ERROR-FLOORS OF THE 802.3AN LDPC CODE FOR NOISE ASSISTED DECODING

by

Tasnuva Tarannum Tithi

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Chris Winstead, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Reyhan Baktur, Ph.D. Committee Member	Todd Moon, Ph.D. Committee Member
Haitao Wang, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding

by

Tasnuva Tarannum Tithi, Doctor of Philosophy

Utah State University, 2019

Major Professor: Chris Winstead, Ph.D.
Department: Electrical and Computer Engineering

This work characterizes a new noise assisted decoding algorithm called the Noisy Gradient Descent Bit-Flip (NGDBF) for Low Density Parity Check (LDPC) codes on AWGN channel. The NGDBF algorithm is an improvement over the Gradient Descent Bit-Flip (GDBF) algorithm. Random perturbations is used in the decision function to escape from local minima. An improvement over NGDBF is proposed called the Re-decoded NGDBF (R-NGDBF). In the re-decoded version, a number of decoding phases are used per received message instead of a large number of iterations in NGDBF. A Markov chain Monte Carlo (MCMC) based technique is proposed to estimate the error-floors of the IEEE 802.3an LDPC code, for different decoding parameters of NGDBF. The error-floor of the IEEE 802.3an LDPC code is dominated by a structure in the code-graph called the (8,8) trapping set. This dissertation focuses on finding decoding parameters for NGDBF algorithm to resolve the (8,8) trapping set errors to ultimately lower the error-floor. In the MCMC estimation technique, every possible decoding decision is represented as a state. With every iteration of the NGDBF algorithm, flipping one or more bits moves the decision from one state to another state. As the probability of a flip is dependent on the random noise added by NGDBF decoder, the different states of the decision can be considered as the states of a Markov chain. A transition matrix containing the flipping probabilities for each bit can

be obtained using the initial state of the received channel message, and the noise variance of the random perturbation. Once the matrix is obtained, it is simple to find the steady state probabilities of the chain. The steady state probabilities reveal the probability of error for a given channel message. Importance sampling was used to obtain erroneous messages at high SNRs for the (8,8) trapping set structure for varying decoding parameters such as "weight (w)" corresponding to the syndrome weight and "noise-scale (η)"corresponding to the scaled noise variance used in the decoder. The obtained error-floor was scaled by 10^4 to account for the multiplicity of the (8,8) trapping set, to obtain a true estimation of the error-floor. Also, another set of simulations were performed to estimate any newly created errors by the random noise in the NGDBF decoder. The ultimate error-floor is estimated to be where the two sets of simulations cross over each other. The decoder was fully synthesized, implemented and validated for the Xilinx VCU118 FPGA board using Vivado 2017.1. The estimated decoding parameters "weight (w)" and "noise-scale (η)" were used in the implementation to validate the MCMC simulation result, and the results corroborated.

(72 pages)

PUBLIC ABSTRACT

Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding

Tasnuva Tarannum Tithi

In digital communication, information is sent as bits, which is corrupted by the noise present in wired/wireless medium known as the channel. The Low Density Parity Check (LDPC) codes are a family of error correction codes used in communication systems to detect and correct erroneous data at the receiver. Data is encoded with error correction coding at the transmitter and decoded at the receiver. The Noisy Gradient Descent Bit-Flip (NGDBF) decoding algorithm is a new algorithm with excellent decoding performance with relatively low implementation requirements. This dissertation aims to characterize the performance of the NGDBF algorithm. A simple improvement over NGDBF called the Re-decoded NGDBF (R-NGDBF) is proposed to enhance the performance of NGDBF decoding algorithm. A general method to estimate the decoding parameters of NGDBF is presented. The estimated parameters are then verified in a hardware implementation of the decoder to validate the accuracy of the estimation technique.

To my family and teachers...

ACKNOWLEDGMENTS

My sincerest gratitude to my advisor Dr. Chris Winstead, for being my mentor, for training me to become an engineer, for the patience and for the most pleasant company. The Low Energy Fault Tolerant (LE/FT) lab, became my home since the summer of 2012. Our lab, though small in size and number of students, has never had any shortage of ideas and projects. Nor was I ever limited by hard-lined research area to work on. I have enjoyed unlimited freedom, acceptance and continued financial support at our lab- that has been a sanctuary to me for the last 7 years.

I came to USU as an MS student in the fall of 2011, and decided to continue my Ph.D. under Dr. Winstead's supervision. Initially overwhelmed by the curriculum here at the ECE department with my background in Applied Physics, it took me a longer time to become a productive Ph.D. student. It started to get easier as my advisor invested a lot of his time and energy training me. I thank him for his patience, friendship, and work philosophy. With his guidance, I have enjoyed working on projects on Information theory - Error correction codes, coding up FPGA boards, Cyber-physical security for autonomous vehicles, and being a TA for microelectronics. Unlike most Ph.D. students, I enjoyed every single day of my time working on my degree. My grateful thanks to all the current and former students at LE/FT lab- Gopal, Yi, Soudeh, Mckay, Rakin and Reejoy for their warm friendship and dedication. Most importantly Gopal, whose research started the Noisy Gradient Bit-Flip decoding (NGDBF) project, and later became my area of research for Ph.D. dissertation, deserves special mention. My Ph.D. work was financially supported by the US National Science Foundation and the Research Catalyst grant from Utah State University.

Over this long period, I have been fortunate to take a lot of the graduate courses offered by the ECE department- which has left me at awe with our faculty. Thanks to Dr. Todd Moon and Dr. Jake Gunther for the class lectures, group help sessions, and the drop-in personal help sessions. It is not only the quality of engineering education at ECE, the paramount dedication to teaching and service is what continues to inspire generations

of students. Thanks to our former Asst. Prof. Dr. Ryan Gerdes, for the guidance on the autonomous vehicles project, and helping me to publish my first journal paper. To our former Asst. Professor, Dr. Edmund Spencer for allowing me to work in his lab in my first semester, so that I could continue to be a research student. To Dr. Reyhan Baktur and Dr. Haitao Wang for serving in my committee and being available for assistance. To our now and former staffs at ECE- Tricia, Kathy, Mary Lee and Rob at the Global Engagement Office for taking care of logistics for me. To our former technical supports Trent and Scott for many hours of help with my computer in the lab.

Thanks to my parents- Mesbah Uddin and Jahanara Pervin, for their trust, love, and support- moral and often financial throughout all these years of graduate school; to my husband- Upal, who gave up his promising career in Bangladesh to raise a family with me here in Logan; to our infant daughter Arnina, for sacrificing the utmost attention that she deserves; to my sister Athoi, my brother-in-law Salman and my best friend Mithila for sharing all the anxieties over the long phone calls. Thanks to my friend Divya, to all the current and former Bangladeshi students/post-docs at USU, especially Asif, Avirup, Shumi-Parvez, Shante-Shajeeb, Sonia-Fahmid, Fariba-Rafsan, Shovon, and Towfiq- for the friendship, the adventures, the parties, and the food. A Ph.D. degree is definitely a teamwork of all the people around the candidate providing their support and guidance. I am forever grateful to all of them for their kindness to me.

Tasnuva Tarannum Tithi

CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOTATION

C	Channel capacity
n	Codeword length
k	Uncoded message length
H	$n \times m$ Binary parity check matrix
$\vec{c} \in \mathbb{Z}_2^n$	Binary codeword
$\hat{c} \in \{+1, 1\}^n$	Bipolar codeword $\hat{c} = 1 - 2\vec{c}$
$\vec{z} \in \mathbb{R}^n$	I.I.D. white noise samples
$\sigma^2 \in \mathbb{R}$	Noise variance
$\vec{y} \in \mathbb{R}^n$	Channel samples $\vec{y} = \hat{c} + \vec{z}$
$\Phi \in \mathbb{Z}$	Maximum number of phases in R-NGDBF
$y_{\max}$	Sample clipping limit
$\tilde{y}$	Quantized channel samples on Q bits within $[-y_{\max}, +y_{\max}]$
$\hat{x} \in \{+1, -1\}^n$	Bipolar hypothesis $\vec{x} = \text{sign}(\vec{y})$
$\hat{x} \in \mathbb{Z}_2^n$	Binary hypothesis $\vec{x} = 0.5(1 - \hat{x})$
$\vec{s} \in \mathbb{Z}_2^m$	Syndrome $\vec{s} = H^T \vec{x}$
$\hat{s} \in \{+1, -1\}^m$	Bipolar syndrome
$\eta \in \mathbb{R}$	Noise-scale
$w \in \mathbb{R}$	Weight (syndrome weight)
$\theta \in \mathbb{R}$	Flipping threshold, $\theta < 0$
λ	Threshold (θ) adaptation parameter, $0 < \lambda \leq 1$
$E_i \in \mathbb{R}$	Energy function for bit i
N_i	Adjacency in H for bit i
M_j	Adjacency in H for parity check j
$L \in \mathbb{Z}$	Maximum number of iterations
$\ell \in \mathbb{Z}$	Iteration number between 0 and L
$\vec{z} \in \mathbb{R}^n$	Decision

ACRONYMS

BCH	Bose, Chaudhuri, and Hocquenghem (BCH)
LDPC	Low Density Parity Check
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
SNR	Signal to Noise Ratio
BSC	Binary Symmetric Channel
BFA	Bit-Flip decoding Algorithm
BP	Belief Propagation
MS	Min-Sum
ML	Maximum-Likelihood
PBFA	Probabilistic Bit-Flip decoding Algorithm
WBF	Weighted Bit-Flip
GDBF	Gradient Descent Bit-Flip
M-GDBF	Multi-bit Gradient Descent Bit-Flip
S-GDBF	Single-bit Gradient Descent Bit-Flip
S-NGDBF	Single-bit Noisy Gradient Descent Bit-Flip
M-NGDBF	Multi-bit Noisy Gradient Descent Bit-Flip
SM-NGDBF	Smoothed Multi-bit Noisy Gradient Descent Bit-Flip
R-NGDBF	Re-decoded Noisy Gradient Descent Bit-Flip
IDB	Improved Differential Binary
IEEE	Institute of Electrical and Electronics Engineers
FPGA	Field Programmable Gate Array
SVD	Singular Value Decomposition
ETU	Early Termination Unit

CHAPTER 1

INTRODUCTION

1.1 History of Coding theory

Published in 1948, Shannon's work "A mathematical theory of communication" initiated the two fields of Information theory and Coding theory. In the paper, Shannon proposed "Channel capacity, C " as a quantitative measure of the upper bound of the rate, R -at which information can be transmitted over a channel. Shannon demonstrated that a proper coding scheme achieves transmission of information over a noisy channel at a rate R less that or equal to channel capacity C with a small frequency of error or equivocation [1]. The field of Information theory is concerned with the bounds on the communication channel when proper encoding is applied. On the other hand, coding theory is concerned with efficient encoding and decoding of information to reach Shannon's limit. Over the past 70 years coding theorists and communication engineers proposed numerous error correcting coding schemes approaching Shannon's limit for various applications. All aspects of digital data transmission and data storage employ error correcting codes.

Error correcting codes can be broadly divided into two categories: block codes and tree codes [2]. In the case of block codes, every k information-symbols are encoded into an n-tuple of channel-symbols, where $n > k$. The codeword of length n is then modulated and transmitted over the channel. Tree codes take the information continuously and associate it with a somewhat longer code sequence. The input sequence to the encoder is broken into a sequence of k_0 symbols and then combined with preceding information symbol which generates an n_0-symbol section of the code. Linear codes are a subclass of all codes which can be defined with symbols chosen from a set of arbitrary size. For a linear code, the sum of two codewords is also a codeword. The first channel coding scheme was proposed by Richard Hamming in 1950, is called the Hamming code. Hamming codes are linear

block codes with the capabililty of correctiong a single error. The Hamming code has low complexity implementation and mild error correcting performance.

Almost a decade after the Hamming codes, the BCH codes and the Reed-Solomon codes were invented. Both the codes have uses in space and satellite communications, data transmissions, data storage and bar codes. The early error correcting code applications were in space and satellite communication for power limited and low spectral efficiency applications. One of the earliest practical applications of error correcting codes were the use of Reed Muller codes in the 1969 Mariner and later the Viking Mars missions to improve uncoded BPSK modulation. With the invention of Trellis decoding for convolutional codes in 1960, it was possible to use powerful codes providing substantial coding gains over uncoded transmission that were used for space and satellite communication by NASA [3].

In 1991 a powerful new class of codes called the Turbo codes were invented to approach Shannon's theoretical capacity limit. Turbo codes are high performance codes with applications in reliable data communication. Turbo codes are widely used in digital systems including 3G and 4G mobile communications, satellite communication, and IEEE 802.16 standards for broadband communications [4–6]. Soon after the Turbo codes, Low Density Parity Check (LDPC) codes were rediscovered by MacKay- who showed that LDPC codes reach Shannon's capacity like the Turbo codes on the AWGN channel [7].

1.2 Low Density Parity Check (LDPC) codes

Low Density Parity Check (LDPC) codes are a class of linear block codes invented by Robert Gallager in 1963 [8]. LDPC codes are widely used in communication systems for their excellent decoding performance. However, because of their large structure and complexity of decoding algorithms, LDPC codes were not practical to implement at the time. After their rediscovery in the mid-nineties, LDPC codes have gained popularity and are used in parallel with, or often preferred over Turbo codes for their low complexity implementation and low error rate. Among many uses of LDPC codes, some are data storage, digital broadcasting, IEEE 802.3an standards for 10GBase-T ethernet, IEEE 802.11 standards for Wi-Fi [9,10]. A k bit information $\vec{u}$ is encoded by multiplying with the generator matrix G

of dimension (k, n) of the LDPC code. The codeword $\vec{c}$ is n bit long with $(n-k)$ redundant bits corresponding the parity check rules of the LDPC code.

$$\vec{c} = \vec{u}G \tag{1.1}$$

The parity check matrix H of the LDPC code is matrix with $n-k$ rows and n columns such that

$$GH^T = 0 \tag{1.2}$$

An LDPC code is characterized by it's parity check matrix (H), which can be represented by a bipartite graph called a Tanner graph consisting of nodes what are called symbol nodes and parity check nodes of the LDPC code. The H matrix is sparse, and the dimension of the H matrix is usually large. For a given code, the columns of the H matrix denote the variable nodes, and the rows denote the check nodes in the tanner graph. A 1 in the H matrix imply a connection between a check node and a variable node. Let us consider the parity check matrix H and the associated tanner graph in Fig. 1.1. The i$^{\text{th}}$ variable node is labeled as v_i and the j$^{\text{th}}$ check node is labeled as c_j.

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{1.3}$$

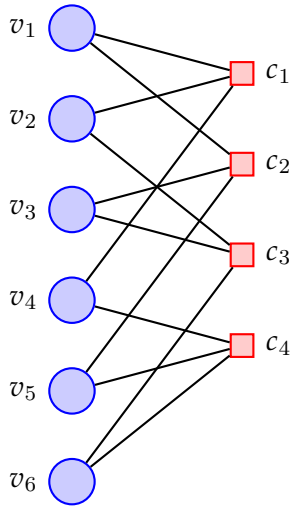


Fig. 1.1: The Tanner graph for the H matrix in (1.3). The i$^{\text{th}}$ variable node is labeled as v_i and the j$^{\text{th}}$ check node is labeled as c_j. An edge between a variable node v_i and a check node c_j implies a 1 in the j$^{\text{th}}$ row and i$^{\text{th}}$ column of the H matrix.

Decoding algorithms for LDPC codes are usually iterative in nature, where messages are sent back and forth between the symbol nodes and parity check nodes represented in a Tanner graph. Besides their structure, performance of LDPC codes depends to a large degree on the algorithms that are deployed in the receiver to decode the originally transmitted bits. The performance of the code and the algorithm are usually represented by an SNR vs BER (bit error rate) graph, as shown in Fig. 1.2 for the PEGReg504x1008 LDPC code with Belief Propagation (BP) and Normalized Min-Sum (NMS) decoding algorithms.

