decoder. Simplified successive-cancellationn (SSC) decoding was the first method to offer major improvements in throughput. It improves on SC decoding by exploiting the recursive nature of polar code construction to increase decoder parallelism. Figure 4.1a shows a tree representation of a polar code in which each constituent code and the two codes whose concatenation it corresponds to are represented using a nodes and its two children. The leaf nodes represent either frozen bits, the white nodes, or information bits, the black nodes.

SC decoding is performed by a node that receives a likelihood vector α_v, calculates the input to its left child, α_l, and uses the bit estimate of that child, β_l, to calculate the likelihood input to the right child, α_r. Once the bit estimate β_r is available, it is combined with β_l to yield β_v, which is then passed to the parent node. The output β_v of a frozen bit is known a priori, 0 when frozen bits are set to 0, and the output of an information node is calculated using α_v as the LLR value.

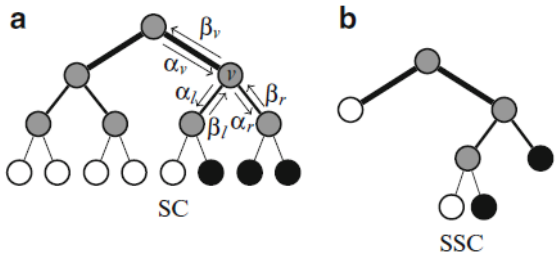


Figure 4.1: Decoder trees corresponding to the SC and SSC algorithms.

It was noted in [25] that a node whose descendants are all frozen nodes corresponds to code of rate 0 and its output β_v is known a priori. More importantly, it was shown that a node whose children are all information bits corresponds to code of rate 1 that can be decoded using maximum-likelihood decoding by applying threshold detection on α_v directly to obtain β_v. Therefore, constituent codes of rate 0 and rate 1 can be decoded directly without traversing the corresponding sub-trees in the decoder graph. This is illustrated in Figure 4.1b,

where the decoder graph is trimmed to remove such sub-trees. Such trimming was shown in [34] to improve the throughput by up to 12 times compared with SC decoding for codes of length 32768.

4.2.1 Two-Phase Successive-Cancellation Decoding

While not fully an SSC decoder, the two-phase successive-cancellation (TPSC) decoder [35] was the first to employ elements of SSC to improve decoding throughput in some parts of the decoder graph. The aim of the TPSC decoder is to reduce implementation complexity and RAM requirements. This is achieved by exploiting the array structure of the polar codes and decoding in two distinct phases. The N bit input vector u, including the frozen bits, is arranged as $\sqrt{N} \times \sqrt{N}$ array, U, which is encoded using $G\sqrt{N}$ to yield $V = G\sqrt{N}$. The codeword array, X, is obtained from V using $X = V^T G\sqrt{N}$. When X is rearranged into a $1 \times N$ vector, it is equal to $x = uGN$. The TPSC decoder divides the decoding process into $\sqrt{N}$ cycles. Each cycle consists of two phases: the first corresponds to X and the second to V. Since the first phase decoder, P1, corresponds to the larger stages in the polar code, it stores computations in RAM, which is area efficient and has addressing logic built-in. While P2 uses flip-flops, which are faster than RAM, but scarcer on FPGAs.

Any soft-input hard-output polar decoder can be used to implement the P2 decoder. The authors in [35] used a parallel SC decoder with 1-bit look-ahead storing its results in registers. The P1 decoder is a soft-input soft-output SC decoder that

outputs $\sqrt{N}$ LLR values in parallel. P1 loads the channel LLR values from RAM and stores the LLR inputs to the boundary stage, $log_2\sqrt{N}$ in registers. P2 reads those values and calculates the partial sums for the boundary stage, which are then used by P1 to continue decoding. The architecture in [35] makes no provisions for storing the results of internal calculations in P1, reducing the memory required by the decoder, but increasing latency as these values are recalculated for every decoding cycle. To reduce latency, the TPSC decoder implements the SSC rate-0 and rate-1 tree pruning operations, but only at the boundary stage.

