Embedded LDPC codes

Final Report

Hanno Jacobs
17000042

Submitted as partial fulfilment of the requirements of Project EPR400
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2022

Study leader: Dr. Filip Paluncic

Part 1. Preamble

This report describes work that I did in implementing Embedded LDPC codes.

Project proposal and technical documentation
This main report contains an unaltered copy of the approved Project Proposal (as Part 2 of the report).

Technical documentation appears in Part 4 (Appendix).

All the code that I developed appears as a separate submission on the AMS.

Project history
This project makes extensive use of existing algorithms on LDPC encoders and decoders based on iterative belief propagation algorithms. Some of the algorithms I used were adapted from a course on LDPC codes by IITM [1]. Where other authors' work has been used, it has been cited appropriately, and the rest of the work reported on here, is entirely my own.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was Hanno Jacobs.

Language editor signature: 
Date: Thursday 10th November, 2022

Declaration

I, Hanno Jacobs understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will be regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a

verbatim copy of the approved Project Proposal.

Hanno Jacobs: 
Date: Thursday 10th November, 2022

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AWGN	Additive white Gaussian noise
BER	Bit error rate
BPSK	Binary Phase-shift keying
SNR	Signal-to-noise-ratio
Eb/No	Energy per bit to noise power spectral density ratio
BER	Bit error rate
PDF	Probability density function
CDF	Cumulative density function
LLR	Log-likelihood ratio
XOR	Exclusive or
dB	Decibel
NR	New-radio

Part 2. Project definition: approved Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unaltered from the final approved version that appears on the AMS.

For use by the Project lecturer	Approved		Revision required	

Feedback

To be completed by the student

PROJECT PROPOSAL 2022

Project no	FP2	Revision no	1

Title	Surname	Initials	Student no	Study leader (title, initials, surname)
Mr	**Jacobs**	**H**	17000042	Dr F. Paluncic

Project title

Embedded implementation of LDPC codes

Language editor name	Language editor signature
Hanno Jacobs	
Student declaration I understand what plagiarism is and that I have to complete my project on my own.	**Study leader declaration** This is a clear and unambiguous description of what is required in this project
Student signature	Study leader signature and date 23/06/2022

1. Project description

What is your project about? What does your system have to do? What is the problem to be solved?

Error correction codes have been prevalent since the adoption of digital communication during the second world war. The problem with digital communication, as with all communication, is message reconstruction after communication over a noisy channel. Early correction codes such as Hamming codes allowed for the detection of errors based on creating parity bits that ensure that there are an even number of ones in a message. This allowed for the development of the first useful error-correcting codes. This parity bit idea was extended to create complex LDPC codes that can be decoded probabilistically using iterative message-passing decoding. Message-passing decoders allow for the intrinsic and extrinsic beliefs for each bit to be determined, resulting in the optimal bit state. These codes can come close to the optimal, so-called, Shannon's limit which is a particular SNR, at or above which, you can make the probability of error arbitrarily small for a given code-rate. LDPC codes will be implemented on a resource-constrained (limited processing speed, memory and storage) embedded processor in this project using a complex iterative message-passing decoding algorithm, in order to decode a message with very large codeword lengths, that has been passed through a simulated Rayleigh fading channel. The Rayleigh fading channel is used as a model for a real wireless communication channel.

2. Technical challenges in this project

Describe the technical challenges that are *beyond* those encountered up to the end of third year and in other final year modules.

2.1 Primary *design* challenges

An LDPC implementation will be designed that can allow very large codewords to be decoded such that it does not result in either memory overflow or very long runtime (improper implementation designs will result in more than 16MB of memory usage per codeword, and more than 1 second per codeword).

An LDPC decoder will be designed based on the iterative message-passing algorithm that can be effectively implemented on an embedded platform.

LDPC code parameter design will be done to ensure that the system comes sufficiently close to Shannon's limit.

LDPC code parameter design will be done to ensure that the system can decode a codeword passed through a noisy Rayleigh channel at a low BER.

Analytics software will be designed that can reliably capture the performance metrics of the LDPC system.

2.2 Primary *implementation* challenges

Implementation of a first-principles LDPC encoder on an embedded platform.
Implementation of a first-principles LDPC decoder on an embedded platform, that can decode a codeword in a given time frame.
First-principles, implementation of the probabilistic iterative message-passing algorithm on an embedded platform.
Implementation of an AWGN noise channel simulation will be carried out on an embedded platform.
Implementation of a Rayleigh fading channel model on a PC (this will necessitate a very good LDPC implementation to result in an effective decoder at a poor signal-to-noise ratio).
Decoding of very long codeword lengths in order to approach the Shannon limit, on resource-constrained embedded platform.

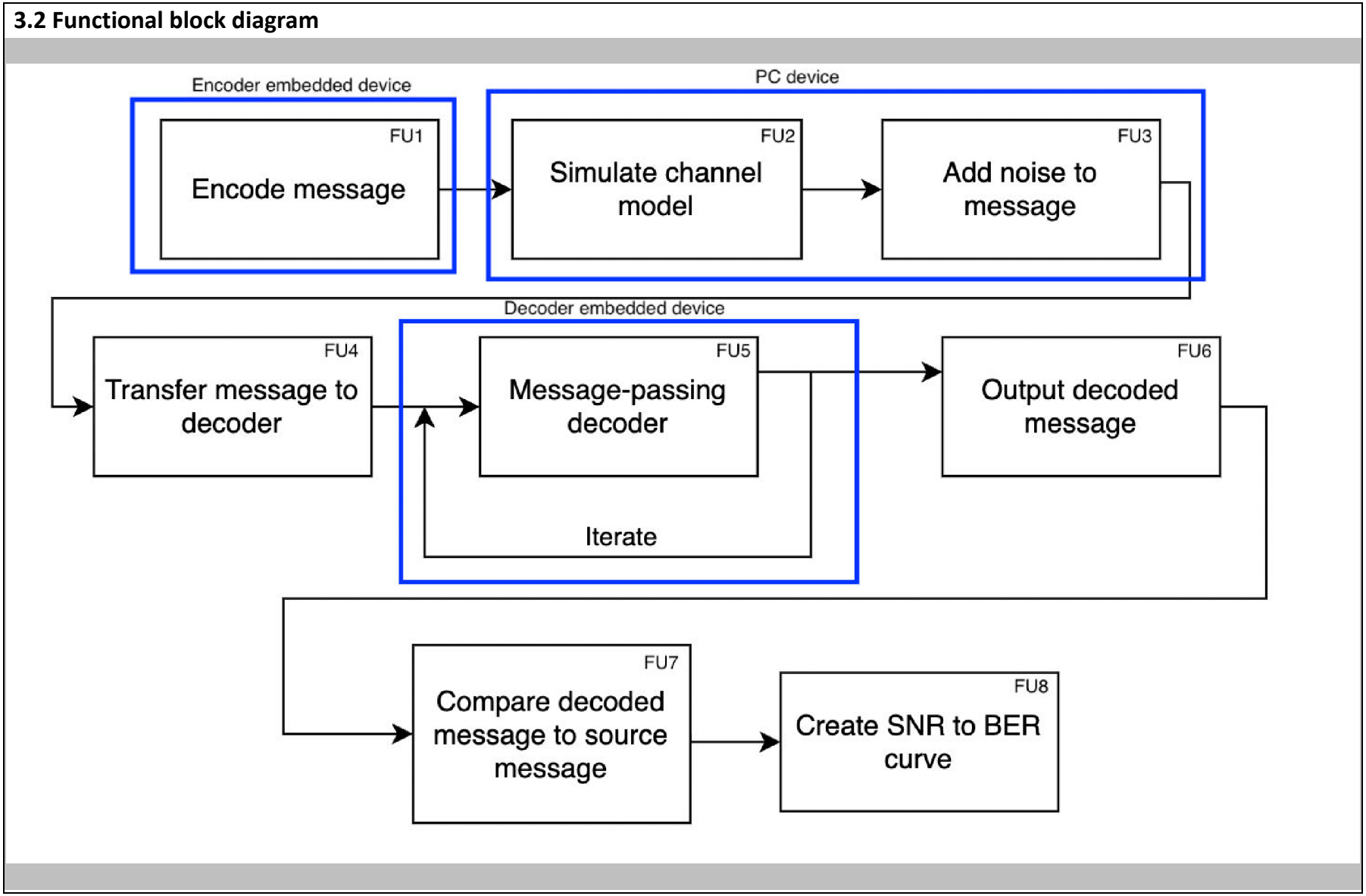
3. Functional analysis

3.1 Functional description

Describe the design in terms of system functions as shown on the functional block diagram in section 3.2. This description should be in *narrative format*.

Initially, on the encoder device, a randomly generated binary source message is generated as a first test case and a real-world audio message as a second case will be encoded using a sparse parity check matrix (FU1) where each bit is protected by multiple parity-check bits that allow for multiple bit flips to be corrected even in the presence of high noise. After the encoding process, the selected channel model (Rayleigh in the first test case and AWGN in the second test case) is simulated (FU2) and the necessary noise is added to the encoded message (FU3). This noisy message is then transferred to the decoder (FU4) which is implemented on another separate embedded device. The message is subsequently decoded using an iterative message-passing algorithm that has been optimized for implementation on a resource-constrained (limited processing speed, memory and storage) embedded device that can deal with very long codeword lengths (FU5). Here the intrinsic and extrinsic beliefs of the bit node and check node values are passed between bit nodes and check nodes so that the believed value of the bit nodes can be updated. This is repeated iteratively so that the believed values converge to the best possible belief of what the source message is. The decoded message is then output (FU6) and compared to the source message (FU7). Finally, an SNR to BER curve is plotted and used to analyse the efficacy of the system (FU8). The functional block diagram is shown below in figure 1.

3.2 Functional block diagram



4. System requirements and specifications

These are the core requirements of the system or product (the mission-critical requirements) in table format IN ORDER OF IMPORTANCE. Requirement 1 is the most fundamental requirement.

	Requirement 1: the fundamental functional and performance requirement of your project	**Requirement 2**	**Requirement 3**
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	The system should be able to decode a message with a low probability of error in such a way that the system comes sufficiently close to Shannon's limit.	The system must decode a codeword passed through a Rayleigh channel simulation, in the presence of noise, with a low probability of error.	The system's decode time must be low enough to ensure useful communication.
2. What is the target specification (in *measurable* terms) to be met in order to achieve this requirement?	The system should be within 2.5 dB of Shannon's limit at a BER of 1x10^-4.	The system must decode a Rayleigh channel codeword in the presence of 24dB Eb/No of signal-to-noise ratio, with a BER lower than 1x10^-4. The stated BER will cause an imperceptible change in audio quality.	The system must not take longer than 1 second to decode 1 codeword of the selected codeword length.
3. Motivation: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)	This will ensure that the system provides near-optimal performance that is in line with the performance obtained in the literature for systems of the selected block length. The stated BER will cause an imperceptible change in audio quality.	From literature LDPC codes, in a Rayleigh channel, have been shown to achieve BER lower than 1x10^-4 at approximately 24dB Eb/No (this is 10dB better than the uncoded BPSK case).	A useful communication system must decode a message in a reasonable time frame. From the literature, this time frame is deemed to be reasonable for high-complexity codes such as LDPC codes on an embedded platform.
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	100 codewords will be decoded at the specified noise specification and codeword length, and the BER will be shown to be lower than 1x10^-4 for the given AWGN channel model.	100 codewords will be decoded at the specified noise specification and codeword length, and the BER will be shown to be lower than 1x10^-4 for the given Rayleigh channel model.	One codeword of the given codeword length will be decoded and the decode time will be measured. The decode time must be less than 1 second.
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*	Implementation of encoder and decoder from first principles.	Implementation of encoder and decoder from first principles.	Implementation of encoder and decoder from first principles.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	An embedded hardware platform such as a DSP, SDR, FPGA or another type of microprocessor.	An embedded hardware platform such as a DSP, SDR, FPGA or another type of microprocessor.	An embedded hardware platform such as a DSP, SDR, FPGA or another type of microprocessor.

System requirements and specifications page 2

	Requirement 4	Requirement 5	Requirement 6
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	The codeword length of the system will be in a certain range to attain requirements 1 and 2.	The code rate of the system must be large enough to ensure useful communication.	
2. What is the target specification (in *measurable* terms) to be met in order to achieve this requirement?	The codeword length used in the encode-decode process should be in the range of 2000 to 5000 bits.	The system will only be implemented for code rates that are greater than 0.32. This means that the number of message bits divided by the total number of bits in a codeword will be greater than 0.32.	
3. Motivation: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)	LDPC systems can only approach Shannon's limit for large codeword lengths. The LDPC system will become more effective as the codeword length increases. In the literature, this block length range satisfies the required specifications.	A useful communication system must use a significant proportion of its encoded bits as message bits. From literature, most implementations of LDPC codes are for code rates above 0.32.	
4. How will you demonstrate at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?	The entire system demonstration will only be done for codeword lengths in the range of 2000 to 5000 bits.	The entire system demonstration will only be done for code rates greater than 0.32.	
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*	Implementation of encoder and decoder from first principles.	Implementation of encoder and decoder from first principles.	
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none"	An embedded hardware platform such as a DSP, SDR, FPGA or another type of microprocessor.	An embedded hardware platform such as a DSP, SDR, FPGA or another type of microprocessor.	

System requirements and specifications page 3

	Requirement 7	**Requirement 8**	**Requirement 9**
1. <u>Core mission requirements of the system or product.</u> Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.			
2. What is the <u>target specification</u> (in *measurable* terms) to be met in order to achieve this requirement?			
3. <u>Motivation</u>: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)			
4. How will you <u>demonstrate</u> at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?			
5. <u>Your own design contribution:</u> what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*			
6. What are the aspects <u>to be taken off the shelf</u> to meet this requirement? If none, indicate "none"			

System requirements and specifications page 4

	Requirement 10	Requirement 11	Requirement 12
1. <u>Core mission requirements of the system or product.</u> Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.			
2. What is the <u>target specification</u> (in *measurable* terms) to be met in order to achieve this requirement?			
3. <u>Motivation</u>: *how* or *why* will meeting the specification given in point 2 above *solve the problem*? (Motivate the *specific* target specification selected)			
4. How will you <u>demonstrate</u> at the examination that this requirement (point 1 above) and specification (point 2 above) has been met?			
5. <u>Your own design contribution:</u> what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*			
6. What are the aspects <u>to be taken off the shelf</u> to meet this requirement? If none, indicate "none"			

5. Field conditions

These are the REAL WORLD CONDITIONS under which your project has to work and has to be demonstrated.

	Field condition 1	**Field condition 2**	**Field condition 3**
Field condition requirement. In which field conditions does the system have to operate? Indicate the one, two or three most important field conditions.	The system must be able to function for very high noise implementations in a Rayleigh fading channel. This channel results in a very high noise implementation that are very hard to decode.		
Field condition specification. What is the specification (in measurable terms) for this field condition?	The Eb/No will be in the range of 1dB to 25dB for the system demonstration.		

6. Student tasks

6.1 Design and implementation tasks

List your primary design and implementation tasks in bullet list format (5-10 bullets). These are *not* product requirements, but *your* tasks.

The first-principles design of an LDPC encoder.
The implementation of an effective AWGN channel model for simulating noise on a transmission channel.
The implementation of an effective Rayleigh fading channel model.
The implementation of a probabilistic iterative message-passing algorithm.
The first-principles design of a probabilistic iterative message-passing decoder that meets the required specifications.
The implementation of analytics software to capture system performance metrics.

6.2 New knowledge to be acquired

Describe what the theoretical foundation to the project is, and which new knowledge you will acquire (*beyond* that covered in any other undergraduate modules).

Acquiring knowledge on how LDPC encoding and decoding works.
Acquiring an understanding of how the probabilistic message-passing algorithm works.
Acquiring an understanding of how to implement efficient approximations of the message-passing algorithm for resource-limited applications.
Acquiring an understanding of what the different performance metrics are for error correction codes.
Acquiring an understanding of what Shannon's limit is for error correction code systems.
Acquiring an understanding of how to effectively implement resource-intensive code on resource-limited embedded platforms.
Acquiring an understanding of how Rayleigh fading channels work, are implemented in simulation and are represented mathematically.

Part 3. Main Report

1. Literature study

The implementation of error correction codes has been prevalent since the adoption of digital communication during the second world war. The problem that faces digital communication, as with all types of communication, is the reconstruction of message signals after communication over a noisy channel. The first method of increasing the likelihood of receiving a correct message relied on increasing the transmission power of the signal with respect to the noise power of the channel. However, this solution is sub-optimal since it requires increased power usage which is impractical for mobile power-limited applications. This increased power usage is subverted with the inception of error correction codes that would transmit messages at the same power level and then be able to reconstruct the original message at the receiver. The first types of error correction codes were simple repetition codes where the original message was repeated a number of times so that the original message could be reconstructed in the case of erasures or bit-flips. This simple method of encoding was further extended to the idea of using parity bits that would ensure an even number of ones in the encoding process.

The first type of codes that used the parity bit idea were the so-called Hamming codes. These codes would split the message up into several parts that were each protected by their own parity bit. This ensured that the presence of errors were able to be determined and even corrected for more complex Hamming codes.

This method of encoding messages with parity bits would result in a greater performance at higher signal-to-noise ratios at the expense of lower code rates. These lower code rates reduced the system efficiency and would require more source bits to be transmitted for each message bit.

More complex encoding schemes such as LDPC codes were first developed in 1963 by Robert G. Gallager [2]. The purpose of LDPC codes is to provide a low density of parity overlap in the case of very long codes where bits in the parity check matrix can be protected by more than one parity bit. This helps with noisy transmissions where even the parity bits of the system can be in error. The idea of low-density parity bit protection is used in order to keep the parity matrix sparsely populated. The sparsity of the parity check matrix makes the encoding process quicker as there is lower density and therefore fewer bits to encode in the process. The sparsity of the parity matrix also makes the decoding process quicker since there are fewer bits to decode in each check node. This sparse parity check matrix is simpler to decode than Hamming codes and would have overlapping sections of the message that were protected by more than one parity check bit in order to provide correction ability even if more than one bit was flipped in one parity bit section of the message.

LDPC codes enable more corrections than simpler codes such as Hamming codes and enable better performance at higher signal-to-noise ratios that allow the system to come closer to the so-called Shannon's limit for digital communication systems.

Apart from the idea of low-density parity-check codes, there is also the idea of probabilistic decoding that allows the system to use soft-decoding where an iterative message passing algorithm [3] can be executed in order to allow the system to more accurately determine what the correct state of each bit should be.

Message-passing decoding in LDPC codes is based on the iterative message-passing algorithm [4] whereby each bit shares its extrinsic beliefs of the bits states of its neighbouring bits should be. This message-passing method is possible due to the encoding that the system possesses. This encoding is implemented in the parity check matrix that ensures that there is an even number of 'ones' in each row. Since each group of bits is protected by its connection with a common check bit there is an intrinsic belief of the optimal value of each bit in the group based on the communication channel through which the messages are passed. This intrinsic belief for each bit is combined with the extrinsic belief for each bit that is obtained through the connection of bits to a common check bit. The initialization of the message-passing process starts with each bit obtaining its intrinsic belief from the communication channel across which the message is passed. This intrinsic belief is obtained in section 3.3.2. The message-passing process then starts with the initial beliefs of each bit passed to each of its check-bit-connected neighbours. Then the extrinsic beliefs of each of the bits are determined as in section 3.3.3 and passed to each of the bits' check-bit-connected neighbours. This is a row operation that takes place in the parity check matrix. This is repeated for each bit in the message. The sum of all the extrinsic beliefs for each bit is calculated to determine the total extrinsic belief of each bit. This is a column operation in the parity check matrix that gives the current belief of the optimal state of each bit.

The iterative message passing algorithm allows the system to use message passing to allow each bit to pass its belief of what the other bits' state should be based on the probabilistic model of the channel [5] and the intrinsic beliefs of each bit to be shared with the other bits in its parity check node [6]. The message-passing algorithm works by first allowing each bit to obtain its intrinsic belief of what its value should be based on the channel over which the message is passed. Then each bit in the check node passes its beliefs of the other bits in the check node to the other bits in the check node. Then each bit receives can determine what its value should be both based on the beliefs that each bit has about the other bits in the check node and on the beliefs that the other bits in the check node have about it.

The LDPC codes can then be encoded using the sparse parity check matrix and decoded using the iterative message passing algorithm to give the decoded value of the noisy bits that are most likely to be correct.

LDPC codes can be used to communicate BPSK signals over a wide variety of communication channels in order to come close to the optimal Shannon's limit for the particular code rate used. Shannon's limit [7] is the Eb/No value, at or above which, the probability of error can be made arbitrarily small for a given code rate. Shannon's limit varies depending on the type of communication channel the signal is sent over and the code rate of the signal. The goal of error-correcting codes is to obtain a result that is as close to Shannon's limit as possible at an error rate that is considered low enough for the nature of the signal that is being communicated over the channel. Shannon's limit is defined as a certain Eb/No value for a given code rate. The code rate used in this project is 0.3235 which results in a Shannon's limit of approximately -0.5309 dB for an AWGN channel [8]. This means that the lowest signal-to-noise ratio for which a code of the given rate's probability of error can be made arbitrarily small is at -0.5309 dB. This means that the probability of error for the system can be made arbitrarily small for a signal power that is 0.89 times the noise power.

Additive white Gaussian noise is noise which is normally distributed [9]. The noise power

is modelled as a normal distribution with noise of higher power being less likely than noise with moderate power [10]. The noise power of an additive white Gaussian system is the most common type of noise that is used to test the efficacy of error-correcting code systems. This is due to the wide range of noise types that can be modelled using a normal distribution. The additive nature of the noise is very simple to implement and is used as the standard by which the BER performance of error-correcting code systems is measured.

A Rayleigh fading channel is a noise channel that is used to model the scattering that takes place in the real world when a signal is transmitted from one point to another in a non-uniform environment where the signal is able to bounce off different surfaces on its way to the receiver device. The Rayleigh channel is a channel where these scattered signals are the only signals that get to the receiver, therefore there is no line of sight component that dominates the transmission on the channel.

The scattering off of physical objects cause a scattered signal to arrive at the receiver at different time instances. This is modelled as a phase delay that is present in the final signal that has to be decoded. The Rayleigh channel is also further extended to model Doppler shift that is present in a system as a result of an object that is moving with respect to the transmitter. This Doppler shift presents a frequency shift in the signal.

The Rayleigh channel is a very adverse channel model that results in much poorer BER performance than the AWGN channel in terms of the BER at a given signal. The model of the Rayleigh channel that will be implemented in the given system is the flat Rayleigh fading channel. The flat fading channel is used to model the performance of error-correcting codes that are used in communication protocols that operate over a narrow bandwidth. This is the case in OFDM systems where each frequency over which a particular carrier in the system operates is a relatively small portion of the total system bandwidth. This results in a system with an amplitude response that can be modelled as approximately flat for the given frequency range.

The contribution of this project is to optimize an LDPC encoder/decoder system to perform well on an embedded system with low processing power and low memory resources. The project aims to implement the system to function not only in the standard case of BPSK over an AWGN channel but also over a Rayleigh flat-fading channel [11] using a randomly generated binary source message as well as a real-world audio signal. Therefore, the challenge is twofold: implementing the system effectively on an embedded platform and optimizing that implementation to provide good BER performance over a very adverse, simulated Rayleigh flat-fading channel.

2. Approach

This project aims to develop a probabilistic LDPC encoder that is able to decode messages effectively to come close to the optimal Shannon's limit for an error-correcting system in the presence of an AWGN channel and implement this error-correcting system on an embedded platform. The project has the secondary goal of being able to accurately decode a real-world audio signal in the presence of a Rayleigh flat-fading channel model.

The system will be able to decode messages using a probabilistic decoding algorithm that is able to use iterative message passing in order to determine the optimal state of each bit in the message. The system will be able to complete the task of decoding noisy bits in such a way that the source message will be able to be reconstructed with a low probability of error.

The system will be able to decode codewords of a length of between 2000 bits and 5000 bits to the specification of a BER of less than 1×10^{-4} at an Eb/No value of 2.5 dB from Shannon's limit for the given code rate of the message. The code rates that will be used in the implementation will be larger than 0.32 in order to ensure that a reasonable proportion of the transmission is the source message.

The system will be optimized in order to allow for a codeword to be decoded in under 1 second on a low processing power, low memory resources embedded device at the desired BER and Eb/No specification.

The requirements of this project are to implement a probabilistic LDPC encoder, decoder and noise generator in software on an embedded platform. The first step in the development of the system will be to develop a fully functional encoder in simulation on a PC platform that is able to encode a rudimentary message using a basic parity check matrix. The second step is to create a simulated AWGN channel and add this simulated noise to the encoded message. The third step of the initial system simulation is to create a basic probabilistic LDPC encoder that can decode messages in the presence of low amounts of AWGN noise.

This is a rudimentary LDPC system that can be used to test the system conceptually and ensure that the system works end-to-end. Steps 1 to 3 are all initially implemented on a PC device in MATLAB this allows for quick testing and development of code that can be simply debugged. MATLAB is also favourable since it allows for matrices to be viewed and represented simply in the debugging process.

Once the rudimentary LDPC system has been developed in the presence of an AWGN channel the optimization process is started and the system is implemented using the 5G-NR base matrix structure for the parity check matrix. This system with an unoptimized encoder and decoder is developed using a 5G-NR base matrix that meets the target specifications for codeword length and code rate.

The system is then optimized using an optimized encoder and decoder structure that makes use of in-place operations in order to improve the memory usage and run-time performance of the system in anticipation of the final embedded implementation.

The decoder is then optimized in order to meet the required BER specifications at the requisite distance from Shannon's limit and is translated into Python in anticipation of embedded

device implementation.

Once this optimized version of the decoder is able to meet the required specifications a Rayleigh flat-fading channel model is created in software and the system is tested in the presence of the required Eb/No value for the given channel. The system is then optimized for the Rayleigh flat-fading channel in order to meet the required specifications.

After the system is tested and able to successfully pass the necessary qualification tests in both the Rayleigh and the AWGN channels then the system is implemented on the selected embedded platform and optimized in order to meet the necessary run-time specifications. The necessary optimizations include: translating the code from Python to C++ so that the system can meet the run-time specification on the embedded platform, optimizing the run-time by making use of in-place calculations that reduce the number of empty spaces in the parity check matrix that the decoder has to traverse over, and BER optimizations such as offset, layering and testing of other algorithms such as the sum-product algorithm. The system is also tested to insure that the memory requirements are met whilst using the optimized 5G-NR base matrix encoder and decoder structure.

Once the system is tested on a single embedded device the system will be split up so that the encoder, noise channel simulation model, decoder and analysis software is split up. The split system is then implemented where the encoder and decoder are on their own separate embedded devices and the channel model is implemented on a PC device that is in between the two embedded devices. The PC device is responsible for facilitating the addition of noise, communication and analysis of the final system's performance.