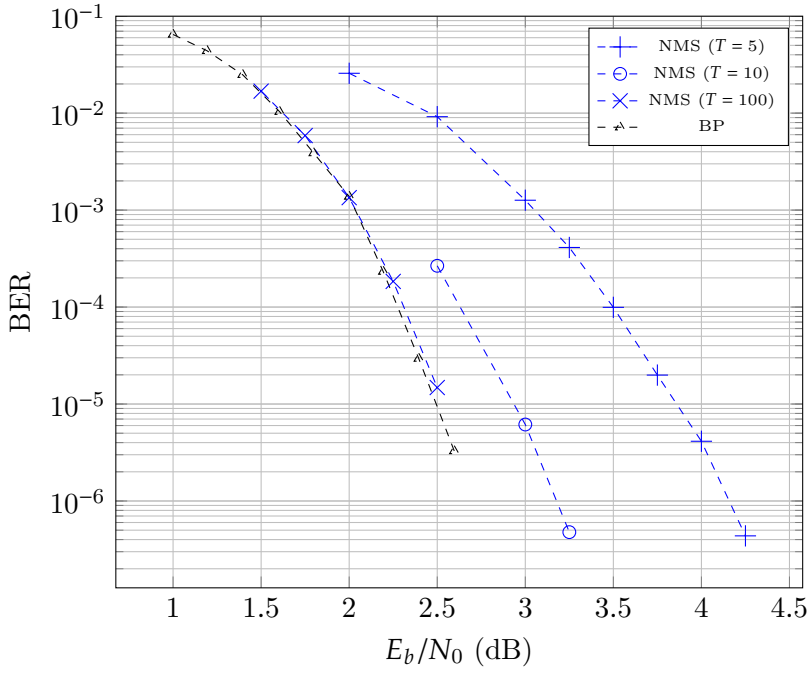


Fig. 1.2: BER results for the PEGReg504x1008 LDPC code for Belief Propagation (BP) and Normalized Min-Sum (NMS) with different number of iterations.

Bit-flip decoding algorithms are reduced complexity decoding algorithms, where messages exchanged between the nodes in the tanner graph are binary valued. Bit-flip decoding algorithms are often inferior in terms of BER performance compared to traditional Belief Propagation and its variants, but offers significant gain in hardware complexity and speed. Wadayama et.al proposed a low complexity algorithm Gradient Descent Bit-Flip (GDBF) decoding in [11]. A significant improvement over the GDBF algorithm is the Noisy Gradient Descent Bit-Flip (NGDBF) algorithm, that was proposed by Sundararanjan et al. [12]. The NGDBF algorithm used random noise to perturb the decisions in the GDBF algorithm to achieve superior performance. A few variants of the NGDBF algorithm was presented in the original paper, including the Single-bit NGDBF (S-NGDBF), Multi-bit NGDBF (M-NGDBF), and with a post-processing operation called the Smoothed-Multi-bit NGDBF (SM-NGDBF).

It is shown in [13] that re-decoding a frame with NGDBF for a number of phases further improves the decoding. Figure 1.3 shows the comparative performance of GDBF, NGDBF,

and Re-decoded NGDBF (R-NGDBF) with traditional Min-Sum (MS), Weighted Bit-Flip (WBF) algorithms.

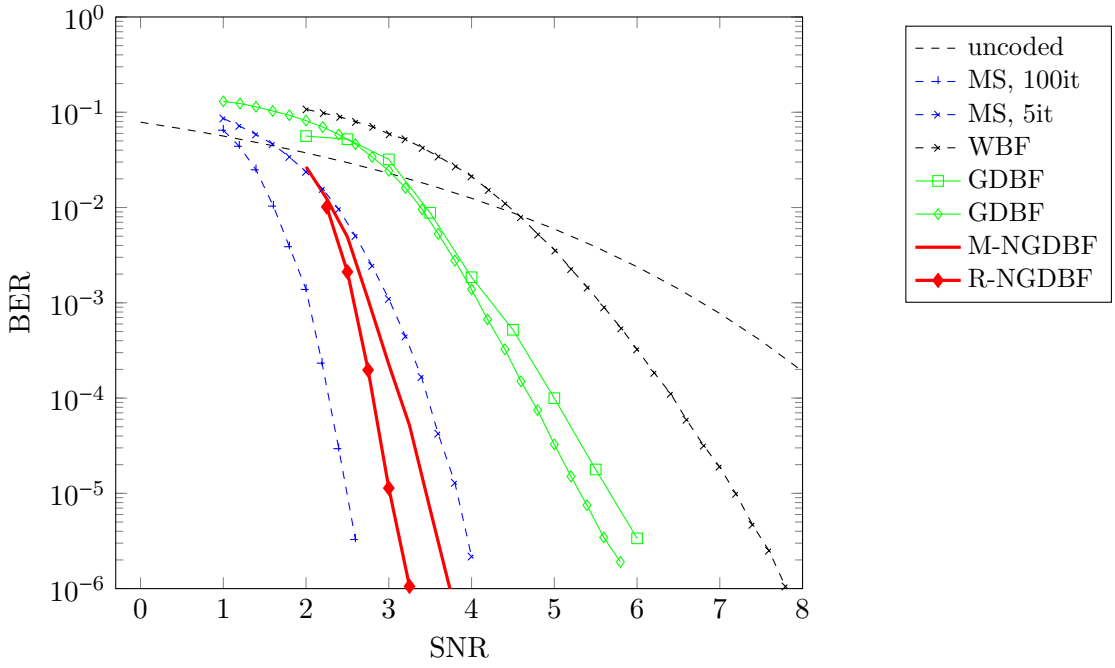


Fig. 1.3: Comparative performance of GDBF, Multi-bit NGDBF (M-NGDBF), and Re-decoded NGDBF (R-NGDBF) with traditional Min-Sum (MS) with 5 and 100 iterations, Weighted Bit-Flip (WBF) algorithms on the PEGReg504x1008 code.

As seen in Fig. 1.2 and Fig. 1.3, the BER rapidly goes down with increasing SNR, which is known as the waterfall region of the graph, followed by a saturation in the improvement where the BER no longer goes down with increasing SNR. This region is called the error-floor of the graph.

1.3 Trapping sets and absorbing sets of LDPC codes

Despite their excellent performance in the waterfall region of the BER curve, the performance of LDPC codes are limited by error-floors at the bottom the waterfall region, where the BER no longer improves with increasing SNR. In the error-floor region, iterative decoders are trapped in subgraphs of the code called the “trapping sets”, which causes the decoders to remain in erroneous states after the maximum number of iterations in the decod-

ing algorithm. Richardson [14] showed that for the AWGN channel and BSC the error-floor region for a rate 0.7969 code of length 4096, all the errors were due to the near-codewords, which are known as "trapping sets". We define trapping sets according to [14].

Trapping sets: For a given input y, let $\mathbf{T}(y)$ be the set of variables nodes that are not eventually corrected by iterative decoding. Therefore, a successful decoding is indicated by $\mathbf{T}(y) = \emptyset$. If $\mathbf{T}(y) \neq \emptyset$, $\mathbf{T}$ is a (a, b) trapping set where a is the number of variable nodes and b is the number of odd degree check nodes in the subgraph $\mathcal{T}$ induced by $\mathbf{T}$. In the subgraph, the check nodes are usually only degree one or degree two check nodes. Higher degree check nodes are possible but unlikely in dominant trapping sets. A related concept "absorbing sets" were described in [15, 16].

Absorbing sets: In [15], Zhang et.al. introduced absorbing sets to describe the failure of iterative decoders. In the experimental work, a hardware emulator was analyzed for various classes of structured LDPC codes. It was found that a set of graphical structures referred to as absorbing caused message passing decoders to fail by converging on non-codewords. Absorbing sets may be viewed as a special case of trapping sets, which is guaranteed to be stable under a bit-flipping decoder. Absorbing sets are purely combinatorial like stopping sets. Absorbing sets are defined as follows:

Let H be the parity check matrix and $\mathcal{T}(H)$ be the tanner graph induced by H. Let a be a subset of variable nodes in the tanner graph and b be the set of their neighboring odd-degree check nodes in the subgraph induced by a. Each of the variable nodes in a are connected to more even degree check nodes than odd degree nodes, and all the remaining check nodes outside the induced subgraph are even degree with respect to $\mathcal{T}(H)$. In that case (a, b) is an absorbing set if for all a', $a' < a$, it does not contain an (a', b) set as its subgraph.

The (8,8) absorbing set of (2048,1723) Reed-Solomon LDPC code: The 802.3an LDPC code is a high performance (2048,1723) Reed-Solomon LDPC code. It was found that the error-floor of the (2048,1723) LDPC code is dominated by an (8,8) fully absorbing set that has the structure showed in Fig. 1.4.

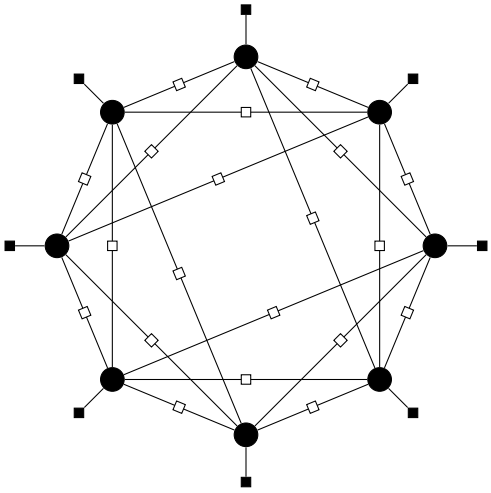


Fig. 1.4: The dominant $(8, 8)$ absorbing set in the 802.3an 10GBASE-T LDPC code.

1.4 The GDBF and the NGDBF decoding algorithm

Wadayama et al. proposed the GDBF algorithm for decoding LDPC codes [11]. At each iteration of the GDBF algorithm, an inversion function for each bit is based on the received channel information, decision at the previous iteration, and parity check messages from the neighbors of the node. The Noisy Gradient Descent Bit-Flip (NGDBF) [12] modifies GDBF by adding a pseudo-random perturbation to the inversion function at each symbol node during each iteration. The noise is used to probabilistically escape from the trapping sets and lower the error-floor for a given LDPC code. Let the received message vector be denoted by $\vec{y}$. The neighborhood of a node refers to all the other nodes that are connected to it. We denote the neighborhood of the i^{th} check node by $\mathcal{N}(i) \triangleq \{j : h_{ij} = 1\}$, and the neighborhood of the j^{th} symbol node by $\mathcal{M}(j) \triangleq \{i : h_{ij} = 1\}$; where h_{ij} is an element of the parity check matrix H.

At each iteration of the NGDBF algorithm, the energy of each bit is calculated according to (1.4),

$$E_k = x_k y_k + w_k \sum_{i \in \mathcal{M}(k)} s_i + q_k \tag{1.4}$$

—where the decision vector at each iteration is denoted by $\vec{x} \in \{-1, 1\}^n$. If $\vec{x}$ is a valid codeword and $\vec{x} \in \hat{C}$, all the parity checks would be satisfied. The i^{th} parity check operation is denoted by $s_i \triangleq \prod_{j \in \mathcal{N}(i)} x_j$. Therefore, a satisfied parity check equation refers to $s_i = 1$. The parameter w_k is predefined weight parameter to scale the sum of the parity check operations. If at iteration t, all the parity checks are satisfied, then $\vec{x}(t)$ is declared as a valid codeword and decoding is terminated. The term q_k is the perturbation noise used by NGDBF which is independently distributed Gaussian random noise samples with zero mean and variance equal to $\sigma^2 = \eta^2 N_0/2$ — proportional to the variance of the channel noise — where $\eta \geq 0$ is a noise-scale parameter. In all algorithms, unsuccessful frames are terminated after a maximum number of iterations T. If the energy of the bit is below a threshold parameter θ, the bit is flipped. Because a number of bits may have energy below the threshold, and subsequently be flipped, this version of the NGDBF algorithm is termed as the Multi-bit NGDBF (M-NGDBF). The M-NGDBF algorithm is outlined as follows:

Step 0: Initialize $\theta_k\,(t = 0) = \theta$ for all k, where θ is the global initial threshold parameter. Optionally saturate sample magnitudes at $y_{\max}$ and set $\vec{x} = \text{sign}\,(\vec{y})$.

Step 1: Compute syndrome components:

$$s_i = \prod_{j \in \mathcal{N}(i)} x_j, \tag{1.5}$$

for all $i \in \{1, 2, ..., m\}$. If $s_i = +1$ for all i, output x and stop.

Step 2: Compute inversion functions:

$$E_k = x_k y_k + w_k \sum_{i \in \mathcal{M}(k)} s_i + q_k \tag{1.6}$$

for $k \in \{1, 2, \ldots, n\}$. where w_k is a syndrome weight parameter and q_k is a Gaussian distributed random variable with zero mean and variance

$\sigma^2 = \eta^2 N_0/2$, where $0 < \eta \leq 1$. All q_k are independent and identically distributed.

Step 3: Bit-flip operations: if $E_k(t) < \theta_k(t)$ then $x_k(t+1) = -x_k(t)$, otherwise $\theta_k(t+1) = \lambda \theta_k(t)$,

where λ is a global threshold adaptation parameter for which $0 < \lambda \leq 1$.

Step 4: Repeat steps 1 to 3 until all $s_i = 1$, for $i = 1, 2, \ldots, m$, or maximum number of iterations T is reached.

Threshold adaptation and Output decision smoothing in Multi-bit NGDBF (M-NGDBF): The authors proposed a threshold adaptation method to improve the convergence of M-NGDBF. Each bit has a threshold θ_k, which is initialized at a global value θ. The threshold θ is then adjusted for each bit at each iteration by a parameter λ. The following steps are added to NGDBF to incorporate threshold adaptation:

- Initialize (t = 0) $\theta_k = \theta$ for all bits, where $\theta \in R$ is the global initial threshold parameter.
- For a bit k, at iteration t, if the energy function $E_k \geq \theta_k(t)$, make the adjustment in $\theta_k(t+1)$ to be used in the next iteration $t+1$ as follows:

$$\theta_k(t+1) = \theta_k(t)\lambda \tag{1.7}$$

where λ is a global adaptation parameter for which $0 < \lambda \leq 1$. In the next iteration $t+1$, the adjusted threshold $\theta_k(t+1)$ is used. If $E_k(t) < \theta_k(t)$, flip the sign of the corresponding decision x_k

1.5 Contributions of this dissertation:

This dissertation started as an parameter optimization project for the original NGDBF decoding algorithm [12]. The primary goal was to optimize the parameters of NGDBF such as noise-scale (η) and weight (w) to lower the error-floors for the PEGReg504×1008 and

the IEEE 802.3 LDPC codes. Subsequently a faster method to analyze and simulate the NGDBF decoding algorithm for the low error-floor was proposed. The results were then validated in Xilinx FPGA platform. The contributions of this dissertation are arranged as follows:

The first contribution to the improvement of NGDBF, the Re-decoded NGDBF (R-NGDBF) is described in Ch. 2. The proposed technique involves re-decoding of failed frames, after a number of decoding iterations have been observed with no success. In Ch. 3 a Markov chain analysis is adopted to study the decoding trajectory of NGDBF. The contributions of the random perturbations of NGDBF to escape the dominant absorbing set is discussed at length. The error-floor is explained as a consequence of the (dominant) absorbing sets that are not resolved by NGDBF decoding algorithm. Based on the Markov chain analysis, a simplified decoding approach is taken to estimate the low error-floors of the NGDBF algorithm by simulation in Ch. 4. The estimated parameters are verified in FPGA implementation in Ch. 5. The FPGA implementation includes a 5-bit resolution decoder, which is fully synthesized, implemented and validated in Xilinx platform.

CHAPTER 2

PROPOSED RE-DECODING FOR NGDBF

2.1 Re-decoding background

In this chapter, we consider the performance of the Noisy Gradient Descent Bit-Flip (NGDBF) decoding algorithm under re-decoding of failed frames. The proposed re-decode procedure obtains improved performance because the perturbations are independent at each re-decoding phase, therefore increasing the likelihood of successful decoding. We examine the benefits of re-decoding for an LDPC code from the IEEE 802.3an standard, and find that only a small fraction of re-decoded frames are needed to obtain significant performance benefits. When re-decoding is used, the NGDBF performance is very close to a benchmark offset min-sum decoder for the 802.3an code.

The recently proposed NGDBF algorithm provides superior performance by injecting random noise into the GDBF inversion function [17]. The noise perturbation is hypothesized to disrupt the activity around trapping sets, so that the decoder has a chance to escape. In this chapter, we observe that failed NGDBF frames can often be correctly decoded by re-decoding the algorithm from its initial state. Re-decoding is shown through simulations to be more effective than increasing the iterations. To explain this result, we hypothesize that the noise perturbations may sometimes stimulate new trapping set conditions that are less likely to be escaped. Because the NGDBF algorithm is non-deterministic, these conditions will not necessarily recur when decoding is performed a second time.

