

# Design and Implementation of Improved Decoding Algorithms for LDPC Convolutional Codes

Entwurf und Implementierung von verbesserten Decodieralgorithmen für LDPC-Faltungscodes

Master-Thesis von Sakthivel Velumani aus Salem, Indien

Tag der Einreichung: 14.01.2019

1. Betreuer: M.Sc. Janik Frenzel
2. Betreuer: M.Sc. Bastian Alt
3. Betreuer: Prof. Dr. techn. Heinz Koepl



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fachbereich Elektrotechnik und  
Informationstechnik  
Bioinspired Communication Systems  
Lab

Design and Implementation of Improved Decoding Algorithms for LDPC Convolutional Codes  
Entwurf und Implementierung von verbesserten Decodieralgorithmen für LDPC-Faltungscodes

Vorgelegte Master-Thesis von Sakthivel Velumani aus Salem, Indien

Matrikelnummer: 2806422

1. Betreuer: M.Sc. Janik Frenzel
2. Betreuer: M.Sc. Bastian Alt
3. Betreuer: Prof. Dr. techn. Heinz Koeppel

Tag der Einreichung: 14.01.2019

Bitte zitieren Sie dieses Dokument als:

URN: urn:nbn:de:tuda-tuprints-82349

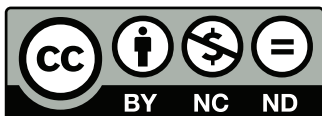
URL: <http://tuprints.ulb.tu-darmstadt.de/8234>

Dieses Dokument wird bereitgestellt von tuprints,

E-Publishing-Service der TU Darmstadt

<http://tuprints.ulb.tu-darmstadt.de>

[tuprints@ulb.tu-darmstadt.de](mailto:tuprints@ulb.tu-darmstadt.de)



Die Veröffentlichung steht unter folgender Creative Commons Lizenz:

Namensnennung – Keine kommerzielle Nutzung – Keine Bearbeitung 2.0 Deutschland

<http://creativecommons.org/licenses/by-nc-nd/2.0/de/>

---

## **Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt**

Hiermit versichere ich, Sakthivel Velumani, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

---

### **English translation for information purposes only:**

#### **Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt**

I herewith formally declare that I, Sakthivel Velumani have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content.

---

Datum / Date:

Unterschrift/Signature:

---

---

---



---

# Acknowledgment

This work represents the results of the six-month Master's thesis project carried out at Intel Deutschland GmbH, Nürnberg in cooperation with Bioinspired Communication Systems Lab in Technische Universität Darmstadt.

First of all, I would like to thank my supervisor M.Sc. Janik Frenzel and his supervisor Dr.-Ing. Stefan Muller-Weinfurter for having faith in me and give me this wonderful opportunity. I would like to also thank my supervisor at the university M.Sc. Bastian Alt and the head of the group Prof. Dr. techn. Heinz Koepl for accepting my thesis work done at Intel Deutschland GmbH.

Especially, I would like to thank Janik Frenzel for introducing me to the field of LDPC Codes with a practical and industrial perspective. The several discussions we had during this project certainly improved my insights in the topic and helped to overcome the ambiguities in my understanding. I greatly appreciate his patience and efforts to answer my questions as well as to review this work.

Special thanks to my family and friends in India for providing moral support. Without them, this work would not have been possible.



---

# Abstract

In this work, techniques to improve the performance of window based decoder for Low-Density Parity-Check Convolutional Code (LDPC-CC) are developed. In order to evaluate the performance of our techniques, we choose the codes from the standard IEEE 1901. Our focus is on terminated LDPC-CC and so we first investigate the termination mechanism in the encoder. We found that a proper end-termination for the IEEE 1901's LDPC-CCs is infeasible. Two improvement techniques for window decoders are developed and evaluated. The first technique focuses on the movement direction of the window in which the window starts and ends at the same position in the Parity Check Matrix (PCM). Two different configurations of the first technique are discussed. The second improvement technique focuses on the early-success criteria for window termination where a dynamic criteria that changes based on window size. The simulations are carried on an Additive White Gaussian Noise (AWGN) channel based system model. The first technique is proved to have better decoding performance at a reduced decoding complexity. The second technique has a reduced complexity with about the same performance as the decoder without the technique.





---

# Contents

<b>Abstract</b>	<b>V</b>
<b>List of Figures</b>	<b>IX</b>
<b>List of Tables</b>	<b>XI</b>
<b>Acronyms</b>	<b>XIII</b>
<b>List of Symbols</b>	<b>XV</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>3</b>
2.1 Introduction to Channel Coding . . . . .	3
2.2 Channel Codes . . . . .	4
2.2.1 Linear Block Codes . . . . .	4
2.2.2 Low-Density Parity-Check Block Codes . . . . .	5
2.2.3 Convolutional Codes . . . . .	6
2.2.4 Low-Density Parity-Check Convolutional Codes . . . . .	6
2.2.5 Termination of Convolutional Codes . . . . .	7
2.2.6 Low-Density Parity-Check Convolutional Code Used in IEEE 1901 . . . . .	8
2.3 Decoding of Low-Density Parity-Check Codes . . . . .	10
2.3.1 Belief Propagation . . . . .	10
2.3.2 LDPC-CC-specific Decoding Techniques . . . . .	13
2.4 System Model . . . . .	16
<b>3 Encoding of the Broadband Power Line Codes</b>	<b>21</b>
3.1 Encoder Design . . . . .	21
3.2 Termination Sequence . . . . .	22
3.2.1 Proper Termination . . . . .	23
3.2.2 Zero-Tail Termination . . . . .	24
<b>4 Decoding Improvements</b>	<b>27</b>
4.1 Existing Techniques and Motivation . . . . .	27
4.2 Base Decoder Configuration . . . . .	27
4.3 Left-Right-Left Windowed Decoding . . . . .	28
4.4 Improved Partial-Syndrome-Check . . . . .	30
<b>5 Simulation Results and Evaluation</b>	<b>33</b>
5.1 Experiment Setup . . . . .	33
5.2 Evaluation of Zero-tail Termination . . . . .	33
5.3 Evaluation of Base Decoder . . . . .	35
5.4 Evaluation of Left-Right-Left Decoder . . . . .	35
5.5 Evaluation of Improved Partial-Syndrome-Check Technique . . . . .	39

---

<b>6</b>	<b>Implementation Aspects</b>	<b>45</b>
6.1	Variable Node Storage Memory Format . . . . .	45
6.2	Variable Node Indexing . . . . .	46
<b>7</b>	<b>Conclusion</b>	<b>49</b>
	<b>Bibliography</b>	<b>XVII</b>

# List of Figures

1.1	Evolution of mobile telephone technologies and type of error-correcting codes used. . . . .	1
2.1	Block diagram of a digital communication system. . . . .	4
2.2	Tanner graph of the code from (2.5). The dark shaded circles represents Variable Nodes (VNs) and the crossed circles Check Nodes (CNs). All CNs have degree 3 whereas the VN degrees vary between 1 and 3. . . . .	5
2.3	Example of a non-recursive convolutional code with rate $R = 1/2$ . $x[i]$ is the input and $y[i]$ is the output. . . . .	6
2.4	Tanner graph of a terminated LDPC-CC. The dark circles and lines are the VNs and edges of terminated code, the light circles and dashed lines are the omitted VNs and edges as a result of termination. . . . .	8
2.5	Relation between A-Posteriori Probability (APP) and Log-likelihood ratio (LLR). . . . .	11
2.6	Example showing that messages from all other CNs sum-up with the channel's LLR to form the V2C message to the first CN. . . . .	11
2.7	Example showing that messages from all other VNs combine using equation (2.28) to form the C2V message to the second VN. . . . .	12
2.8	Illustration of parallel scheduling with an example Tanner graph. . . . .	12
2.9	Illustration of serial scheduling with the same Tanner graph as in Figure 2.8. The processing starts from the left most CN and moves right to the last CN. . . . .	13
2.10	PCM illustrating the pipeline decoding technique for code with $R_\infty = 1/2$ , $m_s = 2$ and $I = 3$ . The arrows indicate the CNs that are processed at current time instance. . . . .	14
2.11	PCM illustrating the Windowed Decoding technique for $R_\infty = 1/2$ codes and $m_s = 2$ . The current window position $\rho = 5$ is indicated by the solid red box and the next window position is indicated by the dashed red box. . . . .	15
2.12	Target VNs for first, middle and the last window positions. . . . .	16
2.13	System model for simulations. . . . .	17
2.14	Symbol constellation of Quadrature Phase Shift Keying (QPSK) modulation with gray mapping. . . . .	17
2.15	Conditional probability density functions of a Binary Phase Shift Keying (BPSK) symbol over AWGN channel. . . . .	18
3.1	Illustration of input and output buffers and its contents. The red boxes represents the position of the parity bits. . . . .	22
3.2	Illustration of a parity bit being generated using information and other parity bits in the output buffer. The arrow indicates the direction of movement of the parity-bit generator. Note: An example code. . . . .	22
3.3	PCM illustrating that a truncated codeword do not satisfy all CNs in a $R_\infty = 2/3$ code with $m_s = 4$ . . . . .	23
3.4	PCM of an example LDPC-CC code with $R_\infty = 2/3$ and $m_s = 4$ illustrating sub-matrices used to determine the termination sequence. Depicted above the PCM is the codeword vector. . . . .	24
3.5	PCM and codeword after adding the zero-tailing termination sequence. . . . .	25

3.6	Illustration of a zero-tail-terminated codeword of a code with $R_\infty = 2/3$ . Black boxes represent information bits, blue boxes represent zero-tail bits and red boxes represent parity bits. . . . .	25
4.1	Probability of error for each bit in the codeword of a code with $R_\infty = 2/3$ , $n_i = 3500$ . Termination bits are excluded. . . . .	29
4.2	PCM illustrating Left-Right-Left (LRL) decoder. . . . .	29
4.3	Different window configurations used in LRL decoder. Left image illustrates LRL configuration-I and right image illustrates LRL configuration-II. . . . .	30
4.4	PCM illustrating complete-VNs, incomplete-VNs and complete-CNs. . . . .	30
5.1	Probability of error for information and parity bits in the codeword. Zero-tail bits are not known at the receiver. Simulation parameters are $n_i = 3500$ , $R_\infty = 2/3$ , $W = 700$ and $\zeta = 2$ dB. . . . .	34
5.2	Probability of error for information and parity bits in the codeword. Zero-tail bits are known at the receiver. Simulation parameters are $n_i = 3500$ , $R_\infty = 2/3$ , $W = 700$ and $\zeta = 2$ dB. . . . .	34
5.3	Block-Error Rate (BLER) vs Signal-to-Noise Ratio (SNR) of the Base Decoder (BD) with $n_i = 3500$ and $R_\infty = 2/3$ . . . . .	35
5.4	BLER vs SNR of the Base Decoder (BD) with $n_i = 3500$ and $R_\infty = 2/3$ . . . . .	36
5.5	BLER vs SNR of the Base Decoder (BD) with $n_i = 3500$ and $W = 500$ . . . . .	36
5.6	Comparison of BLER of the Base Decoder (BD) for different $I$ with $n_i = 3500$ and $W = 300$ . . . . .	37
5.7	Comparison of Average Number of Edge Updates (ANEU) of the Base Decoder (BD) for different $I$ with $n_i = 3500$ and $W = 300$ . . . . .	37
5.8	Comparison of BLER between the Base Decoder and LRL decoder with $W = 300$ . . . . .	38
5.9	Comparison of ANEU between the Base Decoder and LRL decoder with $W = 300$ . . . . .	38
5.10	Comparison of BLER between the Base Decoder and LRL decoder with $W = 700$ . . . . .	39
5.11	Comparison of ANEU between the Base Decoder and LRL decoder with $W = 700$ . . . . .	40
5.12	Comparison of BLER between the LRL decoders configuration-I and configuration-II with $W = 300$ . . . . .	40
5.13	Comparison of Bit-Error Rate (BER) between the LRL decoders configuration-I and configuration-II with $W = 300$ . . . . .	41
5.14	Comparison of ANEU between the LRL decoders configuration-I and configuration-II with $W = 300$ . . . . .	41
5.15	Comparison of BLER between different early-success criteria with $W = 300$ . . . . .	42
5.16	Comparison of ANEU between different early-success criteria with $W = 300$ . . . . .	42
5.17	Comparison of BLER between different early-success criteria with $W = 600$ . . . . .	43
5.18	Comparison of ANEU between different early-success criteria with $W = 600$ . . . . .	43
6.1	Three bits of value 0, 1 and 1 are stored in single byte each. The arrow indicates the position of Least Significant Bit (LSB) where the bit is stored in each byte. The bits marked with x are unused. Note that the bytes are represented in little-endian format. . . . .	45
6.2	Direct access and use of bits. . . . .	45
6.3	Packed bits of bytes. . . . .	45
6.4	Access and use of bits using helper functions. . . . .	46

---

## List of Tables

3.1	Difference between proper termination and zero-tail termination. . . . .	25
4.1	Early-success criteria for Improved Partial-Syndrome-Check (IPSC) . . . . .	31
5.1	Experimental settings for simulations. . . . .	33
6.1	Memory required to store 5000 VNs . . . . .	46
6.2	Number of different operations performed during each call of <i>Load Bit</i> and <i>Store Bit</i> . . . .	46
6.3	Memory requirement and operations required to encode a codeword with 5000 information bits with $R_\infty = 1/2$ . . . . .	47
6.4	Memory requirement and computations required to calculate and store indices for decoding a codeword with 5000 information bits with $R_\infty = 1/2$ , maximum window size $W = 2500$ and $I = 5$ . . . . .	47



---

# Acronyms

ANEU	Average Number of Edge Updates
APP	A-Posteriori Probability
AWGN	Additive White Gaussian Noise
BD	Base Decoder
BEC	Binary Errasure Channel
BER	Bit-Error Rate
BLER	Block-Error Rate
BP	Belief Propagation
BPL	Broadband Power Line
BPSK	Binary Phase Shift Keying
C2V	check-node-to-variable-node
CB	Code Block
CN	Check Node
FIR	Finite Impulse Response
GM	Generator Matrix
ID	Improved Decoder
IIR	Infinite Impulse Response
IPSC	Improved Partial-Syndrome-Check
LAN	Local Area Network
LDPC	Low-Density Parity-Check
LDPC-BC	Low-Density Parity-Check Block Code
LDPC-CC	Low-Density Parity-Check Convolutional Code
LLR	Log-likelihood ratio
LRL	Left-Right-Left
LSB	Least Significant Bit
MAP	Maximum-A-Posteriori
ML	Maximum-Likelihood
MPA	Message-Passing Algorithm
MSA	Min-Sum Algorithm

---

OSI	Open System Interconnection
PCM	Parity Check Matrix
PDF	Probability Density Function
PHY	Physical
PSC	Partial-Syndrome-Check
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
SC-LDPC	Spatially-Coupled LDPC
SNR	Signal-to-Noise Ratio
SPA	sum-product algorithm
V2C	variable-node-to-check-node
VN	Variable Node
WD	Windowed Decoding
WiMAX	Worldwide Interoperability for Microwave Access



---

# List of Symbols

$I$	Maximum number of iterations allowed within a window
$L$	Coupling Length
$P_L$	Block-Error Rate (BLER)
$P_b$	Overall Bit-Error Rate (BER)
$P$	Actual Probability
$R_\infty$	Asymptotic Code Rate
$R$	Code Rate
$W$	Window Size
$\eta$	Average Number of Edge Updates (ANEU)
$\kappa$	Decoder's output bit rate
$\mathbf{G}$	Generator Matrix
$\mathbf{H}$	Parity-Check Matrix
$\mathbf{b}$	Vector of parity bits
$\mathbf{d}$	Vector of source bits
$\mathbf{m}$	Vector of source encoded (information) bits
$\mathbf{s}$	Syndrome
$\mathbf{u}$	Vector of modulated symbols
$\mathbf{v}$	Vector of received symbols
$\mathbf{x}$	Vector of codeword bits
$\mathbf{y}$	Vector of received codeword bits
$\mathbf{z}$	Vector of noise samples
$\psi$	Time taken to perform one CN Processing
$\rho$	Window position in a window decoder
$\hat{\mathbf{d}}$	Vector of estimated source bits
$\hat{\mathbf{m}}$	Vector of estimated information bits
$\hat{\mathbf{x}}$	Vector of estimated codeword bits
$\zeta$	Signal-to-Noise Ratio (SNR) in dB
$k$	Number of information bits in a Code Block
$l_c$	Constraint Length

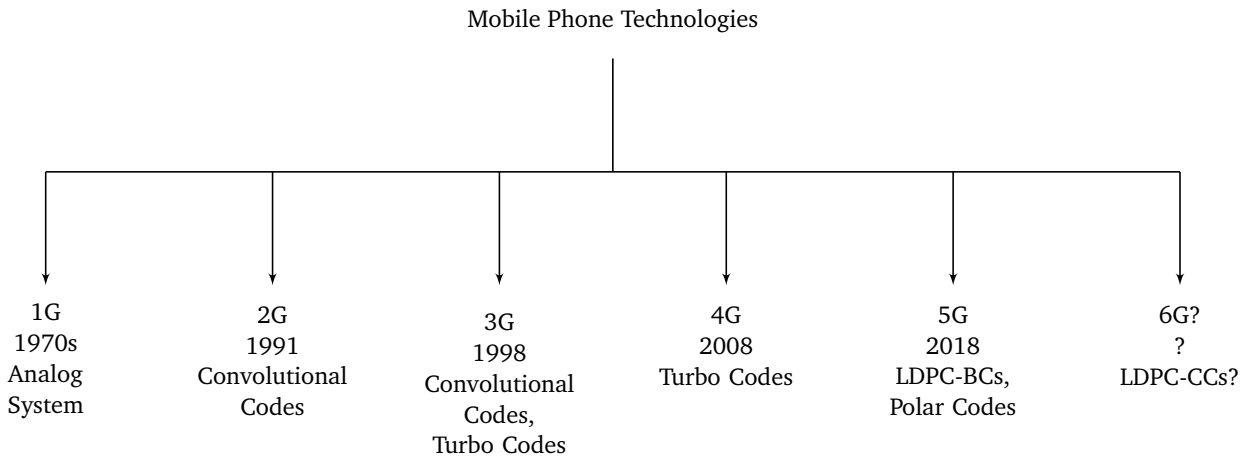
---

$m_s$	Constraint Length (or) Memory of a convolutional code
$n_m$	Number of information bits in the codeword
$n_z$	Number of Zero-tail bits in the codeword
$n$	Number of bits in a Code Block
$o$	Modulation Order
$p$	Probability Density Function
$q$	Variable Node degree
$r$	Check Node degree

# 1 Introduction

LDPC-CCs are a class of linear block codes which are widely used in data transmission and storage systems. LDPC-CCs are proven to be better than Low-Density Parity-Check Block Codes (LDPC-BCs) because of their convolutional nature and the infinitely long codeword. With terminated codes, LDPC-CC are comparable to LDPC-BC if the codeword length is large. Along side the code performance, the LDPC-CC have an advantage in its decodability. Its convolutional structure allows for window decoding instead of full block decoding. However with window decoding, there is always a trade-off between decoding latency and decoding performance.

LDPC-CCs are applications like used in Broadband Power Line (BPL) (IEEE 1901) and Worldwide Interoperability for Microwave Access (WiMAX) (IEEE 802.16) standards to name a few. In more widely used applications like mobile telephony (New Radio) and wireless Local Area Network (LAN), so far LDPC-BC are being used. But due to the better performance and increasing popularity of LDPC-CC, they are more likely to be used in large number of applications in the future.




**Figure 1.1:** Evolution of mobile telephone technologies and type of error-correcting codes used.

There are quite some amount of window decoding algorithms that are proposed in recent years. One of them is the *Zigzag decoder* that was proposed by Abu-Surra in [1]. This decoder moves the window forward and backward in small part of the PCM. Another decoder proposed in [2] used a technique to reduce the error propagating from left to right of the PCM as the window moves. Three more improvement techniques for window decoder are proposed in [3] which are discussed in later chapters. The aforementioned decoding techniques either use flooding schedule or are evaluated with Binary Erasure Channel (BEC). We are interested in serial scheduling and AWGN channel.

In this thesis, we propose a LRL decoder with two configurations and an IPSC technique. The LRL decoder which is motivated by the Zigzag decoder moves the window forward and backward. The IPSC is an improvement over Partial-Syndrom-Check (PSC) proposed in [2]. We also investigate the termination problem in the LDPC-CC used in the standard IEEE 1901. We evaluate the proposed techniques with an AWGN channel system model.

The organization of this thesis is as follows. Chapter 2 contains the theory of channel coding and channel codes with an emphasis on LDPC-CC. We also discuss the system model used for our simulations. In Chapter 3, we discuss the termination problem in IEEE 1901's LDPC-CC. We then discuss the proposed



---

decoding techniques and their evaluation in Chapters 4 and 5 respectively. Finally in Chapter 7, we conclude our findings during the evaluation of our techniques and discuss about possible future works.

---

## 2 Background

In this chapter, we provide an overview of channel coding and some selected channel codes. We start by discussing the need for channel coding. We then describe linear block codes, Low-Density Parity-Check (LDPC) codes and LDPC-CCs which are the main focus of this thesis. Then we go on to discuss Belief Propagation (BP) and the sliding-window technique used in decoders for LDPC-CCs. We finish the section describing our system model.

