NOISY GRADIENT DESCENT BIT FLIP DECODING OF LOW DENSITY PARITY CHECK CODES: ALGORITHM AND IMPLEMENTATION

by

Gopalakrishnan Sundararajan

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

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Dr. Chris Winstead Major Professor	Dr. Jacob Gunther Committee Member
Dr. Sanghamitra Roy Committee Member	Dr. Reyhan Baktur Committee Member
Dr. Haitao Wang Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2016

Abstract

NOISY GRADIENT DESCENT BIT FLIP DECODING OF LOW DENSITY PARITY CHECK CODES: ALGORITHM AND IMPLEMENTATION

by

Gopalakrishnan Sundararajan, Doctor of Philosophy

Utah State University, 2016

Major Professor: Dr. Chris Winstead
Department: Electrical and Computer Engineering

A modified Gradient Descent Bit Flipping (GDBF) algorithm is proposed for decoding Low Density Parity Check (LDPC) codes on the binary-input Additive White Gaussian Noise (AWGN) channel. The new algorithm, called Noisy Gradient Descent Bit Flipping (NGDBF), introduces a random perturbation into each symbol metric at each iteration. The noise perturbation allows the algorithm to escape from undesirable local maxima, resulting in improved performance. A combination of heuristic improvements to the algorithm are proposed and evaluated. When the proposed heuristics are applied, NGDBF performs better than any previously reported GDBF variant, and comes within 0.5 dB of the Belief Propagation (BP) algorithm for several tested codes. Unlike other previous GDBF algorithms that provide an escape from local maxima, the proposed algorithm uses only local, fully parallel operations and does not require computing a global objective function or a sort over symbol metrics, making it highly efficient in comparison. The proposed NGDBF algorithm requires channel state information which must be obtained from a Signal to Noise Ratio (SNR) estimator. Architectural details are presented for implementing the NGDBF algorithm. Complexity analysis and optimizations are also discussed. The proposed algorithm is implemented on a code that is deployed in the Ethernet standard. The design

employs a fully parallel architecture and a flooding schedule. The design employs a two phase decoding approach. In the first phase, noise samples are processed and stored. In the second phase, the stored noise samples are read and are used in decoding. The design is implemented in a 65 nm Complementary Metal Oxide Semiconductor (CMOS) process. The implemented design is compared to other decoder designs and all the necessary performance parameters are measured. The comparisons show that the NGDBF design is more efficient in terms of area consumption compared to other designs. The design is also more energy efficient and has a better error performance compared to other designs that are not based on the Min-Sum (MS) algorithm.

(99 pages)

Public Abstract

NOISY GRADIENT DESCENT BIT FLIP DECODING OF LOW DENSITY PARITY CHECK CODES: ALGORITHM AND IMPLEMENTATION

by

Gopalakrishnan Sundararajan, Doctor of Philosophy

Utah State University, 2016

Major Professor: Dr. Chris Winstead
Department: Electrical and Computer Engineering

Transmission of bits across a medium occurs in the presence of noise. This leads to the distortion and corruption of the information that is transmitted. To overcome this, sophisticated procedures are employed during the reception of those transmitted bits to recover the original information. There are many ways to recover the original information from the corrupted bits. Usually, those procedures and methods are very complex. Hence, there is a greater need to come up with simple procedures and methods of recovering the original transmitted information from the corrupted bits. This work proposes a simple procedure and a physical implementation that is shown to be better in terms of various performance metrics like physical area, energy efficiency and number of computational units used.

To my parents and my siblings....

Acknowledgments

I would like to convey my sincere gratitude to my family, my friends, colleagues, and superiors for their love, support, and patience over the last few years. I would like to thank my advisor, Dr. Chris Winstead, for his encouragement, sound advice, good teaching, good company, lots of good ideas and exposing me to this very exciting area of error correction coding. Dr. Winstead has a clarity of thought on what is publishable research and what is not which I admire a lot. He also agreed to advise me when I was going through some tough times. I will be forever indebted to him for that and also for the countless ways he contributed to this project, but also for the many number of ways he supported me in all of my goals. This dissertation would not have been possible without his expert guidance. I have benefited greatly from the generosity and support of many faculty members. In particular, I would like to thank Dr. Sanghamitra Roy for her encouragement and for serving in my committee. I would like to acknowledge the fact that I was able to come To Utah State because of the funding that Dr. Roy provided to me in my admission letter. I would like to express my gratitude to my committee members: Dr. Jake Gunther, Dr. Haitao Wang, Dr. Reyhan Baktur. My graduate studies would not have been the same without the support and encouragement provided by all my student-colleagues and friends. I am indebted to Yi Luo for allowing me to work with him on the implantable project. I learned a great deal by working with him. I would also like to thank Tasnuva Tithi for her help. I would like to acknowledge the support from National Science Foundation (NSF) for providing our research group with the award 0954747, which funded a part of this dissertation work. I would also like to acknowledge the Sant Fellowship from the engineering school that funded the latter part of this research. I would like to thank Marylee Anderson and our department head Dr. Todd Moon for providing me the Sant fellowship. I would also like to thank Marylee for all of her help and support with the paperwork that has been done all these years. Last but not least, I would like to thank the support of my family members. My parents Sundararajan and Parvathy have endured lots of mental, physical and financial suffering in

helping me attain academic success. I dedicate this dissertation to them and thank them for all the ethics, integrity and values that they have inculcated in me during my childhood and during my teenage years. I thank my sisters Nithya and Aarthi and my brothers-in-law Narayan and Sriram for being second parents to me after I came to the States. I can't thank them enough on numerous occasions, during which they have helped me to sort out various matters and for being pillars of mental support during my masters and doctoral degrees.

Gopalakrishnan Sundararajan

Contents

Page

Abstract . . . . . **iii**

Public Abstract . . . . . **v**

Acknowledgments . . . . . **vii**

List of Tables . . . . . **xi**

List of Figures . . . . . **xii**

Acronyms . . . . . **xvi**

1 Introduction **1**
1.1 Contributions and Outline 3

2 Related Work **4**
2.1 LDPC Codes and Decoding Algorithms 4
2.2 Review of Bit Flipping Algorithms 7
2.3 Review of LDPC Decoder Architectures 12
2.4 Review of 10GBASE-T Ethernet Decoders 13

3 Proposed Noisy GDBF Algorithm and Simulation Results **16**
3.1 Notation 16
3.2 GDBF Algorithm 17
3.3 NGDBF Algorithm 19
3.4 Threshold Adaptation 20
3.5 Output Decision Smoothing 20
3.6 Simulation Results 22
3.6.1 Bit Error Rate (BER) Performance 22
3.6.2 Average Iterations per Frame 26
3.6.3 Sensitivity to Threshold Parameter 27
3.6.4 Sensitivity to Perturbation Variance 29

4 Architecture Considerations **33**
4.1 Implementing Threshold Adaptation 33
4.2 Simplification of Noise Sample Generation 35
4.3 Effects of Quantization on NGDBF Algorithm 37
4.4 Complexity Analysis 38
4.4.1 Comparison with Previous GDBF Algorithms 40
4.4.2 Comparison with MS Algorithm 41
4.4.3 Comparison with Stochastic Decoders 42
4.5 Local Maximum Likelihood Interpretation 43

5 ASIC Implementation of Noisy Gradient Descent Bit Flip Decoder For 10GBASE-T Ethernet Standard **48**
5.1 Error Rate Performance 48
5.2 Architecture of NGDBF Decoder 51
5.2.1 NUU Design 52
5.2.2 Symbol Node Design 59
5.2.3 Check Node and ETU Design 62

6 Implementation Results **63**

7 Conclusions and Future Work **72**

Appendices **79**
A Decoder ASIC Design Flow 80

Vita **81**

List of Tables

Table		Page
2.1	Summary and comparison of bit-flipping algorithms	8
4.1	Threshold adaptation events for $\theta = -0.9$, $\lambda = 0.99$, $Y_{\max} = 2.5$.	37
5.1	Input, output and control signals of the decoder and their widths. Q is number of quantized bits used for representing real-valued signals in the decoder.	54
5.2	Control table.	54
5.3	Table for selector operation.	57
5.4	Scaled syndrome sum lookup table.	60
6.1	Wire complexity of decoders.	64
6.2	NGDBF critical path.	68
6.3	Implementation results for NGDBF and comparison with other works.	70

List of Figures

Figure Page

2.1 Block diagram of a communication system showing LDPC encoding and decoding. 5

2.2 Parity check matrix of a rate 1/3 code. The codelength is 6. The row weight is 3 and the column weight is 2. 6

2.3 Tanner graph of the parity check matrix described in Fig. 2.2. 6

3.1 BER versus E_b/N_0 curves for S-NGDBF with $T = 100$ and $\eta = 1.0$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed S-NGDBF algorithm is indicated by an asterisk (*). 23

3.2 BER versus E_b/N_0 curves for M-NGDBF with $T = 100$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed M-NGDBF algorithms (adaptive and non-adaptive) are indicated by asterisks (*). Several other known algorithms are shown for comparison, including M-GDBF ($T = 100$), AT-GDBF ($T = 100$) and H-GDBF with escape process ($T = 300$). 24

3.3 BER versus E_b/N_0 curves for SM-NGDBF and H-GDBF with $T = 300$ using the rate 1/2 PEGReg504x1008 code over an AWGN channel with binary anitpodal modulation. The proposed algorithms are indicated by an asterisk (*). 25

3.4 BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 1/2 4000.2000.4.244 code over an AWGN channel with binary antipodal modulation. These results were obtained using $\theta = -0.9$, $\lambda = 0.99$, and η varied between 0.625 at low SNR (below 2.8) and 0.7 at higher SNR (above 2.8). . 25

3.5 BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 0.9356 4376.282.4.9598 code over an AWGN channel with binary antipodal modulation. Several other algorithms are also shown for comparison. These results were obtained using $\theta = -0.7$, $\eta = 0.65$ and $\lambda = 0.993$. The dynamic range and syndrome weighting were also modified for this simulation, using $Y_{\max} = 2.0$ and $w = 0.1875$. 26

3.6 Average number of iterations versus E_b/N_0 curves for existing GDBF and proposed NGDBF algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100 except for SM-NGDBF and H-GDBF, which have $T = 300$. The newly proposed algorithms are indicated by an asterisk (*). 28

3.7 Average number of iterations versus BER for the proposed M-NGDBF and SM-NGDBF algorithms. 28

3.8 Threshold sensitivity of the non-adaptive M-NGDBF algorithm with parameters $\lambda = 1.0$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code. 30

3.9 Threshold sensitivity of adaptive M-NGDBF algorithm with parameters $\lambda = 0.9$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code. 30

3.10 Average iterations for adaptive M-NGDBF as a function of the initial threshold parameter θ for the PEGReg504x1008 code. The remaining parameters are $\lambda = 0.9$ and $T = 100$. 31

3.11 Sensitivity of performance for M-NGDBF relative to the global adaptation parameter λ, with parameters $\theta = -0.9$, $T = 100$, $\eta = 0.96$, for the PEGReg504x1008 code. 31

3.12 Sensitivity of performance for SM-NGDBF relative to the noise scale parameter η. 32

4.1 Architecture of the NGDBF decoder. Gaussian-distributed noise samples are produced serially at the output of a Random Number Generator (RNG). The RNG requires inputs η and N_0, and the latter must be generated by a channel parameter estimator (not shown). A shift-register (SR) chain is used to distribute the random Gaussian samples that serve as the q_k perturbations. The symbol $\mathcal{P}_i$ indicates the set of syndrome messages that arrive at symbol node S_i, corresponding to the index set $\mathcal{N}(i)$. The symbol $\mathcal{X}_j$ is the set of messages that arrive at parity-check node P_j, corresponding to the index set $\mathcal{M}(j)$. 34

4.2 Symbol node schematic. F_1 is a toggle flip-flop. The s_i messages are locally indexed. The sgn operator refers to sign-bit extraction. The multiplication $\otimes$ is binary, as it applies only to the sign bit of $\tilde{y}_k$. 36

4.3 Check node schematic showing a tree of XNOR operations over $d_c - 1$ input messages. The x_i messages are locally indexed. 36

4.4 BER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$. 39

4.5 FER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$. 39

4.6 Example of evolution of the LML flip matrix Φ for a (3,6) LDPC code, with $Q = 4$ and an $E_b/N_0 = 3.50\,\text{dB}$. The initial value of P_e is 0.0672. Since Φ is symmetric with $\Phi(i, j) = \Phi(N_Q + 1 - i, d_v + 2 - j)$, only the top $N_Q/2$ rows are shown. 47

4.7 Evolution of the flip matrix for the weighted GDBF algorithm with threshold adaptation. Use of threshold adaptation and syndrome weighting achieves qualitative agreement with the LML flip decisions in Fig. 4.6. 47

5.1 BER for NGDBF compared to a benchmark offset MS decoder for the IEEE 802.3 standard LDPC code with maximum iterations limited to T. 49

5.2 Dominant $(4, 12)$ trapping set in M-GDBF algorithm. The parity checks indicated in black are unsatisfied parity checks. The parity checks in the bottom are satisfied parity checks. 50

5.3 Average number of iterations for NGDBF algorithm for IEEE 802.3 standard LDPC code with maximum iterations limited to T. 50

5.4 Top level decoder architecture showing all the blocks. Input, output and all the control signals are also clearly shown. $d_v(6)$ messages arrive at the symbol node from the interleaver and $d_c(32)$ messages arrive at a check node from the interleaver during a decoding iteration. Symbol node update requires $Q - 1$ bits of noise from the NUU every iteration. 53

5.5 Timing diagram of the decoder. The start of the two operational phases is clearly shown. At $t = 2649$, the decoding operation begins and the first frame is loaded into the decoder. 54

5.6 Architecture of NUU. 55

5.7 Shift registers containing noise samples. 56

5.8 Seven bit sign-magnitude multiplier. Mag corresponds to magnitude and Sgn corresponds to the sign. 57

5.9 Seven bit sign-magnitude adder. Mag corresponds to magnitude and Sgn corresponds to the sign. 58

5.10 Architecture of a symbol node k. Mag corresponds to magnitude and Sgn corresponds to the sign. 60

5.11 Architecture of count module. The module produces a three bit output ($C0 - -C2$). FA indicates a full adder and HA indicates a half adder. . . . 61

5.12 Logic diagram of the lookup table. 61

6.1 Layout view of NGDBF decoder implementation. 64

6.2 BER for routed NGDBF compared to a benchmark offset MS decoder for the IEEE 802.3 standard LDPC code with maximum iterations limited to T. . . 65

6.3 Average number of iterations and average throughput versus E_b/N_0 for routed NGDBF design for IEEE 802.3 standard LDPC code with maximum iterations limited to T. 66

6.4 Energy per bit consumption for routed NGDBF design for IEEE 802.3 standard LDPC code with variation in E_b/N_0. 66

6.5 Critical path showing the start, middle and the end blocks. The actual critical path is highlighted using thick broken lines. 67

6.6 Energy efficiency versus area efficiency of all the reported 10GBASE-T decoders. The ellipse indicates the ideal region. 71

A.1 Design flow 80

Acronyms

LDPC	Low Density Parity Check
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
SNR	Signal to Noise Ratio
CMOS	Complementary Metal Oxide Semiconductor
BSC	Binary Symmetric Channel
BFA	Bit Flipping Algorithm
GBPS	Giga-Bit Per Second
BP	Belief Propagation
MS	Min-Sum
ML	Maximum-Likelihood
PBFA	Probabilistic Bit Flipping Algorithm
WBF	Weighted Bit-Flipping
PWBF	Parallel Weighted Bit-Flipping
MWBF	Modified Weighted Bit-Flipping
IGDBF	Improved Gradient Descent Bit Flipping
IMWBF	Improved Modified Weighted Bit-Flipping
RRWGDBF	Reliability-Ratio Weighted GDBF
GDBF	Gradient Descent Bit Flipping
AT-GDBF	Adaptive Threshold Gradient Descent Bit Flipping
ES-ATGDBF	Early Stopping-Adaptive Threshold Gradient Descent Bit Flipping
H-GDBF	Hybrid Gradient Descent Bit Flipping
IGDBF	Improved Gradient Descent Bit Flipping
M-GDBF	Multi-bit Gradient Descent Bit Flipping
S-GDBF	Single-bit Gradient Descent Bit Flipping
S-NGDBF	Single-bit Noisy Gradient Descent Bit Flipping
M-NGDBF	Multi-bit Noisy Gradient Descent Bit Flipping

SM-NGDBF	Smoothed Multi-bit Noisy Gradient Descent Bit Flipping
IDB	Improved Differential Binary
TFM	Tracking Forecast Memory
MTFM	Majority-based Tracking Forecast Memory
MDD-BMP	Modified Differential Decoding with Binary Message Passing
IEEE	Institute of Electrical and Electronics Engineers
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
LML	Local Maximum-Likelihood
MSB	Most Significant Bit
CLA	Carry Look Ahead
CSAM	Carry Save Array Multiplier
NUU	Noise Update Unit
SNU	Symbol Node Unit
CNU	Check Node Unit
ETU	Early Termination Unit
QC	Quasi-Cyclic
HA	Half Adder
FA	Full Adder
NSF	National Science Foundation

Chapter 1

Introduction

LDPC codes have gained considerable research attention in recent years. Due to their powerful decoding performance, LDPC codes are increasingly deployed in communication standards. The performance and the cost of using LDPC codes are partly determined by the choice of decoding algorithm. LDPC decoding algorithms are usually iterative in nature. They operate by exchanging messages between basic processing nodes. Among the various decoding algorithms, the soft decision BP algorithm and the approximate MS algorithm offer the best performance on the binary-input AWGN channel, but these algorithms require a large number of arithmetic operations repeated over many iterations [1,2]. These operations must be implemented with some degree of parallelism in order to support the throughput requirements of modern communication systems [3, 4]. As a result, LDPC decoders can be highly complex devices.

Significant effort has been invested to develop reduced-complexity decoding algorithms known "bit-flipping" decoders. These algorithms are similar in complexity to hard-decision decoding algorithms, but obtain improved performance by accounting for soft channel information. In most bit-flipping algorithms, the symbol node updates are governed by an *inversion function* that estimates the reliability of received channel samples. The inversion function includes the received channel information, in addition to the hard-decision syndrome components obtained from the code's parity-check equations. In the so-called *single* bit-flipping algorithms, the least reliable bit is flipped during each iteration. In *multiple* bit-flipping algorithms, any bit is flipped if its reliability falls below a designated threshold, hence multiple bits may be flipped in parallel, allowing for faster operation. A recently emerged branch of the bit-flipping family is GDBF, which formulates the inversion function as a gradient descent problem. GDBF algorithms demonstrate a favorable trade-off be-

tween performance and complexity relative to other bit-flipping algorithms. One difficulty for GDBF algorithms is that they are affected by undesirable local maxima which cause the decoder to converge on an erroneous message. Various schemes have been proposed to avoid or escape local maxima, but require additional complexity due to multiple thresholds or computing a global function over the code's entire block length.

In this work, an improved version of the GDBF algorithm called NGDBF is proposed, that offers a low-complexity solution to escape spurious local maxima. The proposed method works by introducing a random perturbation to the inversion function. The resulting algorithm provides improved performance and requires only local operations that can be executed fully in parallel (except for a global binary stopping condition). The proposed NGDBF algorithm comprises a set of heuristic methods that are empirically found to provide good performance for typical codes. Simulation results indicate that the NGDBF's optimal noise variance is proportional to the channel noise variance. This introduces a possible drawback compared to previous GDBF algorithms: NGDBF requires knowledge of the channel noise variance, which must be obtained from an estimator external to the decoder. Because of the heuristic nature of these results, this dissertation is organized to present the algorithm's technical details and empirical results first, followed by theoretical analyses that provide explanations for some of the observed results.

In the second part of this dissertation, a hardware implementation of the NGDBF algorithm is done on a code that is deployed in IEEE (Institute of Electrical and Electronics Engineers) 802.3*an* Ethernet standard. This design employs a fully parallel architecture and operates in two-phases: start-up phase and decoding phase. The two phase operation keeps the high latency operations off-line, thereby reducing the decoding latency during the decoding phase. The design is bench-marked with other state-of-the-art designs on the same code, that employ different algorithms and architectures. The results indicate that the NGDBF decoder has a better area efficiency and a better energy efficiency compared to other state-of-art decoders. When the design is operated in medium to high SNR, the design is able to provide greater than the required minimum throughput of 10 GigaBit Per

Second (GBPS).

1.1 Contributions and Outline

This main objectives of this research are as follows:

- To survey the existing literature on the state of the art LDPC decoding algorithms.
- Propose a novel LDPC decoding algorithm that has high performance but incurs low complexity, that could serve as an alternative to other state-of-the-art LDPC decoding algorithms.
- Analyze the complexity of the proposed algorithm and propose a suitable architecture for the digital implementation of the proposed algorithm.
- Demonstrate an Application Specific Integrated Circuit (ASIC) proof-of-concept implementation of the proposed algorithm for a commercial standard LDPC code, measure its performance and compare them to the other existing state-of-the-art designs.

The remainder of this dissertation is organized as follows: Chapter 2 discusses the related work on bit-flipping algorithms, as well as some recently reported decoding algorithms that benefit from noise perturbations. Chapter 3 describes the notation, summarizes the proposed NGDBF algorithm and its heuristic modifications and also analyses the simulation results. Chapter 4 describes the architecture and also analyses the complexity of the NGDBF algorithm. Chapter 5 describes the architecture of the NGDBF 10GBASE-T decoder. The implementation results are detailed in chapter 6. Conclusions and future work are presented in chapter 7.

Chapter 2

Related Work

In this chapter, a brief review of LDPC codes and decoding algorithms are presented. The recent work on bit-flipping algorithms and other methods related to the new NGDBF algorithms are reviewed. A brief survey of existing LDPC decoder architectures is also presented. The recent work on the implementation of 10GBASE-T Ethernet LDPC decoders is also discussed and the key features of each of those designs is also summarized.

2.1 LDPC Codes and Decoding Algorithms

Fig. 2.1 shows the block diagram of a communication system that involves LDPC encoding and decoding operations. In this system, message u from a data source containing k bits is obtained and redundant bits of length $n-k$ are added to the message vector to obtain a encoded codeword c, which is n bits in length. The encoding operation is described by a matrix multiplication $c = m \times G$, where G is the generator matrix that has dimensions of $k \times n$. The encoded codeword c is modulated to obtain a vector $\hat{c}$. $\hat{c}$ is transmitted through a noisy channel. The receiver acquires the noisy frame. The receiver also makes a hard or a soft decision on the received frame to obtain hard or soft information respectively. This information is fed to the LDPC decoder. The LDPC decoder tries to estimate original encoded codeword. Decoding is successful if the decoder can exactly recover the original encoded codeword from the received frame.

An LDPC code is defined by a parity check matrix H. An (n, k) LDPC code has a parity check matrix with $n-k$ rows and n columns. The elements of H describe the parity check constraint relationships between the n received symbols and the $n-k$ parity checks. Fig. 2.2 shows the parity check matrix for the code which has a rate of 1/3. There are 4 rows and 6 columns. A bit j participates in check i if $H_{i,j} = 1$. The row and column weights

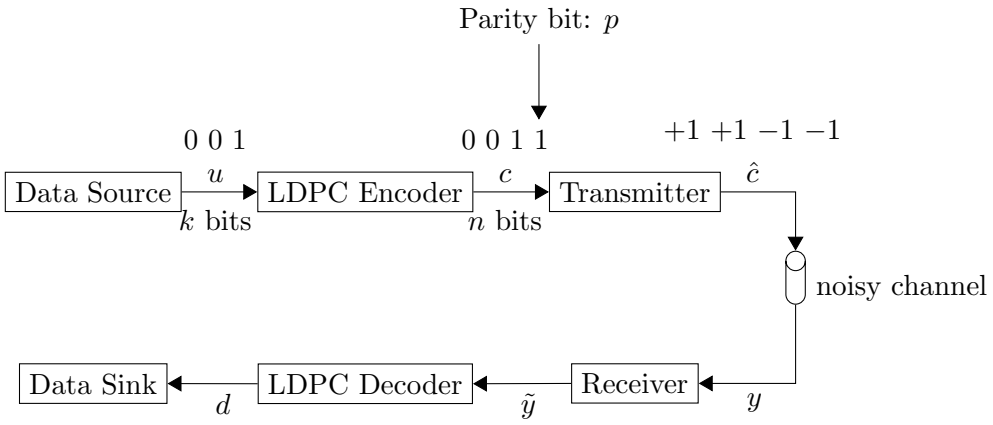

Fig. 2.1: Block diagram of a communication system showing LDPC encoding and decoding.

are defined as the number of non-zero entries in a given row or a column, respectively. If number of non-zeros in all the rows are the same and number of non-zeros in all the columns are same then the code is said to be regular. Otherwise, it is irregular. Another way to describe a LDPC code is by representing the code as a Tanner graph. Fig. 2.3 shows the Tanner graph of the H matrix described in Fig. 2.2. A Tanner graph is a bipartite graph that consists of symbol nodes and check nodes. The symbol nodes represent the columns of the parity check matrix and the checks represent the rows of the parity check matrix. An edge between a symbol node and a check node exists if there is a corresponding non-zero entry in the parity check matrix. The symbol node degrees and the check node degrees are equivalent to the column weight and the row weight of the parity check matrix, respectively.

