DESIGN OF DIRECT ERROR PATTERN TESTING (DEPT)-BASED ITERATIVE DECODER FOR OPEN FORWARD ERROR CORRECTION (OFEC) STANDARD

by

Xiaoting Huang

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
April 2023

Table of Contents

Abstract

oFEC Forward Error Correction (oFEC) is an error correction coding standard for fiber optical communication, which is introduced as an optical transport network (OTN) with high efficiency and code rate. oFEC code was introduced and standadized by Open ROADM [16] targeting metro applications. The code is typically decoded by Chase-Pyndiah algorithm provides an NCG of 11.1dB for BPSK/QPSK with pre-FEC BER of $2e^{-2}$ and 11.6dB for 16QAM after three soft-decision iterations. Recent research has focused on adapting decoding algorithms to oFEC in order to enhance its performance. This is crucial since even a small enhancement can lead significantly boost the transmission distance and efficiency of fiber. People are currently interested in some new algorithms for decoding short codes, such as DEPT and GRAND presented by recent research. These two algorithms have good error correction performance for redundant error correction codes.

The existing research has applied Chase-Pyndiah algorithm on the oFEC and proposed simplified schemes based on power dissipation and correction threshold at 10^{-15} BER. In this thesis, we proposed to apply the concept of short-code decoding and adapt it for iterative coding of oFEC code, so that the error correction performance of oFEC can be further enhanced. Specifically, the suggested short-code decoding algorithm, which includes the DEPT and the DEPT-ORBGRAND algorithm, would optimize the design of PEPs in accordance with the unique properties of oFEC in order to accommodate its specific structure. The external information scale factor employed by the Pyndiah method is improved based on the unique oFEC code structure in order to significantly enhance the performance of oFEC. In addition, the study suggests using table-based hard decoders to replace conventional hard-decision decoders such as the Berlekamp-Massey algorithm. Using the improved decoder, the performance of oFEC can achieve a gain of around 0.22 dB, which is a significant enhancement for the transmission of fiber-optic systems.

List of Abbreviations

FEC Forward Error Correction

oFEC Open Forward Error Correction

NCG Net Coding Gain

BPSK Binary Phase Shift Keying

QPSK Quadrature Phase Shift keying

QAM Quadrature Amplitude Modulation

DEPT Direct Error Pattern Testing

ORBGRAND Ordered Reliability Bits Guessing Random Additive Noise Decoding

PEP Partial Error Pattern

HDD Hard Decision Decoding

SDD Soft Decision Decoding

LLR Log Likelihood Ratio

SISO Soft-in Soft-out

SIHO Hard-in Hard-out

LRB Least Reliable Bits

AWGN Additive White Gaussian Noise

BCH Bose-Chaudhuri-Hocquenghem

SNR Signal To Noise Ratio

Acknowledgements

First and foremost, I am deeply grateful to Professor Dmitry Trukhachev for his guidance and supervision during the research process. His profound knowledge and research experience provides me with many valuable suggestions and guidance during the research process. He used his patience and encouragement to support me in completing this research, especially when the research was in trouble and difficult to advance. I would also like to thank him for his ongoing financial support.

I would also like to thank Professor Kamal EI-Sankary for his support and introduction, which gave me the opportunity to get in touch with this research field and become interested in it, as well as his valuable advice during the research process.

I would also like to thank Reza Hadavian for helping me to start the work on DEPT-ORBGRAND for oFEC, actively discussed relevant content with me during the research process, which greatly accelerated the progress of the project.

I want to thank the other members of my committee. We thank them for taking time out of their busy schedules to evaluate and discuss this study.

I would also like to sincerely thank the Tri-agency and Dalhousie University for their scholarship funding for their financial support.

Finally, I would like to thank my family for their unconditional support, tolerance and encouragement.

Chapter 1

Introduction

1.1 Background

With the development of fiber-optic communication systems, the primary goal of researchers is to maximize transmission speed, precision, and capacity. Forward error correction (FEC), which is the typical optical transport network (OTN), plays a vital role in high-speed fibre optical communication systems due to its high dependability of digital transmission with reduced complexity and resource consumption. In 2000, the ITU-T recommendation G.975 [14] standardized the first-generation FEC coding for the first time. Reed-Solomon (255,239) was selected as the component codes for the FEC frame structure utilized on the 2.5 Gbits/s submarine optical fibre cable systems. The RS(255,239) codes, often referred to as Generic FEC(GFEC), are able to correct up to 8 incorrect byte-symbols out of 255 codeword length with 6.69% overhead and 6.2dB of NCG.

In 2004, ITU-T recommendation G975.1 [19] established Super FEC techniques for dense wavelength division multiplexing (DWDM) submarine systems with large data rates. Combining two FEC codes, such as RS codes and BCH codes, is the principle for achieving better error correction than the first generation FEC. The greatest performance of the suggested scheme can reach 9.4dB of NCG with 25% redundancy. In the years that followed, the staircase code was presented in 2012 as a new class of FEC codes that could be implemented with a data rate of 100Gb/s. Simultaneously, the hard-decision (HD) Staircase code is specified in ITU-T G.709.2 as the first coherent FEC, capable of achieving 9.38dB of NCG with a pre-FEC BER threshold of 4.5e-3. Similar iterative decoding codes, such as Block Turbo Codes (BTCs), Low Density Parity Check (LDPC) codes, and product codes [6], among others, are increasingly being developed.

Due to the great coding gain and reliability of iterative decoding, the topic of iterative decoding has always been considered worthy of in-depth research. The mechanism behind this is to divide the decoding task into several stages, where the decoding result from the previous stage is used to generate the result of the next step, etc. The applicable well-known decoding construction, such as BTCs, staircase codes [17], etc. Unlike hard-decision (HD) iterative decoding, soft-decision (SD) iterative decoding utilizes LLR to transmit data. Extrinsic information needs to be generated in order to bring LLR information up to date before the next iteration, so that it can be used to deliver new information. To boost the feasibility of iterative decoding, the complexity of each iteration's decoding algorithm is always one of the primary research concerns. Therefore, researchers are committed to maximizing decoding performance and data integrity while minimizing the complexity of the system.

Other than that, concatenated code systems first introduced in [8] are also a common choice for error-correction code construction due to its outstanding correction performance and minimal complexity. The basic principle behind concatenated code systems is to series combine two or more codes, often one high-rate outer code and one low-rate inner code. The inner decoder is the first to be developed and is tasked with the responsibility of correcting errors up to their threshold. After deinterleaving, the rest mistakes are corrected by the outer decoder. Recent research has shown that concatenated code systems are frequently used in conjunction with iterative decoding methodology. BTCs is a good illustration of this type of system, [13] [3]. Recent research in [2] investigates the implementation of concatenated FEC with 400Gbps data capacity.

1.2 Motivation

Recently, a novel FEC scheme called Open Forward Error Correction (oFEC) was presented in the Open ROAMD specification document [16], which is designed for high-throughput fiber-optical communications. The architecture of oFEC exhibits a strong resemblance to that of braided block codes (BBCs), which is accomplished by connecting two different block component codes in order to produce an two-dimensinal array. Such that both component codewords examine each other for errors while the parity symbols of one component codeword serve as input for the second component

codeword. As the infinite bit stream braided constantly, the sliding window moved along the diagonal in an infinitely continuous manner. The oFEC system utilizes iterative soft-decision (SD) decoders with a semi-infinite block-based window of memory in order to achieve high efficiency and high coding rate. Instead of connecting two different component codewords, oFEC divides component codewords in half and interconnects them in a predetermined permutation pattern. The primary advantage of the oFEC over the FEC is that, due to the parallel decoding engine design, the oFEC has a very high code rate and can execute a large number of codewords concurrently. In addition, its error correction capabilities is remarkable as a result of the special bit permutation explained in detail in the following section. The obvious drawbacks of oFEC are its latency and its high power consumption, both of which grow dramatically with the number of iterations performed and the level of difficulty of the decoding algorithm that is being used. In a recent article [22], the authors proposed that the oFEC is capable of attaining 400Gbps over optical connection with a correction threshold of 1.81e-2. OFEC is a potential FEC scheme that should be researched and simplified further in order to achieve a data throughput of 800Gbps in the near future.

In this thesis, the DEPT-based decoding algorithms, including DEPT-Pyndiah, and DEPT-ORBGRAND-Pyndiah, are utilized on the oFEC as the alternative decoding algorithms of Chase, and the performance of the DEPT-based decoding algorithms are compared with Chase in terms of accuracy and efficiency. According to the Open ROAMD specification documentation [16], the decoding technique used by oFEC refers to Turbo Product Codes (TPCs), also known as block turbo codes (BTCs), which are formed by serial and parallel concatenation of simple block codes on a matrix with multiple dimensions. Since the introduction of the Chase [1] and Pyndiah [13] algorithms in 1972 and 1998, respectively, the Chase-Pyndiah algorithm has been one of the most frequently adopted iterative soft-in soft-out (SISO) decoding methods in the construction of TPCs. Hence, the Chase-pyndiah algorithm has been presented as one of the SISO decoding methods that can be utilized with oFEC codes, and its implementation has been described in [22]. Throughout the research, [22] implements oFEC using a record 400Gbps 100-chip FPGA and explores the power consumption and correction threshold of oFEC under different iterations and test patterns in each

SISO. The optimal correction threshold is then suggested to be 1.8e-2.

Other than the Chase-Pyndiah algorithm, the DEPT, which was proposed in [21] and the DEPT-ORBGRAND, which was introduced in [5] and [4], are proposed to be used with oFEC as the alternative decoding algorithms in order to improve performance even further than that of utilizing Chase in this work. These algorithms are intended to compete with the Chase algorithm on error sequence generation, for which the DEPT and DEPT-GRAND proposed a method that would choose error patterns based on syndrome as opposed to generating all potential error patterns, which would significantly improve the system's efficiency. In the subsequent sections, the fundamental concepts of decoding algorithms will be explained in further detail. Evaluation and comparison of the algorithms are in the final section of the paper.

1.3 Outline of the Thesis

Section II describes related coding structures, including braided codes, zipper codes, and staircase codes. Section III describes the mechanism of the decoding algorithms applied with oFEC. In Section IV, the oFEC structure's construction is discussed in detail, including an overview of the structure, the iterative encoding and decoding process, bit permutation, and memory design. Section V describes the oFEC implementation procedures in depth, including the elaborate details of each decoding algorithm's adaptation to oFEC. In addition, Section V also further compares and analyzes the simulation results when the DEPT-based decoder employs different numbers of error patterns with the results of applying the Chase. Lastly, the performance of DEPT-based decoding algorithms applied to oFEC is summarized in Section VI, and related future work is discussed.

Chapter 2

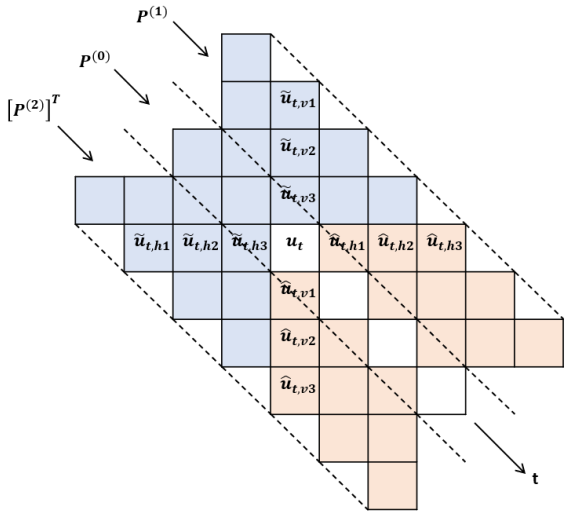
Coding Structure

2.1 Braided Block Codes

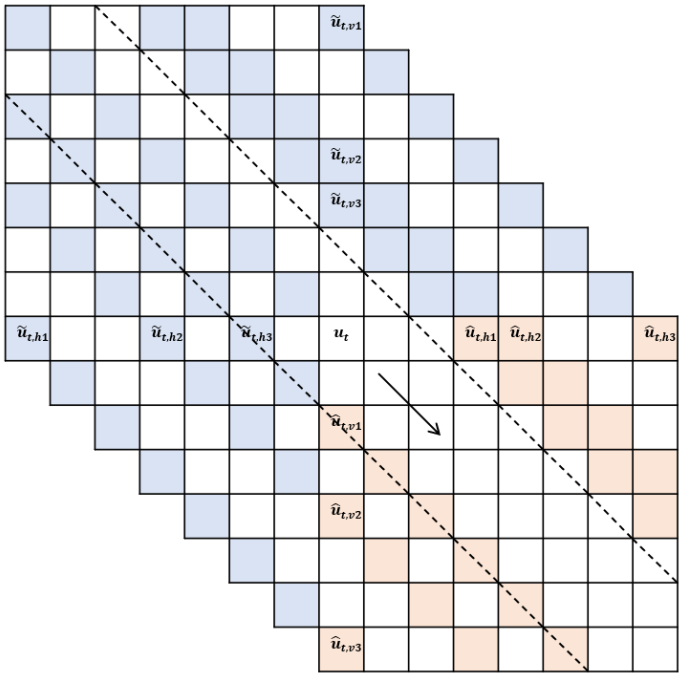
As the fundamental reference structure of oFEC, braided block codes (BBCs) are introduced throughout [10], [20] and [7]. The scheme can be thought of as a family of sliding codes because it is made up of two interconnected block codes that are braided continuously in both the horizontal and vertical directions. However, in comparison to the open FEC, the scheme uses a different permutation. Typically, BBCs are broken down into two subclasses: tightly braided block codes (TBBCs) and sparsely braided block codes (SBBCs). Component codewords are braided and positioned in both the horizontal and vertical directions during the encoding and decoding process. This is in contrast to the normal Elias product code [6]. After each encoding and decoding operation, the array will slide indefinitely down its diagonal, and the previously decoded symbols will be utilized to determine the parity check symbols of newly supplied codewords.