---

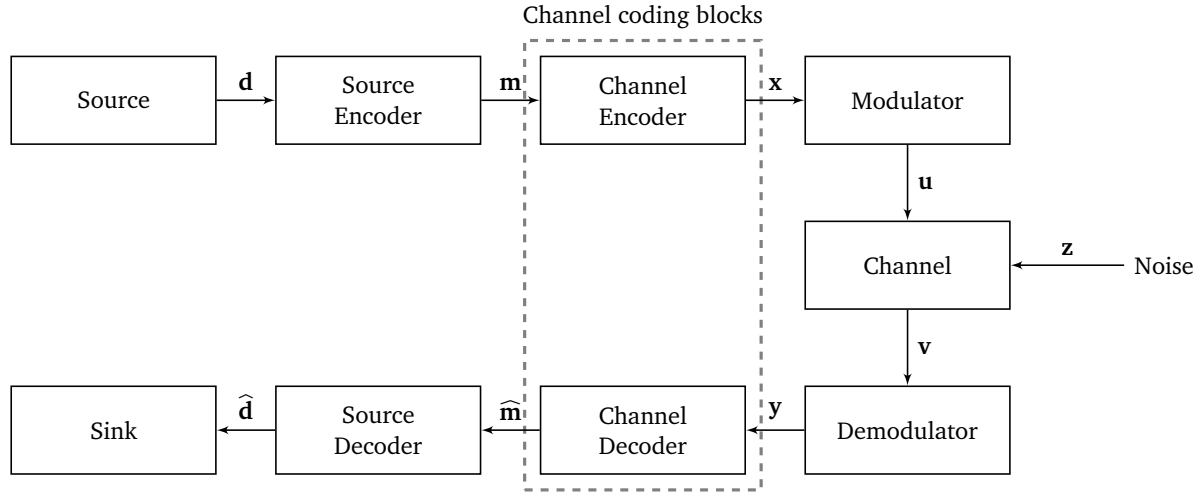
### 2.1 Introduction to Channel Coding

---

The term *coding* is generally associated with the mapping of information to a set of symbols or numbers [4]. Source coding aims to remove the redundancy in the information whereas channel coding aims to make the information immune to random distortion. A model of a digital communication system is shown in Figure 2.1. Let us consider that the *source* block produces a sequence of information bits given by the vector  $\mathbf{d}$ . These bits might stream from any digital information source such as multimedia files, text documents, etc. These vectors of bits are encoded in the *source-encoder* block to produce reduced set of information bit vectors  $\mathbf{m}$  called source codewords. The reduction usually means that the length of  $\mathbf{m}$  is at most the length of  $\mathbf{d}$ . The mapping of information bits to a set of reduced information bits allows unique reconstruction of the information bits at the receiver. The source encoder is chosen depending on the type of the information source [5].

The next block in the digital communication model is the *channel encoder*. Whereas the source encoder compresses the information bit vectors, the channel encoder expands them by adding redundant bits in a structured manner. This structured redundancy makes the transmitted information bits less susceptible to distortions such as interference in the channel noise. A *channel* is a medium through which the information is transferred from transmitter to receiver. A *code* is a set of rules that defines the encoding principle of the encoder. The type of code is chosen depending on the channel and the application requirements. In general, the source encoder or decoder is placed at higher layers of the Open System Interconnection (OSI) model, while channel-coding blocks are placed at the Physical (PHY) layer. The outputs of the channel encoder are called channel codewords. The codeword vectors  $\mathbf{x}$  are then modulated in the *modulator* block where the bits are transformed into symbol vectors  $\mathbf{u}$ . The symbol vectors are then transmitted as analog signals through the channel. Due to the addition of interference and noise, the channel output is in general not the same as the channel input:  $\mathbf{v} \neq \mathbf{u}$ . The *demodulator* converts the received symbol vectors  $\mathbf{v}$  into vectors of bits  $\mathbf{y}$  which corresponds to the vector of encoded bits  $\mathbf{x}$ . The *channel decoder* uses the redundancy in the received codeword to deduce an estimate  $\hat{\mathbf{m}}$  of the source codeword. The source decoder then deduces an estimate  $\hat{\mathbf{d}}$  of the information bit vector from  $\hat{\mathbf{m}}$ .

The addition of redundant bits by the channel encoder enables the mapping between a set of information words and a set of all possible *receive words*. Let us assume the length of an information word  $\mathbf{m}$  to be  $k$  bits and the length of a codeword  $\mathbf{x}$  to be  $n$  bits such that  $n > k$ . Thus, the information word set has  $2^k$  words and the receive word set has  $2^n$  words. The codeword set of size  $2^k$  is a subset of the receive word set. The mapping between different set sizes allows us to detect if the received word is in the codeword set. The information words and codewords contain elements from the binary set  $\mathbb{F}_2 = \{0, 1\}$ .  $\mathbb{F}_2$  or GF(2) is called a finite field of order 2. Hence, all arithmetic operations with information bits and codewords are performed modulo 2.



**Figure 2.1:** Block diagram of a digital communication system.

## 2.2 Channel Codes

There are different types of channel codes. The choice of one depends on the application requirements, type of channel medium and resource availability. In this thesis, we focus on LDPC-CCs, a special class of LDPC codes.

### 2.2.1 Linear Block Codes

Linear block codes are codes in which a codeword is formed by a linear combination of two or more base vectors that span the codeword space [5]. Hence, the base vectors are also codewords. As a result, a linear combination of any two or more codewords forms another codeword. The codeword space of  $2^k$  vectors is a subspace of the space of all  $2^n$  vectors. An  $(n, k)$  linear block code maps  $k$  message bits to  $n$  codeword bits. The remaining  $n - k$  redundant bits are called parity bits and they are determined by an encoding rule. Linear block codes are classified into two categories: *systematic* and *non-systematic*. Systematic codes have all their message bits transmitted in an unaltered manner whereas the non-systematic codes do not have such formation. Without loss of generality, we assume that a codeword of a systematic linear block code has the following structure:

$$\mathbf{x}^T = \begin{bmatrix} \mathbf{m}^T & \mathbf{b}^T \end{bmatrix} \quad (2.1)$$

where  $\mathbf{x} \in \mathbb{F}_2^{n \times 1}$  is the codeword vector,  $\mathbf{m} \in \mathbb{F}_2^{k \times 1}$  and  $\mathbf{b} \in \mathbb{F}_2^{(n-k) \times 1}$  denote message and parity vectors, respectively. The code rate is given by

$$R = \frac{k}{n}. \quad (2.2)$$

Codewords of linear block codes are expressed as

$$\mathbf{x} = \mathbf{G} \odot \mathbf{m} \quad (2.3)$$

where  $\mathbf{G} \in \mathbb{F}_2^{n \times k}$  is the Generator Matrix (GM) and  $\odot$  represents multiplication modulo 2. A parity check is described by the expression

$$\mathbf{s} = \mathbf{H} \odot \mathbf{x} \quad (2.4)$$

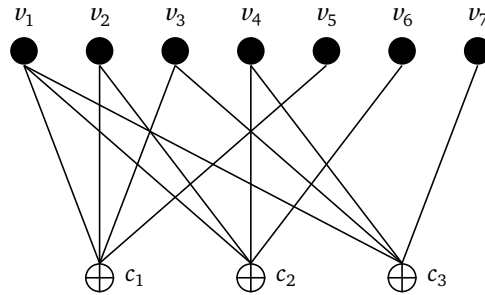
where  $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$  is called the PCM and  $\mathbf{s} \in \mathbb{F}_2^{(n-k) \times 1}$  is called the syndrome. Each row of the PCM represents a parity-check equation. Only when  $\mathbf{s} = \mathbf{0}$ , the parity checks are fulfilled. The relation between PCM and GM is given by  $\mathbf{H} \odot \mathbf{G} = \mathbf{0}$ . With either GM or PCM given, the other one is not unique. For example, if the PCM of a (7, 4) hamming code is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad (2.5)$$

then the GM can be formed by combining any 3 rows of  $\text{null}(\mathbf{H})$ , i.e., the right null space of  $\mathbf{H}$ .

$$\mathbf{G}^T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (2.6)$$

The PCM can be represented by a bipartite graph called Tanner graph [6]. The Tanner graph has two sets of nodes: VNs represent columns and CNs represent rows of the PCM. Each non-zero entry in the PCM is represented by an edge between the respective VN and CN. The *degree* of a node is the number of edges connected to it. The Tanner graph of the example PCM in (2.5) is shown in Figure 2.2.



**Figure 2.2:** Tanner graph of the code from (2.5). The dark shaded circles represents VNs and the crossed circles CNs. All CNs have degree 3 whereas the VN degrees vary between 1 and 3.

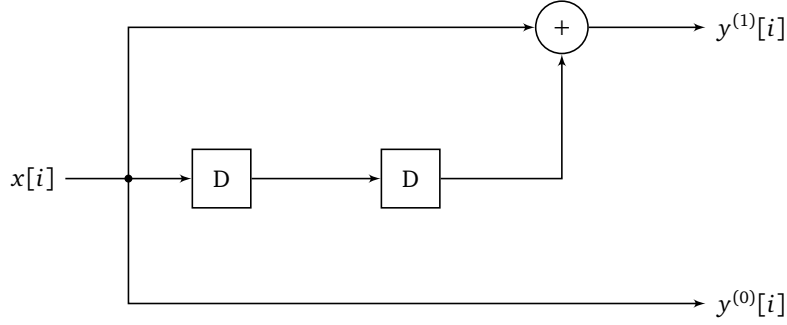
### 2.2.2 Low-Density Parity-Check Block Codes

LDPC-BCs are a class of linear block codes which were introduced by Robert Gallager in 1963 [7]. As the name specifies, they are defined by a sparse PCM containing mostly 0's and relatively few 1's. The sparsity of the PCM or its Tanner graph is a key property that allows for the algorithmic efficiency of decoding LDPC-BCs. These codes are divided into two types: regular and irregular codes.

In a regular  $(n, q, r)$  code, all VNs have degree  $q$  and all CNs have degree  $r$ .

### 2.2.3 Convolutional Codes

Convolutional codes in general, are codes in which the parity bits are generated by convolving information bits or information and parity bits. The polynomial coefficients for the convolution is given by the generator polynomial. This generator polynomial is also the taps of an Finite Impulse Response (FIR) filter in case of non-recursive codes and an Infinite Impulse Response (IIR) filter in case of recursive codes. An example of parity-bit generation in a non-recursive systematic convolutional code is shown in Figure 2.3.



**Figure 2.3:** Example of a non-recursive convolutional code with rate  $R = 1/2$ .  $x[i]$  is the input and  $y[i]$  is the output.

The generator polynomial of this example is given by

$$G^{(0)}(D) = 1 \quad (2.7)$$

$$G^{(1)}(D) = 1 + D^2. \quad (2.8)$$

The impulse response of the parity-bit generator is given by

$$g^{(1)}[i] = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}. \quad (2.9)$$

The output is given by the convolution form:

$$y^{(1)}[i] = x[i] * g^{(1)}[i] \quad (2.10)$$

$$= \sum_{l=0}^2 x[l] g^{(1)}[i-l]. \quad (2.11)$$

The *constraint length* of a convolutional code is  $l_c = m_s + 1$  where  $m_s$  is the largest degree in  $g[i]$ . In the example in Figure 2.3,  $l_c = 3$ .

### 2.2.4 Low-Density Parity-Check Convolutional Codes

LDPC-CCs or Spatially-Coupled LDPC (SC-LDPC) codes are formed by imposing the above mentioned convolutional structure on LDPC-BCs. They were invented by Alberto Felström and Kamil Zigangirov [8]. These codes are characterized by a sparse infinite-length PCM which has a diagonal structure. The PCM of these codes is constructed by coupling PCMs of LDPC-BCs as given by



$$\mathbf{H}_{[-\infty, \infty]} = \begin{bmatrix} \ddots & & \ddots & & \ddots & & \ddots & \\ & \mathbf{H}_{m_s}(t-1) & \dots & \mathbf{H}_1(t-1) & \mathbf{H}_0(t-1) & & & \\ & & \mathbf{H}_{m_s}(t) & \dots & \mathbf{H}_1(t) & \mathbf{H}_0(t) & & \\ & & & \mathbf{H}_{m_s}(t+1) & \dots & \mathbf{H}_1(t+1) & \mathbf{H}_0(t+1) & \\ & & & & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{s}(t-1) \\ \mathbf{s}(t) \\ \mathbf{s}(t+1) \\ \vdots \end{bmatrix} \quad (2.12)$$

where the  $\mathbf{H}_\mu(t) \in \mathbb{F}_2^{(n-k) \times n}$ ,  $\mu = 0, \dots, m_s$  are PCM of different LDPC-BCs of rate  $R = k/n$  for different time instances and  $m_s$  is the memory of the code. Hence, the asymptotic rate of the resulting LDPC-CC is  $R_\infty = k/n$ .  $\mathbf{s}(t) \in \mathbb{F}_2^{(n-k) \times 1}$  denotes the syndromes resulting from the parity check equations. The codewords of such a code have the form

$$\mathbf{x}^T = \left[ \dots \quad \mathbf{x}(t-1)^T \quad \mathbf{x}(t)^T \quad \mathbf{x}(t+1)^T \quad \dots \right] \quad (2.13)$$

where each  $\mathbf{x}(t) \in \mathbb{F}_2^{n \times 1}$ . Given the PCM  $\mathbf{H}$  and a valid codeword  $\mathbf{x}$ , the following expression holds:

$$\mathbf{s}(t) = \sum_{\tau=0}^{m_s} \mathbf{H}_\tau(t) \mathbf{x}(t-\tau) \mod 2. \quad (2.14)$$

The equation (2.14) is a convolution representing the convolutional structure of  $\mathbf{H}$  in (2.12). Similar to equation (2.10) for convolutional codes.

The bits in the codeword  $\mathbf{x}$  are coupled together over a distance called the *constraint length* which is given by  $l_c = (m_s + 1)n$  bits.

---

### 2.2.5 Termination of Convolutional Codes

---

In general, LDPC-CCs have codewords and PCMs of infinite length. For packet-based communication networks, however, the whole packet has to be retransmitted in case of incorrect information bits in higher layers. Also, in a wireless medium the channel parameters change over time which requires the encoder to change its code rate on the fly. For the aforementioned reasons, terminated codes are a better choice.

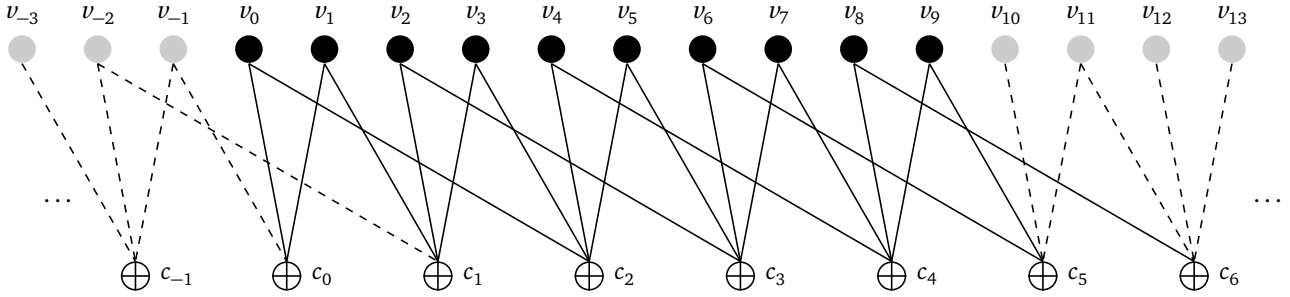
Termination is the process of limiting the coupling length, so that the codewords have finite length. This allows the decoder to stop decoding the current received word if a bit cannot be corrected, thus reducing the decoding complexity. The termination process requires appending *termination bits* to the end of the codeword to ensure that the last  $m_s$  parity-check equations of the terminated PCM are fulfilled. Termination also ensures that the encoder returns to an all-zero state before encoding the next codeword. For recursive convolutional codes, the termination bits are determined by solving a system of linear equations in  $\mathbb{F}_2$ . Whereas for non-recursive convolutional codes, appending a series of bits with value zero is sufficient.

Termination introduces a rate loss because the termination bits do not contain any information. Hence, for the rate calculation of a terminated LDPC-CC, the termination bits are not taken into account. However, the rate loss is compensated by an increase in decoding performance as the termination reduces the CN degrees at the end of the codeword and smaller CN degrees are better.

The PCM of a terminated LDPC-CC is a sub-matrix of the infinitely long PCM of the code (2.12). The terminated PCM has a structure as given by

$$\mathbf{H}_L = \left[ \begin{array}{cccc} \overbrace{\mathbf{H}_0(0)}^{Ln} & & & \\ \mathbf{H}_1(1) & \mathbf{H}_0(1) & & \\ \vdots & \mathbf{H}_1(2) & \ddots & \\ \mathbf{H}_{m_s}(m_s) & \vdots & \ddots & \mathbf{H}_0(L-1) \\ & \mathbf{H}_{m_s}(m_s+1) & \ddots & \mathbf{H}_1(L) \\ & & \vdots & \\ & & & \mathbf{H}_{m_s}(L+m_s) \end{array} \right] \left. \vphantom{\begin{array}{c} \mathbf{H}_0(0) \\ \mathbf{H}_1(1) \\ \vdots \\ \mathbf{H}_{m_s}(m_s) \\ \mathbf{H}_{m_s}(m_s+1) \\ \vdots \\ \mathbf{H}_{m_s}(L+m_s) \end{array}} \right\} (L+m_s)(n-k) \quad (2.15)$$

where  $L$  is the *coupling length* denoting the number of Code Blocks (CBs) in the codeword. Each CB contains  $n$  bits. Hence, the total length of the terminated codeword is  $n_L = Ln$  bits. The effect of termination in the Tanner graph of a  $R_\infty = 1/2$  code is shown in Figure 2.4. The entire graph in



**Figure 2.4:** Tanner graph of a terminated LDPC-CC. The dark circles and lines are the VNs and edges of terminated code, the light circles and dashed lines are the omitted VNs and edges as a result of termination.

Figure 2.4 can be seen as a Tanner graph of an infinitely long LDPC-CC. As a result of termination, only the center part of the graph remains. The dark circles are the VNs of the terminated LDPC-CC and the solid lines are their corresponding edges.

## 2.2.6 Low-Density Parity-Check Convolutional Code Used in IEEE 1901

In this thesis, we use the LDPC-CCs specified in the BPL or IEEE 1901 standard to evaluate our decoder [9]. From now on, we refer to the LDPC-CCs in the IEEE 1901 standard as *BPL codes*. The BPL codes are specified as sets of parity-check polynomials for all asymptotic rates  $R_\infty = k/n$ ,  $n \in \{2, 3, 4, 5\}$  where  $k = n - 1$ . In other words, the BPL codes have only one parity bit in each CB.

The codes are defined as

$$\sum_{i=1}^k A_{i,\tau}(D)M_i(D) + \sum_{i=1}^{n-k} C_{i,\tau}(D)B_i(D) \equiv 0 \pmod{2} \quad (2.16)$$

where  $k$  is the number of message bits in each CB,  $\tau \in \{0, \dots, T-1\}$  is the phase of the code that is given by  $\tau = (t \bmod T)$ ,  $T$  is the periodicity of the codes,  $M_i(D)$ ,  $i = 1, \dots, k$  represents message bits and  $B_i(D)$ ,  $i = 1, \dots, n-k$  represents parity bits.  $A_{i,\tau}$  and  $C_{i,\tau}$  define the connections between the bits based on delay  $D$ .

The memory  $m_s$  of the code is

$$m_s = \max(\{\deg(A_{i,\tau}(D)) : i = 1, \dots, k; \forall \tau\} \cup \{\deg(C_{i,\tau}(D)) : i = 1, \dots, n-k; \forall \tau\}) \quad (2.17)$$

where  $\tau \in \{0, \dots, T-1\}$  and  $\deg(f(x))$  denotes the set of all degrees of  $x$  in  $f(x)$ .

The BPL codes are periodic with  $T = 3$ . Periodic codes have time-varying parity-check polynomials which repeat every  $T$  CBs. For illustration, the parity-check polynomial of the BPL code for  $R_\infty = 2/3$  and  $\tau = 0$  is given by

$$(D^{214} + D^{185} + 1)M_1(D) + (D^{194} + D^{67} + 1)M_2(D) + (D^{215} + D^{145} + 1)B(D) = 0 \pmod{2}, \quad (2.18)$$

for  $\tau = 1$  as

$$(D^{160} + D^{62} + 1)M_1(D) + (D^{226} + D^{209} + 1)M_2(D) + (D^{206} + D^{127} + 1)B(D) = 0 \pmod{2}, \quad (2.19)$$

and for  $\tau = 2$  as

$$(D^{196} + D^{143} + 1)M_1(D) + (D^{115} + D^{104} + 1)M_2(D) + (D^{211} + D^{119} + 1)B(D) = 0 \pmod{2}. \quad (2.20)$$

$m_s = 215$  for  $n = 2$  and  $m_s = 226$  for  $n \neq 2$ . All  $A_{i,\tau}$  and  $C_{i,\tau}$  have three taps for each bit in the CB. There is a maximum of  $3n$  taps or edges per CN.

[9] specifies that termination is achieved by appending bits with value 0 to the end of the message bits before encoding. These bits are called *zero-tail bits*. The number of zero-tail bits  $n_z$  depends on the number of message bits  $n_m$  in the codeword and the asymptotic rate  $R_\infty$ . The number of CBs in the terminated codeword is

$$L = \frac{n_m + n_z}{n - 1}. \quad (2.21)$$

Since the zero-tail bits are known at the receiver, they are not transmitted. Only the parity bits generated from the zero-tail bits are transmitted. Hence, the actual rate of the terminated BPL code is

$$R_L = \frac{n_m}{Ln - n_z}. \quad (2.22)$$

The relation between  $R_L$  and  $R_\infty$  is given by rearranging (2.22)

$$R_L = \nu R_\infty \quad (2.23)$$

where  $\nu = \frac{Ln}{Ln - n_z}$  and  $R_\infty = \frac{n_m}{Ln} = \frac{k}{n}$ . Hence,

$$\lim_{L \rightarrow \infty} R_L = R_\infty. \quad (2.24)$$

Encoding and termination of BPL codes is explained in detail in Chapter 3.

---

## 2.3 Decoding of Low-Density Parity-Check Codes

---

A channel decoder attempts to find the transmitted codeword  $\mathbf{x}$  from the received word  $\mathbf{y}$ . The best decoder in terms of performance is a Maximum-A-Posteriori (MAP) decoder. Its complexity grows exponentially with the information word length because it finds among all possible codewords the codeword that has the highest probability given the received word. The estimate of the transmitted codeword from a MAP decoder is given by

$$\begin{aligned}
 \hat{\mathbf{x}} &= \underset{\mathbf{x}_i}{\operatorname{argmax}} p_{X|Y}(\mathbf{x}_i, \mathbf{y}) \\
 &= \underset{\mathbf{x}_i}{\operatorname{argmax}} \frac{p_{Y|X}(\mathbf{y}, \mathbf{x}_i) P(\mathbf{x}_i)}{P(\mathbf{y})} \\
 &= \underset{\mathbf{x}_i}{\operatorname{argmax}} p_{Y|X}(\mathbf{y}, \mathbf{x}_i) P(\mathbf{x}_i)
 \end{aligned} \tag{2.25}$$

where  $\mathbf{x}_i$  is a codeword from the set of all codewords,  $\mathbf{y}$  is the received word,  $Y$  and  $X$  are random variables representing received word and transmitted codeword respectively. For equiprobable codewords  $\mathbf{x}_i$ , a MAP decoder is equivalent to a Maximum-Likelihood (ML) decoder.

Due to the high complexity of MAP decoders, LDPC codes are usually decoded using iterative Message-Passing Algorithm.