Re-decoding randomizes the decoding trajectory in the Tanner graph and provides a way for the decoder to converge towards an actual codeword. Re-decoding from the same initial condition was considered previously for stochastic decoders in [18, 19], where it was proposed as a method to evade trapping sets. In [20], a two-phase decoding is proposed for stochastic decoders. The frames that are failed after the 1^{st} phase are decoded again with

a modified algorithm in the 2nd phase called the "VN harmonization". VN harmonization changes the log-likelihood ratio (LLR) input to each variable node by a constant. This constant must be found empirically.

2.2 Related work on Re-decoding for decoding of ECC

In [21], two phase decoding has been applied to the Belief Propagation (BP) algorithm. The Normalized Min-Sum (NMS) algorithm is employed in the 1st phase. If a valid codeword is not found, the failed frames enter the 2nd phase. Two approaches are discussed for the 2nd phase: Random Sign Flip (RSF) and Random Initial State (RIS). RSF randomly flips the sign of the outgoing check node messages with a probability p $(0 \leq p \leq 1)$. RIS applies random changes to the initial channel data externally, by adding random vectors to the initial channel values. A similar re-launching process is described for LDPC decoders called the Improved Differential Binary (IDB) message passing decoding in [22]. The re-launching process divides the decoding process into multiple phases. The IDB algorithm uses a look-up table approach, where the initial channel values are modified in a deterministic manner at the start of a given phase according to a heuristic table.

2.3 Re-decoding in the context of NGDBF

The previous works demonstrated BER improvement due to re-decoding, but did not provide a detailed inspection of trapping set behavior. In this chapter, we present some experiments on the (8,8) absorbing set known to be dominant in the 802.3an 10GBASE-T standard LDPC code under Belief Propagation [23]. The induced graph for this set is shown in Fig. 2.1, where the degree-one check nodes are indicated as ■, degree-two check nodes as □, and symbol nodes as ●. While a full trapping set analysis has not yet been developed for NGDBF, in this chapter we verify that the $(8, 8)$ set acts as a trapping set for GDBF and NGDBF, and we inspect the dynamics that allow NGDBF to evade the trapping set during decoding.

To investigate NGDBF dynamics on this absorbing set, a localized simulation was performed on the $(8, 8)$ subgraph. The correct state is assumed to be $\hat{c} = (+1\ +1 \cdots +1)$.

The GDBF and NGDBF algorithms were simulated with identical inputs $\vec{y} = \hat{c} + \vec{z}$, where $\vec{z}$ is a vector of zero-mean Gaussian noise samples with noise variance $\sigma^2 = 1$. The simulations were performed with parameters $\lambda = 1$ (i.e. without threshold adaptation), $w = 1$, $T = 100$ and for NGDBF $\eta = 1$. In these simulations, a frame was considered successful if the correct result, $\vec{x} = \hat{c}$, was obtained for at least one iteration. Failed frames were saved for detailed inspection.

It was found that failed frames typically begin in a metastable initial condition, where one or two early flips determine the ultimate trajectory. Fig. 2.2 shows the trajectory of inversion functions for a case in which GDBF becomes trapped in an oscillating cycle, but NGDBF avoids the oscillation due to a fortuitous early flip. In the early iterations, some of the E_k are negative or weakly positive, so they are likely to be flipped. In later iterations, most of the NGDBF E_k values are strongly positive, so additional flips are unlikely in spite of the noise perturbations.

Fig. 2.3 shows a case where NGDBF failed due to an errant early flip. In this case, NGDBF eventually converged on an all-error state with positive E_k. Because the E_k are positive, future flips are unlikely to occur and the error state is effectively stable. Fig. 2.4 shows a repeated simulation from the same initial condition. In this case, NGDBF made different flips in the first five iterations, and converged on the correct state. This example demonstrates advantages of re-decoding from the same initial state, which cannot be achieved by extending the simulation time. The benefits of re-decoding are more pronounced when threshold adaptation is used, since the evolving thresholds tend to harden the stability of the final state, thereby lowering the probability that NGDBF will escape to the correct state if given more iterations.

These experiments were repeated for several values of σ, and NGDBF was found to have a consistently lower rate of converging on an erroneous state compared to GDBF. The simulation method used here is illuminating about the dynamics and provides motivation for re-decoding, but it is not sufficient to quantify the frame error probability associated with this absorbing set. For $\sigma < 0.7$, we did not obtain any failed cases for NGDBF. In

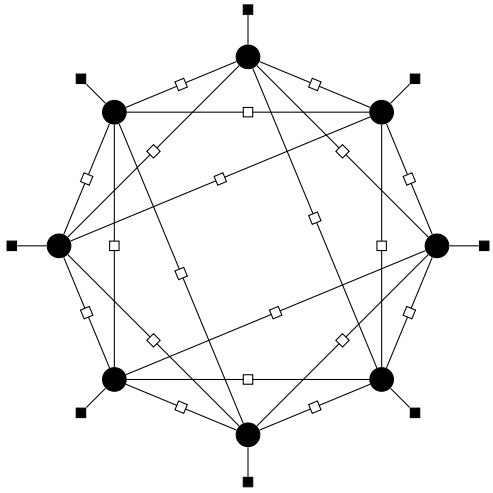


Fig. 2.1: The dominant $(8, 8)$ absorbing set in the 802.3an 10GBASE-T LDPC code.

the sequel, we evaluate the re-decoding method for two practical codes, and show that significant performance benefits are obtained.

2.4 Proposed Re-decoded NGDBF (R-NGDBF)

All the methods discussed above try to change the initial state of the decoder either in a random or in a deterministic manner to improve convergence during a decoding process. Random Initial State (RIS) and Stochastic decoding are random processes, while the re-launching technique is a deterministic process. We leverage the randomness involved in the NGDBF algorithm. Re-decoding a failed frame causes the different stream of random noise to be to used during the decoding process which increases the likelihood of successful convergence. The decoding process is repeated for a number of phases with the original received message, with added perturbations to the inversion function at each phase. As the random perturbations of NGDBF at each phase are independent of each other, the decoding is likely to follow a different path and arrive at different results at each phase, similar to stochastic decoders. This modified algorithm, is referred to as the Re-decoded NGDBF (R-NGDBF) algorithm. The R-NGDBF has two variants: Re-decoded M-NGDBF (R-M-NGDBF) and the Re-decoded SM-NGDBF(R-SM-NGDBF).

The number of phases is limited to a maximum value Φ. Φ is an important parameter of

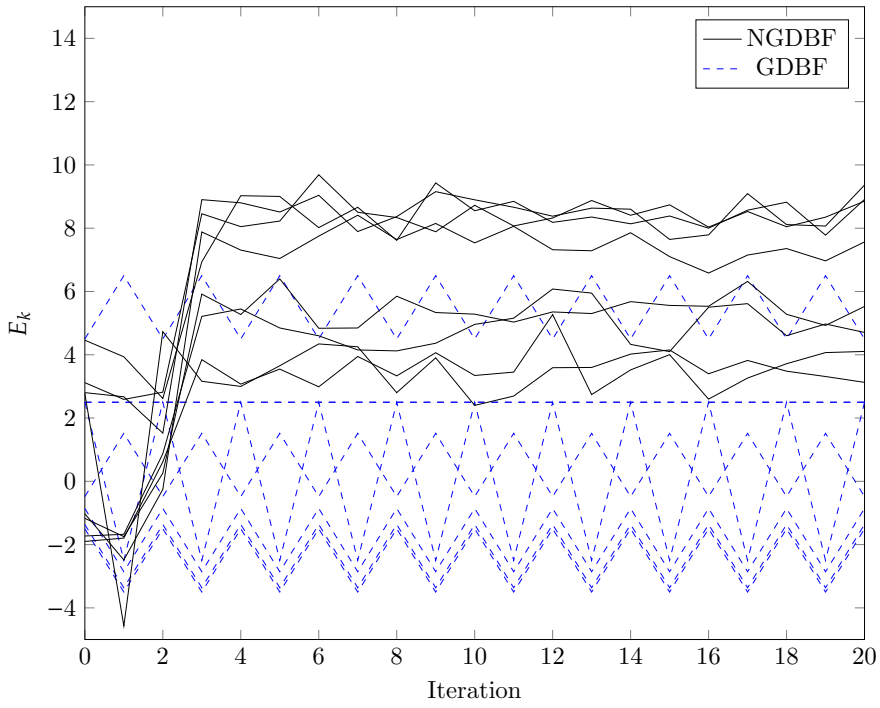


Fig. 2.2: A typical case where GDBF is trapped but NGDBF escapes due to random perturbations.

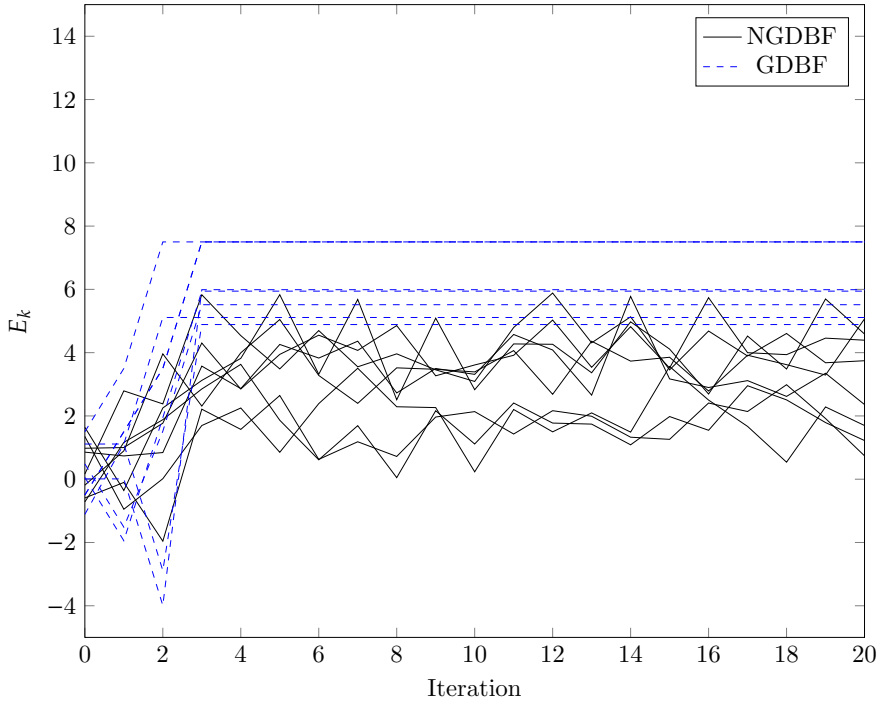


Fig. 2.3: A case where NGDBF settles on an all-error pattern on the $(8, 8)$ absorbing set. Error propagation is triggered by a single errant bit-flip that occurs in the first five iterations.

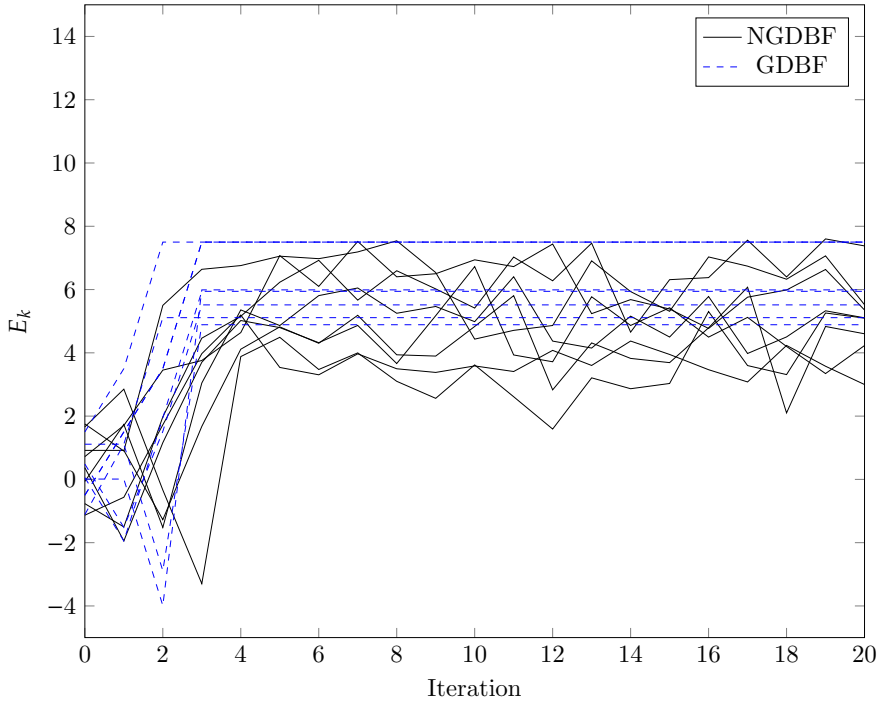


Fig. 2.4: A re-decoded case with the same initial conditions as Fig. 2.3. This time NGDBF evades the erroneous state and corrects all errors.

this algorithm. The convergence is expected to improve as Φ is increased. If a valid codeword is not found by phase Φ, decoding failure is declared. The R-M-NGDBF algorithm can be summarized as follows:

1. Initialize all the symbol nodes to the received channel message vector, and set `phase=1`.
2. Perform decoding operation using M-NGDBF.
3. Set `phase = phase+1`.
4. Repeat steps 2 and 3 with initial values of received vector y_k and with different q_k until all the parity checks are satisfied or `phase =` Φ is reached.

2.5 Simulation results

Simulations were performed using re-decoding with NGDBF for two codes: the rate 1/2 regular $(3, 6)$ LDPC code identified as PEGReg504x1008 in MacKay's online encyclopedia of sparse graph codes [24] and the rate 0.8413 LDPC code defined in IEEE 802.3an standard.

The smoothing and threshold adaptation heuristics are used for the PEGReg504x1008 code, and the algorithm name is indicated as SM-NGDBF in reported simulations. For the 802.3an code, these heuristics were not used. Each frame was allowed to be simulated up to a maximum number Φ of re-decoding phases. At least 200 bit errors and at least 20 word errors were observed to obtain the BER measurements for each simulation.

For the PEGReg504x1008 code, SM-NGDBF was simulated with parameters $T = 300$, $w = 0.816$, $\lambda = 0.98$, initial threshold $\theta = -0.6$ and noise-scale $\eta = 0.75$. Fig. 2.5 shows BER performance results for the PEGReg504x1008 code with $\Phi = 10$. The re-decoding technique has significant gain when applied to the SM-NGDBF algorithm. At BER= 10^{-6}, re-decoding provides a gain of about 0.5 dB. Output smoothing is only used for iterations exceeding $(T - 64)$. Performance of NMS and BP algorithms are presented for comparison. The BER improves with higher values of Φ, but there is a diminishing benefit as Φ is increased. As seen in Fig. 2.6, there is a rapid improvement in performance from $\Phi = 1$ through $\Phi = 5$. As Φ is increased further, the improvement in BER performance becomes less significant. There is slight improvement in $\Phi = 10$ compared to the improvement in the earlier phases.

The simulations show that re-decoding is necessary for a small fraction of frames. Fig. 2.7 shows the distribution of re-decoding phases to complete decoding. Most of the failed frames that are not corrected in the first decoding phase are corrected by the second phase. The frames that are not corrected by the second phase are passed onto the third phase and so forth. Finally, at the last phase, the accumulated failed frames determine the word error rate (WER). This accumulation is evident in the last phase in Fig. 2.7.

Fig. 2.8 shows the BER performance results for NGDBF on the IEEE 802.3an standard LDPC code. Smoothing is not used for simulations with this code, because no significant improvement was achieved using smoothing in this case. The simulation parameters are $T = 1000$, $w = 0.20833$, $\lambda = 1$, $\theta = -0.525$, and $\eta = 0.92$. Since $\lambda = 1$, threshold adaptation is not required for this code. To evaluate the performance we use a recently reported 802.3an Offset Min-Sum (OMS) decoder as a benchmark [25]. Re-decoding provides a gain of 0.25

dB for this code. The BER performance of NGDBF is very close to the benchmark OMS decoder.

2.6 Latency considerations of R-NGDBF

Since re-decoding incurs a substantial latency penalty, we examined the average latency on the 802.3an code. If buffering can be tolerated by the end application, then the average latency penalty is very small since nearly all frames are successfully decoded in the first phase. Fig. 2.9 shows the average latency in terms of clock cycles for all simulated cases, in comparison to the reported latency of the OMS decoder. The OMS decoder uses a semi parallel layered architecture which requires 12 clock cycles to complete an iteration. The reported average number of iterations is therefore scaled by 12 to obtain the average latency depicted in Fig. 2.9. The NGDBF decoder has a much lower complexity than the OMS algorithm, so layering is not required and we can expect every iteration to complete in a single clock cycle. The number of iterations is therefore equivalent to the latency in the NGDBF case. Fig. 2.9 shows that the average latency for NGDBF is quite large at low SNR values, but decreases at higher SNRs where it has a lower latency than OMS. When operating at higher SNR values, Re-decoded NGDBF (R-NGDBF) offers better performance and lower average latency than the benchmark design. To account for consecutive worst case frames, the NGDBF decoder would require a larger frame buffer compared to the OMS decoder. This is a potential drawback of re-decoding with the NGDBF algorithm.