3. Design and implementation

3.1 Design summary

Table 1.
Design summary table part 1

Deliverable	**Implementation**	**Completion of deliverable, and section in report**
Design of basic encoder in AWGN channel.	The basic encoder is successfully implemented in Python.	Completed. Shown in section 3.5.2.
Design of basic decoder in AWGN channel.	The basic decoder is successfully implemented in Python.	Completed. Shown in section 3.6.1.
Design of optimized decoder in AWGN channel.	The decoder is successfully run-time optimized in order to ensure efficient run-time operation and efficient memory usage.	Completed. Shown in section 3.5.5.
BER optimization of decoder in AWGN channel.	The BER performance of the system is optimized successfully in order to provide better performance.	Completed. Shown in section 3.6.2.
Implementation of Rayleigh flat-fading channel simulation.	The Rayleigh flat-fading channel model is successfully created in order to ensure real-world operation and efficacy of the encoder.	Completed. Shown in section 3.4.2.
Implementation of Rayleigh flat-fading channel on encoder and decoder.	The Rayleigh channel model is effectively applied to the encoded signal in order to model the operation in a practical Rayleigh channel.	Completed. Shown in section 3.4.2.
Implementation of encoder, decoder, and channel model on embedded device.	The system is successfully implemented on an embedded device. The BER and run-time performance is better than the required specifications.	Completed. Shown in section 3.12.

Table 2.
Design summary table part 2

Deliverable	**Implementation**	**Completion of deliverable, and section in report**
Implementation of 2 embedded device communication with PC.	The system is successfully implemented on 2 embedded devices and is able to complete the necessary processes in order to facilitate communication between the PC and the embedded devices.	Completed. Shown in section 3.12.8.

3.2 Theoretical background

3.2.1 Additive White Gaussian Noise (AWGN)

This section elaborates on the explanation given in the literature study section and illustrates how the AWGN channel model is created. The distribution of an AWGN channel is shown here to illustrate the noise that is added to the signal in the AWGN case.

A histogram of 10^6 samples of AWGN noise is shown in figure 1 where the magnitude of the noise is compared to the number of samples that fall within that magnitude bin. The normal distribution of the AWGN noise can be observed. The cumulative density function of the AWGN channel is shown in figure 2.

3.2.2 AWGN equation

The probability density function of AWGN is represented in equation 1. The CDF equation is shown in equation 2, where the error function, 'erf', is shown in equation 3, and where x is the Eb/No of the equation.

$$PDF(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \tag{1}$$

$$CDF(x,\sigma) = \frac{1}{2}[1 + erf(\frac{x}{\sigma\sqrt{2}})] \tag{2}$$

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{3}$$

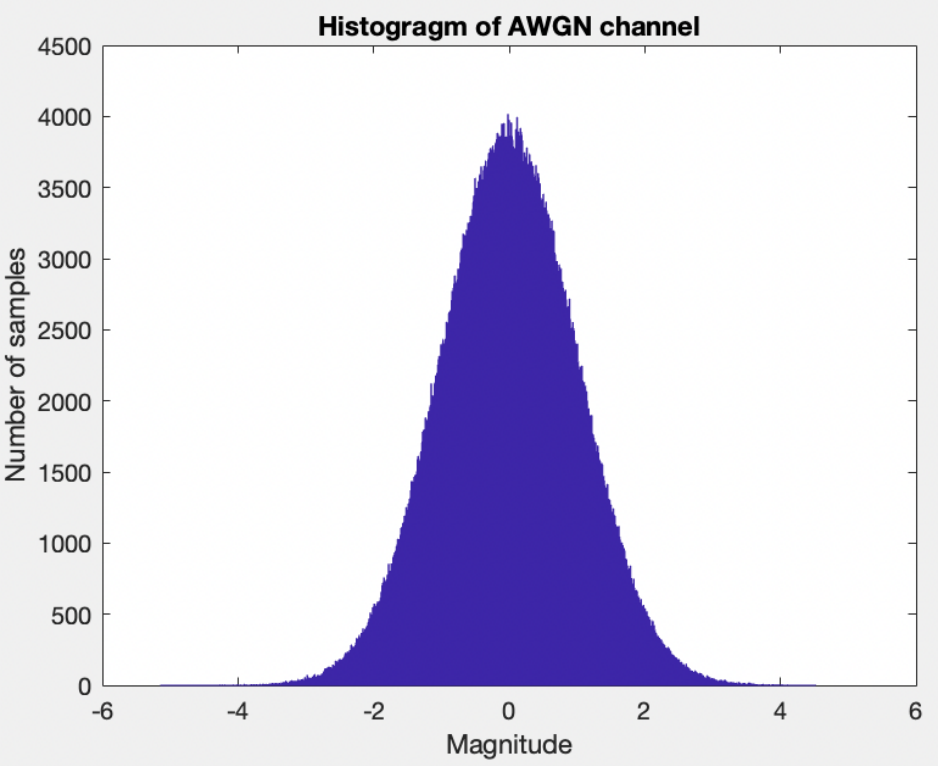


Figure 1.
AWGN histogram

3.2.3 BPSK encoding and decoding

The digital modulation scheme used in this implementation is binary phase-shift keying which conveys information by encoding the phase of the digital signal as either a one or a zero depending on the phase of the signal. BPSK is encoded using the following formula: $S = 1 - 2(r)$ where the symbol "S" is the encoded symbol and "r" is the received modulated bit (either a 1/0). This results in a symbol where a one is -1, and a zero is +1. This is one of the simplest digital modulation schemes and allows for a single message bit to be encoded as a single bit in the transmitted message. This is the most common digital encoding scheme used in order to test the efficacy of different error correction codes and it will be the only modulation scheme considered in this implementation.

3.2.4 BPSK in AWGN channel

When a BPSK message is decoded with a simple encoder that uses a decision boundary at 0 the following BER curve in figure 3 results. The BER can be seen improving as the signal-to-noise ratio of the AWGN channel improves. The goal of an encoding and decoding scheme is to cause the BER curve to drop off more steeply, resulting in a better BER at a more adverse signal-to-noise ratio.

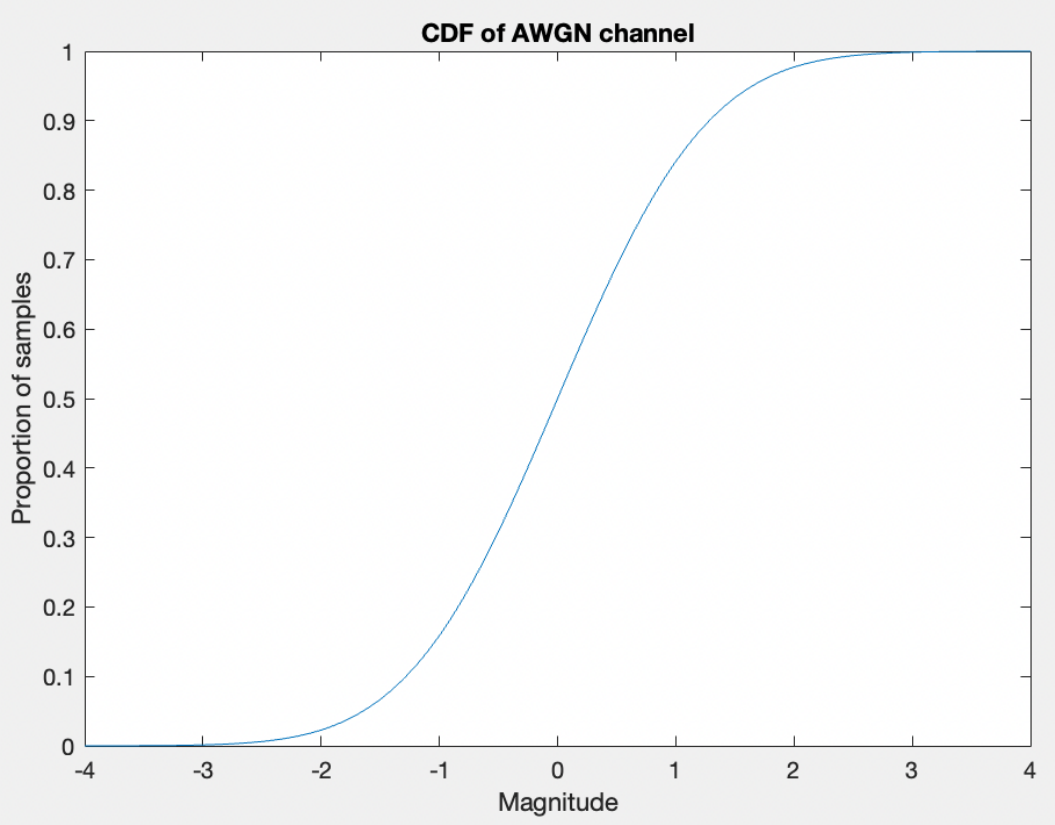


Figure 2.
AWGN CDF

The theoretical equation for the BER curve of an unencoded BPSK system in an AWGN channel is shown in equation 4, where x is the Eb/No of the channel. This equation is compared to a simulated BPSK system in an AWGN channel in figure 3. Where the Q-function is used in order to determine the proportion of noise values at a specific signal-to-noise ratio that will result in the BPSK signal being decoded incorrectly.

$$BER(x) = Q(\sqrt{\frac{2 \times Eb}{No}}) \tag{4}$$

3.2.5 Proportion of bits in error for BPSK in AWGN channel

The proportion of bits in error for a given amount of AWGN noise is shown in table 3.

3.2.6 Q-function in AWGN channel

The theoretical BER curve is drawn using the Q-function [9] in equation 4. The Q-function is defined as the function that takes the integral of the sum of the area under the curve of the PDF of the AWGN distribution for a certain magnitude value range. This value is used to

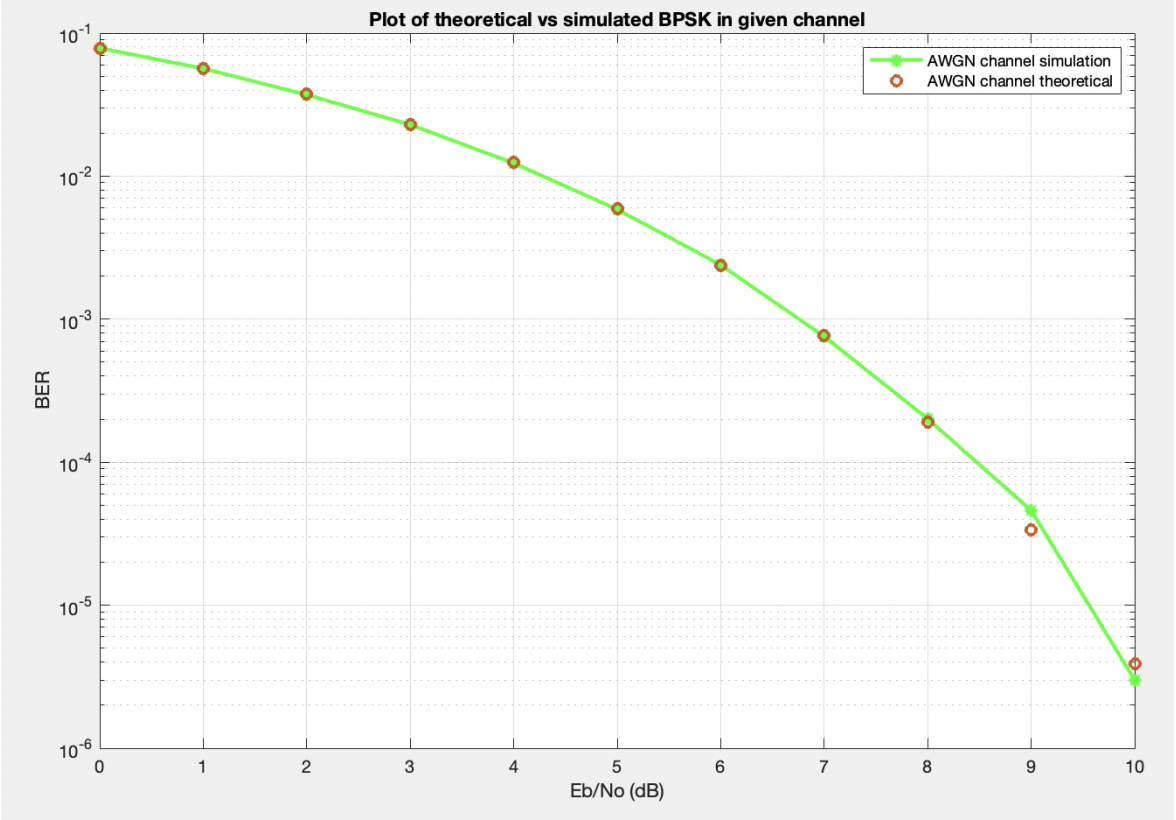


Figure 3.
BPSK decoded in an AWGN channel

determine what the total proportion of magnitude values for the given BPSK bit will cause the bit-value to cross over the zero threshold and thus cause a bit-flip. The integral is shown in equation 5. In this equation, it can be seen that the increasing sigma value will change the PDF of the AWGN distribution and thus increase the Q-function value and thus increase the likelihood of a bit-flip. A further simplification of the Q-function can be made since the form of the value inside the integral is a match for that of the complementary error function, 'erfc', shown in equation 6. This leads to the Q-function being rewritten in equation 7.

$$Q(x_o) = \int_{x_0}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} dx \tag{5}$$

$$erfc(x_o) = \frac{2}{\sqrt{\pi}} \int_{x_0}^{\infty} e^{-x^2} dx \tag{6}$$

$$Q(x_o) = \frac{1}{2} erfc(\frac{x_0}{\sqrt{2}}) \tag{7}$$

Table 3.
Proportion of bits in error

Eb/No (dB)	Proportion of bits in error (%)
1.00	19.8
1.50	18.3
1.85	17.4
2.00	17.0
2.25	16.6
2.50	15.7

3.2.7 The Rayleigh fading channel

This section elaborates on the explanation given in the literature study section and illustrates how the Rayleigh flat-fading channel model is created. The distribution of a Rayleigh fading channel is shown in this section to illustrate the noise that is added to the Rayleigh channel in the second case in which the project is implemented.

The PDF of the magnitude of the flat-fading Rayleigh fading channel is shown in figure 4, with the phase of the PDF shown in figure 5. The CDF of the flat-fading Rayleigh channel is shown in figure 6.

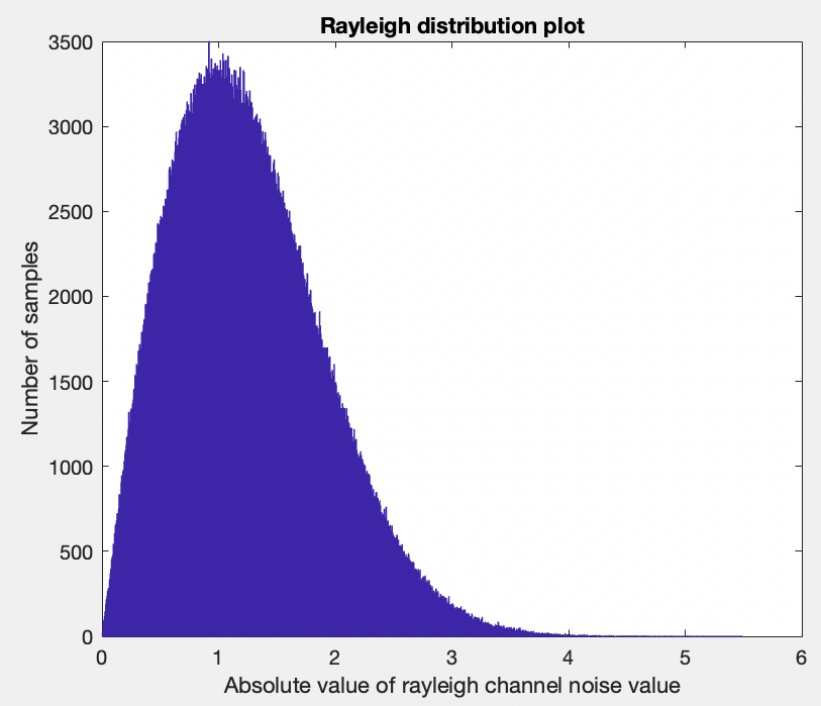


Figure 4.
Rayleigh histogram

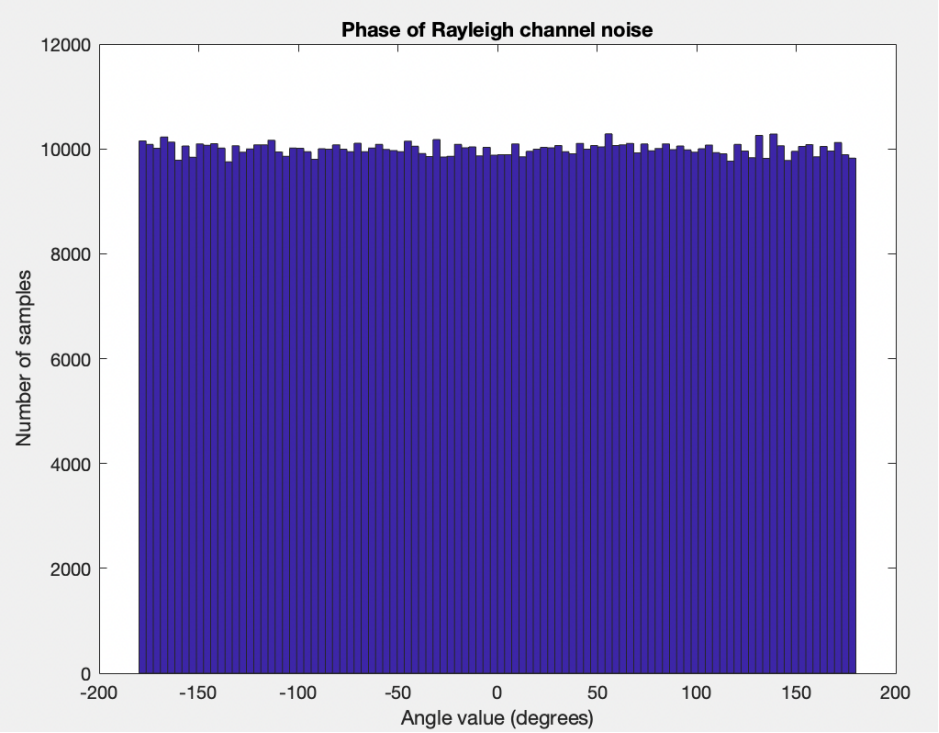


Figure 5.
Rayleigh phase distribution

3.2.8 Rayleigh channel equation

The equation of the flat-fading Rayleigh channel PDF is in equation 8. The equation for the CDF of the flat-fading Rayleigh channel is given in equation 9, where x is the Eb/No of the channel.

$$PDF(x,\sigma) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}} \tag{8}$$

Where: $x >= 0$

$$CDF(x,\sigma) = 1 - e^{\left(\frac{-x^2}{2\sigma^2}\right)} \tag{9}$$

3.2.9 BPSK in a Rayleigh flat-fading channel

The theoretical BER of a BPSK signal in a Rayleigh flat-fading channel is determined by taking the integral over the PDF of the Rayleigh channel and determining, as in the AWGN channel, what the proportion of magnitude of noise values that are able to cause a bit-flip that will cause the value of the decoded bit to cross over the BPSK zero threshold. The theoretical equation

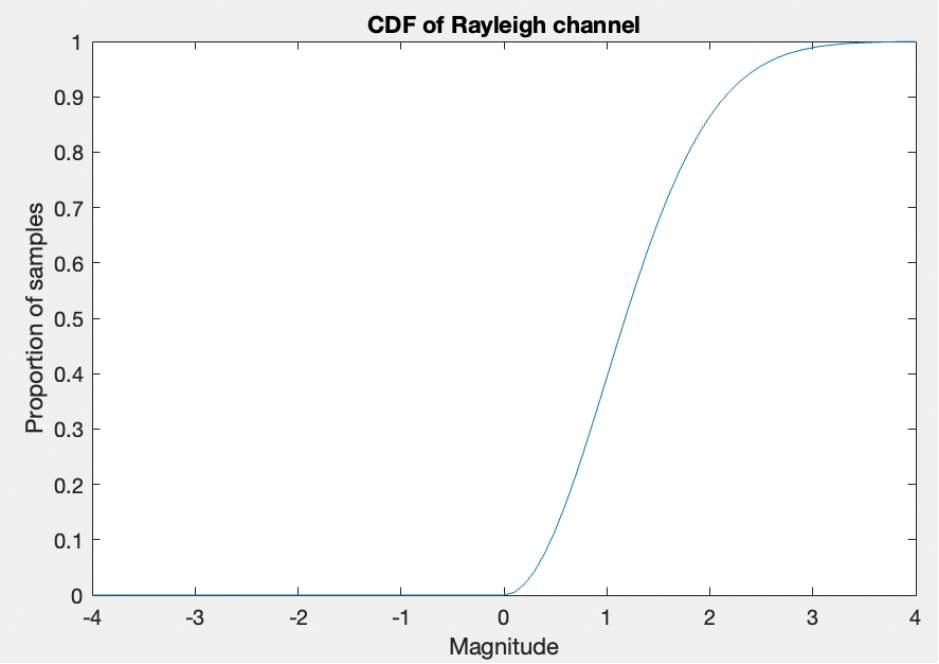


Figure 6.
Rayleigh CDF

for the BER is given in equation 10, where x is the Eb/No of the channel. The theoretical BER vs a simulation of the BER of the Rayleigh flat-fading channel is shown in figure 7, where it can be seen that the BER follows an almost linear slope. The BER of the Rayleigh channel only decreases by a small amount for each incremental increase (improvement) in the signal-to-noise ratio. The BER of the channel is 1×10^{-4} at approximately 34 dB of Eb/No.

This is a much higher Eb/No value than that of BPSK in an AWGN channel which provides a BER of 1×10^{-4} at approximately 8.5 dB of Eb/No. This means that the Rayleigh flat-fading channel is much more adverse than that of an AWGN channel since its BER performance is approximately 25.5 dB worse for a BPSK signal. A comparison of the BER of BPSK in a Rayleigh flat-fading channel vs AWGN channel is shown in figure 8.

$$BER(x) = \frac{1}{2} \times (1 - \sqrt{\frac{x}{x+1}}) \tag{10}$$

3.2.10 Proportion of bits in error for BPSK in Rayleigh flat-fading channel

The proportion of bits in error for a given amount of Rayleigh flat-fading channel noise is shown in table 4.

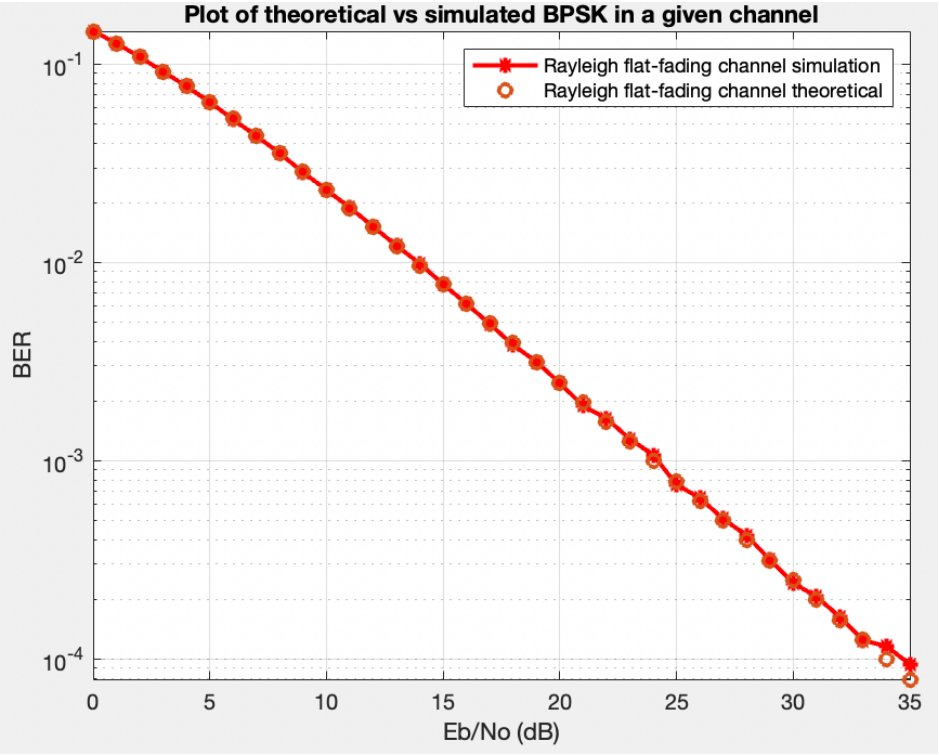


Figure 7.
BPSK in a Rayleigh flat-fading channel

Table 4.
Proportion of bits in error

Eb/No (dB)	Proportion of bits in error (%)
5	14.449
10	6.350
15	2.268
20	0.758
23	0.396
25	0.251
30	0.070

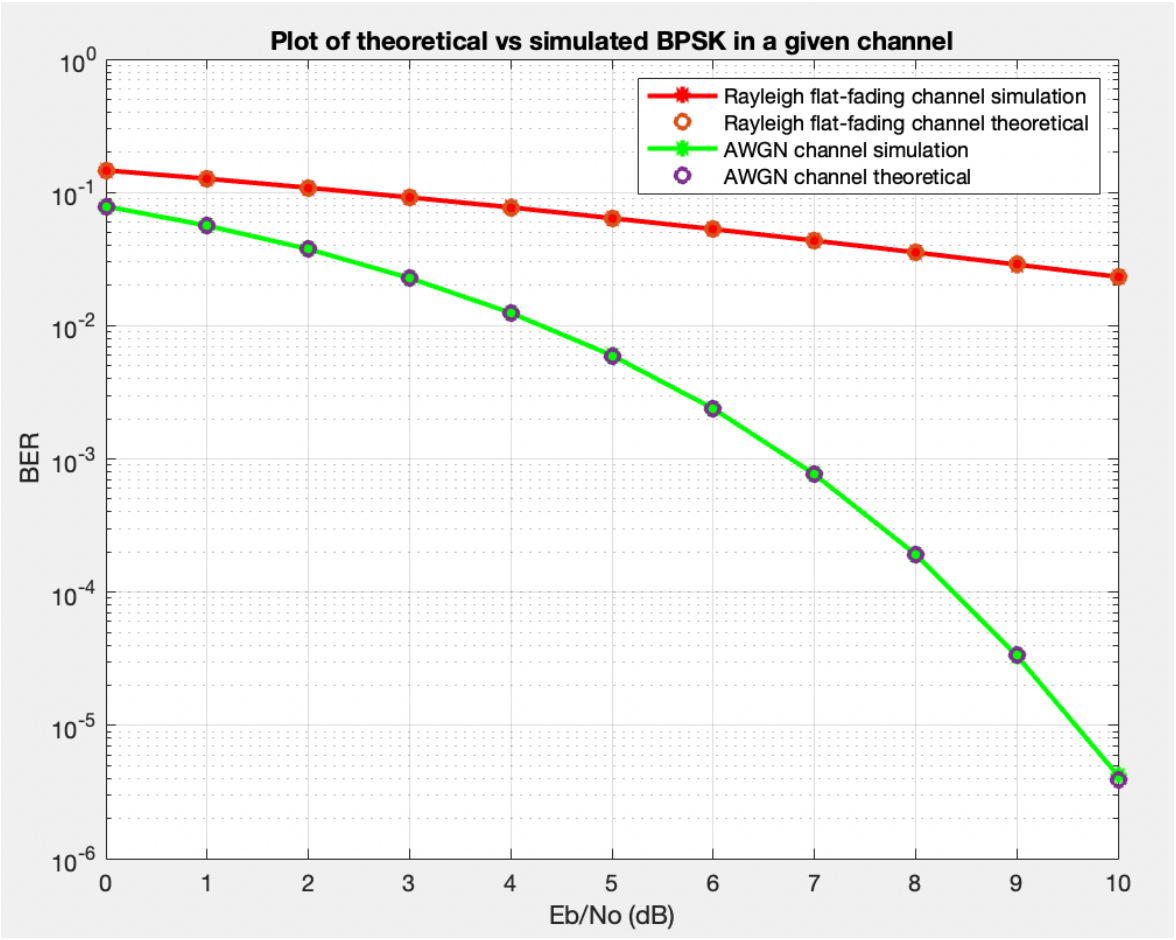


Figure 8.
BPSK in a Rayleigh flat-fading channel vs AWGN channel

3.3 Probabilistic decoding

3.3.1 Log-likelihood ratio (LLR) in AWGN channel

Probabilistic decoding algorithms work to maximize the probability that the decoded bit is the same as the transmitted bit. This is done by determining the probability of a bit in error using Bayes theorem in equation 11 where f is the PDF of AWGN. Similarly, in equation 12 the probability of a correct bit is determined. These two equations are combined in equation 14. Portions of the following derivation are taken from derivations in [1].

$$P(c=0|r=1)=\frac{f(r|c=0)P(c=0)}{f(r)}=\frac{f(r|c=0)(1/2)}{f(r)} \tag{11}$$

$$P(c=1|r=1)=\frac{f(r|c=1)P(c=1)}{f(r)}=\frac{f(r|c=1)(1/2)}{f(r)} \tag{12}$$

This is done by minimizing the log-likelihood ratio of the decoded message in equation 14. This ensures that the "belief" of the message "c" is maximized to the most likely state of 1's and 0's.

$$LLR=ln(\frac{P(c=0|r=1)}{P(c=1|r=1)})=ln(\frac{f(r|c=0)}{f(r|c=1)}) \tag{13}$$

The received value "r" is the value of the bit received plus AWGN: $r=1+N(0,\sigma^2)$. From this the following expansion can be made by substituting $r-1$ for $c=0$, and substituting $r+1$ for $c=1$.

$$\frac{P(c=0|r=1)}{P(c=1|r=1)}=\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-(r-1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-(r+1)^2}{2\sigma^2}}}=e^{(\frac{2r}{\sigma^2})} \tag{14}$$

This log-likelihood ratio for a BPSK symbol simplifies to the following in equation 15. Where "r" is the received bit and sigma is the variance of the noise of the channel. This is 1.0 for an AWGN channel.

$$LLR=ln(e^{(\frac{2r}{\sigma^2})})=\frac{2r}{\sigma^2} \tag{15}$$

This formula is the "intrinsic" belief of the AWGN channel over which the signal is passed.