The framework of a typical TBBC graphic representation is depicted in Figure 2.1a. At each time slot t, a certain number of information bits, denoted by u_t, are inserted onto a diagonal centre ribbon with a width of 1. Simultaneously, the information bit u_t is encoded with three horizontal and vertical parity check bits and generates three additional bits. At a time slot of t, for instance, when u_t is input and horizontally concatenated with $[\tilde{u}_{t,h1}, \tilde{u}_{t,h2}, \tilde{u}_{t,h3}]$, new bits $[\hat{u}_{t,h1}, \hat{u}_{t,h2}, \hat{u}_{t,h3}]$ are generated after encoding. Likewise, when u_t is vertically concatenated with $[\tilde{u}_{t,v1}, \tilde{u}_{t,v2}, \tilde{u}_{t,v3}]$, new bits $[\hat{u}_{t,v1}, \hat{u}_{t,v2}, \hat{u}_{t,v3}]$ are generated.The blue-shaded area represents bits that have already been encoded, while the red-shaded area represents bits that will be produced in consecutive time periods.

However, the symbols stored in the memory cells of sparsely braided block codes (SBBC) are not contiguous; the array cells are split as depicted in Figure 2.1b. Iterative decoding is more efficient for them due to their larger capacity and low storage



(a) TBBC.



(b) SBBC.

Figure 2.1: Grapghic representation of BBC with component codes of Hamming (7,4) codes.

density. The width of the BBCs scheme is determined by the length of the component codeword; if the width of the TBBCs scheme is m, the width of the SBBCs scheme is 2m-1. So far, the width of the oFEC scheme is defined by the component codeword employed and the number of iteration execute. The research from 2005 [24] presents block convolutional codes (BCCs), which utilize convolutional codes as component codes of BBCs, and convolutional permutor. This research [25] suggests that sparsely braided convolutional codes with iterative decoding offer outstanding convergence performance. Utilizing the statistical Markov permutation model, it was also determined that braided convolutional codes provide better distance qualities than conventional turbo codes.

2.2 Zipper Codes

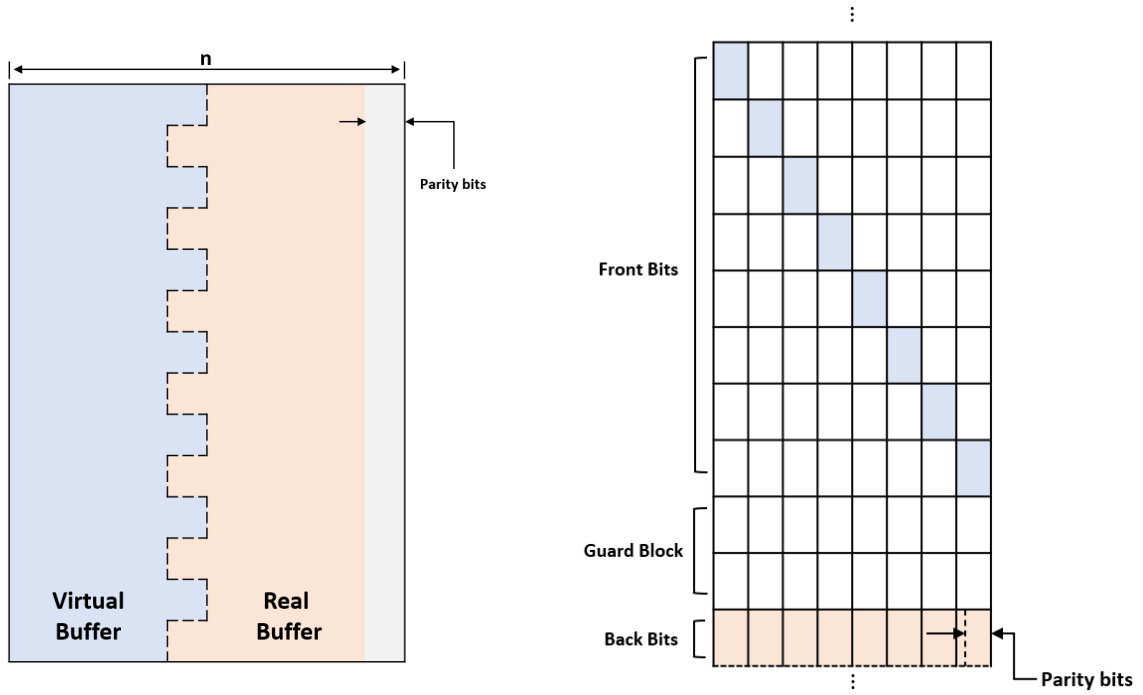
The concept of zipper codes is presented as an innovative framework for describing spatially-coupled product-like codes in [18]. The codes make reference to a variety of well-known constructs, including BBCs, staircase codes, and swizzle codes, amongst others. The structure of a zipper code is comprised of a virtual buffer known as A and a real buffer known as B. These two buffers are referred to as a zipping pair (A, B), and their structural form is depicted in Figure 2.2a, with the appropriate oFEC structure form given in Figure 2.2b.

The new message symbols will only fill in the real buffer; the message that is stored in the virtual buffer is made up of symbols that have been replicated from the position specified by the interleaver map. An interleaver map ϕ is bijective if there exists a map demonstrating the following,

$$\phi : A \to B \tag{2.1}$$

$$\phi^{-1} : B \to A \tag{2.2}$$

such that as $a \in A$ and $b \in B$, it follow the rules $\phi^{-1}(\phi(a)) = a$ and $\phi^{-1}(\phi(b)) = b$. Consequently, it is required to retain in memory some previously encoded and decoded symbols in order to prepare for future decoding. In this regard, oFEC is fairly comparable to zipper in that it is necessary to retain the earlier symbols in order to utilize them for the newly input symbols. In the case of oFEC, the component codewords are divided into front and back bits prior to being stored in distinct regions



(a) Graphical representation of zipper codes.

(b) Graphical representation in form of oFEC.

Figure 2.2: Zipper codes verse oFEC codes.

of memory. This is identical to how the symbols of zipper codes are separately stored in the virtual and real buffers. The newly input symbol is referred to as back bits, which must be concatenated with front bits prior to encoding and decoding, whereas front bits are retrieved from the old memory buffer, which contains encoded and decoded old symbols.

Moreover, [18] describes a subset of the Zipper codes, including tiled diagonal zipper codes and delayed diagonal zipper codes. As the name suggests, tiled diagonal zipper codes are characterized by their "tile-like" structure. Assuming the structure comprised of a number of $w \times w$ tiles, let's the width of the virtual or real buffer is denoted as $m = wL$, such that the width of the entire structure denoted as $n = 2m = 2wL$. If $T_{q,s}$ represent a tile in the q-th row and s-th column, then the interleaver map of a tile can be expressed as $T_{q-s-1,L+s}$, which satisfies the property of bijective, $T_{q,s} = T_{q-s-1,L+s}$. Delayed diagonal zipper codes are variants of tiled diagonal zipper codes with w=1 and with added "delay" δ to the interleave map, demonstrates in

Equation 2.3,

$$\phi_\delta(i,j) = (i - j - \delta, j + m) \tag{2.3}$$

where $i, j \in w$, and $\delta \in \mathbb{Z}$. In the case of Zipper codes, the stall pattern appears as a collection of errors that cannot be rectified through iterative decoding. So, the decoder must identify potential error locations based on some hypotheses on the syndrome. The syndrome is a term used commonly in error-correcting coding to identify the error locations, which are derived by multiplying the encoded message by the transpose of the parity check matrix H. In general, if the received sequence is error-free, the syndrome is a zero vector; otherwise, it is non-zero. The syndrome will indicate the exact error locations for a single error-correcting code, such as the Hamming code. However, for codes that can correct more than one error, a table of possible error positions will be constructed instead, mapping syndromes to corresponding errors. Then the position of errors can be searched from the table based on the syndrome. For Zipper codes, if a component code is t-error-corrected, for example, the decoder will make assumptions and examine possible error locations based on the fact that the error is less than t, and then flip associated bits.

2.3 Staircase Codes

Another structurally comparable code is staircase codes, which were presented in [17] as a family of hard-decision algebraically decodable codes with near-capacity performance. It was suggested as an improvement to the techniques of forward error correction (FEC) that are outlined in ITU-T G.975.1 [19]. The structure of the code is stair-like, as its name suggests, and the neighbouring blocks can be utilized to correct the input error of codewords that have an unterminated length. Recent studies have concentrated on the exploration of effective implementations for increasing throughput speed, reducing latency, and lowering power consumption. To achieve high throughput, [23] developed the staircase code with overhead ranging from 6.25 to 33.3%, reducing the complexity and delay of previous codes with a 20% overhead. And [11] investigate how the hardware design can be altered to reduce the amount of power it uses in compliance with the ITU-T recommendation G.709.2/Y.1331.2 [9] and the OIF-400ZR implementation agreement. In the year 2021, the throughput of

the staircase had surpassed 400Gbps with to the installation of the FEC that was outlined in the 400ZR implementation agreement that was carried out using fiber-optic communication lines, [21].

Chapter 3

Decoding Algorithm

It has been suggested that the decoding of the system will be a soft-input and soft-output (SISO) iterative decoding process. The system employs the Chase-Pyndiah algorithm as a typical effective SISO iterative decoding technique, with decoding based on the reliability of the log-likelihood ratio (LLRs) of the channel output. The alternate decoding algorithms, DEPT-Pyndiah and DEPTGRAND-Pyndiah are applied concurrently with the competing error pattern location technique. Unlike the Chase algorithm, DEPT [21] and DEPTGRAND [4] prefer to identify error patterns with a look-up table (LUT). The coming section will provide a detailed explanation of the oFEC decoding algorithm and a comprehensive overview of the algorithms.

3.1 Chase-Pyndiah Algorithm

3.1.1 Chase Algorithm

David Chase first introduces the Chase algorithm in [1], which is a type of decoding algorithm that utilizes channel measurement information. Wagner decoding [15] introduces the use of channel measurement information with block code decoding. Since the channel measurement information decoding approach is applicable to all block codes, it can also increase the error-correcting capabilities of a given code by a factor of two. Assume that the received sequence, designated by Y, and the possible candidate codewords within the decode radius, denoted by C_i, $C_i = C_1, C_2, C_3...$, are encircled by a Y sphere with a radius of $(d_{min} - 1)/2$, as shown in the shaded area in Figure 3.1.

Based on this approach, [1] presents three techniques for producing codeword sets of C_i. Chase-II algorithm, a standard soft-in hard-out (SIHO) decoding algorithm, is utilized in this study. It only considers error patterns less than $(d_{min} - 1)/2$ errors located outside the set. Thus, p number of least reliable bits (LRBs) are searched

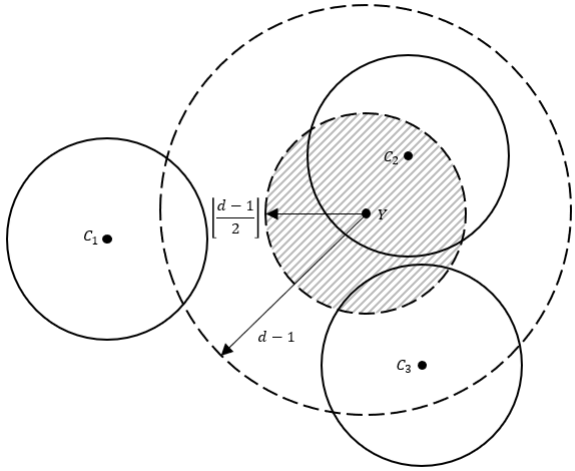


Figure 3.1: Decoding with channel measurement information.

based on the reliability of each bit. The reliability of each bit is established by the absolute value of the LLRs at the channel output; the bigger the absolute LLRs, the greater the reliability in the bit position. As a consequence of this, 2^p number of potential candidate codewords are formed as a result of filling the LRBs slots with different combinations of zeros and ones. The process of candidates generation is described as follows,

Step 1: As the received LLRs output from AWGN channel is given as $R = (r_1, r_2, r_3, \ldots, r_{2N})$, N=128 in this case, the normalized reliability of each bit is determined as $R_{abs} = (|r_1|, |r_2|, |r_3|, \ldots, |r_{2N}|)$.

Step 2: Hard decision of the received LLRs is generated as a sequence $Y_i = (y_1, y_2, y_3, \ldots, y_{2n})$, where,

$$y_i = \begin{cases} 1, & \text{if } \mathrm{r}_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

Step 3: Determine the p least reliable bits from R_{abs} acquired from step 1.

Step 4: Generate 2^p different test patterns T by filling the p least reliable bit positions with a number of 0s and 1s.

Step 5: 2^p of test sequences are generated by perturbing sequences set by using,

$$Z = Y \bigoplus T \tag{3.1}$$

, where $\bigoplus$ denotes modulo-2 addition, Z stands for test sequences, Y stands for the hard decision of LLRs and T stands for error sequences acquired in step 4.

Step 6: Hard decodes 2^p of test sequences to generate 2^p of candidates codewords C_q.

The article [1] introduces the Chase-I and Chase-III algorithms in addition to the Chase-II algorithm. The Chase-II algorithm is the one with the best efficiency and the least amount of complexity among the three. The suggested Chase-I algorithm evaluates all error patterns within the (d-1) radius sphere enclosing the received sequence Y and generates all error sequences with a binary weight less than the minimum distance d. Due to the enormous number of error sequences, Chase-I can only be applied to codes with a limited minimum distance. The Chase-III algorithm is comparable to the Chase-II algorithm, but it offers an extra strategy for lowering the number of error sequences by dividing them depending on the number of d_{min} stands for the code's minimum distance. This helps to reduce the total number of error sequences. As I is the index of candidates if the code has an even number of d, its value is given by i=1,3,...,d-1; if d is an odd number, i is given by i=0,2,3,...,d-1; and when i=0, the test pattern is all zero. Chase-III may be inferior to Chase-I and Chase-II, but its application on codes with a large minimum distance is desired, according to the article.