LDPC decoding can be viewed as iterative message passing between symbol nodes and the check nodes on the edges of the Tanner graph. Each iteration in a message passing algorithm involves the following four steps:

1. (**symbol-to-check)** Each symbol node sends a message to each adjacent check node it is connected to.

2. (**check processing)** Each check node then computes the consistency of incoming messages.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Fig. 2.2: Parity check matrix of a rate 1/3 code. The codelength is 6. The row weight is 3 and the column weight is 2.

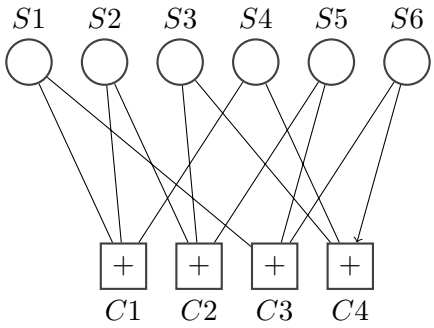

Fig. 2.3: Tanner graph of the parity check matrix described in Fig. 2.2.

3. (**check-to-symbol)** Each check node then sends a message to each adjacent symbol node it is connected to.

4. (**symbol processing)** Each symbol node updates its value.

Message passing algorithm can be classified into two types depending on the receiver and the type of messages that are exchanged during iterative decoding. The first type is hard decision Bit Flipping Algorithm (BFA) which is typically deployed in hard information receivers. In this algorithm, bits that do not satisfy the parity check constraints are continuously flipped during the decoding operation. BFA is sub-optimal and provides low error correcting performance. The second type is the soft decision BP algorithm in which soft information is used during the decoding process. In this algorithm, probabilities of a given

bit being a one or zero is exchanged along the edges of the Tanner graph. BP algorithms are computationally more complex and offer better performance compared to the BFA.

2.2 Review of Bit Flipping Algorithms

A qualitative summary of the considered bit-flipping algorithms and their comparative characteristics are provided in Table 2.1. The performance and complexity comparisons in Table 2.1 are qualitative estimates made solely within the family of bit-flipping algorithms. The original BFA was introduced by Gallager in his seminal work on LDPC codes [5]. Gallager's BFA is a hard-decision algorithm for decoding on the Binary Symmetric Channel (BSC), in which only hard channel bits are available to the decoder. To correct errors, the BFA computes a sum over the adjacent parity-check equations for each bit in the code. If, for any bit, the number of adjacent parity violations exceeds a specified threshold, then the bit is flipped. This process is repeated until all parity checks are satisfied, or until a maximum iteration limit is reached. The BFA has very low complexity since it only requires, in each iteration, a summation over binary parity-check values for each symbol. However, the BFA provides a weak decoding performance. Miladinovic et al. considered a Probabilistic Bit Flipping Algorithm (PBFA) which adds randomness to the bit-flip decision, resulting in improved performance [6]. In PBFA, when a bit's parity-check sum crosses the flip threshold, it is flipped with probability p. The parameter p is optimized empirically and is adapted towards one during successive iterations. Kou et al. introduced the Weighted Bit-Flipping (WBF) algorithm which improves performance over the BFA by incorporating soft channel information, making it better suited for use on the AWGN channel and other soft-information channels [2].

In the WBF algorithm, all parity-check results are weighted by a magnitude that indicates reliability. For each parity-check, the weight value is obtained by finding the lowest magnitude in the set of adjacent channel samples. During each iteration, a summation E_k is computed over the adjacent weighted parity-check results for each symbol position k. The symbol with the maximum E_k (or minimum, depending on convention) is flipped. The weights are only calculated once, at the start of decoding, however the WBF algorithm

Table 2.1: Summary and comparison of bit-flipping algorithms

Algorithm	Performance	Schedule	Arithmetic Complexity
BFA [5]	Poor	Parallel	Minimum
PBFA [6]	Fair	Parallel	Low
WBF [2]	Fair	Serial	Moderate
MWBF [7]	Good	Serial	Moderate
IMWBF [8]	Excellent	Serial	High
PWBF [9]	Excellent	Parallel	High
S-GDBF [10]	Fair	Serial	Low
M-GDBF [10]	Good	Mixed	Low
H-GDBF [10]	Excellent	Mixed	High
AT-GDBF [11]	Good	Parallel	Low
IGDBF [12]	Excellent	Mixed	Moderate
RRWGDBF [13]	Excellent	Parallel	High
Stochastic MTFM [14]	Excellent	Parallel	Low
*S-NGDBF	Fair	Serial	Low
*M-NGDBF	Good	Parallel	Low
*SM-NGDBF	Excellent	Parallel	Low-Moderate

*New algorithms described in this work.

requires at every iteration a summation over several weights for each symbol — a substantial increase in complexity compared to the original BFA. In addition to the increased arithmetic complexity, WBF has two major drawbacks: first, a potentially large number of iterations are required because only one bit may be flipped in each iteration. Second, the algorithm must perform a global search to find the maximum E_k out of all symbols, resulting in a large latency per iteration that increases with codeword length, thereby hindering a high-throughput implementation. Researchers introduced several improvements to the WBF. Zhang et al. introduced the Modified Weighted Bit-Flipping (MWBF) algorithm, which obtained improved performance with a slight increase in complexity. Jiang et al. described another Improved Modified Weighted Bit-Flipping (IMWBF) algorithm which offered further improvement by using the parity-check procedure from the MS algorithm to determine the parity-check weights — another substantial increase in complexity. Both of these methods inherit the two key drawbacks associated with single-bit flipping in the WBF algorithm.

Recently, Wu et al. introduced a Parallel Weighted Bit-Flipping (PWBF) algorithm,

which reduces the drawbacks associated with single-bit flipping in the other WBF varieties [9]. In the PWBF algorithm, the maximum (or minimum) E_i metric is found within the subset of symbols associated with each parity-check. Wu et al. also developed a theory relating PWBF to the BP and MS algorithms, and showed that PWBF has performance comparable to IMWBF [15]. In the PWBF algorithm, it is still necessary to find the maximum E_k from a set of values, which costs delay, but the set size is significantly reduced compared to the other WBF methods, and it is independent of codeword length. In spite of these improvements, PWBF retains the complex arithmetic associated with IMWBF.

To reduce the arithmetic complexity of bit-flipping algorithms, Wadayama et al. devised the GDBF algorithm as a gradient-descent optimization model for the ML decoding problem [10]. Based on this model, the authors of [10] obtained single-bit and multi-bit flipping algorithms that require mainly binary operations, similar to the original BFA. The GDBF methods require summation of binary parity-check values, which is less complex than the WBF algorithms that require summation over independently weighted syndrome values. The Single-bit-Gradient-Descent Bit Flipping algorithm (S-GDBF) requires a global search to discover the least reliable bit at each iteration. The Multi-bit-Gradient Descent Bit Flipping algorithm (M-GDBF) uses local threshold operations instead of a global search, hence achieving a faster initial convergence. In practice, the M-GDBF algorithm did not always provide stable convergence to the final solution.

To improve convergence, the authors of [10] adopted a mode-switching strategy in which M-GDBF decoding is always followed by a phase of S-GDBF decoding, leveraging high-speed in the first phase and accurate convergence in the second. Although the mode-switching strategy provided a significant benefit, the algorithm was still subject to spurious local maxima. Wadayama et al. obtained further improvements by introducing a Hybrid-Gradient Descent Bit Flipping (H-GDBF) algorithm with an escape process to evade local maxima. The H-GDBF algorithm obtains performance comparable to MS, but the escape process requires evaluating a global objective function across all symbols. When the objective function crosses a specified threshold during the S-GDBF phase, the decoder

switches back to M-GDBF mode, then back to S-GDBF mode, and so on until a valid result is reached. To date, H-GDBF is the best performing GDBF variant, but requires a maximum of 300 iterations to obtain its best performance, compared to 100 for M-GDBF and S-GDBF. The major disadvantages of this algorithm are its use of multiple decoding modes, the need to optimize dual thresholds for mode switching and bit flipping, the global search operation and the global objective function used for mode switching. These global operations require an arithmetic operation to be computed over the entire code length, and would be expensive to implement for practical LDPC codes with large codeword length. Several researchers proposed alternative GDBF algorithms in order to obtain fully parallel bit-flipping and improved performance. Ismail et al. proposed an Adaptive Threshold Gradient Descent Bit Flipping (ATGDBF) algorithm that achieves good performance without the use of mode-switching, allowing for fully-parallel operation [11]. The same authors also introduced an Early Stopping-Adaptive Threshold-Gradient Descent Bit Flipping (ES-AT-GDBF) algorithm that significantly reduces the average decoding iterations at lower SNR.

Phromsa-ard et al. proposed a more complex Reliability-Ratio Weighted Gradient Descent Bit Flipping (RRWGDBF) algorithm that uses a weighted summation over syndrome components with an adaptive threshold to obtain reduced latency [13]. The RRWGDBF method has the drawback of increased arithmetic complexity because it performs a summation of weighted syndrome components, similar to previous WBF algorithms. Haga et al. proposed an Improved Gradient Descent Bit Flipping (IGDBF) algorithm that performs very close to the H-GDBF algorithm, but requires a global sort operation to determine which bits to flip [12]. These GDBF algorithms can be divided into two classes: First, the low-complexity class, which includes S-GDBF and M-GDBF, AT-GDBF and RRWGDBF; in this class, mode-switching M-GDBF is the best performer. For low-complexity algorithms, the typical maximum number of iterations is reported as $T = 100$. Second is the high-performance class, which includes H-GDBF and IGDBF. In the high-performance class, significant arithmetic complexity is introduced and a larger number of iterations is re-

ported, $T = 300$. H-GDBF is the best performer in this class, and in this work is considered as a representative of the high-performance GDBF algorithms.

In this work, a new Noisy GDBF algorithm with single-bit and multi-bit versions (S-NGDBF and M-NGDBF, respectively) is proposed. The M-NGDBF algorithm proposed in this work employs a single threshold and also provides an escape from the neighborhood of spurious local maxima, but does not require the mode-switching behavior used in the original M-GDBF. The proposed algorithm also avoids using any sort or maximum-value operations. When using the threshold adaptation procedure borrowed from AT-GDBF, as described in Section 3.4, the proposed M-NGDBF achieves performance close to the H-GDBF and IGDBF methods at high SNR, with a similar number of iterations. A new method called Smoothed Multi-bit Noisy Gradient Descent Bit Flipping (SM-NGDBF) is also introduced, that contributes an additional 0.3 dB gain at the cost of additional iterations. It should be noted that Wadayama et al. proposed using a small random perturbation in the H-GDBF thresholds [10]; the NGDBF methods use a larger perturbation in combination with other heuristics to obtain good performance with very low complexity. Because of its reliance on pseudo-random noise and single-bit messages, the proposed NGDBF algorithms bear some resemblance to the family of stochastic iterative decoders that were first introduced by Gaudet and Rapley [16].

Winstead et al. introduced stochastic decoding for codes with loopy factor graphs [17], and Tehrani et al. later demonstrated stochastic decoding for LDPC codes [18, 19]. High throughput stochastic decoders have been more recently demonstrated by Sharifi Tehrani et al. [14,20,21] and by Onizawa et al. [22]. Stochastic decoders are known to have performance very close to BP, allow for fully-parallel implementations, and use very simple arithmetic while exchanging single-bit messages. They may therefore serve as an appropriate benchmark for comparing complexity against the proposed SM-NGDBF algorithm. In addition to recent work on low-complexity decoding, there has also been some exploration of noise-perturbed decoding using traditional MS and BP algorithms. Leduc et al. demonstrated a beneficial effect of noise perturbations for the BP algorithm, using a method called dithered

belief propagation [23]. Kameni Ngassa et al. examined the effect of noise perturbations on MS decoders and found beneficial effects under certain conditions [24]. The authors of [25] offered the conjecture that noise perturbations assist the MS algorithm in escaping from spurious fixed-point attractors, similar to the hypothesis offered in this work to motivate the NGDBF algorithm. Up to now, there is not yet a developed body of theory for analyzing noise-perturbed decoding algorithms, and the recent research on this topic tends to adopt a heuristic approach. In this work, a heuristic approach is adopted, and through empirical analysis it is shown that noise perturbations improve the performance of GDBF decoders.

2.3 Review of LDPC Decoder Architectures

Generally, the architectures of existing LDPC decoders can broadly be classified into three major categories: fully serial, partially parallel and fully parallel. All the above mentioned architectures have their merits and demerits. LDPC decoder implementation is a trade-off between error correction performance, hardware complexity and decoding throughput. Fully serial decoders have two process blocks and a memory unit [26]. They process one word at a time using a single symbol node and a check node processor. Although they employ less hardware, they suffer from large decoding latency and low throughput. Partially parallel decoders contain multiple processing units and shared memories [3] [27] [28]. The major challenge with partially parallel memories is handling simultaneous memory updates into shared memories. Partially parallel implementations are well suited for certain classes of codes like the Quasi-Cyclic (QC) codes that have some regularity in their matrix structure. Latency and throughput of partially parallel decoders are much better compared to fully serial decoders. Fully parallel decoders on the other hand map every unit in the code's Tanner graph to a hardware unit [29] [30] [31]. They provide high throughput and require no memories to store any messages. They also result in large area and have high routing congestion compared to fully serial and partially parallel decoders.

LDPC decoder architectures can also be classified according to the schedule in which the check and the symbol node messages are updated during a decoding iteration. The two common scheduling methods are flooding schedule and layered decoding schedule. In

flooding schedule, two-phase message passing is deployed [31]. In the first phase, check messages are updated and sent to corresponding neighborhood symbol node units. In the second phase, symbol messages are updated and sent to their corresponding check node neighbors. Second phase will not start until the first phase completes and there is no overlap between the two phases. In layered decoding, parity-check matrix is split into multiple layers either horizontally or vertically [28]. In the case of horizontal layering, matrix is split into many horizontal layers. The check node messages from one layer are passed to all other unprocessed layers that are connected to the same symbol node. During each iteration, horizontal layers are processed sequentially from the top to the bottom. In contrast to the flooding schedule, layered schedule offers faster convergence and is inherently well suited for resource sharing.

2.4 Review of 10GBASE-T Ethernet Decoders

The recent work on 10GBASE-T Ethernet LDPC decoder designs could be characterized by two key aspects: The first one is the algorithm implemented and the second one is the architecture. In this regard, five main designs are reviewed in this section. The first one is the offset MS decoder that was fabricated on a 65 nm CMOS process [32]. This decoder employs a partially parallel and a pipe-lined architecture that provides a moderate throughput. This design is very complex, incurs a large area and has a high energy consumption. However, because of the superiority of the offset MS algorithm, it provides a very good error performance.

The next design that was proposed is the fully parallel split-row MS algorithm [29]. This design implements a low complexity version of the Normalized MS algorithm that significantly reduces the routing complexity. The key idea behind the split-row MS algorithm is to partition the original parity check matrix into many sub-matrices, thereby splitting a row processing operation, into multiple row processing operations. Check node computations for each sub-matrix are performed separately, using limited information from other columns. This reduces the routing congestion as it reduces the number of wires between the row and the column processors. However, when the original matrix is broken into 16

sub-matrices, there is a significant performance degradation of 0.35 dB. This design consumes less area compared to the offset MS decoder and is highly energy efficient compared to the offset MS decoder.

Cevrero et al. proposed a layered implementation of the offset MS algorithm [28]. In this design, original parity check matrix of the 10GBASE-T code is split into six layers in which each layer has 64 rows and 2048 columns. This enables the check node operation to be time multiplexed and the check node processor to be shared across layers. Since only 64 check node processors are active at a time, only 64 check node processors are needed to accomplish successful decoding. Another advantage of the layered implementation is that it provides faster convergence. This design was fabricated in a 90 nm CMOS and achieved a throughput close to the required specification. This design consumes more than a watt of power and is inferior to Zhang's offset MS decoder in terms of energy efficiency.

As an alternative to the offset MS decoding, Tehrani et al. proposed a stochastic Majority-based Tracking Forecast Memory (MTFM) based 10GBASE-T Ethernet LDPC decoder [30]. This design was done in a 90 nm process. This design uses a fully parallel architecture and consumes a smaller area compared to both the offset MS and the split-row MS designs. This design has a better error performance than the split-row MS design. However, the stochastic MTFM decoder still has a significant complexity due to the requirement of a large number of random number generators to convert probabilities to streams of random numbers. All of the above mentioned designs, employ algorithms that are variants of the BP algorithm and are complex.

As an energy efficient alternative to the stochastic decoder, Cushon et al. recently proposed a low complexity design that employs a binary message passing algorithm and was implemented in a 65 nm CMOS process [31]. The algorithm termed Improved Differential Binary (IDB) algorithm consists of simple check and symbol node operations. IDB is a variant of the Modified Differential Decoding-Binary Message Passing (MDD-BMP) algorithm in which binary message passing is employed and is much simpler compared to the MS and the stochastic decoding algorithms [33]. MDD-BMP performs very poorly on

the 10GBASE-T code. So, the authors proposed two modifications to the MDD-BMP algorithm to improve its performance. They are degeneration and relaunching. Degeneration is method in which the symbol node function is modified to enable the MDD-BMP escape the effects of weak absorbing sets. Relaunching is a technique in which failed frames are decoded in successive attempts, with subtle changes to the initial state of the decoder. These changes are deterministic and are based on a look-up table. The IDB decoder employs a fully parallel architecture and renders a very high throughput. It also has a better error correcting performance than the split-row MS algorithm and is the most efficient in terms of area consumption and energy dissipation compared to all other previous 10GBASE-T decoders.

Chapter 3

Proposed Noisy GDBF Algorithm and Simulation Results

3.1 Notation

Let $\mathbf{H}$ be a binary $m \times n$ parity check matrix, where $n > m \geq 1$. To $\mathbf{H}$ is associated a binary linear code defined by $\mathcal{C} \triangleq \{c \in F_2^n : \mathbf{Hc} = 0\}$, where F_2 denotes the binary Galois field. The set of bipolar codewords, $\hat{\mathcal{C}} \subseteq \{-1, +1\}^n$, corresponding to $\mathcal{C}$ is defined by $\hat{\mathcal{C}} \triangleq \{(1-2c_1), (1-2c_2), ..., (1-2c_n) : c \in \mathcal{C}\}$. Symbols are transmitted over a binary input AWGN channel defined by the operation $\mathbf{y} = \hat{\mathbf{c}} + \mathbf{z}$, where $\hat{\mathbf{c}} \in \hat{\mathcal{C}}$, $\mathbf{z}$ is a vector of independent and identically distributed Gaussian random variables with zero mean and variance $N_0/2$, N_0 is the noise spectral density, and $\mathbf{y}$ is the vector of samples obtained at the receiver. A decision vector $\mathbf{x} \in \{-1, +1\}^n$ is defined. $\mathbf{x}(t)$ is the decision vector at a specific iteration t, where t is an integer in the range $[0, T]$, and T is the maximum number of iterations permitted by the algorithm. In iterative bit-flipping algorithms, the decision values may be flipped one or more times during decoding. The dependence on t is often omitted when there is no ambiguity. The decision vector is initialized as the sign of received samples, i.e. $x_k(t=0) = \text{sign}(y_k)$ for $k = 1, 2, \ldots, n$. The parity-check neighborhoods are defined as $\mathcal{N}(i) \triangleq \{j : h_{ij} = 1\}$ for $i = 1, 2, \ldots, m$, where h_{ij} is the $(i, j)^{\text{th}}$ element of the parity check matrix $\mathbf{H}$. The symbol neighborhoods are defined similarly as $\mathcal{M}(j) \triangleq \{i : h_{ij} = 1\}$ for $j = 1, 2, \ldots, n$. The code's parity check conditions can be expressed as bipolar syndrome components $s_i(t) \triangleq \prod_{j \in \mathcal{N}(i)} x_j(t)$ for $i = 1, 2, \ldots, m$. A parity check node is said to be *satisfied* when its corresponding syndrome component is $s_i = +1$.

3.2 GDBF Algorithm

The GDBF algorithm proposed in [10] was derived by considering the maximum likelihood problem as an objective function for gradient descent optimization. The standard ML decoding problem is to find the decision vector $\mathbf{x}_{\mathrm{ML}} \in \hat{\mathcal{C}}$ that has maximum correlation with the received samples $\mathbf{y}$:

$$\mathbf{x}_{\mathrm{ML}} = \arg\max_{\mathbf{x}\in\hat{\mathcal{C}}} \sum_{k=1}^{n} x_k y_k. \tag{3.1}$$

In order to include information from the code's parity check equations, the syndrome components are introduced as a penalty term, resulting in the objective function proposed by Wadayama et al.:

$$f(\mathbf{x}) = \sum_{k=1}^{n} x_k y_k + \sum_{i=1}^{m} s_i. \tag{3.2}$$

In the GDBF algorithm, a stopping criterion is used to enforce the condition $\mathbf{x} \in \hat{\mathcal{C}}$, i.e. any allowable solution $\mathbf{x}$ must be a valid codeword. Under this constraint, a solution that maximizes the objective function (3.2) is also a solution to the ML problem defined by (3.1). This is because for any valid codeword $\mathbf{x}$, the summation $\sum_{i=1}^{m} s_i$ is constant and equal to m. Since the objective functions in (3.1) and (3.2) differ only by a constant term, they must have the same maxima and minima. By taking the partial derivative with respect to a particular symbol x_k, the local inversion function is obtained as

$$E_k = x_k \frac{\partial f(\mathbf{x})}{\partial x_k} = x_k y_k + \sum_{i\in\mathcal{M}(k)} s_i. \tag{3.3}$$

Wadayama et al. showed that the objective function can be increased by flipping one or more x_k with the most negative E_k values. The resulting iterative maximization algorithm is described as follows:

1. Compute syndrome components $s_i = \prod_{j\in\mathcal{N}(i)} x_j$, for all $i \in \{1, 2, ..., m\}$. If $s_i = +1$ for all i, output x and stop.

2. Compute inversion functions. For $k \in \{1, 2, \ldots, n\}$ compute

$$E_k = x_k y_k + \sum_{i \in \mathcal{M}(k)} s_i \tag{3.4}$$

3. Bit-flip operations. Perform one of the following:

   (a) Single-bit version: Flip the bit x_k for $k = \arg\min_{k \in \{1, 2, \ldots, n\}} E_k$.

   (b) Multi-bit version: Flip any bits for which $E_k < \theta$, where $\theta \in \mathbb{R}^-$ is the *inversion threshold*.

4. Repeat steps 1 to 3 till a valid codeword is detected or maximum number of iterations is reached.

The inversion threshold is a negative real number, i.e. $\theta < 0$, to ensure that only bits with negative-valued E_k are flipped. The optimal value of θ is found empirically, as discussed in Section 3.6.3. The single-bit Gradient Descent Bit Flipping (S-GDBF) algorithm incurs a penalty in parallel implementations due to the requirement of finding the minimum from among n values. The multi-bit Gradient Descent Bit Flipping (M-GDBF) algorithm is trivially parallelized, but does not converge well because there tend to be large changes in the objective function after each iteration. The objective function increases rapidly during initial iterations, but is not able to obtain stable convergence unless a mechanism is introduced to reduce the flipping activity during later iterations. In this work, two such mechanisms are considered: *smoothing* and *adaptive thresholds*.