---

### 2.3.1 Belief Propagation

---

The Message-Passing Algorithm (MPA) uses the BP technique [10] to compute the *a-posteriori* probability of the bits in the transmitted codeword given the received word in an iterative fashion. The idea behind belief propagation is exchanging uncertainties between the bits which are connected as defined by the encoder or PCM. Refer to Section 3.1 to see how different bits in the codeword are dependent on each other. The algorithm uses LLRs instead of APPs as in (2.25) for numerical stability. The LLR values given by the channel for the received bits are

$$\mathcal{L}(y_i) = \log \frac{P(X_i = 1 | \mathbf{y})}{P(X_i = 0 | \mathbf{y})} \tag{2.26}$$

where  $i = 0, \dots, n-1$  is the index of bits in the codeword.

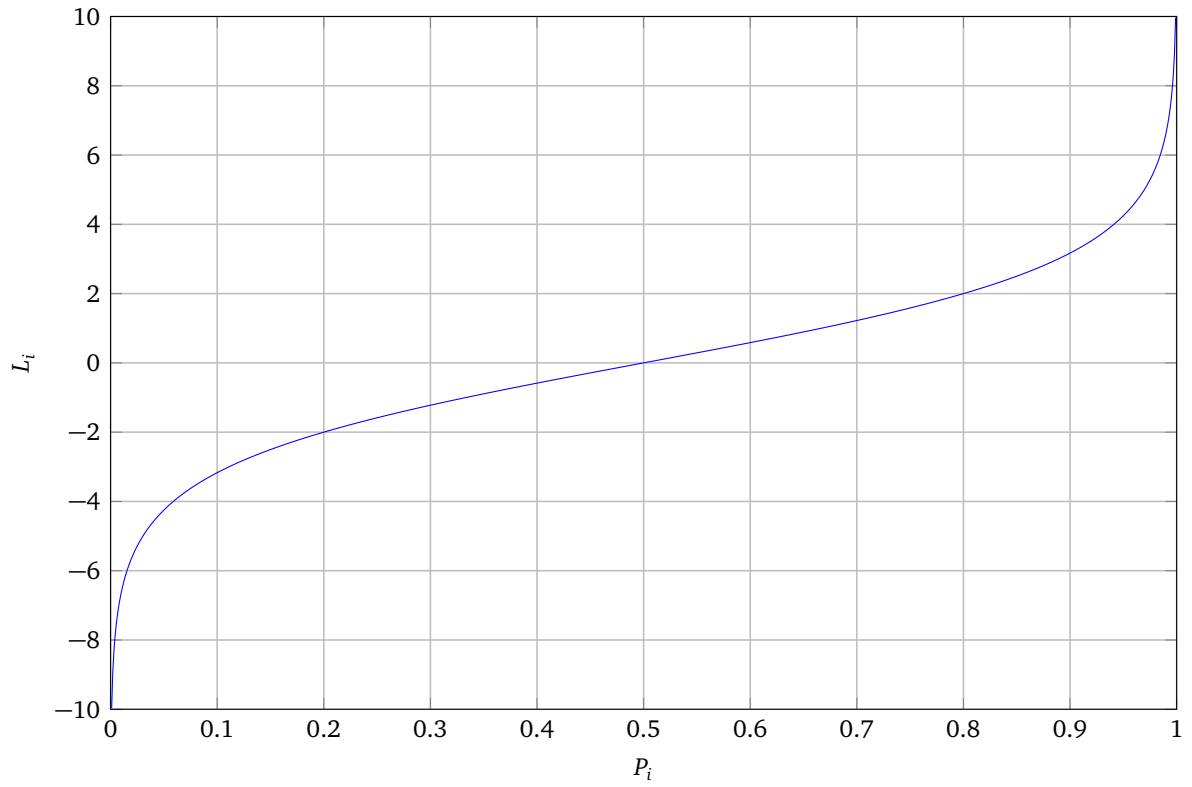
Figure 2.5 shows the relation between  $P(X_i = 0 | \mathbf{y})$  and  $\mathcal{L}(y_i)$ . The APP values range from 0 to 1 while the LLR takes values ranging from  $-\infty$  to  $+\infty$  which makes calculation of messages easier.

In a single iteration of the algorithm, the LLRs of each bits in the codeword are updated through two intermediate message computations: variable-node-to-check-node (V2C) message and check-node-to-variable-node (C2V) message.

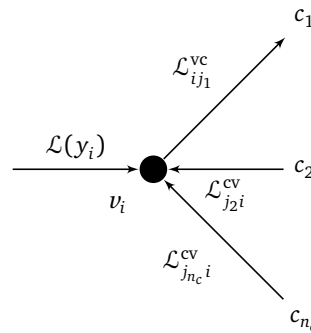
- V2C message: Each VN passes its LLRs on to its neighboring nodes (neighboring nodes are the CNs to which the VN is connected in the Tanner graph). These LLRs contain only extrinsic information from all other CNs in the previous iteration. The expression for the V2C message is given by [10]

$$\mathcal{L}_{ij}^{\text{vc}} = \mathcal{L}(y_i) + \sum_{j' \in \mathcal{E}_v(i) \setminus j} \mathcal{L}_{ji'}^{\text{cv}} \tag{2.27}$$

where  $\mathcal{L}_{ij}^{\text{vc}}$  is the V2C message from the  $i$ -th VN to the  $j$ -th CN,  $\mathcal{L}_{ji}^{\text{cv}}$  is the C2V message from the  $j$ -th CN to the  $i$ -th VN in the previous iteration and  $\mathcal{E}_v(i)$  is the set containing all  $n_c$  CNs connected to the  $i$ -th VN.



**Figure 2.5:** Relation between APP and LLR.

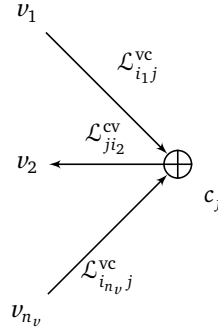


**Figure 2.6:** Example showing that messages from all other CNs sum-up with the channel's LLR to form the V2C message to the first CN.

- C2V message: Each CN processes the received V2C messages and computes extrinsic information for its neighboring VNs. These extrinsic informations contain V2C messages from VNs other than the destination VN. The expression for C2V messages is given by [10]

$$\mathcal{L}_{ji}^{cv} = 2 \operatorname{atanh} \left( \prod_{i' \in \mathcal{E}_c(j) \setminus i} \tanh \left( \frac{\mathcal{L}_{i'j}^{vc}}{2} \right) \right) \quad (2.28)$$

where  $\mathcal{E}_c(j)$  is the set containing all  $n_v$  VNs connected to the  $j$ -th CN.

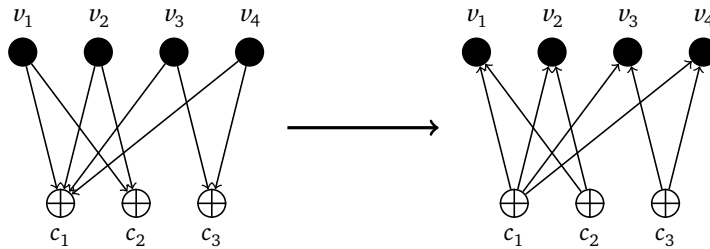


**Figure 2.7:** Example showing that messages from all other VNs combine using equation (2.28) to form the C2V message to the second VN.

The process of sending a V2C message, receiving a C2V message from the same edge, and summing it up with the current LLR is termed an *edge update*. The above steps indicate the BP technique. It is also called sum-product algorithm (SPA).

The high complexity C2V message computation can be approximated by a low-complexity computation called the *Min-Sum Algorithm*. The Min-Sum Algorithm (MSA) version of the expression in (2.28) is given by

$$\mathcal{L}_{ji}^{cv} \approx \left( \prod_{i' \in \mathcal{E}_c(j) \setminus i} \operatorname{sign}(\mathcal{L}_{i'j}^{vc}) \right) \cdot \min_{i'} |\mathcal{L}_{i'j}^{vc}|. \quad (2.29)$$

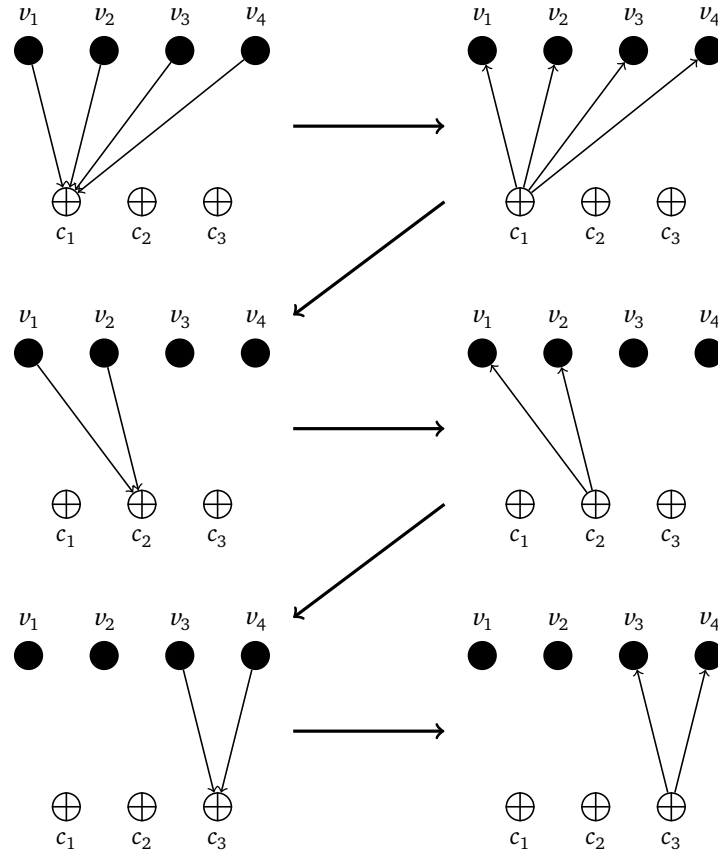


**Figure 2.8:** Illustration of parallel scheduling with an example Tanner graph.

One can perform the edge updates in different sequences. The sequencing of edge updates is termed as scheduling. Two such scheduling methods are *parallel scheduling* and *serial scheduling*. In parallel scheduling which is also referred to as *flooding*, the VNs send V2C messages to all CNs at once and then C2V messages are computed and passed to all VNs. In Figure 2.8, the left graph indicates V2C message passing and the right graph indicated C2V message passing. In serial scheduling [11], the VNs are updated in a CN-by-CN or row-by-row (in the PCM) manner. Each CN processes its incoming V2C

messages and sends the corresponding C2V messages to its neighboring VNs. This is called a *row update* or a *layer update*. Figure 2.9 shows how message passing is performed in serial scheduling. Parallel scheduling is much faster on a parallel computing platform because the edge updates are independent to each other. But compared to its serial counterpart, parallel scheduling lacks performance because the edges are updated in parallel and only the original messages from the VNs are used. Serial scheduling yields better performance than parallel scheduling as the CNs that are being processed later will have updated LLRs from the VNs. In a conventional BP decoder with any type of scheduling, the entire message passing is repeated for a maximum of  $I$  iterations.

With serial scheduling, different orders of CN processing results in decoding performance changes. Figure 2.9 illustrates serial scheduling with CN processing from left to right i.e., top to bottom in the PCM. For irregular codes, a good choice is to start processing the CN that has the lowest VN degree and move to higher VN degree CNs.



**Figure 2.9:** Illustration of serial scheduling with the same Tanner graph as in Figure 2.8. The processing starts from the left most CN and moves right to the last CN.

In our implementation, we use an improved MSA as proposed in [12] which is a combination of (2.28) and (2.29). We do serial scheduling with layer updates starting from top to bottom. We also do a bottom to top layer update which is discussed in Chapter 4.

### 2.3.2 LDPC-CC-specific Decoding Techniques

The conventional BP-based block decoder can be used to decode any LDPC-BC or terminated LDPC-CC in which the BP is performed throughout the whole Tanner graph at once. For LDPC-CCs, the convolutional structure imposes a constraint on the VNs: two VNs of the PCM that are at least  $(m_s + 1)n$  columns apart cannot be involved in the same parity-check equation. This characteristic can be exploited to per-

form iterative BP decoding only to a part of the codeword at a time. Two of such decoding techniques are *pipeline* and *window* decoding.

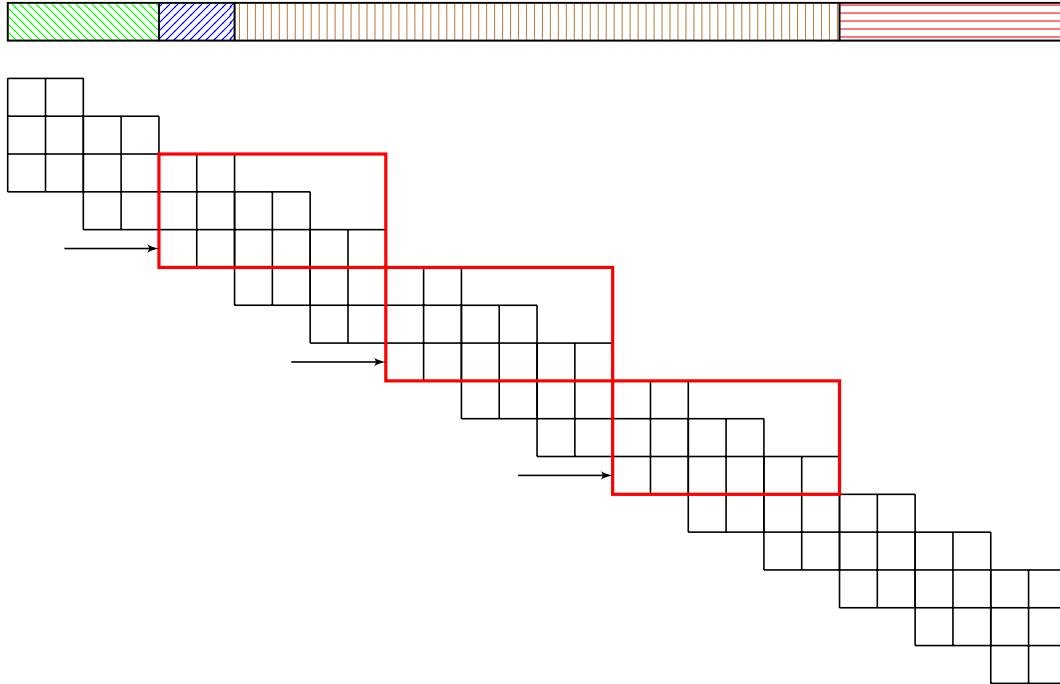
### Pipeline Decoder

A decoding technique that was proved to be efficient for LDPC-CCs is the *pipeline decoding* introduced in [8]. The pipeline decoder employs  $I$  parallel processing units. Each processing unit covers  $l_c = (m_s + 1)n$  VNs, so that during a single decoding iteration the messages are only passed to other VNs within the same processing unit. Hence, all the processing units cover a total of  $Il_c$  VNs. In each time instance,  $n$  VNs and  $n - k$  VNs enter the rightmost processor and  $n$  estimated VNs leave from the leftmost processor. Hence, at the end of  $I$  time instances, all CNs are processed  $I$  times. Figure 2.10 illustrates pipeline decoding of a code with  $R_\infty = 1/2$ ,  $m_s = 3$  and  $I = 3$ . The codeword is indicated by the rectangular bar above the PCM. The red windows are the three processing units. The green (backhatched) part of the codeword indicates the estimated VNs and the blue (hatched) part indicates the next available estimated VNs. The brown (vertically hatched) and red (horizontally hatched) regions indicate the VNs that are currently being processed and VNs that are yet to be processed, respectively. The arrows indicate the CNs that are processed at the current time instance.

The decoding rate (the output rate of the decoder) of a pipeline decoder is given by

$$\kappa_{pd} = \frac{n}{\psi} \text{bits/second} \quad (2.30)$$

where  $\psi$  is the time taken in seconds to perform one CN processing.



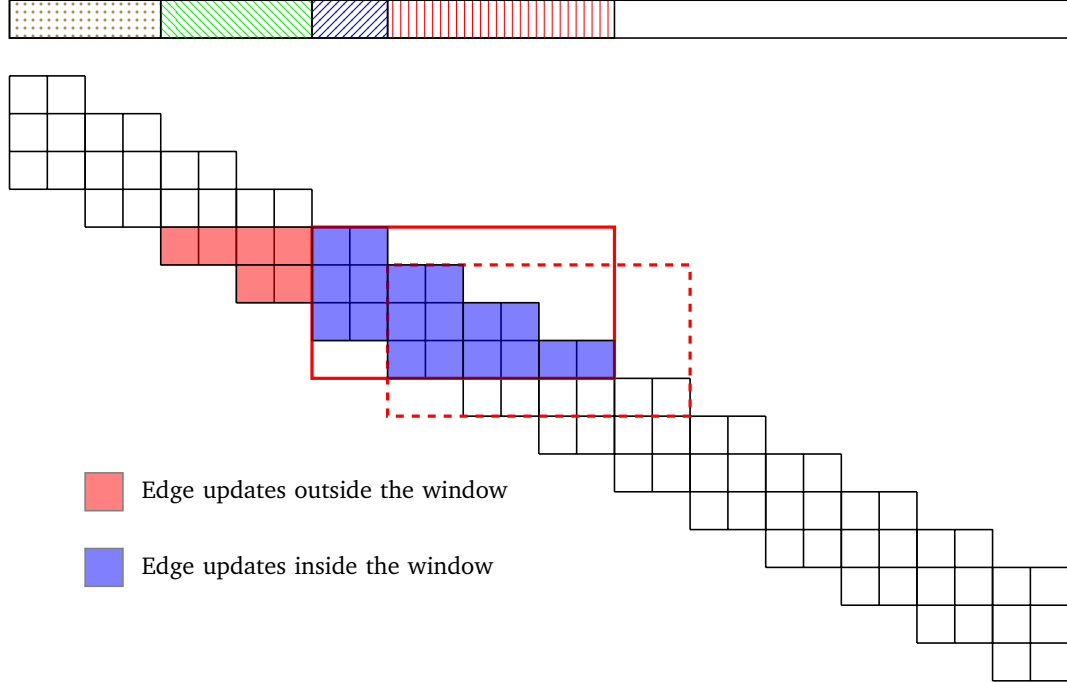
**Figure 2.10:** PCM illustrating the pipeline decoding technique for code with  $R_\infty = 1/2$ ,  $m_s = 2$  and  $I = 3$ . The arrows indicate the CNs that are processed at current time instance.

### Window Decoder

Figure 2.11 illustrates a window decoder with window size of  $W = 4$  and window position  $\rho = 5$ . The solid red box is the current window instance. The blue hatched region of the codeword is the *target VNs*,



the red vertically hatched region is rest of the VNs that are updated inside the current window. The blue colored edges in the PCM are the edges that are updated during the current window instance. The red colored edges are outside the window but they are still updated since their corresponding CNs are inside the window. The green backhatched region of the codeword are the VNs that receive updates from the red colored edges.

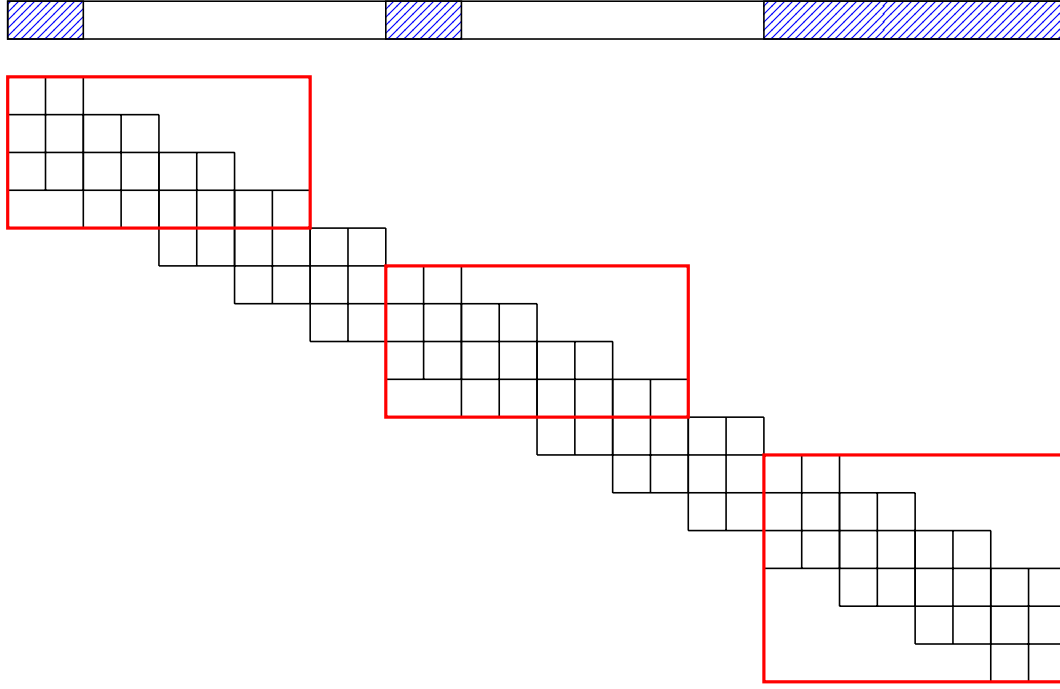


**Figure 2.11:** PCM illustrating the Windowed Decoding technique for  $R_\infty = 1/2$  codes and  $m_s = 2$ . The current window position  $\rho = 5$  is indicated by the solid red box and the next window position is indicated by the dashed red box.

One may choose to not update the edges that are outside the window. This is to prevent the correctly estimated LLR values of the VNs from updating to incorrect values that might propagate from inside the window. Another option might be to only read the LLRs values from VNs connected to the red edges. This prevents the estimated LLR of the VNs from updating to incorrect values while having the advantage of using their LLRs to update the VNs inside the window. In this thesis, we choose to update the VNs in the green backhatched region. This has an advantage of increasing the magnitude of their LLRs which in turn has a positive influence on the VNs in the next window instances. Also, the VNs in the left of the window are more reliable than in the right of the window. Hence, the probability of LLR of the VNs in the green backhatched region flipping their sign is low. This motivates us to choose this window configuration. The brown dotted region of the codeword is the VNs that are already estimated and no longer receive updates from further decoding. The red dashed box indicates the next window instance.