3.1.2 Pyndiah Algorithm

The article [13] provides a description of the decoding technique that is used to estimate the log-likelihood ratio (LLR) of binary decisions that are provided by the Chase decoder with Block Turbo Codes (BTCs). The Pyndiah algorithm is also applicable to this research, as the standard document specifies that any iterative algorithm established for turbo decoding of Product codes can be easily adapted to decode oFEC codewords. To compute the output LLRs, it was necessary to identify the soft decision of the closest codeword D and the second closest codeword C as competing codewords of D based on the Euclidean distance from the received LLRs, where c_j and d_j are not equal. Then, the codewords D and C will be employed to calculate the normalized output LLRs $r_j^{'}$ utilizing Equation 3.2. In other cases, however, the competing codeword C cannot be located. This is probably due to

the fact that the codeword C is located an excessive amount of Euclidean distance away from the received codeword R. At this stage, Equation 3.3 will be utilized to approximate the output LLRs. After estimating all output LLRs, calculate the extrinsic information W by subtracting it from the input LLRs using Equation 3.4. Then update the received codewords for the succeeding iterations using Equation 3.5, where [R] is the sequence that was received for the first iteration and [R(2)] is the sequence that was received for the succeeding iteration, e.g. second iteration. Similarly, $\alpha(2)$ refers to the alpha value used exclusively for the second iteration, and $[W(2)]$ refers to the extrinsic information evaluated in the second iteration. The process of Pyndiah algorithm to estimate the soft output is described as follows,

$$r_j^{'} = \frac{|R-C|^2 - |R-D|^2}{4} \times d_j \tag{3.2}$$

$$r_j^{'} = \beta \times d_j \tag{3.3}$$

$$r_j^{'} = r + w_j \tag{3.4}$$

$$[R(2)] = [R] + \alpha(2)[W(2)] \tag{3.5}$$

Step 1: Map candidate codewords from (0,1) to (-1,1).

Step 2: Compute Euclidean distance from received sequence R to candidate codewords C_q respectively by using,

$$d(R, C_q) = (R - C_q)^2 \tag{3.6}$$

Step 3: Find the codeword D with minimum Euclidean distance, and the competing second closest codeword C, which satisfies $c_j \neq d_j$.

Step 4: Computes soft output $r_j^{'}$ with given C and D by using Equation 3.2 and Equation 3.3 on different conditions.

Step 5: Determine extrinsic information W with Equation 3.4.

Step 6: Update the LLRs for succeeding iterations with applying Equation 3.5.

The decoding technique relies heavily on the weighting factor α and the reliability factor β. Since the standard deviations of the received codeword R and extrinsic

information W are different, especially during the initial iteration of decoding, the scaling factor α is utilized to reduce the effect of W given the relatively high SNR. The scaling factor α will steadily approach one as the number of iterations increases. Because the conditions are different for every scenario, it is impossible to specify the values of the factors. The ideal value is obtained by analyzing the experimental data. In the case of the oFEC code, the factors may vary between the front and back bits, as the divided front and back bits of the codewords are placed in separate block areas, resulting in the front and back bits locating in a different number of iterations with varying SNR in a given time slot. Based on the results of the experiment, the ideal factors for the front bits are determined as,

$$\alpha(j) = [0.2, 0.3, 0.5, 0.6, 0.9, 1.0, 1.0] \tag{3.7}$$

$$\beta(j) = [0.2, 0.4, 0.6, 0.7, 1.0, 1.0, 1.0] \tag{3.8}$$

Due to the unique circumstances of the front and back bits, the number of α and β values is likewise doubled. Assuming that the entire number of iterations is recorded as j, the decoding process requires 2j parameters in total. The parameter $\alpha(2j)$ and $\beta(2j)$ are used while front bits are decoded, and back bits are decoded using $\alpha(2j-1)$ and $\beta(2j-1)$.

3.2 DEPT Algorithm

On the implementation of a soft-input hard-output (SIHO) concatenated forward error-correction (FEC) decoder for the inner Hamming code with 400Gps fiber-optical communication connections in 2021, the DEPT method is introduced [21]. It was suggested that the number of errors may be determined by the last term of the syndrome with the parity-check matrix H as a reference point. As two sets of partial error patterns (PEPs) are predefined for scenarios of odd and even amounts of errors if s$\neq$0 and the final bit of syndrome is 1, the codeword is decoded on the assumption that it has an odd number of errors (e.g. e=1,3,5,7), and error patterns are searched in the odd PEPs set; otherwise, search the even PEPs set (e.g. e=0,2,4,6).

The predefined PEPs for odd and even cases can be used to query for error patterns that clearly identify likely error locations, based on the index of the least reliable

LLRs. It will determine the positions of these 1's by computing the vector e and generating symbols notation for potential error patterns based on the least reliable absolute LLRs. For instance, the notation (1,2,4) indicates that the first, second, and fourth bits of the LRB are 1 and include errors.

All the error patterns are queried from predefined PEPs except for the last error, which is the last 1 to be added to the error patterns. In order to estimate the position of the last error, it is required to determine a term of $s^{'}$, which is obtained by adding the syndrome of received sequence with the (N-1) component column of H matrix as shown in Equation 3.10,

$$H = \begin{bmatrix} \alpha^{n-1} & \alpha^{n-2} & ... & \alpha & 1 \\ \alpha^{3(n-1)} & \alpha^{3(n-1)} & ... & \alpha^{3} & 1 \\ \alpha^{5(n-1)} & \alpha^{5(n-1)} & ... & \alpha^{5} & 1 \\ ... & ... & ... & ... & ... \\ \alpha^{(2t-1)(n-1)} & \alpha^{(2t-1)(n-1)} & ... & \alpha^{2t-1} & 1 \end{bmatrix} \tag{3.9}$$

$$s^{'} = s + h_{j_1} + h_{j_2} + ... + h_{j_{N-1}} \tag{3.10}$$

Then figure out the corresponding index of the error via the function $f(s^{'})$ defined below.

$$f(s^{'}) = \begin{cases} j^{'}, & \text{if } s^{'} = h_j \\ -1, & \text{otherwise} \end{cases}$$

,where $j \in 1, 2, ..., N$. For the case of the number of error is even, $s_{17} = 0$, the last two error positions are required to identify instead. Let's define h_i and h_j as the two component columns of the H matrix with the length of (2N-k), and satisfies $i \neq j$. Such that, instead of compare the acquired term $s^{'}$ with h_j, in order to identify the last two error positions $s^{'}$ required to compare with $h_i + h_j$, which is express as the function below,

$$f^{'}(s^{'}) = \begin{cases} \{i, j\}, & \text{if } s^{'} = h_i + h_j \\ -1, & \text{otherwise} \end{cases}$$

The steps to execute the algorithm are described as follows:

Step 1: Determine syndrome of the received codewords, given $s = v_{HD}H$.

Step 2: If the $s = 0$, it means the codeword is error-free. Output the codewords as $\hat{v} = v_{HD}$. If $s \neq 0$ and $s_{2N-k} = 0$, search even PEPs within a specified decode radius. Otherwise, search the odd PEPs. Generate corresponding test patterns according to the least reliable LLRs.

Step 3: If the received codeword v has odd number of errors, identify the last error position with applying the function $f(s^{'})$. If the codeword v has even number of errors, last two error position is required to be identified using the function $f^{'}(s^{'})$ instead.

Step 4: Obtain error pattern e by adding the error position of the last error with the test pattern determined in step 2. Perform error correction via $\hat{v} = v_{HD} + e$.

In the paper [21], the analog weight of each error pattern is required to be determined by summing up the absolute values of the LLRs of the error position. then selects the pattern with the minimum analog weight, and then performs error correction via $\hat{v} = v_{hd} + e$. However, in this work, the Pyndiah technique is used to estimate the soft output instead. As a result, all places indicating errors are decoded by $\hat{v} = v_{hd} + e$ and compiled into a list of candidate codewords. They are then passed into Pyndiah for further soft output information estimate.

3.3 DEPT-GRAND Algorithm

Guessing Random Additive Noise Decoding (GRAND) is recently introduced in [5] and provides ML decoding for any moderate redundancy block-code construction. The GRAND algorithm is implemented with two core components, a code-book membership checker and a sequential putative noise-effect sequence generator. In order to remove putative noise-effects, the demodulated sequences are rank-ordered in a decreasing likelihood order, from most likely to least likely. Assuming that c^n represents a transmitted codeword, and Z^n represents an independent additive noise effect binary sequence, then the hard decision demodulated sequence y^n satisfies $y^n = c^n + Z^n$. The sorted binary noise effect sequence is z^n, which is subtracted from the demodulated binary sequence y^n, $y^n - z^n$. If the codeword can be queried in the codebook, subtraction will yield the decoding codeword. In general, a code-book requires an average of $2n-k$ queries to find faults, and the number of queries has a direct impact on

the complexity of the decoding process. Hence, the GRAND algorithm's complexity is determined by the number of parity check bits rather than the codeword length.

ORBGRAND (Order Reliability Bits Guessing Random Additive Noise Decoding) is presented as an extension of GRAND. It offers superior soft decoding performance on a variety of codes, including BCH, CA-Polar, and RLC. It inherits the fundamental properties of high throughput and highly parallelized implementation possessed by GRANDS. ORBGRAND, like all GRAND algorithms, involves the generation of queries to identify noise effect, Z^n. The putative noise effect sequence is ranked by increasing logistic weight, often known as the sum of flipped positions. The maximal logistic weight w_L is defined as $w_L(1, 1, ..., 1) = n(n + 1)/2$, with n representing the length of the codeword. In another words, it implies the noise effect queries is capable of recognizing the sequence with the logistic weight W_L within the range of $\{0, 1, 2, ..., n(n + 1)/2\}$. The noise-effect queries are generated according to the pattern shown in Table 3.1.

Logistic Weight W_L	**Index of flipped LRBs**	**Hamming Weight** W_H
W_L=0	[0]	0
W_L=1	[1]	1
W_L=2	[2]	1
W_L=3	[3] [1 2]	1 2
W_L=4	[4] [1 3]	1 2
W_L=5	[5] [1 4] [2 3]	1 2 2
W_L=6	[6] [1 5] [2 4] [1 2 3]	1 2 2 3

Table 3.1: Putative errors queries generation algorithm.

DEPT-ORBGRAND is another algorithm that inherits the characteristics of ORBGRAND and DEPT. Similar to DEPT, DEPT-ORBGRAND classifies sequences as

having an odd or even number of errors based on the final bit of syndrome. However, DEPT requires a set of partial error patterns (PEPs) to determine the likely position of errors; after inheriting the property of ORBGRAND, the set of matrices containing the index of flipped bits under the situations of different logistic weights will serve as PEPs. Aside from that, DEPT-ORBGRAND also recommended determining the location of the last error by comparing the $s^{'}$ with h_j, recall that the term h_j is the component column of the H matrix and j satisfies $j \in \{1, 2, ..., n\}$, and the term $s^{'}$ is determined with the Equation 3.10. If the syndrome is the same as h_j, then the corresponding column will be logged as the index position of the last error. The following examples illustrate the general steps of the DEPT-ORBGRAND:

Step 1: Define the set of probable error sequences beforehand and divide it into odd and even situations depending on the hamming weight of bits that have been flipped.

Step 2: Determine the syndrome of received codewords via $s = v_{HD}H$. If $s_{n-k} = 0$, search error patterns from the odd set of PEPs; otherwise, search the even PEPs set. If $s = 0$, it implies the codeword is error-free, v_{HD} is delivered as the decoded codeword $\hat{v}$.

Step 3: Locate error positions in PEPs with $N - 1$ 1's in position, compute $s^{'}$, and apply the Equation 3.10.

Step 4: Determining the place of the last error by comparing h_j with $s^{'}$ via the function $f(s^{'})$ shown in the previous section. Return the index $j^{'}$ as the error position if $s^{'} = h_j$; else, return noting.

Step 5: Create error patterns e, and execute error correction using via $\hat{v} = v_{HD} + e$.

Step 6: Return to step 3 and repeat the preceding stages until all error sequences have been queried and processed, or until the maximum number of candidate codewords is reached.

3.4 Hard Decoding Algorithm

After finishing processing a number of iterations of soft decoding, there are few errors that remain; however, if another iteration of soft decoding is applied, the efficiency will be greatly reduced. At this point, it is advisable to implement hard decoding iterations instead so that the system requires less time and power to perform error correction. Because the old decoded LLRs are retained in the decoder's memory, hard-decoded binary codewords are typically mapped as 1,-1 prior to being saved in memory. However, in contrast to the decoding memory used for soft iteration, the information stored is limited to only the values -1 and 1. After the decoded sequence R has been obtained from the previous iteration, the sign function is utilized to determine the hard decision of the sequences, which is represented by the formula Y=(sign(R)+1)/2. The front and back bits are then concatenated to produce a sequence of 2N bits, which, after a permutation step, is ready to begin decoding. In addition to that, the Chase algorithm makes use of a hard decoder as well.

3.4.1 Berlekamp-Massey Algorithm

Conventional hard-decision decoder (HDD) techniques, such as the Berlekamp-Massey algorithm, decode codewords by determining error locator polynomials that indicate error positions. The Berlekamp-Massey algorithm takes a different approach to determining the locator polynomial, employing a linear feedback shift register (LFSR) to determine the minimal polynomial. In this article, however, the performance of the BM decoder is inadequate, and a bounded distance hard decoder is better appropriate. After gathering candidate codewords for the hard decoder used in the Chase algorithm, for instance, the system requires a hard decoder that can rectify errors within two. However, sometimes BM-based hard decoders attempt to rectify more than two errors, which can result in codewords being over-decoded and extra errors being generated in oFEC.

3.4.2 Look-up Table (LUT) Based Decoder

To optimize hard decoding, a look-up table (LUT) based hard decoder capable of considering all bits and proposing multiple strategies with different conditions based

on the syndrome is proposed. It utilized syndrome to determine the amount of errors in a manner analogous to that of the DEPT decoding algorithm. If $s \neq 0$ and the last bit of syndrome is 1, the system looks up the error pattern in the table, as there is only one error; otherwise, two errors exist. In this instance, it is necessary to consider the possibility that no error pattern can be identified. In the event that s=0, it indicates that the sequence is error-free and no additional decoding is required. If, however, no error patterns can be identified in the table when s≠0, it indicates that the candidate sequence exceeds the maximum number of errors that the hard decoder can correct, the system should disregard those sequences and pass them to the soft decoder for decoding.

Typically, conventional hard decoders do not account for such a restricted decoding distance during decoding; as a result, this can result in additional errors, which could cause further misdirected decoding during the computation of extrinsic information. Consequently, a LUT-based hard decoder that takes into account the bounded decoding distance is more applicable and significantly improves the decoding performance of oFEC.