To improve performance, the authors of [10] proposed a mode-switching modification for M-GDBF, controlled by a parameter $\mu \in \{0, 1\}$: During a decoding iteration, if $\mu = 1$ then step 3b is executed; otherwise step 3a is executed. At the start of decoding, μ is initialized to 1. After each iteration, the global objective function (3.2) is evaluated. If, during any iteration t, $f(\mathbf{x}(t)) < f(\mathbf{x}(t-1))$, then μ is changed to 0. This modification

adds complexity to the algorithm, but also significantly improves performance. In the sequel (Section 3.4), it is explained that AT-GDBF eliminates the need for mode-switching by using the strictly parallel mechanism of adaptive thresholds [11].

3.3 NGDBF Algorithm

In order to provide a low-complexity mechanism to escape from local maxima in the GDBF algorithm, a random perturbation is introduced in the inversion function. Based on this approach, the step 2 of the GDBF algorithm is modified as follows:

Symbol node update. For $k = 1, 2, \ldots, n$ compute

$$E_k = x_k y_k + w \sum_{i \in \mathcal{M}(k)} s_i + q_k \tag{3.5}$$

where $w \in \mathbb{R}^+$ 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. In this step, a syndrome weighting parameter w is introduced. Typically, w and η are close to one, and are found through numerical optimization. The optimal values for w and η are code dependent, and are found to be weakly SNR dependent in some cases. Throughout this work, this algorithm and its variants is referred to as the NGDBF. Both single-bit and multi-bit versions are possible, and are indicated as Single-bit Noisy Gradient Descent Bit Flipping (S-NGDBF) and Multi-bit Noisy Gradient Descent Bit Flipping (M-NGDBF), respectively. In this work, mode-switching is never used in association with NGDBF; instead, threshold adaptation is employed as explained in the next subsection. The perturbation variance proportional to $N_0/2$ was chosen based on an intuition that the algorithm's random search region should cover the same distance as the original perturbation introduced by the channel noise. The noise-scale parameter η is introduced in order to fine-tune the optimal perturbation variance for each code. The effect of η on performance is studied empirically in Section 3.6.4. For some codes, good performance is obtained when using a single SNR-independent value for η. In other cases, η must be varied to get the best performance at different SNR values.

3.4 Threshold Adaptation

Methods of threshold adaptation were previously investigated in order to improve the convergence of multi-bit flipping algorithms. In this work, a local Adaptive Threshold GDBF (AT-GDBF) algorithm described by Ismail et al. [11] is considered, in which a separate threshold θ_k is associated with each symbol node. For $k = 1, 2, \ldots, n$, the threshold θ_k is adjusted in each iteration by adding these steps to the M-GDBF algorithm:

1. Initialize $\theta_k\,(t = 0) = \theta$ for all k, where $\theta \in \mathbb{R}^-$ is the global initial threshold parameter.

2. Modify step 3 in the NGDBF algorithm as follows: For all k, compute the inversion function E_k. If $E_k\,(t) \geq \theta_k\,(t)$, make the adjustment $\theta_k\,(t+1) = \theta_k\,(t)\,\lambda$, where λ is a global adaptation parameter for which $0 < \lambda \leq 1$. If $E_k\,(t) < \theta_k\,(t)$, flip the sign of the corresponding decision x_k.

In practice, the adaptation parameter λ must be very close to one. The case $\lambda = 1.0$ is equivalent to non-adaptive M-GDBF. According to the authors of [11], AT-GDBF obtains the same performance as M-GDBF with mode-switching, hence it enables fully parallel implementation with only local arithmetic operations. In the sequel, it is shown that threshold adaptation significantly improves performance in the M-NGDBF algorithm, at the cost of some additional complexity in the bit-flip operations.

3.5 Output Decision Smoothing

Convergence failures in the M-NGDBF algorithm may arise from excessive flipping among low-confidence symbols. This may occur as a consequence of the stochastic perturbation term. In this situation, the decoder may converge *in mean* to the correct codeword, but that does not guarantee that it will satisfy all parity checks at any specific time prior to the iteration limit T. More precisely, suppose the decoder is in an initially correct state,

i.e. initially all $x_k = \hat{c}_k$. When the inversion function is computed for some x_k, there is a non-zero probability of erroneous flipping due to the noise contribution:

$$p_{f,\,k} = \Pr\left(x_k y_k + w \sum_{i \in \mathcal{M}(k)} s_i + q_k < \theta \right). \tag{3.6}$$

Now suppose that p_f is the least among the $p_{f,\,k}$ values among all symbols. Then the probability P_F that at least one erroneous flip occurs in an iteration is bounded by

$$P_F \geq 1 - (1 - p_f)^n. \tag{3.7}$$

This probability approaches one as $n \to \infty$ for any $p_f > 0$. For a sufficiently large code, it would be unlikely to satisfy all checks in a small number of iterations, even if all decisions are initially correct. This problem may be compensated by introducing an up/down counter at the output of every x_k. The counter consists of a running sum X_k for each of the N output decisions. At the start of decoding, the counters are initialized at zero. During each decoding iteration, the counter is updated according to the rule

$$X_k\,(t+1) = X_k\,(t) + x_k\,(t) \tag{3.8}$$

If the stopping criterion is met (i.e. all parity checks are satisfied), then x_k is output directly. If the iteration limit T is reached without satisfying the stopping condition, then the smoothed decision is $\overline{x}_k = \text{sign}\,(X_k)$. In practice, the summation in (3.8) can be delayed until the very end of decoding. This saves activity in the up/down counter and hence reduces power consumption. Results in the sequel are obtained with summation only over the interval from $t = T - 64$ up to T. When using this procedure, the algorithm is referred to as the "smoothed" M-GDBF method, or in shortened form as SM-NGDBF.

3.6 Simulation Results

3.6.1 Bit Error Rate (BER) Performance

The proposed NGDBF algorithms were simulated on an AWGN channel with binary antipodal modulation using various regular LDPC codes selected from MacKay's online encyclopedia [34] (all selected codes are partially irregular in parity-check degree, but are still considered regular codes). For each code, the NGDBF decoding parameters, including θ, λ, η and w, were optimized one at a time, holding fixed values for all but one parameter. The free parameter was adjusted using a successive approximation procedure, repeating the BER simulation in each trial, and iteratively shrinking the search domain until the best value was found. This procedure was repeated for each parameter to obtain good-performing parameters. All NGDBF algorithms were evaluated for the rate 1/2 (3, 6) regular LDPC code identified as PEGReg504x1008 in MacKay's encyclopedia, which is commonly used as a benchmark in previous papers on WBF and GDBF algorithms. Because our primary attention is directed at SM-NGDBF, additional simulations were performed to verify this algorithm on the rate 1/2 regular (4, 8) code identified as 4000.2000.4.244, and on the rate 0.9356 (4, 62) code identified as 4376.282.4.9598. Unless stated otherwise, all simulations use double precision floating-point arithmetic, channel samples are saturated at $Y_{\max} = 2.5$, and the syndrome weighting is $w = 0.75$. In each simulation, comparison results are provided for the BP algorithm with 250 iterations, for the MS algorithm with 5, 10 and 100 iterations. Additional appropriate comparisons are described for each result presented in this section. The MS results presented here represent the strict MS algorithm, i.e. they do not reflect performance for offset-MS or normalized-MS. For the M-GDBF results, the mode-switching procedure was used to obtain the best performance in all cases.

The beneficial effect of added noise was first verified on the S-NGDBF algorithm for the PEGReg504x1008 code with maximum number of iterations limited to $T = 100$. The results are shown in Fig. 3.1, with comparative results for S-GDBF ($T = 100$), WBF ($T = 100$), MS and BP. The results show a gain approaching 1 dB for S-NGDBF compared to S-GDBF. This provides a basic empirical validation for the NGDBF concept.

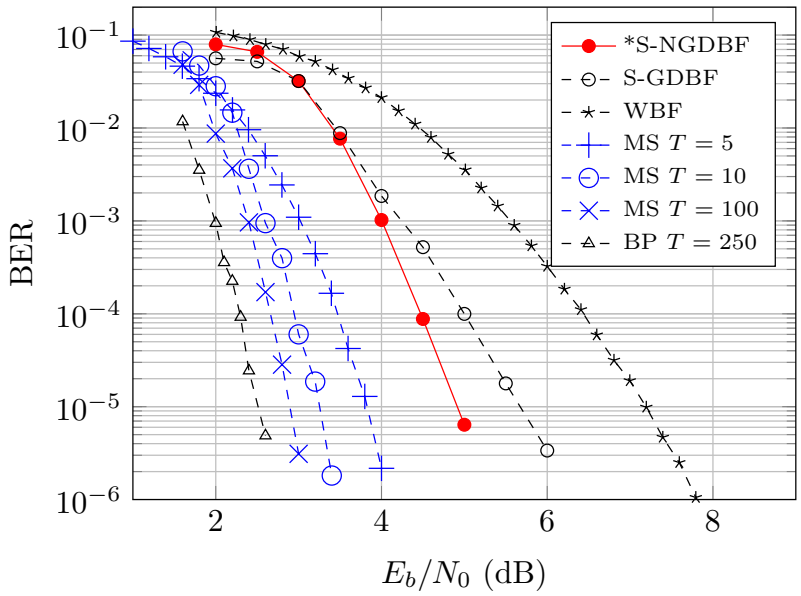

Fig. 3.1: BER versus E_b/N_0 curves for S-NGDBF with $T = 100$ and $\eta = 1.0$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed S-NGDBF algorithm is indicated by an asterisk (*).

Simulation results for the M-NGDBF algorithm are shown in Fig. 3.2. The results in this figure were obtained for the PEGReg504x1008 code with $T = 100$. The M-NGDBF results are shown for the non-adaptive case ($\lambda = 1.0$) and for the adaptive-threshold case with initial threshold $\theta = -0.9$ and $\eta = 0.95$, where η is the noise-scale parameter described in Section 3.3. For SNR < 3.5 dB, the best performance was obtained with an adaptation parameter of $\lambda = 0.99$. At higher SNR values, λ was decreased to 0.97 at 3.5 dB, 0.94 at 4.0 dB, and 0.9 at 4.25 dB and 4.5 dB. The performance of adaptive M-NGDBF is nearly identical to that of H-GDBF with escape process, which requires $T = 300$, and is also very close to MS with $T = 5$.

Results for the SM-NGDBF algorithm are shown in Fig. 3.3. For SM-NGDBF with $T = 100$, performance was equal to M-NGDBF (i.e. there was no gain from smoothing when $T = 100$), so these results are not shown. The most improved results were obtained with $T = 300$, the same number of iterations used for H-GDBF with escape process. SM-NGDBF is found to achieve about 0.3 dB of coding gain compared to H-GDBF, for the same value of T. Compared to M-NGDBF, SM-NGDBF is found to achieve about 0.3 dB of coding gain,

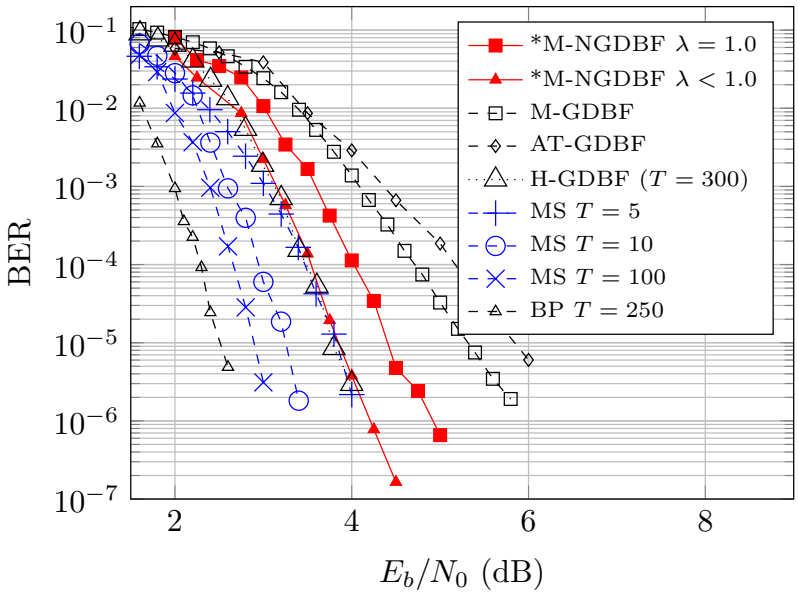

Fig. 3.2: BER versus E_b/N_0 curves for M-NGDBF with $T = 100$ using the rate 1/2 PEGReg504x1008 code simulated over an AWGN channel with binary antipodal modulation. The newly proposed M-NGDBF algorithms (adaptive and non-adaptive) are indicated by asterisks (*). Several other known algorithms are shown for comparison, including M-GDBF ($T = 100$), AT-GDBF ($T = 100$) and H-GDBF with escape process ($T = 300$).

at the cost of additional iterations ($T = 300$ for SM-NGDBF vs $T = 100$ for M-NGDBF). In order to confirm robust performance of the SM-NGDBF algorithm, it was simulated for two other LDPC codes, yielding the results shown in Figs. 3.4 and 3.5. These results confirm that SM-NGDBF achieves good performance on codes with higher variable-node degree (in the case of Fig. 3.4) and for codes with rates above 0.9 (in the case of Fig. 3.5). In both cases, SM-NGDBF remains competitive with MS decoding.

Among the previously reported bit-flip algorithms, the best performance was achieved by Wadayama et al.'s H-GDBF algorithm with escape process. Haga and Usami's IGDBF achieved nearly identical performance to H-GDBF. Both H-GDBF and IGDBF allow a maximum of 300 iterations to achieve the best performance. Our results indicate that the adaptive M-NGDBF algorithm equals the H-GDBF performance with a maximum of only 100 iterations. Furthermore, SM-NGDBF exceeds the H-GDBF performance when 300 iterations are allowed. These results may be interpreted in two ways. First, the adaptive M-NGDBF algorithm requires fewer iterations and is less complex than H-GDBF, but achieves

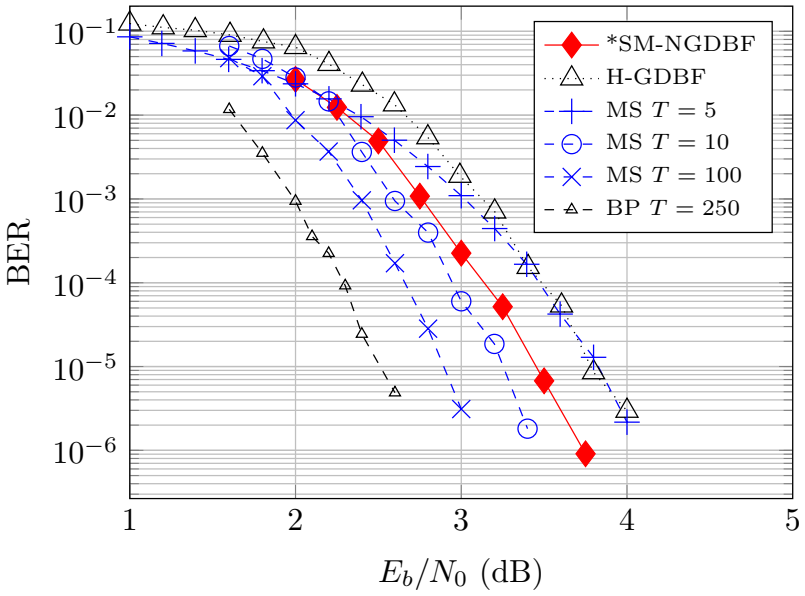

Fig. 3.3: BER versus E_b/N_0 curves for SM-NGDBF and H-GDBF with $T = 300$ using the rate 1/2 PEGReg504x1008 code over an AWGN channel with binary anitpodal modulation. The proposed algorithms are indicated by an asterisk (*).

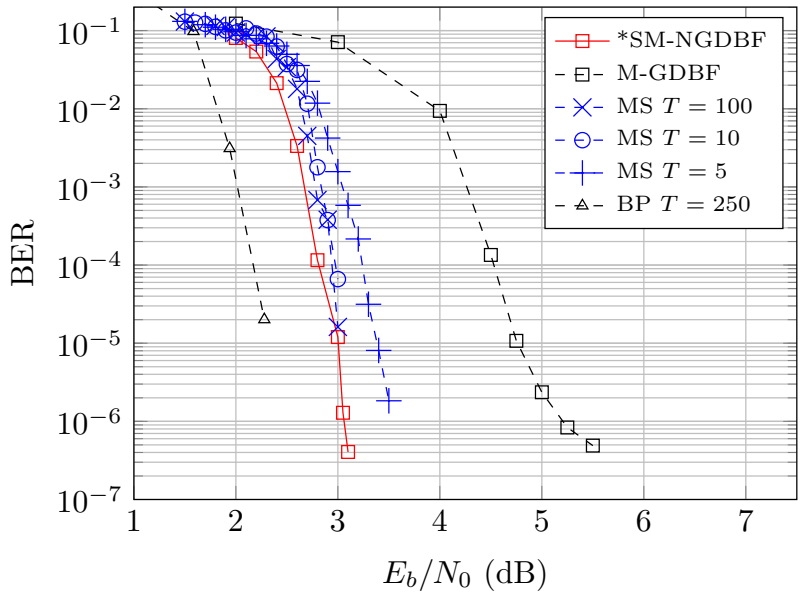

Fig. 3.4: BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 1/2 4000.2000.4.244 code over an AWGN channel with binary antipodal modulation. These results were obtained using $\theta = -0.9$, $\lambda = 0.99$, and η varied between 0.625 at low SNR (below 2.8) and 0.7 at higher SNR (above 2.8).

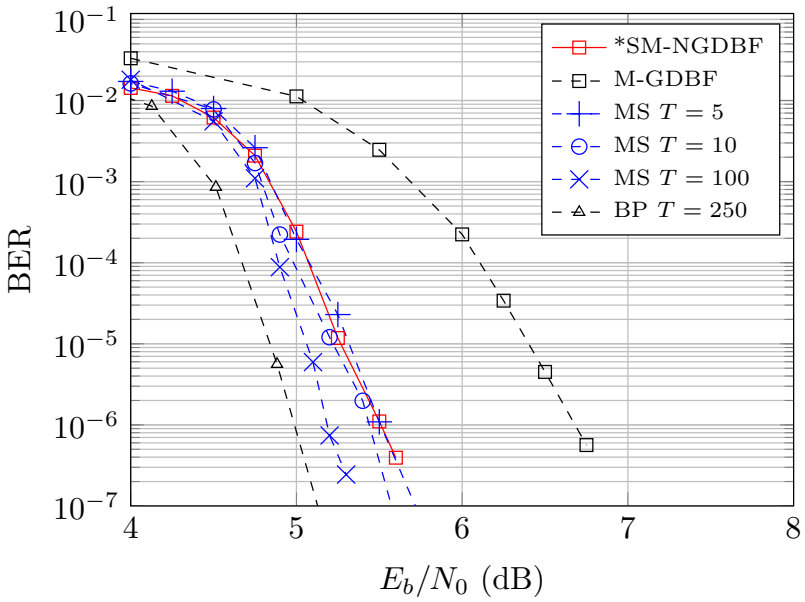

Fig. 3.5: BER versus E_b/N_0 curves for SM-NGDBF with $T = 300$ using the rate 0.9356 4376.282.4.9598 code over an AWGN channel with binary antipodal modulation. Several other algorithms are also shown for comparison. These results were obtained using $\theta = -0.7$, $\eta = 0.65$ and $\lambda = 0.993$. The dynamic range and syndrome weighting were also modified for this simulation, using $Y_{\max} = 2.0$ and $w = 0.1875$.

the same performance (Fig. 3.2) — hence it can be interpreted as a gain in speed and complexity over H-GDBF. Second, by using additional iterations with output smoothing, the speed improvement can be traded for additional coding gain (Fig. 3.3).

3.6.2 Average Iterations per Frame

Fig. 3.6 shows the average number of iterations per frame as a function of E_b/N_0, using the PEGReg504x1008 code. This plot considers results for M-NGDBF and S-GDBF with $T = 100$, and for the SM-NGDBF algorithm which has $T = 300$. The comparison curves show previously known GDBF algorithms with $T = 100$, and also H-GDBF with $T = 300$. For the H-GDBF algorithm, the full iteration profile was not disclosed, but it was stated to be 25.6 iterations at an SNR of 4 dB [10] (shown as a single point in Fig. 3.6). From the plot, it is seen that the S-NGDBF provides no benefit in iteration count compared to previous algorithms. The M-NGDBF algorithms are comparable to previous alternatives. Only the Early Stopping (ES) AT-GDBF algorithm converges faster than M-NGDBF, and

this advantage disappears at higher SNR. At high SNR, i.e. $E_b/N_0 \geq 5\,\text{dB}$, the average iteration count is nearly the same for the M-NGDBF, SM-NGDBF, and ES-AT-GDBF methods. At high SNR, the SM-NGDBF algorithm has the same average iterations as M-NGDBF. Although, SM-NGDBF requires $T = 300$ — three times higher than M-NGDBF — on average these algorithms require the same number of iterations when operating at the same SNR. As an alternative comparison, the average number of iterations needed to achieve a given BER performance is compared. Fig. 3.7 shows the average iterations per frame plotted against the measured BER for M-NGDBF and SM-NGDBF. When compared for the same BER, the number of iterations needed for SM-NGDBF is on average double the number of iterations required for M-NGDBF. This result shows that the performance gain of SM-NGDBF comes at the cost of increased average iterations. Since the average number of iterations for SM-NGDBF tends to be small, the smoothing operation is only used in a fraction of received frames. This is because the smoothing operation is only applied when the number of iterations exceeds $T - 64$. For the PEGReg504x1008 code, the smoothing operation was found to be required for only 6.1% of decoded frames when simulated at an SNR of 2.75 dB, 1.45% of frames at 3.0 dB, 0.51% of frames at 3.25 dB, and 0.16% of frames at 3.5 dB.

3.6.3 Sensitivity to Threshold Parameter

The optimal threshold values for the non-adaptive M-NGDBF algorithm (i.e. with $\lambda = 1.0$) were found empirically through a numerical search. Results from that search are shown in Fig. 3.8 for the PEGReg504x1008 code, in which the algorithm's BER is shown as a function of the threshold parameter. From this figure, it can be seen that the M-NGDBF algorithm is highly sensitive to the value of θ, which may prove problematic if the algorithm is implemented with fixed-point arithmetic at lower precision. The adaptive M-NGDBF algorithm was simulated in a similar way, and the results shown in Fig. 3.9 reveal much less sensitivity to θ. The reduced threshold sensitivity is expected because the local thresholds θ_k are iteratively adjusted during decoding. Since it will take some number of iterations for the θ_k to settle, the optimal initial threshold should be chosen as the value

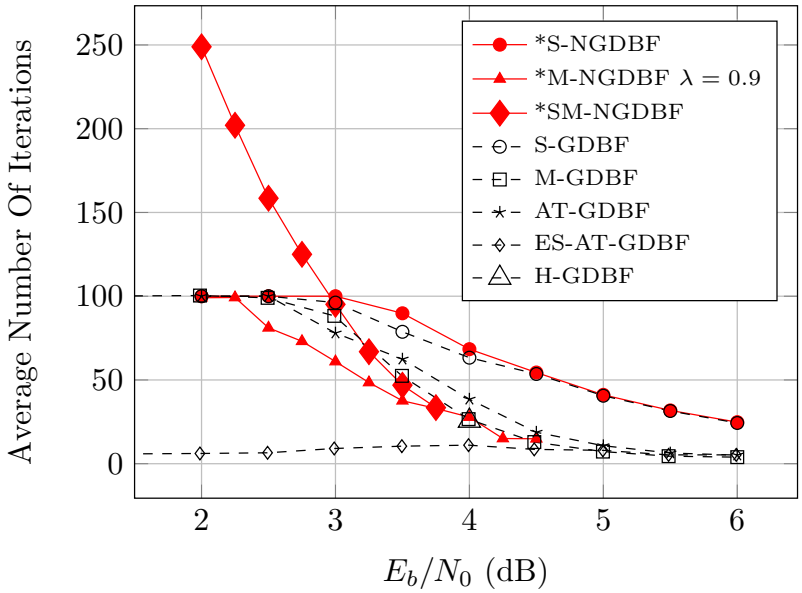

Fig. 3.6: Average number of iterations versus E_b/N_0 curves for existing GDBF and proposed NGDBF algorithms using PEGReg504x1008 code in an AWGN channel, maximum iterations limited to 100 except for SM-NGDBF and H-GDBF, which have $T = 300$. The newly proposed algorithms are indicated by an asterisk (*).

