List Decoding of Polar Codes

EMILIA JOHANSSON
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

List Decoding of Polar Codes

Emilia Johansson
elt12ejo@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Michael Lentmaier,
Saeedeh Moloudi

Examiner: Thomas Johansson

October 31, 2017

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Channel coding is an important instrument used in communication to correct errors that occur on channels. It is interesting to find the best suited channel code for different communication systems.

Polar codes have been in the spotlight lately for their simple structure and performance when in combination with list decoding and cyclic redundancy check code. Polar codes have a recursive structure that makes them interesting to implement in hardware, and they have lately been chosen as a standard for short code communication in 5G to correct bit errors. However, polar codes by themselves are shown to work poorly for practical block lengths, and it is therefore of interest to research them further.

This thesis investigates polar codes with a suggested combination of list decoding and CRC. The combination is shown to improve short polar codes enough to compete with the best-known channel codes today for short block lengths. This thesis investigates why this combination works so well with polar codes. The focus lies on the selection of frozen bits in polar codes, in comparison with the similar Reed-Muller codes, and on the size and bit-placement of the CRCs. All investigations focus on codes with length 128 bits and code rate 0.5.

We find that a slightly modified frozen bit selection can result in huge performance changes of polar codes. We also find how the use of a list decoder with a large list size improves Reed-Muller codes such that they challenge polar codes both with and without added CRCs.

We study if a long CRC is preferred, or if the code performance can be improved by dividing it into several shorter CRCs spread out over the polar code. Results from different modifications to polar codes are presented and discussed.

Keywords: Channel coding, polar codes, list decoding, CRC, Reed-Muller codes, short codes, 5G.

Acknowledgments

First, I want to thank my supervisor Michael Lentmaier for making this Master Thesis project possible. I am grateful for his knowledge and enthusiasm in the project, and all his guidance that helped me solve problems that occurred along the way.

I would also like to thank my co-supervisor Saeedeh Moloudi for all her help; all time spent finding resources to help me understand the theory behind the project and all her valuable hints for writing this thesis.

I also want to thank my housemates for always questioning my ones and zeros, and for their daily reminders that I need to simulate.

Finally, I send thanks to my family and the house dog.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AWGN	Additive white Gaussian noise
B-DMC	Binary-input Discrete Memoryless Channel
BEC	Binary Erasure Channel
BER	Bit Error Rate
BSC	Binary Symmetric Channel
CRC	Cyclic Redundancy Check (Code)
CRC-8	8 bits CRC
LDPC	Low-Density Parity-Check (Code)
LLR	Log-Likelihood Ratio
ML	Maximum Likelihood
PC	Polar Code
RM	Reed-Muller (Code)
SC	Successive Cancellation
SCL	Successive Cancellation List
SNR	Signal-to-Noise Ratio
WER	Word Error Rate

Chapter 1
Introduction

Transferring information between devices plays an important role today. The rising demand for good data transmission and the planning of 5G increase the interest in finding ways to improve the rate and reliability of data transmission. Channel coding plays a significant role in improving these; they let us reduce the number of errors occurring on channels to any desirable level.

Recently, polar codes have been in the spotlight, and much research is currently being performed on them. Polar codes are proven to achieve channel capacity for long block lengths, and they are mathematically simple with a recursive structure and low complexity. These are highly desirable properties in channel codes and make them attractive for, e.g., hardware implementation. Polar codes, therefore, received much attention already from the start. However, they have also been shown to perform relatively poorly for all practical short block lengths, of up to a few thousand bits, and are hence not commonly used in practice without modification.

It is well known that codes that work well for short block lengths are hard to construct. The low complexity and excellent results of polar codes for long block lengths have motivated much research, trying to improve them also for short block lengths.

Recent research has found a way to improve the code performance, using a combination of successive cancellation list (SCL) decoding and cyclic redundancy check (CRC) code with the polar code. The SCL decoder extends the successive cancellation (SC) decoder that was originally used with polar codes. It gives us a list of possible codewords when decoding, and the CRC check is used in the last decoding step to help select the correct word from the list. These added elements reduce the decoding simplicity but have shown impressive results for short block lengths. Polar codes with this combination go from performing relatively bad to competing with well-researched codes for short block lengths. They are even the best performing codes on some channels. Another recent development is that polar codes have been chosen as the standard to implement in 5G for short block lengths.

It is now interesting to find out why. What is it about the specific combination of polar codes, list decoding and CRC that creates better codes?

There are many details about polar codes that can be researched further, and many questions are still unanswered. We know for example that the older Reed-Muller codes have a similar structure to polar codes. Moreover, Reed-Muller codes

are stronger than polar codes with larger minimum distances between codewords. Is it possible to use the same techniques of list decoding and CRC on those codes and get similar improvements? The structural difference between the codes lies in the selection of frozen bits, which in short means what set of possible codewords we can send on the channel. How does the selection of frozen bits influence the improvement caused by list decoding and CRC? In classic uses of CRC, the computed CRC-bits are placed to the end of the code. However, in combination with polar codes, could it possibly be better to put the CRC in another position, or even separate the bits used into a few different CRCs along the codeword? Some early polar code decisions are commonly made on weaker channels, so could an early removal of incorrect paths with CRC in the list possibly improve the code? What it the CRC size needed to fulfill the necessity of the polar code? This thesis investigates and aims to answer some of these questions.

1.1 Project Goal

In this thesis, polar codes with list decoding and CRC are investigated for short block lengths of 128 bits, focusing on how variations of the three components, frozen bits, CRC and list decoding, change the polar code performance. First, the add of list decoding and CRC to polar codes will be compared with the same add to Reed-Muller codes, to understand how the selection of frozen bits influence the result. Second, a few different modifications to the traditional CRC will be made in size and placement in the code, to find if the combination can improve for short block length polar codes.

The project goal is to understand what about the list decoding with CRC that works so well with polar codes, what role the frozen bit selections play, and possibly to find ways to improve the combination.

1.2 Related Work

The interest of this thesis project grew from previous related work on polar codes, their similarity with Reed-Muller codes, and results showing polar codes with list decoding and CRC to outperform other codes available today for short block lengths on some channels.

Arikan first introduced polar codes in his 2008 article, where they were shown to achieve channel capacity for infinite block lengths [1]. Already in that article, the close association between polar codes and the older Reed-Muller (RM) codes was mentioned, and the relation was further explored by Arikan in [2]. A performance comparison between the two codes, without added outer codes and for short block lengths of 32 and 256 bits, was made in [3], finding polar codes to outperform RM codes for short block lengths under SC decoding. They were also compared in [4] using maximum likelihood (ML) decoding, under which the RM code performed better than polar codes, but at a high cost of decoding complexity.

Tal and Vardy suggested the use of list decoding with CRC on polar codes, in the prize awarded [5], with their first results shown in [6]. The article [5] includes results showing improved polar codes of 2048-bit block lengths beating LDPC

codes of length 2304 and rate 0.5. The improved polar code combination was also looked at in [7], showing polar codes of the short block length of 128 bits and rate 0.5 to outperform both LDPC and turbo codes of the same size and code rate, for frame error rates down to 10^{-6}.

Some proposed methods of improving the frozen bit selection for polar codes have been made. Vangala and Viterbo compared some different suggested ways of selecting frozen bits in [8] for (2048, 1024) codes. It was shown in [9] that LLR values of unfrozen bits could be used to identify unreliable channels and swap them with some good frozen channels, improving the polar code performance. Experiments have also been done with combining polar codes and Reed-Muller codes in different ways in [10] and [11], finding improved results of these interpolated/hybrid RM-polar codes.

Reed-Muller codes are well known, they were invented in 1950's but have only recently been proven to reach capacity. These codes, being older than polar codes, have been the subject of a lot of research through the years, including many suggested improvements of the decoder. Recursive list decoding and improvement of the frozen bit selection for short Reed-Muller codes was suggested in [12].

Many studies have also been made, trying to find the best CRC polynomials to use for different block lengths for long burst error detection, among them is [13], describing a good practical way of selecting general CRC polynomials for short codes.

1.3 Thesis Contributions

Previous research has demonstrated that the asymptotic design criteria for polar codes, working well for long polar codes, does not show great performance in short polar codes. However, results in [7] showed that the combination suggested in [5] could be used to improve polar codes enough to compete with the best. The desired polar code simplicity and implementation in 5G motivated us to investigate the code performance of short polar codes further.

In this thesis, we gain a better understanding of why CRC and list decoding work so well with polar codes. The effect of the three variables, list decoding, CRC and the selection of frozen bits, are investigated, together with the connection between polar and Reed-Muller codes.

The thesis first compares some different frozen bit selections, reconstructing and verifying results from recent research for short block lengths, finding how the selection of frozen bits changes the code performance. Comparisons are made with the same combination added to Reed-Muller codes.

The selection of CRC is looked at, and observations are made about how different CRC polynomials change the outcome of the code. The advantages of splitting a longer CRC at the end of the decoder into several shorter CRCs along the decoding process are demonstrated. We suggest how a split CRC can be used to improve code performance without changing expensive parameters such as polar code rate or list size.

All investigations are performed with simulations from code written in Matlab.

1.4 Outline of Report

The report outline is as follows. The Background chapter introduces theory and concepts that this thesis is based on. Theory includes a detailed explanation of polar codes, Reed-Muller codes, list decoding, and cyclic redundancy check code.

Following, every chapter investigates a certain aspect of polar codes, and includes problem description, design method, results, and conclusions.

Chapter 3 reconstructs polar code results from previous research, for short block lengths of size 128 bits. The polar code design adapts the frozen bit selection after the channel it is designing for. This chapter starts by comparing the performance of polar codes designed for different channels. The best performing polar code design for a range of channels is then selected to be used in the rest of this thesis. The code performance is also compared with that of the RM code, both under SC and SCL decoding.

Following, Chapter 4 adds CRCs to both polar codes and Reed-Muller codes with list decoding to find how the sets of frozen bits influences the resulting code.

The fifth chapter contains comparisons between different ways of using the CRC, including varying generator polynomials, check value position and spread-out CRC in the list decoder.

Conclusions and future work are discussed in Chapter 6.

Chapter 2
Background Theory

This chapter introduces relevant theory that the following chapters are based on. First, Section 2.1 explains channel coding and Shannon capacity, including essential terms and definitions. Following, Section 2.2 introduces channel polarization and polar codes, including a detailed description of the encoding and decoding processes, with and without list decoding and CRC. Last, Reed-Muller codes and their similarity with polar codes are explained in Section 2.3.

2.1 Coding Theory

We communicate over different channels daily. Channels include physical transmission mediums, such as wires, and wireless channels such as radio channels. When messages are transferred over a channel, bit errors can occur. It is interesting to construct messages in a way such that errors caused by the noisy channel can get detected and corrected by the receiver. This is done with a channel code, where the idea is to add redundancy to the message on the sender side and remove it in a decoding stage on the receiver side. Redundant bits are added in a way such that the decoder can use them to detect and sometimes correct errors that occurred on the channel. This increases the amount of data that can be reliably transmitted on the channel per time unit.

2.1.1 Notations

In the following chapters, we will use the code notation (N, k), where N is the length of the sent codeword in bits after encoding, and k the bit length of the uncoded information. We will use the channel notation W, improved channel after polarization W^+, and worsened channel W^-.

Bit vectors are represented by x_1^N, as short for $(x_1, ..., x_N)$. The k bit information sequence is denoted u_1^k, encoded codeword v_1^N, input on the channel is x_1^N and output is noted y_1^N. The input on the receiver is r_1^N, and output from the decoder $\hat{u}_1^k$. The Hamming weight is denoted d_h.

Figure 2.1: Encoding in the transmitter lengthens the word to be transmitted from k information bits to N bits. The channel adds noise to the codeword, and the decoder, receiving a modified codeword r_1^N, uses the extra bits to detect and/or correct possible transmission errors. The decided word after the decoder has notation $\hat{u}_1^k$ and is hopefully the same as the original information word u_1^k.

2.1.2 Channel Coding

The interest of coding arose after Shannon's 1948 article [14], where channel capacity was introduced. Channel capacity is the maximum theoretical rate at which information can be reliably transmitted over a specific communication channel. The Shannon capacity theorem sets a theoretical upper bound of the transmission rate at which data can be reliably transmitted over a link. It states that, as long as the transmission data rate does not exceed the channel capacity, coding can be used to reduce error rates on the channel to any desired level.

Channel coding is used to detect and correct errors when transferring data on communication channels. They work by adding redundancy to the information message in ways such that the receiver can use it to detect and/or correct errors that occur on the channel. Codes that can correct errors are commonly called error correcting codes, and the encoding and decoding steps in the transmitter and receiver are shown in Figure 2.1. First, a k-bit information sequence u_1^k is sent to the encoder in the sender from some information source. The k bit long information sequence gets encoded into an N-bit codeword v_1^N, where $N > k$, and transmitted on the channel. The channel is subject to noise, and hence a slightly changed received codeword r_1^N reaches the receiver and enters the decoder. The decoder uses the N received bits to decide the most likely k-bit information word $\hat{u}_1^k$. The cardinality of the set of possible codewords is 2^k.

2.1.3 Channel Models

The channel is the medium over which the data is transmitted. A channel W is usually modeled with an input alphabet $\mathcal{X}$, an output alphabet $\mathcal{Y}$, and transmission probabilities $W(y|x)$, $x \in \mathcal{X}, y \in \mathcal{Y}$. The noise level on a channel is measured in signal-to-noise ratio (SNR), which is the ratio between the signal and noise power. Some more important channel definitions follow below:

B-DMC (Binary-input discrete memoryless channel) Binary-input means that the input alphabet is $\mathcal{X} \in [0, 1]$. A memoryless channel can send bits independently of what was previously sent.

BSC (Binary Symmetric channel) A symmetric channel with alphabet $\mathcal{Y} = \{0, 1\}$. This is a simple channel where the received bit after the channel is either the sent bit, or a flipped bit. The bit is flipped with a small crossover

probability p_e. The channel is symmetric, so $p(0|1) = p(1|0) = p_e$ and $p(0|0) = p(1|1) = 1 - p_e$.

BEC (Binary erasure channel) A symmetric channel on which for each y, out of $W(y|1)$ and $W(y|0)$, one is zero and one is one, or $W(y|1) = W(y|0)$, i.e. the probability for each decision is either 1, 0 or e (erasure symbol). This means that the receiver either receives the correct bit or an erasure bit, letting the receiver know that the bit was not received correctly. This simplified channel model is often easy to analyze, and therefore commonly used to describe codes.