Chapter 4

oFEC Codes

4.1 Code Properties

4.1.1 Front and Back Bits

In this particular study scenario, the extended BCH (256,239) code is regarded as the constituent codeword of oFEC. The length of the code word is denoted by the symbol 2N=256, whereas the length of the information bit is denoted by k=239, of which 2N-K=17 parity bits. The BCH codewords are split into to the front and back bits in oFEC. Suppose that the component codeword extension (256,239) BCH code is $C_i = (c_{i1}, c_{i2}, ..., c_N, ..., c_{2N})$, where c_{i1} to c_N bits are referred to as front bits and c_{N+1} to c_{2N} bits are regarded back bits. Figure 4.1 illustrates that the front and back bits are extracted from distinct locations. The back n bits are the new information bits input to the oFEC system at each time slot of each iteration block. Front bits are bits that have been encoded and decoded earlier and are typically retrieved from the memory of the encoder and decoder in accordance with a specified arrangement pattern, which will be introduced later. Hence, the front and back bits are retrieved from separate locations and concatenated into a 2N BCH code during encoding and decoding.

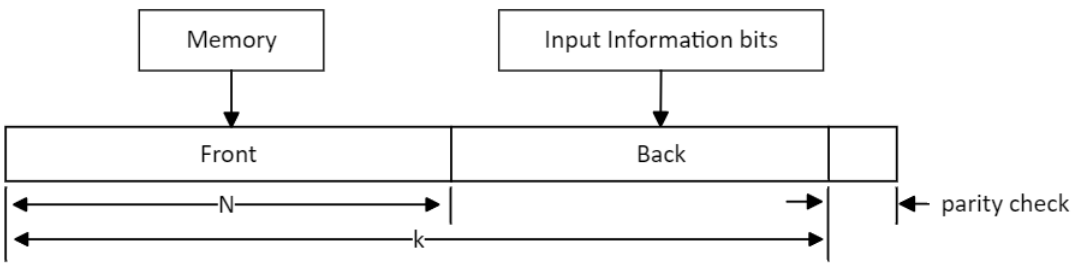


Figure 4.1: Front and back bits of BCH code.

4.1.2 Unit Block

The entire memory structure is composed of a number of unit squares, each of which has the dimensions B×B, where B is equal to 16 in this illustration. The number of block columns in the storage structure is denoted by N/B, where N is half the length of the component codeword. In this study, N=128 is utilized since the component codewords are extended (256,239) BCH codes. In such a structure, information bits are arranged in a particular fashion. For the front bits, it is separated into N/B segments, with each segment typically positioned vertically within N/B blocks. After completing bit stuffing in one block, it advances to the next diagonal block and continues to place bits vertically until all previously front bits have been placed. Suppose that at a specific time slot t, 2B rows of the codewords are input as Figure 4.2 illustrate, which is expressed as $C = [C_1; C_2; C_3; ...; C_j]$, $j \in \{1, 2B\}$. Consider a $2B \times B$ unit block as an example, as depicted in Figure 4.3a; the red-shaded region represents a segment of C. Hence, the line segment from C_1 to C_B is positioned in $\{c_i, r_{B+1} - r_{2B}\}$, i given by $i \in \{1, B\}$. The segments between C_{B+1} to C_{2B} are placed at the block position $\{c_i, r_1 - r_B\}$. Then the next section of C, that is, the blue shaded segments are placed in the next diagonal block area with a size of $2B \times B$ according to the same rules.

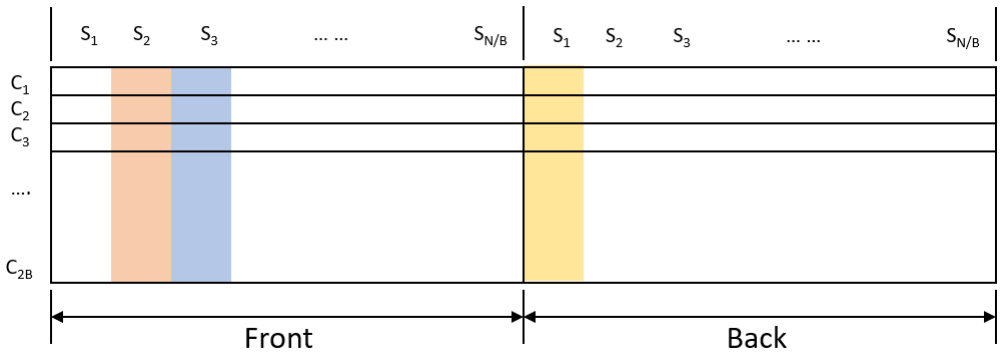
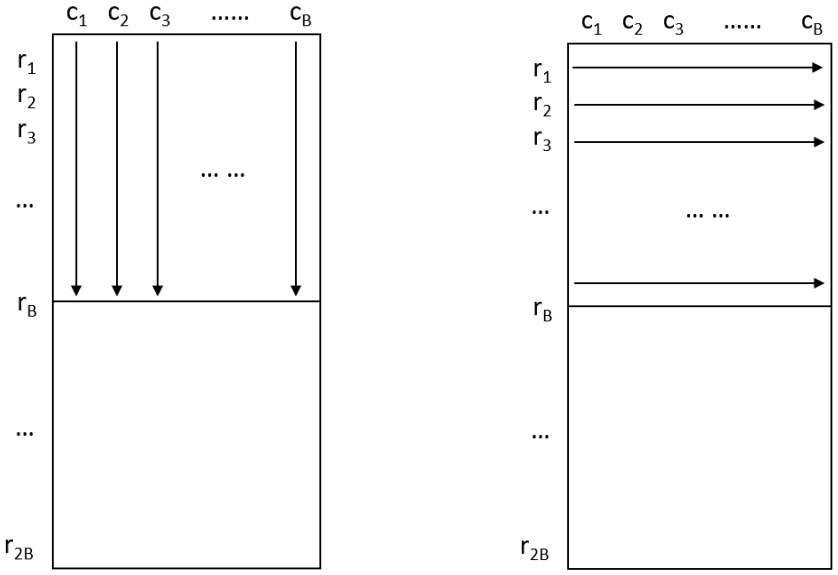


Figure 4.2: Sample constituent codewords split into the segments of $2B \times B$.

The back bit has a somewhat simpler bit arrangement than the front bit. As another illustration, consider a $2B \times B$ unit block, as depicted in Figure 4.3b. The back components are placed horizontally, unlike the front bits. Referring back to the example input codeword C, the yellow-shaded segments are positioned horizontally

in the sample block region between r_1 and r_{2B}, as depicted in the image. The next adjacent segment is positioned in the following $2B \times B$ block horizontally, and so on.



(a) Unit block of front bits. (b) Unit block of back bits.

Figure 4.3: Unit block of front and back bits.

4.1.3 Unit Block Permutation

The performance of an oFEC code is characterized by its "error events," and permutation is a vital step for reducing "error events" so that error correction accuracy can be enhanced.As codewords are read and written from memory, the oFEC structure adheres to the specified permutation depicted in Equation 4.1 and 4.2. Each bit of the matrix block can be identified in the quadruple form $\{R, C, r, c\}$ to locate any bit in the memory structure, where R and C are the block row and column numbers specified by square blocks, and r and c are the bit row and column numbers within each square block. As front and back bits occupy distinct positions, the equations described the permutation under various conditions of k being less than N and greater than or equal to N, where k is denoted by $k = \{0, 1, 2, ..., 2N - 1\}$.

$$\{R \oplus 1 - 2G - \frac{2N}{B} + 2\left\lfloor \frac{k}{B} \right\rfloor, \left\lfloor \frac{k}{B} \right\rfloor, (k \bmod B) \oplus r, r\} \quad \text{if } k < N \tag{4.1}$$

$$\{R, \left\lfloor \frac{k-N}{B} \right\rfloor, r, (k \bmod B) \oplus r\} \quad \text{if } k \geq N \tag{4.2}$$

$k < N$	$k \mod B$	$k > N$	$k \mod B$
0	0	128	0
1	1	129	1
2	2	130	2
3	3	131	3
...	...	...	...
15	15	143	15
16	0	144	0
17	1	145	1
18	2	146	2
...	...	...	...
126	14	254	14
127	15	255	15

Table 4.1: Derivation of bits permutation.

On the unit block side, assuming that the block is expressed as $\{r, c\}$, the permutation of the front bit is indicated as $\{(k \mod B) \oplus r, r\}$, and $\{r, (k \mod B) \oplus r\}$ for the back bits. Table 4.1 illustrates the derivation, which obtained $(k \mod B) \in \{0, 1, ..., 15\}$, $r \in \{0, 1, ..., 15\}$. Hence, the interface of each block after permutation is depicted in Table 4.2. It is important to note that the permutation of the square block has diagonal symmetry, meaning that the front and back bits are permuted in the exact same manner.

4.2 Structure of a Single Iteration

4.2.1 Overview of a Single Iteration Block Window

Figure 4.4 depicts an overview of a single iteration block window, whose number of columns is comprised of N/B unit blocks and whose number of rows depends on the window's design and the quantity of codewords to be processed at each time point. The graphic demonstrates that the block matrix is separated into three compartments.

Before Permutation															
Front								**Back**							
0	0		⋯			0	0	0	1		⋯			14	15
1	1		⋯			1	1	0	1		⋯			14	15
2	2		⋯			2	2	0	1		⋯			14	15
⋮	⋮		⋯			⋮	⋮	⋮	⋮		⋯			⋮	⋮
13	13		⋯			13	13	0	1		⋯			14	15
14	14		⋯			14	14	0	1		⋯			14	15
15	15		⋯			15	15	0	1		⋯			14	15
After Permtation															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	8	11	10	13	12	15	14	1	0	3	2	4	4	7	6
10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	13	14	15	9	8	10	11	4	5	6	7	1	0	2	3
13	12	15	14	8	9	11	10	5	4	7	6	0	1	3	2
14	15	12	13	10	11	8	9	6	7	4	5	2	1	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 4.2: Unit block before after bits permutation.

The front bits are located in rows 0 through 15 of the block, and the back bits are in rows 20 and 21. The area between them is the guard block area, the guard block must have an even row number, and its value is 2G. Moreover, both the system's execution time and the size of the matrix block are sensitive to the value of G. When G rises, the latency of the pipeline rises, allowing for a greater number of encoders and decoders to be parallelized, but at the expense of larger memory blocks.

4.2.2 Permutation of Structure

In addition to illustrating the organization of each unit block, Equation 4.1 and 4.2 depict the arrangement of the block's rows and columns. According to the equation,

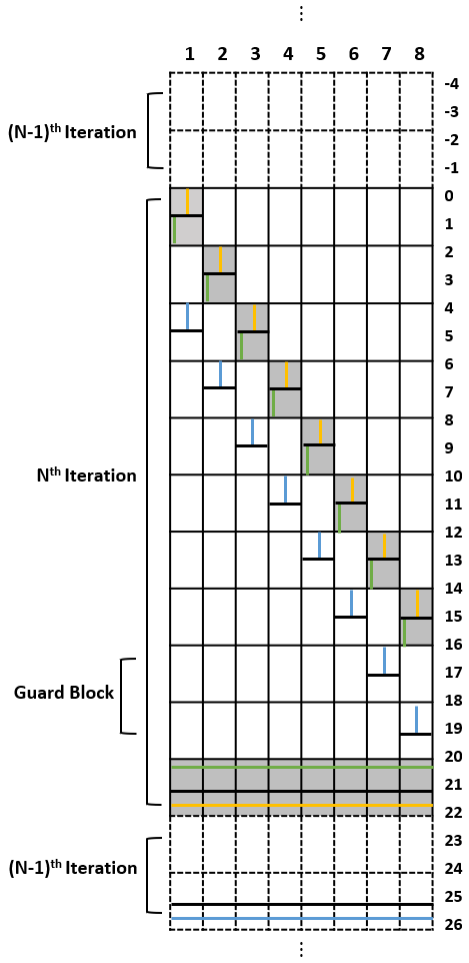


Figure 4.4: Single iteration block window.

R indicates the block row number where the input information bits are located, also known as the block row where the back bits are located. Thus, the front block behaviour $R_{front} = R \oplus 1 - 2G - 2N/B - 2\lfloor k/B \rfloor$ may be derived, where 2G is the row of the protection block, 2N/B is the total number of rows placed in the front block, and $2\lfloor k/B \rfloor$ is the number of rows to be inserted for each segment of front bits. The derivation is provided in Table 4.3, assuming k=1 and G=2.

Notably, when 32 rows of information bits are input, the positions of the first 16 lines and the back 16 lines are swapped. Back bits in row 20 will have their corresponding front bits in row 1, and back bits in row 21 will have their corresponding front bits in row 0. A graphical illustration of this is shown in Figure 4.4; this illustration is based on the assumption that the 32 rows of constituent codewords

R	$R \oplus 1$	**2G**	**k**	$2\lfloor \frac{N}{B} \rfloor$	$2\lfloor \frac{k}{B} \rfloor$	R_{front}
...	...	...	...	...	...	...
20	21	4	1	16	0	1
21	20	4	1	16	0	0
22	23	4	1	16	0	3
23	22	4	1	16	0	2
...	...	...	...	...	...	...

Table 4.3: Derivation of block permutation.

that are being entered into memory are being placed in the area of the figure that is shaded in grey. The green line is a representation of a sample codeword in the first B rows of this codeword, with the back bit located in the corresponding location of row 20 and the first bit located in row 1. In comparison, the yellow line depicts an example constituent codeword in the back B rows, with the back bit on row 21 and the front bit on row 0.

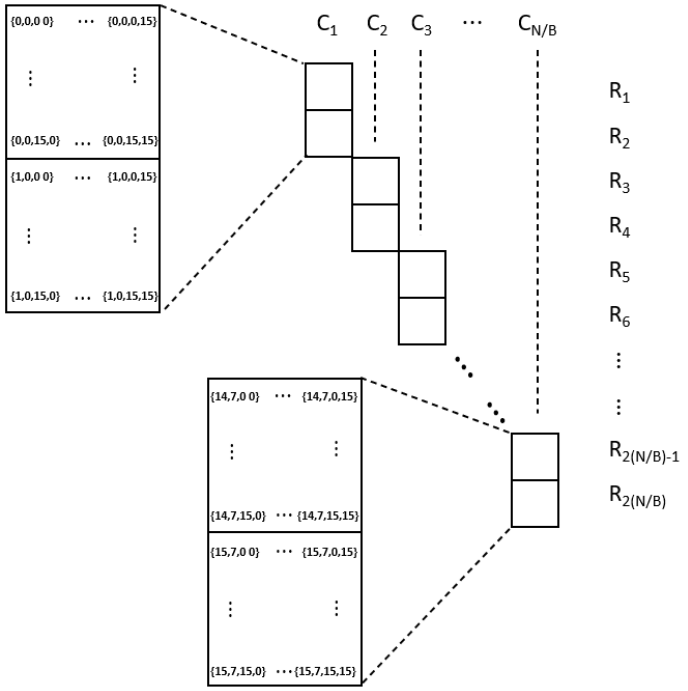
The block columns of the front and back bits are stated as k/B and (k-N)/B, according to Equation 4.1 and Equation 4.2. If the factor R = 20 is predefined, then the derivation for the block row and column of front bits can be determined as $k \in \{0, 1, 2, ..., 2N-1\}$ in Table 4.4.