3.3.2 Intrinsic beliefs

The intrinsic belief of the channel is the probability that the received symbol is either a 1 or a 0 depending on the information that the noisy AWGN channel imparts on the symbol. This means that without encoding a symbol there is already an "intrinsic" probability that the symbol is correct when given the noise energy of the channel that is added to the symbol. This probability of a correct symbol decreases as the energy of the noise (σ) is increased.

Table 5.
LLR of different Eb/No values

Eb/No (dB)	σ value	Log-likelihood ratio
10	0.2236	40.0
9	0.2509	31.7
8	0.2815	25.2
7	0.3159	20.0
6	0.3544	15.9
5	0.3976	12.6
4	0.4461	10.0
3	0.5006	8.0
2	0.5617	6.3
1	0.6302	5.0
0	0.7071	4.0

The log-likelihood of a correct symbol in a noisy channel is shown in table 5.

3.3.3 Extrinsic beliefs

The extrinsic belief of a particular bit is the belief about that symbol that is obtained through encoding. This value will change depending on the encoding scheme and the number of bits that are used to encode the particular bit. In colloquial terms, the extrinsic belief of a certain bit can be seen as being "what the other bits believe about it". For a simple single parity check code with 3 bits: C1, C2, C3, where C1 is the parity bit for C2 and C3, the following setup can be used in order to determine the extrinsic LLR of the bit. The following probabilities P1-P3 are defined in equation 16 to 18, as the probability that a specific bit is zero.

$$P1 = P(C1 = 0|C2, C3) \tag{16}$$

$$P2 = P(C2 = 0|C1, C3) \tag{17}$$

$$P3 = P(C3 = 0|C1, C2) \tag{18}$$

The 4 cases for C1 are given in table 6:

From table 6 the equations 19 and 20 can be set up. In equation 19, P1 is the product of the probabilities that C2 and C3 are zero, plus the probability that they are both one. In equation 20, the probability that C1 is one is the probability that C2 is zero and C3 is one, plus the probability that C2 is one and C3 is zero. From equations 19, 20 and 22 can be set up in order to find the extrinsic LLR for C1 by subtracting the probability that C1 is one from the probability that C1 is zero.

Table 6.
C1 values for values of C2 and C3

C1	C2	C3
0	0	0
0	1	1
1	0	1
1	1	0

$$P1 = P2 \cdot P3 + (1-P2)(1-P3) \tag{19}$$

$$1-P1 = P2(1-P3) + (1-P2)(P3) \tag{20}$$

$$P1-(1-P1) = (P2 \cdot P3 + (1-P2)(1-P3)) - (P2(1-P3) + (1-P2)(P3)) \tag{21}$$

This simplifies to the following equation:

$$P1-(1-P1) = (P2-(1-P2))(P3-(1-P3)) \tag{22}$$

From here the LLR is calculated as in equation 14.

$$\frac{P1-(1-P1)}{P1+(1-P1)} = \frac{(P2-(1-P2))(P3-(1-P3))}{(P2+(1-P2))(P3+(1-P3))} \tag{23}$$

$$\frac{1-\frac{(1-P1)}{P1}}{1+\frac{(1-P1)}{P1}} = \frac{1-\frac{(1-P2)}{P2}}{1+\frac{(1-P2)}{P2}} \cdot \frac{1-\frac{(1-P3)}{P3}}{1+\frac{(1-P3)}{P3}} \tag{24}$$

In equation 24 the following substitution in equation 26 can be made in order to find the extrinsic LLR of the first bit L1. The substitution leads to the following form of equation 24 as shown in equation 27.

$$l_{ext,1} = ln(\frac{P1}{1-P1}) \tag{25}$$

$$e^{-l_{ext,1}} = \frac{1-P1}{P1} \tag{26}$$

$$\frac{1-e^{-l_{ext,1}}}{1+e^{-l_{ext,1}}} = \frac{1-e^{-l_2}}{1+e^{-l_2}} \cdot \frac{1-e^{-l_3}}{1+e^{-l_3}} \tag{27}$$

Using the hyperbolic tangent substitution in equation 28 leads to the following form in equation 29.

$$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{28}$$

$$tanh(\frac{l_{ext,1}}{2}) = tanh(\frac{l_2}{2}) \cdot tanh(\frac{l_3}{2}) \tag{29}$$

Equation 29 is a common form of a bit-wise XOR. Therefore, this equation makes intuitive sense since it is the XOR of 2 bits (C2 XOR C3) where C1 is the result.

3.3.4 Simplification of LLR calculation

Since the tanh function is an odd function, for negative values of x the tanh value is simply the inverse of the positive x value. Therefore, the function can be split up into equations 30 and 31 that are evaluated separately.

$$sgn(l_{ext,1}) = sgn(l_2) \cdot sgn(l_3) \tag{30}$$

$$tanh(|\frac{l_{ext,1}}{2}|) = tanh(|\frac{l_2}{2}|) \cdot tanh(|\frac{l_3}{2}|) \tag{31}$$

This equation is equivalent to the following equation 32

$$ln(tanh(|\frac{l_{ext,1}}{2}|)) = ln(tanh(|\frac{l_2}{2}|)) + ln(tanh(|\frac{l_3}{2}|)) \tag{32}$$

To aid brevity, a function 'f(x)' is defined in equation 33 to be used in further calculations. Where the function is defined for $x > 0$ and where $f^{-1}(x) = f(x)$.

$$f(x) = |ln(tanh(|\frac{x}{2}|))| \tag{33}$$

The function f(x) is shown in figure 9 where it can be seen that the smaller the absolute value of the input is the larger the output value "y". Therefore it shows that the smallest value in the min-sum equation will dominate the output of the output.

With the f(x) applied equation 32 turns into equation 34.

$$f(|l_{ext,1}|) = f(|l_2|) + f(|l_3|) \tag{34}$$

Therefore obtaining the extrinsic LLR is achieved by evaluating equations 35 and 36

$$|l_{ext,1}| = f(f(|l_2|) + f(|l_3|)) \tag{35}$$
$$sgn(l_{ext,1}) = sgn(l_2) \cdot sgn(l_3) \tag{36}$$

3.3.5 Min-sum approximation

For equation 36 there is an approximation that has been shown to be very accurate due to the asymptotic nature of the ln(tanh(x)) function [1]. The approximation states that the addition of two numbers that are approaching the vertical asymptote will be dominated by the answer

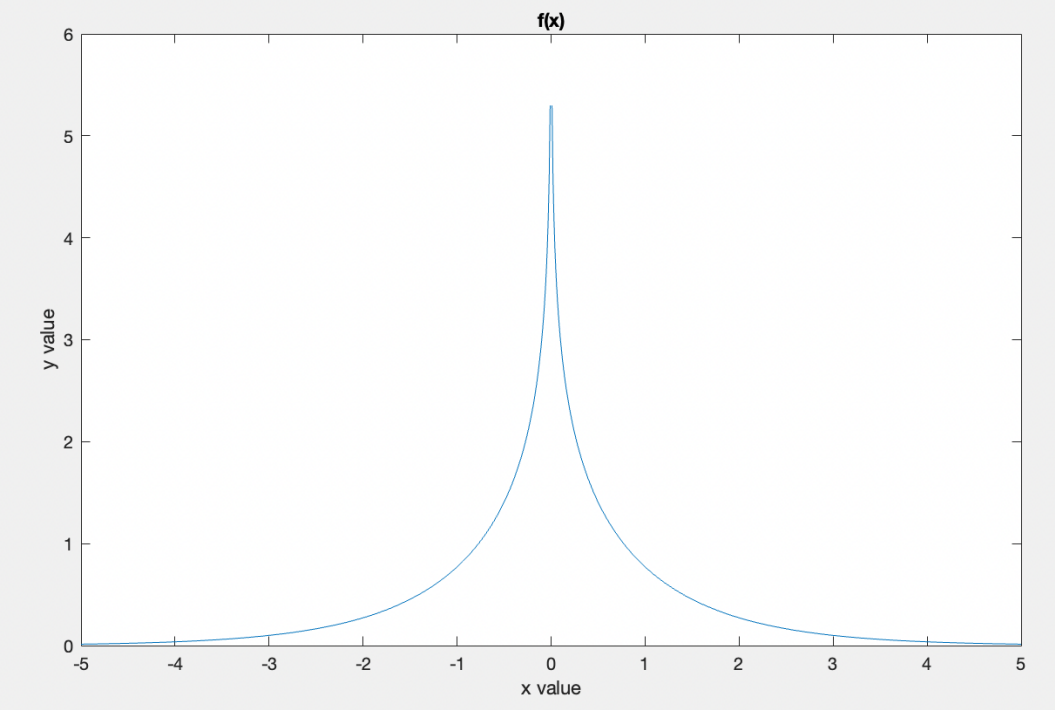


Figure 9.
f(x) function

obtained from calculating the function with the smaller x value. This is demonstrated in equation 37 and used to approximate the extrinsic belief of C1 in equation 38.

$$f(l_2) + f(l_3) \approx f(min(|l2|, |l3|)) \tag{37}$$

$$|l_{ext,1}| \approx f(f(min(|l2|, |l3|))) \approx min(|l2|, |l3|) \tag{38}$$

The full approximation of the extrinsic belief of each bit is calculated by doing the min-sum approximation for each of the bits. This leads to the absolute value of each extrinsic belief being the value obtained by using equation 38. This results in each bit, except the bit with the smallest absolute value, being assigned the value of min($|C_1|$...$|C_N|$). The bit with the smallest absolute value is assigned to the second smallest absolute value of all the bits.

The total extrinsic belief for each bit is then its original sign (+/-) multiplied by the product of all the signs (equation 36) and the absolute value of its extrinsic belief calculated in equation 38.

3.3.6 Message-passing decoding

LDPC codes can be decoded using probabilistic decoding algorithms such as iterative message passing decoding [12] [13]. This allows for near-optimal noise performance and allows messages to be decoded such that the decoder can come close to Shannon's limit for the given

code rate.

This section elaborates on the explanation given in the literature study section and illustrates how the message-passing algorithm operates.

The essence of the message-passing algorithm is that each bit passes a message containing its belief about its neighbouring bits. This message is passed to neighbouring bits and used as information to create beliefs that can be used to repeat the cycle for as many bits as are in the check-bit-connected group. This message-passing cycle is repeated for a set number of iterations or until the beliefs for each bit reach a near-optimal state. The process is demonstrated in figure 10.

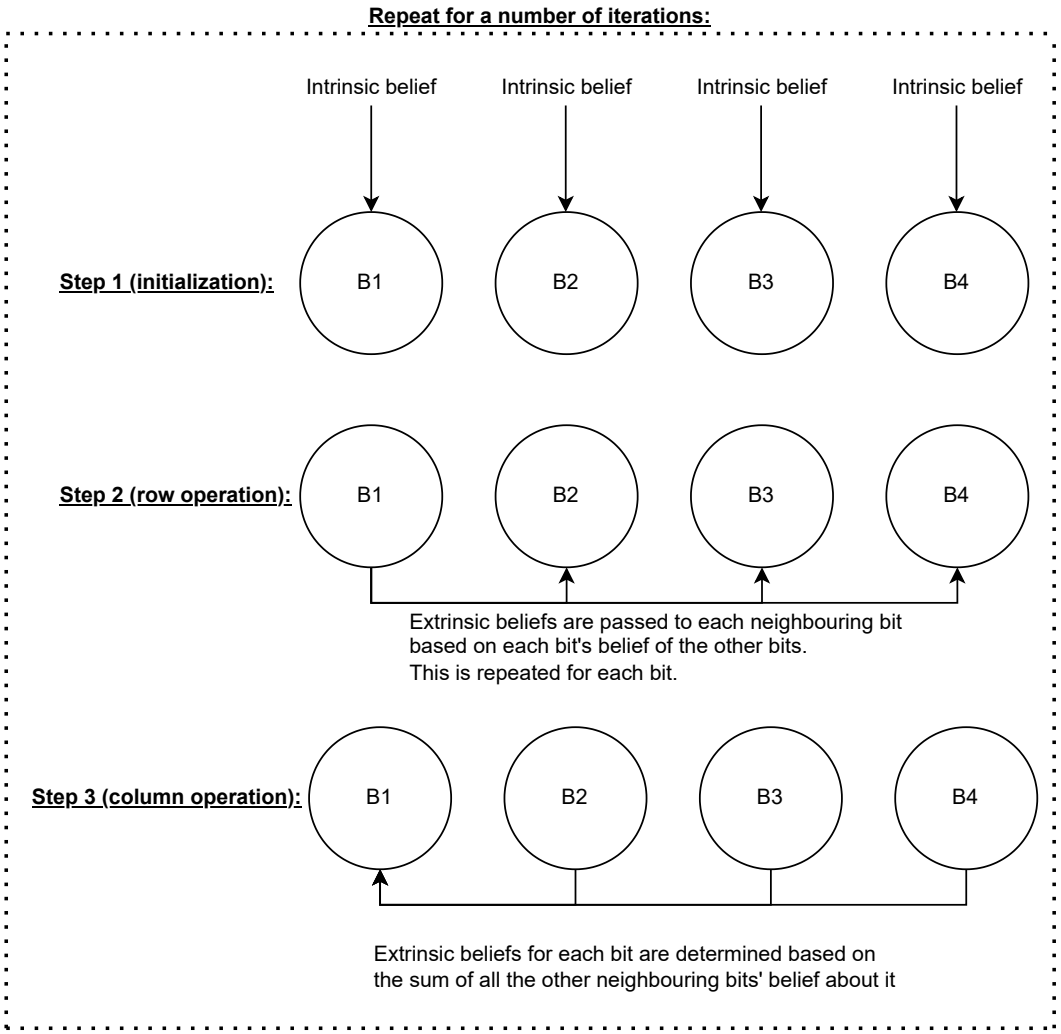


Figure 10.
Message-passing steps

3.4 Channel model simulation implementation

3.4.1 Implementation of Additive White Gaussian Noise (AWGN)

The implementation of AWGN noise on a BPSK signal is done by simply adding a scaled normally-distributed random float to the BPSK symbol. The scaling factor σ is used to scale the noise to a certain Eb/No level. The scaling factor σ is shown in equation 39. Where R is the code rate of the system and Eb/No is the signal-to-noise ratio of the system, Eb/No is calculated from the desired Eb/No (dB) value in equation 40.

$$\sigma = \sqrt{\frac{1}{2R \times Eb/No}} \tag{39}$$

$$Eb/No = 10^{(\frac{Eb/No(dB)}{10})} \tag{40}$$

3.4.2 Implementation of Rayleigh channel

The Rayleigh flat-fading channel is implemented by multiplying $\frac{1}{\sqrt{2}} \times (1 + j)$ by a normally-distributed random variable [14]. The result of this represents a single tap flat-fading channel which can be used for simulating the performance of the system. The second step is to create a model of scaled noise in a Rayleigh flat-fading channel. This is achieved by taking $(1 + j)$ and multiplying it by a normally-distributed random variable as in the first step and then multiplying it by the σ scaling factor from equation 39. The Rayleigh channel model is then completed by multiplying the BPSK symbol by the channel representation in step 1 and adding the scaled noise in step 2 and finally dividing the whole expression by the fading channel representation in step 1. The real part of the result is then obtained, which results in a float value that is sent through the decoder. The pseudo-code for the process is shown in algorithm 1. This code for a Rayleigh flat-fading channel model is similar to that shown in [14].

Algorithm 1 Rayleigh flat-fading channel pseudo code

1: channel_model = $\frac{1}{\sqrt{2}} \times (1 + j) \times$(normally distributed random variable)
2: noise = $\sigma \times (1 + j) \times$(normally distributed random variable)
3: value = channel_model$\times$(BPSK symbol) + noise
4: value = value/channel_model
5: final_symbol_value = real(value)

3.5 Basic encoder algorithm

3.5.1 The 5G-NR standard

The 5G-NR standard [15] is a standard that has been developed to govern how error-correcting codes are implemented for 5G and high-speed mobile applications. The standard used in this implementation of LDPC codes is used to create the parity check matrix that controls the check bit positions and the bits that each check bit protects.

The 5G-NR standard uses base matrices that govern the bit-to-check-bit relationship to create parity check matrices [16]. These check matrices are implemented with the use of "base matrices". These base matrices allow for the given parity check matrices to be applied for codes of both different codeword lengths and also different code rates, whilst maintaining the same desirable bit-to-check-bit relationships. Different code rates can be implemented simply by trimming the number of columns and rows in the base matrix that is used. Decreasing the number of columns and rows in the base matrix that are used increases the code rate. This is favourable since it allows users to apply the same type of encoding relationship for several different types of codes. The base matrices can be found online that apply to certain code rates and also codeword lengths. The so-called base matrices have certain expansion factors, Zc, which are used to take the general base matrices which are 46×68 matrices of numbers where each index in the base matrix stores a number from -1 to $(Zc - 1)$. Each index in the base matrix is used to represent another smaller identity matrix of dimensions $Zc \times Zc$ that is cyclically right-shifted by the number indicated in the base matrix. In the case of -1, the matrix is filled with zeros. This is what allows for the scaling of the parity check matrix to one that is applicable for the desired codeword length. Identity matrices are used in order to illustrate the cyclic right shifting. This effect can be achieved similarly by cyclically right-shifting the codeword in an optimized encoder/decoder algorithm.

The base matrices used in the following implementation are for implementations of expansion factors Zc=30, 44, 60, 72 which result in codewords of length N=2040, 2992, 4080, 4896. The code rate considered in this implementation is R=0.3235 as stated in section 1.

3.5.2 Double-diagonal encoding

Encoding using a base matrix from the 5G-NR starts with encoding using an ingenious parity check matrix structure commonly referred to as a "double-diagonal" structure [17]. This matrix is called the 'E-matrix' in the base matrix structure. This structure is included in each base matrix in the 5G-NR standard. The E-matrix is shown in figure 11. The double diagonal structure allows for the first 4 groups of parity bits in the parity check matrix to be calculated. The -1's correspond to zero matrices, the zeros correspond to identity matrices and the other numbers correspond to cyclically shifted matrices, of dimensions $Zc \times Zc$ as stated previously.

	P1	P2	P3	P4
Message bits 0-k	2	0	-1	-1
Message bits 0-k	-1	0	0	-1
Message bits 0-k	1	-1	0	0
Message bits 0-k	2	-1	-1	0

Figure 11.
E-matrix double diagonal structure

3.5.3 Encoding example

The decoding process can be demonstrated by setting up an example (24,12) code with Zc=3. The encoding process begins by setting up the equations from the example in figure 12. All numerical operations are mod2.

$$M_1 i_1 + M_3 i_2 + M_4 i_1 + P1 i_2 + P2 i_0 = 0 \tag{41}$$

$$M_1 i_2 + M_2 i_0 + M_4 i_2 + P2 i_0 + P3 i_0 = 0 \tag{42}$$

$$M_2 i_2 + M_3 i_1 + M_4 i_0 + P1 i_1 + P3 i_0 + P4 i_0 = 0 \tag{43}$$

$$M_1 i_2 + M_2 i_1 + M_3 i_0 + P1 i_2 + P4 i_0 = 0 \tag{44}$$

Equations 41 to 44 are combined in equation 45, where like terms cancel out due to mod2 addition. Equation 45 gives the result of P1 which is used to solve for P2 and P4 in equation 41 and equation 44. P2 can then be used in order to solve for P3 in equation 42. The expanded versions of figure 13 with all ones substituted in place are shown in figure 14. The expanded version with bit names indicated is shown in figure 15. The parity bits P1 to P4 are then used in larger codes to calculate the rest of the parity bits P5-P_N.

$$P1 i_1 = M_1(i_1) + M_2(i_0 + i_1 + i_2) + M_3(i_0 + i_1 + i_2) + M_4(i_0 + i_1 + i_2) \tag{45}$$

				P1	P2	P3	P4
1	-1	2	1	2	0	-1	-1
2	0	-1	2	-1	0	0	-1
-1	2	1	0	1	-1	0	0
2	1	0	-1	2	-1	-1	0

Figure 12.
base matrix example with double diagonal structure highlighted

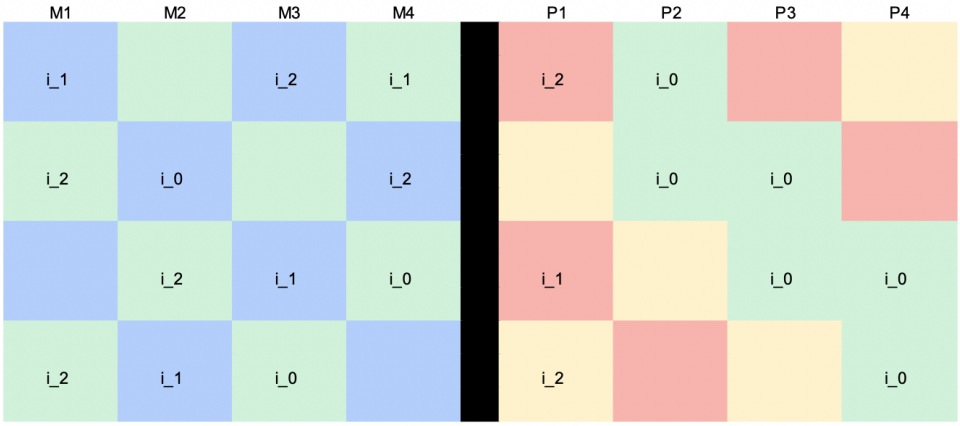


Figure 13.
Zc=3 expansion of parity check matrix with double diagonal structure highlighted ('i_x' stands for identity matrix shifted by x units)

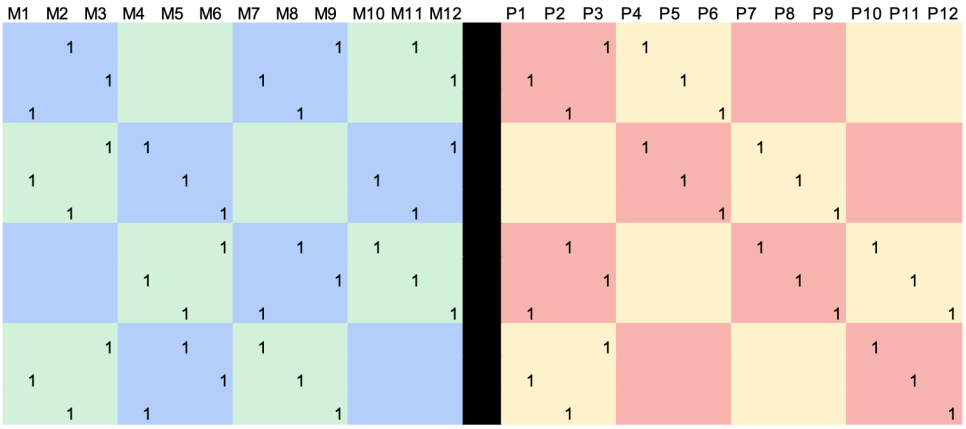


Figure 14.
Expansion of E-matrix with ones shown (zeros are omitted)

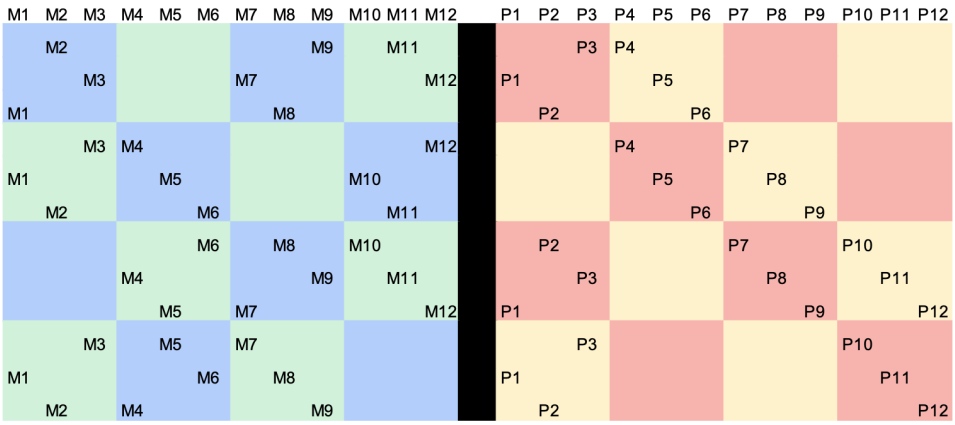


Figure 15.
Expansion of E-matrix with bit names shown (zeros are omitted)

3.5.4 Unoptimized encoder algorithm

The basic encoder algorithm is shown in the pseudo-code in algorithm 2. This encoder is designed 100% from first principles by the author. The basic encoder starts with the first step where the base matrix is fully expanded using the expansion factor Zc, to create the full base matrix. The second step is to make use of the double diagonal structure shown in figure 11 by encoding the first parity bits P1. P1 is a row matrix of Zc number of elements. P1's values are calculated as the mod2 sum of all the other bit values in the parity matrix in the corresponding expanded row. This is shown mathematically in equation 45. After P1's values are calculated the values for P1 are substituted into its other places down the column in the parity matrix.