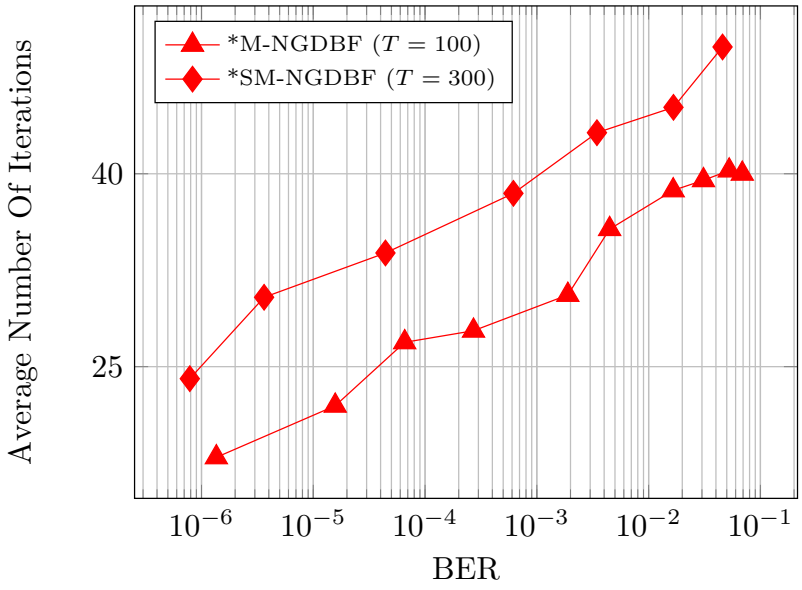

Fig. 3.7: Average number of iterations versus BER for the proposed M-NGDBF and SM-NGDBF algorithms.

that minimizes the average iterations per frame. Fig. 3.10 shows the average number of iterations per frame as a function of θ. The iteration count is seen to be only weakly a function of θ, with the minimum appearing at $\theta = -0.6$.

For adaptive M-NGDBF, the optimal value of λ is found through a similar empirical search. Results from that search are shown in Fig. 3.11 for the PEGReg504x1008 code. These results reveal a smooth relationship between BER and λ, allowing the optimal value of λ to be found reliably. The sensitivity revealed in Fig. 3.11 may prove to be difficult for implementations with quantized arithmetic; this problem is analyzed and resolved in Section 4.1.

3.6.4 Sensitivity to Perturbation Variance

The performance of NGDBF algorithm is sensitive to the precise variance of the noise perturbation terms. As with the θ and λ parameters, the optimal value of the noise-scale parameter η is found through an empirical search. This search may produce different values for different codes and at different SNR values. Example results are shown in Fig. 3.12 for the SM-NGDBF algorithm simulated on the PEGReg504x1008 code. These results show that the optimal η is typically somewhat less than one, and tends to increase toward one at higher SNR.

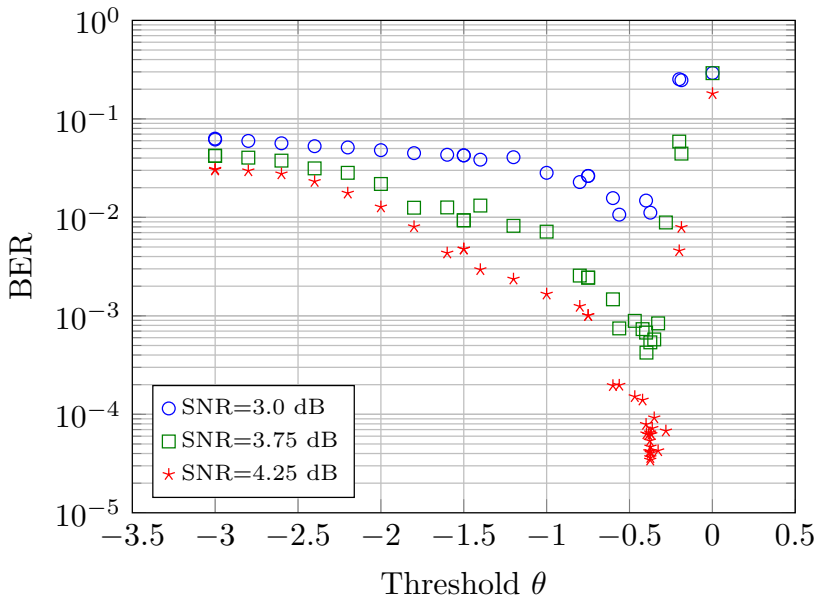

Fig. 3.8: Threshold sensitivity of the non-adaptive M-NGDBF algorithm with parameters $\lambda = 1.0$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code.

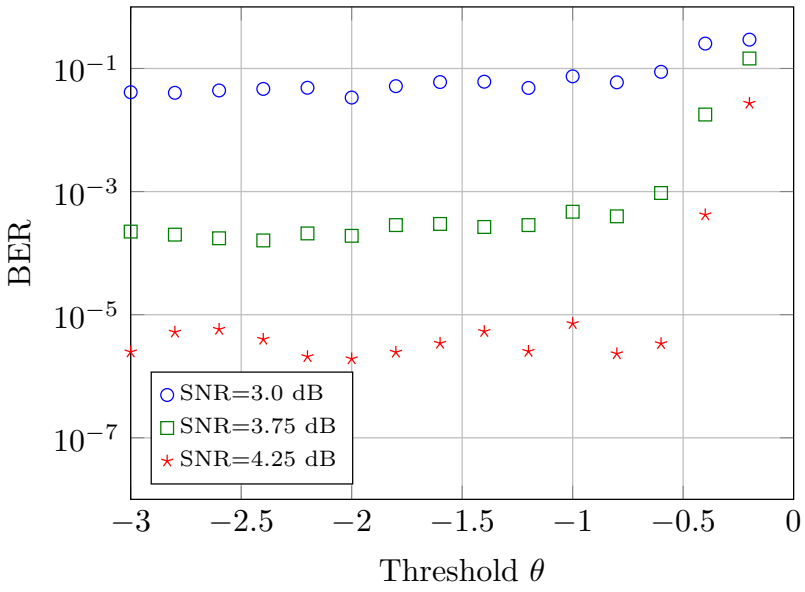

Fig. 3.9: Threshold sensitivity of adaptive M-NGDBF algorithm with parameters $\lambda = 0.9$, $T = 100$, $\eta = 1.0$ for the PEGReg504x1008 code.

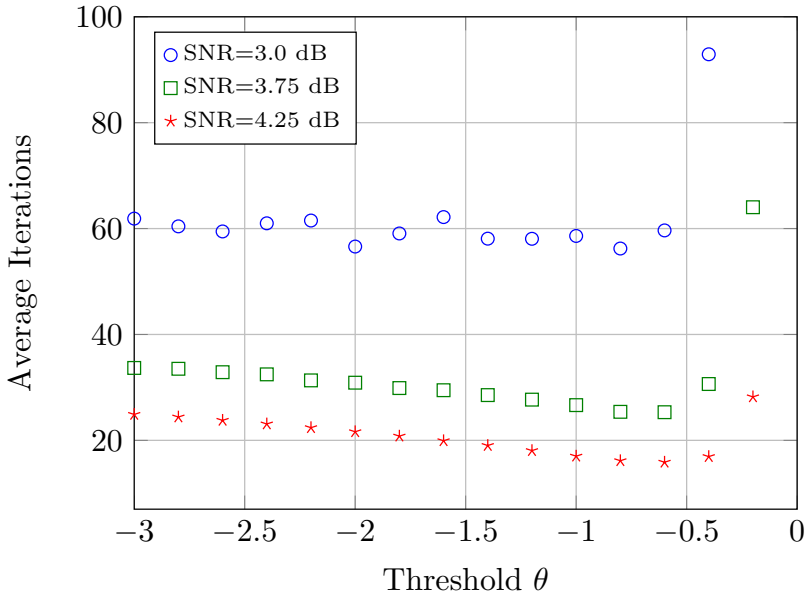

Fig. 3.10: Average iterations for adaptive M-NGDBF as a function of the initial threshold parameter θ for the PEGReg504x1008 code. The remaining parameters are $\lambda = 0.9$ and $T = 100$.

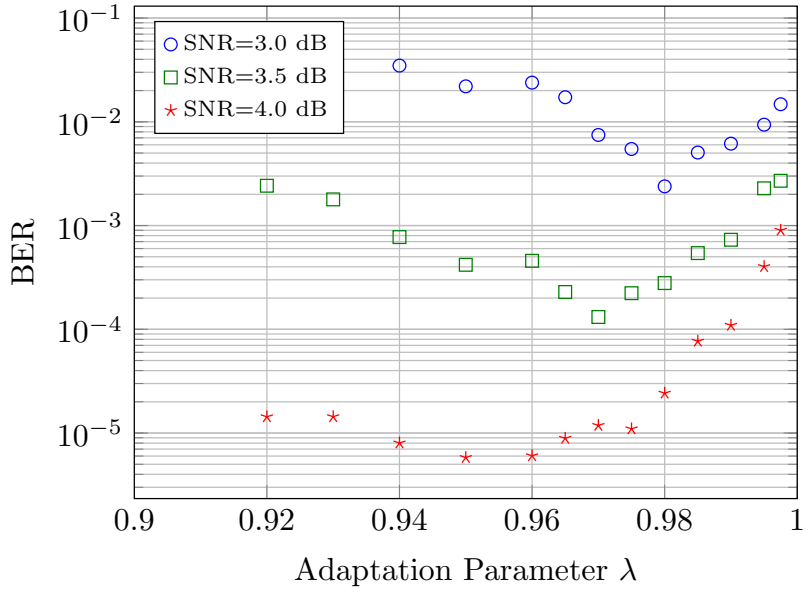

Fig. 3.11: Sensitivity of performance for M-NGDBF relative to the global adaptation parameter λ, with parameters $\theta = -0.9$, $T = 100$, $\eta = 0.96$, for the PEGReg504x1008 code.

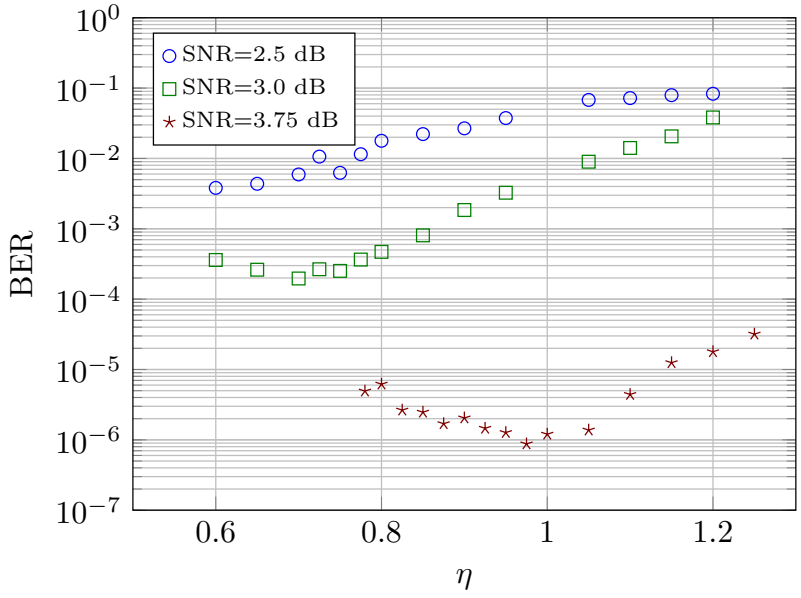

Fig. 3.12: Sensitivity of performance for SM-NGDBF relative to the noise scale parameter η.

Chapter 4

Architecture Considerations

This chapter considers practical concerns for implementing the adaptive SM-NGDBF algorithm. These concerns include limited-precision arithmetic and architectural simplifications.

4.1 Implementing Threshold Adaptation

In Section 3.6, threshold adaptation was shown to provide a significant performance improvement to the M-NGDBF and SM-NGDBF algorithms. When threshold adaptation is applied, as described in Section 3.4, each symbol node must independently implement threshold scaling (by parameter λ) during every iteration. If implemented with arbitrary precision, this would require implementing multiplication and division operations. When the algorithm is implemented with limited precision, however, only a small number of quantized threshold values are required. The threshold adaptation procedure can therefore be expressed as

$$\theta_k(t+1) = \begin{cases} \theta_k(t)\,\lambda & x_k \text{ not flipped} \\ \theta_k(t) & x_k \text{ flipped} \end{cases} \tag{4.1}$$

If quantized arithmetic is used with low precision, and if λ is close to one (as is commonly the case), then it is possible that $g(\theta_k\lambda) = \theta_k$, where $g(\cdot)$ is the quantization function defined by (4.5). This case represents a failure of threshold adaptation because the local threshold is never able to change.

To avoid adaptation failures in quantized arithmetic, the symbol u_k is introduced as a counter for non-flip events. The counter is initialized at the start of decoding as $u_k(t=0) =$

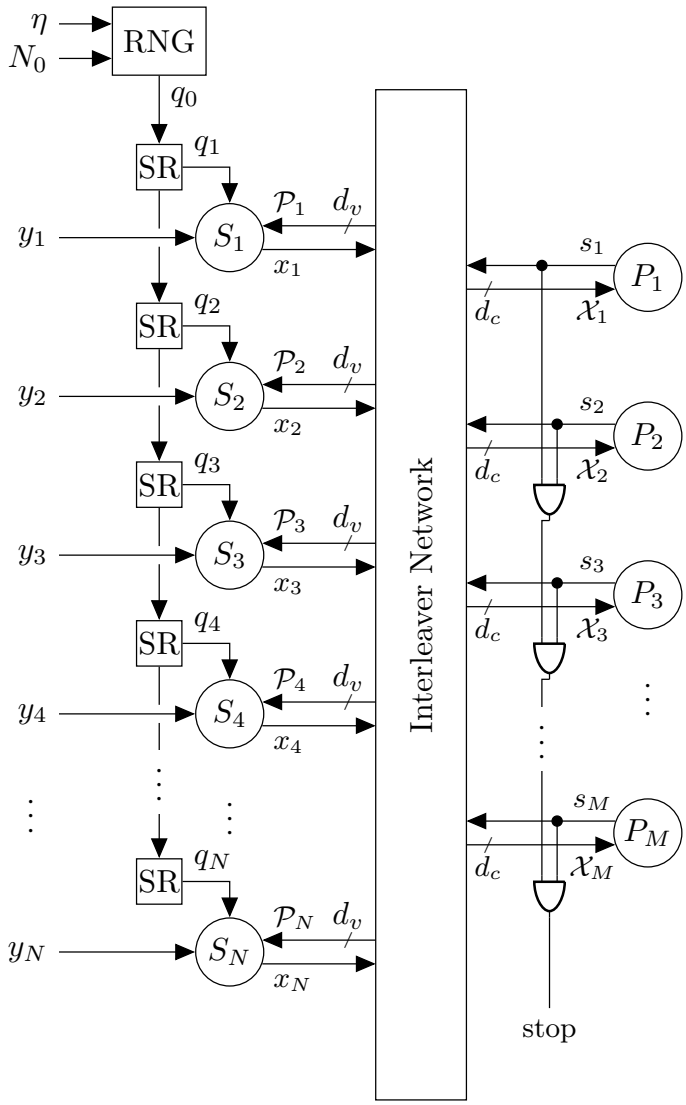

Fig. 4.1: Architecture of the NGDBF decoder. Gaussian-distributed noise samples are produced serially at the output of a Random Number Generator (RNG). The RNG requires inputs η and N_0, and the latter must be generated by a channel parameter estimator (not shown). A shift-register (SR) chain is used to distribute the random Gaussian samples that serve as the q_k perturbations. The symbol $\mathcal{P}_i$ indicates the set of syndrome messages that arrive at symbol node S_i, corresponding to the index set $\mathcal{N}(i)$. The symbol $\mathcal{X}_j$ is the set of messages that arrive at parity-check node P_j, corresponding to the index set $\mathcal{M}(j)$.

0, and the counter is incremented according to the rule

$$u_k(t+1) = u_k(t) + \frac{1+\delta_k(t)}{2} \tag{4.2}$$

where $\delta_k(t) = \text{sign}\left(\tilde{E}_k(t) - \tilde{\theta}_k(t)\right)$. Then the threshold at iteration t can be expressed as

$$\theta_k(u_k(t)) = \theta\lambda^{u_k(t)}, \tag{4.3}$$

so the quantized threshold value is then given by

$$\tilde{\theta}_k(u_k(t)) = g\left(\theta\lambda^{u_k(t)}\right). \tag{4.4}$$

In a finite-precision implementation, the quantized threshold $\tilde{\theta}_k(t)$ only changes for certain values of u_k. An *adaptation event* occurs for some $u_k(t) = \tilde{\tau}$ if $\tilde{\theta}(\tilde{\tau}) \neq \tilde{\theta}(\tilde{\tau}-1)$. Since u_k can only be changed by zero or one during any iteration, the threshold adaptation can be implemented by storing a pre-computed list of adaptation events $\left(\tilde{\theta}, \tilde{\tau}\right)$. Threshold adaptation can thus be implemented using a simple combinational logic circuit that detects when $u_k = \tau^{(i)}$ and outputs the corresponding $\tilde{\theta} = \tilde{\theta}^{(i)}$. Table 4.1 shows threshold adaptation events for an example design with $\theta = -0.9$, $\lambda = 0.99$, $T = 300$ and $Y_{\max} = 2.5$. The table shows the threshold values $\tilde{\theta}^{(i)}$ and the corresponding adaptation level $\tilde{\tau}^{(i)}$ at which the threshold value becomes active. Only two unique threshold values occur when $Q = 3$; three values occur when $Q = 4$; and six values occur when $Q = 5$. There is typically a small number of distinct threshold values, because the values only span a small portion of the quantization range.

4.2 Simplification of Noise Sample Generation

The NGDBF algorithms require generating a Gaussian distributed random number at each symbol node during each iteration. Gaussian random number generators add significant hardware complexity. In order to simplify the implementation, only a single Gaussian Random Number Generator (RNG) is used. The random samples are shifted from one

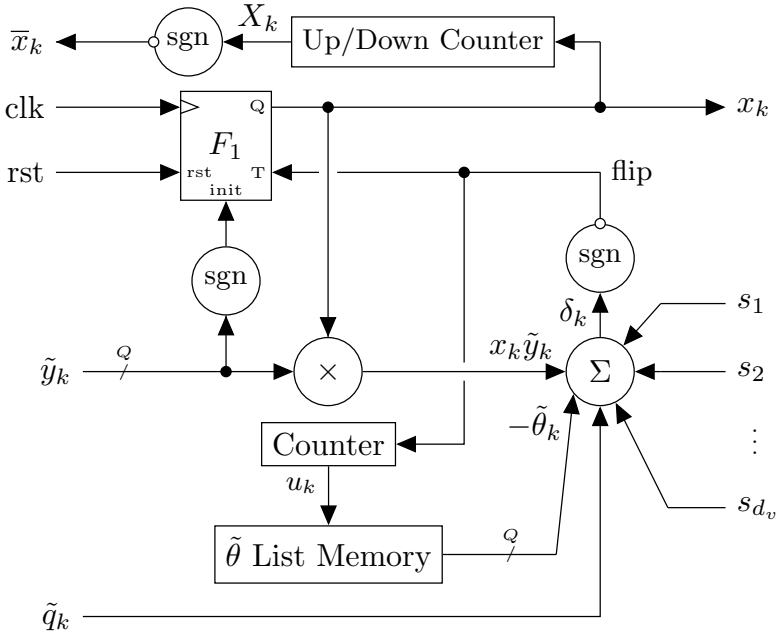

Fig. 4.2: Symbol node schematic. F_1 is a toggle flip-flop. The s_i messages are locally indexed. The sgn operator refers to sign-bit extraction. The multiplication $\otimes$ is binary, as it applies only to the sign bit of $\tilde{y}_k$.

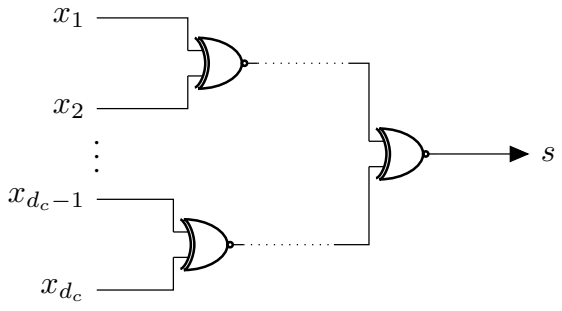

Fig. 4.3: Check node schematic showing a tree of XNOR operations over $d_c - 1$ input messages. The x_i messages are locally indexed.

Table 4.1: Threshold adaptation events for $\theta = -0.9$, $\lambda = 0.99$, $Y_{\max} = 2.5$.

	$Q=3$		$Q=4$		$Q=5$	
i	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$	$\tilde{\theta}^{(i)}$	$\tilde{\tau}^{(i)}$
0	-0.9375	0	-0.7812	0	-0.8594	0
1	-0.3125	37	-0.4688	37	-0.7031	15
2			-0.1562	106	-0.5469	37
3					-0.3906	65
4					-0.2344	106
5					-0.0781	175

symbol node to the next using a shift-register chain, as shown in Fig. 4.1. This method requires a power-up initialization so that all shift registers are pre-loaded with random samples. Simulations were performed using this method to obtain the results shown in Figs. 4.4 and 4.5, which come very close to the floating-point performance. As a further simplification, uniform noise samples may be used in place of Gaussian samples, but with an associated performance loss. When repeating the cases from Figs. 4.4 and 4.5 using uniformly distributed noise samples, a performance loss of 0.1–0.2 dB was observed. Because this performance loss is undesirable, in the remainder of this work only Gaussian distributed noise samples are considered.

4.3 Effects of Quantization on NGDBF Algorithm

In this section, the performance of the M-NGDBF algorithm when implemented with limited precision is considered. The algorithm was simulated with quantized arithmetic using Q bits by applying a uniform quantization with $N_Q = 2^Q$ levels in the range $[-Y_{\max}, Y_{\max}]$ (zero is excluded). The quantized channel sample $\tilde{y}_k$ is given by the quantization function $g(y)$:

$$g(y) = \operatorname{sign}(y) \left(\left\lfloor \frac{|y| N_Q}{2Y_{\max}} \right\rfloor + \frac{1}{2} \right) \left(\frac{2Y_{\max}}{N_Q} \right). \tag{4.5}$$

The quantization function is used to obtain quantized values. The vector of quantized channel samples is denoted by $\tilde{\mathbf{y}}$, and each quantized channel sample is $\tilde{y_k} = g(y_k)$. The same function is used to obtain the quantized inversion threshold, $\tilde{\theta} = g(\theta)$, the noise perturba-

tion, $\tilde{q}_k = g(q_k)$, and the syndrome weight parameter, $\tilde{w} = g(w)$. After quantization, the inversion function is

$$\tilde{E}_k(t) = x_k(t)\,\tilde{y}_k + \tilde{w} \sum_{i \in \mathcal{M}(k)} s_i + \tilde{q}_k(t)\,. \tag{4.6}$$

The adaptive M-NGDBF and SM-NGDBF algorithms were simulated using the quantized inversion function with the PEGReg504x1008 code. The BER results are shown in Fig. 4.4. The quantized simulations reported in this section also use quantized threshold adaptation and the noise sample reuse method described in Section 4.2. The results show that the algorithm is very close to unquantized performance when $Q = 3$, and the best BER performance is reached when $Q = 4$. There is a diminishing benefit to BER when $Q > 4$, however an additional effect is observed in the Frame Error Rate (FER) results shown in Fig. 4.5. Here, an "error flare" effect is seen for all cases for M-NGDBF, i.e. when output smoothing is not used. The flare improves when Q is increased. For SM-NGDBF, i.e. when output smoothing is used, the flare evidently does not occur at all, or occurs at a very low FER. These simulations were performed with parameter values $Y_{\max} = 1.7$–1.75, $\lambda = 0.98$–0.99, $\theta = -0.7$ and $w = 0.67$–0.75. Parameter values were adjusted within these ranges to optimize for BER performance.

