Stochastic Decoding of Low-Density Parity-Check Codes

Saeed Sharifi Tehrani

Doctor of Philosophy

Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec, Canada

January 2011

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Doctor of Philosophy

DEDICATION

I dedicate this dissertation to my parents for their unconditional love and support that made it possible.

ACKNOWLEDGEMENTS

I consider myself fortunate to have been able to work in this doctoral research with professors and talented scholars from universities in Canada, the United States, and France. First and foremost, I would like to express my sincere gratitude to my advisors, Prof. Warren J. Gross and Prof. Shie Mannor, for their guidance as well as their continuous encouragement and support throughout this five-year journey.

I wish to thank Prof. Paul H. Siegel, my host advisor during a six-month visit to the Center for Magnetic Recording Research (CMRR), University of California, San Diego (UCSD). I benefited extensively from his vast knowledge and experience, and was impressed by his open and warm personality. The joint stochastic decoding of low-density parity-check codes and partial-response channels, presented in Chapter 6 of this dissertation, was developed during my visit to the CMRR, UCSD, in 2010.

I am thankful to Prof. Fabrice Labeau and Prof. Zeljko Zilic for taking precious time to serve on my advisory committee. I am also thankful to Prof. Vincent Gaudet at the University of Waterloo for helpful discussions and for encouraging me to start my Ph.D. at McGill University. His interesting paper with Anthony Rapley on stochastic decoding was the first inspiration for this doctoral research. I would like to thank Prof. Chris Winstead at Utah State University, and Prof. Sheryl L. Howard at Northern Arizona University, for helpful discussions and their support during this doctoral research. I am also indebted to my M.Sc. advisors at the University of Alberta, Prof. Bruce F. Cockburn and Prof. Stephen Bates, for their kind support.

I would like to thank the coauthors of my publications for their collaborations. I am thankful to Prof. Christophe Jego at the ENSEIRB-MATMECA

Bordeaux, France, for his help in applying the stochastic decoding approach to the turbo-oriented adaptive belief propagation. I would like to thank Ali Naderi and Guy-Armand Kamendje for their help in the Verilog development and place-and-routing of ASIC stochastic decoders. I also would like to thank Bo Zhu for providing and discussing the initial results for stochastic Reed-Solomon decoding, and Saied Hemati for his helpful suggestions on making hard-decisions in stochastic decoders using majority criterion and on performing the tracking forecast memory operation using one adder/subtractor unit.

I am thankful to my officemates and my wonderful friends at McGill University and UCSD. They have filled the past five years of my life with so many joyful moments.

I am sincerely grateful to my parents and my brother for always being supportive and encouraging in my pursuit of academic excellence. I would like to express my profound appreciation and deepest gratitude to my parents for their constant care and love. Without them, I would not ever have completed this dissertation.

I acknowledge the Natural Science and Engineering Research Council of Canada (NSERC) for awarding the Alexander Graham Bell Canada Graduate Scholarship (CGS Doctoral) and the Michael Smith Foreign Study Scholarship. I also acknowledge the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) for awarding the Quebec International Internship Scholarship. This doctoral research was financially supported by the NSERC, the FQRNT, and the Canada Research Chair (CRC) funds. Also, research work presented in Chapter 6 of this dissertation was supported in part by the United States National Science Foundation (NSF) grant number CCF-0829865. Results reported in Chapter 5 of this dissertation were obtained in part thanks to the use of WestGrid and CLUMEQ computing resources.

ABSTRACT

Low-Density Parity-Check (LDPC) codes are one of the most powerful classes of error-control codes known to date. These codes have been considered for many recent digital communication applications. In this dissertation, we propose stochastic decoding of state-of-the-art LDPC codes and demonstrate it as a competitive approach to practical LDPC decoding algorithms.

In stochastic decoding, probabilities are represented as streams of random bits using Bernoulli sequences in which the information is contained in the statistics of the bit stream. This representation results in low hardware-complexity processing nodes that perform computationally-intensive operations. However, stochastic decoding is prone to the acute problem of latching. This problem is caused by correlated bit streams within cycles in the code's factor graph, and significantly deteriorates the performance of stochastic LDPC decoders.

We propose edge memories, tracking forecast memories, and majority-based tracking forecast memories to address the latching problem. These units efficiently extract the evolving statistics of stochastic bit streams and rerandomize them to disrupt latching. To the best of our knowledge, these methods are the first successful methods for stochastic decoding of state-of-the-art LDPC codes.

We present novel decoder architectures and report on several hardware implementations. The most advanced reported implementation is a stochastic decoder that decodes the (2048,1723) LDPC code from the IEEE 802.3an standard. To the best of our knowledge, this decoder is the most silicon area-efficient and, with a maximum core throughput of 61.3 Gb/s, is one of the fastest fully parallel soft-decision LDPC decoders reported in the literature.

We demonstrate the performance of this decoder in low bit-error-rate regimes.

In addition to stochastic LDPC decoding, we propose the novel application of the stochastic approach for joint decoding of LDPC codes and partial-response channels that are considered in practical magnetic recording applications. Finally, we investigate the application of the stochastic approach for decoding linear block codes with high-density parity-check matrices on factor graphs. We consider Reed-Solomon, Bose-Chaudhuri-Hocquenghem, and block turbo codes.

ABRÉGÉ

À ce jour, les codes Low-Density Parity-Check (LDPC) font partie des codes correcteurs d'erreurs les plus performants. Ces codes sont inclus dans différents standards de communications numériques. Dans ce manuscrit, nous proposons d'utiliser le décodage stochastique pour les codes LDPC. D'autre part, nous démontrons que pour les codes LDPC, le décodage stochastique représente une alternative réaliste aux algorithmes de décodage existants.