$k < N$	R_{front}	$\lfloor \frac{k}{B} \rfloor$	$k > N$	R_{back}	$\lfloor \frac{k-N}{B} \rfloor$
0	1	0	128	20	0
1	1	0	129	20	0
2	1	0	130	20	0
3	1	0	131	20	0
...	...	...	...	...	...
16	3	1	144	20	1
17	3	1	145	20	1
18	3	1	146	20	1
...	...	...	...	...	...
124	15	7	252	20	7
125	15	7	253	20	7
126	15	7	254	20	7
127	15	7	255	20	7

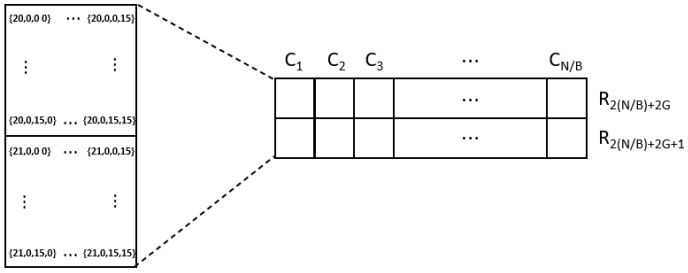
Table 4.4: Derivation of block row and column.

Recalling the constituent codeword in green shown in Figure 4.4 as an example.

As the back bits are positioned in row 20, the front bits of the component codeword are diagonally positioned on an even number of rows. Assuming that each unit block may be expressed as $\{R, C\}$, the front N bits are separated into 8 segments and inserted in column 1 of block {1,1}, {3,2}, {5,3}, {7,4}, {9,5}, {11,6}, {13,7}, {15,8}, respectively. The details illustration of permutation in quadruple form for front and back bits is depicted in Figure 4.5.



(a) Front bits permutation.



(b) Back bits permutation.

Figure 4.5: Front and back bits permutation in quadruple form.

4.3 Iterative Process

In order to implement the iterative process of oFEC, a number of unit oFEC structures introduced in the previous section will be vertically concatenated one after the other, the memory window is depicted as shown in Figure 4.6. The number of iterations depends on the number of unit structures are concatenated.

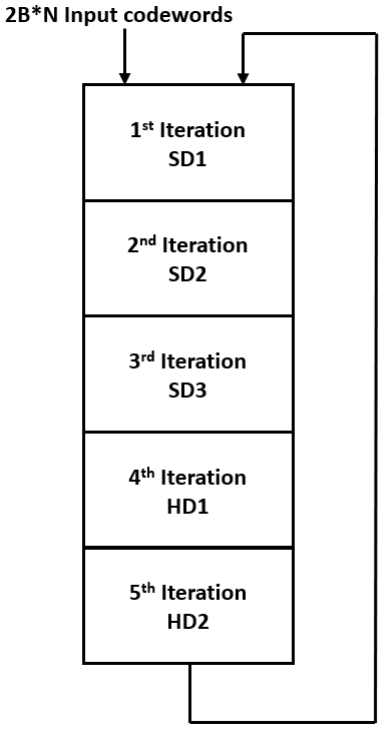


Figure 4.6: Iterative oFEC memory window.

At each time slot, a $2B \times (k - N)$ array of new information bits are fed into the oFEC system. After transmitting the encoded input constituent codewords through the AWGN channel, it's ready to be fed them into the first iteration of decoder. Notice that except for the first iteration, decode symbols output by the previous iteration become the input symbols of the subsequent iteration, which are then decoded by the subsequent iteration, and so on. After decoding process is done, decoded information bits are placed in the corresponding memory position according to the permutation specified in Equation 4.1 and 4.2. Once the encoding of decoding is complete for the time instance of t, the memory buffer will typically shift downward in the preparation for the subsequent encoding and decoding cycle to begin in the following time slot.

Due to 2B rows of new information bits are input, the number 2B is designated as the shift unit at each time instance. As the memory shift with time, the previously decode symbols are interconnected with one another, and cross-decoding with new information bits, until they complete transmit through all iterations.

During the construction of an oFEC structure, it is recommended that a few iterations of hard-decision decoders (HDD) be added after a few soft iterations of decoding in order to increase the decoding efficiency. In the event of decoding sequences with equal or less than two errors, HDD is preferable than SDD in terms of both decoding efficiency and power consumption.

4.4 Memory Buffer

An oFEC system requires a pair of identical memories, one for the encoder and one for the decoder, to store the encoded and decoded information symbols within a given time instance. Note that if the two memories are of different sizes, the stored encoded and decoded symbols will not be in the same time slot, which is likely to result in decoding errors owing to improper bit interconnections. Aside from that, it also simplifies the process of calculating the bit error rate during simulations.

The dimensions of the memory buffer is determined by the constituent codewords utilised by the oFEC system. The number of columns is set to be the same as the half-length of the component codewords, and the number of memory rows is set to be equal to the number of codewords that can be processed in parallel by the system during each time slot. As indicated, each memory iteration block is composed of three compartments. Since the front constituent codewords are split into (N/B) of 16-bits segments, the block area of front bits can be written as $2B \times (N/B)$. The block region for storing back bits is constructed of 2B rows, while the guard block comprises 2G rows. Hence, it is possible to derive the row of a single iteration of memory as follows,

$$D = 2B \times (\frac{N}{B} + 1) + 2G \tag{4.3}$$

Where it is assumed that the term I stands for the total number of iterations performed, and where G equals 2B in this particular instance. The dimension of the

memory can be expressed as,

$$D = I \times 2B \times (\frac{N}{B} + 1 + 2) \tag{4.4}$$

In other words, each input codeword requires $(\frac{N}{B} + 1 + 2) \times I$ number of time instances to traverse all iteration blocks, which 11 time slots are required for each iteration in this case. Assuming that $t \in \{1, 2, 3, ...\}$ is given as the time slot grows indefinitely, the index of the back bits is denoted as,

$$I_B(t) \equiv (I \times (\frac{N}{B} + 1 + 2)) \mod t \tag{4.5}$$

such that the index of the front bits can be expressed as,

$$I_F(t, i) = I_B - (\frac{N}{B} + 1 + 2) + i \tag{4.6}$$

where $i = 0, 1, 2 \ldots, \frac{N}{B} - 1$ as front bits are divided into N/B segments. In addition, according to the description of the Pyndiah algorithm, the LLRs output by the channel are used to determine extrinsic information; therefore, another memory buffer is required to synchronously store the LLRs output by the AWGN channel within a particular time slot in order to assist with decoding.

4.5 Code Rate

Although the component codeword of oFEC is the extended BCH(239,256), only (k-N) fresh information bits are given to the oFEC system during each time slot, and (2N-k) parity check bits are generated after encoding. Consequently, since the code rate of the extended BCH(256,239) is expressed as k/2N, the code rate of oFEC can be determined as follows:

$$R_{BCH} = \frac{2(k - N) + (2N - k)}{2((k - N) + (2N - k))} \tag{4.7}$$

Which assumes $v = k - N$, $p = 2N - k$. Hence it can be simplified as,

$$R_{BCH} = \frac{2v + p}{2(v + p)} \tag{4.8}$$

$$2R_{BCH} = 1 + R_{oFEC} \tag{4.9}$$

In the case of the study, the coding rate of the (256,239) extended BCH code is estimated as $R_{BCH} = 239/256$; afterward, the code rate of the oFEC code can be derived as $R_{oFEC} = 111/128$ after applying the Equation 4.9. This indicates that at each time slot, 111 bits are supplied into the system and 128 bits are removed from the system simultaneously.

Chapter 5

oFEC with DEPT-based Decoders

5.1 oFEC Iterative Coding

The implementation of the oFEC system employs three soft-decision (SD) iterative decoders, as recommended by the document [16]. Additionally, instead of utilizing another soft-decision (SD) iterative decoder to correct the remaining few errors, it is more effective to add two HD iterative decoders after the first set of three SD decoders as suggested in [22]. In order to meet expectations, 256 candidate codewords must be generated from the eight least reliable bits (LRBs) when the Chase algorithm is applied to the three-soft and two-hard (3S2H) iterative decoder structure of oFEC. The overall operational path of the oFEC system is depicted by the block diagram in Figure 5.1.

Let's firstly define the vector R_T as the position of the block row that the input information bits are placed at each time slot of t, $t \in \mathbb{Z}$, which is denoted as the Equation 5.1. it clearly implies that the memory is a circular shifting window with a unit of 55, due to the system employing five iterations. For instance, if the back bits were located in row 55 of the preceding slot, the back bits for the succeeding time slot will be positioned in block row 1, block row 2, so and so on. Notice that in the case of position new informations in row 55 R_T is determined as 0 after following the Equation 5.1. At this time, it means the system finish a round of trip and tend to circular shift to the top of the memory and process next round.

$$R_T \equiv t \mod 55 \tag{5.1}$$

Two function of $\pi(.)$ and $\phi(.)$ are employed to define the permutation of memory structure. Assume $X(r, c)$ represents a bit in a $B \times B$ single unit square block, and $Y(r, c)$ is denoted as the corresponding bit after permutation. Thus, the function $Y(r, c) = \pi(X(r, c))$, which is denoted as the permutation within each unit square block, can be defined as,

$$Y(r,c) = \begin{cases} X((k \mod B) \oplus r, c), & \text{if } k < N \\ X(r, (k \mod B) \oplus c), & \text{if } k \geq N \end{cases}$$

where the vector r and c are defined as $r, c \in [0, 1, 2, ..., B-1]$. As it states $Y = \pi(X)$, it satisfies the property of $\pi(Y) = \pi^{-1}(\pi(X))$.It means that if permutation is applied twice to a unit square, the bit permutation will return to the position before the permutation. The functioon $\phi(.)$ defined used to retreieved old encoded and decoded information from memory. Suppose each $B \times B$ block in the memory structure is defined as $X(R, C)$, and retreived information bits referred as $Y(R, C)$. Such that the function $Y(R, C) = \phi(X(R, C))$ is defined as,

$$Y(R,C) = \begin{cases} X(R_B, C), & \text{if } k \geq N \\ X(R_B \oplus 1 - 10 + C, C), & \text{if } k < N \end{cases}$$

where C is defined by,

$$C = \begin{cases} \lfloor \frac{k}{B} \rfloor, & \text{if } k < N \\ \lfloor \frac{k-N}{B} \rfloor, & \text{if } k \geq N \end{cases}$$

where $C \in 0, 1, 2, ..., \frac{N}{B} - 1$. Note that the vector R_B in the $\phi(.)$ function changes depending on which iteration the decoding takes place.According to the oFEC design in this paper, the new information codeword is fed from the top of the memory. When the information codeword is decoded for the first iteration, $R_B = R_T$, otherwise R_B needs to be derived according to the number of iterations of decoding. Assuming the vector I denoted as the number of iteration that the constituent codewords is experiencing, the vector R_B can be expressed as,

$$R_B \equiv (R_T + Z \times 11) \mod 55 \tag{5.2}$$

$$Z \equiv ((5 - I \mod 5) + 1) \mod 5 \tag{5.3}$$

Thus, retrieved front bits $F(t)$ can be inferred as $F(t) = \phi([X_1(t), X_2(t), ..., X_8(t)])$, after further derivation $F(t)$ is expressed as $F(t) = [X_F(R_B(t) \oplus 1 - 10), X_F(R_B(t) \oplus 1 - 9), ..., X_F(R_B(t) \oplus 1 - 3)]$.Details implementation steps of oFEC are described as follows:

Step 1: Initializing memories, encoder memory is filling with zeros, decoder memories are filling with -1s since LLRs are stored into the memory.

Step 2: At a time slot of t, a $2B \times (k - N)$ bit array of information is transmitted into the oFEC system, denoted as back constituent codewords C(t).

Step 3: Retrieved front constituent codewords $F_e(t)$ from block $X_F(R_B - 11 + i)$ of the encoder memory. Then implement permutation on front bits and concatenated with C(t).

$$C_{in}(t) = [\pi(F_e(t))\, C(t)] \tag{5.4}$$

Step 4: Encode the permutated codewords $C_{in}(t)$ with the generator matrix G, where I_k is the unity matrix of dimension $K \times K$, and P is the parity check bit generating matrix with dimension $(2N-k) \times k$. The function $\phi(.)$ is then applied to encoder memory to store the encoded back bits.

$$C_{out}(t) = C_{in}(t)G \tag{5.5}$$

Step 5: Implement permutation on the back of the encoded information bits $C_{out}(t)$.

$$C^{'}_{out}(t) = [F^{'}_{e}(t)\, \pi(C^{'}_{out}(t))] \tag{5.6}$$

Step 6: Modulate $C^{'}_{out} \in \{0, 1\}$ into $\{-1, 1\}$ and transmit it through AWGN channel. Demodulated the channel output and feed it into the decoder. At the same time fill the demodulated channel output into the channel information memory.

Step 7: Retrieve front constituent codewords $F_d(t)$ with the same block pattern as the step (3) states, then apply $\pi(.)$ function on the concatenated codewords D(t).

$$D^{'}(t) = \pi(D(t)) = \pi([F_b(t)R(t)]) \tag{5.7}$$

Step 8: Applying $\phi(.)$ function to retrieve bits from channel information memory, prepare for the later deocoding steps.