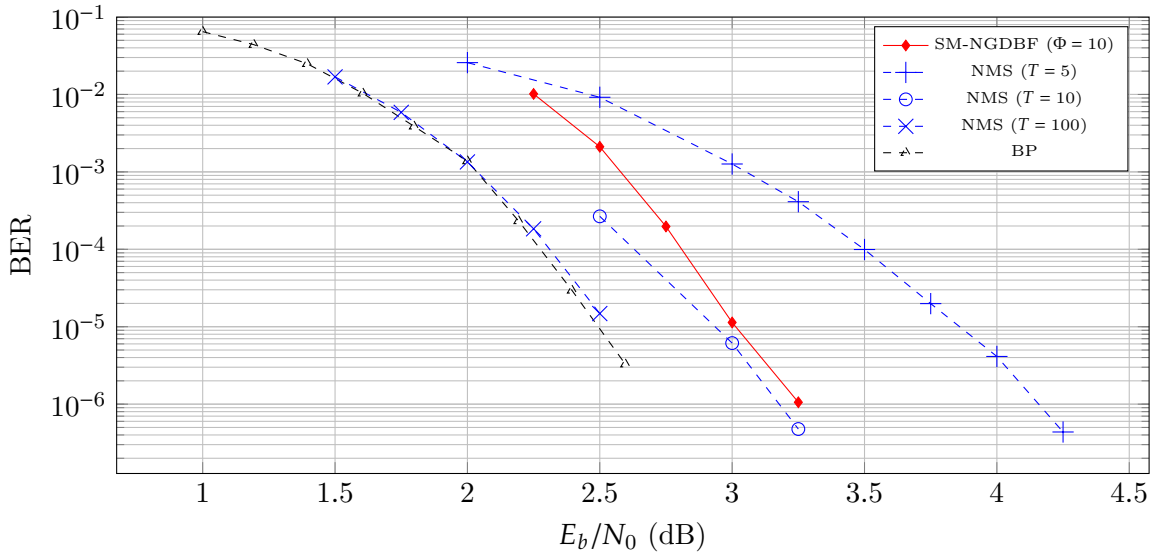


Fig. 2.5: BER results for the PEGReg504x1008 code. Results for Belief Propagation (BP) and Normalized Min-Sum (NMS) with different iterations are provided for comparison.

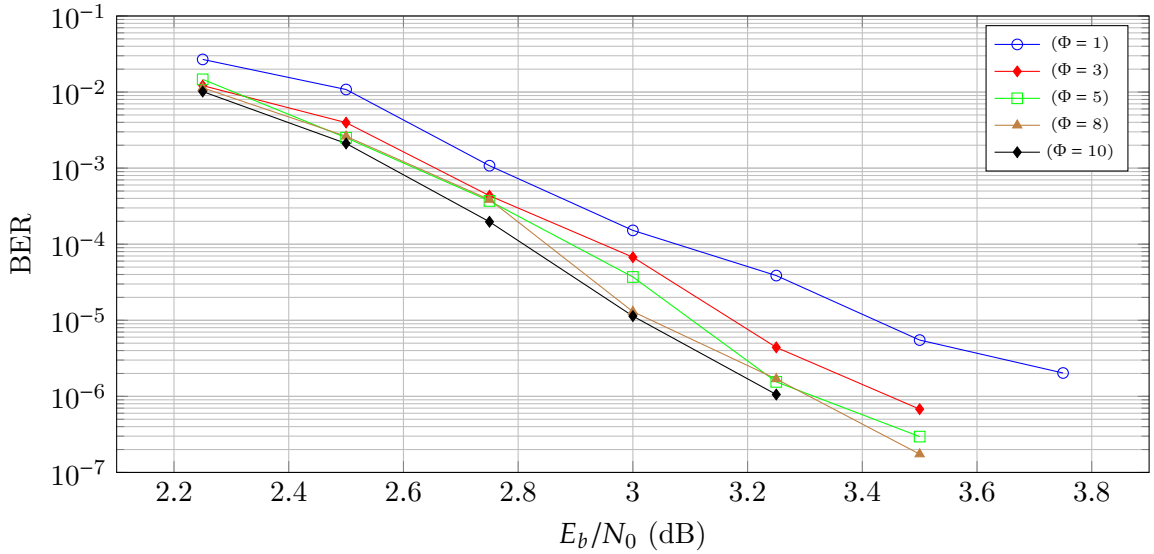


Fig. 2.6: BER for re-decoding with the SM-NGDBF on the PEGReg504x1008 code for different Φs.

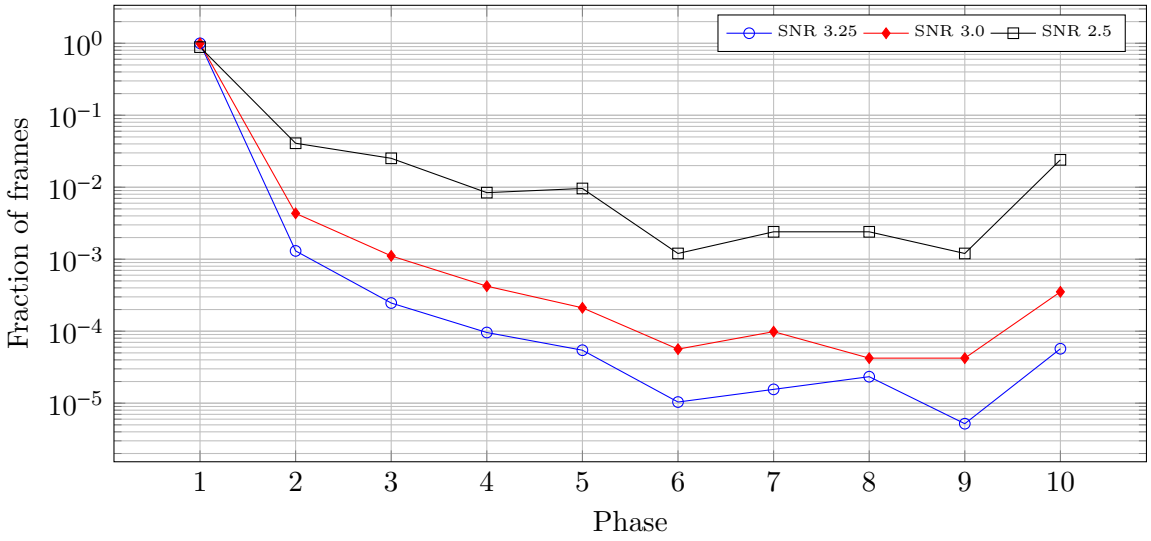


Fig. 2.7: A histogram showing the fraction of frames completed at each decoding phase for SM-NGDBF on the PEGReg504x1008 code, $\Phi = 10$. The increase of frames at the last phase arises due to the accumulation of failed frames.

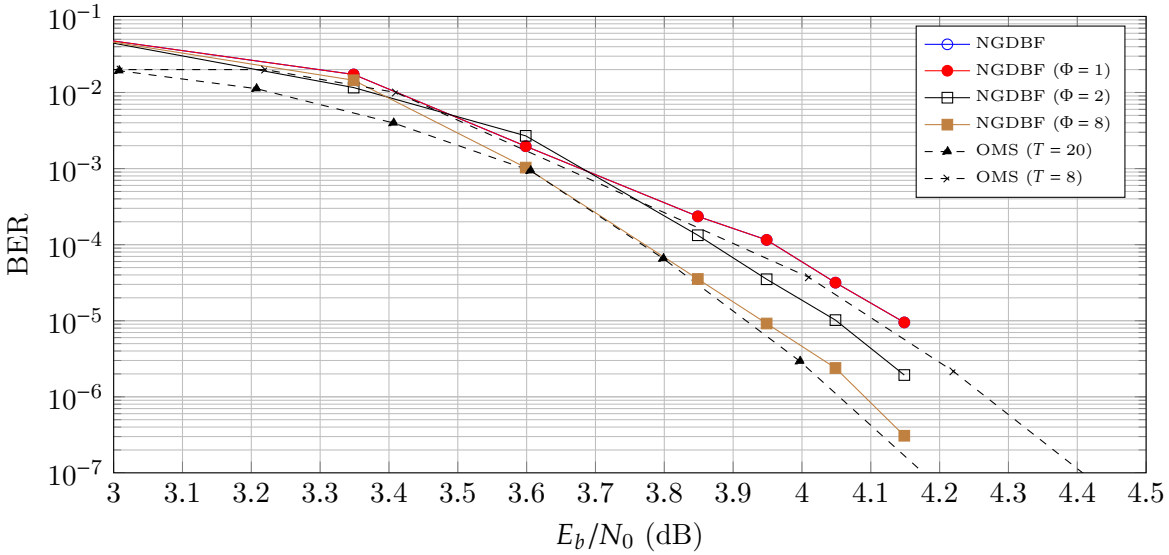


Fig. 2.8: BER for Re-decoded NGDBF compared to a benchmark OMS decoder for the IEEE 802.3an standard LDPC code.

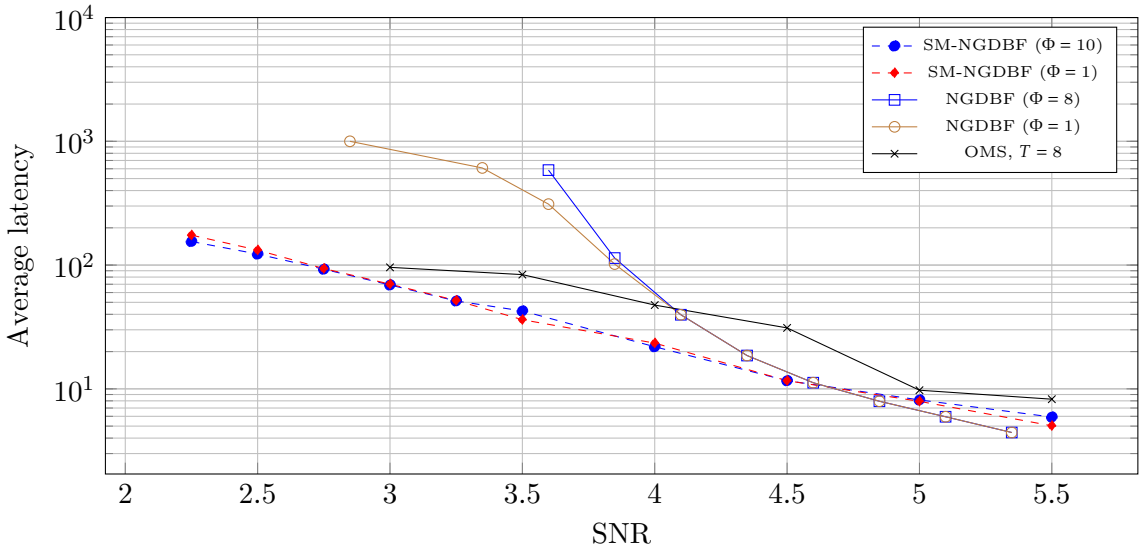


Fig. 2.9: Latency comparison between different algorithms and codes. The dashed lines indicate simulations on the PEGReg504x1008 code, and solid lines indicate simulations on the IEEE 802.3an standard LDPC code.

CHAPTER 3

ANALYSIS OF TRAPPING SETS

3.1 Analysis of trapping sets for the GDBF and NGDBF algorithm

Let us consider an example case to illustrate how an error is trapped in a trapping set for the GDBF and the NGDBF algorithm. Consider the (3,3) trapping set in Fig. 3.1. The graph induced by the trapping set is denoted by $\mathcal{T}_3$. The variable nodes in the trapping set are $\mathcal{X} = \{x_1, x_2, x_3\}$ with degree two check nodes c_1, c_2, c_3 and degree one check nodes c_4, c_5, c_6. The set of all the check nodes is denoted as $\mathcal{C} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$. In Fig. 3.1 the variable nodes are represented as ●, degree-one check nodes as ■, and degree-two check nodes as □.

We assume all-zero codeword transmission. Therefore the codeword for the example is $\vec{c} \in \{0\}$. For AWGN channel $\vec{c}$ is mapped to $\hat{c} \in \{1\}$. The transmitted codeword is corrupted by Gaussian noise with zero mean and variance of $\sigma_z^2 = N_0/2$, where N_0 is the noise spectral density. Therefore, the received signal is $\vec{y} = \hat{c} + \vec{z}$, which is to be decoded. We are only interested in the subgraph $\mathcal{T}_3$ generated by the (3,3) trapping set. We denote the subset of incoming channel messages to the trapping set $\mathcal{T}_3$ as $\vec{y}_T$. For example, let $\vec{y}_T = \{-1.3, 0.2, -0.5\}$.

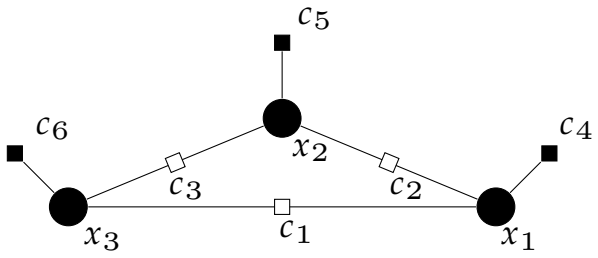


Fig. 3.1: The $(3, 3)$ absorbing set in the 802.3an 10GBASE-T LDPC code.

The initial value of the variable nodes $\mathcal{X} = \{x_1, x_2, x_3\}$ are $\{-1, +1, -1\}$. In GDBF algorithm, the check nodes update by multiplying all the incoming decisions from the neighboring variable nodes. Let the messages from check node c_i to its neighboring variable nodes be denoted by s_i. Check node c_1 is connected to x_1 and x_3.

$\therefore s_1 = (-1) \times (-1) = +1$. Similarly, $\{s_2, s_3, s_4, s_5, s_6\} = \{-1, -1, -1, +1, -1\}$.

In the GDBF algorithm, the enrgy of a bit is calculated as

$$E_i = x_i y_{Ti} + \sum_{j:j \in \mathcal{N}(x_i)} (s_j) \tag{3.1}$$

where the neighbors of variable node v_i are $\mathcal{N}(x_i)$. The incoming message from the neighboring check nodes $\mathcal{N}(x_i)$ of variable node x_i is denoted by s_j, i.e $j \in N(x_i)$.

After the first iteration the energies of the variable nodes are as follows for the GDBF algorithm:

$$E_1 = (-1)(-1.3) + 1 - 1 - 1 = -0.3 \tag{3.2}$$

$$E_2 = (+1)(+0.2) - 1 - 1 + 1 = -0.8 \tag{3.3}$$

$$E_3 = (-1)(-0.5) + 1 - 1 - 1 = -0.5 \tag{3.4}$$

In the GDBF algorithm, the bit with most negative energy value is flipped. Therefore state of node x_2 is flipped from $+1$ to -1. As a result, all the decision bits after the first iteration are $\{-1, -1, -1\}$ and all the variable nodes in the $\mathcal{T}_3$ are in error.

The NGDBF algorithm introduces random perturbation noise q_i to the energy values to perturb the decisions and provide means to escape the trapping set. In NGDBF, the energies of the variable nodes are calculated as

$$E_i = x_i y_{Ti} + \frac{1}{w}\sum_j (s_j) + q_i; \tag{3.5}$$

$$\text{where, } q_i \sim \mathcal{N}(0, \sigma^2), \sigma^2 = \eta \times \sigma_z^2$$

where w is called a weight parameter, and the perturbation noise variance σ^2 is calculated by scaling the channel noise variance σ_z^2 by a factor η. For now, we assume that the weight parameter is equal to unity ($w = 1$).