4.3 Overall Decoder Architecture

The Due to the large number of nodes and node combinations, it was proposed in [36] that representing the polar code structure as a precomputed list of instructions would lead to more efficient decoders. As a result, the proposed decoder has an architecture similar to that of a processor. An overall view of this decoder is shown in figure 4.2. Before the decoding process starts, instructions are loaded into the instruction memory. Channel LLR values are loaded into the channel RAM via the channel loader. The controller fetches the first instruction and the decoding process starts. α values are read from α-RAM and channel RAM and written to α-RAM. Similarly, β values are written to and read from β -RAM and the estimated codeword is written to the codeword RAM. Using separate memories for internal α values and the channel LLR values is required to enable loading-while-decoding, which is required to prevent the decoder from stalling and to maintain throughput.

4.3.1 Processing Unit Architecture

The processing unit contains the logic required to perform the operations needed by all the nodes and the merged nodes. Figure 4.3 shows the architecture of the processing unit performing these operations. The inputs to this unit are: α a vector of up to 2*P* valid LLRs, β_0 a vector of up to *P* valid bit values generated by the left child, β_1 a vector of up to *P* valid bit values generated by the right child, and 0 a *P*-bit all-zero vector. The outputs are: α a vector of up to *P* valid LLRs corresponding to the α*l* or α*r* outputs of the node; β_0 and β_0' , two vectors of up to *P* valid bit values each, corresponding to the β_v output of the node. The two functions producing α outputs are F and G. The multiplexer m_1 selects between their outputs. In addition, G has a specialization when the node's left child is a rate-0 node, denoted G-0R. The multiplexer m_0 is used to set the β input to G function to 0.

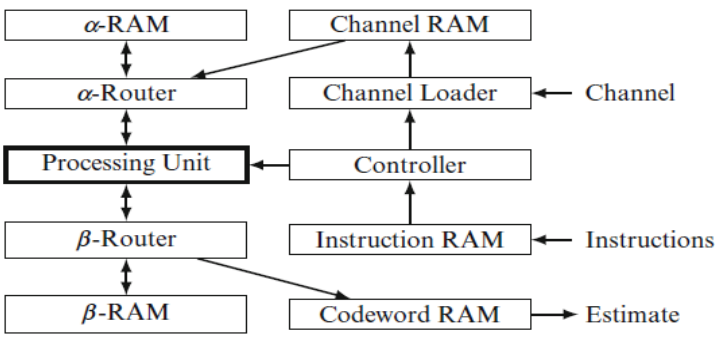


Figure 4.2: Top-level architecture of the Fast-SSC decoder.

Repetition, Repetition-SPC, and ML nodes are limited to constituent codes of length less than or equal to P in [36]. Therefore, their output is entirely contained in β_0'. The multiplexer m_2 selects the correct β_0' from among the candidates, which also include the first half of COMBINE's output

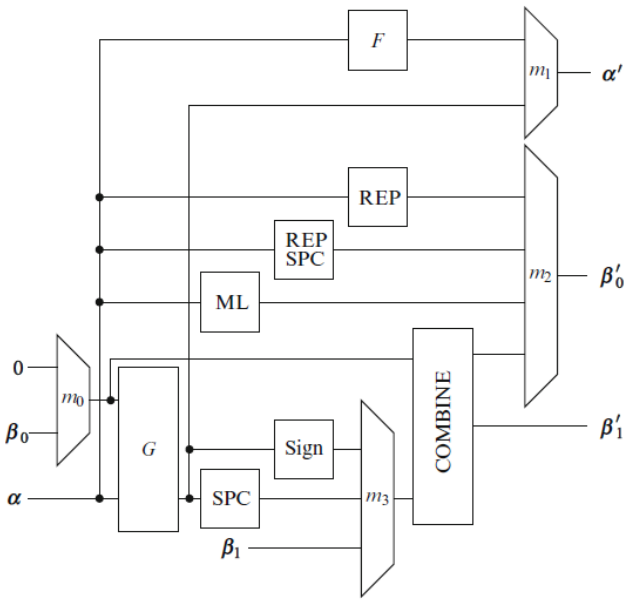


Figure 4.3: Processing Unit Architecture.

β_1 is always the second half of COMBINE's output. The left input to COMBINE is selected by m_0 as either β_0 or 0. Due to node mergers, the second input varies: when the function performed is P-01 or P-R1, it is the sign of the output from G; for P-0SPC and P-RSPC, the input come from the SPC decoder that uses the output of G as its input; finally, in the case of COMBINE and COMBINE-0X, β_1 is used. This selection is process performed by m_3.