The following step is to calculate P2's values. P2's values are determined by doing the same mod2 summation as in P1, but the value of P1 is now known and used to determine the value of P2. Then P2's values are similarly substituted into their corresponding positions in the parity matrix. After P2 is determined the values for P3 can be determined by using the values of P2. After P3 is determined its values are substituted into their corresponding positions in the parity matrix. P4's values are then finally determined in the same manner as with P1-P3 and substituted into the corresponding positions in the parity matrix.

After P1-P4 have been calculated then the E-matrix portion of the parity matrix has been successfully encoded. The parity bits P5 to P46 are then determined one at a time using the values of P1 to P4 and substituted into their corresponding positions in the parity matrix.

The message is then considered to be fully encoded and the values of P1 to P46 can be concatenated to the end of the message to create the fully encoded source message to be sent over a communication channel.

The problem with this encoding algorithm is that it does not take any advantage of the inherent brevity available when using the base matrix structured decoder, which is already far superior to a self-made, case-specific parity matrix. In this implementation, the entirety of the parity matrix is full of zeros that have to be stored even though they are effectively just placeholders for nothing that keep the sparse indexes of the parity matrix in place. This not only wastes vast amounts of memory in storing useless information but also creates a massive amount of run-time overhead because the system is forced to iterate meaninglessly over strings of zeros to get to the useful data in the parity matrix. This causes excessive run times in addition to memory overhead.

A run-time optimized implementation would need the following properties in order to be deemed effective and optimal in the case of the encoder:

- Don't expand the base matrix and store the full expanded base matrix.
- Make use of the base matrix implementation by doing in-place calculations.
- Store the minimum amount of extra data possible in matrices so as to keep memory usage low.
- Reuse temporary storage for each in-place operation.

Algorithm 2 Unoptimized encoder algorithm

```
Expand base matrix into full parity check matrix based on expansion factor Zc.
num_rows = 46*Zc
num_cols = 68*Zc
for i = 0:Zc do
    for j = 0:22*Zc do
        P1 = P1 mod2 all message bits at (i,j)
        P1 = P1 mod2 all message bits at (3*Zc+i,j)
    end for
end for
Substitute P1 into all its positions in all rows of the parity matrix
for i = 0:Zc do
    for j = 0:23*Zc do
        P2 = P2 mod2 all message bits at (i,j)
    end for
end for
Substitute P2 into all its positions in all rows of the parity matrix
for i = 0:Zc do
    for j = 0:24*Zc do
        P3 = P3 mod2 all message bits at (Zc+i,j)
    end for
end for
for i = 0:Zc do
    for j = 0:Zc do
        P3 = P3 mod2 parity matrix values at (Zc+i,23*Zc+j)
    end for
end for
Substitute P3 into all its positions in all rows of the parity matrix
for i = 0:Zc do
    for j = 0:23*Zc do
        P4 = P4 mod2 parity matrix values at (3*Zc+i,j)
    end for
end for
Substitute P4 into all its positions in all rows of the parity matrix
for k = 5:46 do
    for i = 0:Zc do
        for j = 0:26*Zc do
            P_k = P_k mod2 parity matrix values at (k*Zc+i,j)
        end for
    end for
end for
```

3.5.5 Run-time optimized encoder algorithm

The run-time optimized encoder and decoder will make use of a cyclic shifting function shown in algorithm 3's pseudo code. This is then used in the optimized encoder and decoder pseudo code in algorithms 4 and 8. The cyclic right shifting is applied to the codeword instead of the parity check matrix as in the unoptimized encoder. This allows the shifting to be applied to a single-row array only, instead of a full two-dimensional parity matrix of dimensions $Zc \times Zc$.

The run-time optimized algorithm shown in algorithm 4 is proposed in [1] and is adapted and optimized for the current implementation. This implementation is used to improve the usage of in-place operations to save on memory usage and run-time. The algorithm takes full advantage of the base matrix's inherent brevity by making use of in-place calculations. The base matrix is therefore not fully expanded and the system stores only the base matrix and a matrix of maximum size $Z_c \times Z_c$. Temporary storage is also reused in the implementation and therefore keeps storage down even further.

The algorithm follows the same basic steps and order of operations as the unoptimized implementation but just does it in a way that minimizes memory usage and also CPU cycles by skipping all base matrix zeros automatically. The optimized implementation circularly shifts the message matrix to obtain the same effect as though an expanded base matrix were implemented and shifted circularly. The idea of shifting the message matrix instead of shifting an expanded base matrix allows for brevity and in-place operations on one-dimensional arrays instead of using two-dimensional matrices that have to be traversed over in a complex method.

Algorithm 3 Shift_mat algorithm

```
1:  parameter1 = x (row matrix to shift)
2:  parameter2 = k (base matrix value that indicates how many indexes to shift by)
3:  if k = -1 then
4:      replace all the values in x by 0's
5:      return x
6:  else
7:      output = empty row matrix of same size as x
8:      output(0:k) = x(k:end)
9:      output(k:end) = x(0:k)
10:     return output
11: end if
```

Algorithm 4 Run-time optimized encoder algorithm

```
parity_array = row matrix of zeros of length Zc
for x = 0:4 do
    for y = 0:22 do
        shifted_array = shift_mat(message[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
end for
codeword[22*Zc:23*Zc] = shift_mat(parity_array, Zc)
find P2 to P4 below
for x = 0:3 do
    Reassign parity_array matrix's values to zero
    for y = 0:23+x do
        shifted_array = shift_mat(codeword[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
    codeword[(23+x)*Zc:(24+x)*Zc] = parity_array
end for
find P5 to P46 below
for x = 4:46 do
    Reassign parity_array matrix's values to zero
    for y = 0:26 do
        shifted_array = shift_mat(codeword[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
    codeword[(22+x)*Zc:(23+x)*Zc] = parity_array
end for
```

3.5.6 Testing codeword validity

The validity of a codeword can be tested by multiplying the codeword with the parity check matrix. If the result of the calculation gives an all-zeros column matrix then the codeword is valid and correctly encoded. This is shown in algorithm 5's pseudo code.

If any value in the syndrome is not = 0 then there must be an error in the encoding of the message and thus the codeword is invalid. If the encoding process is completed correctly as in the preceding pseudo code in algorithm 4 then the syndrome will always be full of only zeros, and the encoding process has been completed correctly.

Algorithm 5 Codeword validity testing algorithm

```
1: for i = repeat for all rows do
2:     syndrome(i) = expanded base matrix(i) × (Transpose of full codeword)
3: end for
4: for repeat for all rows in syndrome do
5:     if value in syndrome != 0 then
6:         codeword is invalid
7:     end if
8: end for
```

3.6 Basic decoder algorithm

3.6.1 Unoptimized decoder algorithm

The basic decoder algorithm in algorithms 6 and 7, suffers from the same issues inherent in the basic encoder in algorithm 2. The decoder does not take any advantage of the inherent brevity available when using the base matrix structured decoder [18]. This decoder is designed 100% from first principles by the author in Matlab. In this implementation, the entirety of the parity matrix is also fully expanded before the decoding process begins, which makes the parity matrix full of place-holder zeros. This once again, as in the basic encoder implementation, wastes vast amounts of memory to store useless information and creates an even more massive amount of run-time overhead because the system is forced to iterate meaninglessly over these strings of zeros for every step of the min-sum operation and also repeat that same iterative process for every iteration that the decoder must do until it reaches its optimal state. This run-time and memory overhead must be avoided by making use of the base matrix structure to do in-place operations.

A run-time optimized implementation would need the following properties in order to be deemed effective and optimal in the case of the decoder's run-time performance. These properties are the same as those in the encoder's run-time optimization:

- Don't expand the base matrix and store the full expanded base matrix.

- Make use of the base matrix implementation by doing in-place calculations.
- Store the minimum amount of extra data possible in matrices so as to keep memory usage low.
- Reuse temporary storage for each in-place operation.
- Layering must be added to improve the rate of convergence on the optimal state.

Algorithm 6 Unoptimized decoder algorithm part 1

```
Expand base matrix into full parity check matrix based on expansion factor Zc.
num_rows = 46*Zc
num_cols = 68*Zc
L = Expand base matrix into full parity matrix and store it
r = received initial beliefs for message bits
for each row in L do
    for each column in L do
        Replace all ones in L with corresponding 'r' values
    end for
end for
for itr = 0:number of iterations do
    for i = each row in L do
        signs = row matrix to store the signs of each element
        parity = value to store the parity of the entire row
        min1 = abs(smallest value in row)
        pos = position of smallest value in row
        min2 = abs(second smallest value in row)
        for j = each column in L do
            signs(j) = +/- sign of L(i,j)
            parity = parity * sign of L(i,j)
            find min1 value and position in row
            find min2 value in row
        end for
        make all values in row = min1
        make value at pos in row = min2
        multiply every element in row with parity value and signs matrix
    end for
    sum = row matrix to store the sum of all columns
    for i = each row in L do
        for j = each column in L do
            sum(j) = sum(j) + L(i,j)
        end for
    end for
    for i = each row in L do
        for j = each column in L do
            L(i,j) = sum(j) - L(i,j)
        end for
    end for
end for
```

Algorithm 7 Unoptimized decoder algorithm part 2

```
1: final belief = row matrix to store the final value after decoding
2: for i = each row in L do
3:     for j = each column in L do
4:         final belief(j) = final belief(j) + L(i,j)
5:     end for
6: end for
7: decisions = row matrix to store final bit values after decoding
8: for j = every column in L do
9:     if L(j) >= 0 then
10:        decisions(j) = 0
11:    else
12:        decisions(j) = 1
13:    end if
14: end for
15: final bits = decisions
```

3.6.2 Run-time optimized decoder algorithm

The run-time optimized decoder algorithm is shown in the flow chart in figure 16 and algorithm 8. The flow chart shows the flow of the code and the implementation of the code with the 'offset min-sum' algorithm implemented. The 'offset min-sum' implementation is discussed in section 3.7.3. The cyclic right shifting is applied to the codeword instead of the parity check matrix as in the unoptimized decoder. This allows the shifting to be applied to a single row array only, instead of a full two-dimensional parity matrix of dimensions $Zc \times Zc$.

The decoder is made up of 2 outer loops that handle the layering and the repetition of iterations, whilst the 3 inner loops control the segmenting of data for the min-sum algorithm, implementing the min-sum algorithm and placing the data back in the beliefs matrix, respectively. The basic structure of the first and third inner loops that fetch the data, allow in-place operations, and place the data back in the beliefs array is proposed in [1]. This is heavily adapted and further optimized for the current implementation.

Algorithm 16 shows the complex nature of fetching data, executing the min-sum algorithm, and placing data. This complex implementation is necessary due to the two-dimensional nature of the expansion of the base matrix indexes, which necessitates an expansion in both the rows' and columns' direction.

This expansion is handled by saving the data that will be in the min-sum algorithm in a matrix that is then traversed with the use of a so-called, "full_minsum_index" and a "row_minsum_index" index that is incremented and decremented after every fetch and store cycle. The data is not stored in the specific rows and columns in this implementation but all values are stored in a one-dimensional array with their Zc expansion in the same row, to save on computation time. The array is thus of the length (number of non -1s in the base matrix) × (Zc). This leads to saving space since there are absolutely no zeros used for place holding, instead 2 indexes are used to keep track of which values are to be worked with in the min-sum operation.

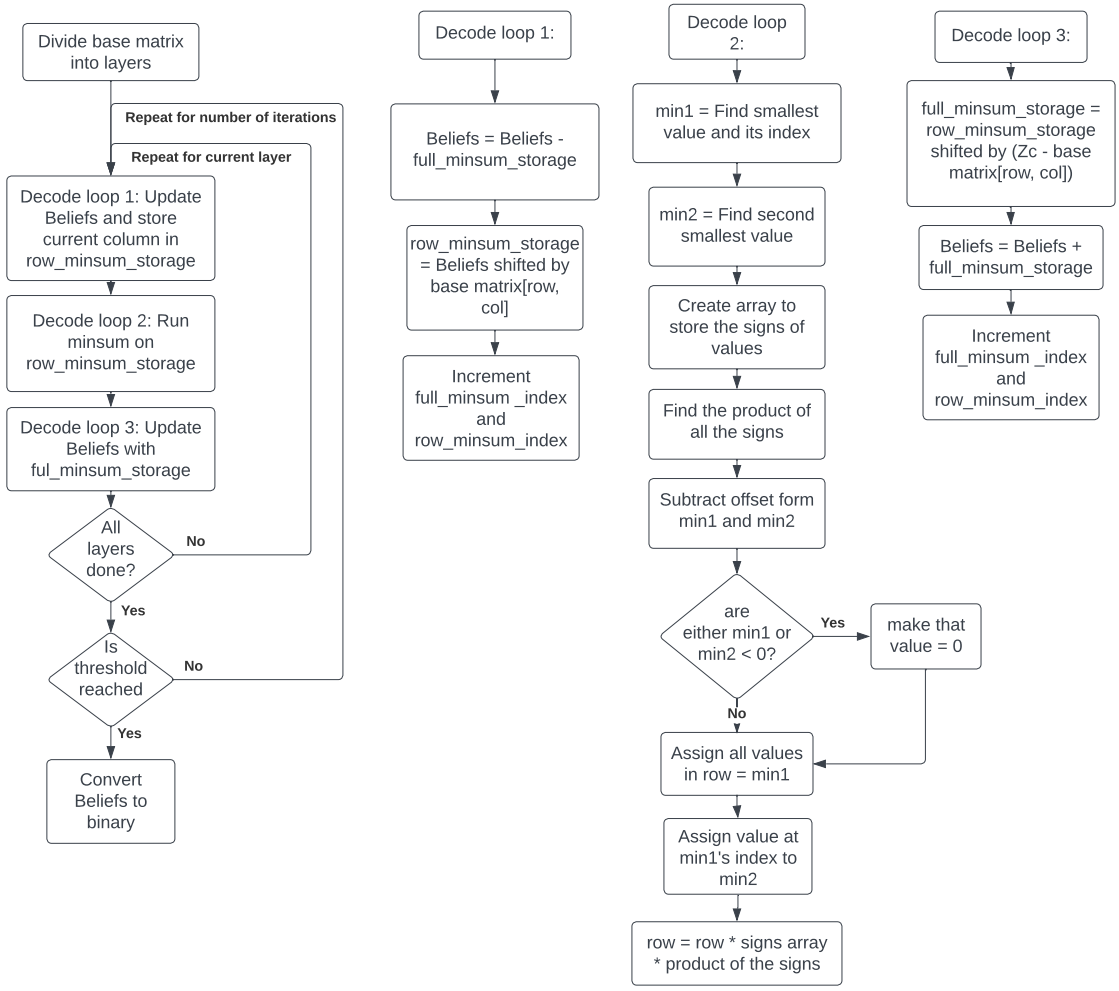


Figure 16.
Run-time optimized decoder algorithm flow chart

Algorithm 8 Run-time optimized decoder pseudo code

```
B = base matrix
total_num_of_non_minus_1s = total number of not -1 values in base matrix
max_num_non_minus_1s_in_row = maximum number of not -1 values in a row of the base matrix
row_minsum_storage = matrix of dimensions (Zc × max_num_non_minus_1s_in_row) for storing the row operations
Beliefs = received codeword
full_minsum_storage = array of length (total_num_of_non_minus_1s × Zc)
divide the base matrix into layers
for repeat for desired number of iterations do
    reset full_minsum_index to 0
    for repeat for each layer do
        for row = repeat for each row in layer do
            reset row_minsum_index to 0
            non_minus_1s_indexes = store column index numbers which values are not -1
            for cols = repeat for each column in non_minus_1s_indexes do
                Beliefs[cols*Zc:(cols+1)*Zc] -= full_minsum_storage[full_minsum_index,:]
                shifted = shift_arr(Beliefs[(cols)*Zc:(cols+1)*Zc], B[row,cols])
                row_minsum_storage[:,row_minsum_index] = shifted
                increment row_minsum_index and full_minsum_index
            end for
            non_min1s_in_row = row_minsum_index
            full_minsum_index = full_minsum_index - row_minsum_index
            reset row_minsum_index to 0
            for i = 0:Zc do
                minsum = row_minsum_storage's current row
                min1 = Find smallest value in minsum's current row
                pos = Find smallest value's position in minsum's current row
                min2 = Find second smallest value in minsum's current row
                sign = Store sign of each value in minsum's current row
                S = overall parity of minsum's current row
                set all values in minsum[i,0:non_min1s_in_row] = min1
                set value at pos in minsum[i,0:non_min1s_in_row] = min2
                multiply each value in minsum[i,0:non_min1s_in_row] with S and sign matrix
            end for
            for cols = repeat for each column in non_minus_1s_indexes do
                shifted = shift_arr(minsum[:,full_minsum_index],Zc-B[row, cols])
                full_minsum_storage[full_minsum_index,:] = shifted
                Beliefs[cols*Zc:(cols+1)*Zc] += row_storage[row_minsum_index,:]
                increment row_minsum_index and full_minsum_index
            end for
        end for
    end for
end for
final_beliefs = Beliefs
```

3.7 Decoder BER optimizations

The BER performance for the decoder implementation can be improved by applying the following adaptations that work to improve the rate at which the decoder beliefs converge on their optimal state or work to optimize the beliefs of a particular row of bits in the min-sum operation. One method of increasing the rate at which the beliefs converge to their optimal state is by using layering. Two methods in which the beliefs of a particular row in the min-sum operation are optimized are by using a normalization factor or an offset factor.

3.7.1 Layering

The layering process splits the base matrix up into a certain number of layers and then applies the entire decoding process to that particular layer independently of other layers in the base matrix. The decoder then moves on to the next layer in the base matrix and does the entire decoding process on that layer, once again, independently of the other layers of the base matrix as if they do not exist. This layering effect works to allow the convergence to an optimal state to occur in fewer iterations as there are fewer rows in the base matrix affecting the process. This can then effectively allow certain layers to converge before other layers that might be more adverse in nature.

3.7.2 Normalization

Normalization is the process where the minimum values that are used for the min-sum process are multiplied by a factor, alpha, before being assigned to their locations in the row on which the min-sum algorithm is being applied. This step effectively further increases the dominance of the terms that dominate the equation that calculates the belief of each bit as can be seen from figure 9.

3.7.3 Offset

Implementation of an offset is another method that is used in order to increase the dominance of the minimum values in the row of the min-sum operation, as in normalization. In the offset method, instead of multiplying the minimum values of the min-sum operation with an alpha factor, there is simply an offset factor that is subtracted from the minimum values. This method enforces the same mechanism whereby the dominance of the terms that dominate the min-sum equation is further increased as can be seen from figure 9. This creates a higher certainty of the state of the bits in the given check-node group that makes up a given row of the parity check matrix.

3.8 Simulation implementation for Min-sum algorithm

3.8.1 Run-time and language selection

In order to obtain the optimal run-time implementation, one of the important considerations to take into account is the language in which the code is implemented. The optimized decoder was originally designed in Matlab and then translated into Python, Java and C++ to find the optimal language for run-time considerations. The Matlab and Python implementations are both interpreter languages that are very simple to code and allow for rapid prototyping and development as a result of the built-in functions that relate specifically to dealing with multidimensional matrices. The run-time to decode one codeword of length 2040 bits, using six iterations for each of the languages is shown in table 7. The compiled languages Java and C++ showed to have approximately two orders of magnitude better run-time than the interpreted languages Matlab and Python. This is because the interpreted languages must analyze each statement in a program over again each time that it is reached whilst compiled languages can analyze the code at compile time and self-optimize for better run-time performance.

Table 7.
Programming languages and associated run-times on a PC device using 6 iterations in the decoder

Programming language	Run-time per codeword (ms)
Matlab	113
Python	263
Java	1.8
C++	3.7

3.8.2 *Normalization*

In figure 17 one can see the impact that adding a normalization factor can have on the BER of the system. The number of iterations in this implementation is 6 and the number of codewords per point in the BER curve is 10k codewords. This results in 20.4 Mb being decoded per point in the BER curve. When the normalization factor is decreased from the unnormalized 1.00 to 0.65 the BER improves up to a best case of 1×10^{-4} at approximately 2.32 dB away from Shannon's limit. The BER then increases as the normalization factor is further decreased.

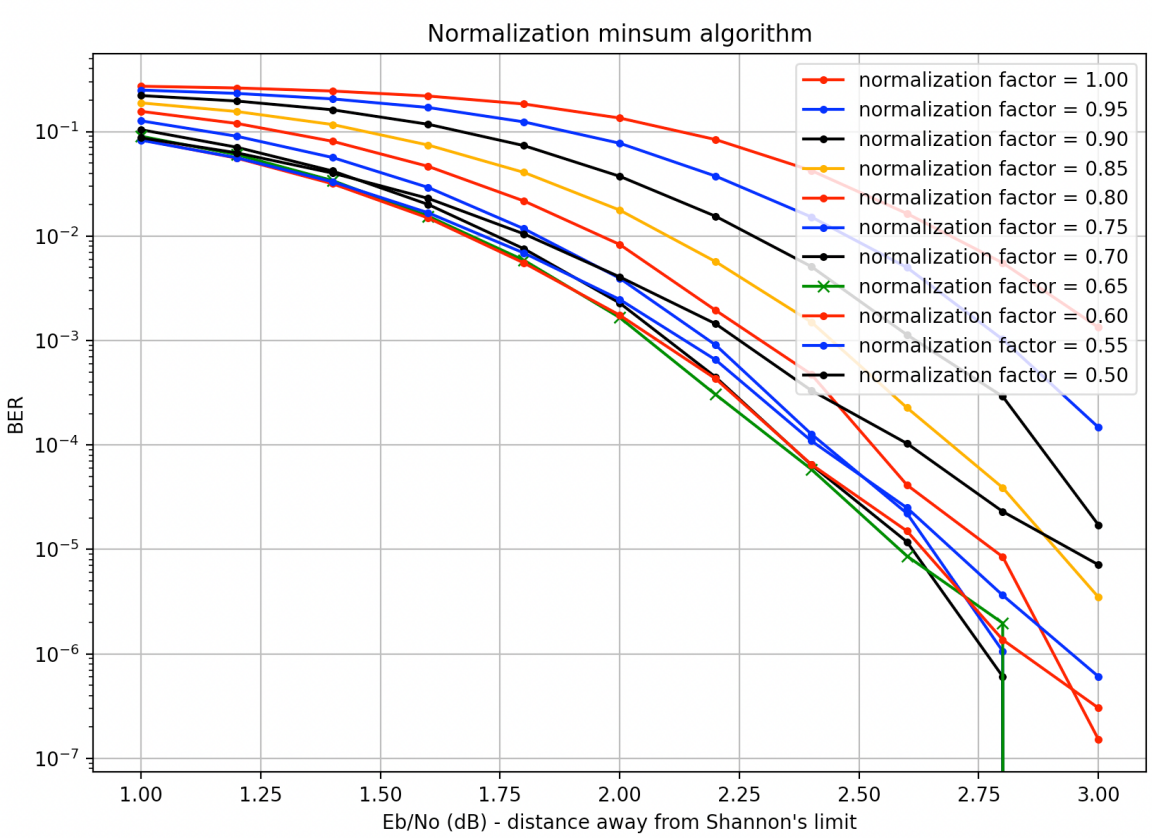


Figure 17.
Normalization factor testing

3.8.3 Offset

In figure 18 one can see the impact that adding an offset factor can have on the BER of the system. The number of iterations in this implementation is 6 and the number of codewords per point in the BER curve is 10k codewords. This results in 20.4 Mb being decoded per point in the BER curve. When the offset factor is increased from 0.05 to the best implementation of 0.35 the BER improves up to a best case of 1×10^{-4} at approximately 2.28dB away from Shannon's limit. The BER then increases as the offset factor is further increased. The offset factor of 0.35 is thus further used in the best implementations of the system since it provides marginally better performance than that of the normalization factor method.

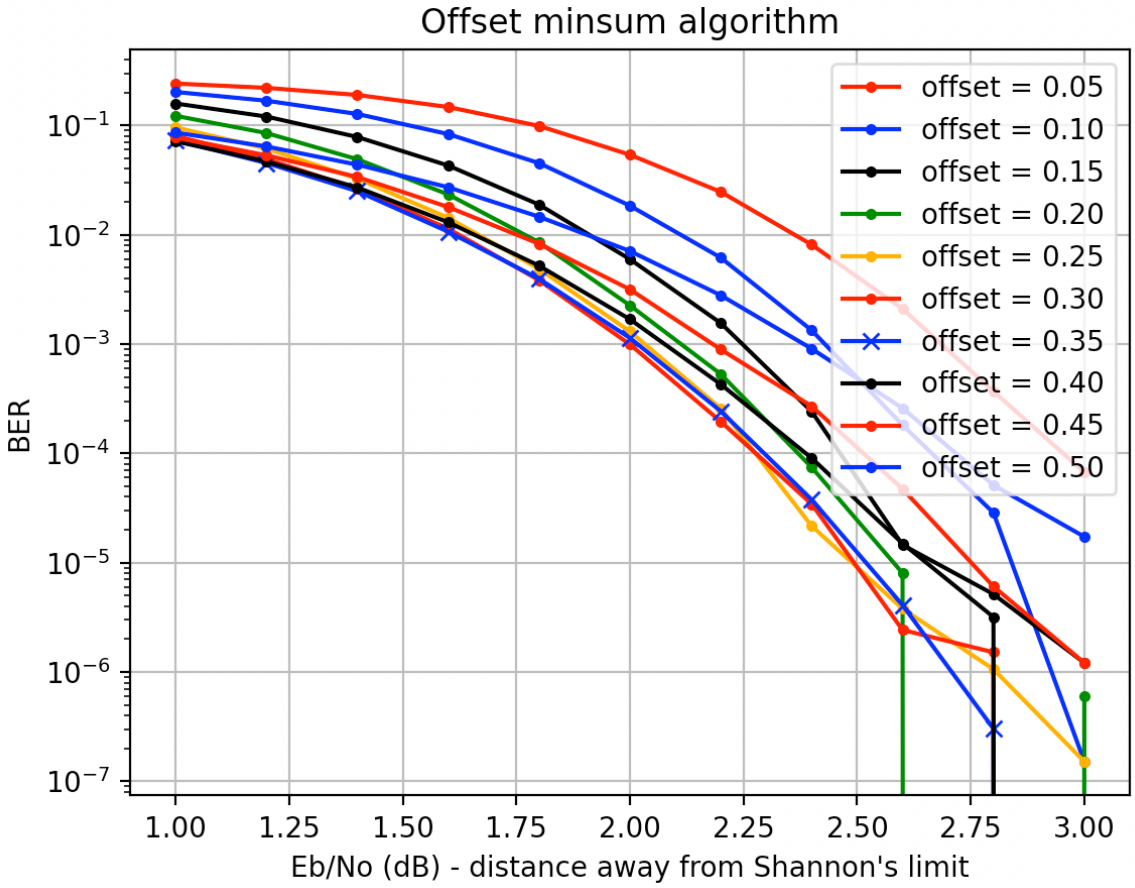


Figure 18.
Offset factor testing

3.8.4 Codeword length testing

In the testing of the codeword length's effect on performance in figure 19 it can be seen that the codeword/block length has a major impact on the BER of the system. The longer the codeword length the better the BER. The 4896-bit codeword has the best performance of 1×10^{-4} at approximately 2.05 dB away from Shannon's limit for the implementation where the number of iterations is 6 and the offset factor is 0.35.