For the NGDBF algorithm, let the perturbation noise vector be $\vec{q} = \{q_1, q_2, q_3\} = \{-0.45, 0.51; -0.22\}$. The energies become

$$E_1 = (-1)(-1.3) + 1 - 1 - 1 + q_1 \tag{3.6}$$

$$= -0.3 - 0.45 = -0.75 \tag{3.7}$$

$$E_2 = (+1)(+0.2) - 1 - 1 + 1 + q_2 \tag{3.8}$$

$$= -0.8 + 0.51 = -0.29 \tag{3.9}$$

$$E_3 = (-1)(-0.5) + 1 - 1 - 1 + q_3 \tag{3.10}$$

$$= -0.5 + -0.22 = -0.72 \tag{3.11}$$

In the case of NGDBF the decision is flipped for variable node x_3, and the one of the faulty bits is corrected. In Multi-bit NGDBF (M-NGDBF), the filp operation at each iteration is governed by a threshold θ. All the bits that have E_i values less than θ is flipped. In the case the NGDBF algorithm, the probability that a bit will be flipped is given by

$$P_{\text{flip}} = \Pr(E_i < \theta) \tag{3.12}$$

If we let the part of the energy without noise be E'_i, i.e $E'_i = E_i - q_i$;

$$P_{\text{flip}} = \Pr(E'_i + q_i < \theta) = Q(\theta, E'_i, \sigma); \text{ where } q_i \sim \mathcal{N}(0, \sigma^2) \tag{3.13}$$

where Q is the the cumulative distribution function at θ with mean E'_i and variance σ^2.

According to (3.13), the probability that a flip will occur depends on the value of threshold θ, perturbation noise variance σ^2, and the energy E'_i calculated in the previous iteration. In the (3,3) trapping set, let E'_i be the energy of a bit x_i after the first iteration (without adding the perturbation noise q_i). For $x \in \{1, -1\}$, a total of $2^3 = 8$ possible combinations of bits can be achieved. Later, the 8 combinations are referred to 8 states of a Markov process. Each combination generates a different energy value for each bit. A energy matrix E of dimension 8×3 can be constructed with the 3 columns corresponding to energies E'_i for each bit $\{x_1, x_2, x_3\}$ and 8 rows corresponding the different combinations of the 3 bits, as shown in Tab. 3.1. Once we have the matrix E, we can compute the probability of flip for all bits under any initial condition applying (3.13) to E. The probability matrix is denoted by P_E, which contains all the probabilities that a bit will flip for a given combination of $\mathcal{X}$.

In the example, we have incoming channel messages $\vec{y}_T = \{-1.3, 0.2, -0.5\}$, to the trapping set $\mathcal{T}_3$, and we have calculated energies for each bit in $\mathcal{X}$, $\{x_1, x_2, x_3\} = \{-1, +1, -1\}$. This combination of bits are represented by the 6th row of the energy matrix. The 6th row of the energy matrix is therefore, $\{E_{61}, E_{62}, E_{63}\} = \{-0.3, -0.8, -0.5\}$. The other entries of E can be calculated for different states of $\mathcal{X}$ in a similar way.

state	Decision			Energy		
	x_1	x_2	x_3	E_1	E_2	E_3
1	1	1	1	E_{11}	E_{12}	E_{13}
2	1	1	-1	E_{21}	E_{22}	E_{23}
3	1	-1	1	E_{31}	E_{32}	E_{33}
4	1	-1	-1	E_{41}	E_{42}	E_{43}
5	-1	1	1	E_{51}	E_{52}	E_{53}
6	-1	1	-1	E_{61}	E_{62}	E_{63}
7	-1	-1	1	E_{71}	E_{72}	E_{73}
8	-1	-1	-1	E_{81}	E_{82}	E_{83}

Table 3.1: Energy values of each bit depending on the decisions

.

3.2 Markov chain analysis of trapping sets in the NGDBF algorithm

With every iteration of the NGDBF algorithm, flipping one or more bits moves the decision from one state to another state. For an n bit codeword, there are a total of 2^n states that the decision can be in. As the probability of flip is dependent on the random noise q_i, the different states of the decision can be considered as the states of a Markov chain. A similar approach has been adopted in [26] to analyze the trapping set dynamics for probabilistic decoding for the BSC channel.

By flipping one or more bits, any state of the chain can be reached from any other state. If every state of a Markov chain can be reached from every other state in one or more moves, the Markov chain is Ergodic or irreducible Markov chain. We can construct a transition matrix for the Markov chain from the probabilities of flip for each bit at every state. The transition matrix that is achieved is of Ergodic type.

To explain how the probability for transition from one state to another, we consider a transition from state 6 to state 7 from the example in Sec. 3.1. The state 6 is represented by

$\{x_1, x_2, x_3\} = \{-1, 1, -1\}$, and the state 7 is represented $\{x_1, x_2, x_3\} = \{-1, -1, 1\}$. We have calculated that the energy of the bits $\{x_1, x_2, x_3\}$ in state 6.as $\{E_1, E_2, E_3\} = \{-1.3, 0.2, -0.5\}$ in state 6. The probability that the bits will flip is found from (3.13). Let the threshold $\theta = -0.1$, and perturbation noise variance $\sigma^2 = 0.25$. The probability of flip for a bit with enrgy E_i is given by (3.13). Therefore,

$$p_{\text{flip}}(x_1) = Q(-0.1, -1.3, \sqrt{0.25}) \tag{3.14}$$

$$p_{\text{flip}}(x_2) = Q(-0.1, 0.2, \sqrt{0.25}) \tag{3.15}$$

$$p_{\text{flip}}(x_3) = Q(-0.1, -0.5, \sqrt{0.25}) \tag{3.16}$$

To reach from state 6 to state 7, the bit x_2 flips from 1 to -1 and the bit x_3 flips from -1 to 1. The bit x_1 does not flip. Therefore the probability of transition from state 6 to state 7 is given by

$$\Pr_{(6\to7)} = (1 - p_{\text{flip}}(x_1)) \times p_{\text{flip}}(x_2) \times p_{\text{flip}}(x_3) \tag{3.17}$$

Similarly, probability of staying in the same state is that no bit will flip:

$$\Pr_{(6\to6)} = (1 - p_{\text{flip}}(x_1)) \times (1 - p_{\text{flip}}(x_2)) \times (1 - p_{\text{flip}}(x_3)) \tag{3.18}$$

A transition matrix for the 8 states can be constructed with the probabilities for the transition from every state to every other state, whose entries are calculated for the energies of the bits in each state and their flip probability in that state. We denote the transition matrix after the first iteration as T_{r_1}.

$$T_{r_1} = \begin{pmatrix} \Pr_{(1\to1)} & \Pr_{(1\to2)} & \Pr_{(1\to3)} & \Pr_{(1\to4)} & \Pr_{(1\to5)} & \Pr_{(1\to6)} & \Pr_{(1\to7)} & \Pr_{(1\to8)} \\ \Pr_{(2\to1)} & \Pr_{(2\to2)} & \Pr_{(2\to3)} & \Pr_{(2\to4)} & \Pr_{(2\to5)} & \Pr_{(2\to6)} & \Pr_{(2\to7)} & \Pr_{(2\to8)} \\ \Pr_{(3\to1)} & \Pr_{(3\to2)} & \Pr_{(3\to3)} & \Pr_{(3\to4)} & \Pr_{(3\to5)} & \Pr_{(3\to6)} & \Pr_{(3\to7)} & \Pr_{(3\to8)} \\ \Pr_{(4\to1)} & \Pr_{(4\to2)} & \Pr_{(4\to3)} & \Pr_{(4\to4)} & \Pr_{(4\to5)} & \Pr_{(4\to6)} & \Pr_{(4\to7)} & \Pr_{(4\to8)} \\ \Pr_{(5\to1)} & \Pr_{(5\to2)} & \Pr_{(5\to3)} & \Pr_{(5\to4)} & \Pr_{(5\to5)} & \Pr_{(5\to6)} & \Pr_{(5\to7)} & \Pr_{(5\to8)} \\ \Pr_{(6\to1)} & \Pr_{(6\to2)} & \Pr_{(6\to3)} & \Pr_{(6\to4)} & \Pr_{(6\to5)} & \Pr_{(6\to6)} & \Pr_{(6\to7)} & \Pr_{(6\to8)} \\ \Pr_{(7\to1)} & \Pr_{(7\to2)} & \Pr_{(7\to3)} & \Pr_{(7\to4)} & \Pr_{(7\to5)} & \Pr_{(7\to6)} & \Pr_{(7\to7)} & \Pr_{(7\to8)} \\ \Pr_{(8\to1)} & \Pr_{(8\to2)} & \Pr_{(8\to3)} & \Pr_{(8\to4)} & \Pr_{(8\to5)} & \Pr_{(8\to6)} & \Pr_{(8\to7)} & \Pr_{(8\to8)} \end{pmatrix} \tag{3.19}$$

The transition matrix describes that once a sample $\{x_1, x_2, x_3\}$ is received at the trapping set T_{r_1}, the initial state of T_{r_1} is determined by $\{x_1, x_2, x_3\}$. The transition matrix T_{r_1} then decides the next transitions at each iteration. The transition matrix $T_{r\ell}$ at the ℓ^{th} iteration is given by

$$T_{r_\ell} = (T_{r_1})^{\ell} \tag{3.20}$$

For a ergodic Markov chain, if the transition matrix (T_r) is irreducible, then there exists exactly one eigenvector $\vec{w}$ such that

$$\vec{w} \times T_r = \vec{w} \tag{3.21}$$

Also, $\vec{w}$ can be chosen such that all its entries are strictly positive. This is the Perron-Frobenius theorem for the context of a Markov chain. The vector $\vec{w}$ denotes the steady state probability of each state. It is the common row of the transition matrix after n steps when the Markov chain reaches steady-state. For the all zero codeword, the correct state

after decoding is the state 1 corresponding to $\{x_1, x_2, x_3\} = \{1, 1, 1\}$. The first component of $\vec{w}$ denotes the probability that the decoder eventually reaches the correct state.

For the all-zero codeword, we are interested in the steady-state probability of the state $\{x_1, x_2, x_3\} = \{1, 1, 1\}$, which the 1^{st} state in the transition matrix. The probability of error for sample is therefore,

$$P_{\text{err}} = 1 - \vec{w}(1) \tag{3.22}$$

If the decoding algorithm is not allowed n iterations to reach the steady state, let the maximum allowed iterations be ℓ such that $ell < n$. The transition matrix at the end of ℓ iterations is T_{r_ℓ}. Then the probability that the decoding is successful after the ℓ iterations starting at state s is given by $T_{r_\ell}(s, 1) = P_{s\to 1}$ calculated at the end of the ℓ iterations, assuming all-zero codeword was transmitted. In that case the probability of error for the sample is

$$P_{\text{err}} = 1 - \Pr_{(s\to 1)} \tag{3.23}$$

Given that the probability of the received sample y_T is P_{sample}, the frame error rate (FER) for the sample is given by

$$\text{FER} = P_{\text{sample}} \times P_{\text{err}} \tag{3.24}$$

In Ch. 4 we use importance sampling to sample received messages close to the error region to estimate the FER for a given (8,8) absorbing set. The maximum frame error rate (FER) that is found from sampling from 1 million samples are reported, as well as the average FER that is obtained.

CHAPTER 4
SIMULATION METHODOLOGY

4.1 Importance sampling for fast simulation of the error-floor

To measure the error-floor, usually a large number of samples are generated for the noisy channel, and then decoded by the decoder. To accurately measure the BER graph and error-floor, sufficient error events must be observed. Because the error-floor is very low for LDPC codes, it may take a very long simulation time to observe an error event at high SNRs. To simulate a fair number of error events at a high SNR, or and to study the trapping sets, importance sampling is widely used [27–30]. The generation of samples are biased in a way that many errors are generated. Later, to measure the true BER, the probability of the samples are weighted to account for the biasing.

The idea of importance sampling is to sample from an important region of the target distribution p. Suppose the problem is the find the expectation of (X) where $X \sim p$.

$$E_p[(X)] = \int xp(x)dx \tag{4.1}$$

$$= \int \frac{xp(x)}{q(x)} q(x)dx \tag{4.2}$$

$$= E_q\Big[\frac{xp(x)}{q(x)}\Big] \tag{4.3}$$

That means sampling x from distribution p is the same as sampling $\frac{xp(x)}{q(x)}$ from sample distribution q. The term $\frac{p(x)}{q(x)}$ is called the importance weight or the likelihood ratio. The distribution p is called the nominal or target distribution and q is the importance distribution or the sample distribution.

For AWGN channel, the noise variance σ_z^2 of the channel is given by

$$\sigma_z^2 = \frac{N_0}{2}, N_0 = \frac{1}{R} \times 10^{(\frac{-SNR}{10})} \tag{4.4}$$

where, N_0 is the noise spectral density of the channel, R is the rate of the code, and SNR is the signal-to-noise ratio of the channel. Therefore, for the all-zero codeword ($\hat{c} = \{1\}^N$), the incoming messages are distributed as a Gaussian distribution with mean 1 and variance σ_z^2, $\mathcal{N}(1, \sigma_z^2)$.

For the all-zero codeword, and AWGN channel, we take the target distribution as $p = \mathcal{N}(1, \sigma_z^2)$, where σ_z^2 is the variance of the channel noise. To sample in the error region, we use mean translation importance sampling, where the mean of the importance distribution is shifted toward the region of interest. We use $q = \mathcal{N}(0.5, \sigma_z^2)$ as the importance distribution to sample more error cases for fast error-floor estimation.

In importance sampling, the average value is taken as the estimated result. However, in a real decoder, if a particularly noisy sample is encountered, that sample would determine the ultimate error-floor of the LDPC code. Therefore, while using importance sampling to sample in the error region, we would consider the highest FER to set the estimated error-floor.

4.2 The dominant (8,8) absorbing set under the NGDBF algorithm

We consider the subgraph induced by the (8,8) absorbing set in the 802.3an LDPC code for our estimation of the error-floor. The $H_{T(8,8)}$ matrix for the absorbing set is given by

$H_{T(8,8)} =$

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}^T \tag{4.5}$$

The subgraph induced by the absorbing set $H_{T(8,8)}$ is shown in Fig. 4.1. The variable nodes are represented as ●, degree-one check nodes as ■, and degree-two check nodes as □.

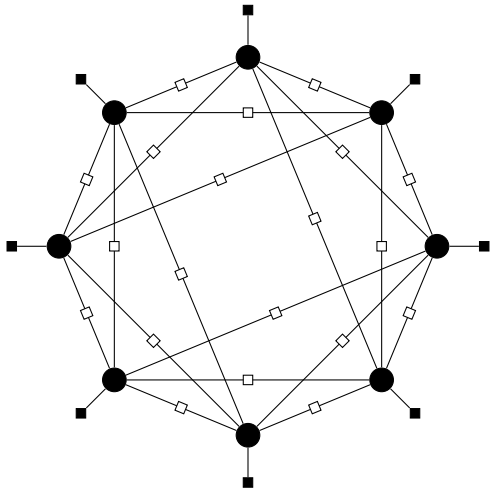


Fig. 4.1: The dominant $(8, 8)$ absorbing set in the 802.3an 10GBASE-T LDPC code. There are 8 degree one check nodes, 20 degree 2 check nodes, and 8 variable nodes.

4.3 Simulation results

For the simulation, a sample y_T of length 8 to the absorbing set is sampled from

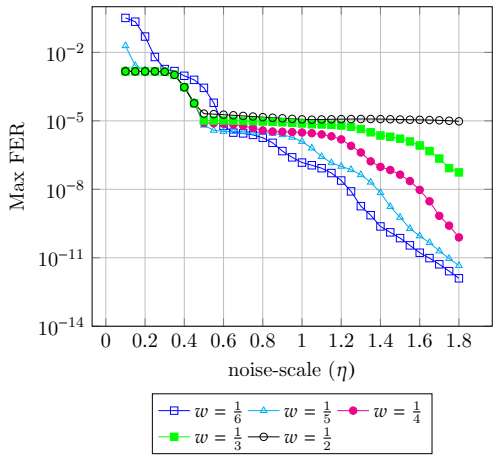
the importance distribution $\mathcal{N}(0.5, \sigma_z^2)$, and decoded with the NGDBF algorithm for 600 iterations. The energy of each bit is calculated as (3.5).

$$E_i = x_i y_{Ti} + \frac{1}{w} \sum_j (s_j) + q_i;$$

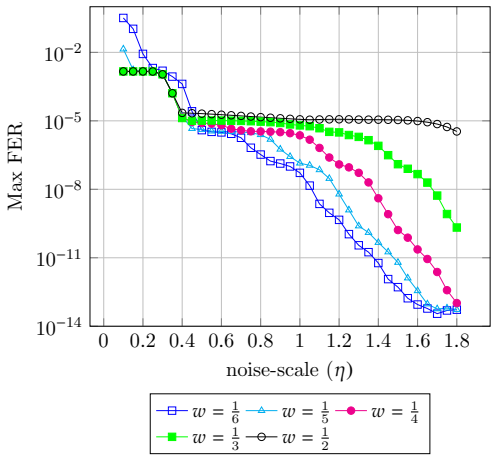
$$\text{where, } q_i \sim \mathcal{N}(0, \sigma^2), \sigma^2 = \eta \times \sigma_z^2$$

where w is called a weight parameter, and the perturbation noise variance σ^2 is calculated by scaling the channel noise variance σ_z^2 by a factor η.