A window must include  $Wn$  VNs and  $W(n-k)$  CNs. Hence, the size of the window should be  $W \geq (m_s + 1)$  because smaller window sizes will not include at least one full CN with all its edges. At each window instance, the first  $n$  VNs are considered to be the target nodes. However, all VNs connected to the  $W(n-k)$  CNs are updated, only the correctness of target nodes are considered as the criteria for moving on to the next window. The BP decoding is performed within the current window for a maximum number of  $I$  iterations or until the parity-check equations involving the target nodes are fulfilled (whichever occurs earlier). Then the window shifts forward such that the next  $n$  VNs become the target nodes. The process continues until all VNs in the received word are decoded.

With the window decoder, there are several ways to perform the decoding when the window reaches the rightmost end of the PCM. The conventional way is to keep moving the window until the last  $n$  VNs



**Figure 2.12:** Target VNs for first, middle and the last window positions.

are the target VNs. Another method is when the window touches the rightmost column of the PCM, the window extends its height to include the remaining  $m_s$  CNs of the PCM. Also, all the VNs inside the last window are considered as target VNs. The last window scenario is illustrated in Figure 2.12. We choose the second method as it reduces the number of iterations compared to the conventional method. The lower CN degree in the end of the PCM compensates to the reduced decoding performance for having fewer iterations.

The decoding rate in bits of a window decoder is given by

$$\kappa_{\text{wd}} = \frac{n}{\psi \cdot I} \text{bits/second.} \quad (2.31)$$

The main benefit of window decoding over full-block decoding is that the memory requirements are reduced because at any instance the BP is performed on a smaller number of VNs rather than the whole graph. For non-packet-wise transmissions, the decoded bits are sent to higher layer for processing rather than waiting for the whole codeword to be decoded as in a block decoder. However, these benefits come with a cost of reduced performance since the BP is limited to fewer VNs and CNs.

In this thesis, we choose window decoding over pipeline decoding because implementing parallel processing units in software is infeasible.

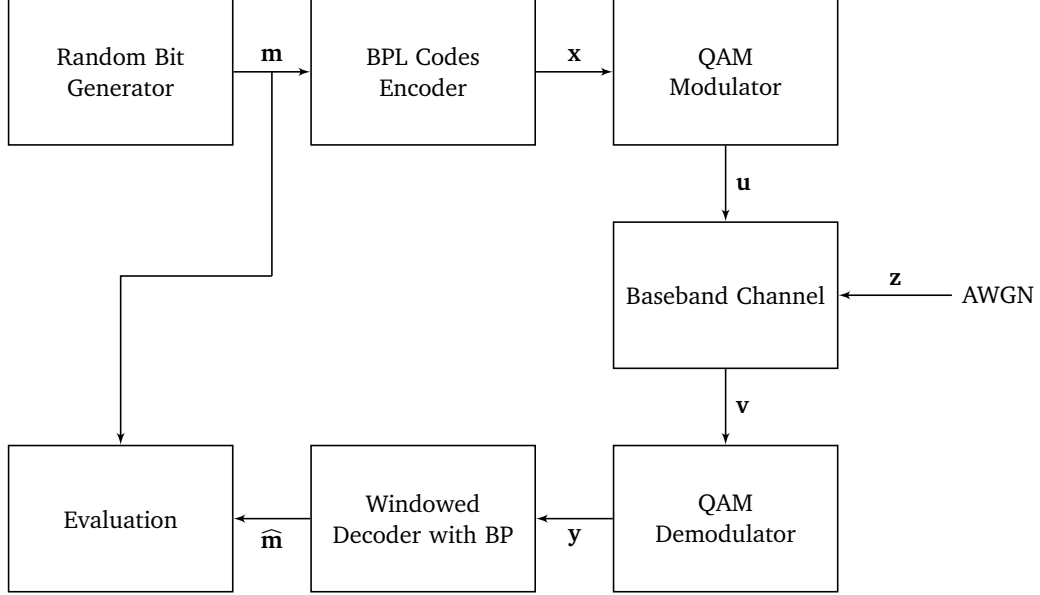
---

## 2.4 System Model

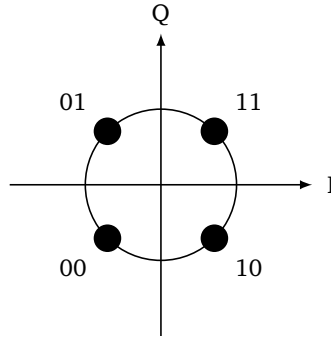
---

In this thesis, we consider a digital baseband system model for simulations as shown in Figure 2.13. The information bits are generated using a random-number generator with uniform distribution. The sequence of bits  $\mathbf{m}$  of the generated random bit is encoded using the *BPL Codes Encoder* block. The details of this block are discussed in Chapter 3. The *Quadrature Amplitude Modulation (QAM) Modulator* block receives the codewords  $\mathbf{x} \in \mathbb{F}_2^{Ln \times 1}$  from the encoder and maps them to complex-valued symbols depending on the chosen modulation scheme. The output of the modulator is a vector of symbols given by  $\mathbf{u} \in \mathbb{C}^{\frac{Ln}{o} \times 1}$  where  $o$  is the order of modulation and lets assume that  $Ln = go, g \in \mathbb{Z}_+$ . Figure 2.14

shows a QPSK symbol constellation. It is seen that the adjacent symbols differ by only one bit. This is called *Gray Mapping* [5]. In all our simulations we use QPSK with gray mapping and no interleaving. Hence,  $\mathbf{u}_i, i = 0, \dots, \frac{Ln}{o} - 1 \in \{e^{j\frac{\pi}{4}}, e^{j\frac{3\pi}{4}}, e^{-j\frac{3\pi}{4}}, e^{-j\frac{\pi}{4}}\}$ .



**Figure 2.13:** System model for simulations.

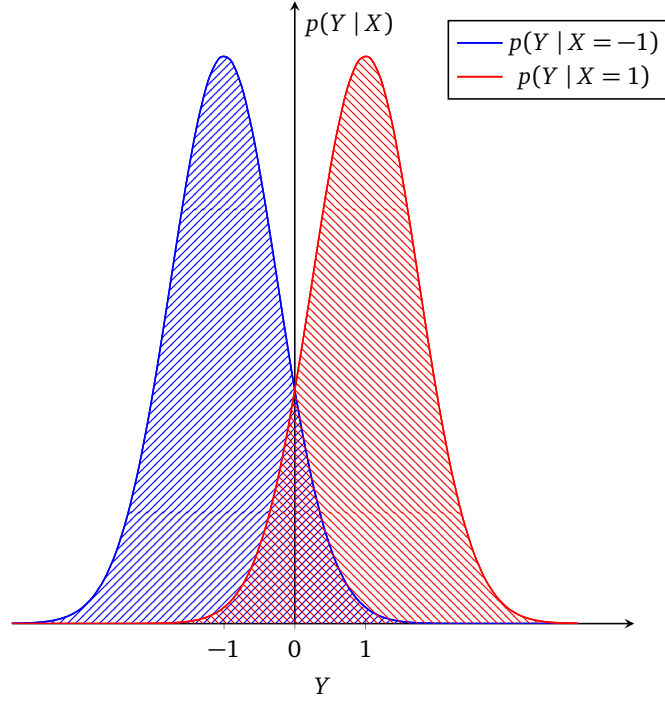


**Figure 2.14:** Symbol constellation of QPSK modulation with gray mapping.

The channel model we consider is a simple AWGN channel with no fading or multi-path components. So the received symbols are given by

$$\mathbf{v} = \mathbf{u} + \mathbf{z} \quad (2.32)$$

where  $\mathbf{z} = \mathcal{CN}(0, 2\sigma^2) \in \mathbb{C}^{\frac{Ln}{o} \times 1}$  is a complex random variable of Gaussian distribution with zero mean and variance  $2\sigma^2$ . Note that only one symbol in  $\mathbf{u}$  is transmitted per channel use. The received symbol vector is  $\mathbf{v} \in \mathbb{C}^{\frac{Ln}{o} \times 1}$ . The output of the *QAM Demodulator* is a vector of LLRs  $\mathbf{y} \in \mathbb{R}^{Ln \times 1}$  corresponding to the bits in  $\mathbf{x}$ . The LLRs are computed using the symbols in  $\mathbf{v}$  and the modulation order. Each QPSK symbol is a combination of two BPSK symbols that are orthogonal to each other. Hence, the LLRs of both



**Figure 2.15:** Conditional probability density functions of a BPSK symbol over AWGN channel.

the bits can be computed separately. The LLR of the first bit of each received QPSK symbol  $y$  is computed as

$$\mathcal{L}_{\text{first}} = \log \frac{P(X_1 = 0 | \text{Re}(y))}{P(X_1 = 1 | \text{Re}(y))} \quad (2.33)$$

and of the second bit as

$$\mathcal{L}_{\text{second}} = \log \frac{P(X_2 = 0 | \text{Im}(y))}{P(X_2 = 1 | \text{Im}(y))}. \quad (2.34)$$

Using Bayes' Theorem and assuming that the bits in  $\mathbf{x}$  are equiprobable, we have

$$\mathcal{L}_{\text{first}} = \log \frac{p_{\text{Re}(Y)|X}(\text{Re}(y), X_1 = 0)}{p_{\text{Re}(Y)|X}(\text{Re}(y), X_1 = 1)}, \quad (2.35)$$

$$\mathcal{L}_{\text{second}} = \log \frac{p_{\text{Im}(Y)|X}(\text{Im}(y), X_2 = 0)}{p_{\text{Im}(Y)|X}(\text{Im}(y), X_2 = 1)}. \quad (2.36)$$

On substituting the conditional Probability Density Functions (PDFs) of BPSK symbols as shown in Figure 2.15, we get

$$\mathcal{L}_{\text{first}} = \log \frac{e^{-\frac{(\text{Re}(y)-1)^2}{2\sigma^2}}}{e^{-\frac{(\text{Re}(y)+1)^2}{2\sigma^2}}} \quad (2.37)$$

$$= \frac{2}{\sigma^2} \text{Re}(y). \quad (2.38)$$

Similarly,

$$\mathcal{L}_{\text{second}} = \frac{2}{\sigma^2} \text{Im}(y). \quad (2.39)$$

Note that the symbol values in Figure 2.15 are  $X_i \in \{-1, +1\}$ ,  $i = 1, 2$  which corresponds to information bits 1 and 0 respectively. The noise variances of the real and imaginary parts of the complex random variable  $Z$  are each  $\sigma^2$  from

$$\text{E}\{|Z|^2\} = \text{E}\{|\text{Re}(Z)|^2\} + \text{E}\{|\text{Im}(Z)|^2\}. \quad (2.40)$$

The SNR  $\zeta$  of the QPSK signal is

$$\frac{E_s}{N_0} = 2 \frac{E_b}{N_0} \quad (2.41)$$

where  $E_s$  is the energy per QPSK symbol,  $E_b$  is the energy per bit and  $N_0$  is the AWGN power spectral density. The SNR of each bit is equivalent to the SNR of BPSK signal and is given by

$$\frac{E_b}{N_0} = \frac{\text{E}\{|X_1|^2\}}{\text{E}\{|\text{Re}(Z)|^2\}} \quad (2.42)$$

$$= \frac{\text{E}\{|X_2|^2\}}{\text{E}\{|\text{Im}(Z)|^2\}} \quad (2.43)$$

$$= \frac{1}{\sigma^2}. \quad (2.44)$$

Hence, the LLRs can be written as

$$\mathcal{L}_{\text{first}} = 2 \frac{E_b}{N_0} \text{Re}(y), \quad (2.45)$$

$$\mathcal{L}_{\text{second}} = 2 \frac{E_b}{N_0} \text{Im}(y). \quad (2.46)$$

After the LLRs are computed, they are decoded using the BP and Windowed Decoding (WD) techniques used inside the *Window Decoder with BP* block. The decoding-improvement techniques that are used in this block are discussed in Chapter 4. The estimated bits  $\hat{\mathbf{m}}$  are then compared with the output of the random bit generator  $\mathbf{m}$  to calculate the BER and BLER. BER  $P_b$  is the ratio between the number of error-bits and the total number of bits transmitted. The probability of error for each bit in the codeword is given by

$$P_b(i) = P\{\hat{X}_i \neq X_i\} \quad (2.47)$$

$$(2.48)$$

and the overall probability of bit error is give by

$$P_b = \frac{1}{Ln} \sum_{i=0}^{Ln-1} P_b(i) \quad (2.49)$$

$$(2.50)$$

BLER  $P_L$  is the ratio between the number of error-blocks and the total number of blocks transmitted. A block is considered to be an error-block if at least one information bit is incorrect. The BLER is given by

$$P_L = P\{\widehat{\mathbf{m}} \neq \mathbf{m}\}. \quad (2.51)$$

With BER and BLER being the metrics for measuring decoding performance, ANEU is the metric for measuring decoding complexity. ANEU  $\eta$  is given by

$$\eta = \frac{1}{\eta_{\max}} \sum_{\rho=0}^{\xi-1} \eta_{\rho} I_{\rho} \quad (2.52)$$

$$(2.53)$$

where  $\xi$  is the total number of window positions,  $\eta_{\rho}$  is the number of edges inside and  $I_{\rho}$  is the number of iterations performed in  $\rho$ -th window position.  $\eta_{\max}$  is the maximum number of edge updates possible and is given by

$$\eta_{\max} = I \sum_{\rho=0}^{\xi-1} \eta_{\rho}. \quad (2.54)$$

## 3 Encoding of the Broadband Power Line Codes

In this chapter, we discuss how the encoder for BPL codes is designed and how termination is handled. In some places of this section, we use examples of LDPC-CCs with small coupling length  $L$  and syndrome-former memory  $m_s$  as the BPL codes are too large to be represented on paper.

### 3.1 Encoder Design

Encoding algorithms are not as complex as decoding algorithm because decoders need to correct the incorrect bits in the codeword. The encoder for any Convolutional code or LDPC-CC only needs to generate the parity bits from the information bits and previously generated parity bits. From equation (2.16), the parity-check term of BPL codes is given by

$$\sum_{i=0}^{m_s} \mathbf{h}_{M,i} \mathbf{m}(t-i) + \sum_{i=0}^{m_s} h_{B,i} b(t-i) \mod 2 \quad (3.1)$$

where  $\mathbf{m}(t) \in \mathbb{F}_2^{k \times 1}$  is a vector of message bits at  $t$ -th time instance or CB,  $h_{B,i} \in \mathbb{F}_2$  is the coefficient of polynomial of the parity bit.  $\mathbf{h}_{M,i} \in \mathbb{F}_2^{1 \times k}$  is a vector of coefficients of polynomials of message bits and is given by (3.3). The encoder generates one parity bit per  $n - 1$  message bits and the expression for generating the parity bit is given by rearranging (3.1) as

$$b(t) = \sum_{i=0}^{m_s} \mathbf{h}_{M,i} \mathbf{m}(t-i) + \sum_{i=1}^{m_s} h_{B,i} b(t-i) \mod 2, \quad (3.2)$$

where  $b(t) \in \mathbb{F}_2$  is the parity bit.

$$\mathbf{h}_{M,i} = \begin{bmatrix} [D^i]A_{1,\tau} & \dots & [D^i]A_{k,\tau} \end{bmatrix}, \quad (3.3)$$

where  $[D^i]A_{1,\tau}$  represents the coefficient of  $D^i$  in polynomial  $A_{1,\tau}(D)$ .

The resulting codewords  $\mathbf{x} \in \mathbb{F}_2^{Ln \times 1}$  have a systematic structure given by

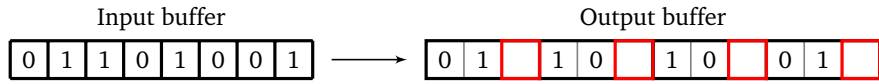
$$\mathbf{x}^T = \begin{bmatrix} \mathbf{x}(0)^T & \mathbf{x}(1)^T & \dots & \mathbf{x}(L-1)^T \end{bmatrix}. \quad (3.4)$$

Similarly, the PCM can be formed using  $\mathbf{h}_{M,i}$  and  $h_{B,i}$  for all  $i = 0, \dots, m_s$ :

$$\mathbf{H} = \begin{bmatrix} [\mathbf{h}_{M,0}, h_{B,0}] & & & \\ [\mathbf{h}_{M,1}, h_{B,1}] & [\mathbf{h}_{M,0}, h_{B,0}] & & \\ \vdots & [\mathbf{h}_{M,1}, h_{B,1}] & \ddots & \\ [\mathbf{h}_{M,m_s}, h_{B,m_s}] & \vdots & \ddots & [\mathbf{h}_{M,0}, h_{B,0}] \\ & [\mathbf{h}_{M,m_s}, h_{B,m_s}] & \ddots & [\mathbf{h}_{M,1}, h_{B,1}] \\ & & \ddots & \vdots \\ & & & [\mathbf{h}_{M,m_s}, h_{B,m_s}] \end{bmatrix}. \quad (3.5)$$

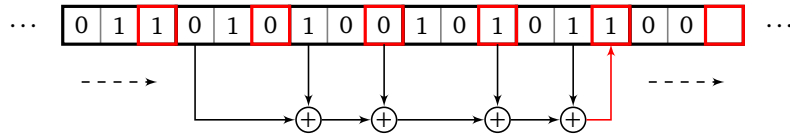
Encoding of BPL codes is done by generating parity bits as per (3.2). This is performed through the following steps:

1. The information bits are divided into several CBs and copied into the output buffer of the encoder. An example of this step for a code with  $R_\infty = 2/3$  is shown in Figure 3.1.



**Figure 3.1:** Illustration of input and output buffers and its contents. The red boxes represents the position of the parity bits.

2. The parity bits are generated by performing addition modulo 2 with the bits (both information and parity bits) in the output buffer according (3.2). An example of the parity-bit generation of a code with  $R_\infty = 2/3$  is shown in Figure 3.2.



**Figure 3.2:** Illustration of a parity bit being generated using information and other parity bits in the output buffer. The arrow indicates the direction of movement of the parity-bit generator. Note: An example code.

3. The termination sequence is appended to the output buffer of the encoder. The procedure to generate the termination sequence is discussed in Section 3.2.

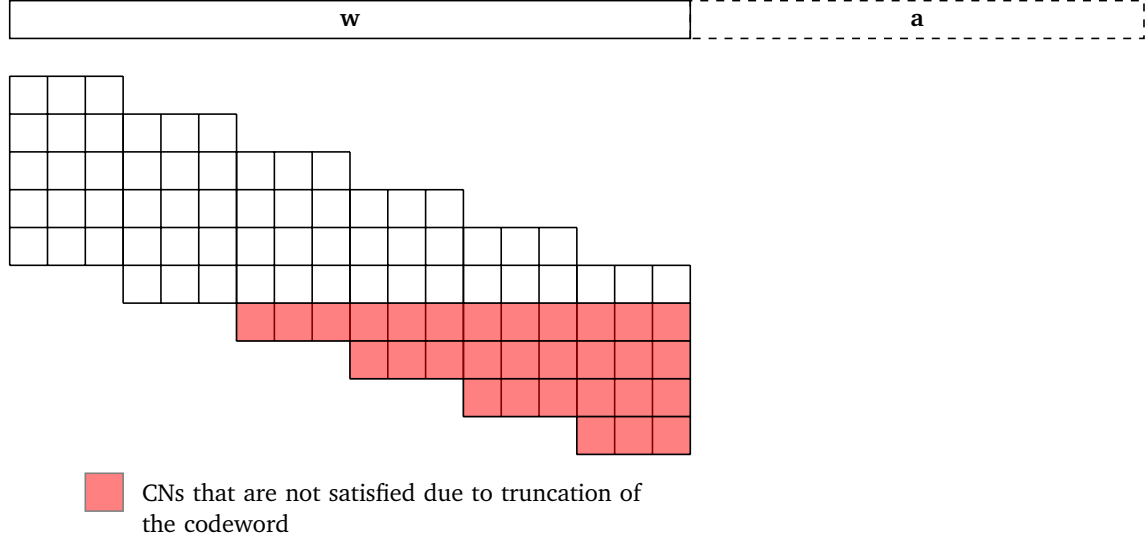
### 3.2 Termination Sequence

The objective of terminating any LDPC-CC is to introduce some low degree CNs in the PCM. Termination essentially means truncating the PCM to have a finite number of rows and columns. Truncating the PCM leads to a condition where the last  $m_s$  bits of the codeword do not satisfy the last  $m_s$  parity-check equations of the PCM. To satisfy the last  $m_s$  parity-check equations, either the last  $m_s$  or lesser bits need to be modified or an appropriate termination sequence  $\mathbf{a}$  must be appended to the codeword. Since the last  $m_s$  bits of the codeword should not be modified as they contain information, the latter approach is used. The appending of termination sequence is illustrated in Figure 3.3 with a PCM that has a structure similar to (3.5).



### 3.2.1 Proper Termination

Since the generator polynomials of the BPL codes are recursive, continuously feeding in zero-bits into the encoder after the information bits will not reset the states of the encoder. i.e., the output of the encoder will never be all zeros. Hence, one must solve a system of linear equations to find a proper termination sequence that brings the encoder to all-zero state.



**Figure 3.3:** PCM illustrating that a truncated codeword does not satisfy all CNs in a  $R_\infty = 2/3$  code with  $m_s = 4$ .

The procedure for determining a termination sequence is reproduced here from [13]. Figure 3.4 illustrates the different matrices and vectors that are used in determining the termination sequence.  $\mathbf{e} \in \mathbb{F}_2^{m_s n \times 1}$  is the last part of the codeword vector, vector  $\mathbf{a} \in \mathbb{F}_2^{l_t \times 1}$  is the termination sequence that should be determined and appended to the end of  $\mathbf{e}$ . The sub-matrix  $\mathbf{P}$  of the PCM contains the last  $m_s$  CNs of the actual PCM. The sub-matrix  $\mathbf{D}$  (containing the blue edges) is an extension to the actual PCM due to the termination sequence  $\mathbf{a}$ . The codeword combined with a proper termination sequence must fulfill all the parity-check rows in matrix  $[\mathbf{P}, \mathbf{D}]$ . Hence, provided a correct termination sequence  $\mathbf{a}$ , the following equations from [13] hold true.