AWGN (Additive white Gaussian noise) A model used to mimic random noise that is added to data on channels. The added noise is normal distributed with uniform power across the frequency band.

In this report, only binary channels are considered.

2.1.4 Symmetric Capacity and the Bhattacharyya Parameter

Two important parameters for B-DMC's are defined below, the symmetric capacity and the Bhattacharyya parameter.

I(W) (Symmetric capacity) The symmetric capacity $I(W) \triangleq \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0)+\frac{1}{2}W(y|1)}$ is the highest transmission rate at which reliable communication is possible across W, using inputs of W with equal frequency. The symmetric capacity equals the Shannon capacity when W is a symmetric channel.

Z(W) (Bhattacharyya parameter) The Bhattacharyya parameter $Z(W) \triangleq \sum_{y\in\mathcal{Y}} \sqrt{W(y|0)W(y|1)}$ measures the reliability of a channel. It is an upper bound of the probability that an error occurs using ML decision, when a channel W is used to transmit a single bit.

Note that both parameters take values [0, 1], and that when one is approximately one, the other is approximately zero. Desirable are channels with Bhattacharyya parameters close to zero.

2.2 Polar Codes

Polar codes are a group of codes using channel polarization, invented by E. Arikan in 2008 [1], and proved to be capacity achieving. Polar codes make use of the fact that channels can be polarized, such that some channels become very reliable and others completely unreliable. All information gets sent on reliable channels. Channel polarization is a phenomenon that occurs naturally. Arikan introduced polar codes on binary-input, discrete, memoryless, symmetric channels.

Polar codes were proven to be capacity achieving for infinite block lengths, i.e., they perform well for long data sequences. They are mathematically simple and have a recursive structure with low complexity, which makes them desirable and efficient to implement in hardware [5].

Following, polar codes are explained in detail.

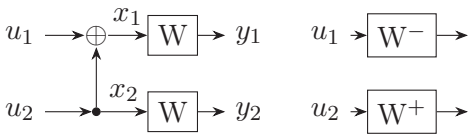

Figure 2.2: The butterfly structure makes up the base of Polar codes. The butterfly structure is shown to the left and the new channel representations to the right.

2.2.1 Introduction to Polar Codes

As the name suggests, and as is mentioned above, polar codes make use of channel polarization. Channel polarization operates on N identical independent copies of a channel W, $\{W_N^{(i)} : 1 \leq i \leq N\}$, to output a second set of N channels where, if N is large, the symmetric capacity $I(W_N^{(i)})$ of close to all channels in the second set, tend towards 0 or 1. The subset of reliable channels is chosen to send data on. All other channels are called frozen and send bits known to both the sender and receiver, usually zeros [1]. An (N, k) polar code uses N channels to send k information bits, and they are sent on the k most reliable channels out of the N possible.

2.2.2 Polarization Phases

Arikan divided channel polarization into two phases: the channel combining phase and the channel splitting phase. The channel combining phase combines two channel copies recursively until N channels are combined. The channel splitting phase splits the combined channel back into N channels, but with changed reliability. Some of the new channels have capacities close to one, some have capacities close to zero, and some are mediocre channels with capacities between one and zero. The fraction of average channels goes to zero as N goes large. That makes it easy to select what channels to send data on, and what channels to choose as frozen. The combined capacity is preserved after polarization.

2.2.3 Butterfly Structure

The butterfly structure in Figure 2.2 lies the base for polar codes and polarization. The figure represents one step in the recursive polarization, polarizing two channel copies. Denoted y_1 and y_2 in the butterfly structure in Figure 2.2 are two copies of the physical channel W and on the left both polarized channels, one with improved reliability and one with inferior. To the right in the figure are the new polarized channel representations.

The encoding step generates x_1 and x_2 as in the first row of Table 2.1. The bit at u_2 gets sent as x_2, and u_1 and u_2 are combined into x_1. The capacities of the two new channels are $I(u_1; y_1, y_2)$ and $I(u_2; y_1, y_2, \hat{u}_1)$.

To understand polarization, consider a BEC with error probability ϵ. First, u_1 is decoded using knowledge from the channel of y_1 and y_2. The decision depends on the bit-addition of the two channel values as in Table 2.1. With a BEC, the decided

Figure 2.2	(1)	(2)
	$x_1 = u_1 \oplus u_2$	$x_2 = u_2$
	$u_1 = x_1 \oplus x_2$	$u_2 = x_1 \oplus u_1$

Table 2.1: Equations representing coding and decoding in Figure 2.2. The first column (1) shows encoding and decoding for the top row in the butterfly structure, and (2) shows the same for the lower part of the figure.

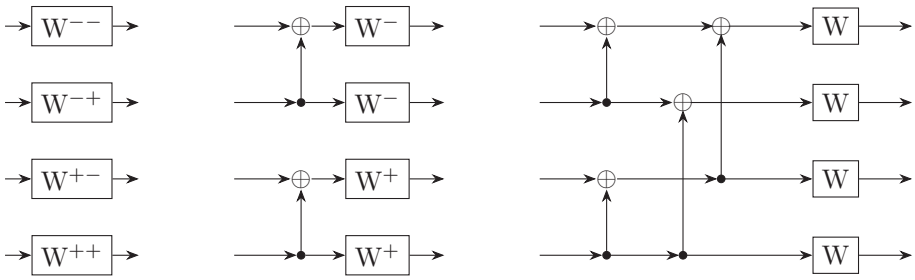

Figure 2.3: The second recursion step of the butterfly structure. The resulting channels of two copies of the same butterfly structure is shown to the left. Two channel copies are combined together in a new butterfly structure, shown in the second part of the figure. To the right is the resulting code structure of the $N = 4$ polar code.

value is correct only if both y_1 and y_2 are correct, hence the error probability is $p = 2\epsilon - \epsilon^2$. The data from u_2 is unknown when u_1 is decided and looked at in the decoder as noise.

After u_1 is decided, the calculations to find u_2 starts. The decision of u_2 is influenced by the decision of u_1. Knowing the value of u_1 hence makes the second part of the butterfly structure a repetition code. Since BEC is considered, the correct value of u_2 is decided if any of the two inputs y_1 and y_2 are correct. The error probability is $p = \epsilon^2$.

It is now easy to see that the first channel has worsened reliability, and that the second is improved. The better channel after polarization is denoted W^+, and the worse W^-. The total channel capacity of the system is conserved since $p(W^-) + p(W^+) = 2\epsilon - \epsilon^2 + \epsilon^2 = 2p(W)$.

Polar codes were introduced using successive cancellation (SC) decoding, which means that one bit is decided before the probabilities for the next bit is calculated. The SC decoder in this case first decides the bit at the worse position u_1, and then at u_2. If u_1 is known to the decoder before decoding, then the probability to decide the correct value at u_2 is improved from the original channel. In polar codes, bad channels are frozen. This means in the mini-case of the butterfly structure that the value at u_1 is known to both the sender and receiver, and only the value at u_2 is unknown before decoding.

A recursive addition of butterfly structures results in more polarized channels. Figure 2.3 shows how the size $N = 4$ polar code is constructed by combining two copies of the butterfly structure. Two copies of the improved channel are combined into a new butterfly structure, and so are two copies of the worsen channel. Together, they make up a polar code structure and four new channels with different reliabilities from the original channel. The figure shows the combined butterfly structure as well as the new channel representations.

Figure 2.4 shows the structure of an $N = 8$ polar code. The input bit sequence u_1^N on the left-hand side is N bits long, of which k bits are information bits, and the other $N - k$ bits are frozen and known to both sender and receiver. In the figure, it is also seen how two copies of polarized channels of the same polarization are always combined to create two new channels.

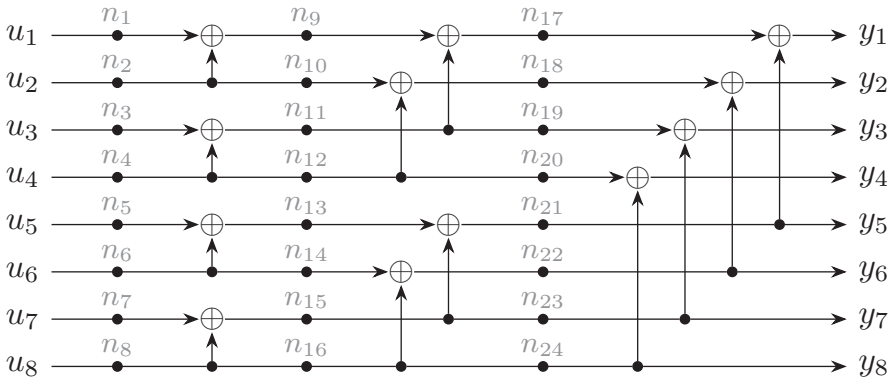

Figure 2.4: Encoding generator matrix structure for the $N = 8$ example.

2.2.4 Frozen Bits and How They Are Selected

The Bhattacharyya parameter was used to select frozen bits in the introduction of polar codes in [1]. Some other frozen bit computations have been suggested and used since then. For the interested, see [8] for a comparison between some different suggested ways of finding frozen bits for polar codes. In this paper, we use the Bhattacharyya parameter when computing channel reliabilities and selecting frozen bits.

The Bhattacharyya parameter measures the reliability of a channel; it is an upper bound of the probability that an error occurs using maximum-likelihood decision when a channel W is used to transmit a single bit. The physical channel has Bhattacharyya parameter

$$Z(W) = e^{-R \cdot Eb/N0} \tag{2.1}$$

and depends on the physical channel that the polar code is designed for.

One butterfly step changes the parameters of both channels involved as in (2.2), with equality in the first equation in the special case of BEC.

$$\begin{cases} Z(W^-) \leq 2Z(W) - Z(W)^2 \\ Z(W^+) = Z(W)^2 \end{cases} \tag{2.2}$$

Iterating these calculations from (2.2) as in Figure 2.5 until the tree has N leaves gives the new channel Bhattacharyya values for all N polarized channels. N channel values are found after $n = \log_2(N)$ iterations. Every step in the tree uses the equations on the current Z-value to find two child node Z-values. The $N - k$ channels with the highest Bhattacharyya values are selected as frozen channels [1]. Frozen bit values are known to both the sender and receiver, and are in this project always set to zero. In this thesis, calculations to decide frozen bits are made with the approximation that $Z(W^-) = 2 \cdot Z(W) - Z(W)^2$ for all channels.

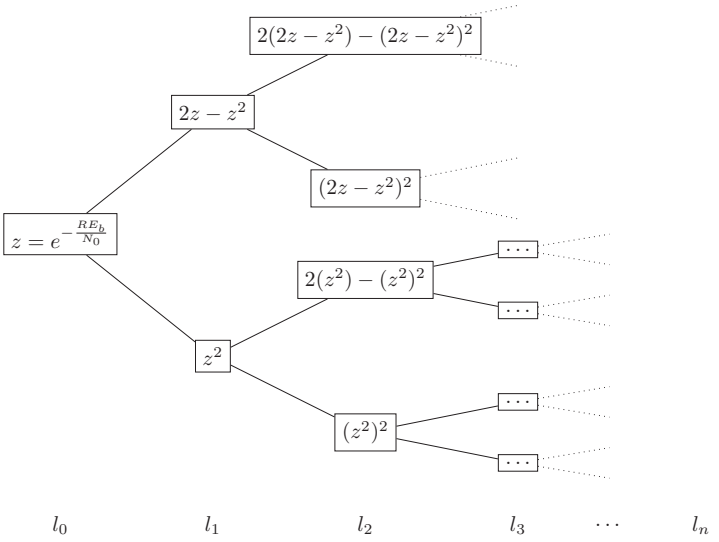

Figure 2.5: Bhattacharyya parameter calculations on polarized channels. The physical channel on level l_0 and all separate polar channels on level l_n, with $n = \log_2(N)$.

Polar codes adapt to channels, so a polar code found for one channel might not be the polar code found for another. This is due to how the Bhattacharyya parameter of the physical channel influence the values of all polarized channels. It is not practical to change the set of frozen bits for a polar code after implementation, hence it is usually optimized for a special design-SNR. The polar code is designed for one channel with some channel parameters, and keep the same set of frozen bits if the channel changes.

The resulting Bhattacharyya parameters of all synthesized channels can be seen in Figure 2.6 for an $N = 128$ code on a channel with design-SNR 1.4186 dB. The choice of design-SNR is made as suggested by Arikan [8], where the initial value in the calculations was $Z = 0.5$. The initial value 0.5 is obtained at 1.4186 dB from equations below in (2.3) and (2.4).

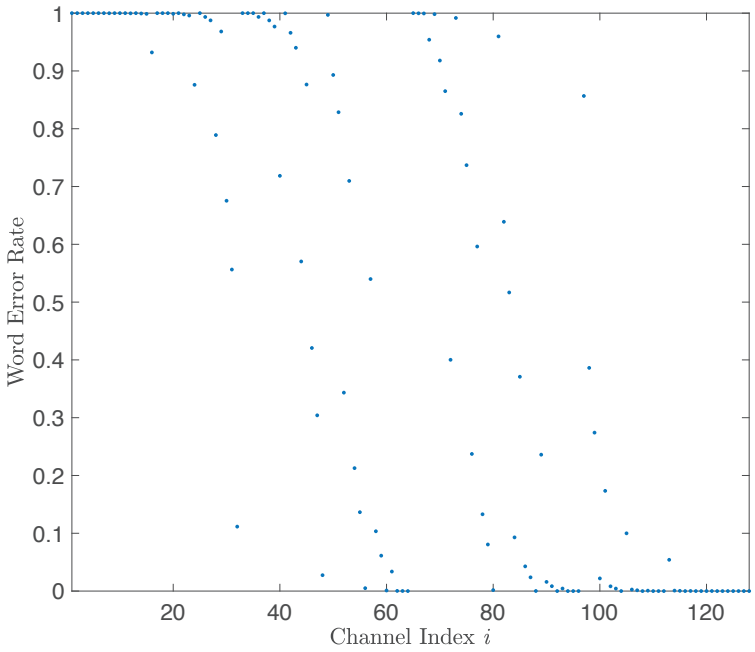

Figure 2.6: Polarization Z-values for all channels, $EbN0 = 1.4186\,\text{dB}$, $N = 128$.

$$designSNR = 10^{\frac{designSNRdB}{10}} \tag{2.3}$$

$$Z_{channel} = e^{-R \cdot designSNR} \tag{2.4}$$

It can be seen in the figure that some channels after polarization are unreliable, some are reliable, and a set of channels are neither. As the size of the polar code construction is increased, the fraction of mediocre channels goes to zero. The Bhattacharyya parameters for a longer polar code with the same design-SNR is shown in Figure 2.7. There, a larger fraction of channels are polarized to values close to 1 or 0. Note that the Bhattacharyya parameter is an upper bound, and some channels might be better than approximated.