4.4 Belief Propagation Decoder Implementation

The first hardware polar decoder implemented was a semi-parallel BP decoder that uses the min-sum algorithm [37]. It was shown that it required a large number of iterations, ~50, to match the error-correction performance of an SC decoder for a (1024, 512) polar code [1]. Table 2 compares the BP decoder [37] with the SP-SC decoder [1] for the (1024, 512) polar code where both decoders are configured to have the same error-correction performance. It can be seen from the table that the SP-SC decoder uses fewer memory resources and has eight times the information throughput compared to the BP decoder.

This large number of required iterations significantly degrades the BP decoder throughput in spite of the inherent parallelism of the algorithm and has limited the research targeted at implementing BP decoders.

Table 2: Implementation results for the BP and SC decoders on the Xilinx Virtex IV XC4VSX35-12 using the (1024, 512) polar code [37].

Algorithm	LUT	FF	BRAM	T/P (Mbps)
BP	2,794	1,600	12	2.78
SP-SC	2,600	1,181	5	22.22

4.5 Implementation Comparison

When comparing the different polar decoder implementations, it is important to ensure that they are capable of sustaining their throughput. The decoders must support loading-while-decoding or their throughput will be degraded. The easiest method to implement loading-while-decoding is to buffer additional codewords. In this section, the RAM numbers were modified where needed to ensure that the decoders can buffer an additional codeword. In that table it can be seen that TP-SC uses the fewest resources and has the highest clock frequency; while Fast-SSC has the highest throughput. The high frequency of TP-SC is a result of the aggressive buffering using registers. Fast-SSC uses 1.6, 1.2, and 1.4 times the LUTs, registers, and RAM compared to TP-SC, respectively, but has 7.8 times the throughput when both decoders use P = 128. Increasing P to 256 in the Fast-SSC increases the LUTs and registers used significantly, and increases the information throughput to 1,091 Mbps.

Table 3: Post-fitting and information througput results for a (16384,14746) code on the Altera Stratix IV EP4SGX530KH40C2 [37].

Algorithm	P	LUTs	Reg.	RAM (bits)	F (MHz)	Info. T/p (Mbps)
SP-SC	64	29,897	17,063	265,984	113	48
TPSC	128	7,815	3,006	196,480	230	106
Fast-SSC	128	13,388	3,688	273,740	106	824
Fast-SSC	256	25,219	6,529	285,336	106	1,091

4.6 Hardware Architecture of Polar Decoder Based on the Proposed BP algorithm

In this section we will explain the hardware architecture of the polar decoder based on the proposed BP algorithm in chapter 3 section 3.4. In this hardware arhitecture we take 128 bits for information signal and 256 bits for encoded signal. In order to get decoded signal from encoded bit stream, soft decoding procedure is applied by using the modified belief propagation algorithm explained in section 3.4 of chapter 3.

Channel decoding in a polar decoder is based on the log likelihood ratio (LLR) of a binary random value $x \in \{\pm 1\}$ or $x \in \{0,1\}$ defined by the following equation,

$$L(c_i) = \log\left\{\frac{P(c_i = 0 | r_i)}{P(c_i = 1 | r_i)}\right\} \qquad (4.1)$$

where c_i represents the LLR of the estimated codeword and r_i represents the LLR of the received codeword from the channel. The BP decoder operates by passing the message (LLR values) on tanner graph. Let $L(Q_{ij})$ denote L value message passed from check node i to variable node j and $L(P_{ij})$ denote L value message passed from variable node i to check node j. Then the check node i will update its message as [30]:

$$L(Q_{ij}) = \prod_{j'} \alpha_{ij'} \phi\left[\sum_{j'} \phi(\beta_{ij'})\right] \quad (j' = 1,2,...,n \text{ and } j' \neq j) \qquad (4.2)$$

where $\alpha_{ij} \triangleq sign[L(P_{ij})]$ and $\beta_{ij} \triangleq |L(P_{ij})|$. The ϕ function is defined as [30]:

$$\phi(x) = -\ln[\tanh(x/2)] = \ln[(e^x + 1)/(e^x - 1)] \qquad (4.3)$$

Similarly, the L value of variable node variable node i connected to check node j can be found as:

$$L\left(P_{ij}\right)=L\left(P_{ij(previous)}\right)+\sum_{i'}L\left(Q_{ij}\right) \quad \left(i'=1,2,...,n \text{ and } i'\neq i\right) \qquad (4.4)$$