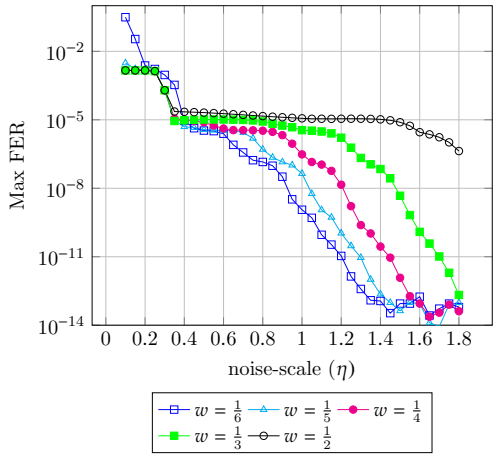
For a given noise-scale (η), increasing the weight w reduces the error-floor a seen in Fig. Fig. 4.2 and 4.3. Error-floor as a function of noise-scale (η) for different weights (w) is shown in Fig. 4.2 and 4.3. For each data point in Fig. 4.2 and 4.3 1000000 samples using importance sampling is used. The FER that are obtained for different weights (w) and noise-scale (η) are scaled by 10^4 to account for the multiplicity of the (8,8) absorbing set in the IEEE802.3an LDPC code.



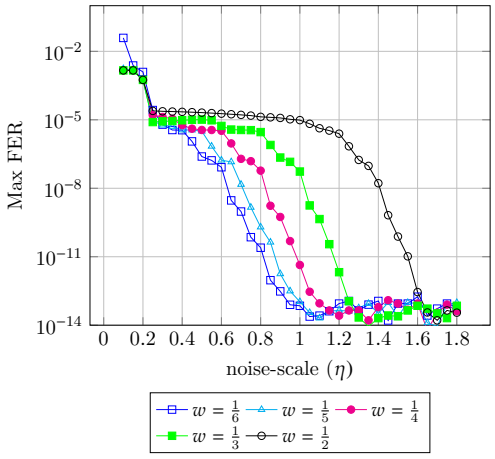
(a) 600 iterations



(b) 5000 iterations

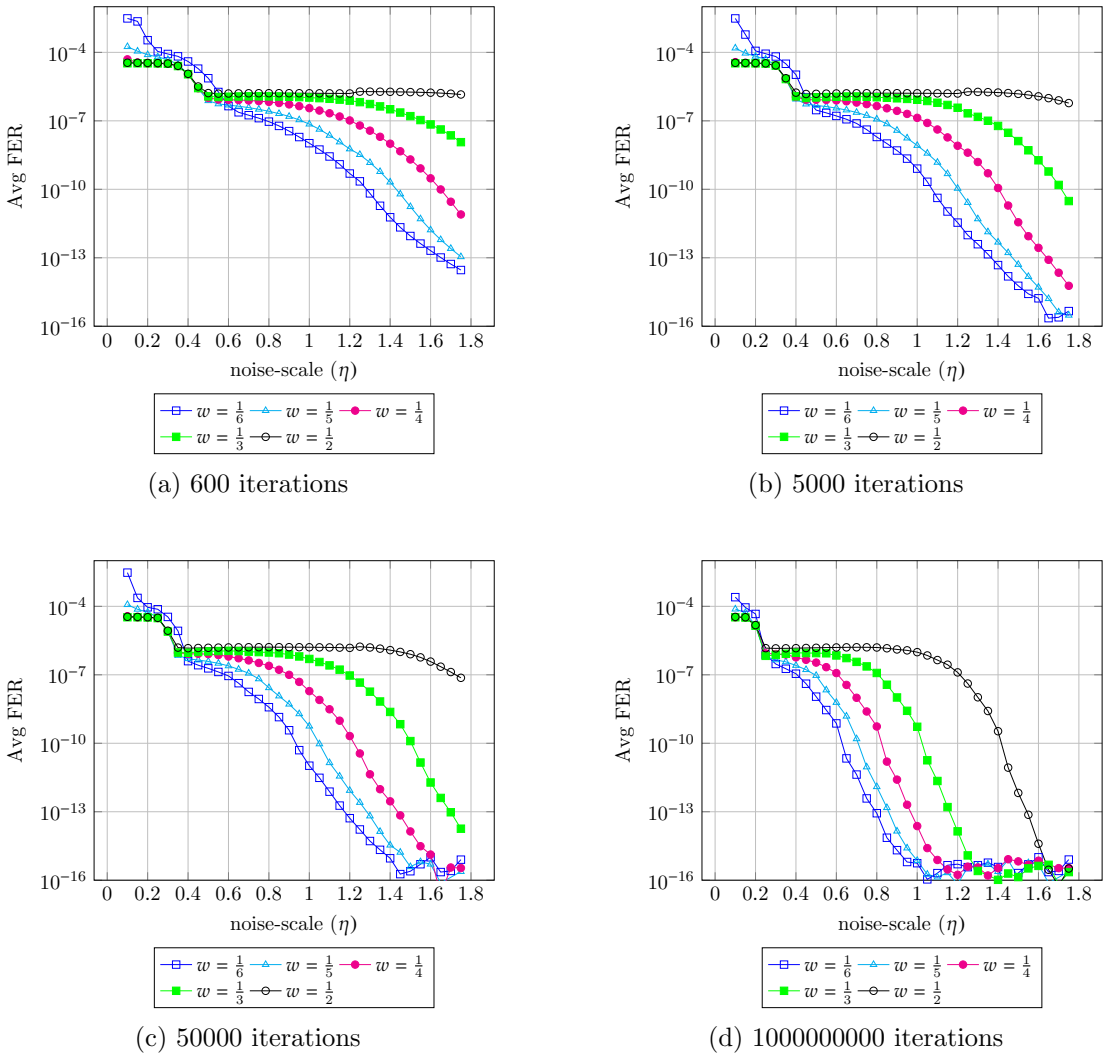


(c) 50000 iterations



(d) 1000000000 iterations

Fig. 4.2: Maximum FER found from 1000000 samples, FER is scaled by 10^4 to account for the multiplicity of the (8,8) absorbing set.



(a) 600 iterations

(b) 5000 iterations

(c) 50000 iterations

(d) 1000000000 iterations

Fig. 4.3: Averge FER found from 1000000 samples, FER is scaled by 10^4 to account for the multiplicity of the (8,8) absorbing set.

4.4 Newly created errors and decoder limits:

The NGDBF algorithm takes advantage of the random noise used in every iteration to escape from local minima. As seen in Fig. 4.3 and Fig. 4.2, the FER continues to improve with increasing iterations. We want to investigate where the error-floor might lie, if there exists one. In practice, while the NGDBF algorithm uses random noise to escapes from the dominant (8,8) absorbing set, it creates new trapping sets with the random noise that is used for decoding. Intuitively, we may suggest that with increasing number of iterations, the

newly created trapping sets keep increasing. Also, with decreasing weight (w), or increasing noise-scale (η), or both, the random noise becomes more dominant in the energy function (3.5), causing more newly created trapping sets to emerge and degrade the performance.

To observe this effect of the random noise, we yet again performed importance sampling to sample in the error free region of the channel-messages. The Markov chain corresponding to the NGDBF algorithm is Ergodic type, as each state is reachable from every other state. Also, all the entries in the transition matrix T_r are strictly positive. For an Ergodic and aperiodic Markov chain, with a positive transition matrix T_r, the steady state transition probabilities can be found using Perron-Frobenius theorem. The Perron-Frobenius theorem states that if all entries of a $n \times n$ matrix T_r are positive, then it has a unique maximal eigenvalue. Its eigenvector has positive entries. It can be showed that 1 is the dominant eigenvalue with a positive eigenvector π. The unique positive eigenvector π is called the Perron-Frobenius vector of T_r. The Perron-Frobenius vector π is a very important characteristic vector of the Markov chain, and is often called the the invariant measure of T_r. The Perron-Frobenius vector π is the unique steady state vector for an ergodic and aperiodic Markov chain. For the irreducible and aperiodic transition matrix P and its Perron-Frobenius vector π, we have for any probability vector $\nu \in R^N$,

$$\lim_{n\to\infty} T_r^n \nu = \pi \tag{4.6}$$

The Perron-Frobenius vector is used to calculate the probability of error at steady state for the error free samples. First, the dominant eigenvalue is found by taking the singular value decomposition (SVD) of the transition matrix for that particular sample. Then the eigenvector corresponding to the dominant eigenvalue was normalized to find steady state probabilities for all the states. Let the all-zero state be represented by state s_0, which is the first state in the transition matrix T_r. The steady state probability of the all-zero state is given by the first entry of the Perron-Frobenius vector π. Therefore probability of error at the end of decoding for the particular channel-message is

$$P_{err} = 1 - \pi(1) \tag{4.7}$$

The simulations reveal that, with decreasing weight (w), or increasing noise-scale (η), or both, creates new errors in otherwise good channel-messages. The results are shown in Fig. 4.4. The reason is, as the variance of the random perturbation is increased (or the weight of the syndrome sum is decreased) to the point that the stochastic behavior completely overshadows the deterministic part (the summation of syndromes) of the energy function. The error-floor is estimated to be in the region where the decreasing error graph for existing trapping sets Fig. 4.2, and Fig. 4.3 and the graph for the newly created errors Fig. 4.4 cross over each other. This is showed in Fig. 4.5.

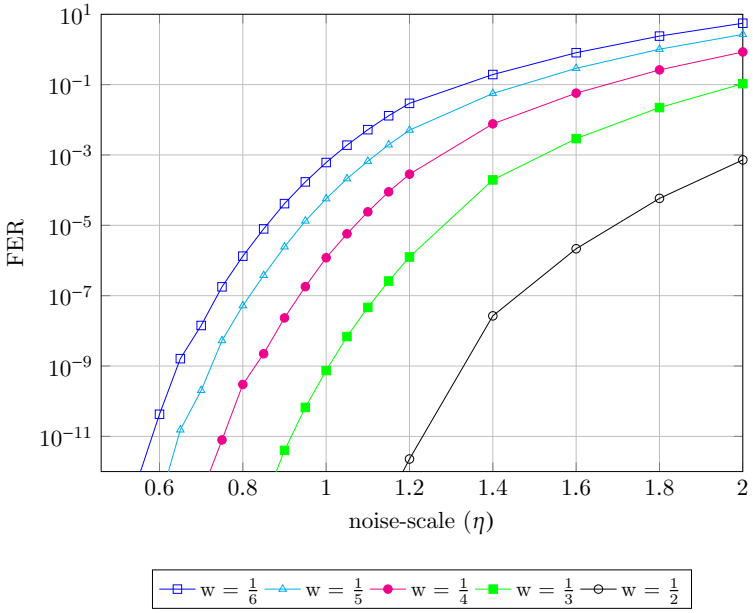
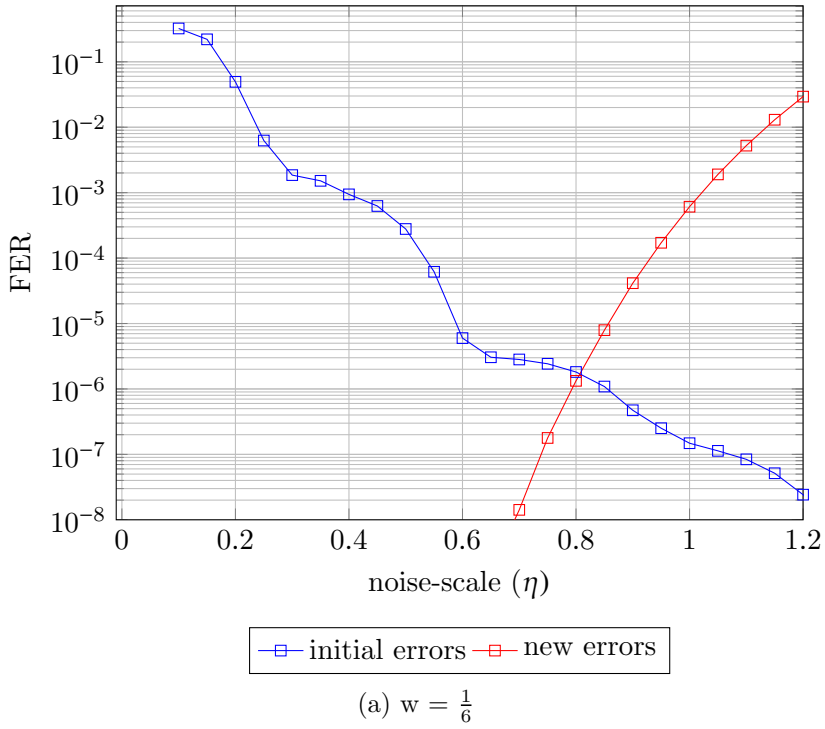
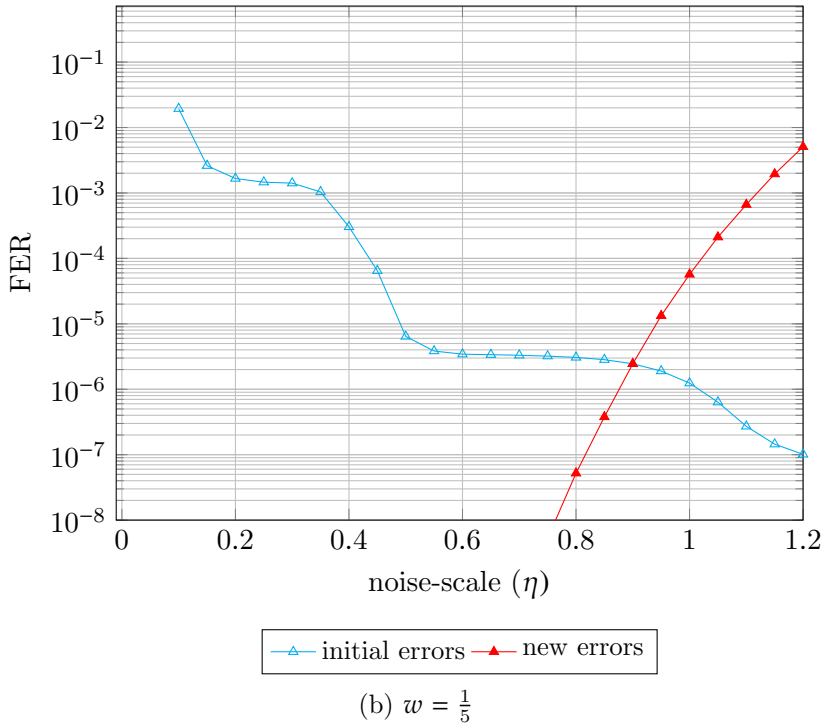


Fig. 4.4: average newly created errors of 1000000 samples of the (8,8) absorbing set. The FER is scaled by 10^4 to account for the multiplicity of the (8,8) absorbing set.



(a) w = $\frac{1}{6}$



(b) $w = \frac{1}{5}$

Fig. 4.5: Estimated error-floor where the resolved trapping sets (decreasing errors) and newly created trapping sets (increasing errors) cross over each other.The FER are scaled by 10^4 to account for the multiplicity of the (8,8) absorbing set.

CHAPTER 5

FPGA IMPLEMENTATION

In order to validate the design, an NGDBF decoder with 5-bit resolution has been fully synthesized, implemented and validated for a Xilinx VCU 118 board using Vivado 2017.1. A screen-shot of the architecture is shown in Fig, 5.1. This chapter describes the NGDBF demonstration design implemented in behavioral SystemVerilog. A basic UART test interface is provided using a Microblaze processor to control the decoder. The test demonstration allows testing a large number of frames of channel data which are input serially via the UART interface.

5.1 RTL description:

The FPGA implementation of the entire system is shown in the block diagram of Fig. 5.1. There are 6 decoding modules. Each of these modules contain a channel emulator and an NGDBF decoder. The modules have the same internal architecture. However, the stochastic behavior of the noise perturbation samples and channel samples allow them to have different decoding behavior from each other. The decoder's architecture and interface are depicted in Fig. 5.2. Details of the algorithm and its calculations are given in Sec. 5.2. Primary I/Os are indicated in red, and handshaking I/Os are indicated in blue. All channel and noise samples have resolution Q = 5 bits in sign-magnitude format.