Bhattacharyya parameters are usually calculated in log domain to expand the numerical span of results from [0 1]. The design-channel log-domain Bhattacharyya parameter $z(W)$ is calculated as

$$z(W) = -R \cdot \textit{design-SNR}. \tag{2.5}$$

The code rate R is 0.5 throughout this report.

2.2.5 Encoding

Encoding of polar codes can be done in two ways: recursively, using the butterfly structures of the code shown in for example Figure 2.4; or more straightforward

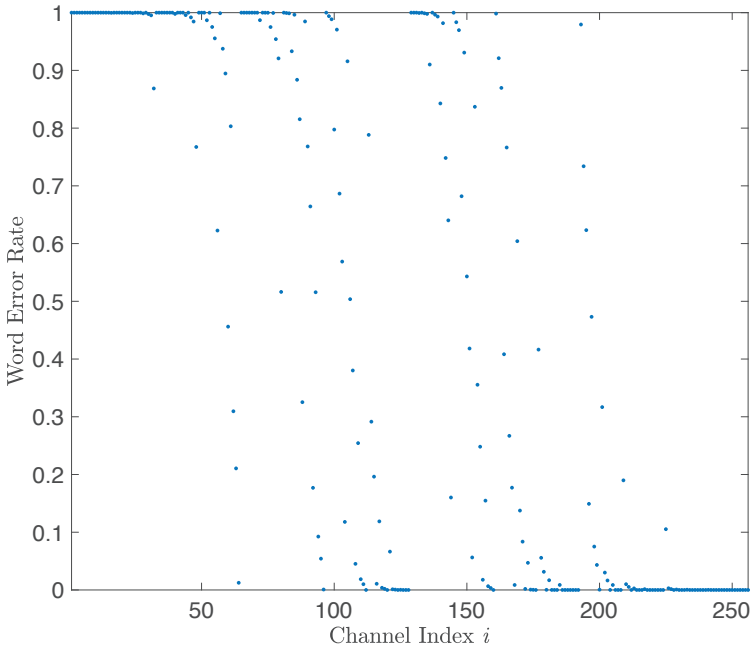

Figure 2.7: Polarization Z-values for all channels, $E_b/N_0 = 1.4186\,\text{dB}$, $N = 256$.

through multiplication with the generator matrix G. For a code with block length N, the generator matrix G is found from the base matrix F by taking the Kronecker power $F^{\otimes n}$, where $n = \lfloor \log_2(N) \rfloor$ and F is defined as

$$F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.6}$$

The Kronecker product $A \otimes B$ and Kronecker power $A^{\otimes N}$ are explained below. Consider $m \times n$ matrix A and $r \times s$ matrix B, then the Kronecker product of A and B is a $mr \times ns$ matrix defined as

$$A \otimes B = \begin{bmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{bmatrix}. \tag{2.7}$$

As an example, see matrices A and B,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \tag{2.8}$$

and their Kronecker product

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}. \tag{2.9}$$

The B-matrix is multiplied with every value of the A-matrix separately, and the result is saved in a new matrix. The Kronecker power is defined as

$$C = A^{\otimes n} = A^{\otimes(n-1)} \otimes A \tag{2.10}$$

with $A^{\otimes 0} \triangleq 1$, and n being an unsigned integer [1]. This means that the generator matrix G is found from the n:th Kronecker power of F

$$G = F^{\otimes n}. \tag{2.11}$$

The encoding of information word $\mathbf{u}$ to codeword $\mathbf{v}$ is simply yielded from the matrix multiplication in 2.12

$$\mathbf{v} = \mathbf{u} \cdot G. \tag{2.12}$$

Frozen bits are set to zero in the vector $\mathbf{u}$ before multiplication with the generator matrix as in (2.12).

After encoding, the codeword $\mathbf{v}$ is sent on the channel.

2.2.6 Decoding

Polar codes were introduced with a successive cancellation (SC) decoding algorithm. Successive cancellation implies that the bits are decoded one at the time in a specified order, here from top to bottom as in Figure 2.4, first deciding $\hat{u}_1$ and last $\hat{u}_N$, using hard decision decoding. One bit decision $\hat{u}_i$ is made before the calculations to find the next bit $\hat{u}_{i+1}$ starts, and already decided bits influence the decision of following bit decisions.

To make one decision, first, all associated likelihood ratios must be calculated. All positions that influence the bit are relevant. For bit u_1 in Figure 2.4, these positions are y_1 to y_8, n_{17} to n_{20}, n_9, n_{10} and n_1. Likelihood ratios on all those nodes are computed in the mentioned order (recursively) before the first decision can be made.

Likelihood ratios (LR) $L(u_i) = \log \frac{p(u_i=0|\mathbf{r})}{p(u_i=1|\mathbf{r})}$ are used to make bit decisions during decoding. They get calculated in two different ways for polar codes depending on decoding node position and are calculated as functions $f(L1, L2)$ or $g(L1, L2)$ as

$$\begin{pmatrix} f(L_1, L_2) \\ \\ g(u_s, L_1, L_2) \end{pmatrix} = \begin{pmatrix} \frac{L_1 L_2 + 1}{L_1 + L_2} \\ \\ L_2 \cdot L_1^{(1-2\cdot u_s)} \end{pmatrix}. \tag{2.13}$$

This can be compared to the butterfly structures in Figure 2.8. As seen in the figure, LR values from the f-function depend only on LR-values of nodes a

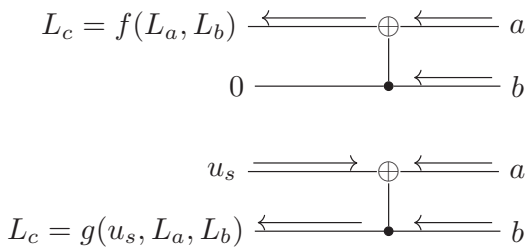

Figure 2.8: Likelihood calculation influences.

and b, and LR values calculated with the g-function are in addition to that also influenced by the result from the f-function. The g-function is calculated as $L_1 \cdot L_2$ if the decided bit on position u_s is zero, or $\frac{L_2}{L_1}$ if the decision of u_s is one.

In general, the decision of any bit $\hat{u}_i$ in u_1^N, $\{1 \leq i \leq N\}$, is influenced by all previous decisions $\hat{u}_1^{i-1}$.

Because of numerical issues in calculating likelihood ratios, all calculations are made in log domain to avoid underflow [24]. Log-likelihood ratios (LLR) of L_1 and L_2 are written as l_1 and l_2, and they are calculated as

$$\begin{pmatrix} \ln(f(e^{l_1}, e^{l_2})) \\ \\ \ln(g(e^{l_1}, e^{l_2}, u)) \end{pmatrix} = \begin{pmatrix} \ln \dfrac{1 + \exp(l_1 + l_2)}{\exp(l_1) + \exp(l_2)} \\ \\ l_2 + (-1)^{u_s} l_1 \end{pmatrix}. \tag{2.14}$$

Log-likelihood ratios on channel values in case of AWGN channels are found as $L_{ch} = \frac{2}{\sigma^2}\mathbf{r}$

The decision made in the f-function decides if the g-function LLR is calculated as $l2 + l1$ or $l2 - l1$. The f-function is often referred to as the box-plus operation and is also used in LDPC decoding. The function can be approximated to $\ln(f(e^{l1}, e^{l2})) \approx \text{sign}(l1) \cdot \text{sign}(l2) \cdot \min(|l1|, |l2|)$ for faster calculations, but this approximation is not used in this project.

The estimate $\hat{u}_i$ is u_i if u_i is frozen, or decided as h_i in the decision function defined as

$$h_i = \begin{cases} 0, & \text{if } \dfrac{W_N^{(i)}(y, \hat{u}|0)}{W_N^{(i)}(y, \hat{u}|1)} \geq 1 \\ 1, & \text{otherwise} \end{cases} \tag{2.15}$$

After a hard decision is made, that value is saved together with all decided node bit values. After deciding nodes $\hat{u}_1$ to $\hat{u}_4$, bit decisions at nodes n_1 to n_4, n_9 to n_{12}, n_{17} and n_{18} in Figure 2.4 are found and saved. These values are used in all future g-function calculations in the decoder [1].

2.2.7 Decoding Example

To show how decoding is done, this section describes the SC decoder in a step-by-step example. Consider the $(8, 4)$ polar code in Figure 2.4. Frozen bits are u_1, u_2, u_3 and u_5, found with the use of Bhattacharyya parameters. Known at the start of decoding are only the channel values and their LLRs are calculated as described in previous section, in the figure LLRs of y_1^N.

The SC decoder starts by making a hard decision of the bit at $\hat{u}_1$, after calculating the LLR value at node n_1. The LLR value at node n_1 depends on all channel values in the channel vector y_1^N, and LLR-values at positions n_{17}, n_{18}, n_{19}, n_{20}, n_9, n_{10}, and n_1. These are all calculated in the mentioned order, using the f-function from (2.13) on previous level values, due to upper node positions. Since the node u_1 is frozen, the decision $\hat{u}_i$ is zero. The LLR-value of node n_1 is found even though u_1 is frozen because of implementation simplicity.

The LLR-value at n_2 depends on LLR-values of nodes n_9 and n_{10}, both already calculated, and $\hat{u}_1$, and is found using the g-function. The decision is again made from being in a frozen position. Here we can see that we only need to update nodes from level 1, level 0 being decision positions, and level 3 being channel values.

When looking at the whole decoding algorithm, there is a pattern in how many levels of nodes that need to be updated at each decoding step. The number of levels that needs updating is found from the bit-representation of the decision-node in u, numbering the nodes from zero to $N - 1$. The first position in the bit-representation being a one, from the least significant bit (LSB), is how many steps needs updating. For example, the node just calculated is represented as [001], the first 1 being at position one from LSB. We, therefore, know that we only need to update from level one in decoding implementation. We also know that LLRs of the deepest level to be calculated every iteration is found with the g-function.

The third bit to decide is $\hat{u}_3$ in the figure, bit represented by [010]. The second bit from LSB being the first 1, so we know that we need to update LLRs from level 2. We calculate LLRs at nodes n_{11} and n_{12} using the g-function with decided values for n_9 and n_{10} being known, and last n_3 with the f-function. The frozen decision is made.

This decision order continues. The LLR of n_4 is decided using the g-function, and the decision value is decided using (2.15). We have now hard decision values for nodes n_{1-4}, n_{9-12} and n_{17-20}, and the later will be used in the next g-functions, starting calculations for the decision of $\hat{u}_5$, the last frozen bit.

In the decision of $\hat{u}_6$, we find the LLR-value of n_6 by $\ln(g(e^{l_1}, e^{l_2}, u))$, using l-values of nodes n_{13-14}, and $\hat{u}_5$. The decision of $\hat{u}_6$ is again made with decision cases in 2.15. Finding the last two decision values is a repetition of finding LLR-values for $\hat{u}_{3-4}$, only decisions are made from those calculated values since they are not frozen.

After decoding, we find the 4 bit decided information message to be $[\hat{u}_4, \hat{u}_6, \hat{u}_7, \hat{u}_8]$.

2.2.8 List Decoding

The SC decoder is very straightforward, easy to implement and fast. However, it has some limitations, since we only get competitive performance for unreasonably

long polar codes using this decoder. Successive cancellation list decoding together with CRC code has been found to improve the polar code and make it perform well for shorter code lengths [6].

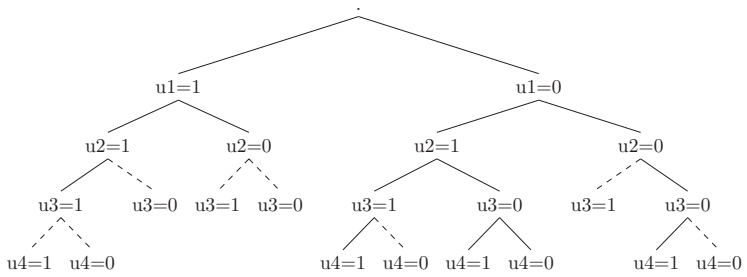

Figure 2.9: List decoding, $L = 4$ example. Dotted lines represent paths with low probability that are erased.

List decoding is a well-known concept that has been used for other channel codes before polar codes and means that we consider that one early decision error could have significant consequences for the rest of the decoding process. For polar codes, if one bit is incorrectly decided, this cannot be corrected later in the SC decoder and could influence the decoder to make later bit errors.

Instead of only saving the most likely path after every decision level, a list decoder saves all possible paths, together with their calculated likelihood. At the end of decoding, the probabilities of all paths in the list are compared, and the most likely path, in the end, is decided. The list decoder is expensive; it increases the decoding complexity from $O(n \log n)$ for SC to $O(Ln \log n)$ for optimized SCL decoding, L being the list size. A straightforward implementation of the SCL decoder takes $O(Ln^2)$ [5].

For a better trade-off between complexity and performance, a fixed number of most likely paths can be saved in the list, instead of every possible path. Figure 2.9 shows some early steps of a list decoder with list size L=4. The first step saves all possible paths, since the list is not filled, and so does the second step. From step 3 and forward, probabilities of all different paths are calculated as a sum of likelihoods for each step, but only the four most likely paths are saved in the next list step. The dotted lines in the figure represent the least likely paths that are not saved, and the full lines represent all paths saved in the list for that step. Children of a node represent the continuing of a path with either a decided zero or one.

2.2.9 Cyclic Redundancy Check and List Decoding

Cyclic Redundancy Check (CRC) codes are simple channel codes that can detect, but not correct errors. Redundant bits, found through the polynomial division of the code word with a predefined generator polynomial, are added to the end of the message as a check value before transmission. The decoder receives the code word including redundant bits, and perform the same polynomial operations as the encoder. If the calculated check value is the same as in the received word, the code word gets accepted, otherwise usually discarded.