And the L value for the decoding decision can be found as [30]:

$$L\left(P_j\right)=L\left(c_j\right)+\sum_i L\left(P_{ij}\right) \qquad (4.5)$$

Let:

$$c_j=\begin{cases}1 & if\ L(P_j)<0\\ 0 & else\end{cases} \qquad (4.6)$$

During this decoding process, the messages are exchanged back and forth in a number of decoding iterations between variable nodes and check nodes. We can divide the hardware architecture of the BP decoder into four blocks: initialization block, check node update block, variable node update block and decoding decision block as shown in figure 4.4 - 4.7. The equations 4.2 and 4.4 are responsible for the implementation check node update block and variable node update block, respectively. While equation 4.5 is responsible for the decoding decisions in the final stage. In the decoding architecture, these equations are implemented in different stages and the working principle of these stages is discussed next.

Decoder Datapath:

The datapath of the parallel decoder is illustrated in figure 4.4 where for the purpose of clarity only one variable node and one check node have been shown. The messages from variable nodes to check nodes and the messages returning from the check nodes to the variable nodes are carried on distinct sets of wires. On first inspection, this doubles the number of message wires that need to be routed when

compared to using a single set of wires in a time division multiplexing fashion. However, the control signal distribution, bidirectional buffers, and logic overhead associated with reusing a single set of wires negate the potential routing advantage of using a single set of wires [38]. To ensure correct synchronous execution of the message passing algorithm, it is necessary to insert registers into the datapath to align the messages corresponding to each decoder iteration. By associating the registers with the variable nodes as shown in figure 4.4, the check nodes become purely combinatorial logic blocks which simplifies the floor planning of the overall decoder [38].

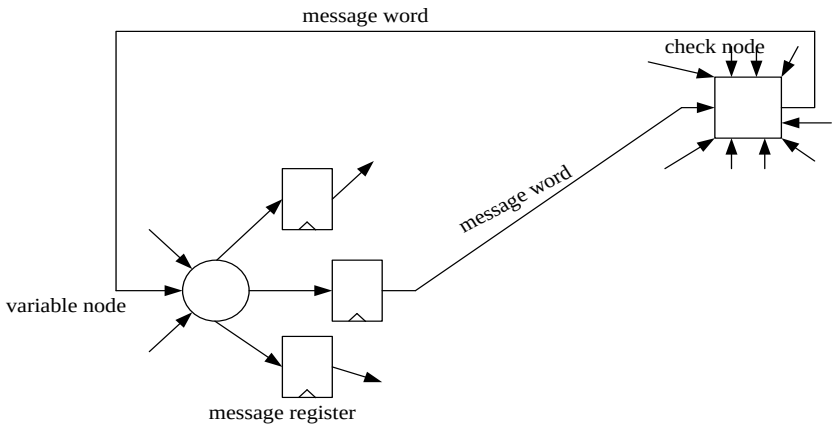


Figure 4.4: Datapath architecture for the decoder [38].

Initialization Block:

This block receives 8 packets each of 14 bits which contains the channel LLR values c_i, whose value is shifted by 2 and then again shifted by the value of σ^2 according to the following equation [30]:

$$L\left(P_{ij}\right) = L\left(c_i\right) = 2r_i / \sigma^2$$

This value is then stored in the register $L\left(P_j\right)$ for further processing. The hardware architecture of this block is shown in figure 4.5.