Step 9: Decoding the concatenated codewords with applying SISO decoding algorithm, e.g. Chase-Pyndiah decoder, DEPT decoder. Apply inverse permutation on decoded bits, $D^{'}_{out}(t)$ denoted as decoder output.

$$D^{'}_{out}(t) = \phi(D^{'}(t)) \tag{5.8}$$

Step 10: Restore the decoded and permuted codeword $D'_{out}(t)$ to decoder memory using the $phi(.)$ function.

Step 11: Circularly shift all memories downward with a unit of 2B. Enter into next time instances t_{i+1}. Go back to step (2), and start the next round of encoding and decoding, so and so on.

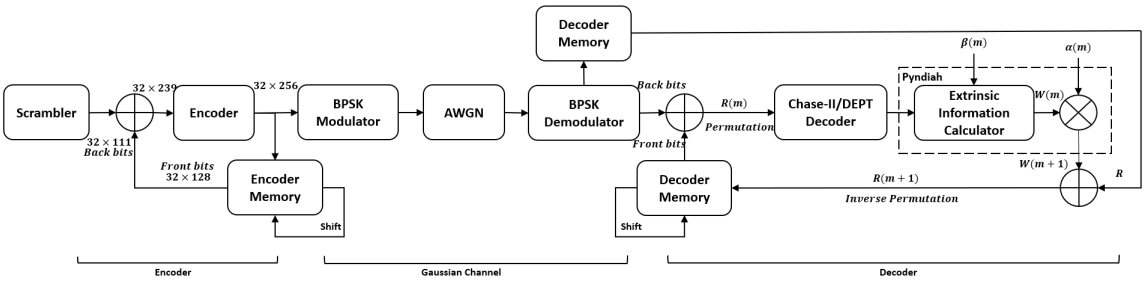


Figure 5.1: oFEC system implementation block diagram.

5.1.1 Net Coding Gain

The oFEC code described in the document [16] may achieve 10^{-15} bit error rate (BER) with a Net Coding Gain (NCG) of 11.1dB when utilizing BPSK or QPSK modulation and demodulation and 11.6dB when applying 16QAM after three soft decision (SD) iterations. The Equation 5.9 clearly depicts the calculation of Net Coding Gain (NCG), where SNR_{in} and SNR_{out} are input and output SNR when the bit error rate (BER) reaches 10^{-15}, and $10log_{10}(R_{oFEC})$ is a normalization term for comparing the performance of oFEC code with others.

In the instance where BPSK is utilized, the uncoded BER approaches 10^{-15} at approximately 14.99dB after the extended (256,239) BCH codeword has passed over the AWGN channel. Given that the R_{oFEC} is 111/128 and the NCG is 11.1dB, the output SNR may be calculated to be 3.27dB. However, when 16QAM is used, the uncoded BER approaches 10^{-15} at approximately 18.93dB; thus, the output SNR of 16QAM is calculated to be 12.73dB.

$$NCG = SNR_{in} - SNR_{out} + 10\log_{10}(R_{oFEC}) \tag{5.9}$$

5.1.2 SNR Normalization

The signal-to-noise ratio (SNR) is normalized as energy per information bit to spectral density ratio (Eb/N0), also known as "SNR per bit," in order to facilitate more accurate comparisons of bit error rate (BER) performance. If E_s represents the energy per symbol and E_b represents the energy per information bit, then the relationship between E_s and E_b can be represented as Equation 5.11 considering the code rate of oFEC.

$$\frac{E_b}{N_0} = \frac{A^2}{N_0} = \frac{d_{min}^2}{4N_0} \tag{5.10}$$

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} \times log_2(M) \times R_{oFEC} \tag{5.11}$$

The number of different modulation symbols, represented by the letter M, varies according on the type of modulation being used. In this article, both conditions of employing BPSK and 16QAM are considered for the case study, with M=2 for BPSK and M=16 for 16QAM. According to the definition of the AWGN, the noise vector is independent and normally distributed from a zero-mean normal distribution with variance N, expressed as $n \sim N(0, \sigma^2)$. The term N_0 represents noise variance σ^2. The purpose of normalization is to facilitate error correction performance comparison. Comparing the energy per symbol is unfair because different modulations result in a varied amount of bits per symbol. Furthermore, the normalization is closely related to the code rate as well. As a result, the most cases, E_s/N_0 needs to be converted to E_b/N_0.

For instance, if BPSK is utilized, each symbol consists of one bit. The relationship between E_b/N_0 and E_s/N_0 is expressed as Equation 5.12 illustrates when $N_0 = \sigma^2$. To convert them into the unit of decibels, apply logarithms on both sides of the expression as shown in Equation 5.13, since the signal-to-noise ratio in dB is given as $SNR_{dB} = 10log_{10}(E_b/N_0)$. Eventually, the relationship is expressed as Equation 5.14.

$$\frac{E_b}{N_0} = \frac{Es}{N_0 R_{oFEC}} \tag{5.12}$$

$$10\log_{10}\frac{Eb}{\sigma^2} = 10\log_{10}\frac{Es}{\sigma^2} - 10\log_{10} R_{oFEC} \tag{5.13}$$

$$\frac{E_b}{N_0}[dB] = \frac{E_s}{N_0}[dB] - 10log_{10}(R_{oFEC}) \tag{5.14}$$

5.1.3 Simulation with Different Number of LRBs

As noted previously, when utilizing the chase algorithm, the system uses eight LRB to generate 256 candidate codewords for decoding and collaborates with the 3S2H oFEC structure to obtain a bit error rate (BER) of 10^{-15}. In addition to p=8, the Chase algorithm can use alternative numbers of LRBs, such as p=5,6,7, to reach a similar level of performance as p=8. As a result of reducing the number of LRBs to be selected, the collection of candidate codewords that need to be processed and assessed has also been reduced. Hence, less time is required due to decreased complexity. Yet, the accuracy of error correction has decreased. In order to obtain the desired BER at the specified net coding gain, it is necessary to increase the number of iterations as the number of candidates p decreases. As the number of iterations rises, the size of the memory window may change, but the procedures are identical when p=8. For instance, when p=6, four SD iterations may require to combined with two HD iterations. Also, the values of alpha and beta must be modified according to the situation. For instance, the oFEC structure is created as 4S2H at p=6, where the alpha and beta factors cannot grow as steeply as they do at p=8. As p lowers, fewer candidate codewords are generated, and the error-correction capabilities of each individual iteration weakens. Hence, alpha and beta values must be gradually increased to prevent over-decoding.

5.1.4 Weighting and Reliability Factor Selection

The parameter α and β play a crucial role on decoding performance while Pyndiah algorithm is utilized. Since the standard deviations of the received codeword R and extrinsic information W are different, particularly in the first iteration of decoding, the scaling factor α is used to reduce the effect of W given the relatively high SNR.

As the number of iterations increases, the factor α and β gradually increase to one. As the [12] states, α and β are vary for different cases, such as depends on the type of modulation and component codeword used. Theoretically, there's no definition equation can be used to calculate exact value of α and β in all of cases, the optimal alpha and beta values are chosen by simulation, which is a tedious procedure. To avoid such a tedious procedure of seeking suitable value, some papers conclude and deduced some equations of calculating α and β within some limited situation. The [12] proposed and derive the relationship equation between the extrinsic information W with the factors in the case of product codes and block turbo codes.

5.2 Table-based Hard Decoder for oFEC

Table-based hard decoder is an advancement of hard decision decoding technology in compared with convenience HDD technologies such as the Berlekamp-Massey algorithm. Figure 5.2 illustrates the BER curves of oFEC implemented using the Berlekamp-Massey and table-based HDD techniques, respectively. By implementing the table-based hard decoder, the efficiency of error correction has been significantly enhanced. While the length of the code word must fulfill $2^M - 1$ when the Berlekamp-Massey algorithm is applied, the last bit of the code word is ignored during decoding. Similar to the DEPT, table-based hard decoders identify cases with either one or two errors, depending on the last bit of the syndrome. Table 5.1 demonstrates that the table-based hard decoder improves oFEC performance by approximately 0.22dB.

Decoding Algorithm		**Number of Error Patterns**	**pre-FEC BER Threshold**
Chase BM	p=8	256	$1.81e^{-2}$
Chase Table-based	p=8	256	$2.06e^{-2}$

Table 5.1: Comparison of pre-FEC BER threshold of applying BM hard decoder and table-based hard decoder.

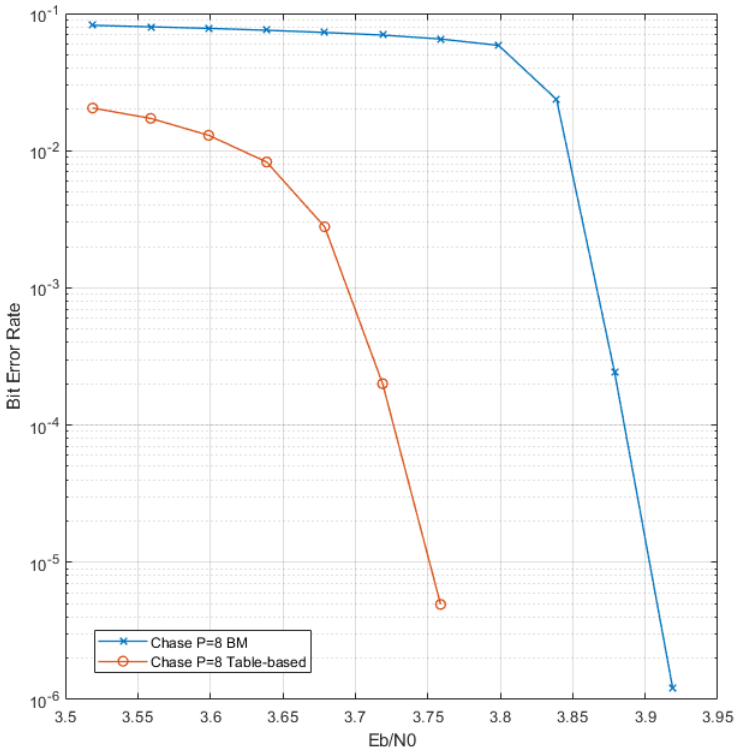


Figure 5.2: BM Chase verses table-based Chase.

5.3 DEPT Algorithm for oFEC

5.3.1 Partial Error Patterns (PEPs)

Partial error patterns (PEPs) is a predetermined set containing all potential error patterns. In order to collect as many error patterns as possible prior to decoding, it is important to simulate all the procedures, including encoding, modulation, transmission through an AWGN channel with a given signa-to-noise ratio, and demodulation, throughout a number of frames. The SNR utilized to generate the PEP is dependent on the desired SNR level during oFEC system simulation; for example, if, while utilizing BPSK, oFEC is expected to achieve a pre-FEC BER threshold of $2e^{-2}$ at 3.27dB, then 3.27dB is used as the reference for creating the appropriate PEP.

Assuming that information contains random 239 bits, u, is input and encoded as C, then after it passes through the AWGN channel, the hard decision of the codewords is obtained as v_{HD}. Hence, errors can be identified as $e = C + v_{HD}$. If the vector e is nonzero, the codeword contains an error; otherwise, it is error-free. Then, record the error positions, excluding the last member, if the number of errors is greater than zero but less than the maximum number of errors supplied. To rectify all conceivable error patterns, it is important to simulate as many frames as feasible.

After all of the frames have been worked on, it is necessary to make an estimation of the error probability using the total number of bits and the total number of error patterns. The number of error patterns will be increment by one when the likelihood of an error is larger than zero. Eventually, the number of error patterns corresponding to the i number of error weight can be determined, where $i = 1, 2, ..., m$ is denoted as the weight of errors, and m is denoted as the maximum weight of errors. Meanwhile, another vital information matrix containing the index positions of all conceivable errors under each error weight condition is constructed. However, the volume of error patterns with error probability greater than 0 is enormous. Thus, error patterns with less reliability are typically eliminated. Based on error probability, a list of PEPs with error location indices is produced.

5.3.2 oFEC Simulation Result with DEPT

Figure 5.3 depicts the performance of oFEC utilizing DEPT-Pyndiah decoding algorithm when BPSK signalling is used. In the illustration, oFEC employs the Chase-Pyndiah decoding algorithm with the LRBs p=6,7,8. The performance of error correction steadily improves along with an increase in the number of LRBs that are used. The reason for this is that as long as there are more LRBs, the error correction range, given by the variable p, will increase. Simultaneously, the decoder will create more candidate codewords. Hence, the decoding performance of the oFEC system is enhanced as well. Nevertheless, the ever-increasing number of candidate codewords that need to be processed causes an exponential increase in both the cost of the operation and the complexity of the system as well as the amount of time it takes to analyze the information.

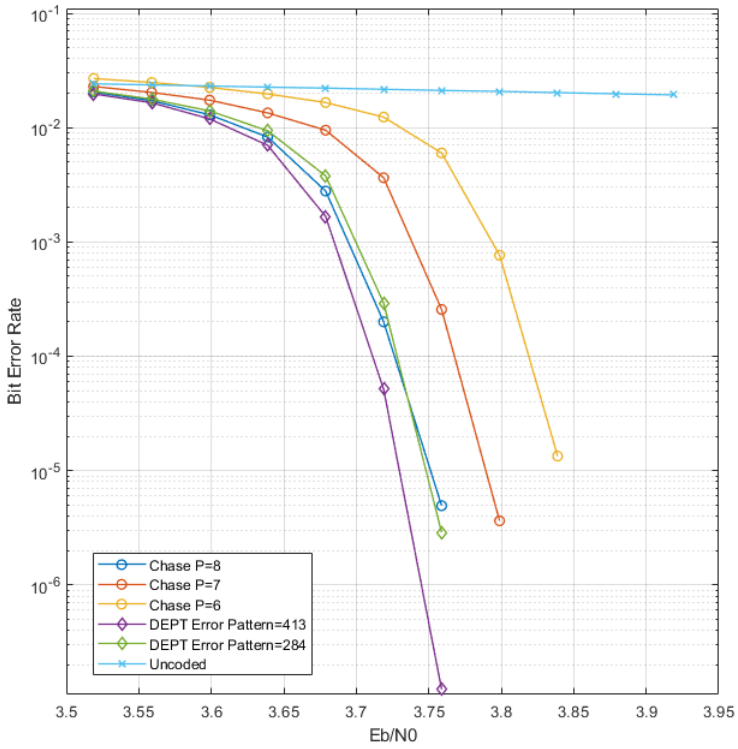


Figure 5.3: BPSK oFEC with Chase-Pyndiah p=6, 7, 8 verses DEPT-Pyndiah.