In polar codes with list decoding, CRC comes in handy in the last step of decoding. The L most likely paths get saved in a list during decoding, and in the last decoding step, the most likely path out of those L paths gets selected as the decoded codeword. It was found in [5] when errors occurred in polar codes with list decoding, that the correct codeword was often in the final list, but that it was not the codeword with the highest likelihood and hence not selected in the last step of the decoder. With a CRC added to the codeword, the CRC check could be used to make the decoder choose the correct codeword out of the list. If a word in the list in the end of decoding has higher calculated probability than the real codeword, but is not a valid codeword according to the CRC check, it gets discarded, and the most likely codeword that is valid with its CRC gets selected instead. This was shown to improve error rates for polar codes in [5]. Note that adding CRCs to polar codes changes the polar code rate, but not the total code rate.

The selection of generator polynomials plays a big part in maximizing the CRC performance. The ability to find long burst errors depend on it and many studies have been made trying to find the best polynomials to use. A longer polynomial can in general correct longer burst errors, but has a higher cost of code rate. Many polynomials that work well for some code length do not work well for other [13]. In use with polar codes, it is not clear if long burst errors, that CRCs are designed to correct, are the types of errors that occur for the polar code, or if polar code errors are spread out over the word.

2.3 Reed-Muller Codes

The Reed-Muller (RM) codes were first invented by Reed [21] and Muller [22] in 1954. They were some of the first used channel codes, and much research is available for the interested, see for example [12, 23] on improved RM decoders. The codes were not proven to reach capacity until recently and are not used a lot in practice. They are similar to polar codes construction-wise.

The encoding of Reed-Muller codes can be done through multiplication with the same encoding generator matrix as for polar codes, only differing in the choice of frozen bits. Where polar codes base the selection of frozen bits on channel parameters, for example with the use of the Bhattacharyya parameter, Reed-Muller codes instead choose the rows in the same generator matrix with the lowest Hamming weights as frozen. A generator matrix for size 8 Reed-Muller code and Hamming weights of each row can be seen in (2.16).

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} d_h = 1 \\ d_h = 2 \\ d_h = 2 \\ d_h = 4 \\ d_h = 2 \\ d_h = 4 \\ d_h = 4 \\ d_h = 8 \end{matrix} \tag{2.16}$$

Reed-Muller codes were originally defined to freeze all rows of up to a certain Hamming weight. They were hence only defined for a few different code rates. With for example the code of length N with the generator matrix (2.16), the rate $R = 5/8$ RM code is not defined.

In this thesis we need to freeze bits of other code rates than those originally defined for Reed-Muller codes, to keep the code rate to $R = 0.5$ after CRCs are added. We call all codes where the decision of frozen rows depends on the Hamming weight for Reed-Muller codes. This means for example that there are three different possible (8, 5) RM codes, since there are three rows in the generator matrix with Hamming weight $d_h = 4$.

RM and polar codes have the same frozen bits for code size smaller than 32 and rate 0.5 [2]. The different frozen bit selections for longer codes make RM code stronger with a larger minimum distance between codewords. However, RM codes are channel-independent while polar codes pay attention to channel polarization and work better with SC decoding [1]. RM codes, in contrast to polar codes, get weaker for increasing block lengths [20].

Reed decoding, which was used when RM codes were first invented, is based on the fact that every possible codeword is a linear combination of the rows in the generator matrix G, and one of the rows is the all-one vector. Hence, the codeword is $v = a_0v_0 + a_1v_1 + .. + a_kv_k$ and the received vector $r = (y_0..y_N)$ and decoding is done with simple linear computation. Some linear combination of active rows in the generator matrix is closest to the received codeword bitwise. Since the invention of RM code, many different decoders have been used. In fact, both SC decoding and list decoding were introduced for RM code long before polar codes were introduced [1].

2.3.1 Reed Muller Codes with CRC

When Reed-Muller codes are investigated with list decoding and CRC, it is important to specify what bits to freeze when several matrix rows in the generator matrix have the same Hamming weight. Several different frozen bit selections can be used to create for example a (128,71) RM code. These various codes might show different performances.

Chapter 3

Polar Codes and Reed Muller Codes for Short Block Lengths

In this chapter, some results found in the previous research of polar codes (PC) and Reed-Muller (RM) codes are re-investigated for short 128-bit block lengths. First, the performances of polar codes designed with different design-SNRs are compared. Simulation results are used to decide the design-SNR to be used in all following investigations of polar codes. The performance results found from using different design-SNRs for the (128, 64) PC are compared with the same design-SNR investigation of the (1024, 512) PC.

Later, RM and polar code performances are compared, first under SC decoding and later with the SCL decoder.

The basic polar code was implemented in Matlab with some inspiration from the open source code found in [24] and expanded for this project. Simulation results are presented as word error rate (WER) and bit error rate (BER) performances as a functions of channel-SNRs on AWGN channels.

3.1 Polar Code Design-SNR

The polar code design suggested in [1] varies for different channel-SNRs. Modified channel parameters lead to a changed set of frozen bits.

To change the frozen bit set of an implemented polar code is known to be unpractical. A more practical approach for implementation is to choose a polar code that works for a range of channels, so that the polar code still performs well when a channel changes slightly. Therefore, the investigation of polar codes started with a comparison of polar codes constructed with different design-SNRs over a changing channel. The design-SNR of the best performing polar code for the range of channels was then selected as the design-SNR to use in the rest of the project.

The choice of design-SNRs to compare was motivated by results in [8], and chosen so that design-SNRs were close to the channel-SNRs that were simulated for. Results are shown in Figure 3.1.

It can be seen in the figure how a polar code with a certain design-SNR is not necessarily the polar code that performs best for that channel-SNR. Examples of this are found at channel-SNR 4 dB and 5 dB in the figure. The polar code

designed for the 5 dB channel works better than that for the 4 dB channel when the channel-SNR is 4 dB. We conclude that it is not trivial to find the best (128,64) polar code for a known channel.

The results suggest that all lower design-SNRs, 3 dB to 5 dB, work well for low SNR channels, but that 5 dB gives the most reliable code out of them for the span of channels that are simulated over. Design-SNRs 6 dB and 7 dB perform worse for most of the channel span. However, they catch up with the better codes around 5 dB.

Since the polar code designed for the 5 dB channel worked well for the whole span, 5 dB was selected as design-SNR to be used when designing polar codes for all further polar code investigations in this project.

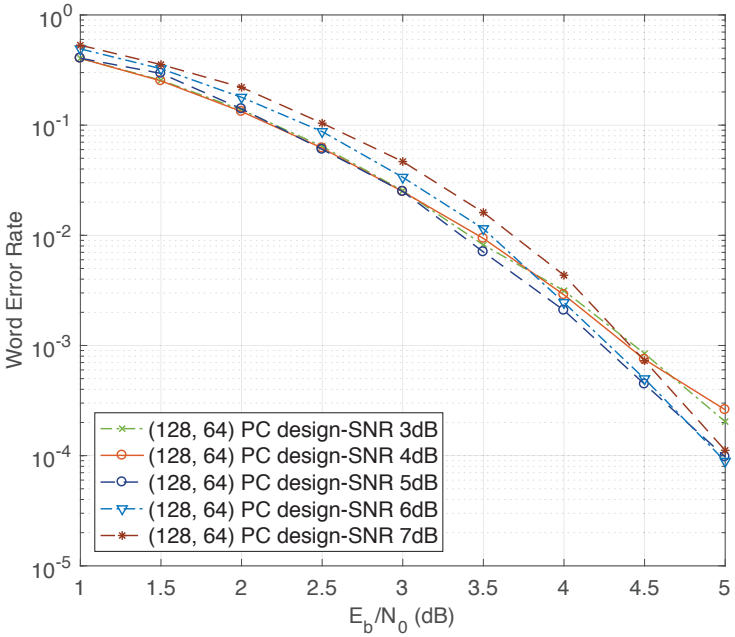

Figure 3.1: (128, 64) Polar code performance for some different design-SNRs.

3.2 Design-SNR for a Longer Polar Code

To control if results from the previous section were consistent with other short polar codes, and not a special case for the (128,64) polar code, the design-SNR test was complemented with a run for the longer (1024, 512) polar code, with resulting word error rates shown in Figure 3.2.

Just as for the shorter code can we observe that the polar code designed for 4 dB channels is not the best for the channel of that SNR. The polar code that adapts for every new channel-SNR does not work well compared to the other codes with fixed design-SNRs. Conclusions from the previous section stand for

both tested short polar codes. It is not trivial to find the best short polar code for a channel, given a channel parameter, and the selection of frozen bits is essential for polar code performances.

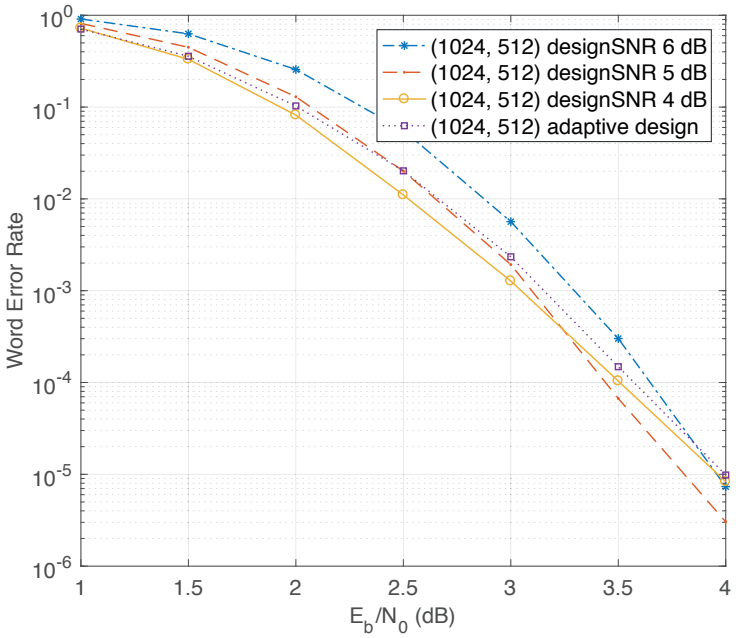

Figure 3.2: (1024, 512) Polar code performance for some different design-SNRs, and for the adaptive polar code.

3.3 Reed-Muller Code and Polar Code

Previous results show that the design-SNR changes the polar code performance, which means that the selection of frozen bits is vital. The different set of frozen bits for the Reed-Muller code has been shown to have worse performance than the polar code [2]. This section re-investigates these results by comparing the RM set of frozen bits with the previously selected polar code design.

The (128,64) Reed-Muller and (128,64) polar code with design-SNR 5 dB were compared under SC decoding, and the resulting WER and BER are shown in Figures 3.3 and 3.4. The polar code shows better results under the SC decoder than the RM code, as expected and found in previous research. This can be explained by that even though the RM code is stronger, the polar code is designed to work well with the SC decoder and therefore work better in this case. The strong RM code needs a stronger decoder to show good performance.

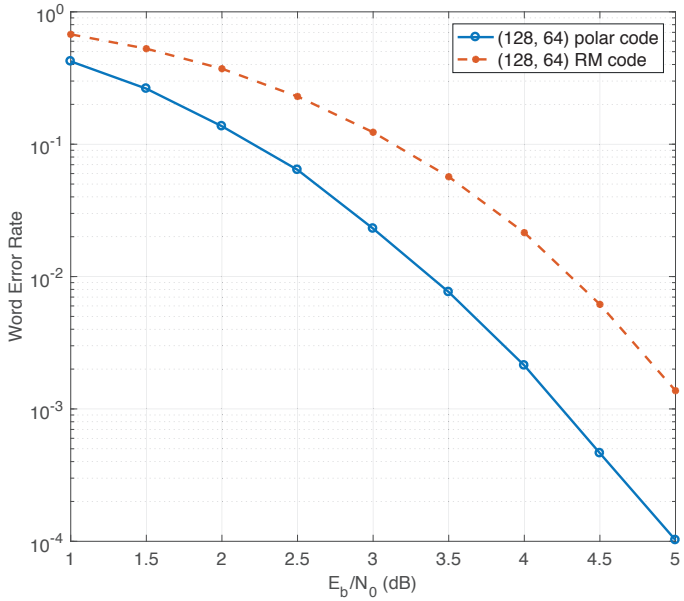

Figure 3.3: WER of polar code and Reed-Muller code under SC decoding. Polar code frozen bits are selected with Bhattacharya parameters with design-SNR $5\,\mathrm{dB}$.

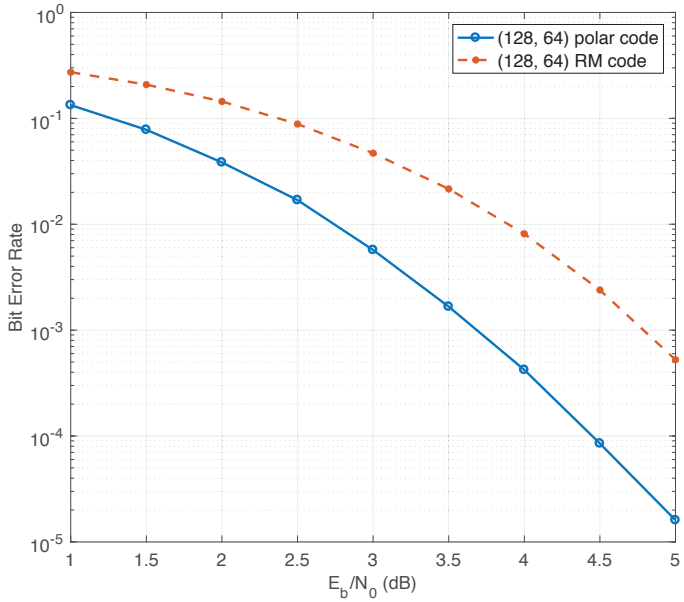

Figure 3.4: BER of polar code and Reed-Muller code under SC decoding. Polar code frozen bits are selected with Bhattacharya parameters with design-SNR $5\,\mathrm{dB}$.

3.4 Polar Code with SC List Decoding

Polar codes with $N = 2048$ have shown to perform slightly better under successive cancellation list decoding than SC decoding [5].

The motivation of adding a list decoder is to go around the negative aspects of a hard decision decoder. When a false decision occurs in the decoder, the other correct path is saved too, and we rely on that the final correct codeword has a higher likelihood at the end of the decoder so that it gets picked over the incorrect path.