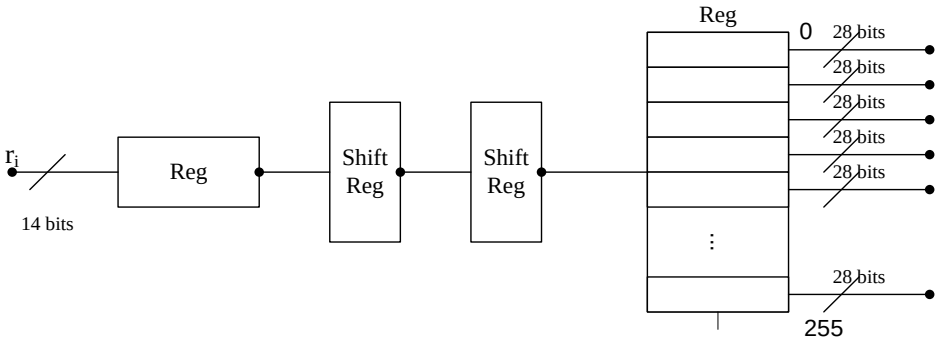


Figure 4.5: Initialization block.

Check Node Update Block:

In this block equation 4.2 and 4.3 are realized to hardware architecture as shown in figure 4.6. The messages received from variable nodes $L\left(P_{ij}\right)$ are processed in this block. All the check nodes calculate the equation 4.3 according to the look up table (LUT). The LUT is implemented using ROM. From the LUT the messages are stored in phi1 register. Then all the values of the check nodes are added and subtracted from their own values to calculate the sum_phi1 register's values. Again the values are sent to LUT and the final messages are sent to variable nodes. The messages in this block are executes as below:

1. Take the message from LLR memory and calculate the $\phi\left(x\right)$ function.

2. All the check nodes are added and then subtracted individually from their own values.

3. Again, take the message from LLR memory and calculate the $\phi(x)$ function.

4. For each x, perform in temporary register $C(j)=T(j)\times V(j)$ where $j=0,1,...,x-1$.

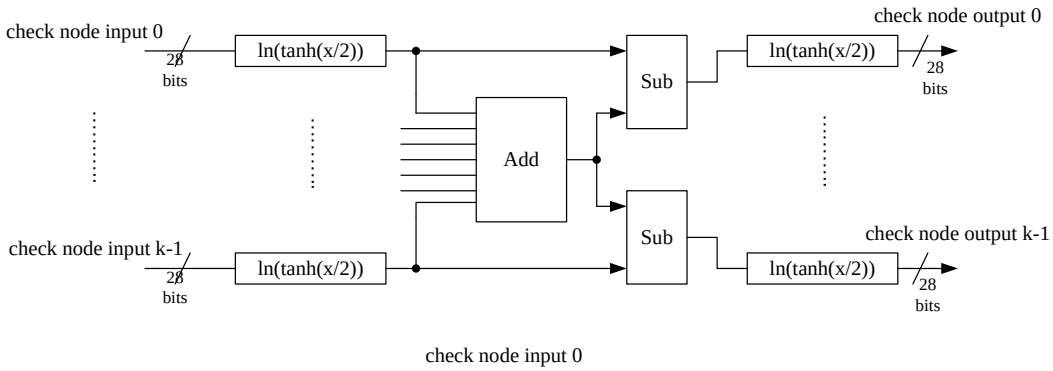
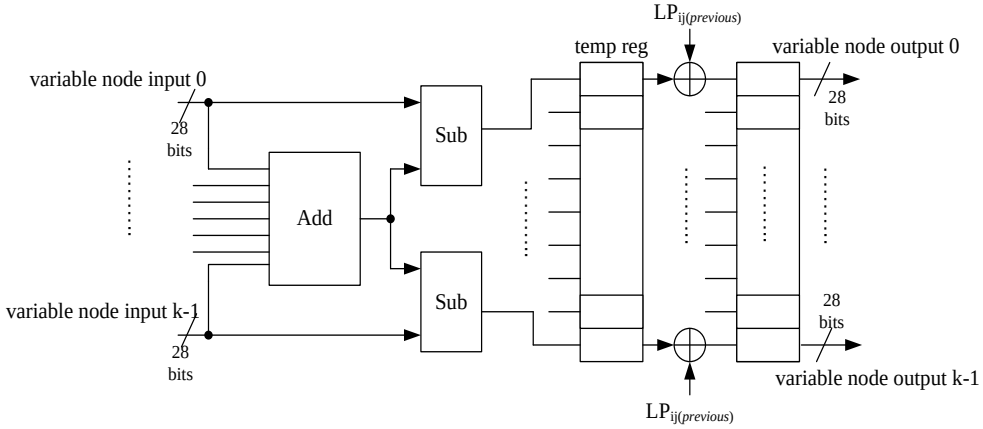


Figure 4.6: Architecture for check node with k-inputs.