5.1.1 Operating sequence of the decoder:

Operation sequence of the decoder is as follows:

1. When the powerup signal is high, the registers are filled sequentially with samples from noise_in. In principle, sample values could be coded as initial states for each register in order to avoid the latency incurred by this powerup sequence. When powerup is low, the samples simply recirculate Simulations have shown that recirculating samples are sufficient

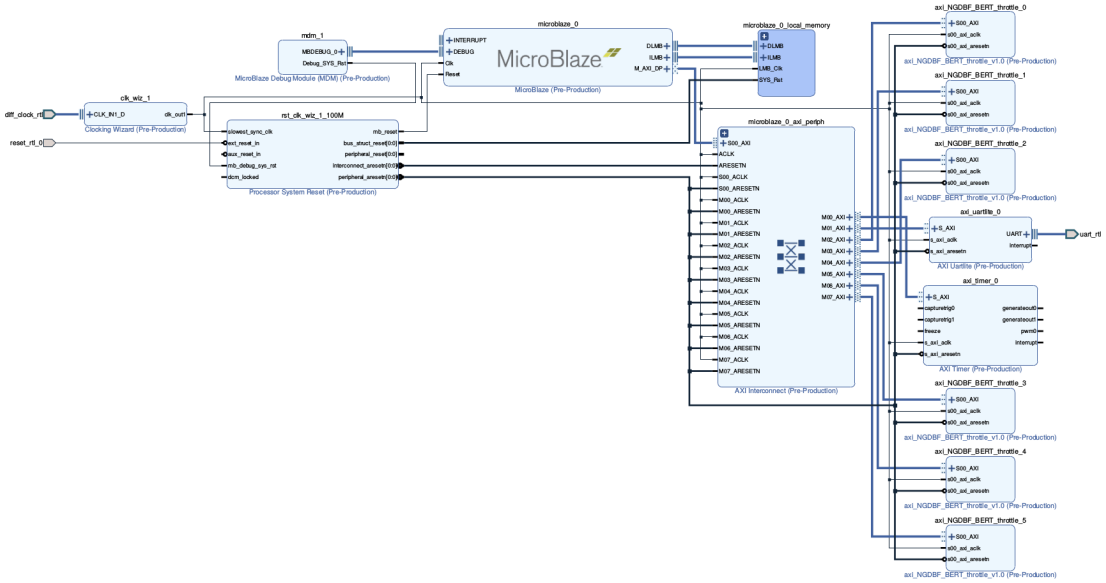


Fig. 5.1: test-bench for NGDBF decoder with microblaze controller

to achieve high performance in the NGDBF algorithm. When all the registers have been initialized, the scanchain_full signal is taken high to indicate that the decoder is ready.

2. Then the initialize signal is high, the decoder's decisions are initialized as the sign bits of channel_messages. Decoding begins when initialize is low. The decoder will continue indefinitely until it is reinitialized. When the decoder detects that the decisions are a valid codeword, the done signal is asserted high to indicate that the stopping condition has been reached. The decoder itself does not track its iterations or declare failure.

3. The parity check nodes and early stopping decision are implemented via assign statements in the decoder's top module. The symbol nodes have a more complex implementation, and use 8-bit arithmetic internally.

All pre-scaling adjustments to channel and noise samples (as described in Sec. 5.2) are performed in the C simulation prior to generating the data files. The test-bench demonstrates use of the test controller, where channel and noise samples are input serially prior to decoding. After decoding, the number of errors for each iteration is reported and the simulation terminates.

5.1.2 Brief descriptions of the modules

Below are descripotions of the important modules of the decoder:

1. The channel emulator: The channel emulator provides noisy samples assuming all zero codeword, modulated as +1s. This does not allow for a system with real codewords. To provide real codewords to the system, the channel samples are compared with real codewords. The sign of the sample is changed to be negative (1 is modulated as -1) if a "1" is encountered in the real codeword. The conversion is done in the symbol node description. The SNR of the channel can be provided from the external interface of the UART manually, by choosing a value called the channel index. For 5 db channel, the channel index is 20.

2. The decoder: The decoder has 3 main building blocks: the symbol node description, the check node description, and the noise perturbation scanchain as shown in Fig. 5.2.

2a. Symbol node: At the symbol node, all-zero samples added with AWGN channel noise come from the channel emulator. The channel messages are then converted to real codewords (not all-zero) by changing the signs of the samples according to a set of predefined codewords, before sending off to neighboring check nodes. As the syndrome calculations are received by the symbol node from the neighboring check nodes, the energy is calculated to decide whether the bit will be flipped or not.

2b. Check node: The check node description contains the connections of each of the check nodes to their neighboring symbol nodes. The check node has an early termination unit to signal "done" if the decoding is successfully completed before the maximum number of iterations.

2c. Scanchain/ Decoder noise perturbations: The algorithm relies on noise perturbations which must be unique to each symbol node at each iteration. Rather than include a random number generator in the design, it is sufficient to use a circular shift register. The "scanchain" module is essentially the same as the channel emulator. However, because the channel emulator provides all-zero transmission (modulated as +1s), the noise perturbations are obtained by subtracting +1 to provide zero-mean noise. It is to be noted that only 600+2048 samples are initialized before the start of the decoding (once for each set of

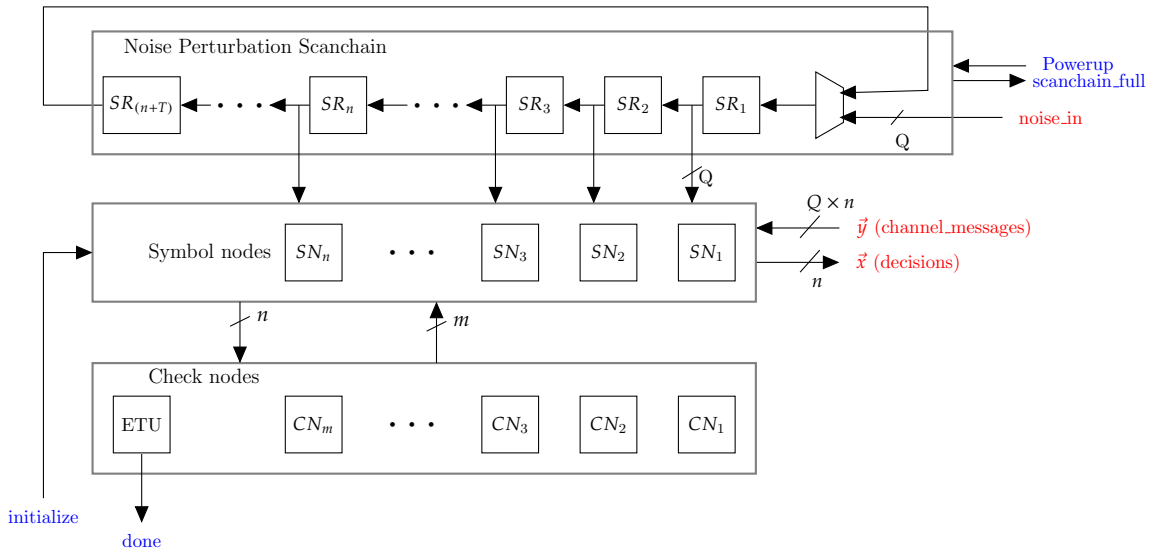


Fig. 5.2: The NGDBF decoder with 2048 symbol nodes, 2648 noise perturbation registers(2048 symbol nodes + 600 iterations) in circular scanchain, 384 parity check nodes and an early termination unit (ETU)

tests), assuming a maximum of 600 iterations, and for the 2048 bits of a sample. The noise perturbations are statistically independent, allowing the same noise samples to be reused.

The SNR of the noise perturbations can also be provided via the UART manually. For noisescale $\eta = 1$, for our 5 db channel (channel index 20), the noise index is also 20.

5.2 Theory of the design

An n-bit codeword is transmitted across an antipodal binary-input Additive White Gaussian Noise (AWGN) channel. The decoder receives quantized, clipped samples from the channel. For each sample, an initial hypothesis bit is taken based on the samples sign, where a positive sign indicates a 0 and a negative sign indicates a 1. The decoder then computes the parity syndrome from the hypothesis bits. Then, for each bit, an energy function is computed. If the energy falls below a specified threshold, the bit is flipped. Then the syndrome is computed again and the procedure is iterated until all parity checks are satisfied, or a maximum number of iterations is reached. For a precise statement of the algorithm, the following symbols are defined

H	$n \times m$ Binary parity check matrix
$\vec{c} \in \mathbb{Z}_2^n$	Binary codeword
$\hat{c} \in \{+1, 1\}^n$	Bipolar codeword $\hat{c} = 1 - 2\vec{c}$
$\vec{z} \in \mathbb{R}^n$	I.I.D. white noise samples
$y_{\max}$	Sample clipping limit
$\tilde{y}$	Quantized channel samples on Q bits within $[-y_{\max}, +y_{\max}]$
$\hat{x} \in \{+1, -1\}^n$	Bipolar hypothesis $\vec{x} = \text{sign}(\vec{y})$
$\hat{x} \in \mathbb{Z}_2^n$	Binary hypothesis $\vec{x} = 0.5(1 - \hat{x})$
$\vec{s} \in \mathbb{Z}_2^m$	Syndrome $\vec{s} = H^T \vec{x}$
$\hat{s} \in \{+1, -1\}^m$	Bipolar syndrome
$\eta \in \mathbb{R}$	Noise-scale
$w \in \mathbb{R}$	Weight (syndrome weight)
$\theta \in \mathbb{R}$	Flipping threshold, $\theta < 0$
λ	Threshold (θ) adaptation parameter, $0 < \lambda \leq 1$
$E_i \in \mathbb{R}$	Energy function for bit i
N_i	Adjacency in H for bit i
M_j	Adjacency in H for parity check j
$L \in \mathbb{Z}$	Maximum number of iterations
$\ell \in \mathbb{Z}$	Iteration number between 0 and L
$\vec{z} \in \mathbb{R}^n$	Decision

Table 5.1: Symbols for the algorithm

The NGDBF algorithm is traditionally developed using bipolar notation (e.g. $\hat{x}$, $\hat{s}$), which is convenient for algebraic analysis. The bipolar notation is presented here and translated into binary form appropriate for hardware description. The algorithms precise steps are as follows:

1. Initialize $\hat{x}^{(\ell)} = \text{sign}\ (\vec{y})$ and reset iteration number $l = 0$.

2. Compute the syndrome $\vec{s}^{l} = H^{T}\vec{x}^{(\ell)}$.

3. Early stopping condition: If $\vec{s} = \vec{0}$ then output $\vec{d} = \vec{x}^{l}$ and halt.

4. For bits $i = 1, 2, ..., n$:

   (a) Compute $E_i^{(\ell)} = \hat{x}_i \tilde{y}_i + w \sum_{j \in \mathcal{N}} \hat{s}_j + z_i^{(\ell)}$

   (b) If $E_i^{(\ell)} < \theta$ then set $\hat{x}_i^{(\ell+1)} = \hat{x}_i^{(\ell)}$

5. If $\ell < L$ then increment ℓ and return to step 2. Else output $\vec{x}$, declare failure and halt.

5.3 Syndrome and early stopping condition

$$\text{bipolar: } \hat{s}_j = \prod\nolimits_{i \in \mathcal{M}_j} \hat{x}_j \tag{5.1}$$

$$\text{binary: } s_j = \oplus_{i \in \mathcal{M}_j} x_i \tag{5.2}$$

where '$\oplus$' indicates addition in $GF2$. In hardware terms, this can be interpreted as a tree of binary XOR operations.

The early stopping condition is most easily expressed in binary notation:

$$\text{stop if } \cup_{j \in \{1,2,...,m\}} S_j \tag{5.3}$$

5.4 Energy function and threshold

The bulk of the algorithm's arithmetic lies in the energy function. For implementation, both the energy function and threshold operation are combined in one expression:

$$\text{flip if, } E_i - \theta < 0 \tag{5.4}$$

$$\Rightarrow \hat{x}_i \tilde{y}_i + w \sum\nolimits_{j \in \mathcal{N}_i} \hat{s}_j + z_i - \theta < 0 \tag{5.5}$$

To translate this into binary format, we may assume variables are represented in a sign-magnitude format { sgn, mag }. Then the flip decision may be expressed as

$$\{x_i \oplus \text{sign}(\tilde{y}_i), \text{mag}(\tilde{y}_i)\} + w \sum_{j \in \mathcal{N}_i} (1 - 2s_j) + z_i - \theta < 0 \tag{5.6}$$

Since the IEEE 802.3an LDPC code is regular, $|\mathcal{N}_i| = 6$ for every i, yielding some simplification:

$$\{x_i \oplus \text{sign}(\tilde{y}_i), \text{mag}(\tilde{y}_i)\} + w\left(6 - 2\sum_{j \in \mathcal{N}_i} s_j\right) + z_i - \theta < 0 \tag{5.7}$$

$$\left\{x_i \oplus \text{sign}(\tilde{y}_i), \frac{\text{mag}(\tilde{y}_i)}{2w}\right\} + 3 - \sum_{j \in \mathcal{N}_i} s_j + \left(\frac{z_i - \theta}{2w}\right) < 0 \tag{5.8}$$

$$\left\{x_i \oplus \text{sign}(\tilde{y}_i), \frac{\text{mag}(\tilde{y}_i)}{2w}\right\} - \sum_{j \in \mathcal{N}_i} s_j + \left(\frac{z_i - \theta}{2w} + 3\right) < 0 \tag{5.9}$$

This representation allows some of the operations to be moved into the noise samples and pre-scaling of the channel samples. The main advantage of this is to separate algorithm parameters from architecture parameters. The architecture will be designed around a default set of choices allowing for fixed thresholds and constant scaling factors. Further design exploration and "fine tuning" can be done by adjusting the channel and noise sample statistics.

In order to avoid subtraction, we note that

$$\sum_{j \in \mathcal{N}_i} \bar{s}_j = 6 - \sum_{j \in \mathcal{N}_i} s_j \tag{5.10}$$

so we can make the further simplification

$$\left\{x_i \oplus \text{sign}(\tilde{y}_i), \frac{\text{mag}(\tilde{y}_i)}{2w}\right\} + \sum_{j \in \mathcal{N}_i} \bar{s}_j + \left(\frac{z_i - \theta}{2w} + 3 - 6\right) < 0 \tag{5.11}$$

https://www.overleaf.com/project/5a510ffa2abc9179e05c1337 When the noise samples are pre-shifted by the threshold , the resulting non-zero mean reduces the dynamic range of noise perturbations and may introduce bias. Through empirical optimization, good NGDBF parameters for the IEEE 802.3an were found to be $\theta = 0.55$ and $w = \frac{1}{6}$ in [12], so that the mean of z' is $\theta/2w - 3 = 1.65 - 3 = -1.35$. The dynamic range is improved by introducing a constant shift of + 2 into the samples:

$$\text{flip if } \left\{x_i \oplus \text{sign}(\tilde{y}_i), \frac{\text{mag}(\tilde{y}_i)}{2w}\right\} + \sum_{j \in \mathcal{N}_i} \bar{s}_j + \left(\frac{z_i - \theta}{2w} - 3 + 2\right) < 2 \tag{5.12}$$

Finally, we define the adjusted channel and noise samples as

$$y'_i = \frac{\tilde{y}_i}{2w} \tag{5.13}$$

$$z'_i = \frac{z_i - \theta}{2w} - 1 \tag{5.14}$$

Then the flip decision is simplified as

$$\text{flip if } \{x_i \oplus \text{sign}(\tilde{y}_i), \text{mag}(\tilde{y}_i)\} + \sum_{j \in \mathcal{N}_i} \bar{s}_j + z'_i < 2 \tag{5.15}$$

5.5 Quantization and Least Significant Bit (LSB) corrections

In order to physically realize the energy calculation, two further problems need to be addressed: the first is the occurrence of ±0 in sign-magnitude representations. The second problem is that y', q' and θ' are quantized real numbers requiring fractional bits, so the syndrome summation must be adjusted to account for the quantization.

The first problem is solved by appending one LSB to each channel and noise sample, and to the modified threshold θ' . This means that signals are widened by one bit when performing symbol node arithmetic. We refer to this as the "LSB correction".