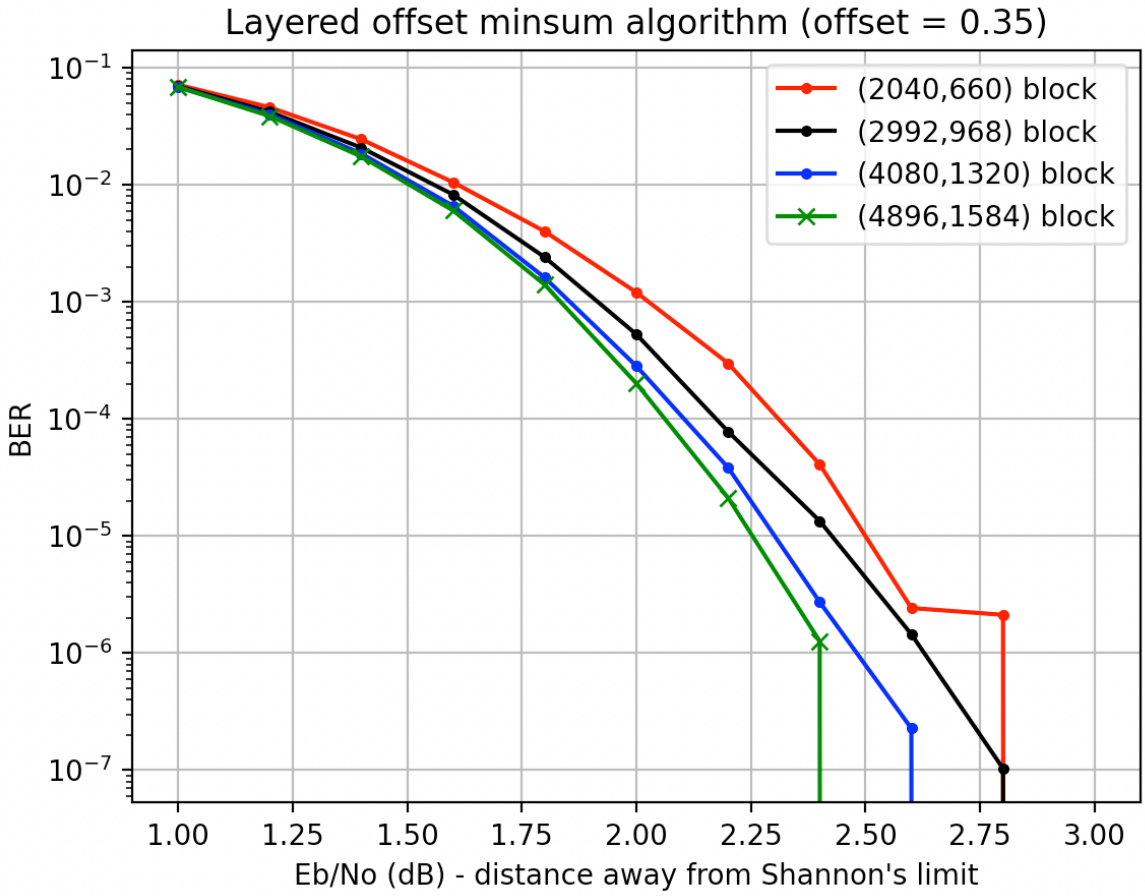


Figure 19.
Codeword length testing

3.8.5 Iterations testing

In the testing of the number of iterations on the BER for the given implementation, it can be seen in figure 20 that the number of iterations used has the most profound impact on the BER of the system in question. The best implementation of 30 iterations provides a BER of 1×10^{-4} at approximately 1.70dB away from Shannon's limit. The larger the number of iterations the better the BER of the system. There is however a number of iterations where the gains in BER do not warrant the extra run-time. This point is around 15 to 30 iterations. Using more than this amount of iterations in the implementation only marginally improves the BER of the system. The implementation used in the other tests is 6 iterations. This provides a reasonable BER and a good run-time trade-off.

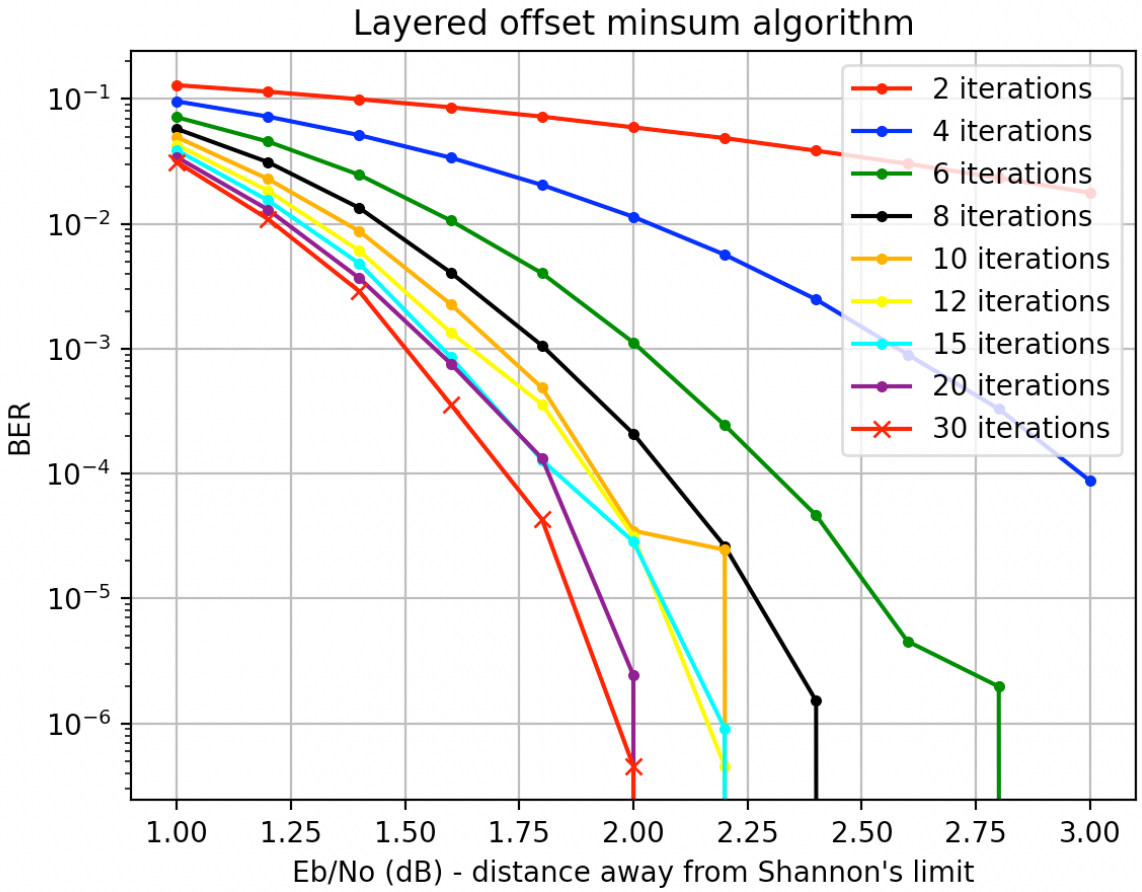


Figure 20.
Iterations testing

3.8.6 "Thresholding" and early stopping

Implementation of the decoder algorithm is improved by introducing "early stopping". This stops the decoding process when all the beliefs of the message bits are a certain distance away from zero. This distance is self-named as the "threshold" for early stopping. The goal of this optimization is to avoid running for more iterations than necessary in cases where the optimal bit-state has already been determined. This allows the system to achieve approximately the same BER performance as is achieved in the 20-iterations case as seen in figure 21, but with the average run-time of a decoder setup with much fewer iterations. Hereby, improving the average run-time of the decoder system.

The system keeps on running for more iterations until the point where all the bit's beliefs are above the given threshold. In testing, the average number of iterations for which the system will continue to decode is approximately 8 iterations. Therefore, only in extreme cases where the system is unable to determine whether the bit is a 1, or a 0, will it continue for iterations up to a maximum of 50 iterations, per codeword. This is very unlikely since AWGN is based on the Normal distribution.

The "thresholding" has only been implemented for the AWGN channel decoder since in testing it was observed that it provided no benefit in the case of the Rayleigh flat-fading channel since the channel is so adverse that it reaches the maximum number of iterations in almost all cases, therefore on average there is no noticeable benefit as seen in the AWGN case.

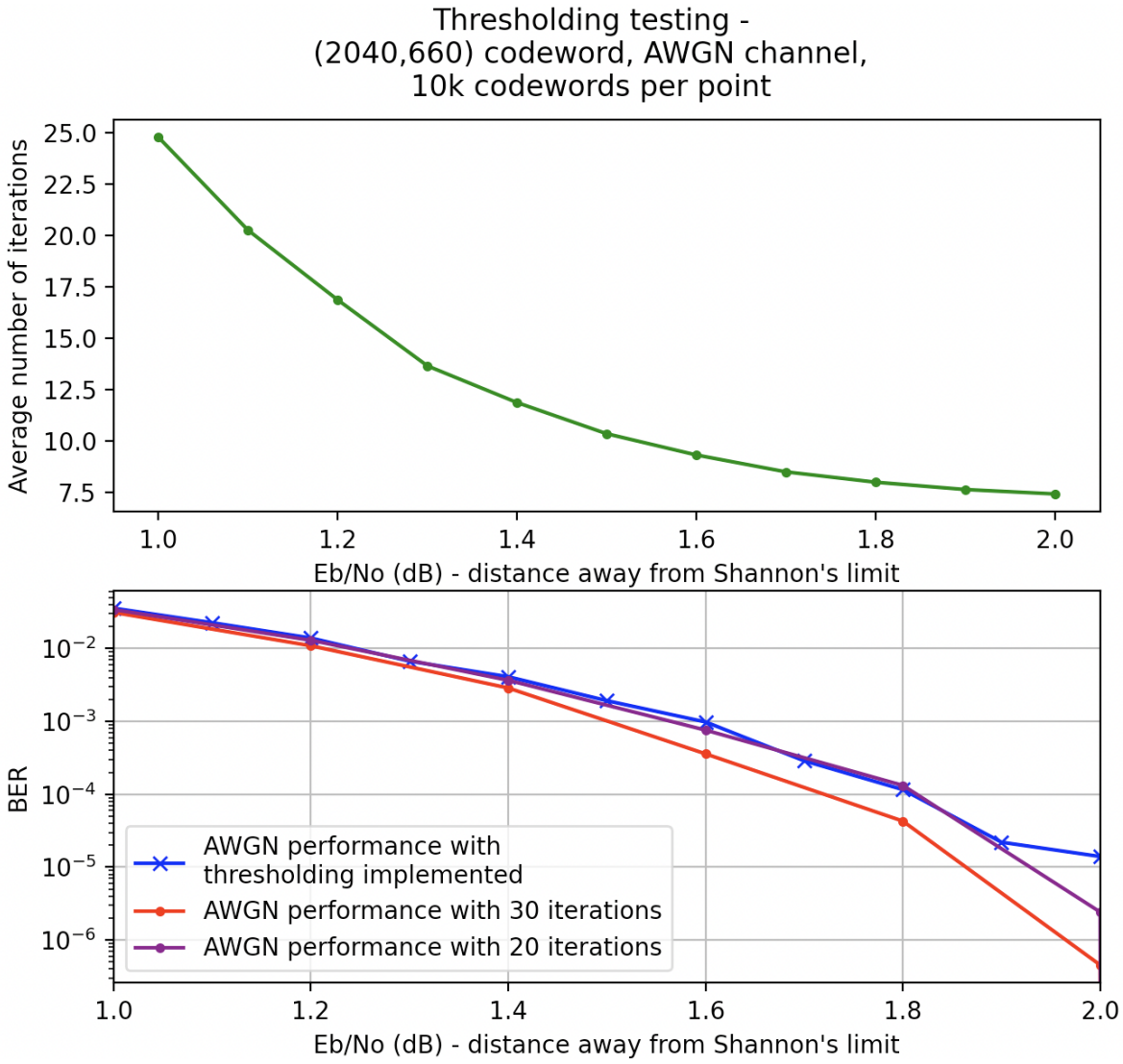


Figure 21.
"Thresholding" testing

3.8.7 Layering

The implementation of layering shown in figure 22 in the decoder was theoretically supposed to provide a better BER but the implementation in question provided no visible benefit when implemented on the 2040 bit codeword using 6 iterations and an offset factor of 0.35. The reason for the number of iterations tested being 1,23 and 46 layers are that the number of rows in the base matrix is 46. Therefore, the layering was implemented to provide an even number of rows per layer and the only factors of 46 are 1,2,23,46 (2 was not tested since it is very close to 1).

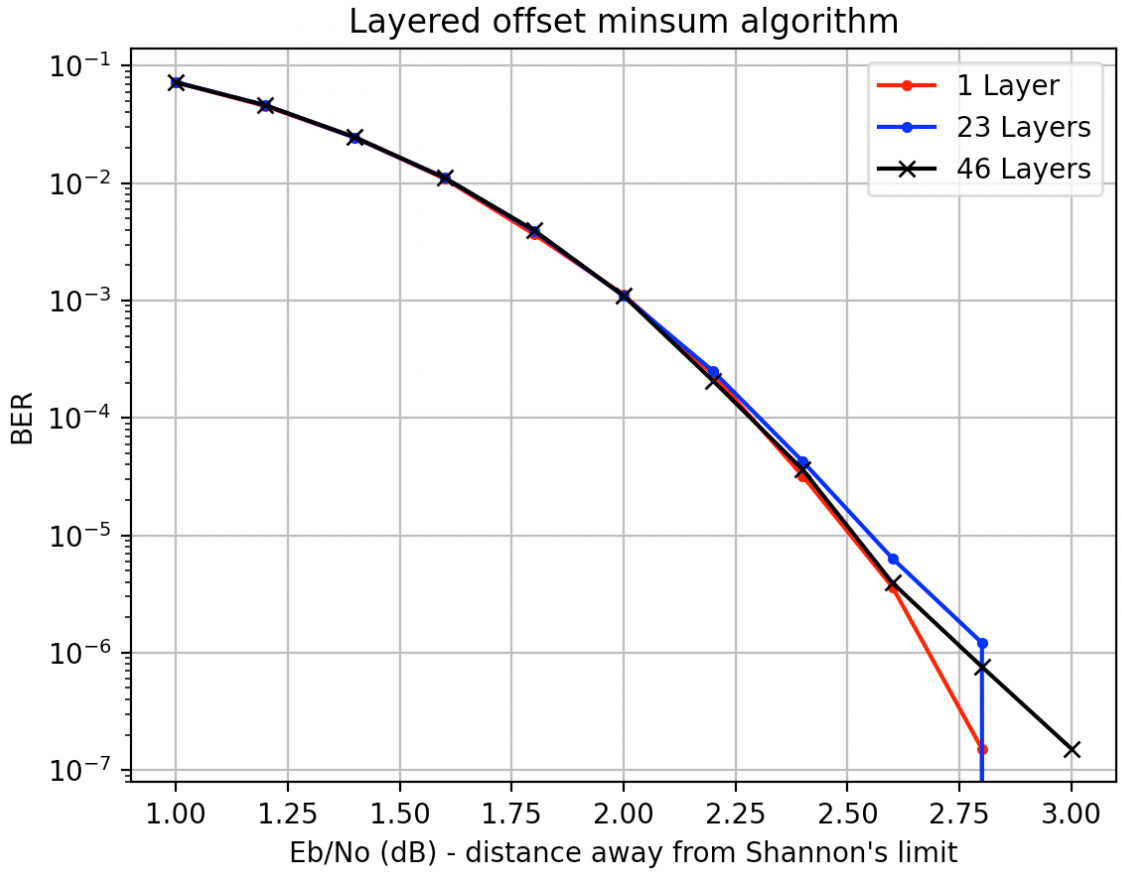


Figure 22.
Layering testing

3.9 Simulation implementation for Sum-product algorithm

3.9.1 Difference between Min-sum and Sum-product algorithm

The difference between the Min-sum algorithm and the Sum-product algorithm is that the Sum-product's goal is to use the relative value of each bit in the check node group to determine its value whereas the Min-sum algorithm assumes that the smallest value in the check node group will completely dominate the equation. The Sum-product uses the relative value of each bit and then sends that value through an equation called ϕ, which then determines each bit's value more independently of other values than the Min-sum algorithm.

The Sum-product algorithm has been shown to be better than the unoptimized Min-sum algorithm in terms of BER performance, but much slower since there must be an actual calculation done for each bit in the check node group, whereas the Min-sum algorithm doesn't need to do a calculation at all.

3.9.2 Sum-product algorithm

The Sum-product algorithm is the same for the check node operations but only different for the bit node operations. The algorithm for the Sum-product bit-node operation is given in algorithm 9. Sections of this pseudo-code are similar to that given in [17].

Algorithm 9 Sum-product bit-node operation pseudo code

```
1:  sum_elements = 0
2:  for i = elements in check node group do
3:      sum_elements = sum_elements + φ(||val[i]||)
4:  end for
5:  sign = store sign of each value in check node group
6:  S = overall parity of check node group
7:  for i = elements in check node group do
8:      val[i] = φ(sum_elements - φ(||val[i]||))
9:  end for
10: for i = elements in check node group do
11:     val[i] = val[i]*sign[i]*S
12: end for
```

Where ϕ is the function in equation 46.

$$\phi(x) = log(\frac{e^x + 1}{e^x - 1}) \tag{46}$$

3.9.3 Run-time analysis

The run-time of the algorithm has yielded a run-time of approximately 6.9 ms per codeword for a (2040,660) codeword using 6 iterations on the same PC platform as the Min-sum algorithm for the same codeword and number of iterations. This is 72.5% longer than the 4.0 ms per codeword using the Min-sum algorithm. Therefore, it seems that the speed of the Min-sum algorithm is much better than that of the Sum-product algorithm.

3.9.4 AWGN channel BER performance testing

The Sum-product algorithm is implemented using the same (2040,660) codeword as in the run-time analysis and is compared against the Min-sum algorithm's BER performance for an implementation using 25 decoding iterations. The BER comparison is given in figure 23.

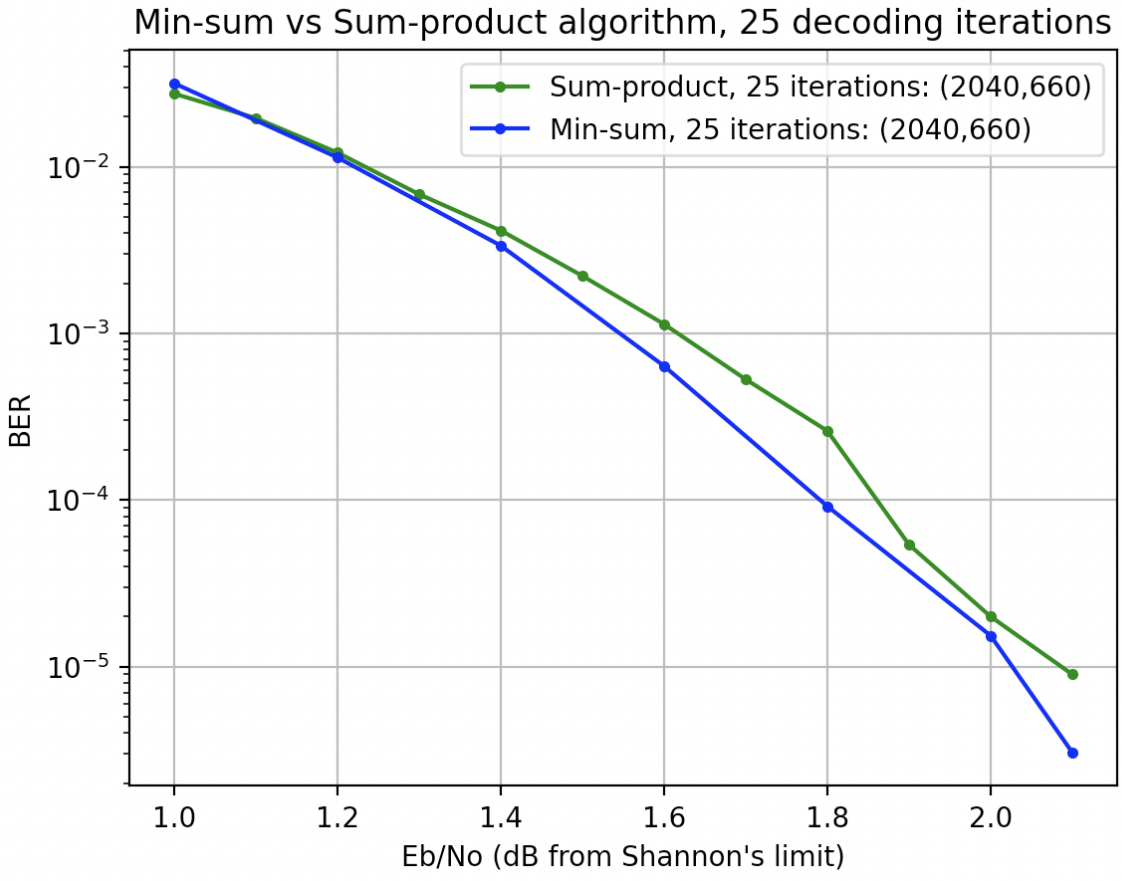


Figure 23.
AWGN BER comparison between Min-sum and Sum-product algorithms

3.9.5 Rayleigh channel BER performance testing

The Sum-product algorithm is implemented using the same (2040,660) codeword as in the run-time analysis and is compared against the Min-sum algorithm's BER performance in the Rayleigh flat-fading channel model for an implementation using 6 decoding iterations. The BER comparison is given in figure 24.

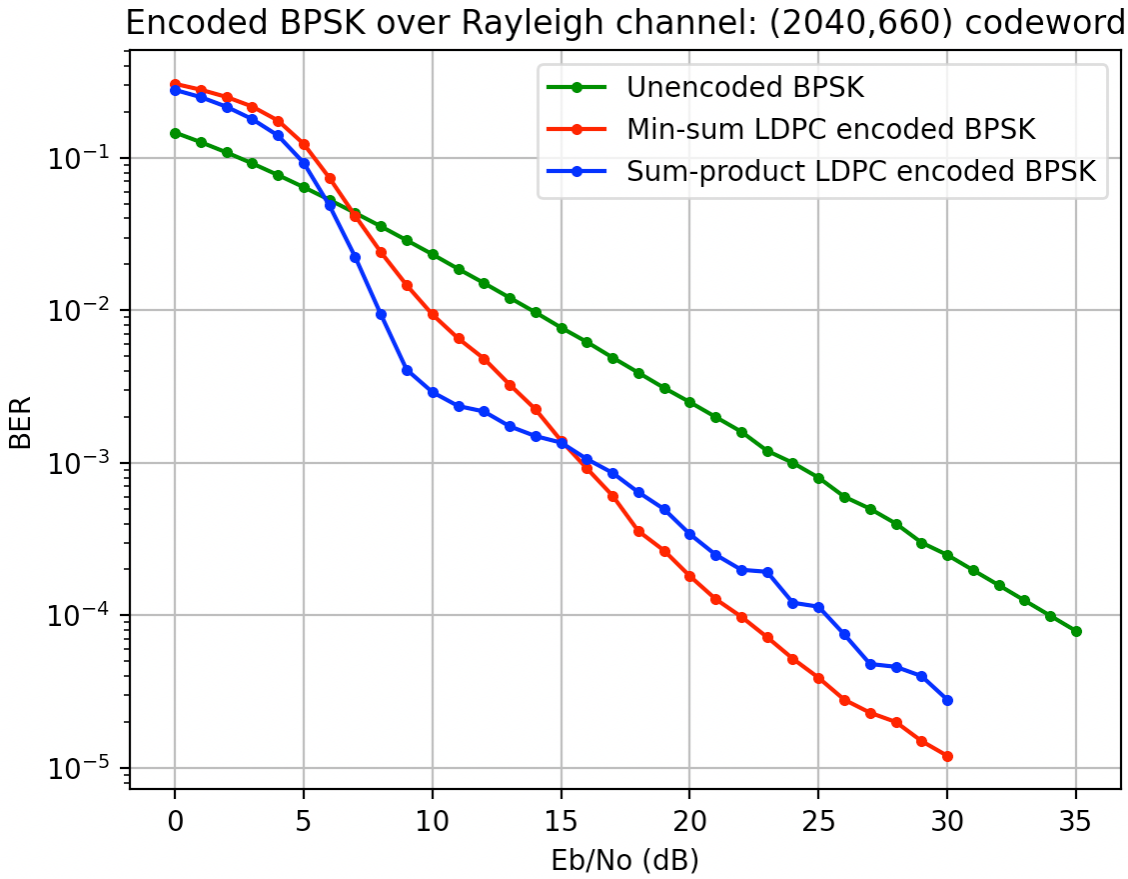


Figure 24.
Rayleigh flat-fading BER comparison between Min-sum and Sum-product algorithms

3.9.6 Comparison with Min-sum algorithm

From the run-time that is 72.5% longer in the case of the Sum-product algorithm and the BER performance that is approximately 0.07 dB better at a BER of 1×10^{-5}. From these simulations, it is clear that the optimized Offset-Min-sum algorithm is the best implementation for the AWGN channel. The Rayleigh flat-fading channel implementation is also approximately 3 dB better in the Min-sum algorithm implementation for Eb/No values between 16 dB and 30 dB. Therefore, the best overall implementation is the Offset-Min-sum decoder algorithm.