The performances of the (128, 64) polar code with SCL decoding of different list sizes are shown in Figures 3.5 and 3.6. We observe that list decoding improves the code performance, but that a larger list size does not show significant performance gain compared to a smaller. The longer list size is expensive, and the result does not motivate the use of a long list size in applications.

The fact that a longer list size does not improve the polar code mush is a consequence of that the decoder does not select the correct codes at the end of decoding, which has been previously observed in [5].

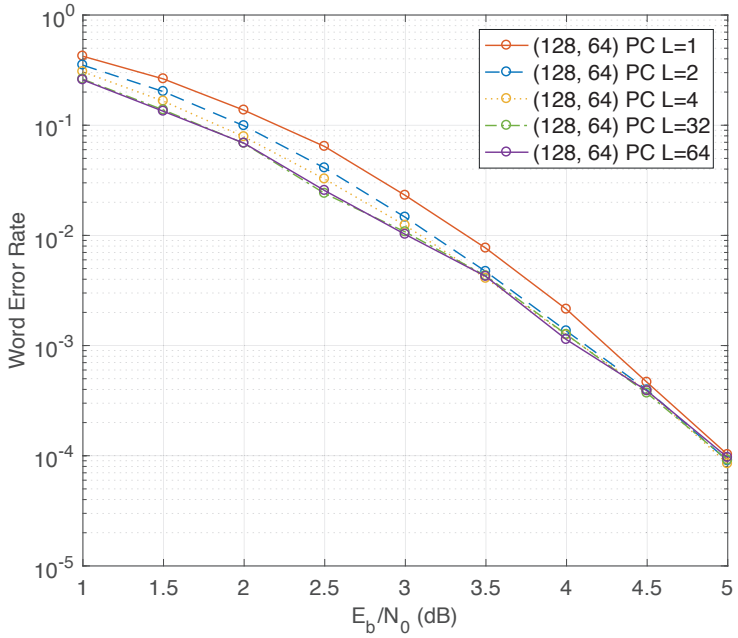

Figure 3.5: WER of (128,64) polar code with list decoding, varying list sizes L.

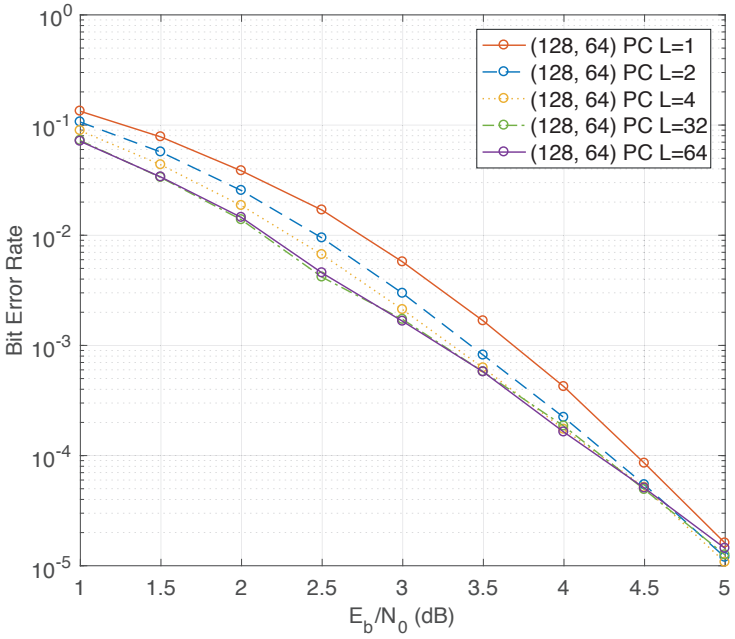

Figure 3.6: BER of (128,64) polar code with list decoding, varying list sizes L.

3.5 Reed-Muller Code with SC List Decoding

SCL decoding of the (128, 64) RM code was investigated for different list sizes, with results found in Figures 3.7 and 3.8.

The Reed-Muller code shows a much greater performance improvement than the polar code from the improved, more complex decoder. Greater list sizes improve the performance, but less per list size step as the performance gets closer to the ML performance. Only a short list is needed to make RM code comparable with the same size polar codes, and longer list results in the RM code outperforming the polar code. Results comparing RM and polar codes with and without list decoding are shown in Figures 3.9 and 3.10. It can be seen how the RM code performs better after list decoding is added.

Another observation is that the difference in performance between the two codes under list decoding is larger for high SNRs than for low. This can be explained by that the benefits of a strong code in combination with a good decoder always can be observed at high SNRs, but not always at low.

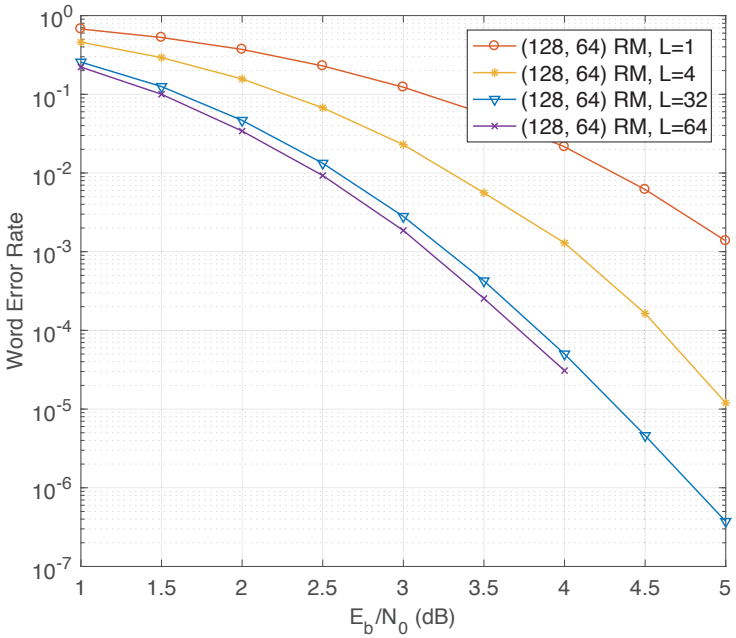

Figure 3.7: WER of (128,64) RM with list decoding, varying list sizes L.

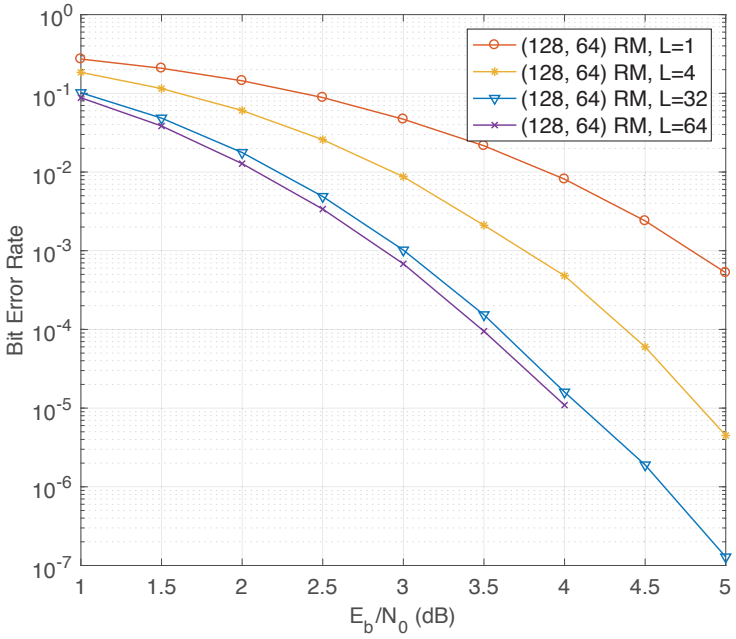

Figure 3.8: BER of (128,64) RM with list decoding, varying list sizes L.

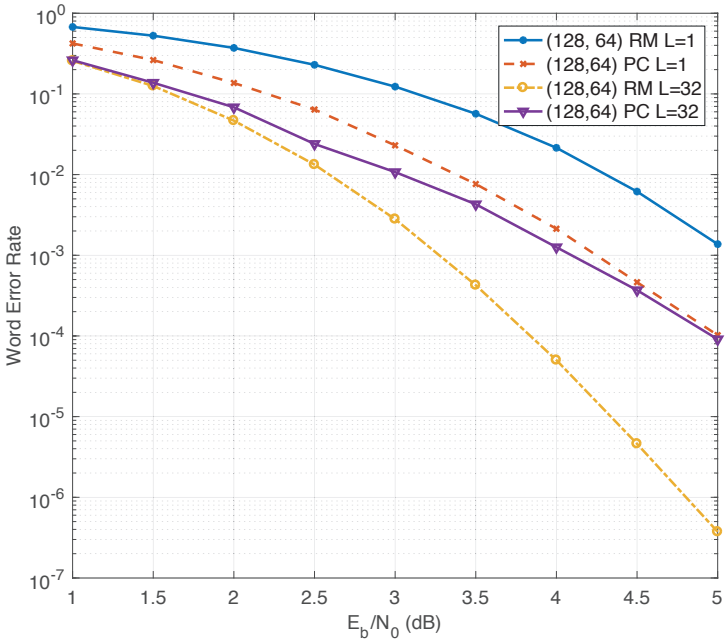

Figure 3.9: WER comparison of (128,64) polar and RM codes under and SC and SCL decoding.

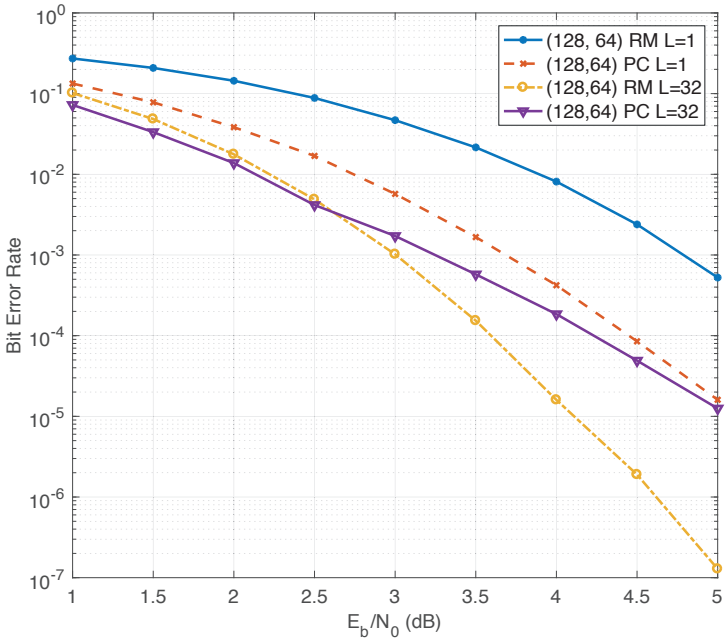

Figure 3.10: BER comparison of (128,64) polar and RM codes under and SC and SCL decoding.

3.6 Chapter Summary

The polar code design is not uniquely defined, and a small change in selected frozen bits, for example as a result of a changed design-SNR, can have significant impacts on the code performance. A polar code designed with Bhattacharyya parameters for one channel might not be the best code for that channel, but can work better for a different channel.

Polar codes perform better than RM codes using SC decoding, but the RM code gains more from the use of list decoding, and perform better than polar codes for long list sizes.

Chapter 4

Polar and Reed Muller Codes with List Decoding and CRC

Previous research has shown that short polar codes with list decoding and CRC can perform better than both turbo codes and LDPC codes of the same length and rate. This chapter re-investigates the polar code performance with list decoding and CRC and different sets of frozen bits, and compare them to Reed-Muller codes with the same combination.

We look at how an added CRC changes the performances of first the polar codes, and later the RM codes. The resulting codes are compared, and so are the performances of polar codes with some different CRC sizes.

4.1 Adding CRC

From research suggesting the use of CRC with list decoding for polar codes, it is not clear what type of errors that the CRC was added to correct. Do polar codes have problems with long burst errors or spread out short errors? CRC codes are built and optimized for burst errors, but it is not a certainty that a CRC polynomial that finds the longest burst errors is the best to use for polar codes.

Some selected CRC generator polynomials for this project are listed in Table 4.1. All generator polynomials were found from [17] and [18], where good and commonly used CRC polynomials are listed. Two different CRC polynomials are compared for CRC-4 and CRC-8, to see the effect of the polynomial selection for polar codes.

4.2 Polar Code with List Decoding and CRC-7

Polar codes with list decoding and CRC have shown superior performance for some short block lengths and channels. The (128,71) polar code with list decoding and CRC performed well in [7], so we use the same parameters to investigate the polar code presented in this thesis. The result from the (128,71) polar code designed with design-SNR 5 dB, frozen bits found with the Bhattacharyya parameter, and CRC-7 from Table 4.1, is shown in Figures 4.1 and 4.2, compared with polar code results from the previous chapter. Observed is that the added CRC improves the

Name	*Type*	*CRC*	*GeneratorPolynomial*
p1	PP	CRC-1	$x+1$
p2	PP	CRC-2	x^2+x+1
p4a	PP	CRC-4	x^4+x+1
p4b	-	CRC-4	$x^4+x^3+x^2+x+1$
p5	PP	CRC-5	x^5+x^2+1
p7	PP	CRC-7	$x^7+x^6+x^3+x+1$
p8a	PP	CRC-8	$x^8+x^4+x^3+x^2+1$
p8b	-	CRC-8	$x^8+x^7+x^6+x^4+x^2+1$

Table 4.1: List of generator polynomials used for CRCs in this project. Type PP stand for primitive polynomials. Two commonly used not primitive CRC polynomials for CRC-4 nd CRC-8 are used for comparison. The polynomial names will be used to reference to polynomials further in the text.

performance and make the polar code a stronger code, working better for high SNRs.

The combined polar code in [7] used a different way of calculating channel estimates, resulting in slightly changed selections of frozen bits. The used CRC-7 polynomial in that paper was also another. The resulting combined polar codes in [7] showed a somewhat better performance than the (128, 71) polar code in Figures 4.1 and 4.2.

4.3 Reed-Muller Code with List Decoding and CRC

The otherwise uniquely specified RM codes are modified to be combined with the CRC without changing the code rate. The selection of frozen bits for Reed-Muller codes with added CRC is not uniquely specified for these rates. This happens when many rows in the generator matrix have the same Hamming weight, but only a few of them are selected to send data. In the case of finding a (128, 71) RM code, there are 36 rows in the generator matrix with Hamming weight 8, but only 7 of them are needed to send data.