For the second problem, the quantization is corrected by identifying the integer value corresponding to a pre-quantized "1.0". The parameter $y_{\max}$ dictates the maximum channel sample magnitude. For modified samples, the maximum is translated to $y'_{\max} = y_{\max}/(2w)$. Since samples are widened by one bit for the LSB correction, there are 2^Q samples between 0 and $y'_{\max}$. We may therefore correct the syndrome summation with a scale multiplier defined as

$$S_{\text{mult}} = \lceil \frac{1.0}{y'_{\max}} (2^Q) \rceil \tag{5.16}$$

$$\theta_{hw} = 2S_{\text{mult}} + 1 \tag{5.17}$$

Then the flip decision is revised to its final form:

$$\text{flip if } \{x_i \oplus \text{sign}(\tilde{y}_i), \text{mag}(\tilde{y}_i), 1\} + S_{\text{mult}} \sum_{j \in \mathcal{N}_i} \bar{s}_j + \left\{z'_i, 1\right\} < 2S_{\text{mult}} + 1 \tag{5.18}$$

$$\Rightarrow \{x_i \oplus \text{sign}(\tilde{y}_i), \text{mag}(\tilde{y}_i), 1\} + S_{\text{mult}} \sum_{j \in \mathcal{N}_i} \bar{s}_j + \left\{z'_i, 1\right\} < \theta_{\text{hw}} \tag{5.19}$$

The typical values used in our decoder are: $y_{\max} = 1.625$, $\theta = -0.525$, $Q = 5$. The values of S_{mult} and θ_{hw} change with changing w as follows:

w	S_{mult}	θ_{hw}
$\frac{1}{6}$	7	15
$\frac{1}{5}$	8	16
$\frac{1}{4}$	10	21

Table 5.2: Decoder parameters for varying weights

5.6 FPGA results

The decoder was synthesized for three weights, $w = \{\frac{1}{6}, \frac{1}{5}, \frac{1}{4}\}$. The results are plotted in Fig. 5.3, Fig. 5.4 and Fig. 5.5 for the weights in respective order. In the figures, the simulation results for decreasing error-floor for initial trapping sets and the increasing error-floor for new trapping sets are shown for comparison. The actual error-floor is expected to be affected by both the increasing new trapping sets and decreasing initial trapping sets. The optimal choice for weight (w) and noise-scale (η) are where the two graphs cross over each other, where the error-floor is the lowest.

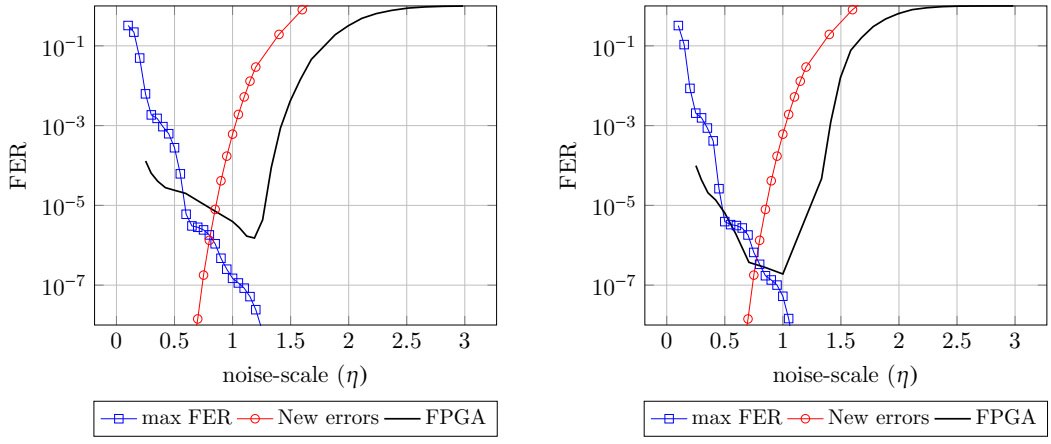
As seen in the figures, the results are very close the prediction for all the presented results. To determine the FER contributions from the new trapping sets, the steady state error probabilities are considered. Which theoretically means that we are considering an infinite number of iterations for new trapping sets to be emerged. Therefore, the FER contributions from the new trapping sets is a pessimistic over-estimation. In the hardware implementation, the decoder is able to achieve an optimum error-floor slightly lower than the prediction, as evident in the figures.

w	Estimated FER	FPGA FER
$\frac{1}{6}$	1.0×10^{-06} $(\eta = 0.80)$	1.5×10^{-06} $(\eta = 1.19)$
$\frac{1}{5}$	2.5×10^{-06} $(\eta = 0.90)$	1.96×10^{-07} $(\eta = 1.26)$
$\frac{1}{4}$	3.0×10^{-06} $(\eta = 1.00)$	4.25×10^{-07} $(\eta = 1.40)$

Table 5.3: Estimated FER vs Obtained FER from FPGA, for 600 iterations

w	Estimated FER	FPGA FER
$\frac{1}{6}$	4×10^{-07} $(\eta = 0.75)$	2.0×10^{-07} $(\eta = 1.0)$
$\frac{1}{5}$	1×10^{-06} $(\eta = 0.85)$	3.5×10^{-08} $(\eta = 1.33)$
$\frac{1}{4}$	2×10^{-06} $(\eta = 1.00)$	6.6×10^{-08} $(\eta = 1.40)$

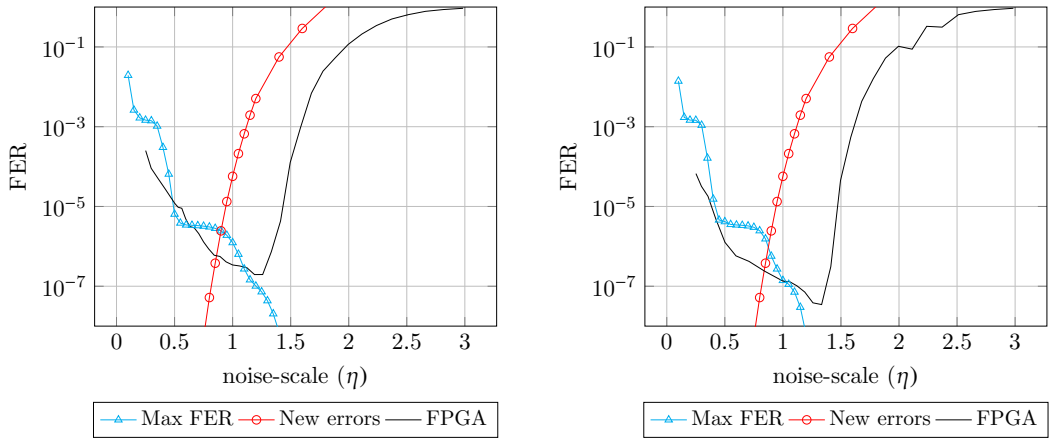
Table 5.4: Estimated FER vs Obtained FER from FPGA, for 5000 iterations



(a) 600 iteration

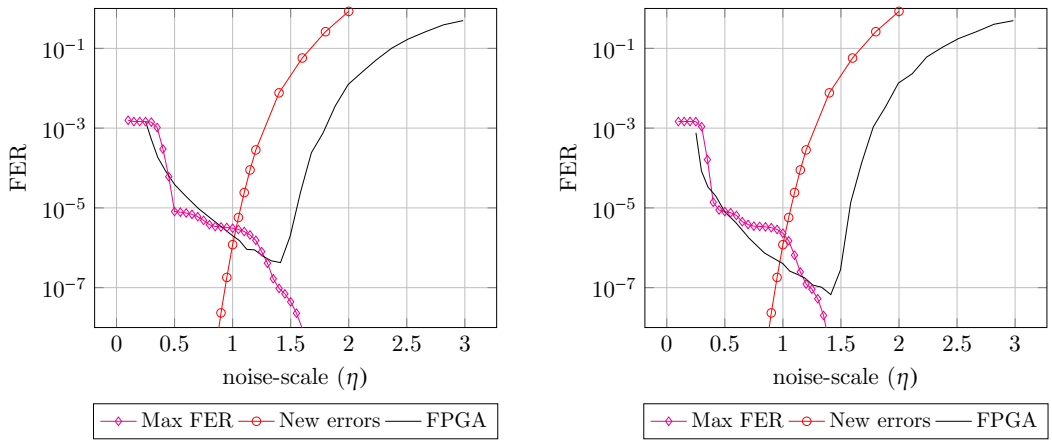
(b) 5000 iterations

Fig. 5.3: Comparing estimated FER with FPGA implementation results, wt = $\frac{1}{6}$



(a) 600 iterations (b) 5000 iterations

Fig. 5.4: Comparing estimated FER with FPGA implementation results, wt = $\frac{1}{5}$



(a) 600 iterations (b) 5000 iterations

Fig. 5.5: Comparing estimated FER with FPGA implementation results, wt = $\frac{1}{4}$

CHAPTER 6

CONCLUSION AND FUTURE WORK

This work studies various aspects of the NGDBF algorithm for decoding LDPC codes. For high performance codes, the error-floors are very low to be simulated. To encounter multiple failed frames after decoding at high SNRs to reliably measure the error-floor takes very long simulation time. Evaluation of very low error-floors is performed by the implemented decoder. The algorithm is assisted by random perturbation in the decoding function, which gives the algorithm stochastic behavior. The perturbation noise in the NGDBF algorithm enables the decoder to escape from trapping sets. Taking further advantage of the stochastic nature of the algorithm, an improvement called the Re-decoded NGDBF (R-NGDBF) was proposed in Ch. 2. In R-NGDBF, a number of decoding phases were introduced with a limited number of iterations. If the message was not successfully decoded by a phase, then it was passed on to the next phase. The initial conditions for decoding are random in a new phase because of the random perturbations. This gives the decoder a new decoding path to follow that may result in successful decoding at a new phase. The primary goal of this dissertation was to find a faster and deterministic technique to estimate the decoding parameters for the NGDBF decoding algorithm for LDPC code. The IEEE 802.3an LDPC code has a number of trapping sets, the (8,8) trapping set being the dominant one in the error-floor region. A method using a Markov chain representation of the states for the (8,8) trapping set was proved to be successful in estimating the parameters for the IEEE 802.3an LDPC code. As beneficial as the random perturbations are for decoding, it is also responsible for creating new trapping sets that ultimately limit the performance of the decoder. The error-floor is estimated to be where the two graphs, one for the escaped trapping sets and another for the newly created trapping sets cross over each other. The error-floor is also dependent on the weight of the sum of the parity check messages. As seen in Ch .4, at high SNRs, increasing the weight lowers the error-floor. However, at low SNRs a large

weight will cause the parity check message to dominate the inversion function (1.4), and minimize the benefits of the added perturbation noise.

Implementation of the NGDBF decoder to validate the theoretical findings is another key part of this dissertation. To evaluate the performance of the IEEE 802.3 LDPC code at high SNRs, parallel FPGA implementation of multiple NGDBF decoders is done with Xilinx VCU118 platform.

Re-decoding a frame for a number of phases has been proposed by the author in [13]. Because of the random perturbation noise, every phase has an independent and random probability of successfully decoding a received frame. Implementation of the re-decoding method would imply a number of parallel decoders operating simultaneously on a single frame until one of the decoders finishes decoding. However, the benefits of increasing the number of phases tends to saturate after a number of phases. Optimization of the number of employed decoders on an FPGA for maximum speed in terms of clock frequency, critical path and iteration numbers would be done in future research.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[2] W. W. Peterson and E. J. Weldon, *Error-correcting codes*. MIT press, 1972.

[3] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker, "Applications of error-control coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2531–2560, Oct 1998.

[4] P. M. Shah, P. D. Vyavahare, and A. Jain, "Modern error correcting codes for 4g and beyond: Turbo codes and ldpc codes," in *2015 Radio and Antenna Days of the Indian Ocean (RADIO)*, Sept 2015, pp. 1–2.

[5] E. ETSI, "Digital video broadcasting (dvb); interaction channel for satellite distribution systems," *ETSI EN*, vol. 301, p. 790, 2005.

[6] A. Ghosh, D. R. Wolter, J. G. Andrews, and R. Chen, "Broadband wireless access with wimax/802.16: current performance benchmarks and future potential," *IEEE Communications Magazine*, vol. 43, no. 2, pp. 129–136, Feb 2005.

[7] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEEE Electronics Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

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

[9] E. Perahia and R. Stacey, *Next generation wireless LANs: 802.11 n and 802.11 ac*. Cambridge university press, 2013.

[10] B. Bellalta, L. Bononi, R. Bruno, and A. Kassler, "Next generation ieee 802.11 wireless local area networks: Current status, future directions and open challenges," *Computer Communications*, vol. 75, pp. 1–25, 2016.

[11] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *Communications, IEEE Transactions on*, vol. 58, no. 6, pp. 1610–1614, 2010.

[12] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for ldpc codes," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3385–3400, Oct 2014.

[13] T. Tithi, C. Winstead, and G. Sundararajan, "Decoding ldpc codes via noisy gradient descent bit-flipping with re-decoding," *arXiv preprint arXiv:1503.08913*, 2015.

[14] T. Richardson, "Error floors of ldpc codes," in *Proceedings of the annual Allerton conference on communication control and computing*, vol. 41, no. 3. The University; 1998, 2003, pp. 1426–1435.

[15] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Gen03-6: Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *IEEE Globecom 2006*, Nov 2006, pp. 1–6.

[16] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based ldpc codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 181–201, Jan 2010.

[17] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for decoding LDPC codes," *Communications, IEEE Transactions on*, in press 2014b.

[18] F. Leduc-Primeau, S. Hemati, W. J. Gross, and S. Mannor, "A relaxed half-stochastic iterative decoder for ldpc codes," in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, Nov 2009, pp. 1–6.

[19] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *Signal Processing, IEEE Transactions on*, vol. 58, no. 9, pp. 4883–4896, 2010.

[20] Leduc-Primeau *et al.*, "Relaxed half-stochastic belief propagation," *Communications, IEEE Transactions on*, vol. 61, no. 5, pp. 1648–1659, May 2013.

[21] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. Gross, "Dithered belief propagation decoding," *Communications, IEEE Transactions on*, vol. 60, no. 8, pp. 2042–2047, 2012.

[22] K. Cushon *et al.*, "High-throughput energy-efficient LDPC decoders using differential binary message passing," *Signal Processing, IEEE Transactions on*, vol. 62, no. 3, pp. 619–631, Feb 2014.

[23] S. Zhang and C. Schlegel, "Controlling the error floor in LDPC decoding," *Communications, IEEE Transactions on*, vol. 61, no. 9, pp. 3566–3575, September 2013.

[24] D. J. C. MacKay. Encyclopedia of sparse graph codes. Accessed: 2014-06-20. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[25] Z. Zhang *et al.*, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circ.*, vol. 45, pp. 843–855, Apr. 2010.

[26] L. Trung, "New direction on low complexity implementation of probabilistic gradient descent bit-flipping decoder," Ph.D. dissertation, University of Cergy-Pontoise, 2017.

[27] C. A. Cole, S. G. Wilson, E. Hall, T. R. Giallorenzi *et al.*, "A general method for finding low error rates of ldpc codes," *arXiv preprint cs/0605051*, 2006.

[28] T. Sakai and K. Shibata, "Importance sampling for ldpc codes based on optimal simulation probability density function," in *2010 International Symposium On Information Theory Its Applications*, Oct 2010, pp. 389–393.

[29] E. Cavus, C. L. Haymes, and B. Daneshrad, "Low ber performance estimation of ldpc codes via application of importance sampling to trapping sets," *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1886–1888, July 2009.

[30] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of structured ldpc codes: deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 908–917, August 2009.

CURRICULUM VITAE

Tasnuva Tarannum Tithi

Published Journal Articles

- T. Tithi, B. Deka, R. Gerdes, C. Winstead, M. Li, and K. Heaslip. "Analysis of Friendly Jamming for Secure Location Verification of Vehicles". IEEE Transactions on Vehicular Technology, 2018.

- C. Winstead, T. Tithi, E. Boutillon, F. Ghaffari. "Recent Advances on Stochastic and Noise Enhanced Methods in Error Correction Decoders". 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC-2018).

- T. Tithi, R. Gerdes, C. Winstead. "Viability of Using Shadows Cast by Vehicles for Position Verification in Vehicle Platooning". (accepted) IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-17).

- T. Tithi, R. Gerdes, and C. Winstead. "Poster: Position Verification in Vehicular Platoons Using a Euclidean Distance Matrix". IEEE Symposium on Security and Privacy 2015.

- T. Tithi, C. Winstead, and G. Sundararajan. "Decoding LDPC codes via noisy gradient descent bit-flipping with re-decoding". arXiv preprint arXiv:1503.08913 (2015).