4.4 Complexity Analysis

The foregoing considerations are combined to arrive at the top-level architecture shown in Fig. 4.1. In addition to the shown architecture, a channel SNR estimator is required in order to obtain σ. The symbol node implementation is shown in Fig. 4.2 and the check node implementation is shown in Fig. 4.3. The check node implementation is uncomplicated and standard, requiring only $d_c - 1$ binary XNOR operations per parity-check node. The symbol node requires one ordinary counter and one up/down counter, a $(\tilde{\theta}, \tilde{\tau})$ memory and a signed adder with three Q-bit inputs and d_v single-bit inputs. Four single-bit operations are also required, including a toggle flip-flop, a sign multiplier (equivalent to an XNOR operation) and two inverters. The most complex operation is the multi-input adder. In order to remove the weight parameter $\tilde{w}$ from the syndrome inputs, all $\tilde{y}_k$, $\tilde{q}_k$ and $\tilde{\theta}$ values

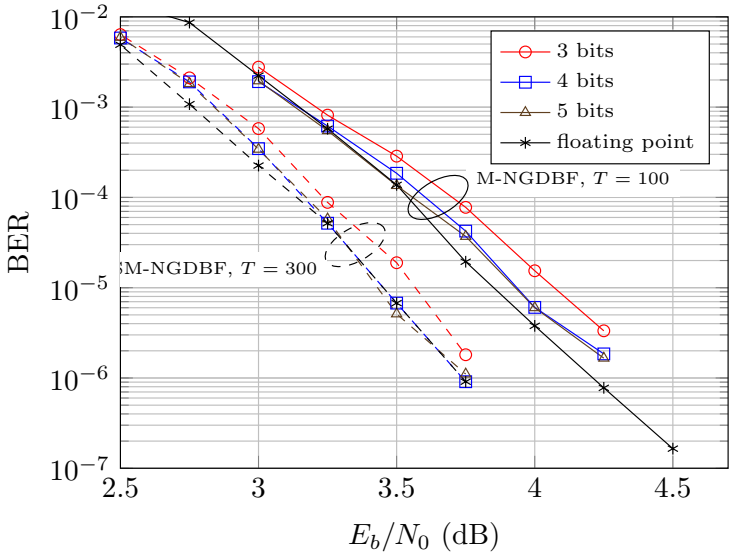

Fig. 4.4: BER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$.

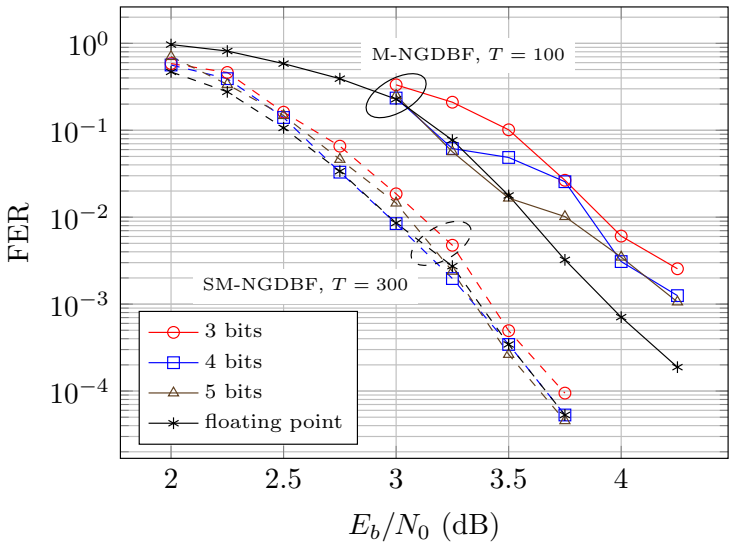

Fig. 4.5: FER versus E_b/N_0 curves for quantized implementations of the proposed M-NGDBF and SM-NGDBF algorithms, using the PEGReg504x1008 code on an AWGN channel with binary antipodal modulation. Solid curves indicate results for M-GDBF with $T = 100$, and dashed curves indicate results for SM-NGDBF with $T = 300$.

are pre-scaled by the factor $\tilde{w}^{-1}$ (this pre-scaling is not expressly indicated in Fig. 4.2). Then the scaled inversion function is

$$\tilde{E}_k\tilde{w}^{-1} = x_k\tilde{y}_k\tilde{w}^{-1} + \tilde{q}_k\tilde{w}^{-1} + \sum_{i\in M(k)} s_i \tag{4.7}$$

and the flip decision can be expressed as the sign of the difference $\delta_k = \tilde{E}_k\tilde{w}^{-1} - \tilde{\theta}_k\tilde{w}^{-1}$. This detail allows for the simplified adder implementation shown in Fig. 4.2. The effective complexity of this operation is that of two Q-bit binary adders and a d_v-bit adder. Based on this proposed architecture, it is possible to make some high-level complexity comparisons against other related decoding methods. When making comparisons at this level, it is not possible to make strong predictions about power consumption, throughput, gate count or energy efficiency, but it is possible to make some interesting observations about the algorithms' comparative features.

4.4.1 Comparison with Previous GDBF Algorithms

Previously reported GDBF algorithms do not depend on the channel SNR, so NGDBF introduces a fixed complexity cost (i.e. the cost is independent of the code's length) because it requires a channel parameter estimator. At minimum, all GDBF algorithms require $d_c - 1$ XNOR operations in each parity-check node. At the symbol nodes, they require addition over the d_v single-bit syndromes in each symbol node, which has gate complexity equivalent to a d_v-bit adder. A second Q-bit addition is needed to incorporate the channel information. In the S-GDBF algorithm, the minimum metric must be found, requiring $n - 1$ comparisons. Since a comparison can be implemented using a signed adder, the S-GDBF algorithm requires a total of $3n - 1$ additions and $m\,(d_c - 1)$ XNOR operations. In the M-GDBF algorithm, the global comparison is not required, but a comparison must still be made in each symbol node to implement the threshold operation, hence M-GDBF requires $3n$ additions. The SM-NGDBF algorithm requires $3n$ adders, and also requires an additional $2n$ counters (a counter requires fewer gates than an adder). A single RNG module and a channel parameter estimator are also required, but these are fixed overhead

that does not scale with n.

4.4.2 Comparison with MS Algorithm

The MS algorithm does not require channel SNR information. NGDBF again incurs a fixed complexity penalty due to channel parameter estimation. In a single iteration, the MS algorithm requires at least $2d_v$ additions for every symbol node. For every parity-check node, $2d_c$ comparisons and $2d_c - 1$ XNOR operations are needed. MS decoders typically allow an internal dynamic range that exceeds the channel quantization of Q bits, so the arithmetic is assumed to be quantized on $Q + D$ bits, where $D \geq 0$ is the number of extra bits to accommodate the larger dynamic range. The messages exchanged between symbol and parity-check nodes are also comprised of $Q + D$ bits. The gate requirements are clearly less for SM-NGDBF compared to MS. Based on the foregoing analysis, and using the $(3, 6)$ PEGReg504x1008 code as an example, SM-NGDBF requires 75% fewer additions and comparisons. In terms of message routing, all GDBF algorithms exchange a total of $n+m$ single-bit signals in each iteration, compared to $2nd_v\,(Q + D)$ for MS. In the example code, assuming channel quantization with $Q = 5$ and $D = 3$, this means the required signal routing is reduced by 78.27%. Based on these comparisons, it may be concluded that the GDBF algorithms (including SM-NGDBF) require substantially less circuit area than MS. This analysis is not sufficient to evaluate throughput or power efficiency, since those figures depend on a variety of circuit-level considerations such as critical path delay and average switching activity in combination with the average number of iterations per frame. Another aspect of complexity is the algorithms' decoding latency. On first inspection, it could be observed that the MS algorithm requires much fewer iterations than the SM-NGDBF algorithm. For example, only 4.1 iterations are needed on average for MS decoding (assuming $T = 10$ with stopping condition) on the PEGReg504x1008 code, operating at 3.5 dB. The SM-NGDBF algorithm, at the same SNR, requires an average of 47 iterations. While it appears that latency is much greater for SM-NGDBF, the latency per iteration in the two algorithms must also be accounted. In typical implementations, MS decoders utilize multiple clock cycles per iteration; for example, Zhang et al. used 12 clock cycles per iteration [32],

which is used here as a representative value. Due to SM-NGDBF's comparatively low gate and routing complexity, an SM-NGDBF decoder is expected to require only one clock cycle per iteration. The average latency of MS decoding could be estimated at 49 clock cycles, compared to 47 clock cycles for SM-NGDBF. It is therefore anticipated that an eventual implementation of SM-NGDBF could be comparable to previous MS implementations in terms of total latency.

4.4.3 Comparison with Stochastic Decoders

The M-NGDBF algorithm bears some similarity to stochastic LDPC decoders, as was mentioned in chapter 2. Stochastic LDPC decoders are known to provide performance within 0.5 dB of BP while exchanging single-bit messages with low-complexity logic processing. Stochastic decoders also require channel SNR estimation, so they share this fixed complexity cost with NGDBF. The most efficient stochastic decoding strategy is the Tracking Forecast Memory (TFM) described by Tehrani et al. [20]. The TFM-based decoder requires $2d_c - 1$ XOR operations at each parity-check node, nearly twice the number of XNOR operations needed by GDBF algorithms. At each symbol node, $2d_v$ Q-bit adders and d_v comparisons are used, for a total of $3nd_v$ equivalent additions. Some additional supporting logic is also required, including a Linear Feedback Shift Register (LFSR) to generate random bits, and a control circuit to regulate the inputs to the TFM adder. Tehrani et al. also described a reduced-complexity Majority TFM (MTFM) design which reduces the required additions to approximately $3n$, making it very close to SM-NGDBF [14]. The MTFM decoder exchanges a total of $2nd_v$ single-bit messages per iteration, compared to $n + m$ for GDBF algorithms. For the PEGReg504x1008 code, SM-NGDBF exchanges about 50% fewer single-bit messages than an MTFM stochastic decoder for the same code. In terms of total iterations, MTFM-based stochastic decoders require about 20–40 iterations at higher SNR, whereas SM-NGDBF requires a comparable number at 30–50 iterations for similar SNR values [14]. It may be concluded that these algorithms have very similar complexity, but SM-NGDBF should require less circuit area due to the reduced message signal routing, and because fewer XNOR operations are required.

4.5 Local Maximum Likelihood Interpretation

The GDBF and NGDBF algorithms are developed based on heuristic approaches for combinatorial optimization of the global ML objective function. In this section, a theoretical analysis to motivate the use of threshold adaptation and syndrome weighting is provided. To explain the beneficial effects of these heuristics, the Local Maximum Likelihood (LML) bit-flip decision at the symbol node level given the local information from the channel and adjacent partial syndrome values is considered. The LML analysis predicts a pattern by which the flip decisions should evolve as the decoder converges toward an error-free codeword. When using threshold adaptation and syndrome weighting heuristics with a GDBF algorithm, the evolution of flip decisions is brought into closer correspondence with the LML decisions. During the initial iterations, LML decisions are found to be mainly determined by the channel information. In later iterations, the LML decisions are more heavily influenced by the partial syndrome values. It is shown that this behavior is very close to that of GDBF under threshold adaptation. It is further proposed that GDBF can be improved by introducing a weight factor to the syndrome components, so that the local flip decisions evolve similarly to the LML decisions.

In this section, one minor change in notation is introduced. Since only scalar values are considered in this section, bold-faced letters are used to indicate random variables instead of vector quantities. The problem of gradient descent decoding on a local channel sample $\tilde{y}_k$ and a set of d_v adjacent syndromes s_i, $i \in \mathcal{M}(k)$ is considered. The channel sample is assumed to be quantized using the quantization procedure described in Section 4.3. For a binary-input AWGN channel, the probability masses for $\tilde{y}_k$ conditioned on the transmitted symbol $\hat{\mathbf{c}}_\mathbf{k}$ is obtained as follows:

$$\Pr\left(\tilde{y}_k \,|\, \hat{\mathbf{c}}_\mathbf{k} = -1\right) = F_{-1}\left(\tilde{y}_k^+\right) - F_{-1}\left(\tilde{y}_k^-\right), \tag{4.8}$$

$$\Pr\left(\tilde{y}_k \,|\, \hat{\mathbf{c}}_\mathbf{k} = +1\right) = F_{1}\left(\tilde{y}_k^+\right) - F_{1}\left(\tilde{y}_k^-\right), \tag{4.9}$$

where $\tilde{y}_k^+$ and $\tilde{y}_k^-$ are the upper and lower boundary points of the quantization range that contains $\tilde{y}_k$, and F_{-1} and F_1 are cumulative Gaussian distribution functions with variance

$\sigma^2 = N_0/2$ and means -1 and $+1$, respectively.

Initially, the decision $x_k\,(t=0)$ has error probability $p_e^{(0)} = P(x_k \neq \hat{\mathbf{c}}_\mathbf{k})$ given by $p_e = F_1\,(0)$. Recalling that the syndrome values are given by $s_i = \prod_{j\in\mathcal{N}(i)} x_j$, $s_i = x_k\nu_{ik}$ where ν_{ik} is the partial syndrome at parity-check i, excluding the influence of symbol node k. Finally, S_k is defined as the penalty term $S_k = \sum_{i\in\mathcal{M}(k)} s_i$. The penalty term can also be expressed as $S_k = \left(\sum_{i\in\mathcal{M}(k)} \nu_{ik}\right) x_k$.

From this, the partial syndrome error probabilities is directly obtained as $p_c = \Pr\left(\nu_{ik} \neq \hat{\mathbf{c}}_\mathbf{k}\right)$ by enumerating over combinations in which an odd number of symbol errors has occurred out of $d_c - 1$ independent, identically distributed neighbors:

$$p_c = \sum_{j=1}^{\left\lceil \frac{d_c-1}{2} \right\rceil} \binom{d_c-1}{2j-1} (1-p_e)^{d_c-2j}\, p_e^{2j-1}. \tag{4.10}$$

The probability $P(n_e)$ of having n_e errors among the partial syndrome values ν_{ik} is thus:

$$P(n_e) = \binom{d_v}{n_e} p_c^{n_e} (1-p_c)^{d_v-n_e}. \tag{4.11}$$

If $x_k = \hat{\mathbf{c}}_\mathbf{k}$, then n_e errors give a penalty $S_k = d_v - 2n_e$ (summation of $d_v - n_e$ syndrome $+1$ and n_e syndromes -1). Symmetrically, if $x_k = -\hat{\mathbf{c}}_\mathbf{k}$, then $S_k = 2n_e - d_v$. In other words, knowing the observation S_k, the following could be deduced:

$$P(S_k \mid x_k = \hat{\mathbf{c}}_\mathbf{k}) = P(n_e = (d_v - S_k)/2) \tag{4.12}$$

and

$$P(S_k \mid x_k = -\hat{\mathbf{c}}_\mathbf{k}) = P(n_e = (d_v + S_k)/2) \tag{4.13}$$

Then the LML decision is

$$\hat{x}_{k,\mathrm{LML}} = \arg\max_{x_k} \Pr\left(\tilde{y}_k \,|x_k\right) \Pr\left(S_k \,|x_k\right). \tag{4.14}$$

To relate the LML result to bit flipping algorithms, it can be expressed as an LML flip decision ϕ, defined by

$$\phi\left(x_k,\, \tilde{y_k},\, S_k\right) = \operatorname{sign}\log\left(\frac{\Pr\left(\tilde{y}_k\,|x_k\right)\Pr\left(S_k\,|x_k\right)}{\Pr\left(\tilde{y}_k\,|-x_k\right)\Pr\left(S_k\,|-x_k\right)}\right). \tag{4.15}$$

If $\phi = -1$, then the optimal decision is to flip x_k.

In order to visualize the LML behavior on a quantized channel, the decisions are arranged in a *flip matrix* Φ that expresses all possible states; the rows of Φ correspond to the possible channel sample values, and the columns correspond to the possible values of S_k. There are $d_v + 1$ possible values of S_k: $-d_v, -d_v + 2, \ldots, d_v - 2, d_v$. These values are indexed in ascending order as $S_k^{(j)}$, $j = 1,\, 2,\, \ldots, d_v + 1$. The possible $\tilde{y}_k$ values are similarly indexed in ascending order as $\tilde{y}_k^{(i)}$, $i = 1,\, 2,\, \ldots, N_Q$. Then Φ is an $N_Q \times (d_v + 1)$ matrix with entries $\phi_{i,\,j} = \phi\left(x_k, \tilde{y}_k^{(i)},\, S_k^{(j)}\right)$. For a given locally received $\tilde{y}_k$ and S_k, if the corresponding $\phi_{ij} = -1$, then the corresponding decision x_k should be flipped.

The LML flip decision depends on the partial syndrome error probabilities, which change in successive iterations. In order to understand how the LML decision evolves across iterations, it is supposed that the bit error probability is a function of the iteration number t, and that $p_e\,(t)$ is decreasing with successive iterations. As the error probability decreases, the flip matrix is found to evolve from an initial pattern in which decisions are heavily dependent on $\tilde{y}_k$, with increasing dependence on S_k in later iterations as $p_e(t)$ decreases toward zero. An example of this evolution is shown in Fig. 4.6, for the case $x_k = 1$ with parameters $Y_{\max} = 1.5$, $\sigma = 0.668$, $Q = 4$ and $d_v = 3$. For a $(3, 6)$ LDPC code, this corresponds to $E_b/N_0 = 3.5\,\mathrm{dB}$.

With threshold adaptation, the GDBF algorithm's behavior is similar to the LML flip matrix. Initially, the threshold θ is set to a significantly negative value, say $\theta = -1.0$. For a given set of parameters, a matrix E of values for the inversion function is obtained, with members $E_{i,\,j} = \tilde{y}_k^{(i)} + S_k^{(j)}$. By applying the threshold θ to all the elements of E, the flip matrix is obtained for the GDBF algorithm. As the threshold is adapted toward zero, the flip matrix evolves to place increased weight on the syndrome information, similar to

the LML flip matrix evolution. The GDBF flip matrices do not correspond perfectly to the LML predictions. To bring closer agreement, a weight factor is introduced, giving a modified weighted inversion function

$$\tilde{E}_k = x_k \tilde{y}_k + w \sum_{i \in M(k)} s_i + \tilde{q}_k, \tag{4.16}$$

The best value for w is found empirically and may be code dependent. Based on the parameters $\sigma = 0.668$, $Y_{\max} = 1.5$, $w = 0.75$ and $d_v = 3$, the flip matrix evolution corresponding to $\tilde{E}_k$ is shown in Fig. 4.7.

The relationship between LML and the GDBF heuristics is not an exact correspondence. The LML analysis predicts the desirable behavior of the flip matrix over time during a successful decoding event. When GDBF is augmented by introducing the threshold adaptation and syndrome weighting heuristics, its behavior is brought into approximate correspondence with the LML prediction. This provides a new theoretical motivation for using these heuristic methods, which has not been addressed in the previous literature.

$$\underset{p_e(0)=0.0672}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}} \rightarrow \underset{p_e=0.5p_e(0)}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}} \rightarrow \underset{p_e=0.01p_e(0)}{\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}}$$

Fig. 4.6: Example of evolution of the LML flip matrix Φ for a (3,6) LDPC code, with $Q = 4$ and an $E_b/N_0 = 3.50\,\text{dB}$. The initial value of P_e is 0.0672. Since Φ is symmetric with $\Phi(i,\, j) = \Phi(N_Q + 1 - i,\, d_v + 2 - j)$, only the top $N_Q/2$ rows are shown.

$$\underset{\theta=-0.9}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \end{bmatrix}} \rightarrow \underset{\theta=-0.3}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}} \rightarrow \underset{\theta=0.0}{\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}}$$

Fig. 4.7: Evolution of the flip matrix for the weighted GDBF algorithm with threshold adaptation. Use of threshold adaptation and syndrome weighting achieves qualitative agreement with the LML flip decisions in Fig. 4.6.

Chapter 5

ASIC Implementation of Noisy Gradient Descent Bit Flip Decoder For 10GBASE-T Ethernet Standard

This chapter discusses the detailed ASIC implementation of the NGDBF decoder for IEEE 802.3*an* 10GBASE-T Ethernet Standard. The code deployed in this standard is a Reed-Solomon (RS) LDPC code and is also a common benchmark code for highly parallel implementations [35]. The parity check matrix corresponding to this code has 2048 columns and 384 rows. The code rate is 0.841. The degree distribution is $(6, 32)$. The decoder employs a noise addition method in a simplistic manner, thereby leading to a very efficient design. Of all the heuristics discussed in chapter 3.3, only the syndrome scaling heuristic is used to obtain an error performance that is comparable to other state of the art decoders reported on the IEEE 802.3*an* 10GBASE-T Standard. The chosen syndrome weight is 0.166 (1/6). The inversion threshold is chosen to be -0.55. The magnitude of the channel samples was limited to 2.95. All the above mentioned parameters are estimated empirically from simulations for the best performance.

5.1 Error Rate Performance

Fig. 5.1 shows the performance of the NGDBF algorithm in comparison with other algorithms. The other algorithms shown in the plot are: stochastic MTFM decoding algorithm, IDB algorithm, GDBF algorithm and the Offset Min-Sum algorithm. From the plot, it could be observed that the NGDBF algorithm performs better than the IDB algorithm. The IDB algorithm reaches an error rate of 10^{-7} at an E_b/N_0 of 4.5 dB, while the NGDBF decoder is able to reach the same error rate at an E_b/N_0 of 4.45 dB similar to the stochastic decoder. In the case of the IDB decoder, the failed frames are re-decoded six more times, with maximum number of iterations limited to 45 for each phase. The M-GDBF algorithm

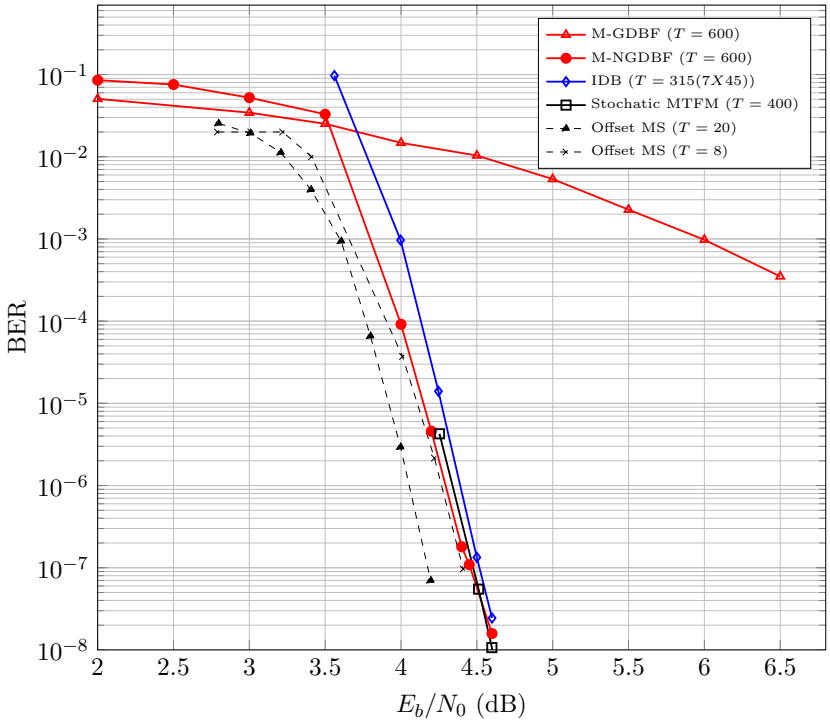

Fig. 5.1: BER for NGDBF compared to a benchmark offset MS decoder for the IEEE 802.3 standard LDPC code with maximum iterations limited to T.

performs very poorly on this code compared to other algorithms as shown in the plot. M-GDBF is struck in trapping sets and is unable to escape the effects of those trapping sets that are dominant in the waterfall region. Fig. 5.2 shows the dominant trapping set that occurs in the case of M-GDBF decoding. From the figure, it is very clear to see as to how this trapping set results in a decoding failure. In Fig. 5.2, each of the four symbol nodes is connected to three satisfied and three unsatisfied parity checks. This means that three of the incoming messages are opposing the current value of the hard decision bit and three of the incoming messages are reinforcing the current value of the hard decision bit. The syndrome sum corresponds to zero in this case. This leads the decoder to a locked state, resulting in a decoding failure.