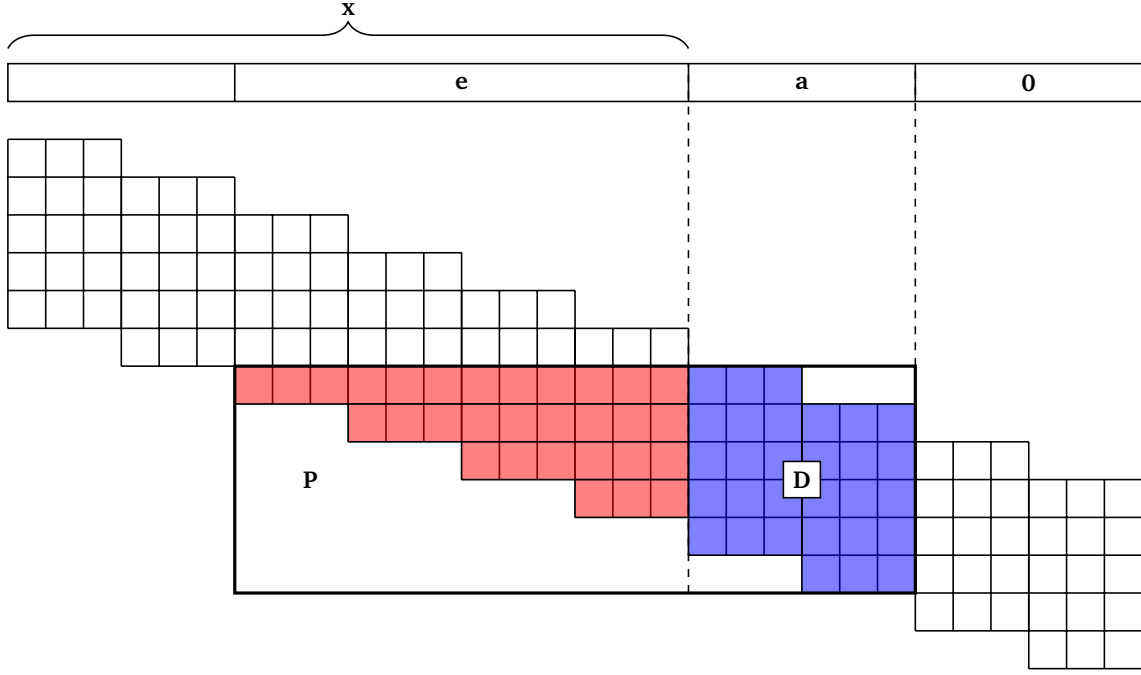
$$[\mathbf{P} \quad \mathbf{D}] \odot \begin{bmatrix} \mathbf{e} \\ \mathbf{a} \end{bmatrix} = \mathbf{0} \quad (3.6)$$

$$\mathbf{P} \odot \mathbf{e} + \mathbf{D} \odot \mathbf{a} = \mathbf{0} \quad (3.7)$$

One possible solution under the constraint that  $\mathbf{D}$  is square and invertible is given by

$$\mathbf{a} = \mathbf{D}^{-1} \odot \mathbf{P} \odot \mathbf{e}. \quad (3.8)$$

The maximum length  $l_t$  of the termination sequence  $\mathbf{a}$  should not exceed the one given in the BPL standard. However, a shorter termination sequence with  $l_t \geq n$  can be chosen such that  $\mathbf{D}$  is a square matrix. Once the termination sequence  $\mathbf{a}$  is determined from (3.8), it is appended to the codeword and the remaining part of the termination sequence (if any) is filled with zeros. These zeros do not contribute to any parity checks. For an encoder that is implemented with feedback shift registers, these zeros are required to bring the encoder to an all-zero state. But, we omit these zeros since we implement the



**Figure 3.4:** PCM of an example LDPC-CC code with  $R_\infty = 2/3$  and  $m_s = 4$  illustrating sub-matrices used to determine the termination sequence. Depicted above the PCM is the codeword vector.

encoder in software. Hence, the resulting PCM and codeword looks like the one shown in Figure 3.4 with no zeros at the end of the codeword and without their corresponding edges in the PCM.

### 3.2.2 Zero-Tail Termination

Unfortunately, we found that a proper termination sequence cannot be computed for BPL codes. It is found through numerical evaluations that  $\mathbf{D}$  is not full-rank and hence its inverse do not exist. A full-rank  $\mathbf{D}$  matrix is possible if  $\mathbf{D} \in \mathbb{F}_2^{r \times c}$  and  $r > c$ . But, with the help of numerical solvers, we found that the solution for such an overdetermined system does not exist either. Hence, a termination sequence to satisfy the last  $m_s$  parity checks cannot be found for BPL codes. So, we choose to omit the last  $m_s$  parity checks from the PCM and do the termination by *zero-tailing* as mentioned in the standard.

The steps for performing zero-tailing termination are as follows.

1. The  $n_z$  zero-tail bits are appended to the input buffer of the encoder after the  $n_m$  information bits.
2. The encoding as mentioned in Section 3.1 is performed using all the information bits and zero-tail bits.

With the zero-tail termination, the zero-tail bits are always known at the receiver. Hence, the zero-tail bits are not transmitted in an actual system. Only the parity bits generated from the zero-tail bits are transmitted and used during decoding.

The zero-tail termination does not satisfy the last  $m_s$  parity checks of the PCM. Hence, the PCM of a zero-tail-terminated BPL codes looks like the one shown in Figure 3.5.

The PCM of the zero-tail-terminated codes has the same CN degrees in the middle and in the end. By contrast, properly terminated codes have lower CN degrees at the end. The lack of low CN degrees reduce the performance of the code as lower CN degree means better reliability of the variable nodes connected to them. However, the zero-tail bits improve the reliability of the connected VNs during the BP decoding process since the zero-tail bits are known at the receiver. The structure of a zero-tail-terminated codeword looks like the one shown in Figure 3.6.



---

We conclude this section by summarizing that proper termination for BPL codes are not feasible because of the nature of the PCMs and hence, we perform zero-tail termination. The important differences between these two types of terminations are listed in Table 3.1 and we see that both terminations have advantages and disadvantages over the other.

---

## 4 Decoding Improvements

In this chapter, we discuss the techniques that are investigated in this thesis to improve the decoding performance. We start with the discussion of existing techniques that improve the decoding performance. We then discuss the techniques investigated in this thesis and possible areas of further research and improvements. We finish this section by discussing the results from our simulations.

---

### 4.1 Existing Techniques and Motivation

Improvement techniques for window decoding of LDPC-CC are being widely discussed in literatures. An attractive technique that we found in the literature is the *Zigzag decoder* proposed in [1]. In a normal window decoder, the window moves from left to right of the PCM. So, the messages pass from left to right of the codeword. The VNs in the left and right ends of the codeword are more reliable than that are in the middle. The Zigzag decoder moves the window from right to left of the PCM within a small part where the VNs are not decoded. This allows the messages to pass from right to left of the codeword and hence influencing the high reliability of the VNs in the right on the VNs in the left.

The Zigzag decoder performs better than the conventional left to right decoder but the decoding complexity in terms of number of iterations are higher. The implementation complexity is also high due to the nature of the decoder. Hence, this motivates us to develop a technique to utilize the high reliability of VNs in right of the codeword while have similar decoding and implementation complexity as the conventional window decoder.

Another improvement technique that we found to be interesting is the *early-success* technique in which the decoding stops before reaching the maximum number of iterations  $I$ . Although its very natural to stop decoding when the target VN are decoded, one should carefully choose an optimum criterion for deciding whether the target VN are decoded. A heuristic choice is to check whether the *target-CN*s i.e., the CNs connected to the target VNs are fulfilled. A new early-success criterion was proposed in [2] based on reliable VNs and is called PSC. They distinguish the VNs inside the window as *complete-VNs* and *incomplete-VNs*. Complete-VNs are VNs whose connected CNs are completely inside the window. The remaining VNs in the window are called the incomplete-VNs. This is illustrated in Figure 4.4. The authors say that the complete-VNs are more reliable than the incomplete ones as they get updates from all their connected CNs. Hence, correctness on only the complete-CNs i.e., the CNs connected to only the complete-VNs are considered as early-success criterion.

Although the PSC rule proposed in [2] reduces the number of iterations with same performance, they are suitable only for small windows where  $W < 2(m_s + 1)$ . For larger window sizes, the number complete-VNs are greater than the number of incomplete-VNs. This also makes the number of complete-CN to be greater than the number of target-CN. So for larger windows, the PSC rule actually increases the number of iterations. This motivates us develop a better early-success criterion that depends on the window size.

---

### 4.2 Base Decoder Configuration

Before discussing the details of the techniques used in the improved decoder, it is essential to define a decoder configuration to which the improvements are made. This allows better understanding of the techniques and evaluation of the simulation results. Let us call the decoder without out improvements as Base Decoder (BD) and the one with our improvements as Improved Decoder (ID). The BD uses the WD technique to decoder the BPL codes. The window slides from the left end of the PCM to the right

end. Within each window, serial scheduling is performed by updating the CNs from top to bottom for a maximum of  $I_{\max}$  iterations.

In the last window instance i.e., when the window touches the right most column of the PCM, all the VNs inside the window are considered as target VNs. Hence, the early success criterion is to check if all CNs inside the window are fulfilled. This idea was proposed in [3] with flooding schedule. We adapt this technique in our BD to minimize the total number to iterations.

An early success technique is used in our BD. This technique prevents the decoder from exhausting all iteration in a window if the target VNs are decoder before reaching the maximum number of iteration. Thus saving computational effort. The criterion for finding the correctness of the target VNs is to check if all the CNs connected to the target VNs fulfill their parity-checks.

We now know that the BPL codes are terminated using the zero-tail termination technique. The zero-tail bits and their positions are always known at the receiver. Hence, the LLRs of these bits are made to  $+\infty$  which indicates that these bits most certainly have the value 0.

The configurations of the BD is summarized below.

1. BP based window decoder and the window moves from left to right.
2. Maximum number of iterations for each window is  $I_{\text{BD}}$ .
3. Serial scheduling with top to bottom CN update order.
4. In the last window, all VNs are considered as target VNs.
5. Target CNs are considered as early-success criterion.

---

### 4.3 Left-Right-Left Windowed Decoding

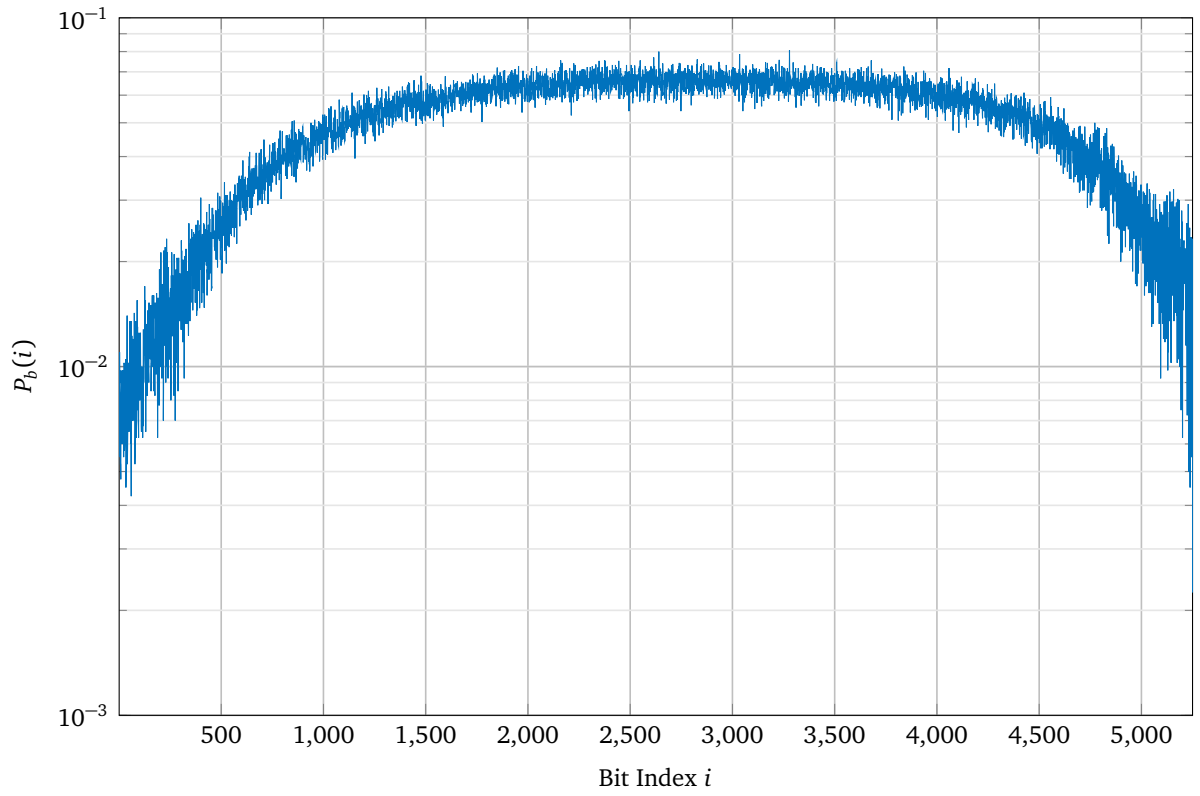
---

In this thesis, we propose the *LRL decoder*. The LRL decoder moves the window from left to right and from right to left of the PCM unlike the BD that moves the window only from left to right. As already mentioned earlier, the VNs in the left and right of the codeword are more reliable than the VNs in the middle due to the lower CN degrees in both the ends. This can be seen in Figure 4.1 which plots the individual probability of error for all the bits in the codeword. We can see that the bits in the left and right of the codeword have lower probability of error than the bits in the middle. During the first half decoding phase of the LRL decoder, the window moves from the first window position to the last window position of the PCM. During the second phase of the decoding i.e., after decoding the last window position, the window moves to the left till it reaches the first window position. This is illustrated in Figure 4.2.

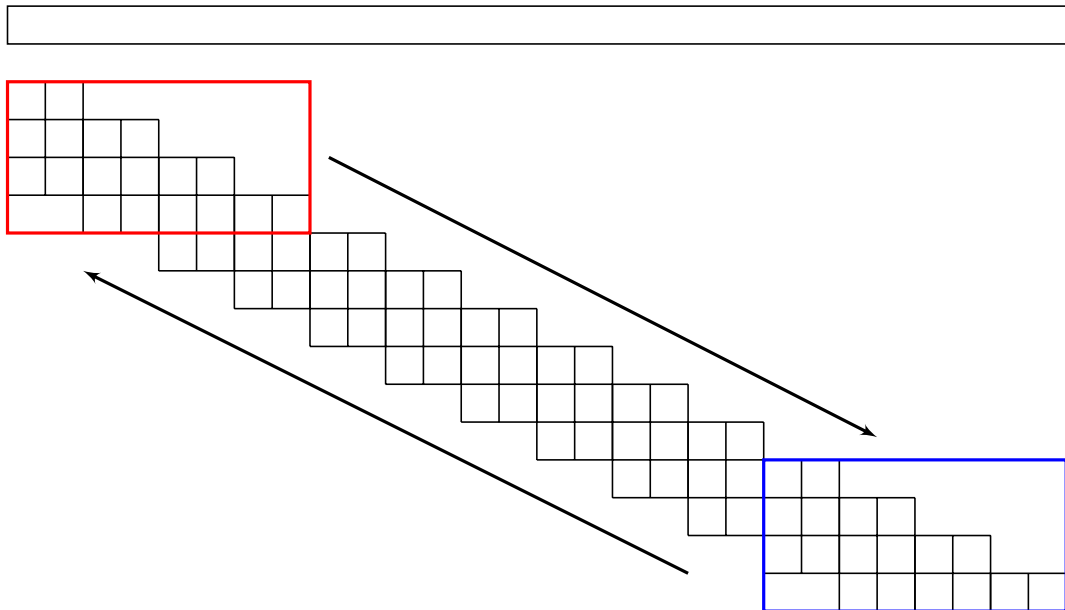
So during the first decoding phase, the VNs in the left increases the reliability of the VNs to the right as the window moves forward. Then during the second decoding phase, the better VNs in the right of the codeword increase the reliability of the VNs to the left as the window moves backwards. The maximum number of iterations within each window is half of the maximum number of iterations in the BD i.e.,  $I_{\text{LRL}} = I_{\text{BD}}/2$ . This is done to maintain the same decoding complexity as the BD because each window in the LRL decoder is decoded twice.

With the LRL decoder, we propose two different configurations of windows. They are illustrated in Figure 4.3. The red box indicates the window. The figure to the left shows a window configuration where the left most VNs (blue hatched) are the target VNs. The order of CN update is from top to bottom as one indicated by the brown arrow. Let us call this LRL configuration-I. The illustration to the right of the Figure 4.3 shows that the right most VNs in the window are the target VNs. The order of CN update is from bottom to top as indicated by the brown arrow. Let us call this LRL configuration-II.

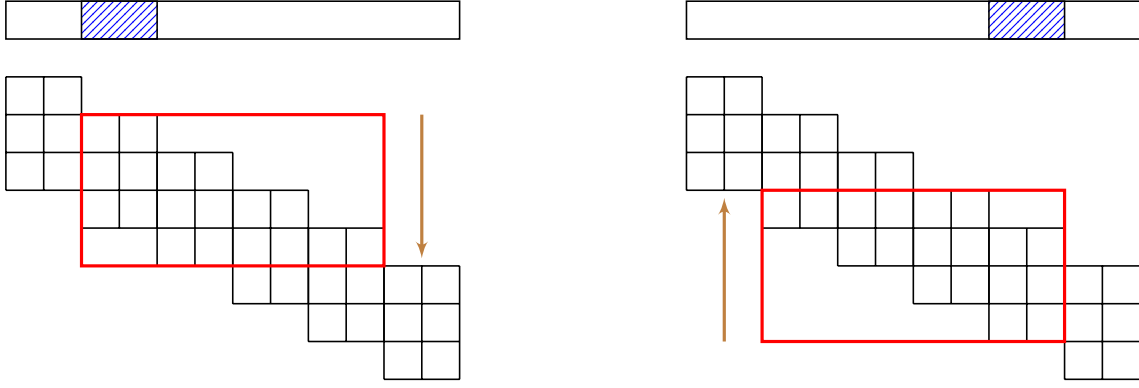
We discuss the performance of the LRL decoder with both configurations. First we use LRL configuration-I in both phases of the LRL decoder. When LRL configuration-I is used in the second phase, the window iterates for more number iterations than when LRL configuration-II is used. This is because the target VN are at the end of the window corresponding to the window direction.



**Figure 4.1:** Probability of error for each bit in the codeword of a code with  $R_\infty = 2/3$ ,  $n_i = 3500$ . Termination bits are excluded.



**Figure 4.2:** PCM illustrating LRL decoder.

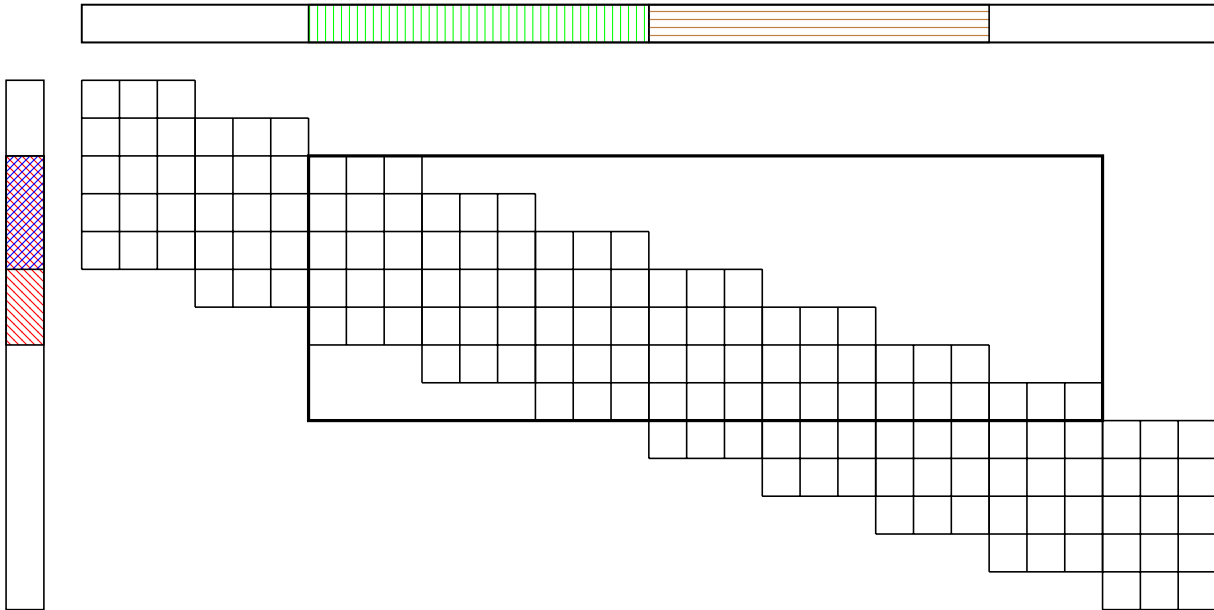


**Figure 4.3:** Different window configurations used in LRL decoder. Left image illustrates LRL configuration-I and right image illustrates LRL configuration-II.

Then we use the LRL configuration-I during the first phase and LRL configuration-II during the second phase. During the second phase the CN updates are done from bottom to top to increase the effect of high reliability of VNs in the right on to the VNs in the middle. The simulation results of both the configurations are evaluated in Chapter 5.

#### 4.4 Improved Partial-Syndrome-Check

The next technique we propose is the IPSC rule. As mentioned earlier, a PSC was proposed in [2] that uses only complete-CN as early-success criterion. In the IPSC rule, we introduce an additional early-success rule for  $W > 2(m_s + 1)$ . That is, when the window size  $W > 2(m_s + 1)$  the target CN are considered as early-success criterion. This could reduce the number of iterations or edge updates because for  $W > 2(m_s + 1)$  the number of complete-CN are greater than the number of target VN and so the window could converge sooner. A window is said to have converged when the target VN are decoded. Table 4.1 summarizes the IPSC technique. The simulation results of the IPSC are evaluated in Chapter 5.



**Figure 4.4:** PCM illustrating complete-VNs, incomplete-VNs and complete-CN.



S.No.	Window Size	Early-success Criterion
1.	$W \leq 2(m_s + 1)$	Complete-VNs.
2.	$W > 2(m_s + 1)$	Target VNs.

**Table 4.1:** Early-success criteria for IPSC



---

## 5 Simulation Results and Evaluation

In this chapter, we analyze the simulation results of our proposed techniques. We start by discussing the simulation setup. Then we analyze the plots to evaluate the performance of our techniques.

---

### 5.1 Experiment Setup

---

For our simulations we used the baseband system model described in section 2.4. The Table 5.1 lists the different parameters of the simulation setup. All the simulations are performed with these parameters unless otherwise specified in the plot captions.