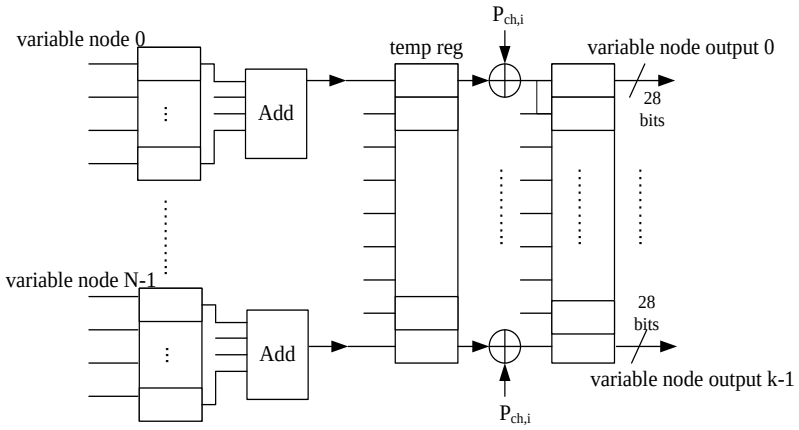
Variable Node Update Block:

The architecture of the variable nodes is shown in figure 4.7. The variable nodes contain all of the registers in the decoder, including the decoder message registers and the shift registers. At the packet start signal, the decoding of a new packet is commenced and the previous packets results are loaded into the output shift registers. For the first decoding iteration, the messages sent to the check nodes are the sign-magnitude representations of the log-likelihood of the received value, since for a Gaussian channel the received values are the log-likelihoods up to a

scaling factor. At the first iteration, each variable node adds all the received messages and subtracts their own messages. This message is then sent to the check nodes. In the next iteration the variable node adds its' previous value as well according to the proposed BP method in chapter 3 section 3.4. The messages in this block are updated by the following procedure:



(a)



(b)

Figure 4.7: Architecture of variable node. (a) the architecture based on proposed BP decoder. (b) outputs produced for the decision block.

1. All the messages received by each variable node is added and then subtracted by its own value.

2. The previous value is added to these values of variable nodes.

3. The variable nodes' value is updated by adding the channel values to their current values.

Decoding Decision Block:

After finishing the check node and variable node blocks, check node block's memory contains all the nodes and edges of the processing information. In this block equation 4.5 and 4.6 is implemented. Then, data is being read from the decoder memory and used as the decoded output.

From the above discussion, the whole decoding process is explained through the LLR operation, CNU and VNU blocks and the final decoding stage. It is performed by passing messages from check nodes to variable nodes and vice versa. Therefore, this decoding is an iterative process and messages are decoded from the code after the certain iterations.

5. CONCLUSIONS

5.1 Summary

In this thesis, we studied the BER performance of polar decoders under belief propagation decoding algorithm. We described a damping belief propagation approach that can be used to achieve a better performance for short polar codes while keeping the complexity same as of the original belief propagation decoder. We then provided an in-depth analysis of the modified belief propagation decoder. We showed that if the variable nodes in the Tanner graph of polar codes are multiplied by their previous messages, it can improve the reliability of propagated messages in the Tanner graph, which results in the better performance of polar codes. We also analyzed the factor graph of polar codes and showed that the BP algorithm can be run on the Tanner graph constructed by the parity check matrix of polar codes. Simulation results show that the proposed method achieves a gain of 2 dB at higher SNR values over the original BP decoder. It is also shown that the proposed method does not increase the complexity of decoding; the complexity is the same as that of the original BP decoder. We also give the number of iterations taken by both the decoders i.e. standard and modified BP algorithm, and it is shown that the number of iterations taken by the modified BP decoder is less by 10-25 iterations as compared to the standard BP. In the end, we discuss the hardware architecture of the proposed BP decoder and showed that it can be use for the practical purposes.