Fig. 5.3 shows the average number of iterations taken by the NGDBF decoder to converge with variation in E_b/N_0. From the plot, it could be seen that the IDB decoder has faster convergence compared to the NGDBF decoder. NGDBF converges faster compared

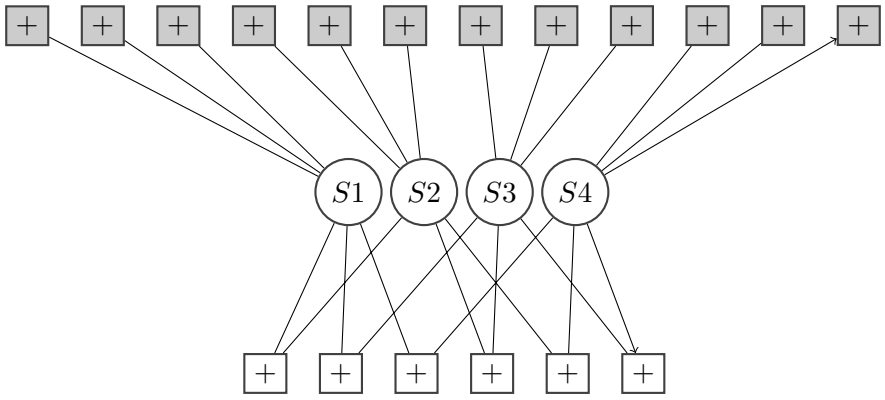

Fig. 5.2: Dominant $(4, 12)$ trapping set in M-GDBF algorithm. The parity checks indicated in black are unsatisfied parity checks. The parity checks in the bottom are satisfied parity checks.

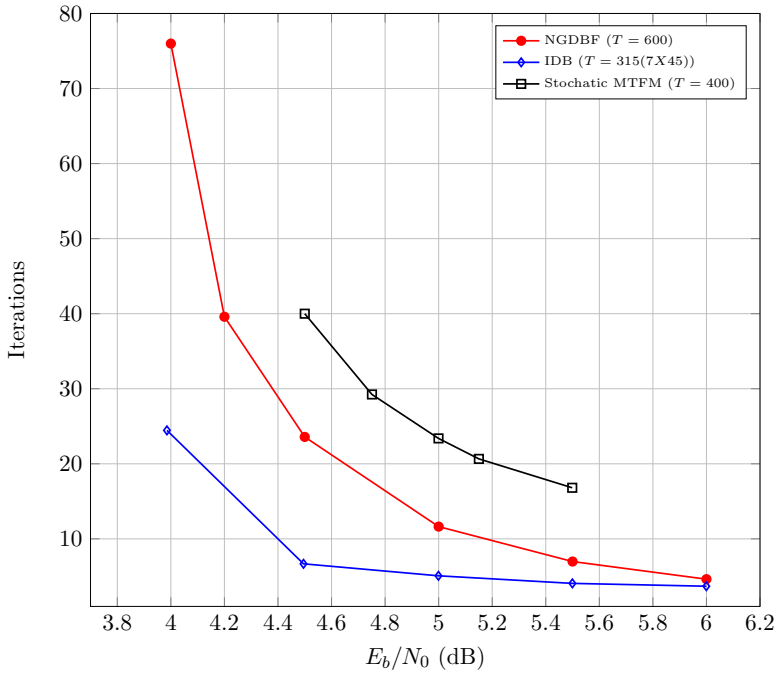

Fig. 5.3: Average number of iterations for NGDBF algorithm for IEEE 802.3 standard LDPC code with maximum iterations limited to T.

to the stochastic MTFM decoder. With increase in E_b/N_0, the gap in the average number of iterations between NGDBF and IDB reduces.

5.2 Architecture of NGDBF Decoder

Fig. 5.4 shows the top level architecture of the NGDBF decoder. The decoder has a fully parallel architecture and adopts a flooding schedule. The decoder consists of five main blocks: Noise Update Unit (NUU), Symbol Node Unit (SNU), Check Node Unit (CNU), Early Termination Unit (ETU) and the interleaver network. The check nodes are updated first and the symbol nodes are updated later during a decoding iteration. A decoding iteration takes a clock cycle. The fully parallel NGDBF decoder has 2048 symbol node processors and 384 check node processors. The decoder has 4 inputs, 2 outputs and 4 control signals. Table 5.1 shows the input, output and the control signals related to the decoder and their widths. The decoder is operated in two phases: The first phase $\phi 1$ is termed as the start-up phase. In this phase, noise samples are obtained, processed and are stored in a set of registers. The operation in this phase only involves the NUU. During the second phase $\phi 2$, decoding operation is initiated and the decoder starts to decode. ETU detects convergence, signals the need for the current frame to be removed and a new frame to be loaded in. Fig. 5.5 shows the waveform diagram corresponding to the proper operation of the decoder. The decoder employs an active low reset signal. During the first 2648 cycles, standard Gaussian samples of width Q bits are loaded in from the input $Noisein$ at the rate of one sample every clock cycle. NUU processes each sample and stores them in a $Q-1$ bit register. After the end of 2648 clock cycles, the first frame is loaded and decoding starts. Every iteration takes a clock cycle. Decoding throughput of an LDPC decoder can be calculated as follows:

$$Throughput = \frac{Frequency(f) \times Block\ Length(N)}{Cycles\ per\ Iter(s) \times Num\ of\ Iter(t)} \tag{5.1}$$

where f is the maximum speed of the decoder and is determined by the latency of one iterative check and symbol node processing. Since the decoder completes one iteration per cycle, $s = 1$. Table 5.2 shows the control logic for synchronizing and controlling all the phases of the decoder operation. At $t = k$, the $ChkOut$ is low and the decoder converges.

5.2.1 NUU Design

From section 4.3, the following equation for the symbol node update of a symbol node k is obtained after quantization:

$$\tilde{E}_k(t) = x_k(t)\,\tilde{y}_k + \tilde{w} \sum_{i \in \mathcal{M}(k)} s_i + \tilde{q}_k(t)\,. \tag{5.2}$$

The symbol node evaluates the right-hand side of the above equation and then flips the decision bit (x_k), if $\tilde{E}_k$ is less than inversion threshold $\tilde{\theta}$. This is done by calculating the sign of the below equation:

$$\tilde{E}_k(t) - \tilde{\theta} = x_k(t)\,\tilde{y}_k + \tilde{w} \sum_{i \in \mathcal{M}(k)} s_i + \tilde{q}_k(t) - \tilde{\theta}. \tag{5.3}$$

The right-hand side of equation (5.3) contains four terms. Only the first two terms involve quantities that will be updated during a decoding iteration. The first term involves current hard decision and the second term involves summation of the syndromes that are obtained from the neighboring check nodes. The third term and the fourth term involve operations that are independent of either symbol node or check node updates and could be done prior to the start of decoding to reduce the decoding latency. In this design, these operations are done in the start-up phase by the NUU. Fig. 5.6 shows the architecture of the NUU. As discussed in section 4.2, noise generation can be simplified by generating all the samples during the start and then reusing them by just shifting the noise samples from one register to another during a decoding iteration. However, the architecture described in section 4.2 still requires one Gaussian random number generator. Gaussian random number generators are very complex and incur large complexity both in space and in time [36] [37] [38].

As an alternative to the architecture described in section 4.2, a noise generation method is proposed in this section that is more efficient in comparison and does not require an on-chip Gaussian random number generator. NUU consists of a seven bit sign-magnitude multiplier, a seven bit sign-magnitude adder and a set of 2648 $Q-1$ bit shift registers. During

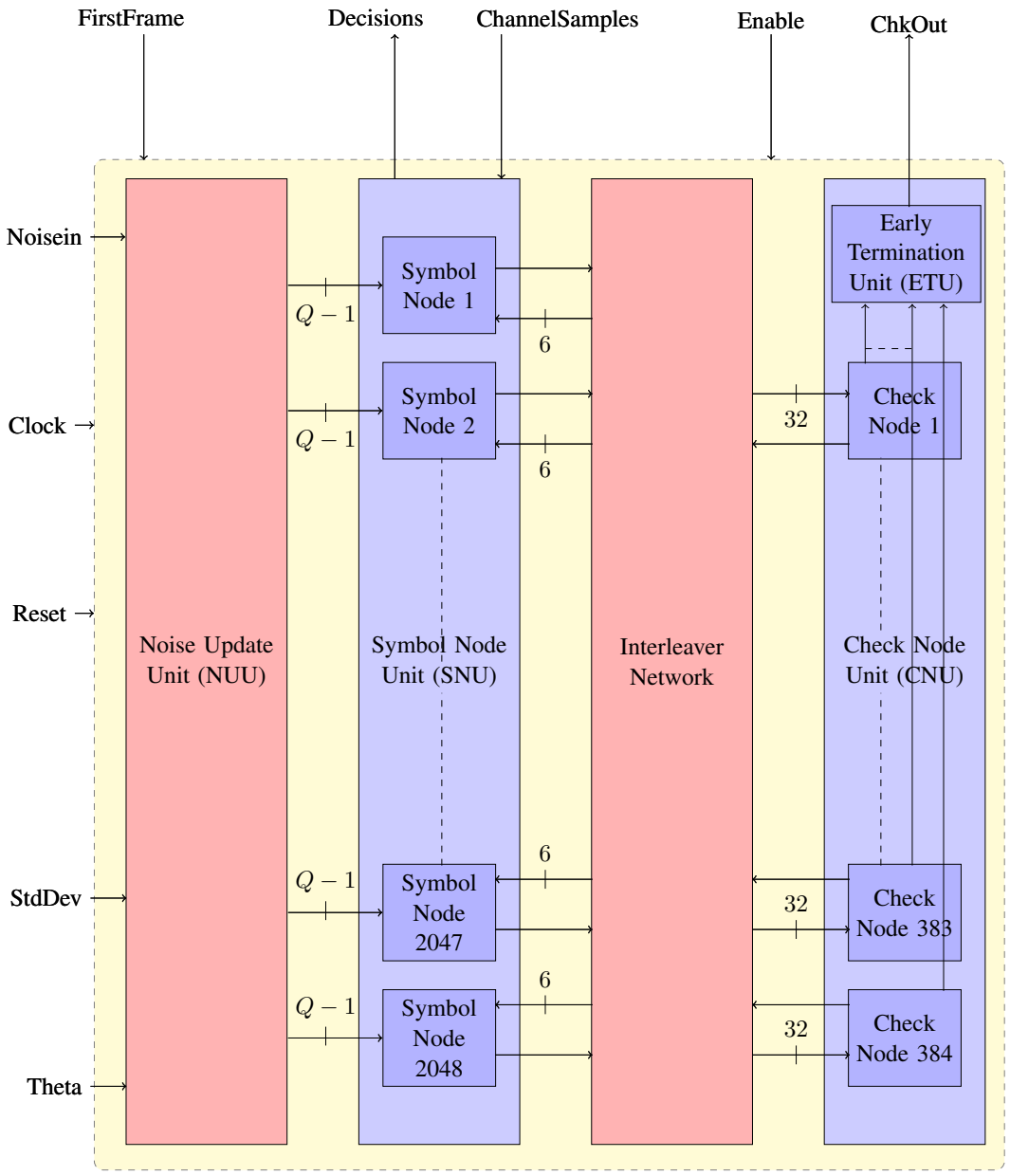

Fig. 5.4: Top level decoder architecture showing all the blocks. Input, output and all the control signals are also clearly shown. $d_v(6)$ messages arrive at the symbol node from the interleaver and $d_c(32)$ messages arrive at a check node from the interleaver during a decoding iteration. Symbol node update requires $Q-1$ bits of noise from the NUU every iteration.

the start-up phase, NUU receives a standard Gaussian sample (mean (μ)=0, standard deviation (σ)=1) from the input $Noisein$. All the received samples are in sign-magnitude format. The total length of the sample is $Q = 7$ bits, with one bit representing the sign, two bits representing the integer part and four bits representing the fractional part. The sample

Table 5.1: Input, output and control signals of the decoder and their widths. Q is number of quantized bits used for representing real-valued signals in the decoder.

Name	**Type**	**Width**
FirstFrame	control	1
Decisions	output	2048
ChannelSamples	input	$2048 * Q$
Enable	control	1
ChkOut	output	1
Noisein	input	Q
Clock	control	1
Reset	control	1
stdDev	input	Q
Theta	input	Q

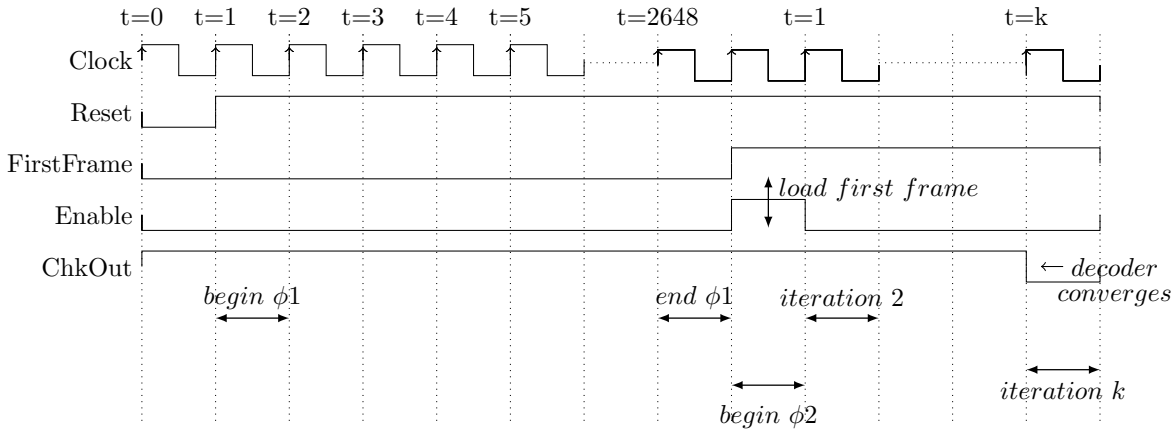

Fig. 5.5: Timing diagram of the decoder. The start of the two operational phases is clearly shown. At $t = 2649$, the decoding operation begins and the first frame is loaded into the decoder.

Table 5.2: Control table.

FirstFrame	**Enable**	**Operation**
0	0	$\phi 1$ begins
1	1	$\phi 2$ begins and new frame loaded
1	0	Decoding iteration begins

is then multiplied with standard deviation of the channel noise received from input *StdDev* that is Q bits (7 bits) wide. The multiplier output is then added with the inversion threshold

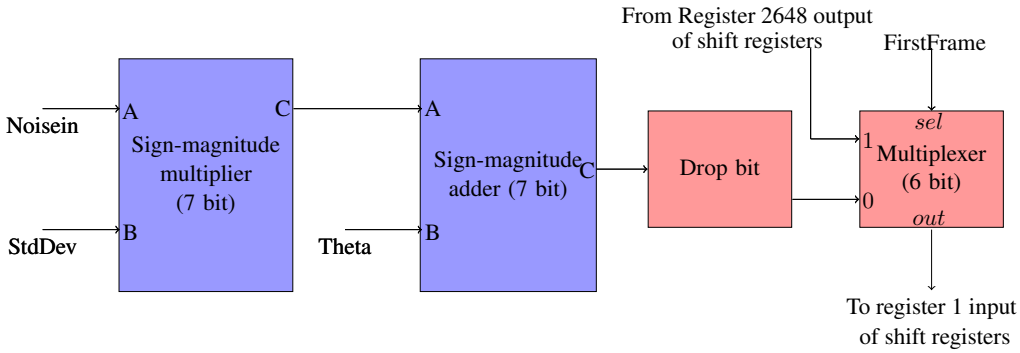

Fig. 5.6: Architecture of NUU.

Theta and a 7 bit sum is obtained. The Most Significant Bit (MSB) of the integer part of the sum is dropped and the remaining six bits are written to the first 6 bit register Reg 1 of a series of shift registers, as shown in Fig. 5.7. The sixth bit in the sum corresponds to MSB of the integer part of the sum. This bit was mostly found to be zero from our simulations and is dropped without significantly affecting the error performance. Dropping a bit results in having 6 bit registers instead of 7 bit registers, thereby leading to a considerable reduction in area and power dissipation. During the second cycle, same operation is repeated and the output is written to the first register while the previous contents of the first register are shifted to the second register. This operation proceeds until all the 2648 registers are loaded with noise samples that have the same statistics as the channel noise. During the phase $\phi 2$, all the samples in the registers are barrel shifted from top to bottom. As shown in Fig. 5.7, outputs of registers 1-2048 are connected to the symbol nodes 1-2048 respectively. Barrel shifting will ensure that a new noise sample is introduced into the symbol node updates of the decoder during a decoding iteration. The switch between the two phases is realized by a two input multiplexer that is controlled by the FirstFrame signal.

Fig. 5.8 shows the architecture of the seven bit sign-magnitude multiplier. The signs of the two inputs A and B are passed on as inputs to a XOR gate to produce the output sign C. The magnitude values of two inputs are passed on as inputs to a six bit Carry Save Array Multiplier (CSAM) and a twelve bit product is obtained. The bits 0, 1, 2, 3, 10 and 11 are discarded to obtain the final magnitude. The procedure for performing a sign

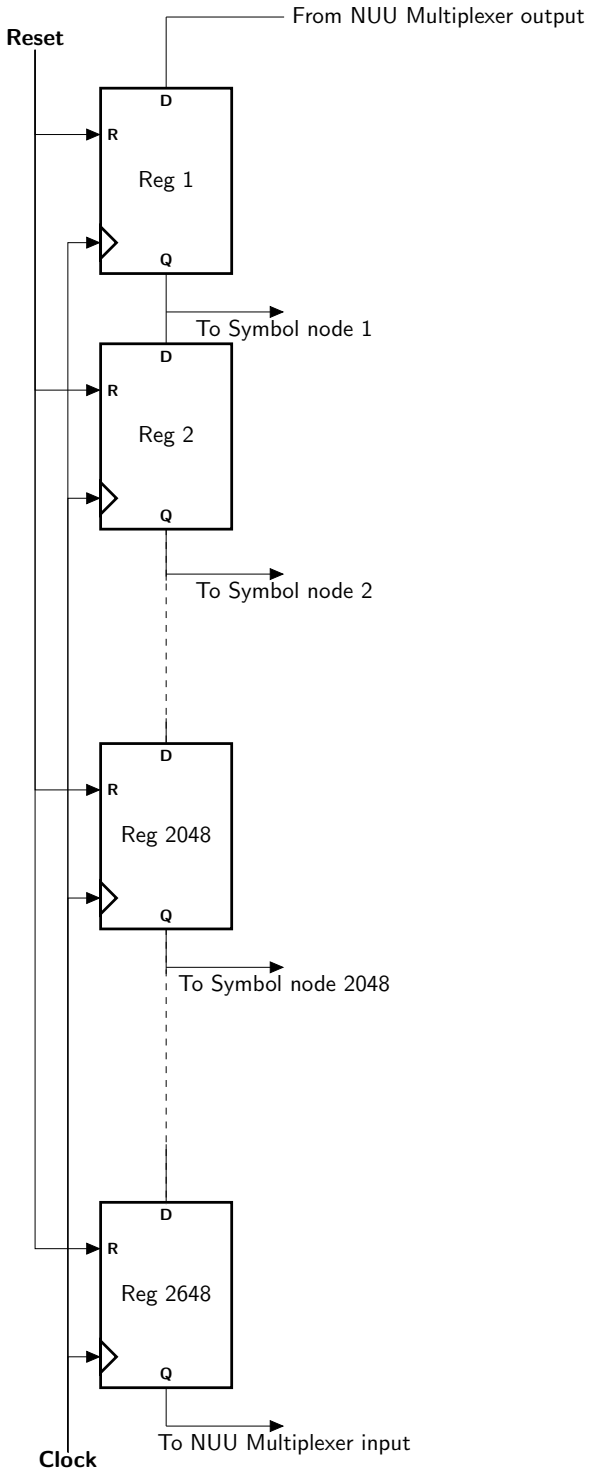

Fig. 5.7: Shift registers containing noise samples.

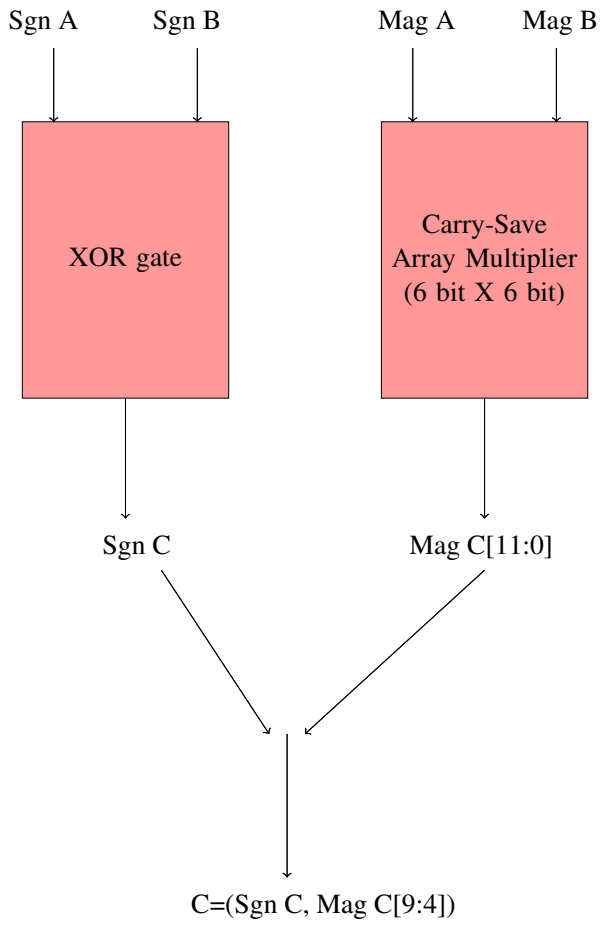

Fig. 5.8: Seven bit sign-magnitude multiplier. Mag corresponds to magnitude and Sgn corresponds to the sign.