S.No.	Parameter	Value
1.	No. of information bits $n_i$	3500
2.	Asymptotic code rate $R_\infty$	2/3
3.	No. of termination bits $m_t$	380
5.	Modulation	QPSK
6.	Window Size $W$	300, 700
7.	No. of Iterations $I$	10

**Table 5.1:** Experimental settings for simulations.

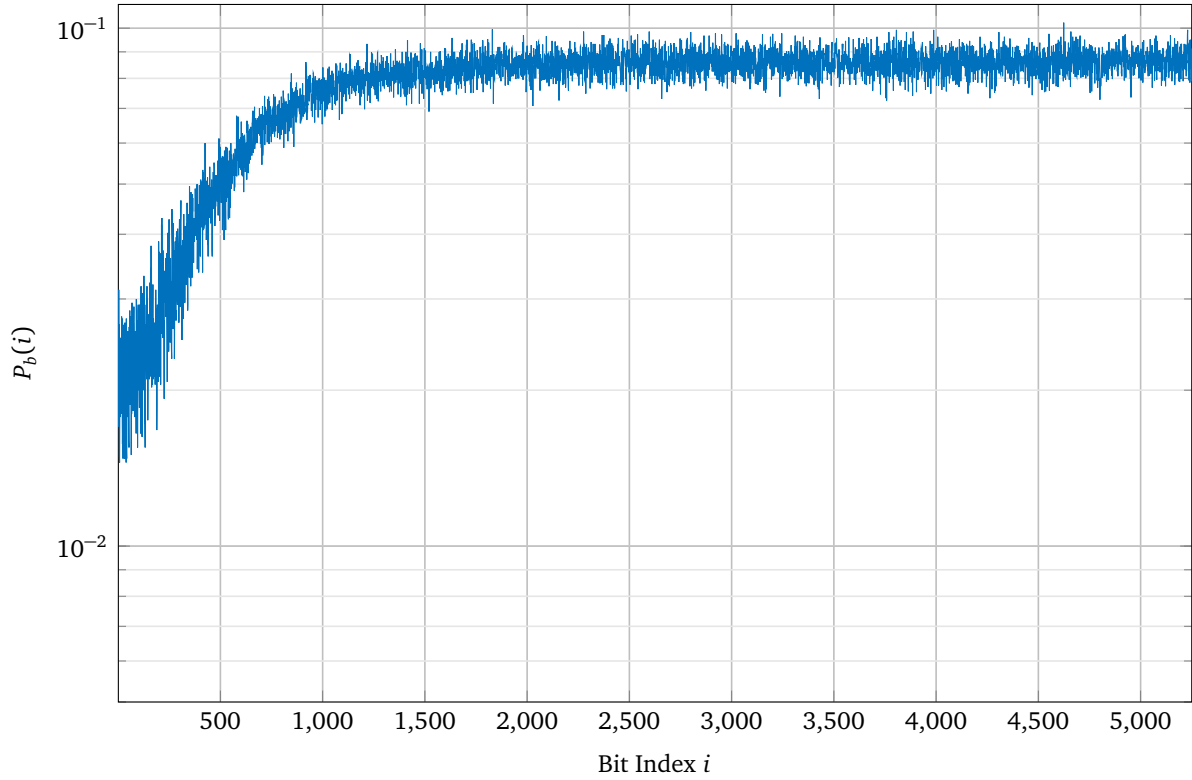
---

### 5.2 Evaluation of Zero-tail Termination

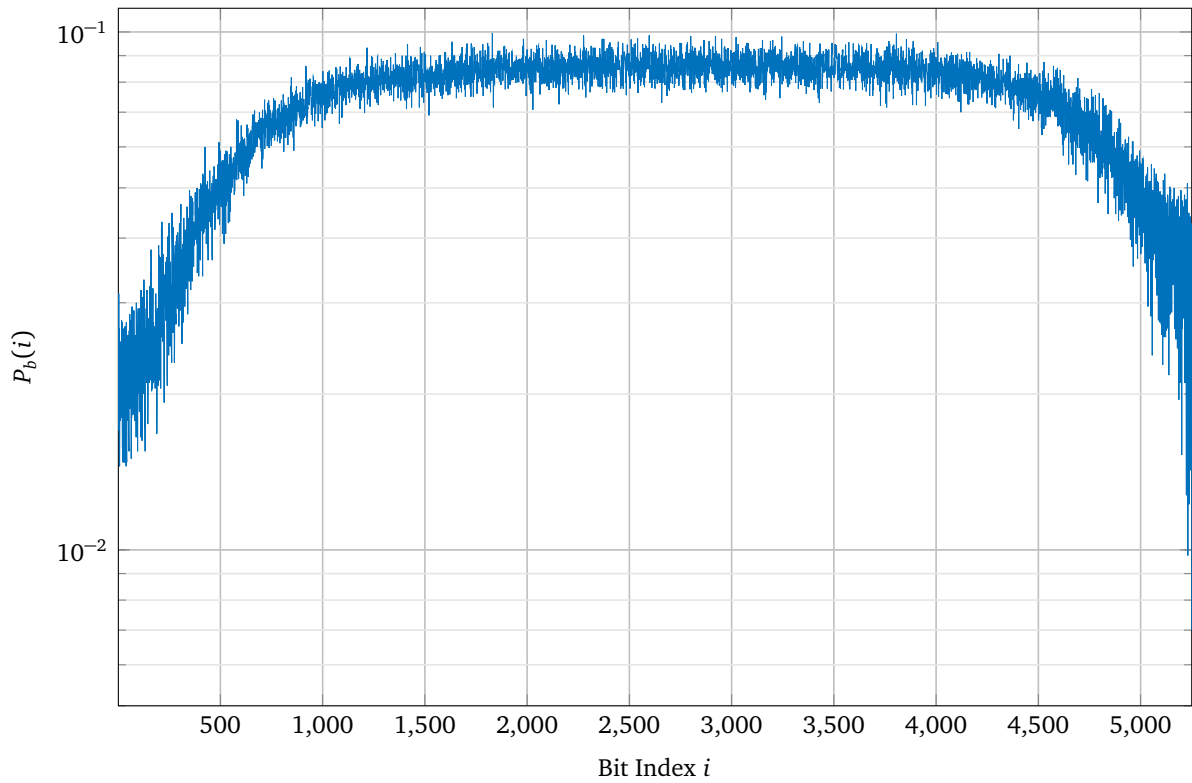
---

In Chapter 3 we concluded that the termination for BPL codes are performed through zero-tail termination. Here, we will evaluate the effect of zero-tail termination on the probability of error for each bit in the codeword. Figure 5.1 shows the probability of error for each bit in the codeword which are calculated after being decoded by the BD. The decoding is performed assuming that the zero-tail bits are not known at the receiver and so no effect of termination is applied on the codeword bits. Figure 5.2 shows the probability of bits in the codeword when zero-tail bits are known at the receiver. The plots include the information bits and the parity bits but do not include the termination sequence.

With all zero-tail terminated BPL codes, the zero-tail bits are always known at the receiver. From Figure 5.2 we can see that when zero-tail bits are known at the receiver, the decoder reduces the  $P(i)$  of the information bits and parity bits in the right of the codeword. While, Figure 5.1 shows that the lack of knowledge of zero-tail bits at the receiver do not reduce  $P(i)$  at the end of the codeword. So, the lack of proper termination does not reduce the  $P(i)$  of the bits in the right of the codeword as seen in Figure 5.1 but the zero-tail termination reduces the  $P(i)$  which is seen in Figure 5.2. Hence, zero-tail termination is an acceptable alternative to proper termination.



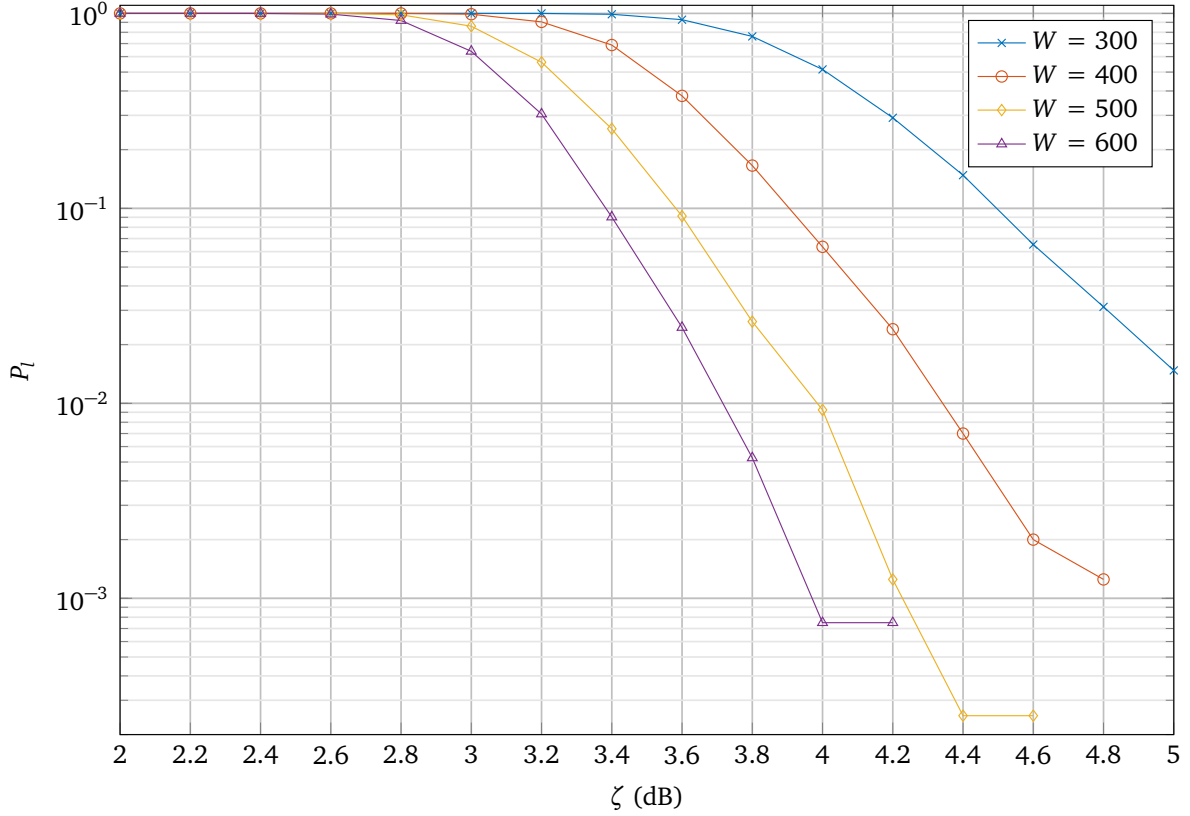
**Figure 5.1:** Probability of error for information and parity bits in the codeword. Zero-tail bits are not known at the receiver. Simulation parameters are  $n_i = 3500$ ,  $R_\infty = 2/3$ ,  $W = 700$  and  $\zeta = 2$  dB.



**Figure 5.2:** Probability of error for information and parity bits in the codeword. Zero-tail bits are known at the receiver. Simulation parameters are  $n_i = 3500$ ,  $R_\infty = 2/3$ ,  $W = 700$  and  $\zeta = 2$  dB.

### 5.3 Evaluation of Base Decoder

Here, we analyze the performance of our BD. First we analyze the performance vs complexity over different window sizes. Figure 5.3 shows the overall BLER  $P_l$  over an SNR range of  $2 \leq \zeta$  (dB)  $\leq 5$  over different window sizes. Figure 5.4 shows the ANEU over the same range of SNR.



**Figure 5.3:** BLER vs SNR of the Base Decoder (BD) with  $n_i = 3500$  and  $R_\infty = 2/3$ .

In Figure 5.3 we see the BLER improves with increasing window size  $W$ . When the window size is increased, more VNs are included in the window enabling messaging passing over a large number of VNs. This is seen the Figure 5.4 as higher window size has higher ANEU in the low SNR region. In high SNR region, larger window size has an advantage in improving the VN reliability thus converging the window quicker. This is why the complexity is less compared to smaller windows in high SNRs.

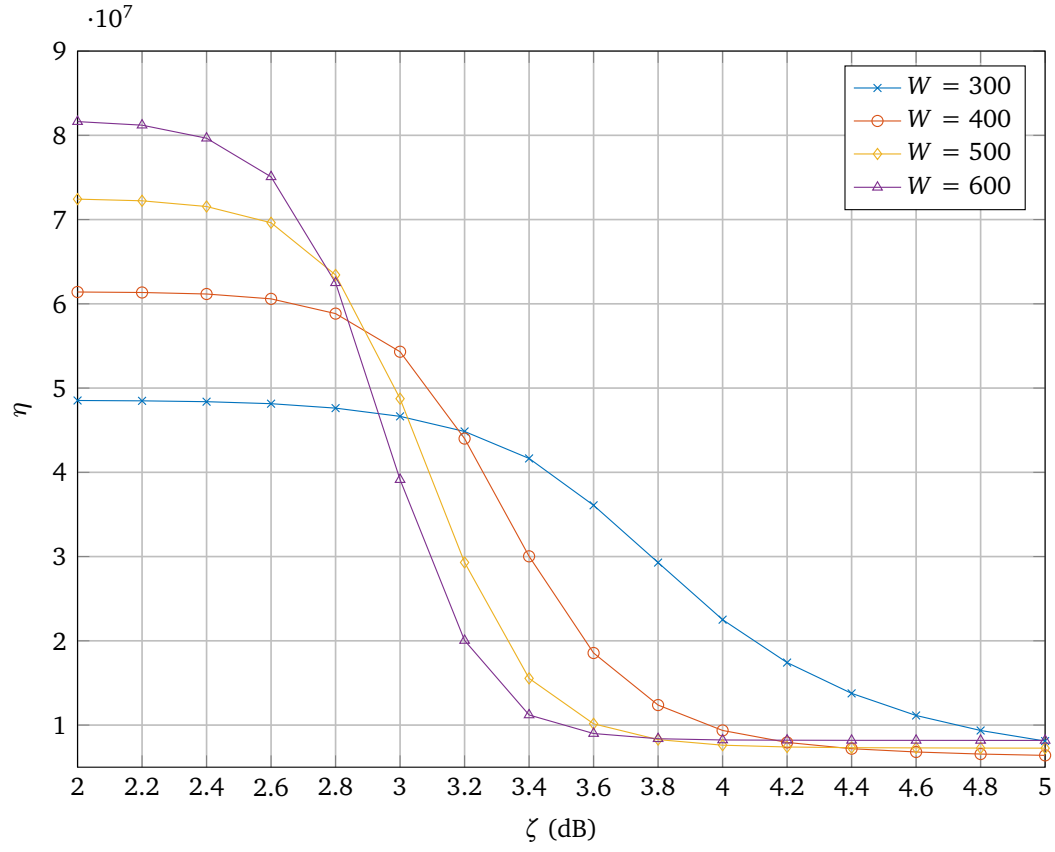
Figure 5.5 shows different BLER plots for all available rates  $R_\infty$  for BPL codes. It is well known that the performance of the code increases with decreasing  $R_\infty$ .

Figure 5.6 and Figure 5.7 shows the BD performance and complexity over different number of iterations  $I$  per window. We can see that the performance of the decoder increases with increasing number of iterations because more iterations of message passing increases the reliability of the VNs. And with more iterations comes more complexity.

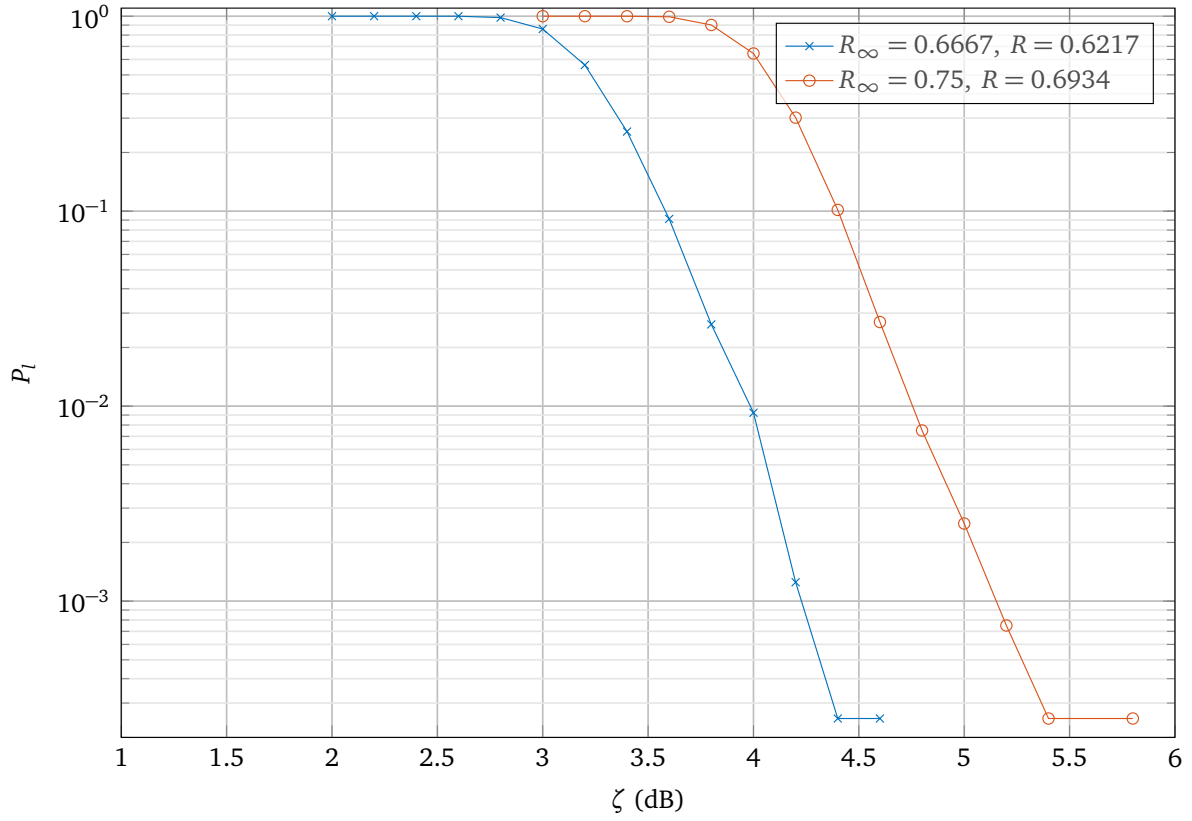
### 5.4 Evaluation of Left-Right-Left Decoder

Now we compare and evaluate the performance of LRL decoder configuration-I with the BD. Figure 5.8 shows two plots of BLER for BD and LRL decoder with window size of  $W = 300$ . Figure 5.9 shows two plots of ANEU for BD and LRL decoder with window size of  $W = 300$ .

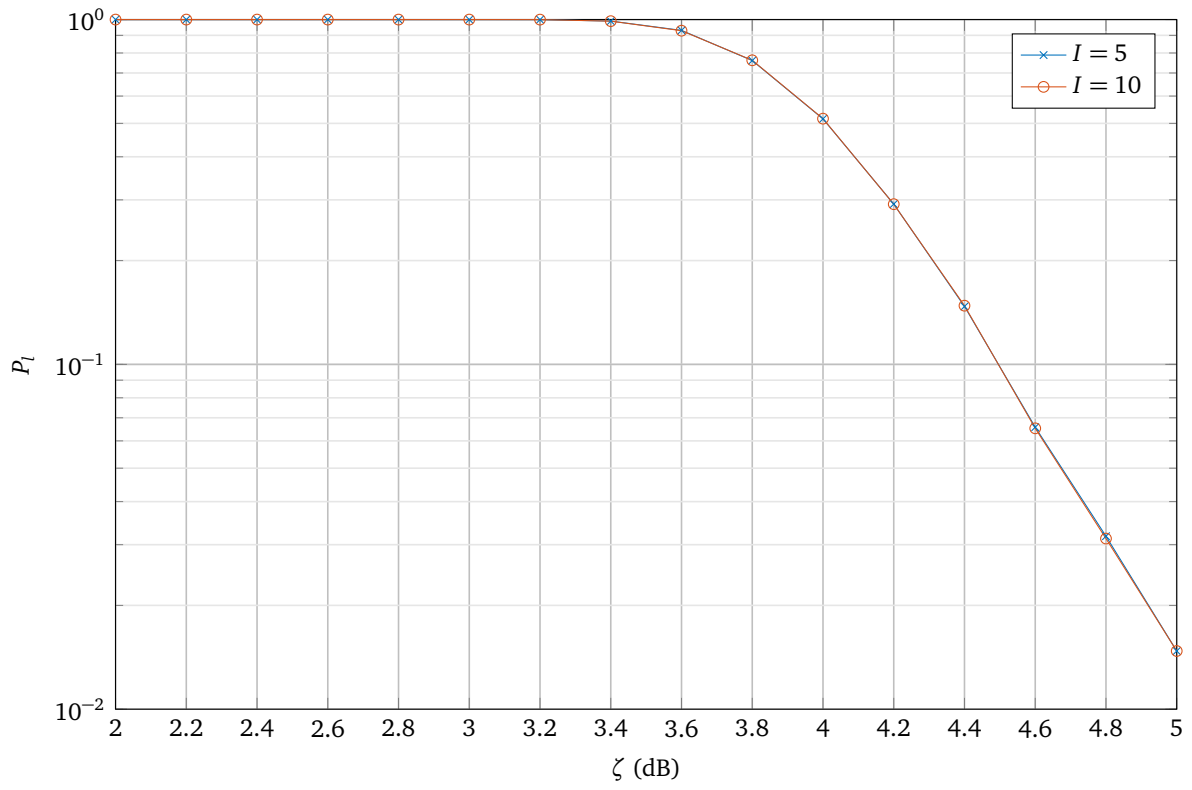
From both the figures, we see a significant decrease in BLER and ANEU for the LRL decoder. The second phase of the LRL decoder has improved the certainty of the VNs through the highly reliable VNs



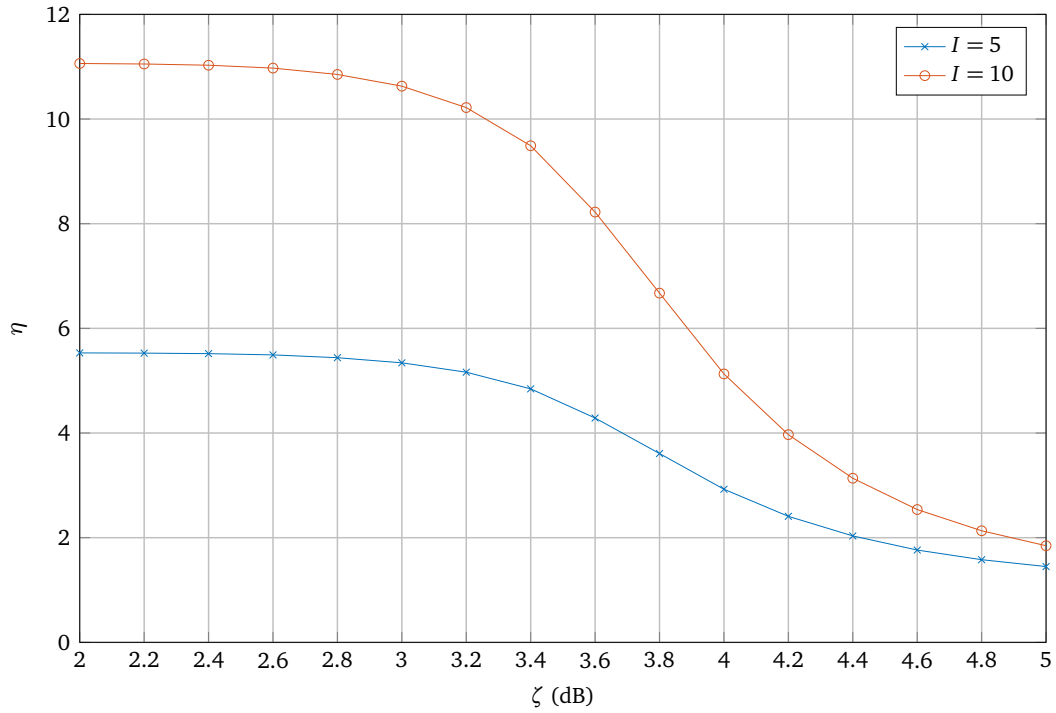
**Figure 5.4:** BLER vs SNR of the Base Decoder (BD) with  $n_i = 3500$  and  $R_\infty = 2/3$ .