3.10 Best simulation implementation

The best implementation is shown in figure 25. The implementation is shown for the 2040 and 4896-bit codewords each using 30 iterations of the Min-sum decoder with an offset factor of 0.35. The best implementation for the 4896-bit codeword is also using the Min-sum algorithm. This implementation provided a BER of 1×10^{-4} at approximately 1.38 dB away from Shannon's limit and the 2040 bit codeword provided a BER of 1×10^{-4} at approximately 1.75 dB away from Shannon's limit. This is very good performance that is around the upper end of what is obtained in literature [17] for codewords of the given length and code rates of approximately 0.32.

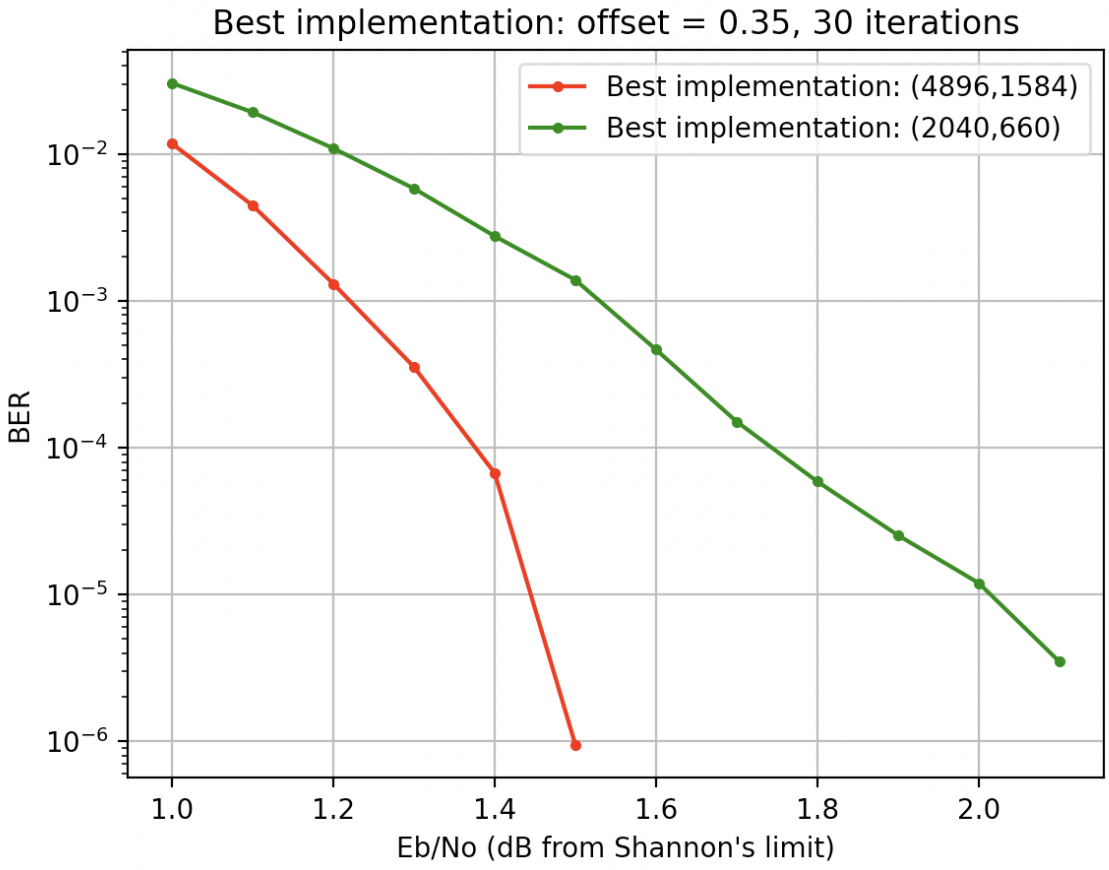


Figure 25.
Best implementation with 2k and 4.9k codeword length

3.10.1 AWGN channel comparison

In figure 26 the best BER implementation for the implemented decoder is compared to the performance of an unencoded BPSK signal in an AWGN channel. The performance improvement that the LDPC decoder provides is approximately 7.5 dB of Eb/No at a BER

of 1×10^{-4}. This means that the same performance of an unencoded BPSK system can be achieved in the presence of 7.5 dB more noise. This clearly shows the efficacy of the implemented LDPC system.

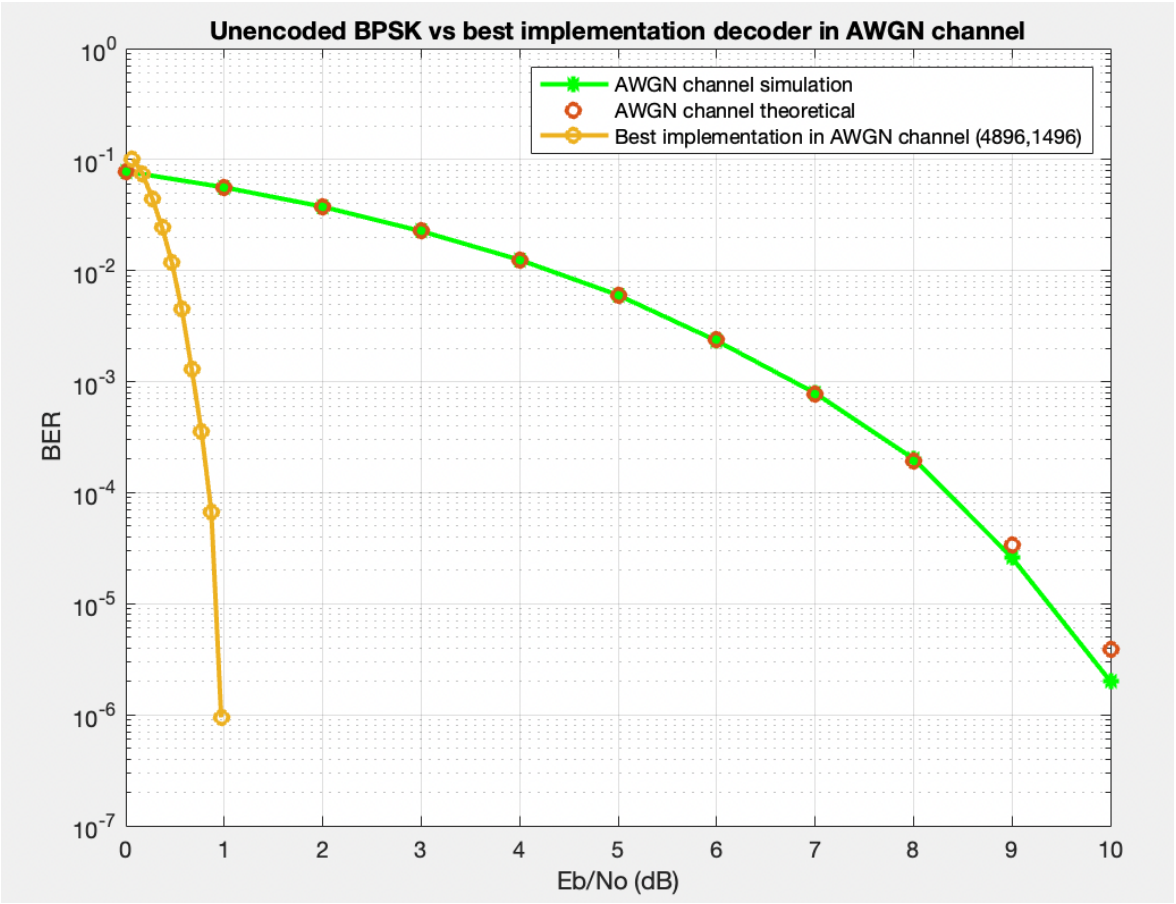


Figure 26.
Unencoded BPSK vs best implementation BPSK in AWGN channel

3.10.2 Uncoded vs coded BPSK in Rayleigh flat-fading channel with Min-sum algorithm

Figure 27 shows the improvement in BER achieved from the implemented LDPC decoder when compared to the performance of an unencoded BPSK signal in a Rayleigh flat-fading channel. The implemented decoder implementation is able to achieve approximately a 12 dB improvement in Eb/No when compared to an unencoded BPSK signal in the given Rayleigh flat-fading channel. This shows that the BER gains in the system are also present in the presence of a Rayleigh flat-fading channel as well as the benchmark AWGN channel.

The same specifications for the decoder that are determined in the AWGN case will also be the best implementation in the Rayleigh flat-fading channel. These optimizations will however have a smaller impact in the case of the Rayleigh flat-fading channel since the channel is so adverse that the improvement in performance will almost be negligible. Nevertheless, the implementation will be carried over from the AWGN optimization.

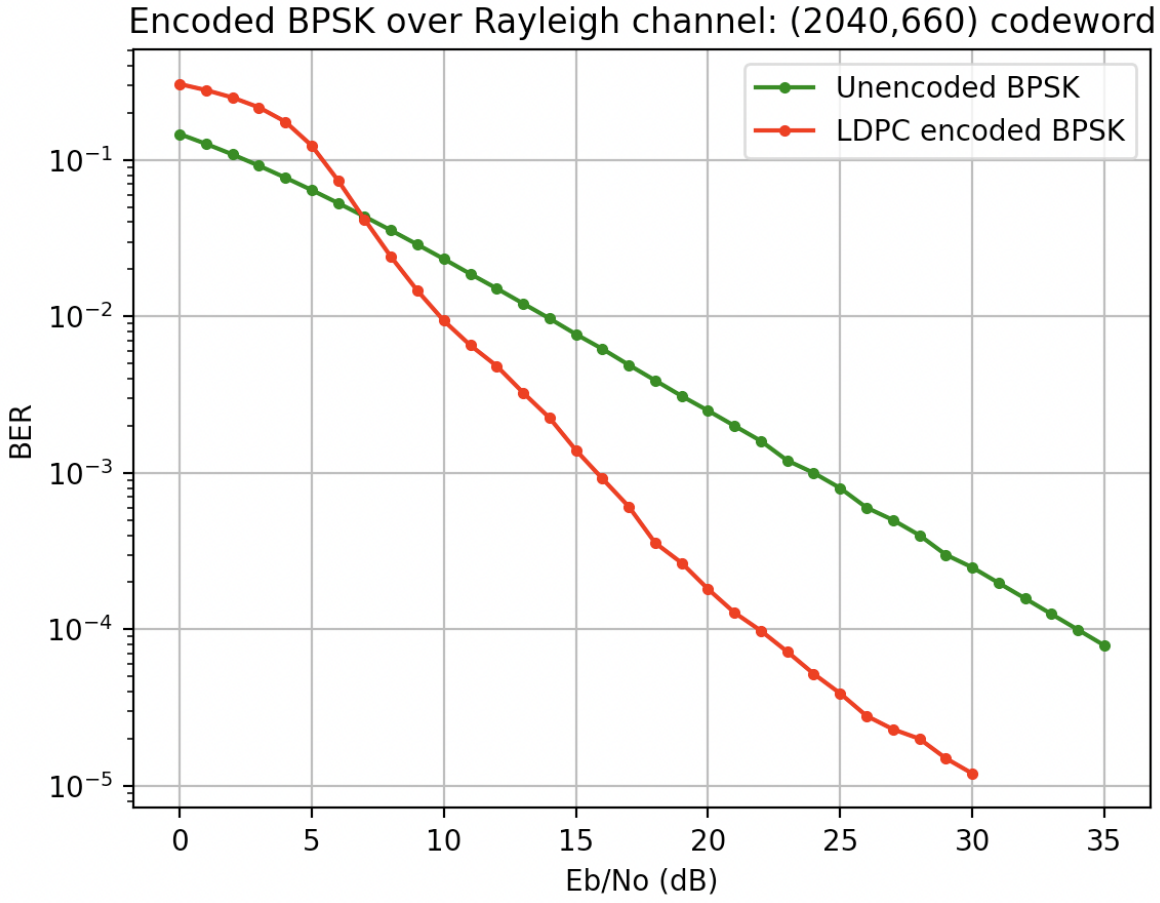


Figure 27.
implemented BPSK LDPC decoder comparison with uncoded BPSK in a Rayleigh flat-fading channel

3.10.3 Bit belief convergence

The convergence of the first 20 bits in the message is plotted to show how they converge to their optimal belief value and away from the decision threshold at $y = 0$. This convergence is shown in figure 28 and 29. Only the bits that were in error, and which were subsequently corrected to the correct side of the decision boundary are shown. It can be seen that the bits' beliefs diverge from the decision boundary with the number of iterations of the decoding algorithm.

To illustrate the power of the decoding algorithm, massive errors are introduced on the first bit in the message and then the bit's belief values are plotted to show how the bit's belief crosses over to the correct side of the decision boundary. This is illustrated in figure 30 and 31.

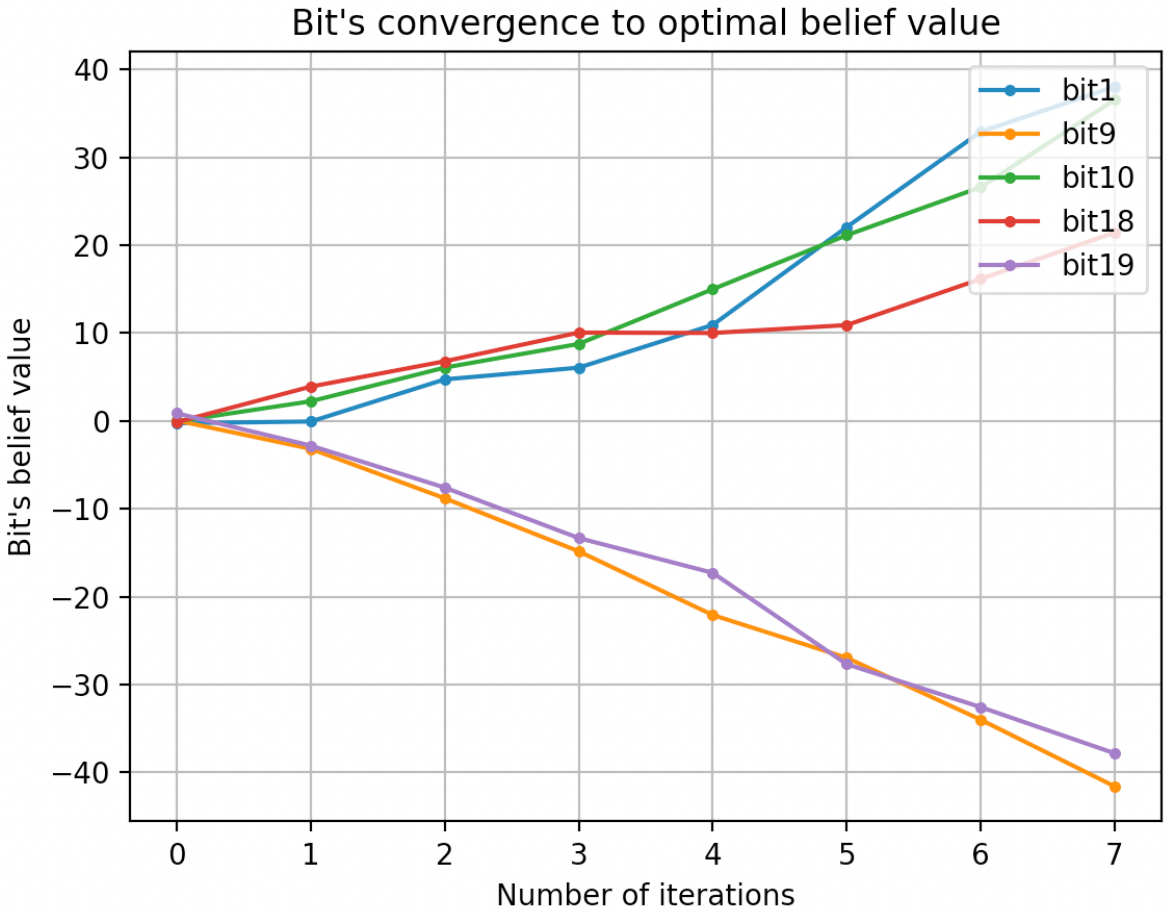


Figure 28.
Divergence of the bit belief values away from the decision boundary

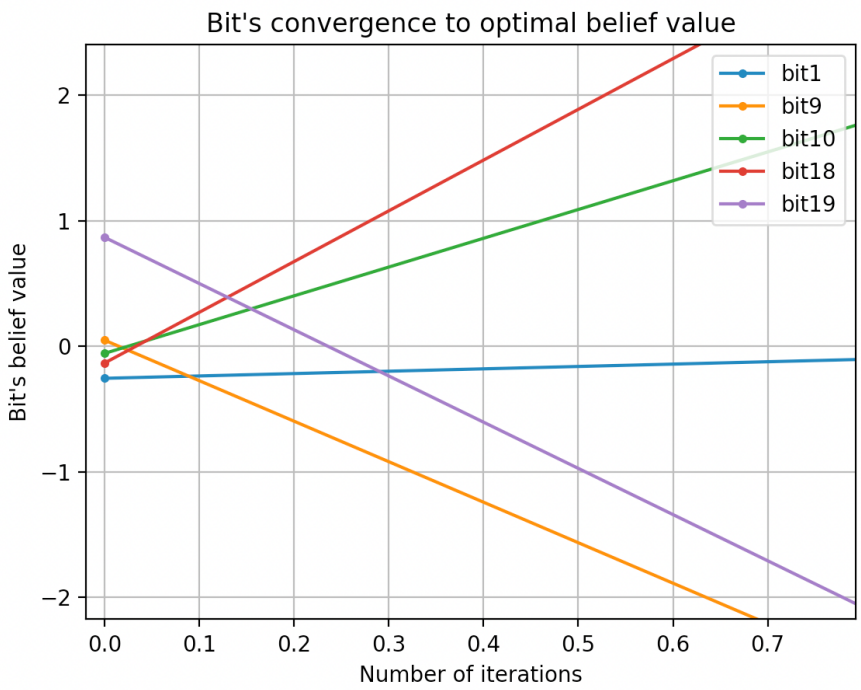


Figure 29.
Bit beliefs crossing over the decision boundary (zoomed in)

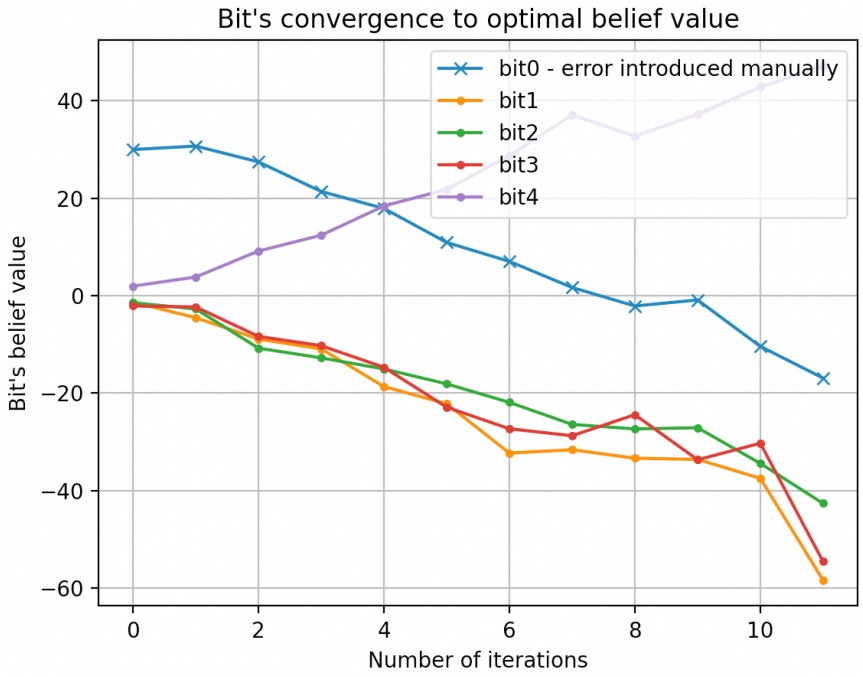


Figure 30.
Bit crossing over the decision boundary

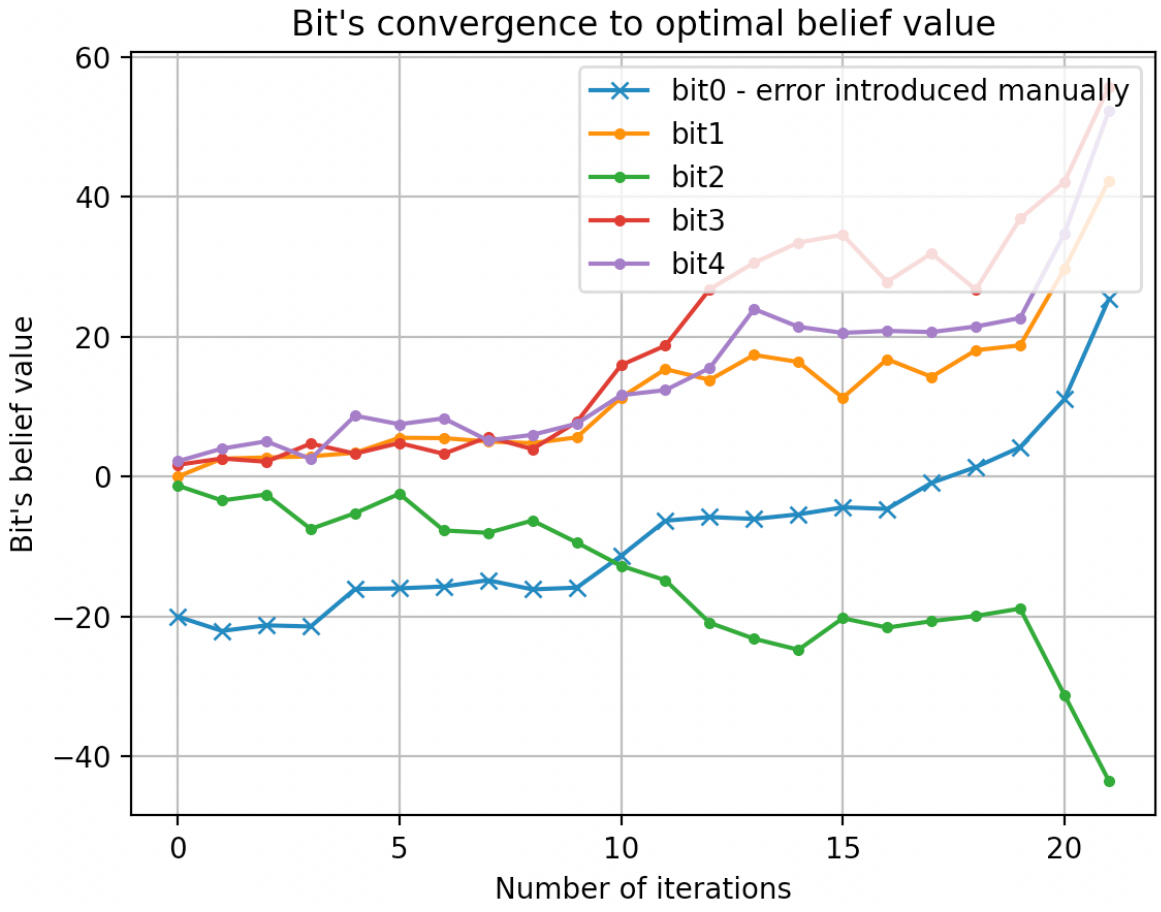


Figure 31.
Bit crossing over the decision boundary

3.10.4 Audio playback

The decoder is applied to a binary snippet of an audio track in the presence of a simulated Rayleigh flat-fading channel with 23 dB of Eb/No. Firstly, the audio snippet is converted into a binary source message which is sent through the encoder. Then the encoded message is sent through the Rayleigh fading channel which will introduce errors that must be corrected with the decoder. The noisy message is then sent through the decoder and errors are corrected. Since the flat-fading channel will have an Eb/No value of 23 dB there will still be approximately 1 error in every 10,000 bits. Therefore the message will still have errors. The decoded binary message is then played back with the errors that have not been corrected still present. The audio snippet is played back, and a small amount of "scratching" can be heard since there are bit flips that slightly distort the audio quality.

The encoding of a 10-second snippet of audio in Matlab into codewords of (2040,660) bits requires the 10-second audio snippet to be split up into approximately 10,000 codewords. This results in a decode time in Matlab of approximately 1160 seconds or 19 minutes of decode time per 10-second audio snippet. This decoding however is only done in Matlab in order to simply facilitate the conversion of songs from mp3 to binary and back again. Therefore, the song could be encoded and decoded on an embedded device in the desired compiled programming language and then converted from binary to mp3 for playback separately as it would be done in any other message's case. An exploration of the embedded device implementation with the desired specification of 1 decoded codeword per second would lead to approximately 10,000 seconds of decode time per 10-second audio snippet, or in other words, it would take approximately 1000 times the audio snippet length to decode on an embedded device. This seems excessive but it makes sense since the purpose of LDPC codes is to encode a message that is to be sent over very high-noise channels. In this case, the decode time is less relevant than the ability to decode a message with high accuracy in the presence of a very adverse communication channel.

3.11 Simulation of other code rates

Different code rates can be implemented simply by trimming the number of columns and rows in the base matrix that are used. Decreasing the number of columns and rows in the base matrix that are used increases the code rate. The decoder with message of length $k = 660$ is implemented at different code rates and the results are shown in figure 32.

The Shannon's limit for code rates of 0.323, 0.50, 0.667 and 0.76 are -0.53 dB, 0.188 dB, 1.084 dB and 1.708 dB respectively [7]. The resulting distance away from Shannon's limit for a BER of 1×10^{-4} are 1.82 dB, 2.31 dB, 3.12 dB and 3.5 dB respectively.