Table 5.3: Table for selector operation.

a	**b**	**c0**	**c1**	**outa**	**outb**
Mag A	Mag B	0	0	Mag A	Mag B
Mag A	Mag B	0	1	$\overline{MagA}$	Mag B
Mag A	Mag B	1	0	Mag A	Mag B
Mag A	Mag B	1	1	Mag A	$\overline{MagB}$

magnitude addition could be described as follows: The signs of two inputs are checked for similarity. If they are same, the magnitudes of two inputs are added and a six bit sum is obtained as the final magnitude. The sign of input A is taken as the final sign. If the signs

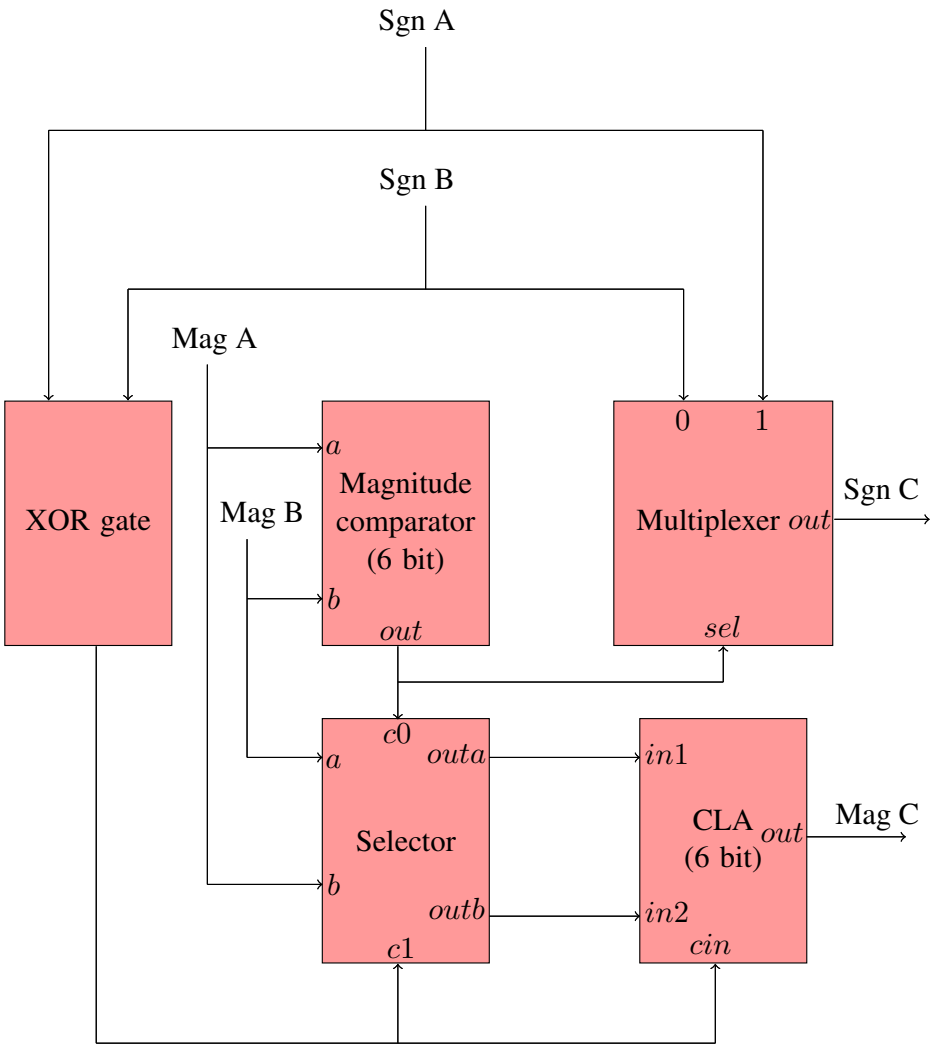

Fig. 5.9: Seven bit sign-magnitude adder. Mag corresponds to magnitude and Sgn corresponds to the sign.

are different, a comparison is done among the magnitudes of two inputs to find the greater one. The magnitude of the smaller value is subtracted from the magnitude of the larger one to obtain the final magnitude. The sign of the larger number is taken as the final sign. Fig. 5.9 shows the architecture of the seven bit sign magnitude adder. The sign-magnitude adder consists of a XOR gate, a magnitude comparator, a selector and a multiplexer. The signs of two inputs are passed on as inputs to the XOR gate. The outputs of the XOR gate and the magnitude comparator are passed on as control signals to the selector. Based on those control signals, the selector selects a combination of input magnitudes or their inverted ones. This combination is chosen based on Table 5.3. They are passed on as inputs to a six bit Carry Look Ahead (CLA) adder. CLA also takes in the output of XOR gate as the carry input. The adder output gives the final magnitude. The two input multiplexer is controlled by the output of the magnitude comparator and the multiplexer selects the final sign.

5.2.2 Symbol Node Design

Fig. 5.10 shows the architecture of the symbol node k. The node accepts a channel sample y_k and computes the initial hard decision of the sample. The initial hard decision (original x_k) is the sign of the received sample. The initial decision is passed on to a multiplexer that is controlled by the Enable signal. The Enable signal is set high for only one cycle, when the channel sample is obtained. The decision is then passed on to the interleaver. The interleaver sends the hard decision to all the check nodes that are connected to the symbol node. The hard decision is also passed on to the XOR gate and the correct sign of $x_k y_k$ is obtained and is then combined with magnitude of the channel sample y_k to obtain the final $x_k y_k$. The interleaver obtains all the six syndromes from the adjacent check nodes. They are passed on to the scaled syndrome sum module.

The scaled syndrome sum module counts the number of ones in the incoming syndromes, computes the sum and scales down the syndrome sum by a factor of six. Fig. 5.11 shows the architecture of the count part of the scaled syndrome module. The count module consists of three full adders and a half adder. The count module takes in all the six one

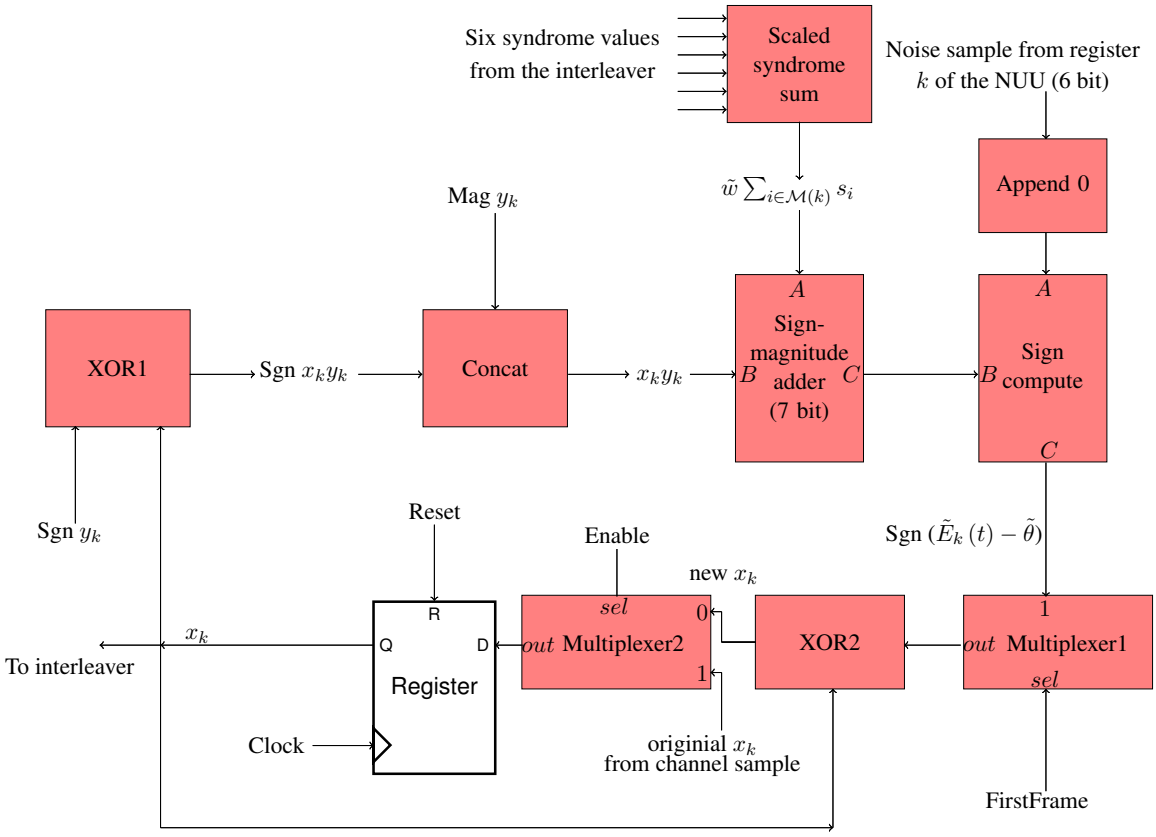

Fig. 5.10: Architecture of a symbol node k. Mag corresponds to magnitude and Sgn corresponds to the sign.

Table 5.4: Scaled syndrome sum lookup table.

Count	**Scaled syndrome sum values**	**Sign-magnitude values**
0	6/6(1)	0010000
1	4/6(0.666)	0001010
2	2/6(0.333)	0000101
3	0/6(0)	0000000
4	−2/6(−0.333)	1000101
5	−4/6(−0.666)	1001010
6	−6/6(−1)	1010000

bit syndromes and produces a three bit output that indicate the number of ones in the incoming syndromes. The number of ones can vary from zero to six. Since, sign-magnitude

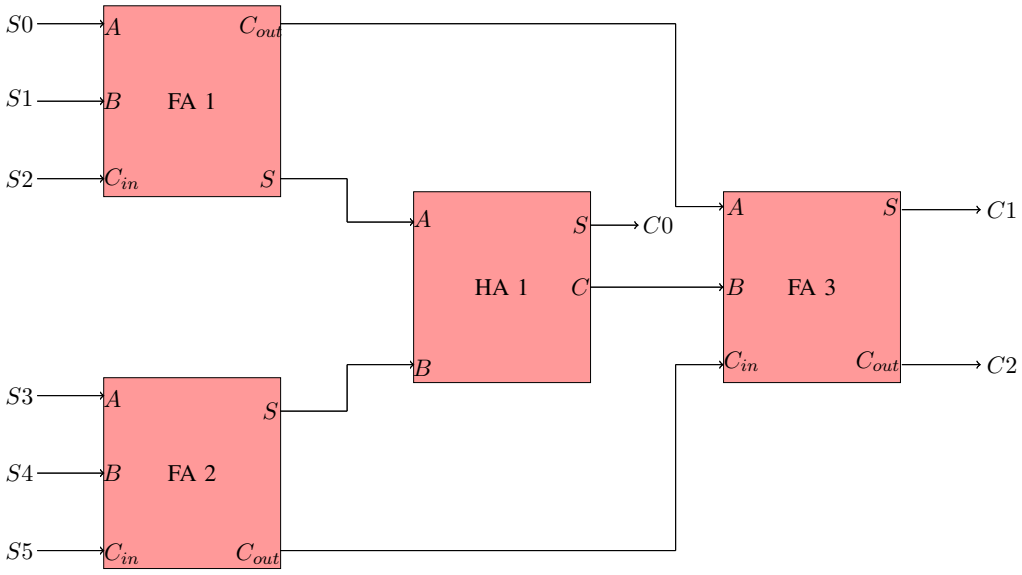

Fig. 5.11: Architecture of count module. The module produces a three bit output ($C0 - -C2$). FA indicates a full adder and HA indicates a half adder.

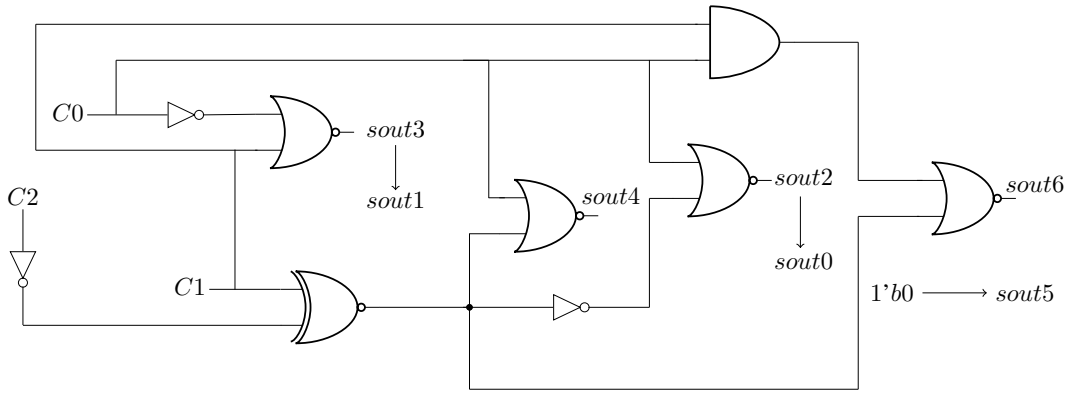

Fig. 5.12: Logic diagram of the lookup table.

format is used for representation, every count of one corresponds to a negative one and every count of zero corresponds to a positive one. Hence, the final output of the scaled syndrome sum can be obtained from Table 5.4. Then according to the count, a look-up table is implemented that calculates the final output in the sign-magnitude format. The logic diagram of the look-up table is shown in Fig. 5.12. The look-up table consists of three

inverters, four NOR gates, one XNOR gate and one AND gate. The computed $x_k y_k$ and the scaled syndrome values are passed on as inputs to a seven bit sign-magnitude adder and the sum is passed on to the sign compute module through a multiplexer that is controlled by FirstFrame signal. The output of the sign compute module is only passed to the XOR gate during the decoding phase (FirstFrame=1). The sign compute module is the same as the sign-magnitude adder except that it does not contain the CLA adder and the selector. Since, the objective is to calculate the final sign, the magnitude of the sum is not computed and is ignored. The six bit noise sample stored in the register k is obtained and a zero is added to MSB of the integer part to obtain a seven bit sample. This sample is provided as another input to the sign compute module. The sign compute module obtains the final sign and is then passed on as input to an XOR gate with the current decision being the another input. The XOR output is the new decision. This new decision that appears at the input of the register is made transparent on the next positive clock edge and is passed on to the interleaver and the XOR gates. All the above mentioned operations are repeated again.

5.2.3 Check Node and ETU Design

The check node is an XOR gate that takes in 32 decisions from the neighboring symbol nodes and calculates the syndrome. The purpose of the ETU is to detect convergence. The output of all the check nodes are passed on as inputs to the ETU. The ETU is an OR gate that computes the ChkOut signal. The ChkOut is low only when all the syndromes are low, thereby detecting convergence.

Chapter 6

Implementation Results

This chapter discusses the implementation results and compares all the parameters of the NGDBF decoder to other existing state of the art decoders. The decoder was designed in Verilog. A detailed description of the ASIC design flow is provided in appendix A. The design was synthesized using Synopsys Design Compiler and placed and routed using Cadence SOC Encounter. Synopsys Primetime was used for post-layout timing analysis and power analysis. The designs were synthesized to a commercial 65 nm low-power standard cell libraries. All the presented results use nominal operating conditions. Nominal operating conditions correspond to nominal process corner, supply voltage of 1 V and temperature of 25 °C. Fig. 6.1 shows the post-layout GDS-II implementation of the final routed design. Table. 6.1 shows the wiring complexity of the implemented decoder in comparison to the IDB decoder. The total wire-length of NGDBF decoder is 7.37 m, while the total wire-length of the IDB decoder is 10.95 m. The wire-length of the NGDBF decoder is 0.672 times smaller compared to the IDB decoder.

Fig. 6.2 shows the BER results obtained from the post-layout functional simulations. From the plot, it is observed that the post-layout simulation matches closely with the system level simulation. The NGDBF decoder is able to achieve an error rate of 10^{-7} at an E_b/N_0 of 4.45 dB. Fig. 6.3 shows the average number of iterations taken by the routed NGDBF decoder to converge. The average throughput obtained is also plotted. Throughput is low at lower values of E_b/N_0 and increases with increase in E_b/N_0. The maximum frequency at which the decoder could operate successfully was estimated to be 133.33 MHz (T =7.5 ns). At this frequency, the decoder crosses the required minimum throughput of 10 GBPS at an E_b/N_0 of 4.3 dB. At E_b/N_0 of 4.45 dB, where the decoder attains an error rate of 10^{-7}, the average throughput is 13.5 GBPS. Fig. 6.4 shows the plot of energy per bit consumed

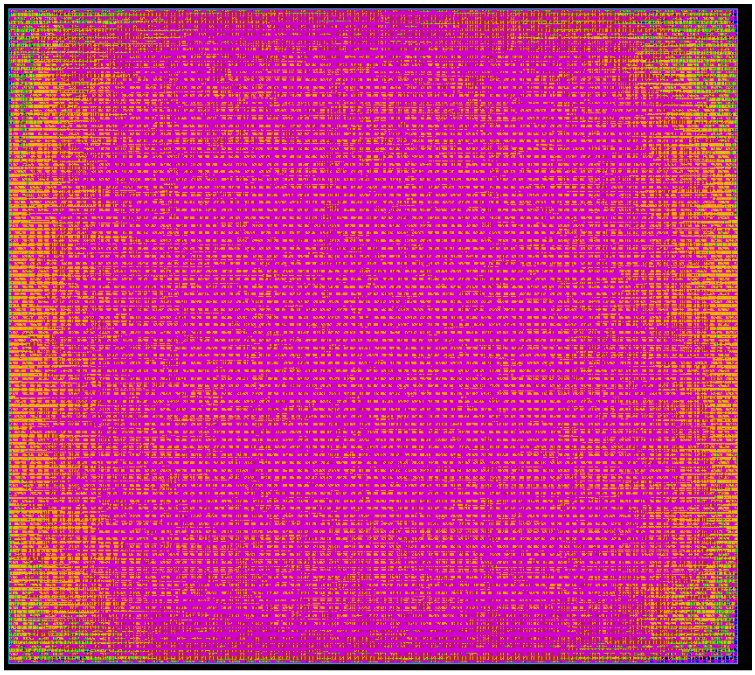

Fig. 6.1: Layout view of NGDBF decoder implementation.

Table 6.1: Wire complexity of decoders.

Design	**Wirelength(m)**
IDB	10.95
NGDBF	7.37

and the average power dissipated at different values of E_b/N_0. Energy per bit is obtained as follows:

$$Energy\ per\ bit = \frac{Average\ Power}{Average\ Throughput} \tag{6.1}$$

Energy per bit decreases with increase in E_b/N_0. This is due to increase in throughput with reduction in average number of iterations. Even though the average power consumption increases with increase in E_b/N_0, that increase is less pronounced compared to the increase in the average throughput. This causes the energy consumption to decrease with E_b/N_0.

Fig. 6.5 shows the critical path of the routed design. In any synchronous design,

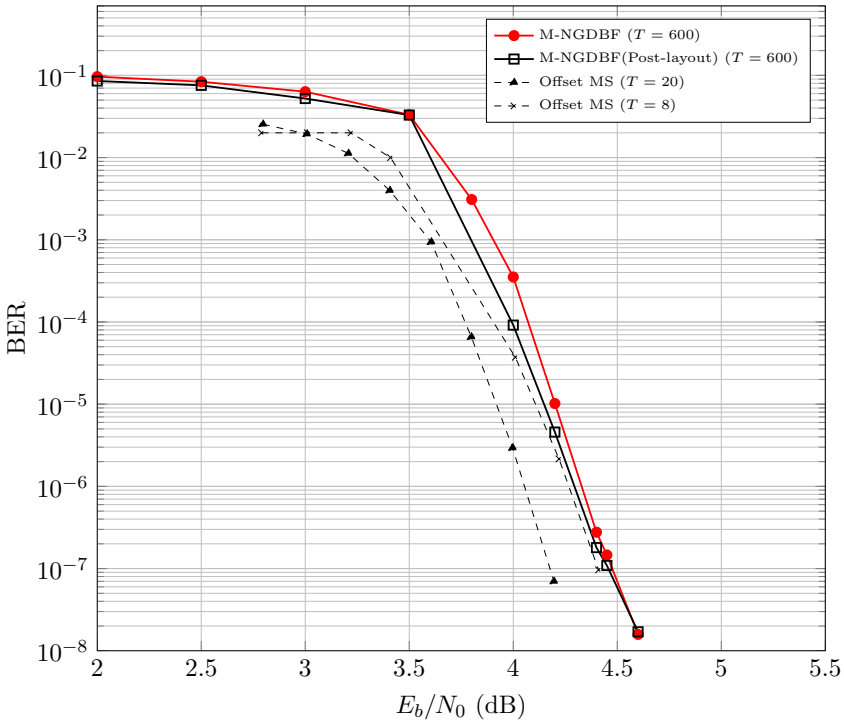

Fig. 6.2: BER for routed NGDBF compared to a benchmark offset MS decoder for the IEEE 802.3 standard LDPC code with maximum iterations limited to T.

the critical path always starts at the output of a register, propagates through a set of combinational logic and ends at the input of a register. In the case of our decoder, the critical path starts at the output of the register in the start symbol node and ends at the input of the register at the destination symbol node. Based on that, the critical path of the decoder can be split into three main sub-paths. The first sub-path involves the decision register of the start symbol node, buffers between the start symbol node and the adjacent check node. The second sub-path involves the check node and the buffers between the check node and the destination symbol node. The third sub-path involves the combinational logic in the destination symbol node that ends at the input of the destination register. The start symbol node is symbol node 1798. The check node is check node 55. The terminating symbol node is symbol node 102. Table 6.2 shows the contribution of each component to the critical path delay. As expected, the check node and symbol node delays dominate the critical path delay. The delay in the check node is 2.12 ns. The delay in the symbol node

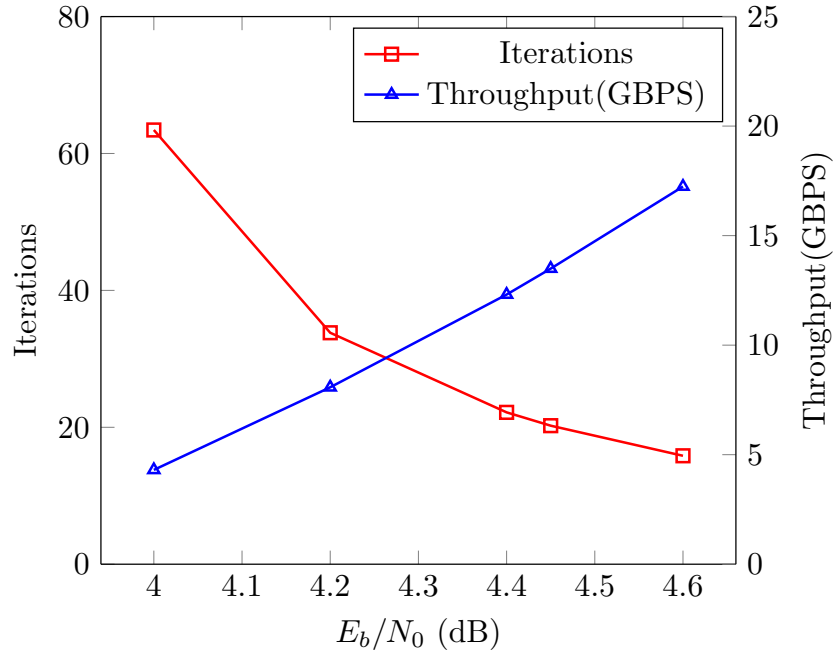

Fig. 6.3: Average number of iterations and average throughput versus E_b/N_0 for routed NGDBF design for IEEE 802.3 standard LDPC code with maximum iterations limited to T.

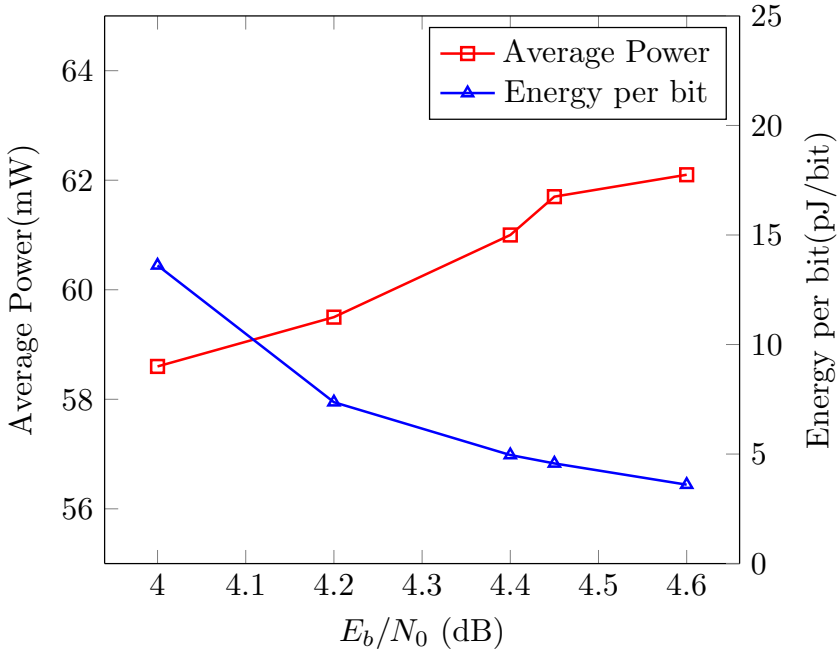

Fig. 6.4: Energy per bit consumption for routed NGDBF design for IEEE 802.3 standard LDPC code with variation in E_b/N_0.