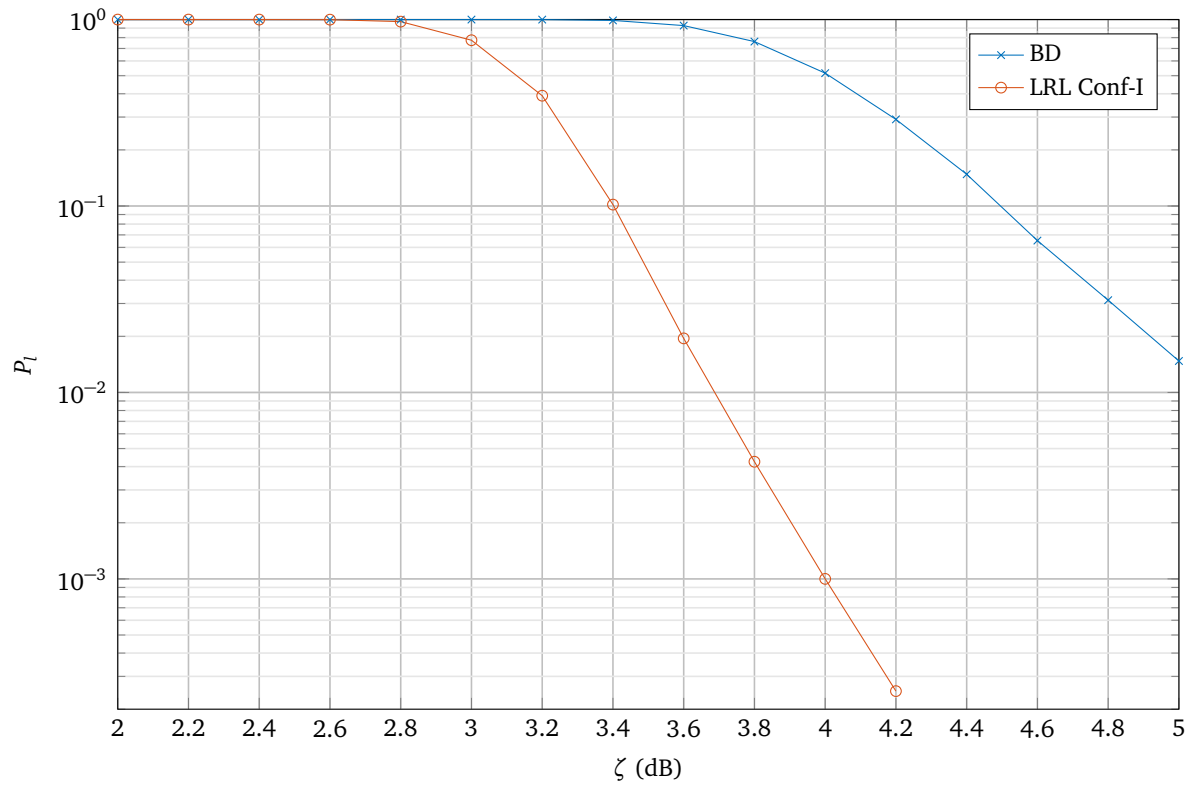
**Figure 5.5:** BLER vs SNR of the Base Decoder (BD) with  $n_i = 3500$  and  $W = 500$ .



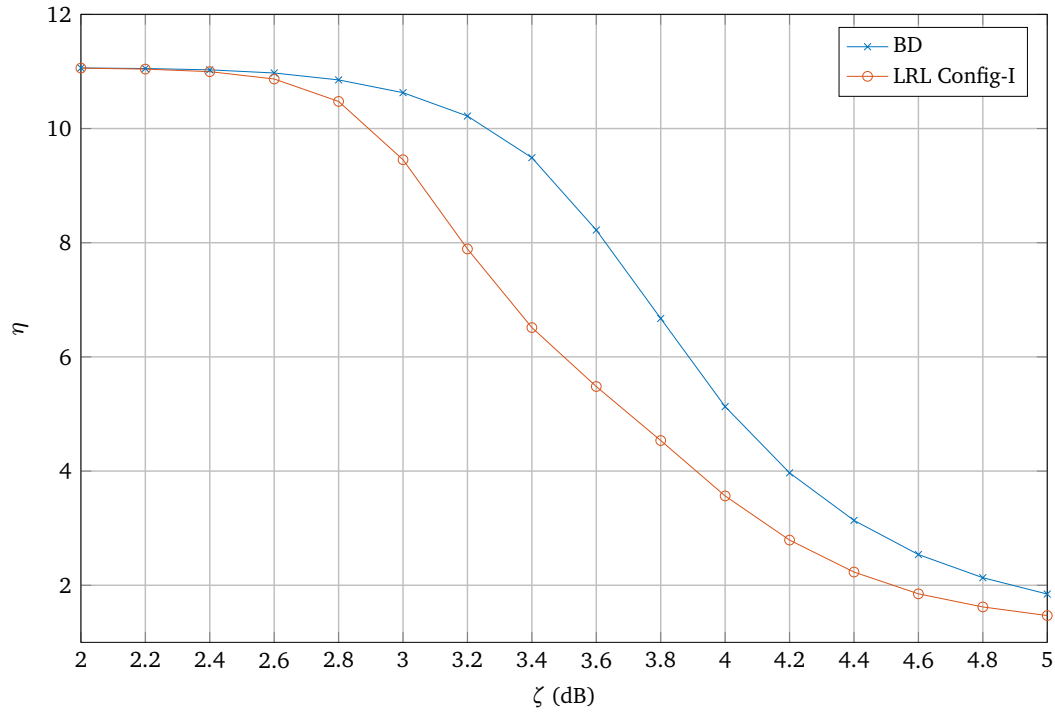
**Figure 5.6:** Comparison of BLER of the Base Decoder (BD) for different  $I$  with  $n_l = 3500$  and  $W = 300$ .



**Figure 5.7:** Comparison of ANEU of the Base Decoder (BD) for different  $I$  with  $n_l = 3500$  and  $W = 300$ .



**Figure 5.8:** Comparison of BLER between the Base Decoder and LRL decoder with  $W = 300$ .

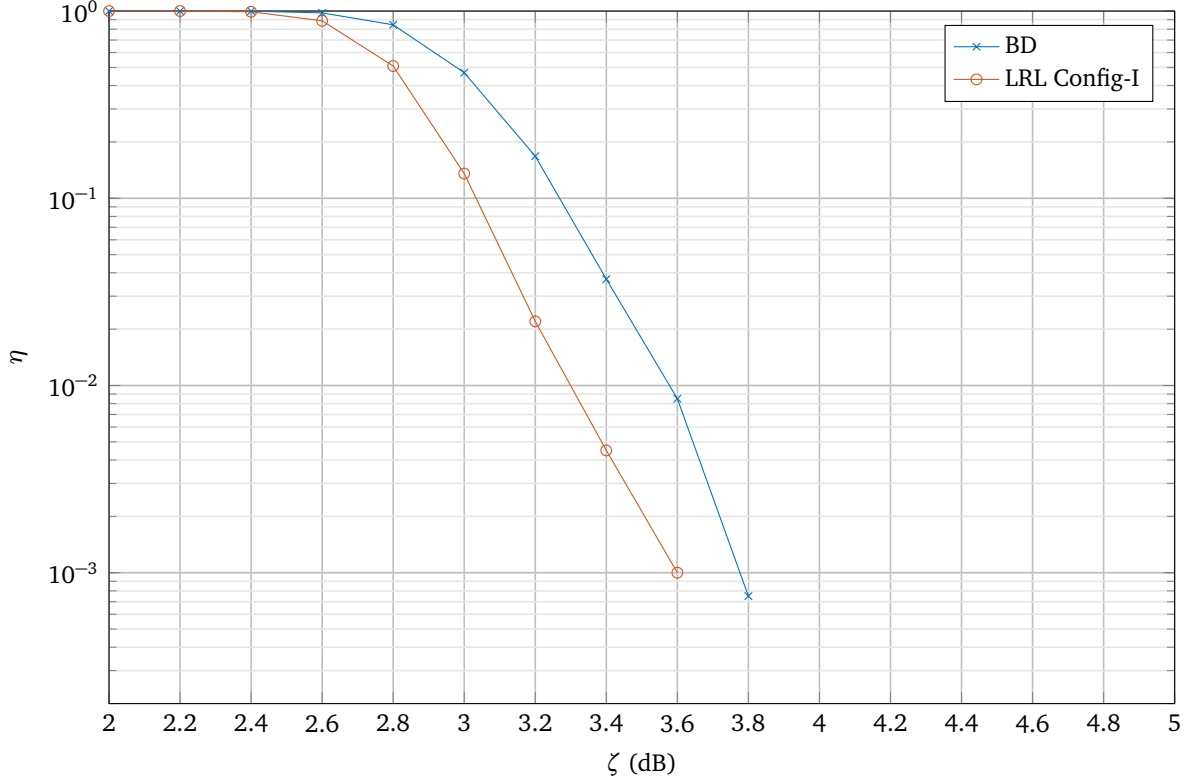


**Figure 5.9:** Comparison of ANEU between the Base Decoder and LRL decoder with  $W = 300$ .



in the right end of the codeword. Hence, the proposed LRL decoder is better than the BD in terms of performance and complexity.

Figure 5.10 and Figure 5.11 compares the performance and complexity between the BD and LRL decoder configuration-I for window size  $W = 700$ . We see that as the window size increases there is an improvement in BLER but not much decrease in the complexity. This is because with larger windows, each window converges quicker than smaller windows. So, it is more likely that not all iterations are used in the BD and the second phase of the LRL decoder has an advantage in decreasing BLER.



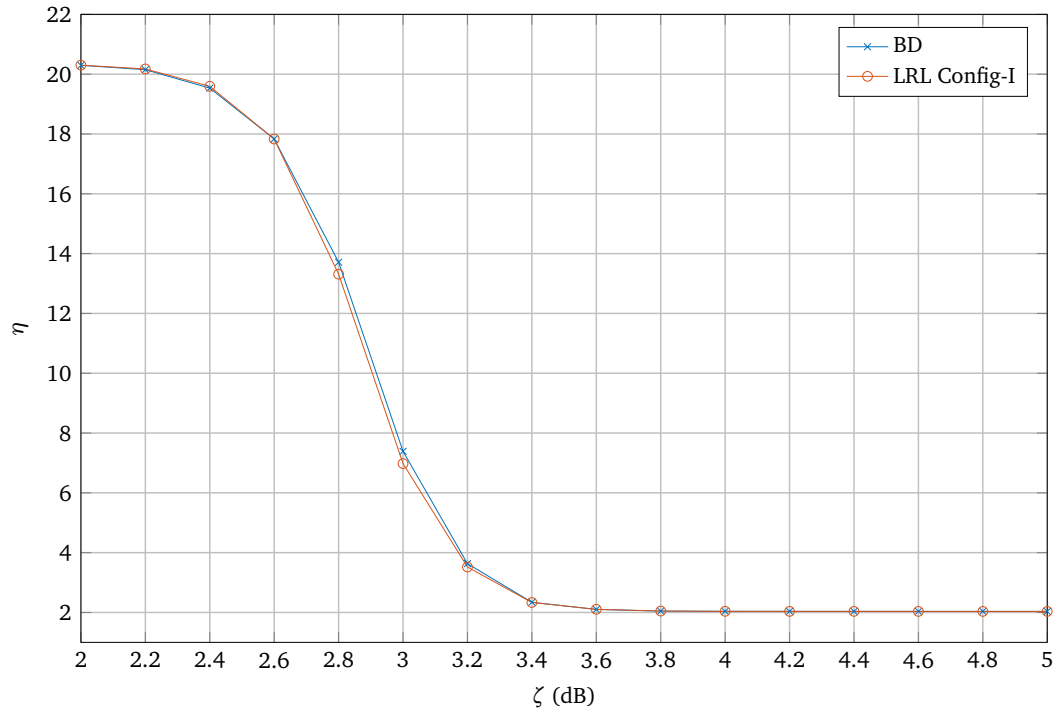
**Figure 5.10:** Comparison of BLER between the Base Decoder and LRL decoder with  $W = 700$ .

Figure 5.12 and Figure 5.14 compares the performance and complexity between the LRL decoders configuration-I and configuration-II. We see that the LRL decoder with configuration-II yields the same BLER performance with a slightly reduced complexity. The BER plot from Figure 5.13 also indicates the same. The reduced complexity is mainly because of the bottom to top CN update in the second phase of the LRL decoder configuration-II. Also since the target VN are in the right end of the window, the window converges faster than the configuration-I decoder.

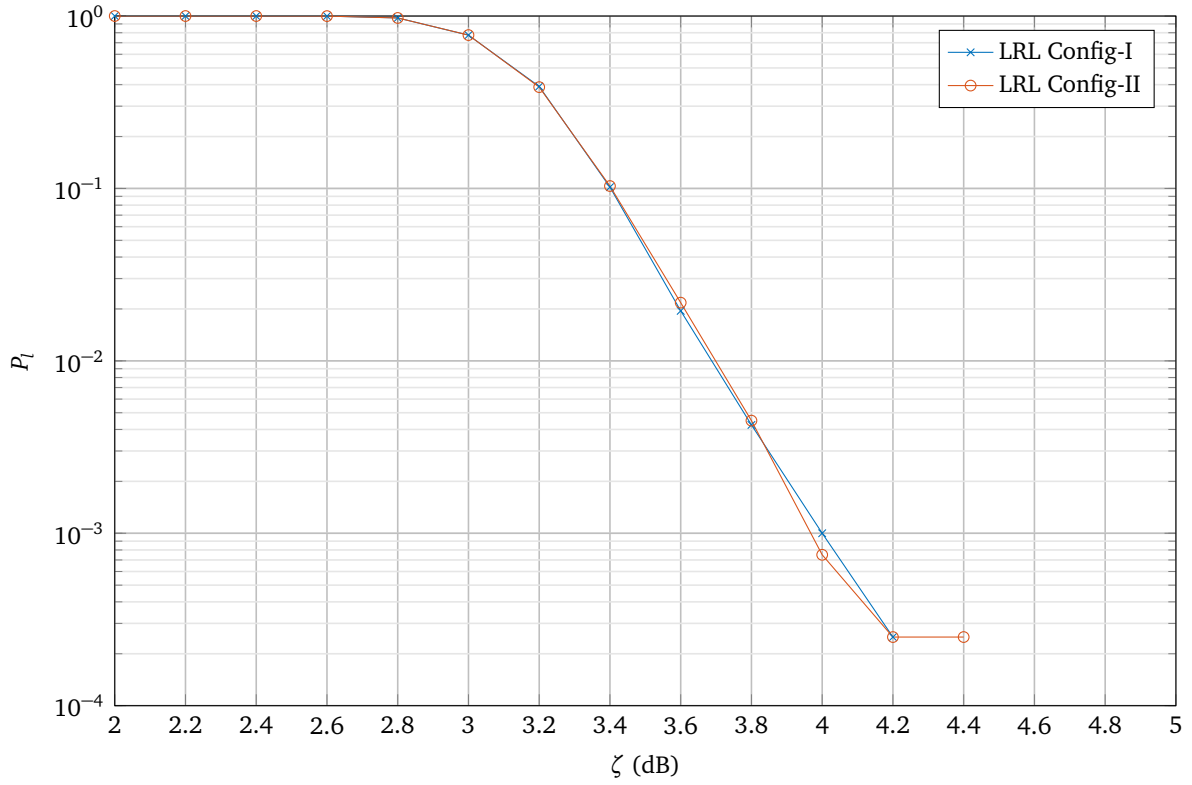
## 5.5 Evaluation of Improved Partial-Syndrome-Check Technique

Here, we evaluate the performance of our IPSC technique. Figure 5.15 and Figure 5.16 compares the performance and complexity between the BD with different early-success criteria with  $W = 300$ . Similarly, Figure 5.17 and Figure 5.18 compares for  $W = 600$ .

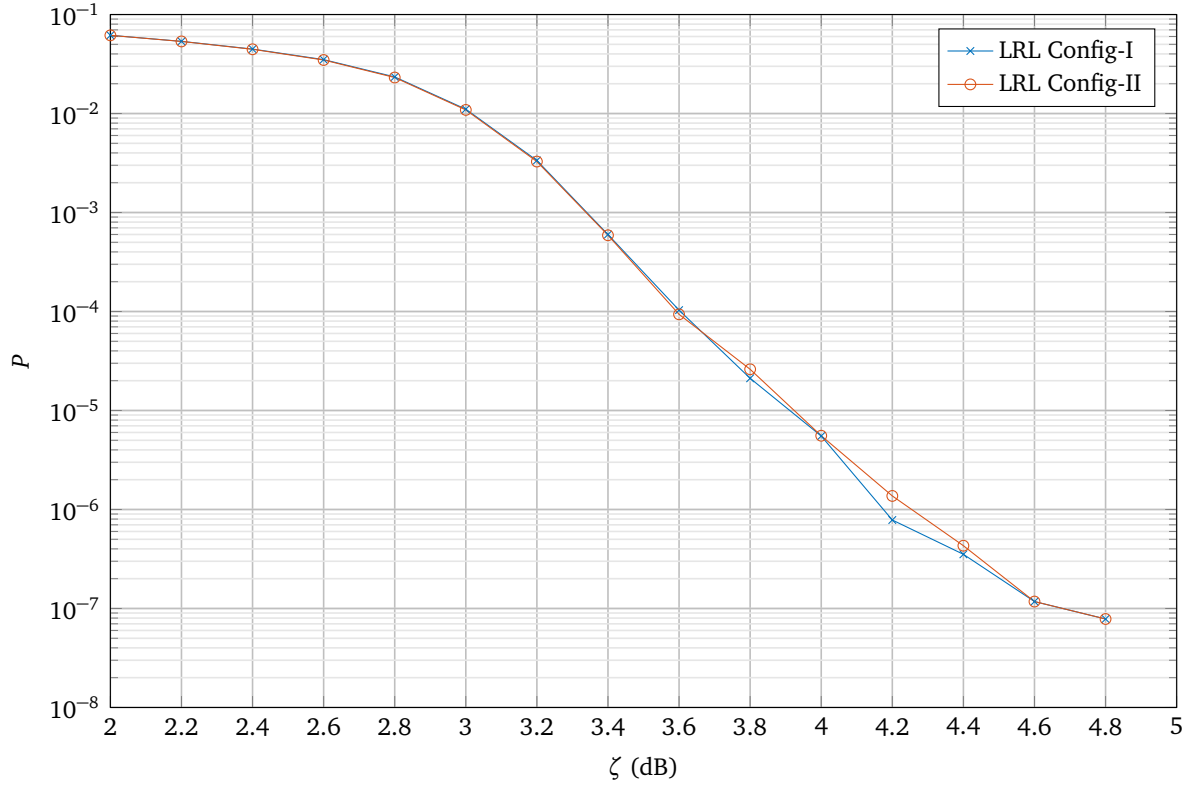
From the Figures 5.15, 5.16, 5.17 and 5.18 we see that for  $W \geq 2(m_s + 1)$ , checking only the target CNs as early-success criteria decreases the decoding complexity.



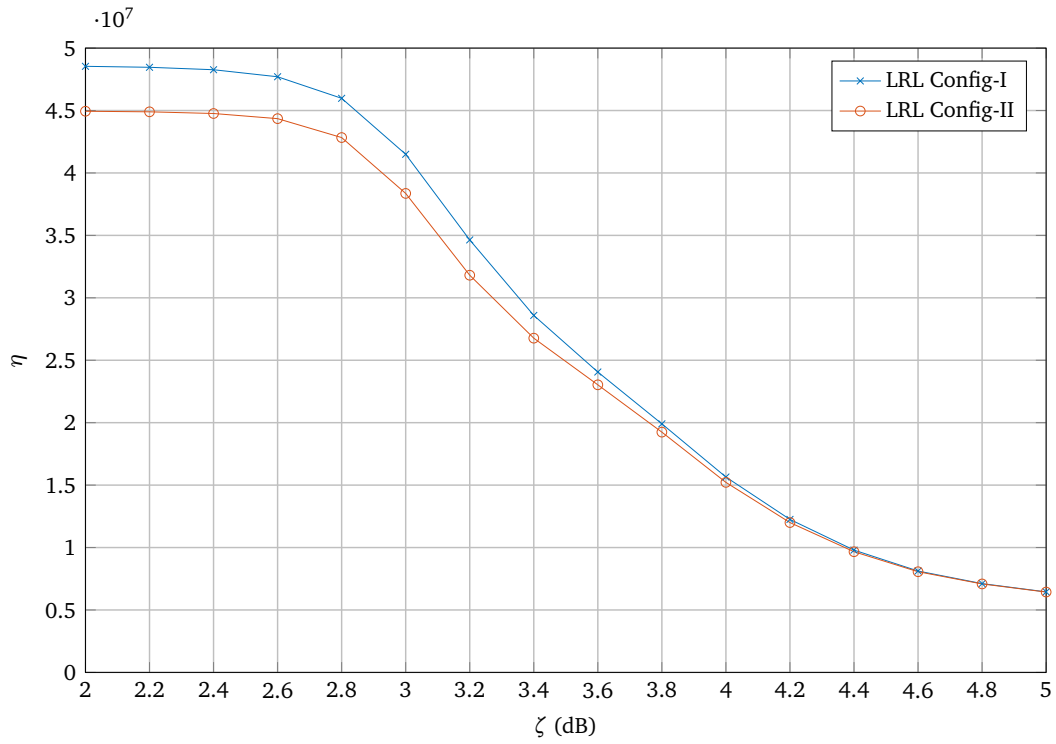
**Figure 5.11:** Comparison of ANEU between the Base Decoder and LRL decoder with  $W = 700$ .