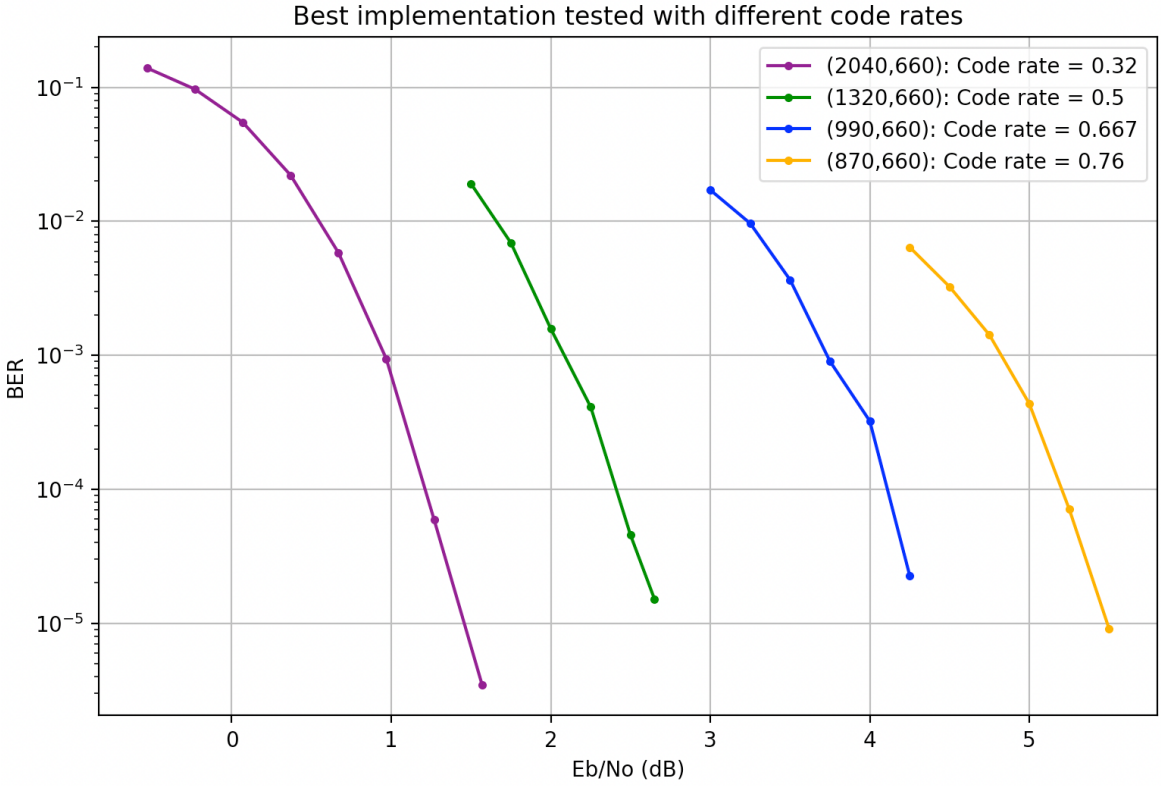


Figure 32.
Code rate testing

3.12 Embedded implementation

3.12.1 Implementation considerations

The main consideration of implementing the simulated system on an embedded device is the decode time. This is due to the knowledge that the embedded device will likely be orders of magnitude slower than a high-powered PC device. This means that if the optimized decoder is implemented with the specifications of 6 iterations in Python, which can take 0.263 seconds per codeword (from table 7) on an embedded device, then in order to meet the run-time specification of 1 second per codeword then the embedded device would only be allowed to be $1/0.263 = 3.8$ times slower than the high powered PC. This is not possible in the budget and embedded device constraints of the project. Therefore, care must be taken to use the optimal implementation in the fastest programming language possible. Only in the case described will the system be able to achieve the desired run-time, whilst still using the desired amount of iterations in the decoding process in order to still meet the specification that allows a BER of less than 1×10^{-4} at an Eb/No value that is satisfactorily close to Shannon's limit for a BPSK system in an AWGN channel, and also the same stated BER at an Eb/No value of less than 24 dB in a simulated Rayleigh channel.

3.12.2 Embedded platform selection

The selected embedded platform must be high-speed but not overly expensive and application specific. The ideology behind the selection is to show that the embedded implementation is so good that it can meet the desired specifications even if it is not a 'super-high-powered' device that is optimized for speed, but instead a mid-range device that is of medium specifications.

The selected device should however have more than 100MB of memory and be simple to interface with. The selected embedded device is the Beaglebone Black rev C. This embedded platform has a 1GHz ARM processor with 512MB of RAM. This device is selected because it has moderate speed and is a common choice for embedded applications, yet it has enough processing speed and memory for decoding large codeword lengths in a reasonable time frame.

The Beaglebone Black is also very simple to load code onto using SSH from the terminal of the connected PC using its local IP address. The Beaglebone Black is selected over an STM32 device since it allows a higher speed than low-end STM32 devices and is cheaper than very high-powered application-specific STM32 devices.

3.12.3 Run-time and specification adherence

The unoptimized decoder is tested in Matlab and the average run-time per codeword is shown to be 1.75 seconds per codeword. The optimized decoder in Matlab is shown to take

approximately 0.14 seconds per codeword. Therefore at this point, there is no point to continue testing with the unoptimized decoder algorithm since it is more than an order of magnitude slower. The Matlab results are also slower than the required 1 second per codeword, even when implemented on the PC platform which is orders of magnitude faster than the embedded device. Therefore, the difficulty of achieving the run-time specification is apparent since the optimized decoder algorithm must be used and implemented in the fastest language possible in order to meet the run-time specification.

3.12.4 Implementation limitations

3.12.5 Memory limitations

Storing all the values of the expanded H-matrix for a 2k bit codeword will be over 2.8 Mbits to store in RAM. Which will be traversed over numerous times per decode iteration. This is not feasible therefore use the 5G-NR base matrix with in-place calculations that rely on expanding the base matrix only when necessary, as indicated in the optimized decoder algorithm in algorithm 8.

3.12.6 Speed limitations

The Implementation must be fast enough to decode a large codeword in 1 second. Therefore, translate code into a compiler programming language such as C++ or Java. The processor must also be as fast as possible whilst taking budget limitations into account. The decoder algorithm must also be limited to only run for the minimum number of iterations necessary in order to obtain the required BER specifications, as will be the case when "thresholding" is implemented.

3.12.7 Trade-offs

There is a constant trade-off present in all LDPC decoders. The main trade-off variables are:

- Codeword length.
- Distance away from Shannon's limit for a given BER.
- Number of iterations used in the decoder.
- Run-time per codeword.

As the codeword length increases, the BER decreases and therefore the system accuracy increases. This however increases the run-time and memory usage of the system. Increasing the number of iterations in the decoder decreases the BER and therefore, increases the accuracy

but using more iterations in the decoder also linearly increases the run-time of the system. Therefore, a trade-off must be made and decided on in order to provide as good accuracy as possible and therefore a low BER at a given Eb/No value whilst still keeping the run-time and the memory usage down to an acceptable level.

3.12.8 Communication

The communication in the embedded implementation will allow a codeword to be encoded on an encoder Beaglebone Black device and then the encoded message will be sent to the PC device where the message will be sent through a desired simulated channel, adding noise and distortion as necessary. Then the noisy message will be sent to a decoder Beaglebone Black device and decoded using the selected decoder algorithm. The decoded message will then be sent back to the PC device and compared to the source message, the number of errors will be noted and the BER will be determined for the given signal. The BER will be lower than the target specification of 1×10^{-4} at the desired distance away from Shannon's limit or the desired Eb/No value in the case of the Rayleigh flat-fading channel.

The communication process is done by using SSH to connect to the two Beaglebone Black embedded devices. Both of the devices are connected using SSH which gives the PC terminal access to the file system of the devices. The communication process is then executed as shown in figure 33 which shows the order of operations necessary to transfer the information to and from the two embedded devices. The architecture of the system is shown in figure 34.

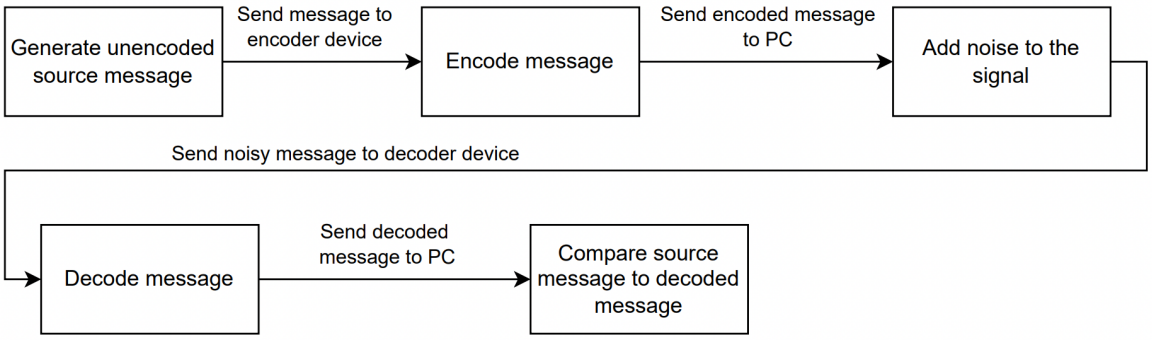


Figure 33.
Communication diagram

3.12.9 Challenges

There are several challenges in the embedded implementation of the system which have been addressed in simulations. The main challenge that is addressed is creating a decoder that is able to make use of the base matrix approach in the 5G-NR standard in order to do in-place operations. This saves on memory usage and allows the system to be implemented on low-speed and low-memory devices. The in-place calculations allow for the system to decode the codewords rapidly in under 1 second per codeword. This is a very stringent run-time

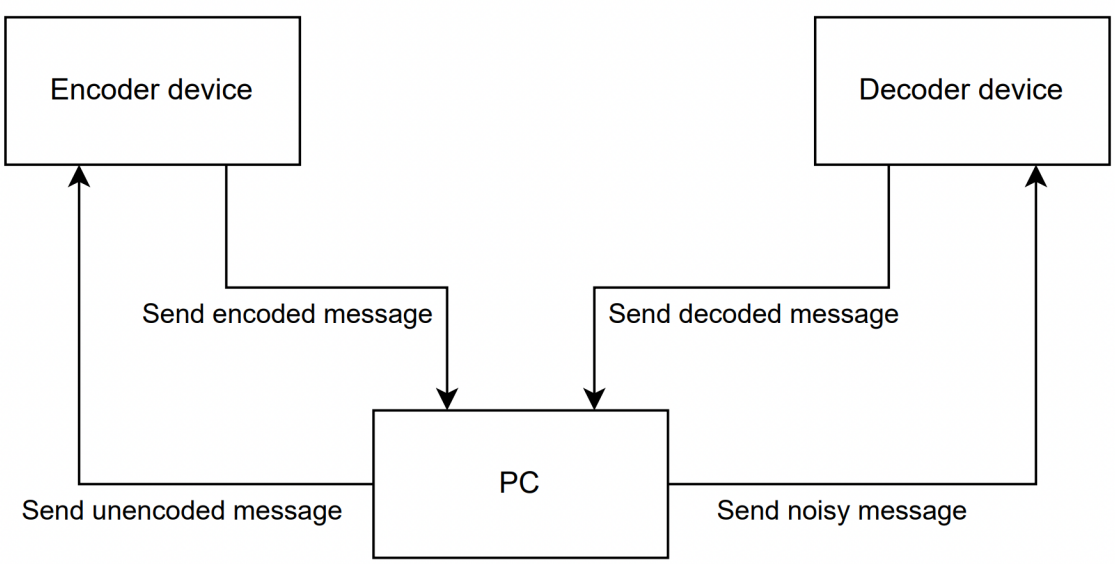


Figure 34.
Communication architecture diagram

specification when taking into account the extremely long codeword lengths and the accuracy that LDPC systems provide in the presence of high-noise communication channels.

Furthermore, the very stringent BER specifications achieved at low Eb/No levels are in-line with some of the best-published results in the literature for codewords in the range of between 2000 and 5000 bits at a code rate of 0.323. The achieved distance away from Shannon's limit in the AWGN channel of approximately 1.38 dB at a BER of approximately 1×10^{-4} is very impressive and will allow near error-free communication in a very adverse AWGN channel. The improvement of approximately 7.5 dB over the unencoded BPSK case is impressive and showing that the same performance can be achieved on a cheap, low-powered embedded device is to be noted.

The approximately 12 dB improvement over unencoded BPSK in a Rayleigh flat-fading channel at a BER of less than 1×10^{-4} on a low-powered embedded device allows for reliable communication to be ensured in very adverse channels.

3.12.10 Embedded results in AWGN channel

3.12.11 Run-time results

The embedded implementation in the presence of a simulated AWGN channel is tested with a codeword length of 2040 bits at a code rate of 0.3235 (as in the rest of the simulations) with early stopping implemented.

The decode time for a single codeword is approximately 0.11 seconds per 2040-bit codeword.

This is well below the specified 1 second per codeword. This results in 660 message bits being decoded in 0.11 seconds which is an approximate decode bit rate of 6 kbits/second.

3.12.12 BER results

Each point in the plotted BER curve symbolizes the average BER when 1000 codewords have been decoded. The BER results can be seen to align with what is expected in simulations. The Eb/No distance away from Shannon's limit where the BER is approximately 1×10^{-4} is approximately 1.81 dB. With Shannon's limit of approximately -0.531 dB, the absolute Eb/No value is approximately 1.28 dB of Eb/No for the required BER. This distance away from Shannon's limit is significantly better than the required 2.5 dB away from Shannon's limit at the BER of 1×10^{-4}.

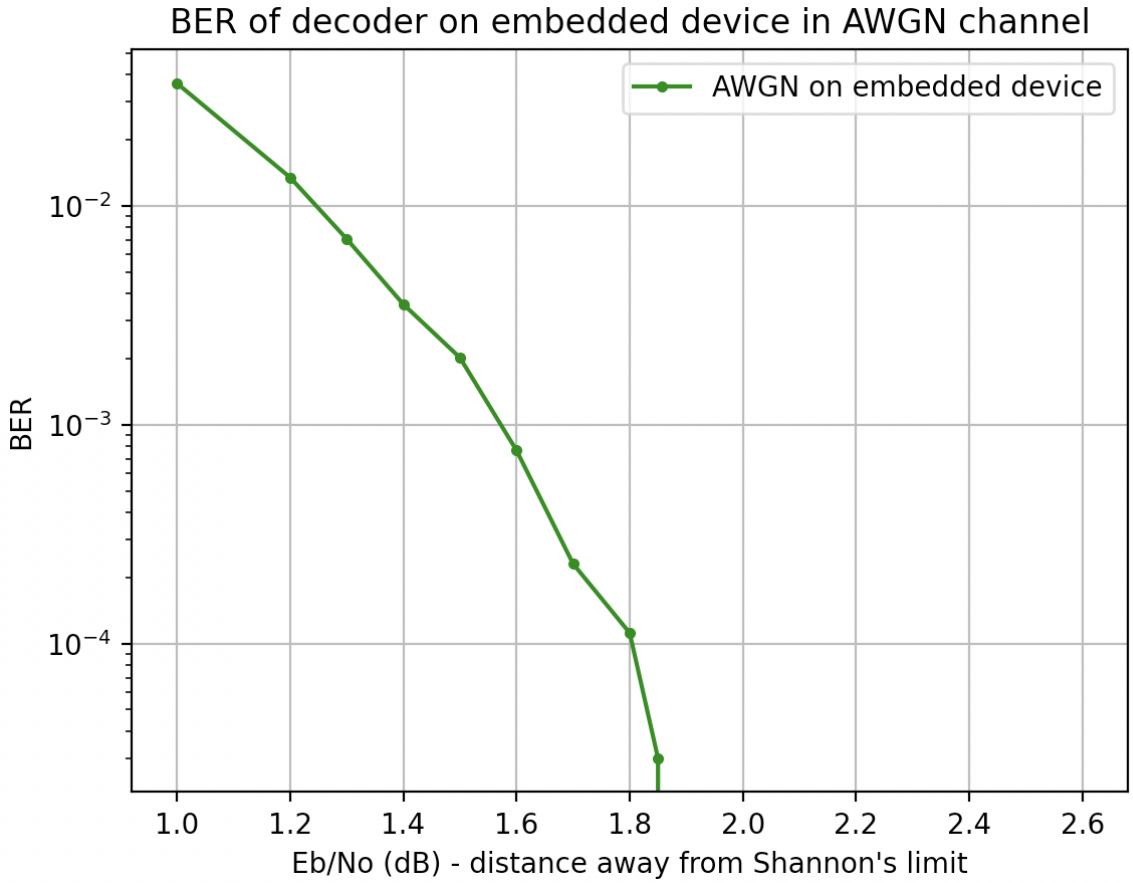


Figure 35.
Results of running the decoder on the embedded platform in an AWGN channel model

3.12.13 Embedded results in Rayleigh flat-fading channel

3.12.14 Run-time results

The embedded implementation in the presence of a simulated Rayleigh flat-fading channel is tested with a codeword length of 2040 bits at a code rate of 0.3235 (as in the rest of the simulations) using "thresholding" that will allow for early stopping when all bits' beliefs are a certain distance away from the decision threshold at zero. Using more than 6 iterations in the Rayleigh flat-fading channel does not provide significant gains in BER performance since the uncoded Rayleigh channel is essentially flat as seen in figure 7 and figure 8. Therefore, large performance improvements that would be gained by using more iterations as seen in the AWGN channel in figure 20 does not apply in the Rayleigh flat-fading channel. Therefore, using fewer iterations in order to gain better run-time performance is the chosen implementation.

The decode time for a single codeword is approximately 0.11 to 0.25 seconds per 2040-bit codeword. This is well below the specified 1 second per codeword.

3.12.15 BER results

Each point in the BER curve in figure 36 symbolizes the average BER when 1000 codewords have been decoded. The BER results can be seen to align with what is expected in simulations. The Eb/No for the Rayleigh flat-fading channel where the BER is approximately 1×10^{-4} is approximately 22 dB. This Eb/No value is slightly better than the required 24 dB at the BER of 1×10^{-4}.

The reason that the end of the BER curve is jagged is that it would take too long to decode 100,000 codewords on the embedded platform that is necessary to give a smooth curve as in the case of the simulation in figure 27. However, it can still be seen that the results match what is expected from simulation results.

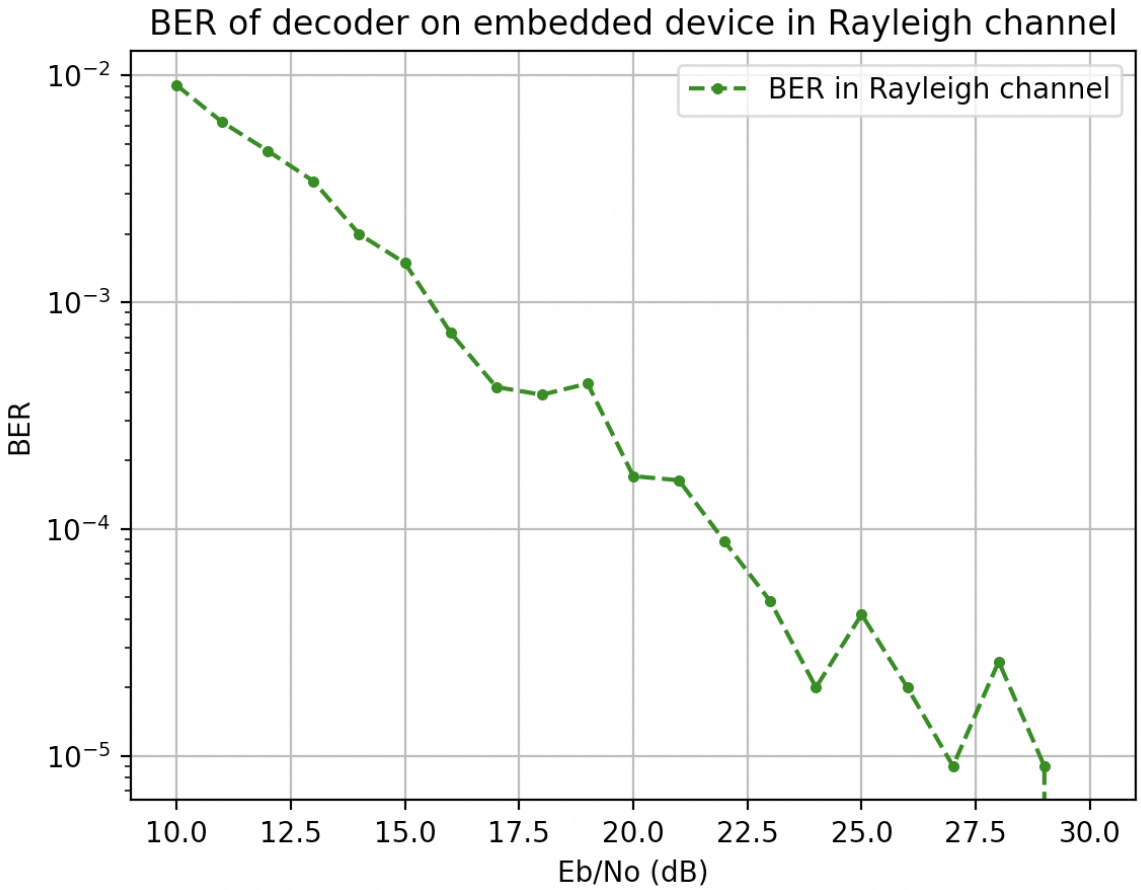


Figure 36.
Results of running the decoder on the embedded platform in a Rayleigh flat-fading channel model

3.13 Demonstration

3.13.1 Communication setup

Initially, the communication setup will be accomplished by connecting the 2 external devices in a way that allows them to be controlled externally by the terminal of the PC. This will be accomplished by using SSH. The 2 devices will be connected and then the message will be generated on the PC and sent to the encoder device. Then the encoding process will be carried out and the encoded message will be written to a file that will be copied to the PC where noise will be added. Then the PC will transfer the noisy message as a file to the decoder where the noisy message will be decoded and the output message will be written to a file that is copied from the decoder device to the PC. Then the PC will read the decoded message bits from a file and then compare the message to that of the source message. This process will be repeated once for the AWGN channel and once for the Rayleigh channel.

3.13.2 AWGN/Rayleigh channel 100 codewords decoding of random message

The demonstration will be carried out using the AWGN channel model. The Eb/No value will be set to a value below that of the required specification and then the BER will be observed to be below a BER of 1×10^{-4}. This same process is repeated with the Rayleigh channel model and specifications.

3.13.3 Matlab pre-decoded audio playback

Furthermore, to show that the BER selected will result in low signal distortion an audio snippet will be played back that has been decoded using the decoder system previously (not decoded live) and therefore has errors up to a BER of 1×10^{-4}. The audio snippet will be played back and compared to the snippet of audio without errors. The reason why this will not be decoded live is that a 10-second snippet of audio will require approximately 10,000 codewords which will take several minutes to decode. This is simply to illustrate what the effect of a BER of the required specification would sound like in audio form.

4. Results

4.1 Summary of results

Table 8 and 9 show the summary of results that have been achieved.

Table 8.
Summary of results part 1

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
The system should be within 2.5 dB of Shannon's limit at a BER of 1x10^-4, in a simulated AWGN channel.	The distance away from Shannon's limit in a simulated AWGN channel at the required BER is 1.83 dB (This is better than what is required).	Section 3.12.12.
The system must decode a codeword sent through a Rayleigh flat-fading channel model in the presence of 24dB Eb/No of signal-to-noise ratio, with a BER lower than 1x10^-4.	The system can decode a codeword sent through a Rayleigh flat-fading channel model in the presence of 22 dB of Eb/No (This is better than what is required).	Section 3.12.15.
The system must decode a codeword in under 1 second.	The system decodes a codeword in 0.25 seconds per codeword in the AWGN case and 0.11 seconds in the Rayleigh channel case.	Section 3.12.11 and 3.12.14.
The codeword length must be within 2000 to 5000 bits per codeword.	The system is implemented on the embedded platform with a codeword length of 2040 bits.	Section 3.12.11 and 3.12.14.

Table 9.
Summary of results part 2

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
The system will be implemented for code-rates above 0.32.	The system is implemented for a code-rate of 0.323.	Section 3.12.11 and 3.12.14.
Field condition requirements and specifications		
The system will function for a Raleigh flat-fading channel model in the range of 1 dB to 25 dB.	The system works for a Rayleigh flat-fading channel model in the Eb/No range specified.	Figure 3.12.15.

4.2 Qualification tests

4.2.1 Qualification test1: AWGN channel BER test

4.2.2 Objectives of test

Determining the BER of the decoder and the run-time of the system in an AWGN channel model.

4.2.3 Equipment used

Software only. The system will count the number of errors.

4.2.4 Test setup

None.

4.2.5 Test steps

Generate random codewords and add AWGN noise to them of the required noise level. Then decode the message and count the number of errors. Repeat these steps for 10000 codewords and count the total number of errors present at the given Eb/No noise level. Repeat this setup at different Eb/No values until the BER achieved is just below the required 1×10^{-4}.

4.2.6 Results or measurements

The lowest Eb/No value at which the system has a BER below 1×10^{-4} is 1.82 dB from Shannon's limit. This comes from the BER curve in figure 35. The average run-time per codeword is 0.25 seconds. This is significantly better than the 1 second per codeword that is required.

4.2.7 Observations

None.

4.2.8 Statistical analysis

The use of 1000 codewords of length (2040,660) means that there $660 \times 1000 = 0.66$ Mbits used in the test therefore a BER of 1×10^{-4} will result in 66 errors.

Statistical analysis is done for 1000 codewords at 1.82 dB away from Shannon's limit. Using 600 samples of tests with 1000 codewords at a significance level of 0.001 leads to a 99.9% confidence interval for the BER of between 0.719×10^{-4} and 0.939×10^{-4}. Therefore, the BER will fall between the two bounds that are below 1×10^{-4} with a 99.9% confidence level at 1.82 dB from Shannon's limit, which is below the 2.5 dB that is required.

4.2.9 Qualification test2: Rayleigh flat-fading channel BER test

4.2.10 Objectives of test

Determining the BER of the decoder and the run-time of the system in a Rayleigh flat-fading channel model.

4.2.11 Equipment used

Software only. The system will count the number of errors.

4.2.12 Test setup

None.

4.2.13 Test steps

Generate random codewords and add Rayleigh flat-fading noise to them of the required noise level. Then decode the message and count the number of errors. Repeat these steps for 10000 codewords and count the total number of errors present at the given Eb/No noise level. Repeat this setup at different Eb/No values until the BER achieved is just below the required 1×10^{-4}.

4.2.14 Results or measurements

The lowest Eb/No value at which the system has a BER below 1×10^{-4} is approximately 22 dB. This comes from the BER curve in figure 36. The average run-time per codeword is 0.11 seconds. This is significantly better than the 1 second per codeword that is required.

4.2.15 Observations

None.

4.2.16 Statistical analysis

The use of 1000 codewords of length (2040,660) means that there $660 \times 1000 = 0.66$ Mbits used in the test therefore a BER of 1×10^{-4} will result in 66 errors.

Statistical analysis is done for 1000 codewords at 22 dB of Eb/No. Using 500 samples of tests with 1000 codewords at a significance level of 0.001 leads to a 99.9% confidence interval for the BER of between 0.928×10^{-4} and 1.05×10^{-4}. Therefore, the BER will fall between the two bounds that are below 1.05×10^{-4} with a 99.9% confidence level at 22 dB of Eb/No, which is below the 24 dB required.

5. Discussion

The system has been successfully implemented in order to provide an error correction system for applications that work in the presence of very high noise channels such as the Rayleigh flat-fading channel. The system implemented in the project is able to solve this problem to a good extent and provides results that are marginally better than expected. The project's performance is in line with the results achieved in other research papers in the presence of an AWGN channel [13] and is able to achieve results that are in line with those achieved in similar error correction code applications in literature [11].