It is known from polar code results in the previous chapter that a small change of frozen bits can change the code performance. To see how much some different frozen bit selections affect the performance, two (128,71) RM codes with CRC-7 were compared under L-32 list decoding. One code selected the seven first rows to send data on, from the range of rows with the same Hamming weight in the generator matrix. The other code instead chose the seven last rows with the same Hamming weight. From Figure 2.6 and Figure 2.7, showing the distribution of calculated Z-values over all channels, it can be seen how earlier channels are more likely to be unreliable, and late channels are more likely to be reliable. It is therefore presumed that the RM code with an early selection of frozen bits could perform better than the RM code with later frozen bits.

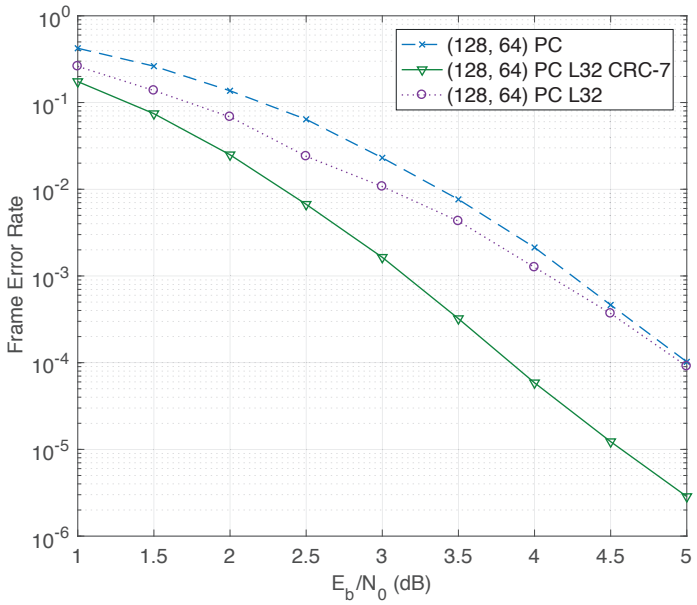

Figure 4.1: Word error rates of polar codes with list decoding and CRC, compared with the performance of polar codes under SC decoding and list decoding. $N = 128$, $L = 32$ and *CRC-7*.

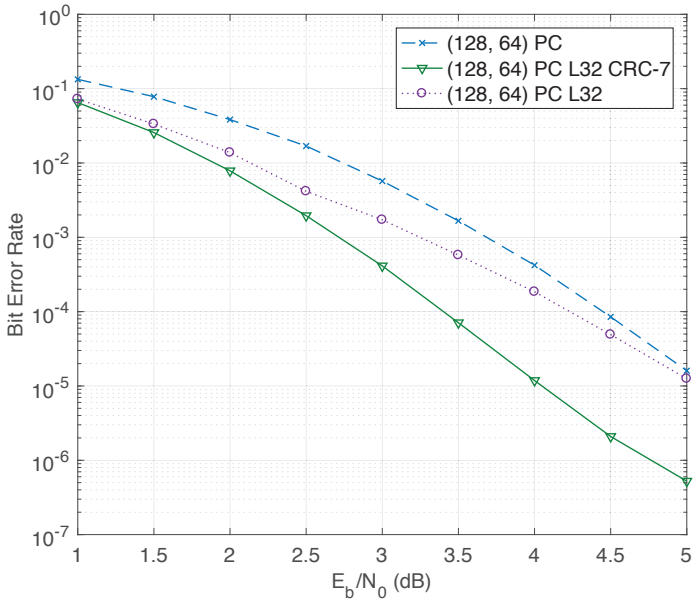

Figure 4.2: Bit error rates of polar codes with list decoding and CRC, compared with the performance of polar codes under SC decoding and list decoding. $N = 128$, $L = 32$ and *CRC-7*.

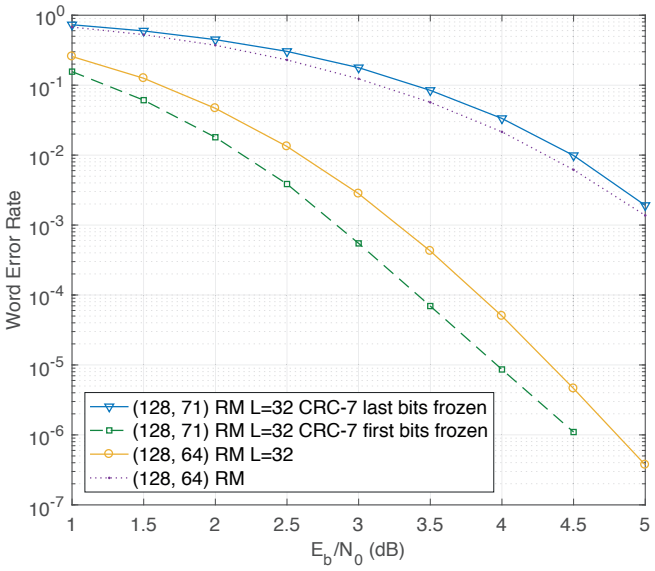

Figure 4.3: Word error rates of RM codes with list decoding and CRC, compared with the performance of RM codes under SC decoding and list decoding. Two RM codes with different sets of frozen bits are compared.

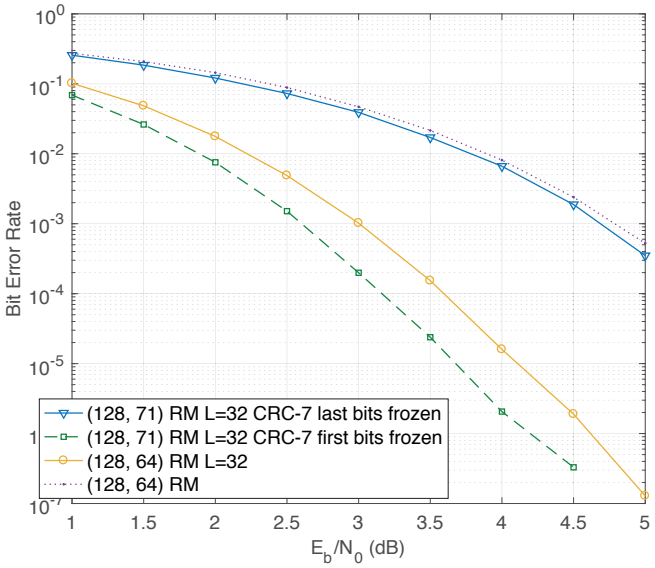

Figure 4.4: Bit Error Rates of RM codes with list decoding and CRC, compared with the performance of RM codes under SC decoding and list decoding. Two RM codes with different sets of frozen bits are compared.

The result comparing the two RM codes under list decoding with CRC is shown in Figure 4.3 for word error rates and Figure 4.4 for bit error rates. The figures also include the performance of the (128,64) RM code with and without list decoding.

It is easy to see that the frozen bit selection changes the code performance dramatically. As per predicted did the RM code with data sent on later channels perform better than RM code sending data on earlier channels. The better RM code improved the code performance with CRC from that with only list decoding, but the bad RM code instead worsened the performance. The second (128, 71) RM code got higher frame error rates than the (128, 64) RM code under SC decoding. The frozen bit selection again shows to be of great importance for these codes.

4.4 Compared Results of RM and Polar Codes with CRC

The better performing RM code and the polar code with list decoding and CRC are compared in Figure 4.5. From the figure, we can observe that the RM code still outperforms the polar code after adding CRC under long list decoding. However, as previously mentioned, this polar code shows a worse performance than the polar code with the same parameters in previous research [7]. That combined polar code performed better than the RM code in Figure 4.5.

It is not only the selection of frozen bits for the polar code that can change. The set of frozen bits for the RM code is chosen arbitrarily from the channel position, and could also be improved. We can therefore not conclude from results in Figure 4.5 which of a polar code or an RM code would work better if optimized under these circumstances. We can, however, conclude that these results are exciting and that further research is necessary.

4.5 CRC Lengths, Polar Code

So far, all test performed with list decoding and CRC are simulated for (128, 71) codes with list size 32. Here, that combination is compared with other polar codes of different list sizes and CRC combinations. Results are shown in Figures 4.6 and 4.7. We can observe that the (128,71) polar code performs well compared to some other CRC-list combinations.

We observe that the shortest list size resulted in the worst performance. A correct codeword has a higher chance of making it to the final list in the decoder if the list is long. The added CRC can only help the decoder to choose the correct codeword if it is in the final list.

The longer lists perform better with CRC-7 than with both CRC-4 and CRC-8. The CRC-4 check seems to be too short to find the correct codeword, and the CRC-8 could be too long. Different CRC sizes changes the polar code rate. The more bits allocated for the CRC, the worse does the actual polar code perform, since data is sent on more channels, and the polar code needs to select previously frozen channels to transmit data on. A long CRC can decrease the polar code performance more than it improves the combined code.

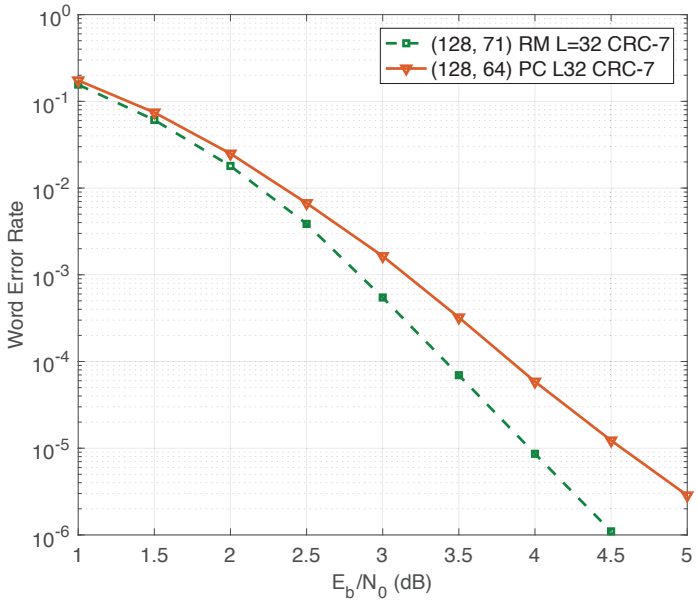

Figure 4.5: WER Performance comparison of (128, 71) polar code and Reed Muller code with CRC-7 and list decoding.

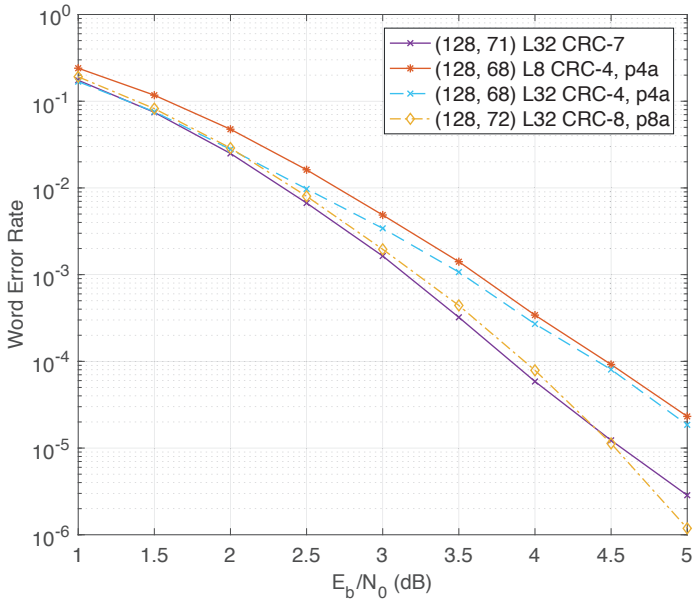

Figure 4.6: WER $N = 128$ polar codes with different list size and CRC combinations.

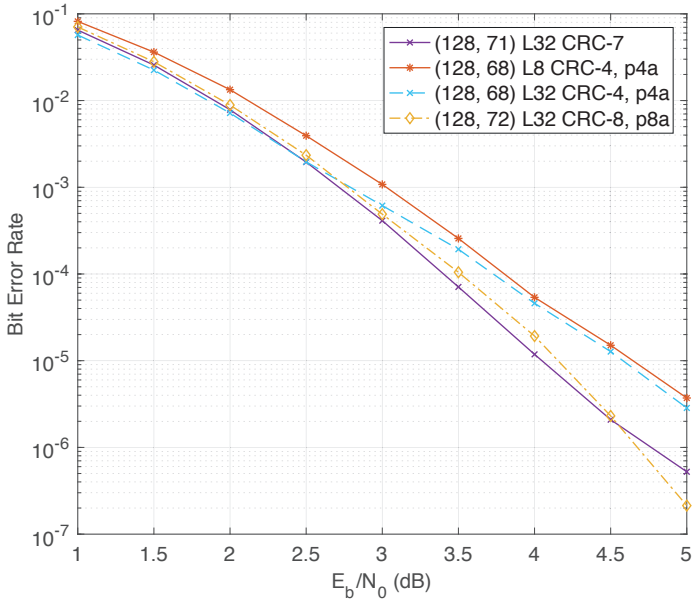

Figure 4.7: BER $N = 128$ polar codes with different list size and CRC combinations.

4.6 Chapter Summary

We have looked at the performance gain of polar codes and RM codes, when in combination with list decoding and CRC. Both polar and RM codes can be improved with the add of a CRC to the end of the code, compared to using list decoding only. The CRC must be of a good length to enhance the code to a maximum. It should not be too long nor too short. The selected frozen bits for the changed rate RM code also has to be optimized.

The combined (128, 71) RM code performs better than the polar code of the same size. This does not have to mean that every RM code of this size perform better than polar codes, but it is an interesting observation for these codes and list lengths.

Chapter 5

List Decoding and CRC, Separated CRC and Changed CRC Positioning

Cyclic redundancy check codes are usually implemented with the check sequence added to the end of the data. A longer CRC polynomial adds a longer check data sequence to the code and can detect longer burst errors in the message (provided that a suitable polynomial is used). This gives an outcome that is typically preferred when CRC is used and maximizes the performance gain of every added bit to the message. Previous researchers interested in how to select good CRC polynomials have tried to optimize CRC such that they can detect the longest possible burst errors.