The figure depicts two DEPT outcomes, which employed 413 and 284 number of

error patterns respectively during decoding process. In fact, both of them are generated from the same PEPs, but they employ different strategies to eliminate those unnecessary low-probability error patterns. Specifically, the PEP is compiled from all potential error positions on a certain Eb/N0. The NCG of the system determines, in general, the Eb/N0 ratio used to generate PEPs. Generally speaking, it is impractical to employ all error patterns during decoding. Thus, error patterns with a lower probability are filtered out, resulting in two BER curves shown in the figure decoding with a different number of error patterns. There are 256 error sequences produced when the Chase p=8 algorithm is used in conjunction with oFEC. Nonetheless, the figure demonstrates that the performance of DEPT with 284 error patterns is comparable to Chase p=8. DEPT has a steeper curve than Chase p=8 with a similar number of error sequences. And the DEPT curve with 413 error sequences is vastly superior to the Chase p=8 curve. However, the number of frames used during PEP synthesis may have an effect on the outcomes, as a larger number of frames allows the PEP to cover the maximum number of conceivable error patterns. Hence increasing the precision of error correction.

Figure 5.4 depicts the performance of oFEC employs 16QAM utilizing DEPT-Pyndiah decoding algorithm. Similarly, oFEC employs the decoding algorithms of Chase p=6,7,8 in compare with DEPT algorithm. As predicted, the error correction accuracy improves as p grows when the Chase algorithm is utilized. In the 16QAM scenario, DEPT performed noticeably better than Chase p=8, and around 184 error patterns were utilized for both the odd and even cases respectively. Clearly, DEPT enhances the accuracy of error correction significantly by utilizing fewer error patterns. As DEPT does not list all possible candidate codewords within a specified decoding radius, it splits the error amount into even and odd situations based on the final component of the syndrome mentioned in the preceding section. As a result, the range of error correction is narrowed, and not only is the precision of error correction enhanced, but also the time required for error correction is decreased.

The statistical performance comparison between decoding algorithms is displayed in Table 5.2 and Table 5.3, which is respectively under the BPSK and 16QAM signaling. For BPSK, DEPT with 284 error patterns has almost identical performance with the Chase p=8, in which their pre-FEC BER threshold is $2.06e^{-2}$ and $2.07e^{-2}$. But

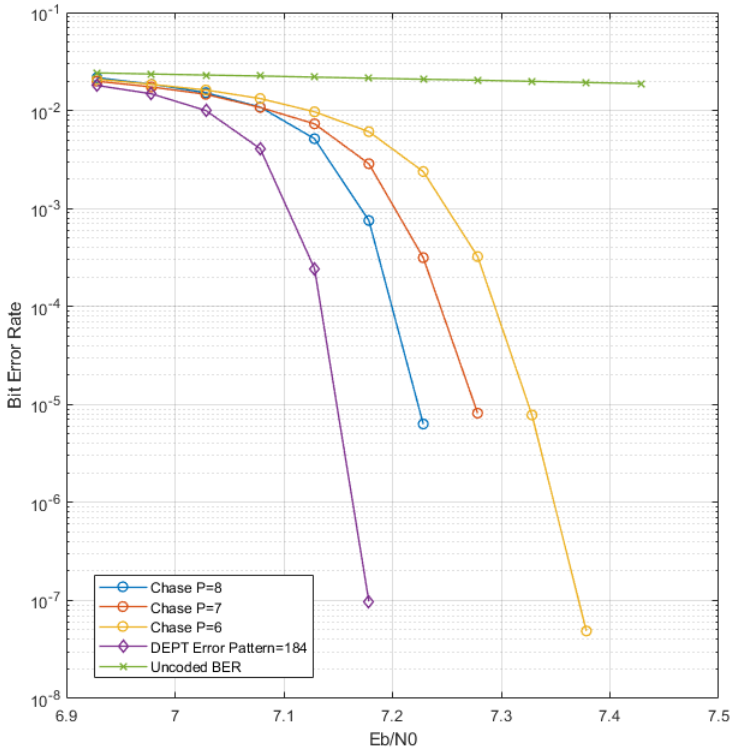


Figure 5.4: 16QAM oFEC Chase-Pyndiah p=6, 7, 8 verses DEPT-Pyndiah.

the DEPT using 413 error patterns has an obvious improvement which is capable of reaching $2.09e^{-2}$. In the case of 16QAM, the performance of the DEPT greatly surpasses the Chase p=8, which its pre-FEC BER threshold of reaching 10^{-15} is $2.10e^{-2}$, while the pre-FEC BER threshold of the Chase p=8 is $2.02e^{-2}$.

Decoding Algorithm		**Number of Error Patterns**	**pre-FEC BER Threshold**
Chase BM	p=8	256	$1.81e^{-2}$
Chase Table-based	p=7	128	$2.0e^{-2}$
	p=8	256	$2.06e^{-2}$
DEPT	decode radius=10	284	$2.07e^{-2}$
	decode radius=10	413	$2.09e^{-2}$

Table 5.2: Comparison of the number of error patterns and the pre-FEC BER threshold of Chase and DEPT on BPSK oFEC.

Decoding Algorithm		Number of Error Patterns	pre-FEC BER Threshold
Chase Table-based	p=7	128	$2.0e^{-2}$
	p=8	256	$2.02e^{-2}$
DEPT	decode radius=10	184	$2.10e^{-2}$

Table 5.3: Comparison of the number of error patterns and the pre-FEC BER threshold of Chase and DEPT on 16QAM oFEC.

5.3.3 DEPT With Different PEPs

As the time slot expands, the front and back bits may be in different blocks due to the unique construction of oFEC. Hence, over time, the front and back bits of the codewords may be decoded in different iterations, and the uncoded BER may vary. Table 5.4 displays the bit error rate (BER) for the front and back bits for each iteration. The table reveals that the uncoded BER of the front bit is always greater than that of the back bit, which explains why the β and α values employed by the Pyndiah algorithm are different for the front and back bits. When the input SNR is 3.27dB, the uncoded BER after the first iteration is $1.967e^{-2}$ corresponding to the SNR 3.69, and the uncoded BER before the bit is $1.417e^{-2}$ corresponding to the SNR 3.809, the front bit always has a lower number of errors, which implies it has higher reliabilty and requires higher values of α and β throughout the decoding process. When utilizing the DEPT technique to construct the PEP, it is therefore important to account for the varying Eb/N0 of the front and back bits. The input signal-to-noise ratios of the front and back pre-FEC are 3.81dB and 3.27dB before normalization, respectively, based on the results of numerous tests.

Despite the fact that the uncoded BER values for the front and back bits are different in each iteration, it is not impractical to use its specific PEPs for each iteration. Except for the initial iteration, the bit error rate is unstable in subsequent iterations. Due to the fact that the input codeword is fully random, it is difficult to construct an exact PEP suitable for all circumstances based on the bit error rate because the input codeword is completely random. Hence, the generation of PEP relies solely on the BER of distinct front and back bits for the first iteration.

Figure 5.5 depicts the comparison curves before and after examining the front and

Number of Iteration	BER		SNR	
	Front	Back	Front	Back
1^{st} Iteration	$1.417e^{-2}$	$1.967e^{-2}$	3.809	3.269
2^{nd} Iteration	$2.127e^{-3}$	$1.131e^{-2}$	6.113	4.147
3^{rd} Iteration	$1.277e^{-5}$	$6.198e^{-4}$	9.475	7.173

Table 5.4: Estimation of uncoded BER of front and back bits.

back bits individually. When the front and back bits are considered independently, the BER curve is somewhat slightly worse at lower SNR. As the SNR increases, the BER curve begins to steepen, and when the SNR reaches approximately 3.73dB, it outperforms both Chase and the original DEPT (do not considering the front and back bits separately). Thus, considering a variety of circumstances during the creation of PEPs can result in modest gains at higher SNR stages.

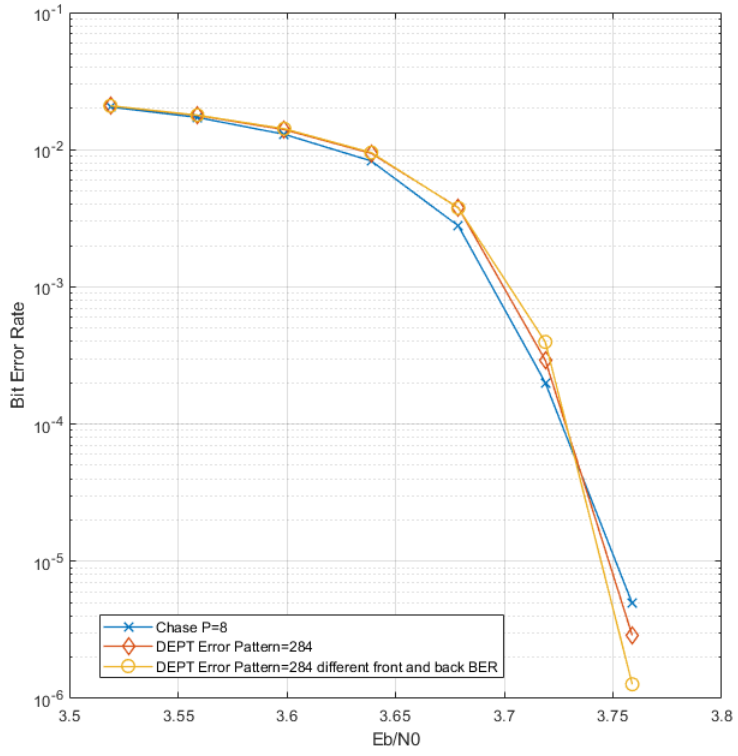


Figure 5.5: On BPSK oFEC, compare the original PEP and generate PEPs with considered front and back bits respectively.

5.3.4 Quantization

The oFEC system employs two uniform quantizers with evenly spaced quantization stages. After symbols pass through the AWGN channel, one of the quantizers is implemented. Assuming that the number of quantization bits is n, the output of the channel y is quantized to y_q, which has 2^n intervals within the given range [L, -L]. The symbol generated by the channel after 16QAM modulation is a complex number, such as $y = (y^I, y^Q)$, and both the real and imaginary components must be quantized uniformly. The second quantizer follows the demodulator, and the demodulated symbols are converted to rational values based on the log-likelihood ratio. In a similar fashion, 2^n intervals are differentiated evenly within the quantization interval [L, -L], and the quantization range is determined by the maximum and minimum values of LLR. The quantized LLR is multiplied by the scale factor α to enable further decoding.

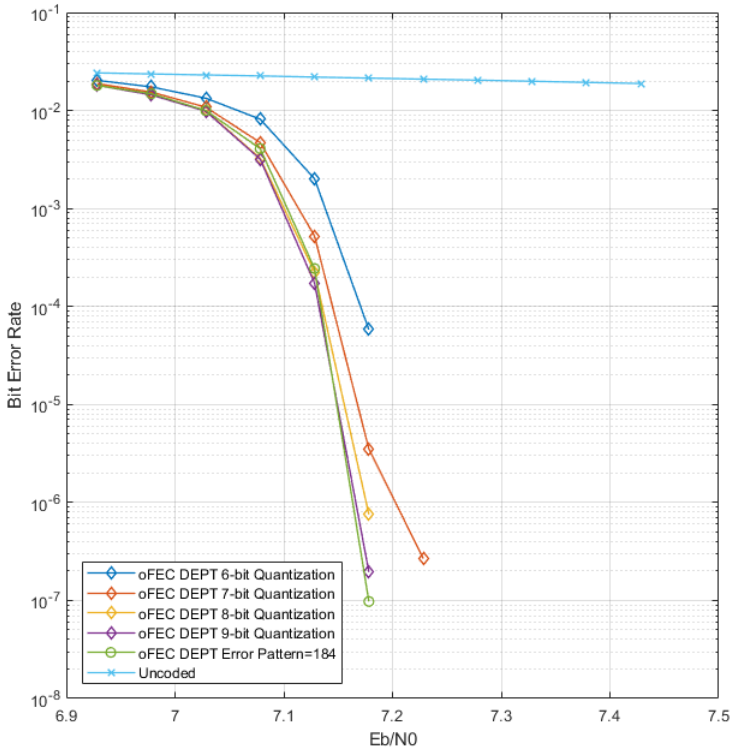


Figure 5.6: 16QAM oFEC N-bit quantization comparison.

Figure 5.6 depicts a comparison of the curves resulting from the application of various quantization levels. When the number of quantization bits is increased to 9

bits, performance is nearly identical to that of oFEC without quantization.

5.4 DEPT-ORBGRAND Algorithm for oFEC

5.4.1 oFEC Simulation Result with DEPT-ORBGRAND

During a simulation with DEPTGRAND, the algorithm generates a set of queries whose logic is based on the Table 3.1. It uses a number of queries $Q = 2^{12}$ to search for C quantities of potentially most likely error sequences. Similar to DEPT, DEPT-ORBGRAND's query set includes all allowable potential error patterns, but with logistic error weights. However, only partial queries are utilized during implementation. Likewise, the DEPT-ORBGRAND decoding algorithm is applied to oFEC with BPSK and 16QAM, and the simulation results are displayed in Figures 5.7 and 5.8. DEPT-ORBGRAND with the parameter of C=8 and $Q = 2^{12}$ considerably outperforms Chase p=8 according to both graphs.