5.2 Future Work

As we have discussed that the there are many proposed schemes in order to get better BER performance of channel codes under belief propagation algorithm. One such scheme was employed in [13], where guessing algorithm is used to improve the BER performance of polar codes. In this work we proposed the use of damping algorithm to improve the performance of polar codes. As a future work other schemes proposed for improving the performance of channel codes under belief propagation algorithm can be studied and their performances can be studied in different scenarios for polar codes.

Figure 4.4 shows the datapath architecture for the decoder. It can be seen in the figure that each variable node is connected to check node with the help of wires. The messages between variable nodes and check nodes are passed over these wired connections. Thus by increasing the code length the number of variable nodes and check nodes will increase or decrease. Furthermore, the connections between variable nodes and check nodes have to be rewired according to the new parity check matrix H. This is the limitation of this hardware that we have to redesign it for different code lengths. The hardware is non scalable and non-configurable. Further work is required to make this hardware scalable and reconfigurable to compare with modern hardware architectures.

REFERENCES

[1] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes. Proceedings of IEEE International Symposium on Information Theory (ISIT), pp. 1173-1177, 2008.

[2] E. Arıkan and E. Telatar. On the rate of channel polarization. IEEE Int. Symp. Inform. Theory, Seoul, Korea, pp. 1493-1495, 2009

[3] S.B. Korada et al. Polar Codes: Characterization of Exponent, Bounds, and Constructions. IEEE Transactions on Information Theory, vol. 56, no. 12, pp. 6253–6264, 2010

[4] P. Som et al. Improved large-MIMO detection based on damped belief propagation. IEEE International Information Theory Workshop (ITW 2010), Cairo, pp. 1–5, 2010.

[5] C.E. Shannon. A mathematical theory of communication. Bell Syst. Tech. journal, pp. 623-656, July 1948.

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

[7] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. Proceedigs of IEEE International Conference on Communications (ICC), vol. 2, pp. 1064-1070, 1993.

[8] I. Tal and A. Vardy. List Decoding of Polar Codes. Proceedings of the IEEE International Symposium on Information Theory (ISIT ’11), St Petersburg, Russia, pp. 1–5, 2011.

[9] K. Niu and K. Chen. Stack decoding of polar codes. IEEE Electronics Letters, Vol. 48, no. 12, 2012.

[10] K. Niu et al. Improved Successive Cancellation Decoding of Polar Codes. IEEE Transactions on Communications, vol. 61, no. 8, pp. 3100–3107, 2013.

[11] K. Niu and K. Chen. CRC-Aided Decoding of Polar Codes. IEEE Communications Letters, vol. 16, no. 10, pp. 1668–1671, 2012.

[12] E. Arıkan. A Performance Comparison of Polar Codes and Reed-Muller Codes. IEEE Communications Letters, vol. 12, no. 6, pp.447–449, 2008.

[13] E. Arıkan et al. Performance of short polar codes under ML decoding. ICT-Mobile Summit Conf. Proc, 2009.

[14] A. Eslami and H. Pishro-Nik. On Bit Error Rate Performance of Polar Codes in Finite Regime. Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing, Allerton (AACCCC '10), Allerton, Ill, USA, pp. 188–194, 2010.

[15] Ubaid U. Fayyaz, "Polar Code Design and Decoding for magnetic Recording", PhD Thesis, Georgia Institute of Technology, USA.

[16] D.J.C. MacKay and R.M. Neal. Near Shannon limit performance of low density parity check codes. Electron. Lett., vol.33, no. 6, March 23, 1997.

[17] D.J.C. MacKay and R.M. Neal. Good error-correcting codes based on very sparse matrices", available online.

[18] Dino Sejdinovic, "Topics in Fountain coding", Master's Thesis, University of Bristol, 2009.

[19] T. Richardson and R. Urbanke. Modern Coding Theory, Cambridge University Press, NY, p.49-65, 2008.

[20] S. Lin and Daniel J. Costello. Error Control Coding. Linear Block Codes. Pearson Prentice Hall, USA.

[21] E. Arikan. A survey of reed-muller codes from polar coding perspective. Information Theory Workshop (ITW), 2010 IEEE, Jan 2010, pp. 1–5, 2010.

[22] R. Mori and T. Tanaka. Performance of polar codes with the construction using density evolution. Communications Letters, IEEE, vol. 13, no. 7, pp.