Adding CRC to the end of the message in polar codes improved the code performance under list decoding. The CRC is added to aid the decoder in the step where a codeword is decided from a list. It was found in [5] to improve the polar code performance dramatically for some CRCs. However, the correct codeword was not always selected at the end of the decoder even after this aid was added. We found that this in most cases was not because the CRC had approved incorrect messages, but since the correct codeword had not made it to the final list of possible codewords in the last decoder step. A more extended list could solve this problem, but would also increase the unwanted decoding complexity.

Instead of adding a longer list, we investigated if it is possible to gain the same improved performance by changing the CRC structure. Since we are not sure what type of errors it is that normally occur for polar codes, we do not know if the CRC is aiding the decoder in finding long burst errors or short spread out errors. Therefore, we do not know if the longer CRC, in the end, is the best option, or for example, a few short spread out over the decoder. An earlier elimination of incorrect paths in the decoder could maximize the gain from a shorter list size with polar codes since it would minimize the risk that the correct codeword gets incorrectly discarded in an early decoding step.

In this chapter, we try to find out if there is a simple way to improve the CRC, either by dividing a longer CRC into shorter spread out, to discard incorrect paths earlier, or by moving the whole CRC to an earlier position in the code. We also compare the use of different CRC generator polynomials, to see how much the polynomial selection could influence the combined code performance.

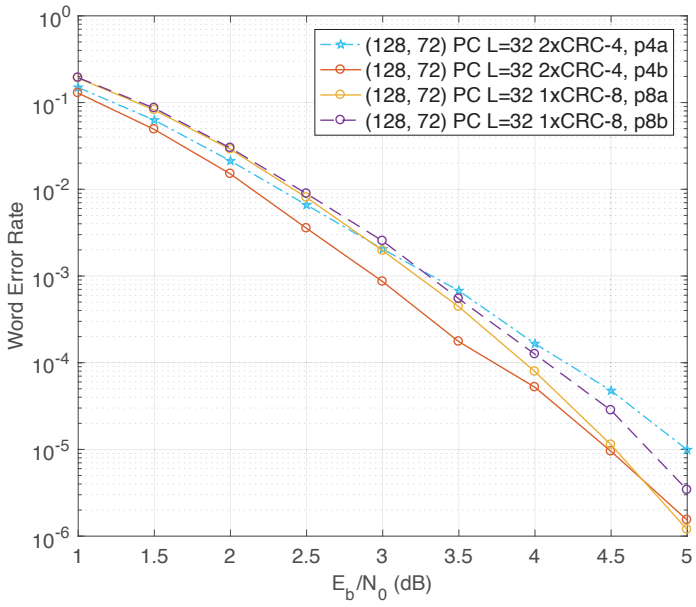

Figure 5.1: WER of polar codes with different CRC polynomial selections.

5.1 (128,72) Polar Code Performances Depending on the CRC Polynomial

The first test compares different CRC polynomials for a (128,72) polar code under list decoding. It is interesting to see if the selection of generator polynomial affects the final result significantly. Compared are the two CRC-4 and two CRC-8 polynomials. The CRC-4 polynomials are divided, as will be described in detail in the next section. In short, two CRC-4s with the same generator polynomial are added to the message in different locations before polar encoding. Simulation results are presented in Figures 5.1 and 5.2.

We observe that the polar code performance changes for both CRC lengths depending on generator polynomials. Most significant is the difference between the two CRC-4 polynomials.

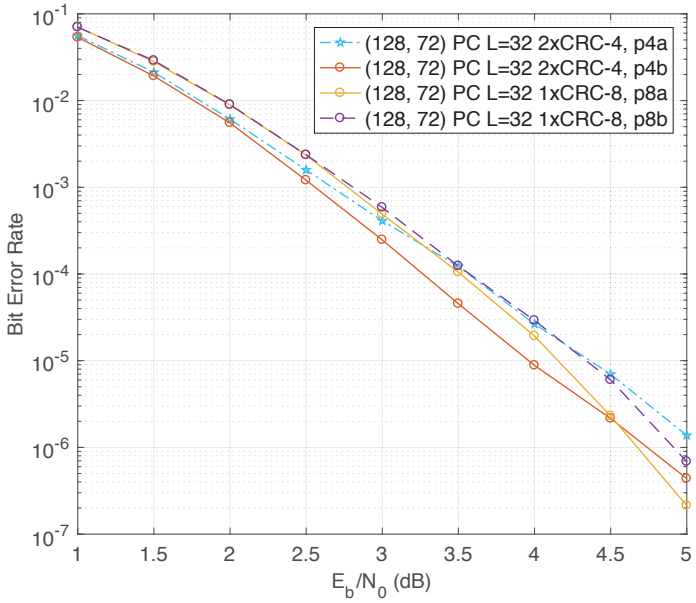

Figure 5.2: BER of polar codes with different CRC polynomial selections.

5.2 (128,72) L32 Polar Code Performances for Changed CRCs

When adding a CRC to the polar code, the resulting code rate is preserved, but the polar code rate increases. As described earlier, more channels are used to send data on, and all newly selected channels have lower computed reliability than the channels used for the lower rate polar code. When adding CRCs, we hence have to balance between the gain of the CRC and worsen polar code performance. We want to find ways to optimize the use of CRCs.

To see how a changed CRC structures influence the performance, we fix the polar code rate and list size and test for some CRCs of different compositions. In Figure 5.3 and Figure 5.4, this is done for the (128,72) polar code with list size $L = 32$.

The 8 bits dedicated to CRC are used in a few different ways. Comparisons are made between the single classic CRC-8 at the end of the message, and shorter CRCs spread out over the message. We check if a single CRC placed earlier in the decoder can improve the code, compared to a single check at the end.

The performance of dividing the eight bits into two CRC-4s is already shown in Figure 5.1 and Figure 5.2. Here, this change is further compared. We check how the code performance changes when the bits are divided into four CRC-2s and eight CRC-1s, and call this separated CRCs.

Separated CRCs are added independently to different parts of the message. To for example use four CRC-2s, the original 64-bit message is divided into four equal 16-bit parts. Each piece gets a CRC-2 check value calculated and added to the end of it, and all four new 18 bit parts are then concatenated before they are

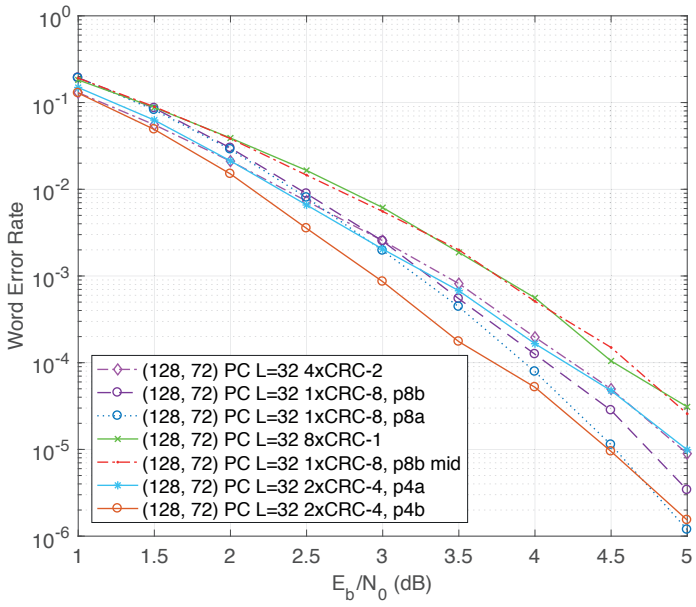

Figure 5.3: WER of (128, 74) polar codes with different CRC constructions.

encoded in the polar code. The decoder first decodes the first 18 information bits, then removes incorrect paths from the list with help of the first CRC check, before continuing the decoding process for the next set of bits, and so on.

Results for the (128,72) polar codes with L-32 list decoding are shown in Figure 5.3 and Figure 5.4 for frame error and bit error rates. In both figures, we see that a separated CRC into two CRC-4 improves the polar code performance for low SNRs. Shorter CRCs, however, does not help with the polar code decision. The worst performance is found for the eight separated CRC-8.

Another test shown in the same figures is the performance of polar code with one CRC-8 positioned halfway through the message. The hypothesis was that removing all incorrect paths earlier in the decoder, when the correct path was more likely to be in the list of possible codewords, would keep the correct path in the list until the end. Since many other paths were removed early, the correct path could possibly influence the rest of the decoding process so that the decoder decided the correct word in the end without help from a CRC check. Repositioning the CRC turned out to not work in favor of this polar code, as seen in Figures 5.3 and 5.4.

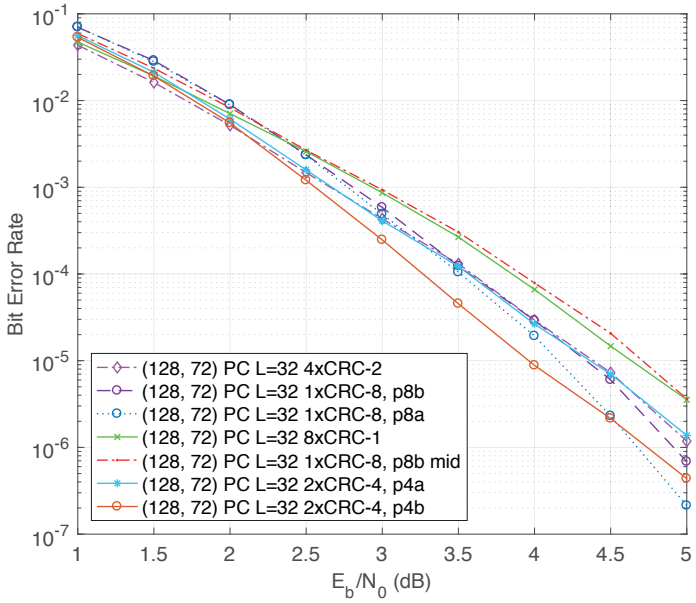

Figure 5.4: BER of (128, 74) polar codes with different CRC constructions.

5.3 (128,72) L8 Polar Code Performances for Changed CRCs

Similar comparisons as in the previous section were performed for the polar codes with list size 8 SCL decoding and different CRCs. The results are shown in Figures 5.5 and 5.6. One CRC-8 is compared with two CRC-4, finding that dividing the CRC improves performance. Both (128, 72) codes were also compared with one (128, 68) code with a single CRC-4 positioned at the end. The CRC-4 work better for low SNRs than the single CRC-8, but not as well as the divided CRC.

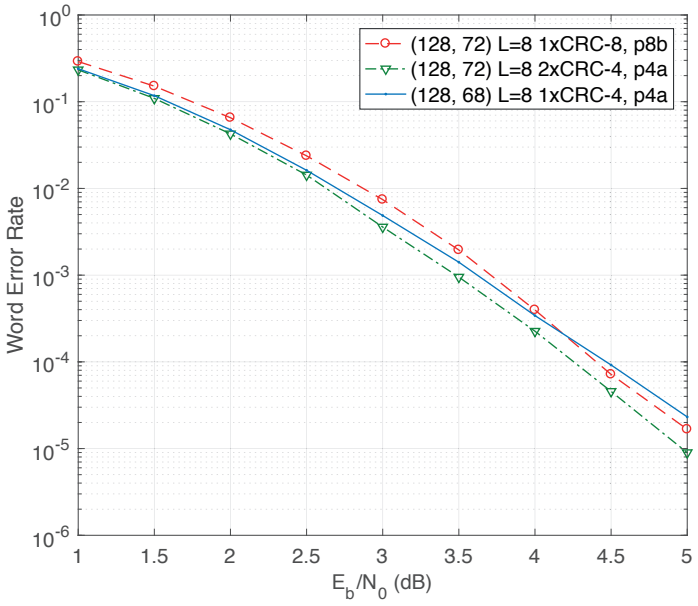

Figure 5.5: (128,64) PC with $L = 8$ and split CRC compared to shorter CRC, WER.

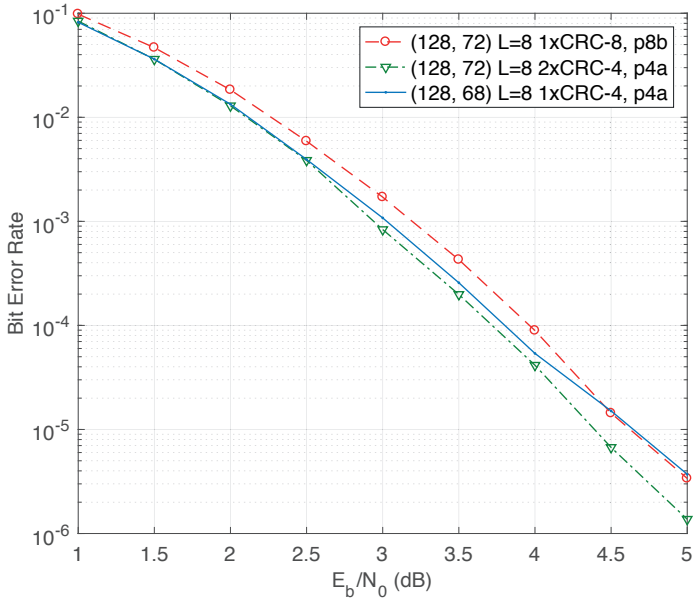

Figure 5.6: (128,64) PC with $L = 8$ and split CRC compared to shorter CRC, BER

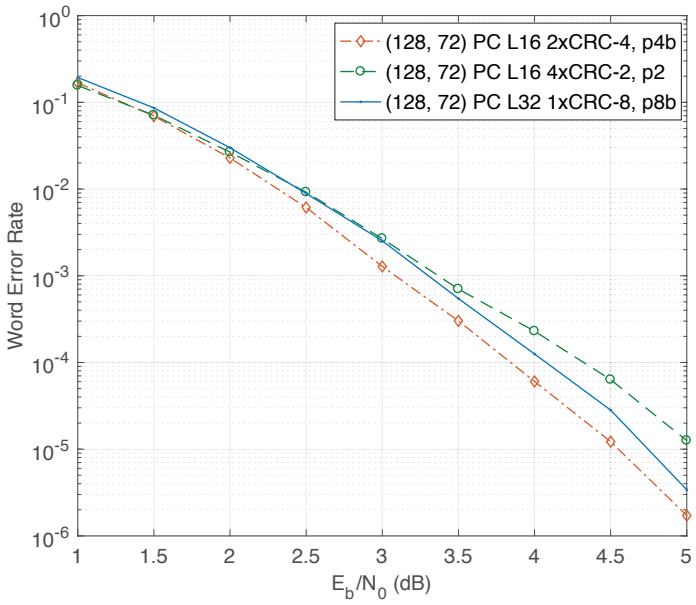

Figure 5.7: WER of (128,72) PC with L16 and split CRC to compensate for a longer list.