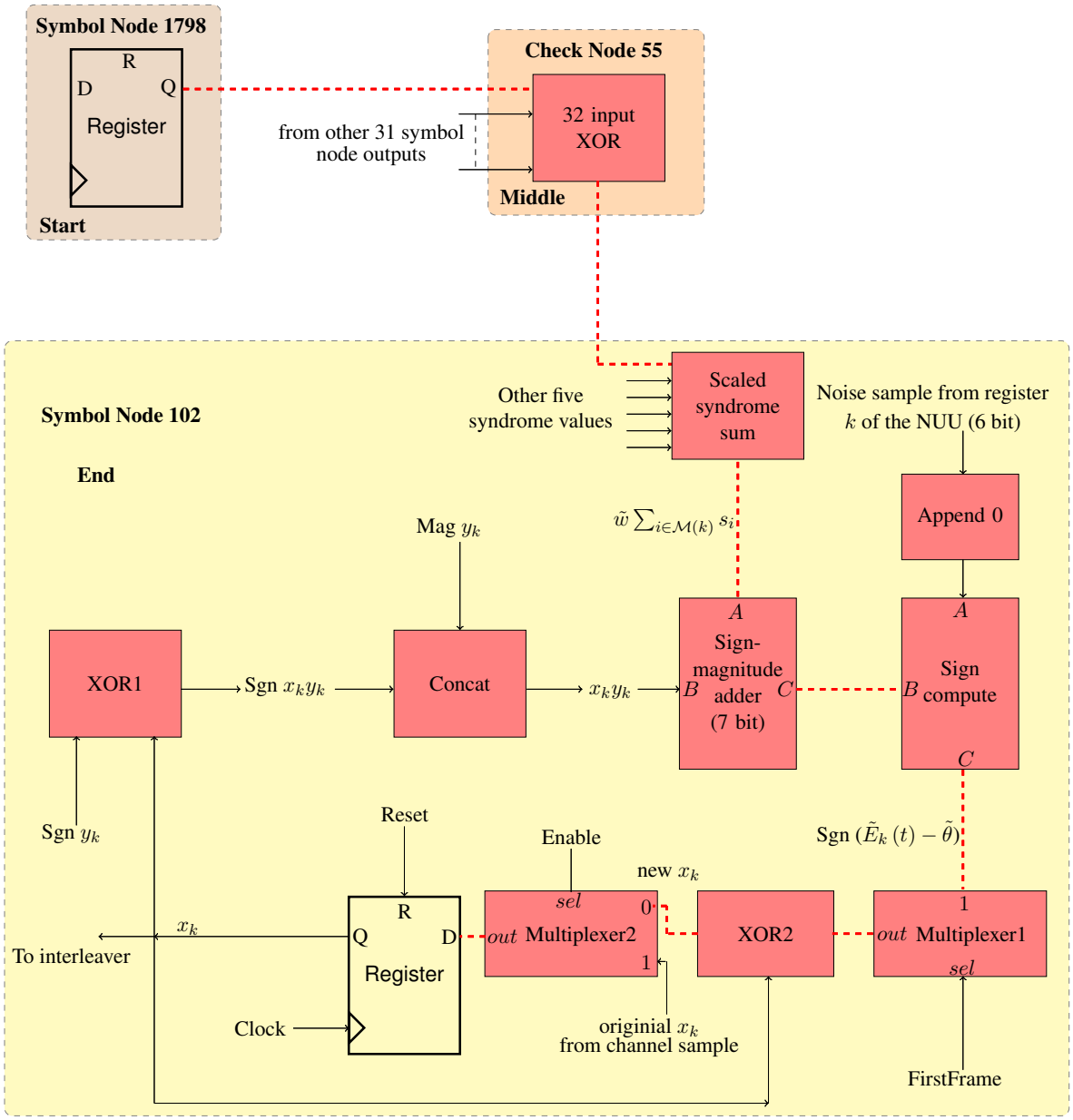

Fig. 6.5: Critical path showing the start, middle and the end blocks. The actual critical path is highlighted using thick broken lines.

Table 6.2: NGDBF critical path.

Component	**Delay(ns)**
Symbol node 1798 clock buffers	0.564
Symbol node 1798 clock-Q	0.254
Symbol node 1798 to check node 55 buffers and interconnect	0.287
Check Node 55 XOR gates	2.12
Check node 55 buffers to symbol node 102 buffers and interconnect	0.69
Symbol node 102 combinational logic	3.58
Total Delay	**7.495**

is 3.58 ns. The major contribution in the symbol node delay comes from the adders. The total delay of adders present in the syndrome scale unit and in the sign-magnitude adder unit is 1.068 ns. Critical path is highlighted in Fig. 6.5 by thick broken lines.

The implementation results for NGDBF decoder are summarized in Table 6.3, along with other works on the 10GBASE-T Ethernet decoder. The NGDBF decoder attains significant improvements in area, error rate and energy efficiency. The area of the routed NGDBF design is 0.81 mm^2. Among other works listed in Table. 6.3, IDB decoder has the smallest area. The IDB decoder consumes 1.44 mm^2. The NGDBF decoder occupies 43.7% smaller silicon area than the IDB decoder. It also occupies 75% smaller area than stochastic MTFM decoder, which has the second smallest area. Hence, the proposed NGDBF decoder outperforms other existing state of the art decoders in terms of area.

At E_b/N_0 = 4.55 dB, the average throughput of the NGDBF decoder is 14.6 GBPS. This is lower than the IDB and the split-row MS decoders. This is due the fact that at E_b/N_0 = 4.55 dB, average number of iterations of the NGDBF decoder is much higher compared to both the IDB and the split-row MS algorithms. However, the NGDBF decoder is able to operate at a throughput that is higher than the required minimum limit of 10 GBPS. However, at E_b/N_0 = 4.55 dB, the NGDBF outperforms IDB and the split-row MS in terms of average power consumption. The average power consumption is 61.6 mW. This is much lower than the IDB and the split-row MS decoders. The standard metric for comparison of any decoders is energy per bit and throughput per unit area [39]. At E_b/N_0 = 4.55 dB,

NGDBF decoder is second best after IDB in terms of energy efficiency. The energy per bit of NGDBF design at $E_b/N_0 = 4.55$ dB is 2.05 times better than the normalized energy per bit of the split-row MS design. In terms of throughput per unit area, both the IDB and split-row MS outperform the NGDBF decoder. The throughput per unit area of the NGDBF decoder is 4.3 times better than layered offset MS decoder at $E_b/N_0 = 4.55$ dB.

At $E_b/N_0 = 5.5$ dB, the average throughput of the NGDBF decoder is 36.4 GBPS. This is lower than the IDB, stochastic MTFM and the offset MS decoders. However, the NGDBF decoder is able to operate at a throughput that is much higher than the required limit of 10 GBPS. At $E_b/N_0 = 5.5$ dB, the NGDBF outperforms IDB, stochastic MTFM and the offset MS decoders in terms of average power consumption. The average power consumption is 63 mW. This is much lower than the IDB and the NGDBF decoders. At $E_b/N_0 = 5.5$ dB, NGDBF is the most energy efficient decoder. The energy per bit of NGDBF at $E_b/N_0 = 5.5$ dB is 1.6 times better than IDB design and 23.52 times better than the normalized energy per bit of the offset MS design. This energy efficiency is the lowest reported in the research literature for 10GBASE-T Ethernet decoder. NGDBF has the second best throughput per unit area after IDB and outperforms the offset MS, layered offset MS decoder and the stochastic decoder at $E_b/N_0 = 5.5$ dB. The throughput per unit area of the NGDBF decoder is 10.73 times better than layered Offset MS decoder at $E_b/N_0 = 5.5$ dB.

The offset MS decoder takes lowest E_b/N_0 to reach the error rate of 10^{-7}. However, offset MS algorithm is very complex and incurs a large area overheard. Among the other simplified implementations, NGDBF has the best performance. NGDBF achieves a gain of 0.05 dB compared to the IDB and 0.1 dB compared to split-row MS. Even though the stochastic MTFM algorithm has a much higher complexity compared to the NGDBF algorithm, NGDBF takes the same E_b/N_0 as the stochastic MTFM to reach an error rate of 10^{-7}.

Fig. 6.6 shows the plot of energy efficiency versus the area efficiency of all the reported 10GBASE-T Ethernet decoders operating at a minimum required throughput of 10 Gbit/s.

Table 6.3: Implementation results for NGDBF and comparison with other works.

Parameter	**This Work**	[31]	[29]	[30]	[32]	[28]
Decoding algorithm	M-NGDBF	IDB	Split-Row MS	Stochastic MTFM	Offset MS	Layered Offset MS
Technology	65 nm	65 nm	65 nm	90 nm	65 nm	90 nm
Quantization bits	7	6	5	6	4	4
Area (scaled to 65 nm) (mm^2)	**0.81**	1.44	4.84	6.38 (3.33)	5.35	5.35(2.79)
Maximum iterations	600	315	11	400	$8+6$ post-processing	4
E_b/N_0 at BER = 10^{-7}	4.45	4.5	4.55	4.45	4.25	4.4
Supply voltage (V)	1.0	1.0	1.3	1.0	1.2	1.2
Clock frequency (Mhz)	133.33	520	195	500	700	137
Minimum throughput (GBPS)	0.46	3.38	36.3	2.56	14.9	11.7
At E_b/N_0 = 4.55 dB						
Average power (mW)	61.6	462	1359	–	–	–
Average throughput (GBPS)	14.6	126.3	92.8	–	–	11.7
Energy per bit (pJ/bit)	4.21	3.65	14.6	–	–	–
Throughput per scaled area (GBPS/mm2)	18.02	87.7	19.2	–	–	2.18
At E_b/N_0 = 5.5 dB						
Average power (mW)	63	478	–	–	2800	–
Average throughput (GBPS)	36.4	171.8	–	61.3	47.7	11.7
Energy per bit (pJ/bit)	**1.73**	2.78	–	–	58.7	–
Throughput per scaled area (GBPS/mm2)	44.94	119.3	–	9.61	8.92	2.18

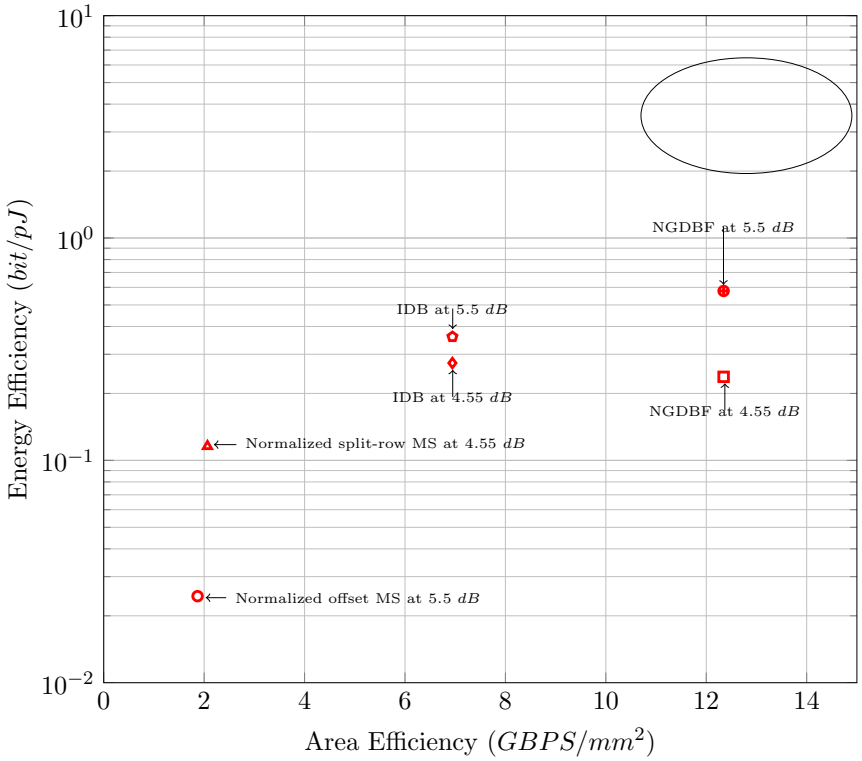

Fig. 6.6: Energy efficiency versus area efficiency of all the reported 10GBASE-T decoders. The ellipse indicates the ideal region.

In this plot, less efficient designs are placed on the lower left corner. The most efficient designs should be placed at the upper right corner as indicated by the ellipse. From the plot, it could be seen that the NGDBF decoder operating at $E_b/N_0 = 5.5$ dB is the most proximate decoder to the ideal region and is the most efficient decoder. The NGDBF decoder at $E_b/N_0 = 4.55$ dB is second most efficient decoder. The offset MS decoder at $E_b/N_0 = 5.5$ dB is the least efficient decoder.

Chapter 7

Conclusions and Future Work

This work proposed a novel low complexity bit flipping algorithm that has a comparable error performance to the standard high complexity BP algorithms and its variants. This algorithm was tested on a variety of codes with variable code-lengths and different degree distributions. The algorithm termed NGDBF, comes within 0.5 dB of the belief propagation algorithm for several tested codes. Unlike other previous GDBF algorithms that provide an escape from local maxima, the proposed algorithm uses only low latency arithmetic operations, which makes them highly efficient in comparison to other bit flipping algorithms and their variants. An ASIC implementation of the algorithm was implemented on a code that is deployed in 10GBASE-T Ethernet standard and the design was shown to be highly efficient both in terms of area and energy consumption at higher values of E_b/N_0 and was able to meet the minimum throughput requirements. Compared to other existing simplified implementations of 10GBASE-T Ethernet decoder, NGDBF implementation also performs better in terms of error rate. The NGDBF decoder also consumes lesser wirelength compared to the IDB decoder.

The main drawback of the NGDBF decoder is that it provides lower throughput compared to other state of the art decoders. This could be compensated by using multiple cores of NGDBF decoder and decoding many frames at the same time to increase the throughput. Since, NGDBF consumes a very low area, n instances of decoder could be deployed for decoding, thereby increasing the throughput by n. The error performance of the NGDBF algorithm could be further boosted by re-decoding. Re-decoding is also applicable to NGDBF, in which the failed frames are re-decoded in successive attempts with different noise values, thereby leading to a possibility of convergence. Re-decoding has already shown to be successful with NGDBF and the performance matches closely with

that of the offset MS algorithm [40]. Even though re-decoding process consumes more energy, the energy dissipated should be much less compared to the energy dissipated by the offset MS decoder. Since all LDPC decoding algorithms are prone to error-floor at higher E_b/N_0, the error floor performance of the NGDBF algorithm needs to be investigated by Field Programmable Gate Array (FPGA) emulation. This could also lead to the analysis and characterization of trapping sets of the NGDBF algorithm on certain standard codes. The relationship between the statistics of the added noise and the ability to escape certain trapping sets needs to be studied carefully.

References

[1] 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.

[2] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *Information Theory, IEEE Transactions on*, vol. 47, no. 7, pp. 2711–2736, 2001.

[3] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 6, pp. 976–996, 2003.

[4] F. Guilloud, E. Boutillon, J. Tousch, and J.-L. Danger, "Generic description and synthesis of LDPC decoders," *Communications, IEEE Transactions on*, vol. 55, no. 11, pp. 2084–2091, 2007.

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

[6] N. Miladinovic and M. P. C. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *Information Theory, IEEE Transactions on*, vol. 51, no. 4, pp. 1594–1606, 2005.

[7] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *Communications Letters, IEEE*, vol. 8, no. 3, pp. 165–167, Mar. 2004.

[8] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *Communications Letters, IEEE*, vol. 9, no. 9, pp. 814–816, 2005.

[9] X. Wu, C. Zhao, and X. You, "Parallel weighted bit-flipping decoding," *Communications Letters, IEEE*, vol. 11, no. 8, pp. 671–673, Aug. 2007.

[10] 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.

[11] M. Ismail, I. Ahmed, and J. Coon, "Low power decoding of LDPC codes," *ISRN Sensor Networks*, vol. 2013, no. 650740, 2013.

[12] R. Haga and S. Usami, "Multi-bit flip type gradient descent bit flipping decoding using no thresholds," in *Information Theory and its Applications (ISITA), 2012 International Symposium on*, 2012, pp. 6–10.

[13] T. Phromsa-ard, J. Arpornsiripat, J. Wetcharungsri, P. Sangwongngam, K. Sripimanwat, and P. Vanichchanunt, "Improved gradient descent bit flipping algorithms for LDPC decoding," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, 2012, pp. 324–328.

[14] 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.

[15] X. Wu, C. Ling, M. Jiang, E. Xu, C. Zhao, and X. You, "New insights into weighted bit-flipping decoding," *Communications, IEEE Transactions on*, vol. 57, no. 8, pp. 2177–2180, Aug. 2009.

[16] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, 2003.

[17] C. Winstead, V. C. Gaudet, A. C. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*. IEEE, 2005, pp. 1116–1120.

[18] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *Communications Letters, IEEE*, vol. 10, no. 10, pp. 716–718, 2006.

[19] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *Signal Processing, IEEE Transactions on*, vol. 56, no. 11, pp. 5692–5703, 2008.

[20] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross, "Tracking forecast memories for stochastic decoding," *Journal of Signal Processing Systems*, vol. 63, no. 1, pp. 117–127, 2011.

[21] S. Sharifi Tehrani, C. Winstead, W. J. Gross, S. Mannor, S. L. Howard, and V. C. Gaudet, "Relaxation dynamics in stochastic iterative decoders," *Signal Processing, IEEE Transactions on*, vol. 58, no. 11, pp. 5955–5961, 2010.

[22] N. Onizawa, W. J. Gross, T. Hanyu, and V. C. Gaudet, "Clockless stochastic decoding of low-density parity-check codes: Architecture and simulation model," *Journal of Signal Processing Systems*, pp. 1–10, 2013.

[23] 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.

[24] C. K. Ngassa, V. Savin, and D. Declercq, "Unconventional behavior of the noisy min-sum decoder over the binary symmetric channel," in *Proc. International Workshop on Information Theory and its Applications (ITA)*, San Diego, Feb. 2014.

[25] ——, "Min-sum-based decoders running on noisy hardware," in *Proc. IEEE GLOBECOM*, Dec. 2013.

[26] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "Vlsi architectures for iterative decoders in magnetic recording channels," *Magnetics, IEEE Transactions on*, vol. 37, no. 2, pp. 748–755, Mar 2001.

[27] T. Zhang and K. Parhi, "Vlsi implementation-oriented (3, k)-regular low-density parity-check codes," in *Signal Processing Systems, 2001 IEEE Workshop on*, 2001, pp. 25–36.

[28] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm2 10gbase-t ethernet ldpc decoder chip in 90 nm cmos," in *Solid State Circuits Conference (A-SSCC), 2010 IEEE Asian*, Nov 2010, pp. 1–4.

[29] T. Mohsenin *et al.*, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Trans. Circ. Syst. I, Reg. Papers*, vol. 57, pp. 1048–1061, May 2010.

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

[31] 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.

[32] 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.

[33] N. Mobini, A. Banihashemi, and S. Hemati, "A differential binary message-passing ldpc decoder," *Communications, IEEE Transactions on*, vol. 57, no. 9, pp. 2518–2523, September 2009.

[34] 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

[35] I. P802.3an. 10gbase-t task force. [Online]. Available: http://www.ieee802.org/3/an

[36] D.-U. Lee, W. Luk, J. Villasenor, G. Zhang, and P. Leong, "A hardware gaussian noise generator using the wallace method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 911–920, Aug 2005.

[37] J. Malik, A. Hemani, and N. Gohar, "Unifying cordic and box-muller algorithms: An accurate and efficient gaussian random number generator," in *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, June 2013, pp. 277–280.

[38] J.-L. Danger, A. Ghazel, E. Boutillon, and H. Laamari, "Efficient fpga implementation of gaussian noise generator for communication channel emulation," in *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on*, vol. 1, 2000, pp. 366–369 vol.1.

[39] F. Kienle, N. Wehn, and H. Meyr, "On complexity, energy- and implementation-efficiency of channel decoders," *Communications, IEEE Transactions on*, vol. 59, no. 12, pp. 3301–3310, December 2011.

[40] T. Tithi, C. Winstead, and G. Sundararajan, "Decoding LDPC codes via noisy gradient descent bit-flipping with re-decoding," *CoRR*, vol. abs/1503.08913, 2015. [Online]. Available: http://arxiv.org/abs/1503.08913

Appendices

Appendix A

Decoder ASIC Design Flow

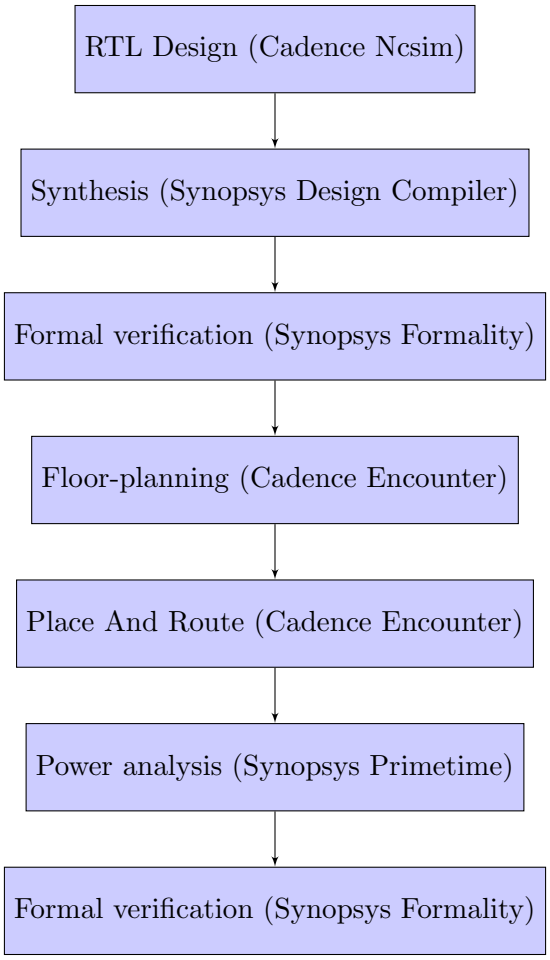

Fig. A.1: Design flow

Vita

Gopalakrishnan Sundararajan

Journal Articles

- **Sundararajan, G.**; Winstead, C.; Boutillon, E., "Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes," IEEE Transactions on Communications, vol.62, no.10, pp.3385,3400, Oct. 2014.
- **Sundararajan, G.**; Winstead, C., "ASIC Implementation Of a Noisy Gradient Descent Bit-Flipping Decoder," IEEE Transactions on Signal Processing, Under Preparation 2015.

Conference Papers

- Winstead, C.; **Sundararajan, G.**; "A Case Study in Noise Enhanced Computing: Noisy Gradient Descent Bit Flip Decoding," Designing with Uncertainty - Opportunities & Challenges workshop, 17-19 Mar. 2014.
- **Sundararajan, G.**; Winstead, C., "A Winner-Take-All circuit with improved accuracy and tolerance to mismatch and process variations," Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), vol., no., pp.265,268, 4-7 Aug. 2013.
- Tang, Yangyang and **Sundararajan, G.** and Winstead, Chris and Boutillon, Emmanuel and Jégo, Christophe and Jézéquel, Miche "Techniques and prospects for fault-tolerance in post-CMOS ULSI," ULSIWS 2012: 21st International Workshop on Post-Binary ULSI Systems, vol., no., pp.1-7, 2012.

- Stine, J.E.; Jun Chen; Castellanos, I.; **Sundararajan, G.**; Qayam, M.; Kumar, P.; Remington, J.; Sohoni, S., "FreePDK v2.0: Transitioning VLSI education towards nanometer variation-aware designs," IEEE International Conference on Microelectronic Systems Education, 2009. MSE '09, vol., no., pp.100,103, 25-27 July 2009.