The system is implemented to work well when used for high noise channel models where the speed of communication is not as important as obtaining good relative BERs in adverse channels. The system is also implemented for a code rate of 0.323 which aids good BER performance but is not optimal in situations where overall power consumption is to be optimized. The system will, therefore, like other error-correcting code systems, be wasting energy and bandwidth by transmitting bits that do not contribute to the intended message.

5.1 Interpretation of results

The system's results are better than what is expected in terms of both BER performance and run-time specifications. The system achieved a better run-time result than what was expected as a result of the very good optimized encoder and decoder algorithm that makes use of in-place operations that don't expand the base matrix fully before the decoder algorithm is commenced. The effect of this is that the system is able to run much faster than expected and is able to decode codewords on a mid-range embedded system in a time frame that is under 1 second.

There are no external factors that influenced the measurements made since the BER is simply determined by counting the number of errors in the decoded message when compared to the source message.

5.2 Critical evaluation of the design

The current design has no obvious shortcomings and adheres to the requirements set in the project proposal. The BER performance is better than the required specifications in both the Rayleigh flat-fading and the AWGN channel. The BER and run-time performance are both strong points of the implementation since they are better than expected.

The circumstances under which the system will fail to decode messages with a BER lower than the required 1×10^{-4} is when the Eb/No level is higher than that of the specified results. This will occur if the communication channel over which the message is sent is more adverse than expected/specified.

5.3 Design ergonomics

There are no specific design ergonomics to note in this project.

5.4 Health, safety and environmental impact

There are no specific health, safety and environmental considerations to note.

5.5 Social and legal impact of the design

There are also no social or legal impacts of the design to note.

6. Conclusion

In the given implementation of the system, it can be seen that the system was successfully implemented on an embedded platform in order to obtain a BER that is better than the required specification on both the AWGN and the Rayleigh flat-fading channel. The system is able to decode a codeword in under 1 second per codeword on the embedded platform, which satisfies the run-time requirement for the system. The system is able to decode both a real-world audio signal and also a randomly generated signal.

From the given implementation the system satisfies all specified requirements and is completely up to specification.

6.1 Summary of the work completed

In summary, the project involved three main challenges. Firstly, the design and implementation of an optimized encoder that is able to efficiently encode a message based on the 5G-NR base matrix standard. This involves making use of in-place operations in order to cut down on memory usage and run time. Secondly, the design and implementation of an optimized decoder that is able to efficiently decode a message in presence of high noise in the 5G-NR base matrix standard. This involves making use of in-place calculations that cuts down on memory usage and run time. Thirdly, the system must be implemented on an embedded platform such that the system meets the required run-time requirement and also doesn't run out of memory due to the long codeword lengths used in the system.

6.2 Summary of observations and findings

The system is the first LDPC decoder that I could find in literature that has been applied to decoding messages that were sent over a Rayleigh flat-fading channel and therefore is a benchmark when it comes to the performance of a probabilistic LDPC system in the presence of a Rayleigh flat-fading channel.

The system is able to decode messages in the presence of very high noise and is able to do so in a short run-time on low memory, low processing power embedded device. This is noteworthy since it provides a benchmark for the performance of future projects in the space of LDPC codes on embedded devices.

6.3 Contribution

6.3.1 New software used

A new programming language used was Java. This was not used in the embedded implementation but was used in PC simulations to test the ultimate performance of the system and used

to cut down on run-time for drawing BER curves which take a great deal of time to run.

A new networking concept learnt was SSH (Secure Socket Shell). SSH was used to communicate between the embedded devices and the PC device on which noise is added and analysis is done. Another communication protocol used was TCP (Transmission Control Protocol). This enabled sharing of files between the embedded devices.

6.3.2 New theory mastered

New concepts and theories mastered include: briefly understanding how belief propagation works in general terms, and how it works specifically in the case of a communication system's parity check matrix, understanding how to encode and decode messages in the 5G-NR base matrix standard. Understanding the working of a model of a Rayleigh channel and how it represents a real-life cellular communication system. Understanding and implementing the optimization proposed in [1] which helps to improve in-place operations to save resource usage and how it is applied to the current LDPC implementation.

6.3.3 New hardware used

The only new hardware used is the Beaglebone Black Rev C. Two of these are the embedded devices on which the encoder and decoder run.

6.3.4 Own contributions

The basic/unoptimized encoder and decoder are completely developed from first principles without any code used from other sources. The optimized decoder structure (only the structure) is taken from [1] and is heavily adapted by swapping the row and column operations to speed up run-time, layering is added in order to allow quicker convergence on the desired bit state. Normalization and offset are added in order to test the updated BER performance. The sum-product algorithm is also completely implemented from first principles with no code taken from any other resources. The code for the implementation of a Rayleigh flat-fading channel model is adapted from the mathematical model of the Rayleigh flat-fading channel shown in [14].

6.3.5 Study leader interactions

Bi-weekly meetings involved showing my progress and asking guiding questions about how to improve the quality of my work and also how to improve the breadth of the investigation done to find the best solution for the given problem. These questions often went along the lines of "should I implement and test 'so-and-so' algorithm?" and "I have compared my performance to 'this research paper's results' is this performance what you would expect?". We also discussed on many occasions whether or not the work that I had done is in line with work that has been done in previous years and if my work is of the high standard that is expected to do

well in the exam.

6.4 Future work

Future work that can be taken on is to adapt the project to work for other coding schemes such as QPSK and QAM. The system can also be adapted and tested on other communication channel models such as the Rician channel.

The system can also be applied on an embedded platform that supports Java. Since this was the fastest language tested it will allow for a speed improvement over the C++ implementation.

Other than the recommendations provided above there is no known method that can be used to improve the overall BER performance of the system other than implementing the system for even longer codewords than that of the 5k codeword lengths tested in the current implementation.

Further investigation can be done into discovering why the introduction of layering provided no visible performance benefit when results from literature [17] show that it should have provided improved BER performance.

There can be further investigation done into determining how to improve the performance of the system in the presence of a Rayleigh flat fading channel. This can lead to a study into determining a more optimal equation to be used in the min-sum algorithm to more accurately model the intrinsic beliefs of the channel that are sent to each bit in the decoding process.

7. References

[1] A. Thangaraj, "LDPC and Polar Codes in 5G Standard (online course)." Electrical Engineering department NPTEL-NOC IITM, January–February 2018.

[2] R. Gallager, "Low-density parity-check codes," *Information Theory, IEEE Transactions on*, vol. 8, no. 1, pp. 21–28, 1962.

[3] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[4] T. Richardson and R. Urbanke, "The renaissance of Gallager's low-density parity-check codes," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 126–131, 2003.

[5] P. Elechi and B. Bakare, "Performance Analysis of BER and SNR of BPSK in AWGN Channel," *International Journal of Digital Analog Communication Systems*, vol. 7, pp. 27–38, 02 2022.

[6] Z. Wang, Z. Cui, and S. Jin, "VLSI Design for Low-Density Parity-Check Code Decoding," *Circuits and Systems Magazine, IEEE*, vol. 11, pp. 52 – 69, 02 2011.

[7] G. Forney, *Lecture notes in Principles of digital communication 2, MIT 6.451*, 2005, pp. 35–41.

[8] D. Costello and S. Lin, *Error Control Coding*, 2nd ed. Prentice-Hall, Inc., 2004, pp. 851–947.

[9] M. Viswanathan. Q function and Error functions : demystified (Gaussian Waves website). [Online]. Available: https://www.gaussianwaves.com/2012/07/q-function-and-error-functions/

[10] H. B. Celebi, "Noise and multipath characteristics of power line communication channels," *Masters Thesis, Univ. of South Florida*, 2010.

[11] R. van Nobelen, "Coding for the rayleigh fading channel," *Univ. of Canterbury Christchurch Press*, 1996.

[12] K. Yang, J. Feldman, and X. Wang, "Nonlinear programming approaches to decoding low-density parity-check codes," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1603–1613, 2006.

[13] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[14] M. Viswanathan, *Simulation of Digital Communication Systems Using Matlab*. Independently published, 2020, pp. 135–145.

[15] H. Li, B. Bai, X. Mu, J. Zhang, and H. Xu, "Algebra-assisted construction of quasi-cyclic ldpc codes for 5g new radio," *IEEE Access*, vol. PP, pp. 1–1, 09 2018.

[16] N. Chelikani. 5G-NR DL-SCH LDPC Channel Coding Base Graph selection and Coding Procedure (Linkedin Post). [Online]. Available: https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani/

[17] L. Wang, "Implementation of Low-Density Parity-Check codes for 5G NR shared channels," *Masters Thesis, KTH Royal Institute of Technology school of Electrical Engineering and Compute Science, Stockholm Sweden*, 2021.

[18] T. Thi Bao Nguyen, T. Nguyen Tan, and H. Lee, "Low-Complexity High-Throughput QC-LDPC Decoder for 5G New Radio Wireless Communication," *Electronics*, vol. 10, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/4/516

Part 4. Appendix: technical documentation

Technical Documentation

Record 1: System block diagram

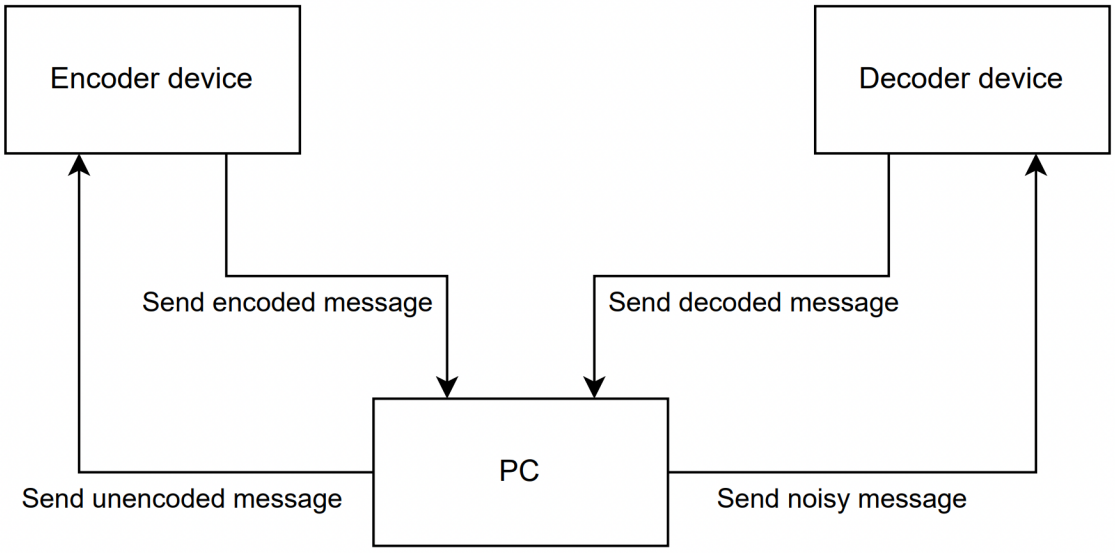


Figure 37.
System block diagram

Record 2: Systems level description of the design

The system communicates in the following way, as shown in figure 38.

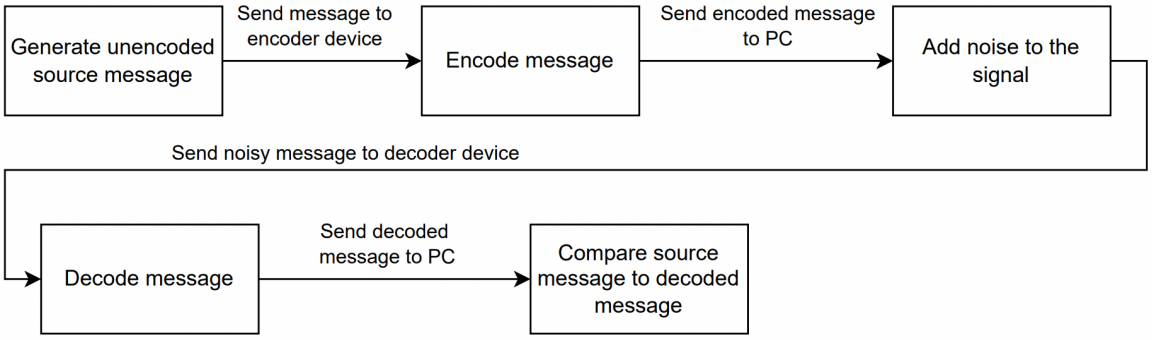


Figure 38.
Communication diagram

Record 3: Complete circuit diagrams and description

Not applicable.

Record 4: Hardware acceptance test procedure

1. Connect power supplies to the DC barrel jack of the encoder and decoder embedded devices as well as the network router.

2. Connect an RJ45 network cable between LAN port 1 of the network router and the PC.
3. Connect an RJ45 cable between LAN port 2 of the network router and the encoder device.
4. Connect an RJ45 cable between LAN port 3 of the network router and the decoder device
5. Download "IP scanner" software on the PC device and find the IP address of the encoder and decoder devices.
6. Copy the IP addresses into the following terminal command:

   ```
   "ssh debian@<IP address of device>"
   ```

7. Repeat the previous step for the decoder device.

Record 5: User guide

1. Execute steps from the Hardware acceptance test procedure above (section 6.4).
2. Create local folders called "awgn" and "rayl" in which the system will work and create files and results during running.
3. Substitute the file locations and IP addresses into the following command and run the command as a one-line command in the terminal on the user's PC device.

   ```
   cd <local awgn folder location>;
   g++ generate_message.cpp -o generate_message;
   ./generate_message;
   scp message_unencoded.txt debian@<Encoder IP address>:
   ~/ssh_testing/awgn;
   ssh -t debian@<Encoder IP address> "cd ~;
   cd ssh_testing/awgn/; g++ encoder.cpp -o encoder;
   ./encoder";
   scp debian@<Encoder IP address>:
   ~/ssh_testing/awgn/message_encoded.txt
   <local awgn folder location>;
   g++ add_noise_awgn.cpp -o add_noise_awgn;
   ./add_noise_awgn;
   scp message_noisy.txt debian@<Decoder IP address>:
   ~/ssh_testing/awgn;
   ssh -t debian@<Decoder IP address> "cd ~;
   cd ssh_testing/awgn/; g++ decoder.cpp -o decoder;
   ./decoder";
   scp debian@<Decoder IP address>:
   ~/ssh_testing/awgn/message_decoded.txt
   <local awgn folder location>;
   python3 compare_messages.py
   ```

4. Run the command in the terminal on the PC device and enter "temppwd" as the password when prompted.
5. Observe the results of the run of the software in the AWGN channel.
6. Substitute the file locations and IP addresses into the following command and run the command as a one-line command in the terminal on the user's PC device.

```
cd <local rayl folder location>;
g++ generate_message.cpp -o generate_message;
./generate_message;
scp message_unencoded.txt debian@<Encoder IP address>:
~/ssh_testing/rayl;
ssh -t debian@<Encoder IP address> "cd ~;
cd ssh_testing/rayl/; g++ encoder.cpp -o encoder;
./encoder"; scp debian@<Encoder IP address>:
~/ssh_testing/rayl/message_encoded.txt
<local rayl folder location>;
g++ add_noise_rayl.cpp -o add_noise_rayl;
./add_noise_rayl;
scp message_noisy.txt debian@<Decoder IP address>:
~/ssh_testing/rayl;
ssh -t debian@<Decoder IP address> "cd ~;
cd ssh_testing/rayl/;
g++ decoder.cpp -o decoder; ./decoder";
scp debian@<Decoder IP address>:
~/ssh_testing/rayl/message_decoded.txt
<local rayl folder location>;
python3 compare_messages.py
```

7. Run the command in the terminal on the PC device and enter "temppwd" as the password when prompted.
8. Observe the results of the run of the software in the Rayleigh flat-fading channel.

Record 6: Software process flow diagrams

The communication of the system is shown in figure 38. The encoder operates as shown in the following pseudo-code in algorithm 10. The decoder operates as shown in the flow chart in figure 39.

Algorithm 10 Encoder algorithm

```
parity_array = row matrix of zeros of length Zc
for x = 0:4 do
    for y = 0:22 do
        shifted_array = shift_mat(message[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
end for
codeword[22*Zc:23*Zc] = shift_mat(parity_array, Zc)
find P2 to P4 below
for x = 0:3 do
    Reassign parity_array matrix's values to zero
    for y = 0:23+x do
        shifted_array = shift_mat(codeword[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
    codeword[(23+x)*Zc:(24+x)*Zc] = parity_array
end for
find P5 to P46 below
for x = 4:46 do
    Reassign parity_array matrix's values to zero
    for y = 0:26 do
        shifted_array = shift_mat(codeword[y*Zc:(y+1)*Zc], B[x,y])
        parity_array = parity_array XOR shifted_array
    end for
    codeword[(22+x)*Zc:(23+x)*Zc] = parity_array
end for
```

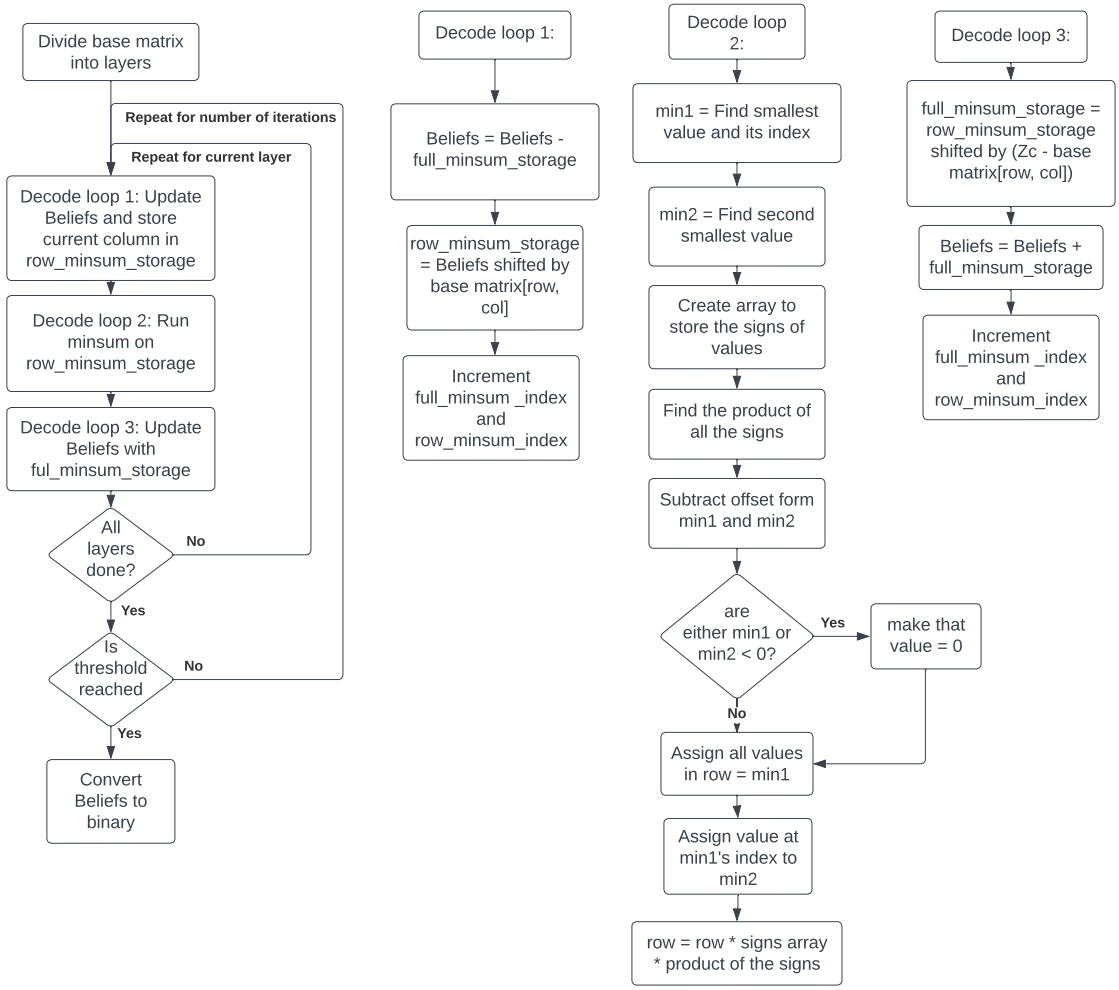


Figure 39.
Decoder algorithm flow chart

Record 7: Explanation of software modules

Explained in the algorithm and flow chart above.

Record 8: Complete source code

Complete code has been submitted separately on the AMS.

Record 9: Software acceptance test procedure

Explained in User guide section (6.4).

Record 10: Software user guide

Explained in User guide section (6.4).

Record 11: Experimental data

The results of running the system on the embedded platform are given below in figures 40 and 41.

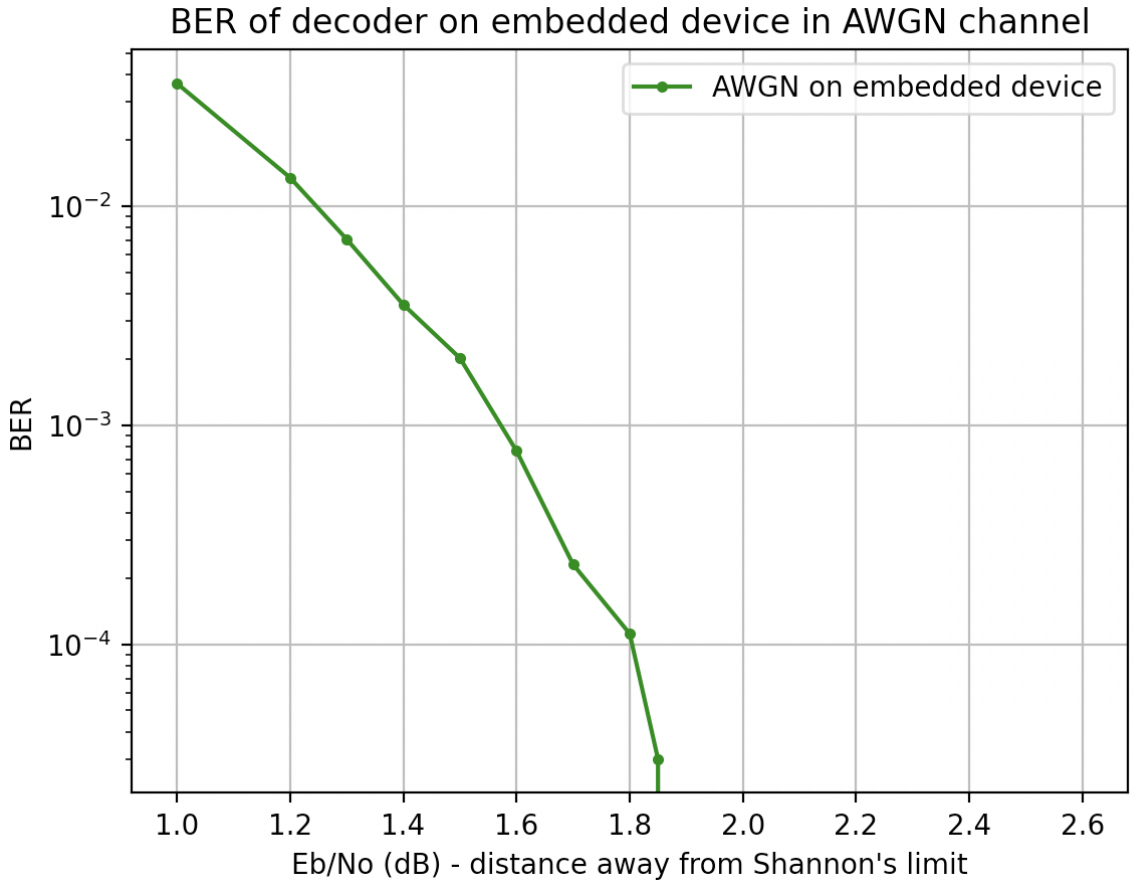


Figure 40.
AWGN results

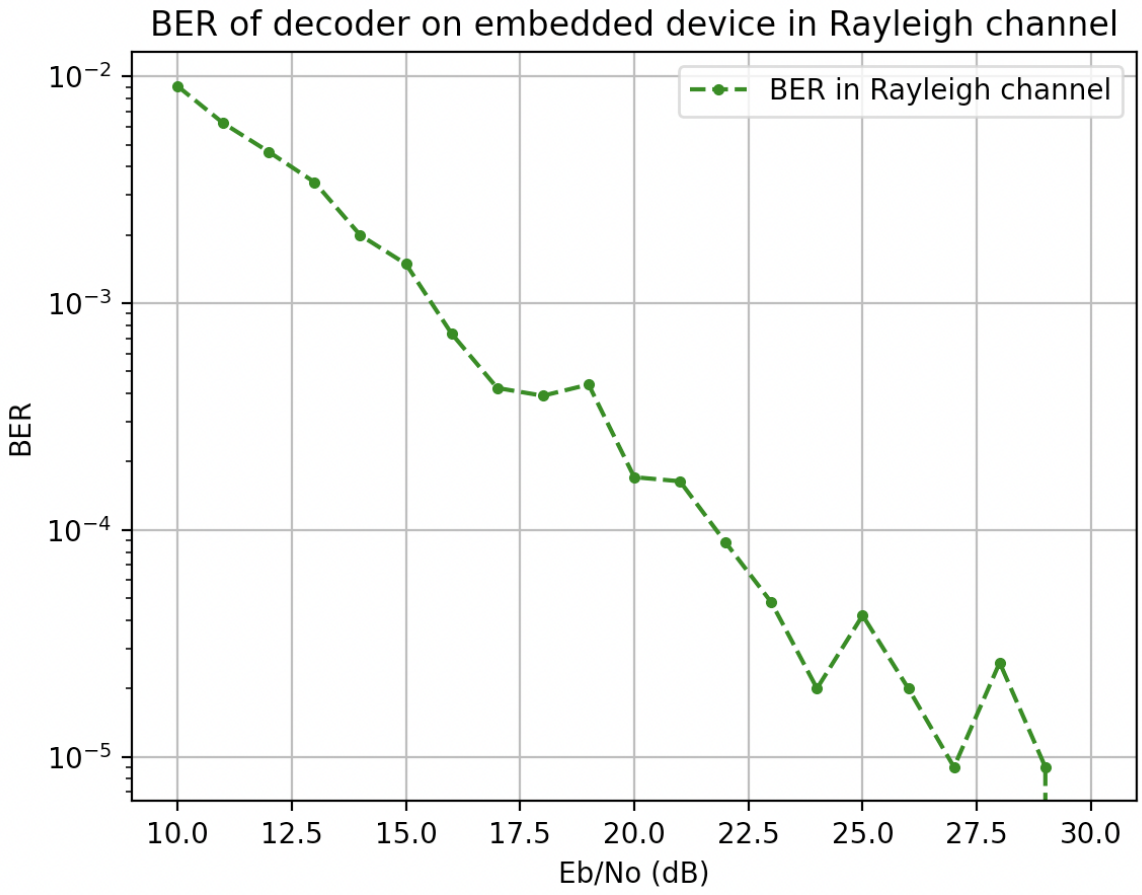


Figure 41.
Rayleigh flat-fading channel results