5.4 (128, 72) Polar Code, Divided CRC Used to Lower Complexity

Observed in previous results is that a split CRC can improve performance compared with the same total length CRC positioned at the end of the message. It is interesting to investigate if these results lead to that we can find the performance of some polar codes with longer list sizes, in other polar codes with shorter list sizes, by dividing the CRC polynomial. A shorter list means lower decoder complexity but has until now also resulted in worse code performance.

To see if this is possible for some polar codes, the (128, 72) polar code is further investigated with different list sizes. Results are shown in Figures 5.7 and 5.8. One CRC-8 at the end of the code with list size 32 is compared with split CRCs at the end of the polar code with list size 16. Results show that a good choice of split CRC can improve the code performance to compensate for a longer list size.

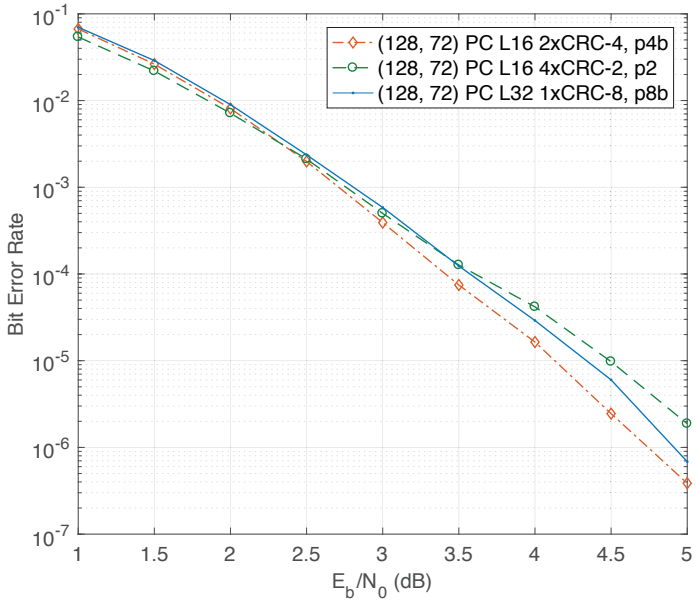

Figure 5.8: BER of (128,72) PC with L16 and split CRC to compensate for a longer list.

5.5 Chapter Summary

In this chapter, we observe that several shorter CRCs split over the original message can work better than a single longer CRC at the end of the decoder. These findings can in some cases be used to lower the complexity of a polar code with list decoding and CRC, since a divided CRC seem to be able to compensate for a shorter list size. A too short CRC, however, does not work well with polar codes. The choice of CRC-polynomial is also critical in finding the best polar code.

More tests must be run for several combinations to be able to make general conclusions about the polar code combination with list decoding and CRC.

Chapter 6
Conclusions and Future Work

Polar codes are relatively new, and there are still many open questions about how the code should be constructed to optimize performance for different channels and short code lengths. Some beneficial changes to short polar codes were recently suggested, including combining polar codes with list decoding and CRC.

This thesis tried to understand how modifications of the three parameters CRC, list decoding, and frozen bits, change the code performance of short 128-bit polar codes. We contribute with a comparison between short polar and RM codes with list decoding and CRC and investigations with modified CRCs.

6.1 The Dependence of a Good Frozen Bit Selection

The project started with a study of the frozen bit decision in polar codes. Early findings showed that small changes in frozen bits resulted in substantial changes of polar code performances. Interesting results from polar codes with different design-SNRs revealed that a polar code designed for one SNR sometimes did not perform well for channels with that SNR, but that the same code could be the best performing for a channel with some other SNR. The better design-SNR seem to be slightly shifted from the channel-SNR. A possible explanation for this could be that the design uses the Bhattacharyya parameter. The parameter is an upper bound of the probability that an error occurs, which means that the real probabilities could be lower than we approximate them for some channels.

We conclude from this result that it is not trivial to construct the best polar code for short channels and that testing some variations is necessary when trying to find the best short polar code.

The frozen bit selection was further investigated by comparing polar codes with Reed-Muller codes, both under SC and SCL decoding. We observed that polar codes were better performing under SC decoding. However, RM codes showed significant improvements under SCL decoding, where polar codes only showed slight improvements. Only a relatively short list size was needed for RM codes to perform better than polar codes under list decoding. RM codes are stronger with larger minimum distance due to how they are constructed. The results showed the benefits of selecting a stronger code together with a good decoder.

6.2 The Adding of CRCs to Polar and RM Codes

Interesting observations were made when using the combination of CRC and list decoding on polar and RM codes. Polar codes improved with the combination compared to those without CRC, which is consistent with what was found in previous research. The RM code also improved, provided that the extended frozen bit selection was done with care. Interesting was that the RM code performed better than the polar code after these modifications. This could however not be used to make general assumptions about the codes, since it seem that both the modified RM and polar code compared could be improved. A shorter list size could also possibly result in a better polar than RM code. From comparison with previous research, we see that the way of selecting frozen bits in [7] should be preferred over the use of Bhattacharyya parameters, and is recommended for further research.

From comparing the two RM codes with different frozen bit selections in Figure 4.3, it was observed how much some changes in the frozen bit selection can change the performance. The reason that a Red-Muller code with the combination could perform worse than the original RM code, is that the performance gain from using a CRC did not compensate for the loss from changing the code rate and transmitting on unreliable channels, and hence we get a worse performing code.

We conclude that both RM and polar codes can work well with list decoding and CRC, that we cannot say that one of the two codes is better under all circumstances, and again that the selection of frozen bits is critical when constructing short polar codes.

6.3 Modified CRCs with Polar Codes

The add of CRCs to polar codes was further looked into for modified CRCs. We investigated varying CRC polynomials, the CRC placement in the decoder and long CRCs split into smaller CRCs spread out over the code. All performance tests were run for $N = 128$ polar codes with the combined code rate of 0.5 and changing list sizes in the SCL decoder.

We found that a split CRC sometimes improved the polar code combination compared to a single longer CRC at the end of the decoder. In some cases, divided CRCs could be used to improve the code complexity, since the split code worked as well for a shorter list size as the original CRC did for a more extended list decoder.

Other observations from adding CRCs to polar codes with list decoding, was that the CRC could be both too short and too long. Shorter CRCs did not find enough errors to help the decoder, and too long CRCs changed the polar code rate too much, so that more information got sent on unreliable channels. Moving the CRC to the middle of the decoder also showed to be a bad idea. Errors occurred again after the CRC removed wrong paths halfway through the decoder, those errors were not corrected, and the decoder continued to choose incorrect paths as most likely paths in the last decoder step.

6.4 The Three Parameters of Modified Polar Codes

Important to note when analyzing polar codes with list decoding and CRC, is that the addition of both CRC and list decoding change the polar code, and this thesis does not look at in detail how the three parameters affect each other.

The design-SNR used in this project was optimized for polar codes of rate 0.5, but in the codes with CRCs, the combined code rate is 0.5, and the polar code rate is therefore higher. The preferred design-SNR, in that case, could be another. The design-SNR also optimizes the polar code under SC decoding. We have seen when comparing with RM codes the the frozen bit set optimal for SC decoding was not the best performing set under SCL decoding. The frozen bit selection should, therefore, be investigated more in combination with list decoding and CRC before any conclusions are drawn. Also, the selection of CRC polynomials must be controlled since the choice changes performance.

6.5 Conclusions

We conclude that it is not trivial to find the best short polar code to use on any channel. From observations, we know that many factors play a role in the search for the perfect short polar code. The frozen bits play a significant role, and a small change in the frozen bit selection result in considerable performance changes. The Bhattacharyya parameter is not be the best way to select frozen bits for different channels.

Both the construction and placement of CRCs play a part in finding an optimal polar code for a channel. Generator polynomial and lengths of divided CRCs must be optimized and tested before implementation, since all parameters influence the code performance. The whole CRC should however not be moved to a earlier decoder step.

In short, we conclude from the results in this thesis, that more elaborate tests must be done to find the best polar code for every system. We have to be careful when designing polar codes for short messages, such as in 5G, since all parameters change the code performance and it is not self-evident how to find the best code for any short code system.

We should also not exclude the RM code or a combined RM-polar from the search of the best short codes.

6.6 Future Work

A lot can be done with polar codes to continue the research.

This project ran tests for some different selected polar codes with list decoding and CRC, with the goal to understand how the parameters added to the excellent performance of polar codes that were found in previous research. These tests were not elaborate enough to draw any general conclusions from, or suggest design parameters for the best polar codes for different channels. This thesis found that every part of polar codes changes the performance, but not how they affect each other's performances.

Future work should therefore include investigations of whether and how the add of CRC and list decoding change the preferred polar code design-SNR for different list sizes in the decoders. The selection of frozen bits in polar codes should also be updated to some other with better working channel estimates than that with the Bhattacharyya parameter. The best CRC polynomials to use with polar codes needs to be found, together with an analysis of if the same CRC polynomials are preferred for all short polar codes. It is interesting to investigate if unevenly split CRCs could improve the outcome further.

This thesis found exciting results for the $N = 128$ polar code, and they should be complemented with similar simulations for some different length short polar codes, to see if results are consistent between all short polar codes.

The exciting finding that RM codes sometimes work better than short polar codes with CRC, in use of a long list size, leads to new questions. The parameter combination is that suggested for polar codes to make them applicable in 5G, and we here observe that RM codes can work as well or better under some circumstances. From all Reed-Muller code results, we observe that the main improvements of the code are due to the use of a long list size. One can assume that polar codes would outperform all RM codes if short enough list sizes are used. Since low complexity is desirable in most cases, it is interesting to test if this hypothesis validates.

The three parameters investigated in this thesis can be combined in many ways for short polar codes. What combination that is the best depends on the system it is constructed for, but more tests, aiming to find more general patterns, could be performed. A frozen selection influenced by both RM and polar codes could in some cases improve the combined code performance, according to [10] and [11], and should be firmly investigated in combination with list decoding and CRC.

In short, there is a lot that should be investigated and many tests that should be run for short polar codes before they get implemented, to optimize performance when using the combination of polar codes with list decoding and CRC.

References

[1] E. Arikan, *Channel polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels*, IEEE Trans. Inform. Theory, vol. 55, no. 7, July 2009.

[2] E. Arikan, *A Survey of Reed-Muller Codes from Polar Coding Perspective*, IEEE Inform. Theory Workshop on Information Theory, Jan 2010.

[3] E. Arikan, *A Performance Comparison of Polar Codes and Reed-Muller Codes*, IEEE Communications Letters, vol. 12, no. 6, June 2008.

[4] E. Arikan, H. Kim, G. Markarian, Ü. Özgür, E. Poyraz, *Performance of short polar codes under ML decoding*, Proceedings of the ICT-Mobile Summit Conference, June 2009.

[5] I. Tal, A. Vardy, *List Decoding of polar Codes*, IEEE Transactions on Information Theory, 61 (5), 2213 - 2226, 2015.

[6] I. Tal, A. Vardy, *List Decoding of polar Codes*, IEEE International Symposium on Information Theory Proceedings, 2011.

[7] G. Liva, L. Gaudio, T. Ninacs, T. Jerkovits, *Code Design for Short Blocks: A Survey*, arXiv preprint arXiv:1610.00873, Oct. 2016.

[8] H. Vangala, E. Viterbo, *A Comparative Study of Polar Code Constructions for the AWGN Channel*, arXiv, Jan. 2015.

[9] M. Qin, J. Guo, A. Bhatia, A. Fàbregas, *Polar Code Constructions Based on LLR Evolution*, IEEE Communications Letters, vol. 21, no. 6, June 2017.

[10] B. Li, H. Shen, D. Tse, *A RM-Polar codes*, arXiv, July 2014.

[11] M. Mondelli, S. H. Hassani, R. L. Urbanke, *From Polar to Reed-Muller Codes: A Technique to Improve the Finite-Length Performance*, IEEE Transactions on Communications, vol. 62, no. 9, Sept. 2014.

[12] I. Dumer, K. Shabunov, *Soft Decision decoding of Reed-Muller codes: recursive lists*, IEEE Trans. Inform. Theory, vol. 52, pp. 1260 to 1266, 2006.

[13] P. Koopman, T. Chakravarty *Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks*, Preprint: The International Conference on Dependable Systems and Networks, DSN-2004.

[14] C. E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, July, Oct. 1948.

[15] G. Lindell, *Introduction to Digital Communications*, Lund University, Electrical and Information Technology, Aug. 2006.

[16] I. Tal, A. Vardy, *How to Construct Polar Codes*, IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 6562, 6582, Oct. 2013.

[17] P. Koopman, *Best CRC Polynomials*, Carnegie Mellon University https://users.ece.cmu.edu/ koopman/crc/index.html.

[18] Mathworks Documentation, *comm.CRCGenerator System object*, https://se.mathworks.com/help/comm/ref/comm.crcgenerator-class.html.

[19] H. D. Pfister, *A Brief Introduction to polar Codes*, Lecture Notes, April 2012.

[20] K. K. Nagar, K. Sharma, S. Tyagi, *Implementation of REED MULLER CODE in MATLAB*, International Journal of Scientific and Engineering Research, vol. 9, no. 9, Sept. 2013.

[21] I. Reed, *A class of multiple-error-correcting codes and the decoding scheme*, IRE Trans. Inform. Theory, vol. 4, pp. 39-44, Sept. 1954.

[22] D. E. Muller, *Application of Boolean Algebra to Switching Circuit Design and to Error Detection*, IRE Trans. Electronic Computers, col. EC-3, pp. 6-12, Sept. 1954.

[23] R. Lucas, M. Bossert, A. Dammann, *Improved soft-decision decoding of Reed-Muller codes as generalized multiple concatinated codes*, Proc. ITG Conf. on Source and Channel Coding, Aachen, Germany, pp. 137-141, 1998.

[24] *Polar Codes*, polarcodes.com

Printed by Tryckeriet i E-huset, Lund 2017

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2017-607

http://www.eit.lth.se