519 –521, July 2009.

[23] P. Trifonov and P. Semenov. Generalized concatenated codes based on polar codes. Wireless Communication Systems (ISWCS), 2011 8th International Symposium on, pp. 442 –446, Nov. 2011.

[24] K. Niu, K. Chen, Jiaru lin, and Q.T. Zhang. Polar codes: Primary concepts and practical decoding algorithms. IEEE Communications Magazine, July 2014.

[25] A. A. Yazdi and F. R. Kschischang. A Simplified Successive-Cancellation Decoder for Polar Codes. IEEE Commun. Lett., vol. 15, no. 12, pp. 1378-80, Dec. 2011.

[26] Nana Traore, Shashi Kant, Tobias Linstorm Jensen. Message passing algorithm and linear programming decoding for LDPC and linear block codes. Report. Aalborg University, Denmark.

[27] S. B. Korada. Polar codes for channel and source coding. Ph.D. dissertation, EPFL, Lausanne, 2009. [Online]. Available: http://library.epfl.ch/theses/?nr=4461.

[28] Rumi Aoyama and Yuichi Kaji. Improvement of the Forced-Convergence Decoding for LDPC Codes. International Symposium on Information Theory and its Applications, ISITA, 2008.

[29] S. Lin and Daniel J. Costello. Error Control Coding. Low Density Parity Check Codes. Pearson Prentice Hall, USA.

[30] Yuan Jiang. A Practical guide to Error-Control Coding Using MATLAB. Modern Codes. Artech House, Norwood, MA.

[31] N. Goela et al. On LP Decoding of Polar Codes. IEEE Information Theory Workshop (ITW) , 2010.

[32] C. Leroux, A. Raymond, G. Sarkis, W. Gross. A semi-parallel successive-

cancellation decoder for polar codes. IEEE Trans Signal Process 61(2):289–299, 2013

[33] G. Sarkis and W. J.Gross. Increasing the throughput of polar decoders. IEEE Commun Lett. 17(4):725–728, 2013

[34] A. Pamuk and E.Arikan. A two phase successive cancellation decoder architecture for polar codes. In: Proceedings of the IEEE international symposium on information theory ISIT 2013, July 2013, pp 1–5, 2013

[35] G. Sarkis, P. Giard, A. Vardy, Thibeault C, Gross WJ (2014) Fast polar decoders: algorithm and implementation. IEEE J Sel Areas Commun (JSAC).

[36] A. Pamuk. An FPGA implementation architecture for decoding of polar codes. In: 2011 8th international symposium on wireless communication systems (ISWCS), pp 437–441, 2011.

[37] C. Chavet and P. Coussy. Advanced hardware design for error correcting codes. Springer International Publishing Switzerland, pp. 33-45, 2015

[38] Andrew J. Blanksby and Chris J. Howland. A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low density parity check code decoder. IEEE Journal of Solid State Circuits, vol. 37, no. 3, March 2002.

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my advisor Prof. Dr. GoangSeog Choi. This thesis would not have been in current shape without his patient mentoring despite my frequent mistakes and constructive criticism. He is the friendliest, caring, easy going and encouraging advisor that anyone could wish for. I would also like to thank Prof. Dr. Young-Sik Kim for helpful discussions throughout my Master's degree.

I would also like to thank Prof. Dr. Younk-Suk Shin and Prof. Dr. Young-Sik Kim for serving on my thesis defense committee.

I would like to thank all my lab mates in System on Chip Design Lab. Tariq, Adnan, Sadeque, Park, Jong and Shim helped me in the hours of distress and made my journey in the lab a lot easier. Especially Adnan who not only helped me in the research work by giving innovative ideas but he also helped me by being a good friend.

I would like to express my heartfelt thanks to my friends Hasan, Mohsin, Dibash, Lynn and Ahlam who made my life at Chosun University full of fun.

There are no words in which I can say thanks to my parents who always helped me and provided me with all the support I needed throughout my life. My journey to Korea would not have been possible without their continuous moral support and prayers. Living away from them was the hardest sacrifice I have made for graduate studies. Also, my sister Rabia and brothers Bilal, Adeel and Nabeel helped me a lot in this journey. Thank you all!