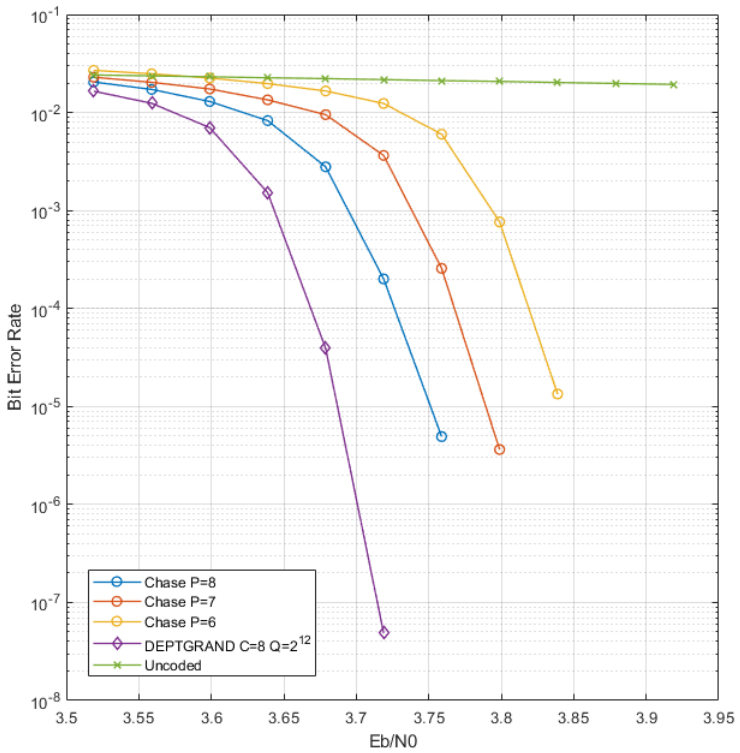


Figure 5.7: BPSK oFEC Chase-Pyndiah p=6, 7, 8 verses DEPTGRAND-Pyndiah.

Table 5.5 makes it more apparent that the pre-FEC BER threshold of DEPT-ORBGRAND for BPSK is approximately $2.12e^{-2}$, with parameters C=8 and $Q = 2^{12}$. In the case of 16QAM, when the number of query Q is unchanged, the maximum number of candidate codewords examined rises to 10 and the pre-FEC BER threshold reaches around $2.12e^{-2}$ according to Table 5.6. Nonetheless, the pre-FEC BER threshold for Chase p=8 is $2.06e^{-2}$. Further than that, after comparing the performance of DEPT in the previous section, it can be found that DEPT-ORBGRAND with using such value of parameter of C and Q can outperforms than DEPT we well. Since DEPT-ORBGRAND select C most likely candidate codewords, which significantly enhances the accuracy of LLRs correction in the subsequent implementation of the Pyndiah algorithm.

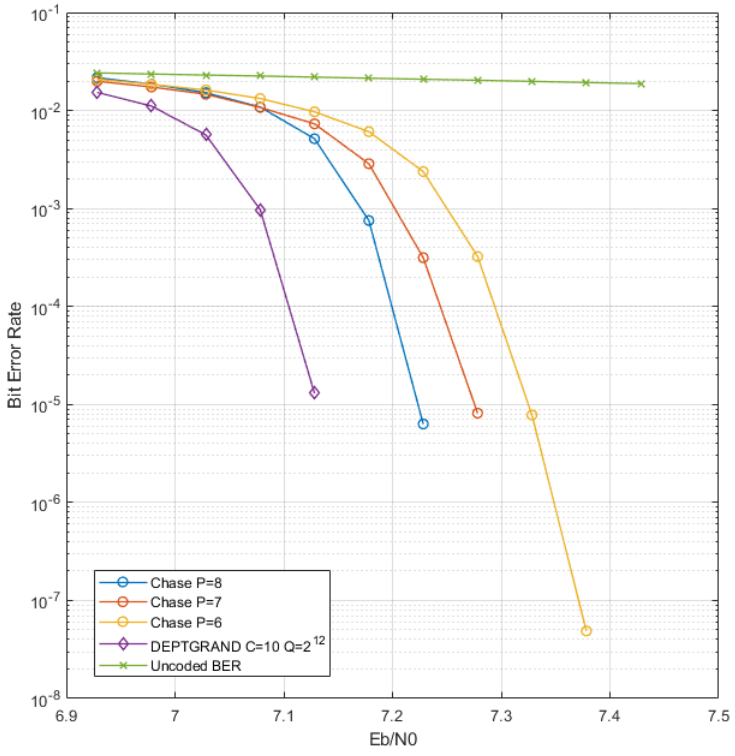


Figure 5.8: 16QAM oFEC Chase-Pyndiah p=6, 7, 8 verses DEPTGRAND-Pyndiah.

Notably, the decoder must scan the all queries until the specified maximum number of error patterns, C, is reached. This implies that the size of the queries directly influences the complexity of the system. Consequently, despite the fact that the

DEPT-ORBGRAND algorithm applied to oFEC provides improved decoding performance, its decoding complexity is relatively higher than the DEPT and Chase.

Decoding Algorithm		**Number of Error Patterns**	**pre-FEC BER Threshold**
Chase table-based	p=7	128	$2.0e^{-2}$
	p=8	256	$2.06e^{-2}$
DEPT-ORBGRAND	C=8, Q=2^{12}	8	$2.12e^{-2}$

Table 5.5: Comparison of the number of error patterns and the pre-FEC BER threshold of Chase and DEPT-ORBGRAND on BPSK oFEC.

Decoding Algorithm		**Number of Error Patterns**	**pre-FEC BER Threshold**
Chase table-based	p=7	128	$2.0e^{-2}$
	p=8	256	$2.02e^{-2}$
DEPT-ORBGRAND	C=10, Q=2^{12}	10	$2.12e^{-2}$

Table 5.6: Comparison of the number of error patterns and the pre-FEC BER threshold of Chase and DEPT-ORBGRAND on 16QAM oFEC.

5.4.2 DEPT-ORBGRAND Varies the Number of Query and Maximum Searched Error Sequences

As stated in the previous section, the complexity of the system is exponentially proportional to the number of error patterns contain in a query set. Thus, the DEPT-ORBGRAND implementation with the parameters C=8 and $Q = 2^{12}$ may not the most efficient in terms of power efficiency and latency during the comparison with the Chase and DEPT.

As illustrated in Figure 5.9, DEPTGRAND simulates using three different numbers of queries, $Q = 2^{10}$, $Q = 2^{11}$ and $Q = 2^{12}$, with the maximum search candidate codeword C=8. After comparison, it is obvious that the parameter Q needs to reach at least 2^{11}, indicating that $Q = 2^{11}$ is the threshold for DEPT-ORBGRAND to surpass the performance of the Chase p=8 with around eight maximum number of candidate codewords required to search.

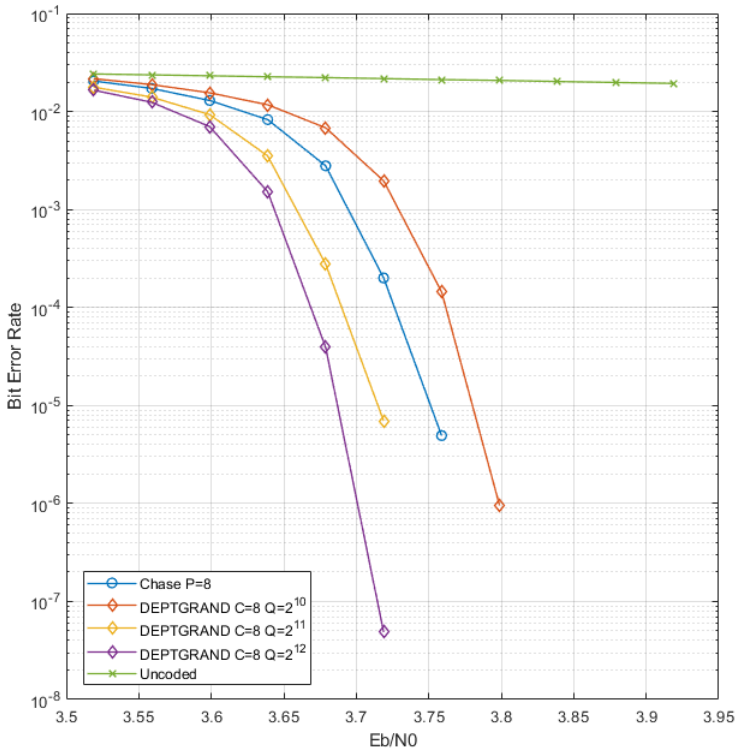


Figure 5.9: Performance of DEPTGRAND on oFEC with the different parameters of C and Q.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This research presents an in-depth analysis of oFEC by applying DEPT-based decoding algorithm and compare with the typical Chase-Pyndiah decoding algorithm. DEPT-based decoding algorithm proposed different mechanisms of generating candidate codewords. Chase list out all 2^p feasible candidate codewords as p number of LRBs are selected based on the reliability of LLRs. However, DEPT prefers to search for error patterns in predefined PEPs according to the error probabilities of their respective error weights. Decoder estimates the number of error is even or odd and search corresponds PEP set and locate possible error location. It enables for a more precise determination of the error positions. The simulation results illustrates in the previous section also prove that DEPT with the decode radius of 10 has higher efficiency and accuracy on error correction compared with Chase p=8. In comparison with the Chase-Pyndiah algorithm, DEPT provides around 0.02dB of improvement when it apppled with oFEC.

DEPT-ORBGRAND provides another further improvements based on DEPT algorithm. It inherit the idea of estimating the number of errors in the cases of even and odd based on syndrome, proposed the idea of generating queries regarding to the logistic weight of errors and selected C number of great possible error patterns to implement decoding. The pre-FEC BER threshold of DEPT-ORBGRAND with utilizing the parameter of C=10 and $Q = 2^{12}$ achieves around $2.12e^{-2}$ according to the result illustrates in the tables. In other words, DEPOT-ORBGRAND brings a 0.05dB improvement under the specified parameters compared to oFEC with Chase algorithm applied. However, the complexity of the DEPT-ORBGRAND is the highest among the three algorithms. In addition, the table-based hard decoder, consider bounded decode distance, is more appliable for oFEC than the conventional hard decoder, bringing an additional 0.22dB gain.

6.2 Future Work

The error correction of DEPT-ORBGRAND is quite inefficient due to its high complexity. Searching for error patterns from 2^{12} queries at each stage of decoding is tedious. In terms of software implementation, the time consumption of simulation is relatively high. Power consumption is also a major issue during the hardware implementation of DEPT-ORBGRAND. As a result, DEPT-ORBGRAND is worthy of further research and seeking a method to improve the efficiency of error pattern query, such as by using parallelization techniques to speed up the process. Alternatively, further filtering out unnecessary error patterns from queries is another possible option for efficiency improvement.

Bibliography

[1] D. Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, 18(1):170–182, 1972.

[2] Kevin Cushon, Per Larsson-Edefors, and Peter Andrekson. Low-power 400-gbps soft-decision ldpc fec for optical transport networks. *Journal of Lightwave Technology*, 34(18):4304–4311, 2016.

[3] Sameep Dave, Junghwan Kim, and Subhash C Kwatra. An efficient decoding algorithm for block turbo codes. *IEEE Transactions on communications*, 49(1):41–46, 2001.

[4] Ken R Duffy, Wei An, and Muriel Médard. Ordered reliability bits guessing random additive noise decoding. *IEEE Transactions on Signal Processing*, 70:4528–4542, 2022.

[5] Ken R Duffy, Jiange Li, and Muriel Médard. Guessing noise, not code-words. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 671–675. IEEE, 2018.

[6] Peter Elias. Error-free coding. pages 29–37, 1954.

[7] Alberto JimÉnez Feltstrom, Dmitri Truhachev, Michael Lentmaier, and Kamil Sh. Zigangirov. Braided block codes. *IEEE Transactions on Information Theory*, 55(6):2640–2658, 2009.

[8] G David Forney. Concatenated codes. 1965.

[9] G ITU-T. Interfaces for the optical transport network (otn). *Recommendation G*, 709, 2009.

[10] Michael Lentmaier, Dmitri Truhachev, and Kamil Zigangirov. Iteratively decodable sliding codes on graphs. In *Workshop on Algebraic and Combinatorial Coding Theory (ACCT)*, pages 190–193, 2002.

[11] Shizhong Li, Kamal El-Sankary, Alireza Karami, and Dmitri Truhachev. Area- and power-efficient staircase encoder implementation for high-throughput fiber-optical communications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(3):843–847, 2020.

[12] Annie Picart and Ramesh Pyndiah. Adapted iterative decoding of product codes. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99.(Cat. No. 99CH37042)*, volume 5, pages 2357–2362. IEEE, 1999.

[13] R.M. Pyndiah. Near-optimum decoding of product codes: block turbo codes. *IEEE Transactions on Communications*, 46(8):1003–1010, 1998.

[14] ITU Recommendation. Itu-t g. 975, 2000.

[15] RA Silverman and Martin Balser. Coding for constant-data-rate systems-part i. a new error-correcting code. *Proceedings of the IRE*, 42(9):1428–1435, 1954.

[16] Mike A Sluysk. Open roadm msa 3.01 w-port digital specification (200g-400g). *Acacia Commun. Inc. 3 Mill and Main*, 2019.

[17] Benjamin P. Smith, Arash Farhood, Andrew Hunt, Frank R. Kschischang, and John Lodge. Staircase codes: Fec for 100 gb/s otn. *Journal of Lightwave Technology*, 30(1):110–117, 2012.

[18] Alvin Yonathan Sukmadji. *Zipper codes: High-rate spatially-coupled codes with algebraic component codes*. University of Toronto (Canada), 2020.

[19] ITU Telecommunication. Itu-t g. 975.1 recommendation: Forward error correction for high bit-rate dwdm submarine systems. *SERIES G Recommendation: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS: Digital Sections and Digital Line Systems—Optical Fibre Submarine Cable Systems*, 2004.

[20] D. Truhachev, M. Lentmaier, and K. Zigangirov. On braided block codes. In *IEEE International Symposium on Information Theory, 2003. Proceedings.*, pages 32–, 2003.

[21] Dmitri Truhachev, Kamal El-Sankary, Alireza Karami, Abolfazl Zokaei, and Shizhong Li. Efficient implementation of 400 gbps optical communication fec. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):496–509, 2021.

[22] Weiming Wang, Weifeng Qian, Kai Tao, Zitao Wei, Shihua Zhang, Yan Xia, and Yong Chen. Investigation of potential fec schemes for 800g-zr forward error correction. In *2022 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3. IEEE, 2022.

[23] Lei M. Zhang and Frank R. Kschischang. Staircase codes with 6 *Journal of Lightwave Technology*, 32(10):1999–2002, 2014.

[24] Wei Zhang, Michael Lentmaier, Daniel J. Costello, and K.Sh. Zigangirov. Braided convolutional codes. In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pages 592–596, 2005.

[25] Wei Zhang, Michael Lentmaier, Kamil Sh. Zigangirov, and Daniel J. Costello. Braided convolutional codes: A new class of turbo-like codes. *IEEE Transactions on Information Theory*, 56(1):316–331, 2010.