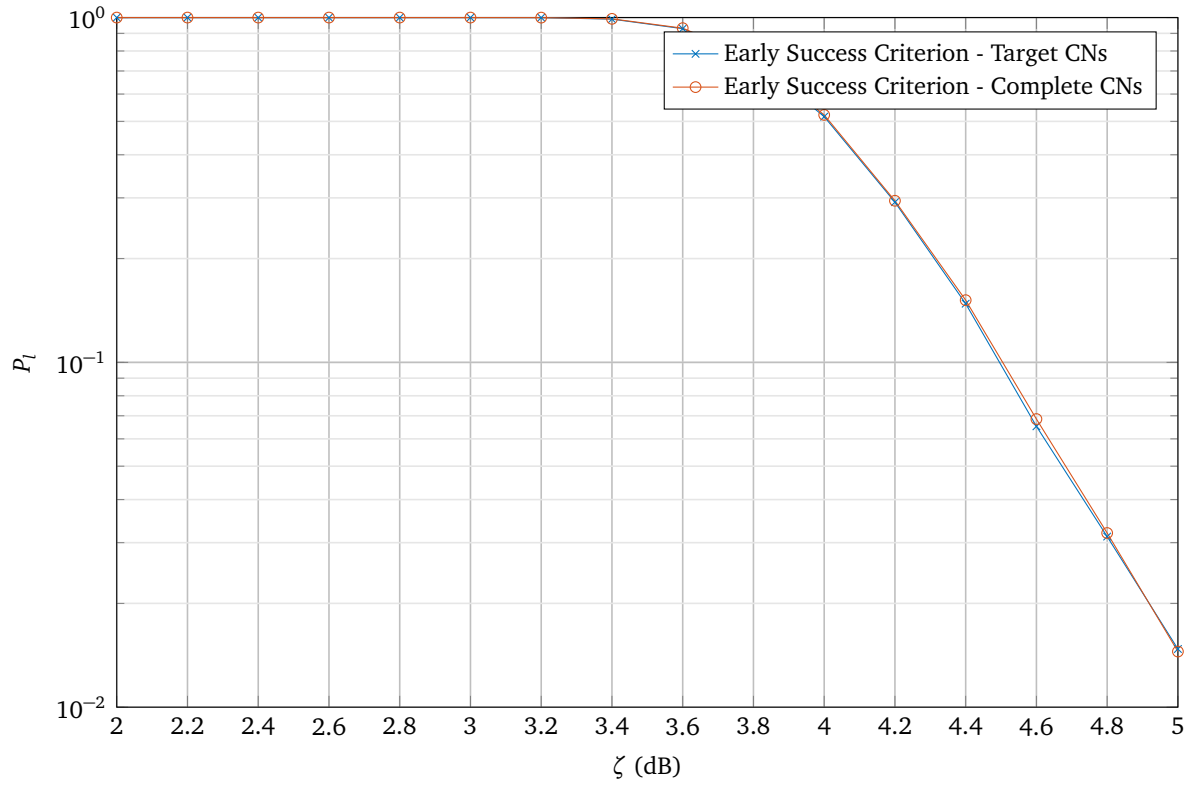
**Figure 5.12:** Comparison of BLER between the LRL decoders configuration-I and configuration-II with  $W = 300$ .



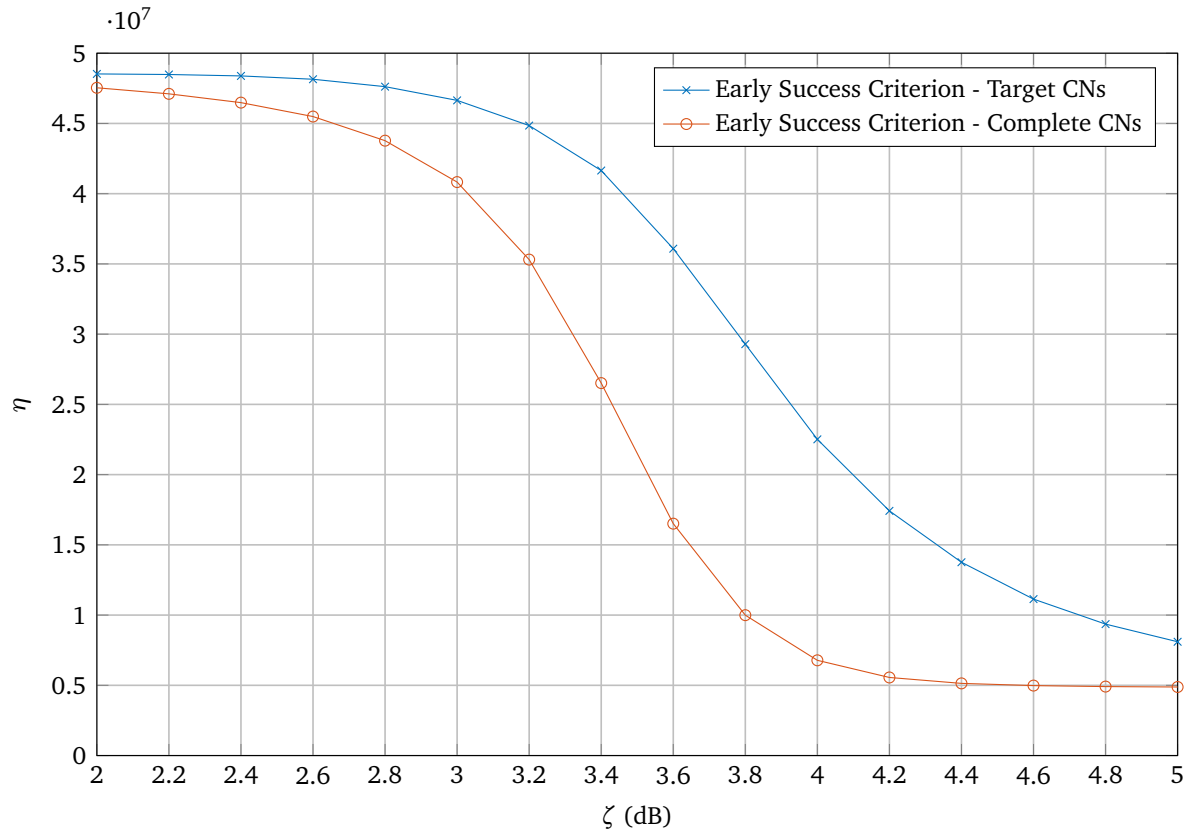
**Figure 5.13:** Comparison of BER between the LRL decoders configuration-I and configuration-II with  $W = 300$ .



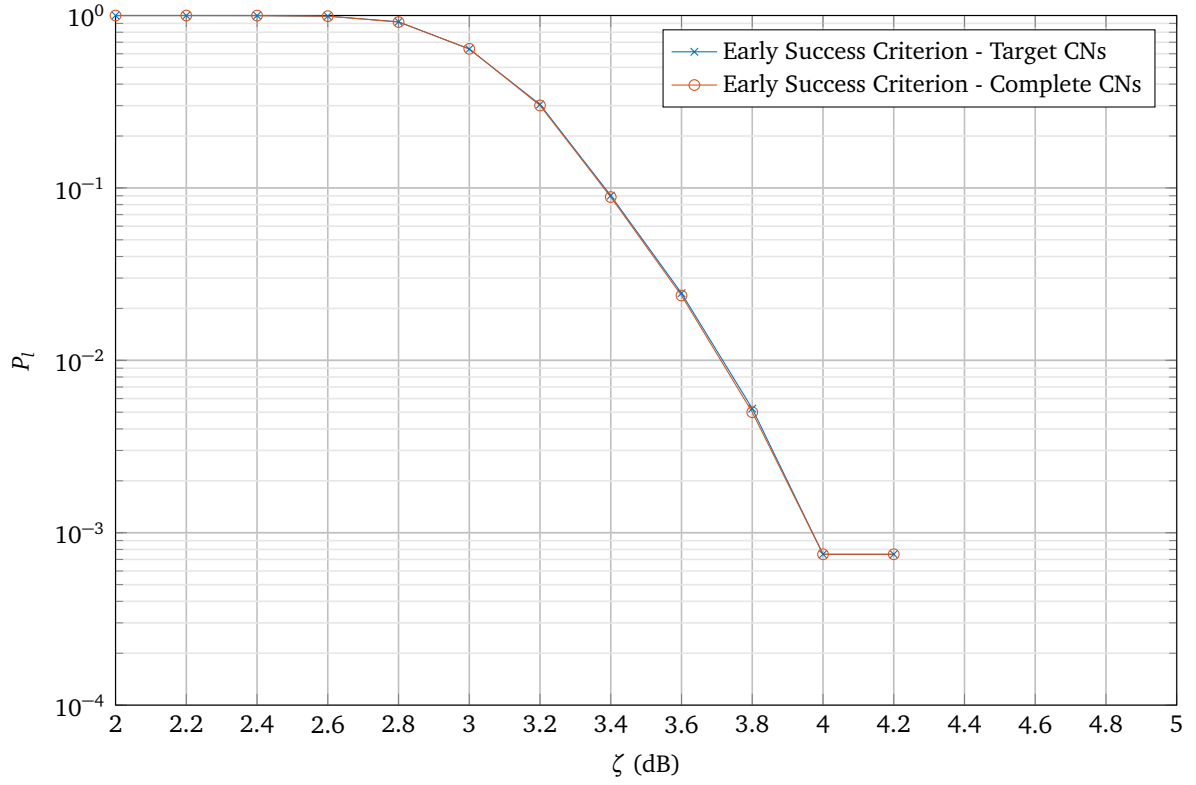
**Figure 5.14:** Comparison of ANEU between the LRL decoders configuration-I and configuration-II with  $W = 300$ .



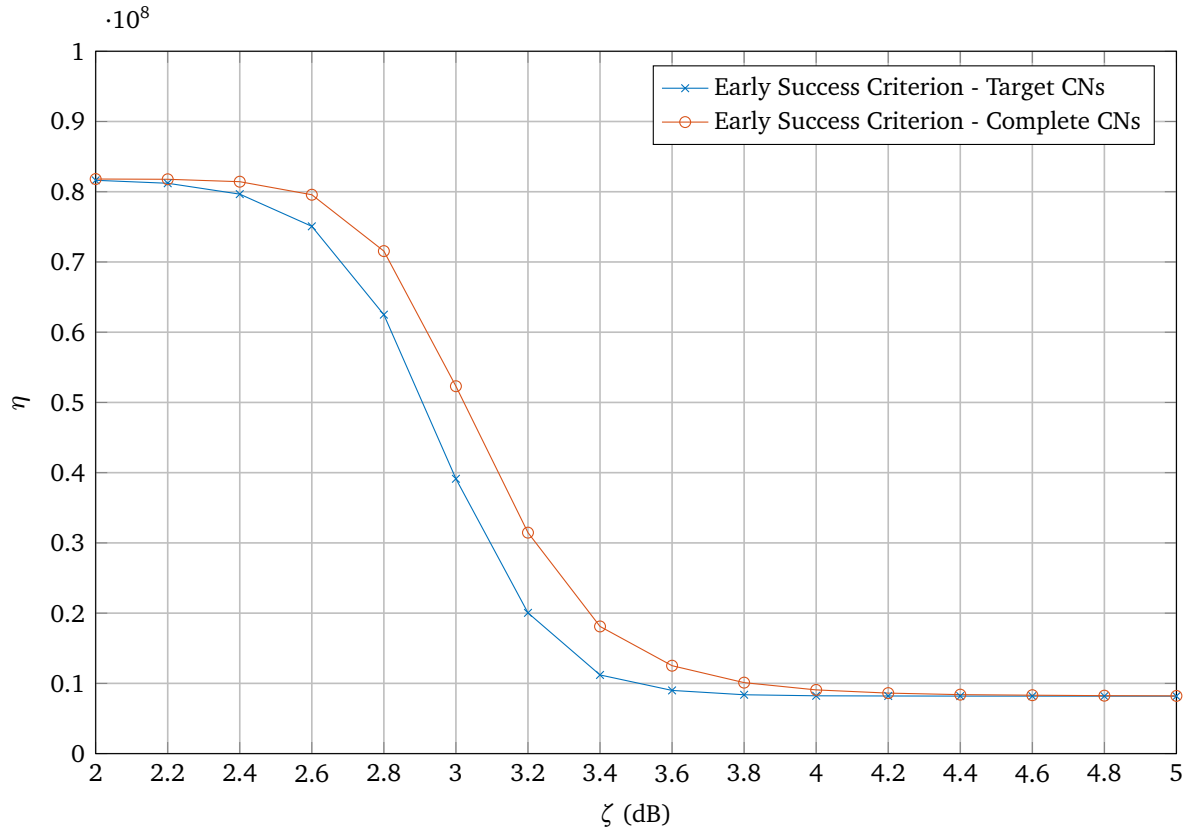
**Figure 5.15:** Comparison of BLER between different early-success criteria with  $W = 300$ .



**Figure 5.16:** Comparison of ANEU between different early-success criteria with  $W = 300$ .



**Figure 5.17:** Comparison of BLER between different early-success criteria with  $W = 600$ .



**Figure 5.18:** Comparison of ANEU between different early-success criteria with  $W = 600$ .



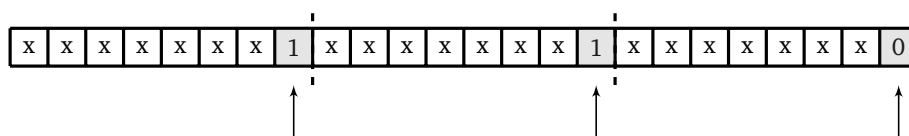
## 6 Implementation Aspects

In this chapter, we will discuss how the encoder and decoder are implemented. We start with describing the encoder's implementation and reasons for the chosen method. Then we discuss about the implementation of the decoder and the implications of different implementations on the execution complexity.

### 6.1 Variable Node Storage Memory Format

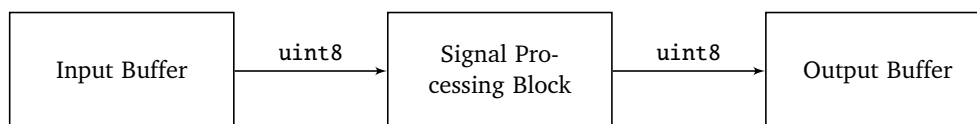
The VNs or the bits in the codeword can be stored in two different ways:

1. Byte for a bit: Each byte of memory contains eight bits out of which the LSB represents one VN. An example of such a storage scheme is shown in Figure 6.1. This form of storage allows us to directly



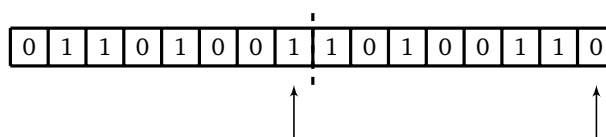
**Figure 6.1:** Three bits of value 0, 1 and 1 are stored in single byte each. The arrow indicates the position of LSB where the bit is stored in each byte. The bits marked with x are unused. Note that the bytes are represented in little-endian format.

access each bit as uint8 and use them to perform signal processing as shown in Figure 6.2.



**Figure 6.2:** Direct access and use of bits.

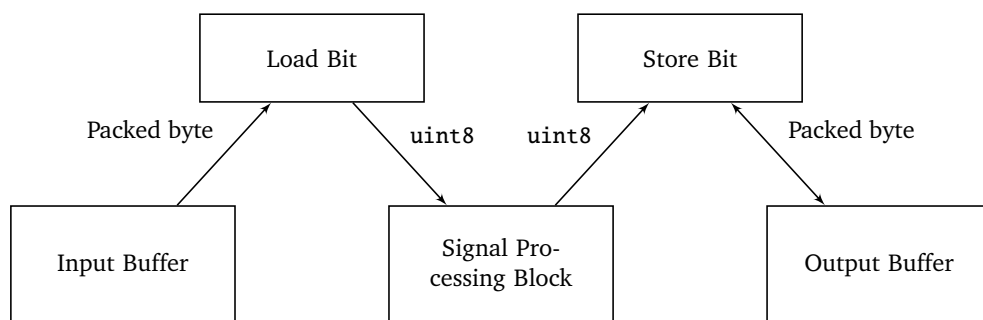
2. Packed byte of bits: Each byte of memory contains eight bits representing eight VNs. An example of such storage is shown in Figure 6.3. With this form of storage, additional functions are required



**Figure 6.3:** Packed bits of bytes.

to access and store each bit from and to its corresponding position because the minimum quantity of bits that can be accessed from the memory at once is a byte or char or uint8. This is shown in figure 6.4.

In our implementations, we use packed byte of bits format. This format reduces the memory requirements for storing information and codeword bits by a factor of eight. Table 6.1 shows the memory required to store 5000 VNs.



**Figure 6.4:** Access and use of bits using helper functions.

Storage Format	Memory Required (Bytes)
Byte for a bit	5000
Packed byte of bits	625

**Table 6.1:** Memory required to store 5000 VNs

However, the gain in minimum memory requirement comes with a cost of increased execution complexity. The *Load Bit* blocks performs memory-read and bit-shift operations. The *Store Bit* block performs some arithmetic and logical operations along with memory-read and write operations. Table 6.2 shows the operations performed in each call of *Load Bit* and *Store Bit*. Note that this is just one way of implementing the functions.

Function called	Reads	Writes	Bit Shift	Arithmetic	Logical
Load Bit	1	0	2	0	0
Store Bit	1	1	1	1	2

**Table 6.2:** Number of different operations performed during each call of *Load Bit* and *Store Bit*.

The Table 6.3 compares the memory requirement and operations required to encode a codeword with 5000 information bits with  $R_{\infty} = 1/2$ .

Although the execution complexity of using *Packed byte of bits* format is high compared to the other format, it is very negligible compared to the overall decoding complexity. Hence, the *Packed byte of bits* format is chosen to reduce the memory requirement.

## 6.2 Variable Node Indexing

The LLR values of VNs are stored in an array of memory where each element is a 32-bit floating point value. During each CN update, the BP algorithm selects the corresponding VN's LLR to perform the updates. This selection of VNs is done via indexing. There are two ways to perform the selection:

1. *Method 1:* During each CN update, compute the indices of the corresponding VNs so that the BP algorithm selects them.
2. *Method 2:* Before start of decode, compute and store the indices of VNs for all CNs in the whole PCM.



Storage Format	Memory Required (Bytes)	Reads	Writes	Bit Shift	Arithmetic	Logical
Byte for a bit	15,000	5,000	10,000	0	0	0
Packed byte of bits	1,875	15,000	10,000	30,000	10,000	20,000

**Table 6.3:** Memory requirement and operations required to encode a codeword with 5000 information bits with  $R_{\infty} = 1/2$ .

The method 1 computes the indices on the fly during each CN update. So, the complexity of calculating the indices increases linearly with  $I$  and  $Ln$ . While, when the method 2 is used there is no increase in index computation complexity with increasing  $I$  or  $Ln$ . But the memory required to store the indices of all CNs increases linearly with number of CNs. Each index is stored in a uint32 Table 6.4 shows the memory and computation requirements for indices calculation in method 1 and method 2.

Method	Memory Required (Bytes)	CN Index Calculations
Method 1	$3 \cdot n \cdot \text{size of uint32} = 24$	No. of CNs $\cdot I = 12,500$
Method 2	$3 \cdot n \cdot \text{size of uint32} = 24 \cdot \text{No. of CNs} = 60,000$	No. of CNs = 2,500

**Table 6.4:** Memory requirement and computations required to calculate and store indices for decoding a codeword with 5000 information bits with  $R_{\infty} = 1/2$ , maximum window size  $W = 2500$  and  $I = 5$ .



---

## 7 Conclusion

---

# Bibliography

- [1] S. Abu-Surra, E. Pisek, and R. Toari, “Spatially-coupled low-density parity check codes: Zigzag-window decoding and code-family design considerations,” in *2015 Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, Feb. 2015.
- [2] P. Kang, Y. Xie, L. Yang, and J. Yuan, “Reliability-based windowed decoding for spatially coupled ldpc codes,” *IEEE Communications Letters*, vol. 22, no. 7, pp. 1322–1325, Jul. 2018.
- [3] I. Ali, J.-H. Kim, S.-H. Kim, H. Kwak, and J.-S. No, “Improving windowed decoding of sc ldpc codes by effective decoding termination, message reuse, and amplification,” *IEEE Access*, vol. 6, pp. 9336–9346, Mar. 2018.
- [4] M. Bossert, *Channel Coding for Telecommunications*. West Sussex, England: John Wiley & Sons, 1999.
- [5] J. G. Proakis, *Digital Communication*. McGraw-Hill, 1995.
- [6] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inf. Theory*, vol. 27, no. 15, pp. 533–547, 1981.
- [7] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: MIT Press, 1963.
- [8] A. Felström and K. Zigangirov, “Time-varying periodic convolutional codes with low-density parity-check matrix,” *IEEE Trans. Inf. Theory*, vol. 45, no. 10, pp. 2181–2191, 1999.
- [9] “IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications,” *IEEE Std. 1901-2010*, pp. 1175–1180, 2010.
- [10] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
- [11] Juntan Zhang and M. Fossorier, “Shuffled belief propagation decoding,” in *Conf. Rec. 36th Asilomar Conf. Signals, Syst. Comput. 2002*, Pacific Grove, CA, USA, Nov. 2002, pp. 8–15.
- [12] C. Jones, E. Valles, M. Smith, and J. Villasenor, “Approximate-min\* constraint node updating for LDPC code decoding,” in *Proc. 2003 IEEE Military Commun. Conf. (MILCOM)*, Boston, MA, USA, Oct. 2003, pp. 157–162.
- [13] Z. Chen, S. Bates, D. Elliott, and T. Brandon, “Efficient encoding and termination of low-density parity-check convolutional codes,” in *Proc. 2006 IEEE Global Commun. Conf. (GLOBECOM)*, University of Alberta, Edmonton, Canada, Nov. 2006.