Dans le processus de décodage stochastique, les probabilités sont représentées sous forme de séquences de Bernoulli. L'information est contenue dans la statistique de ces flux binaires aléatoires. Cette représentation particulière permet d'exécuter des calculs intensifs avec une faible complexité matérielle. Cependant le décodage stochastique est enclin au problème du verrouillage ("latching"). La corrélation entre les bits des différents flux au sein des cycles du graphe biparti dégrade les performances du décodage stochastique des codes LDPC.

Pour résoudre le problème du verrouillage, nous proposons trois solutions: les mémoires de branche, les mémoires de suivi, et les mémoires de suivi à majorité. Ces différents composants permettent de suivre l'évolution de la statistique des flux binaires et de réintroduire des éléments aléatoires au sein des séquences observées, minimisant ainsi le phénomène de verrouillage. À notre connaissance, il s'agit là des premiers résultats probants permettant un décodage stochastique efficace des codes LDPC.

Nous proposons de nouvelles architectures de décodeurs associées à leurs implantations matérielles respectives. La plus perfectionnée des architectures présentée ici est celle d'un décodeur stochastique pour le code LDPC (2048,1723) associé au standard IEEE 802.3an. À notre connaissance, en comparaison avec

l'état de l'art actuel, ce décodeur dispose du meilleur rapport vitesse/complexité. Le débit maximum (au niveau du coeur), est de 61.3 Gb/s, il s'agit là du plus rapide des décodeurs de codes LDPC à décisions souples connu à ce jour. Nous présentons par ailleurs les performances de ce décodeur à très faible taux d'erreurs binaire.

De plus, nous proposons d'appliquer le calcul stochastique au décodage conjoint des codes LDPC et des canaux à réponse partielle qui est utilisé dans les applications d'enregistrement magnétique. Enfin, nous étudions l'extension du décodage stochastique au décodage des codes en blocs ayant une matrice de parité à forte densité. Nous appliquons le décodage stochastique sur des graphes biparti aux codes Reed-Solomon, Bose-Chaudhuri-Hocquenghem, et aux turbocodes en blocs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivations

Error-control coding (channel coding) is a powerful technique in digital communications used to ensure reliable communication over an unreliable channel. In this technique, error-control codes are used to efficiently add redundant structure to the transmitted data to allow the receiver to detect and correct errors introduced during passage through a noisy and distorting communication channel (see Figure 1–1). Error-control coding has evolved since the advent of information theory by Shannon in 1948 [81] and it has become essential in a wide variety of modern applications [22]. In particular, error-control coding has received a lot of attention in recent years because of the significant progress in designing powerful error-control codes as well as the progress in Very-Large-Scale Integration (VLSI) technology, which has facilitated the hardware implementation of computationally-complex decoding algorithms.

Among different classes of error-control codes, Low-Density Parity-Check (LDPC) codes [33, 34] are one of the most powerful classes known to date. LDPC codes have been considered as forward error correction in several recent communication applications and standards including digital video broadcasting (DVB-S2) [1], 10 Gb/s Ethernet (IEEE 802.3an or 10GBASE-T) standard [2], broadband wireless access (IEEE 802.16e or WiMAX) standard [3],

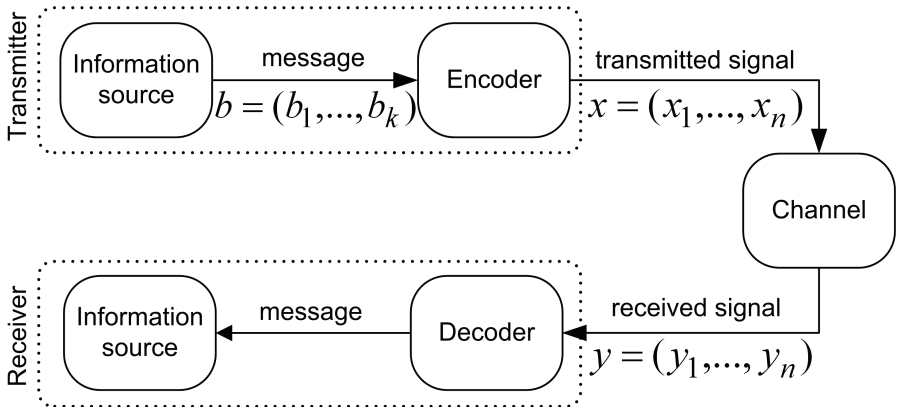


Figure 1–1: The schematic diagram of a communication system.

wireless local area network (IEEE 802.11n or WiFi) standard [4], and deep-space communications [9]. LDPC codes were invented by R. G. Gallager in 1962 [34]. Gallager discovered and applied an iterative decoding algorithm to a new class of error-control codes. He named these codes low-density parity-check codes, because their parity-check matrices had to be sparse in order to have good performance. However, LDPC codes had been ignored for almost three decades mainly because of the requirement for high complexity computation, particularly for LDPC codes with long code length. The advent of turbo codes in 1993 [12] and the observation of their remarkable "capacity-approaching" performance raised many questions and triggered many research efforts toward iterative decoding. Turbo codes made it possible to get within a few tenths of a decibel (dB) away from the Shannon capacity limit at a Bit-Error-Rate (BER) of 10^{-5}. This in fact started a new paradigm in the design of error correcting codes, which led to the rediscovery of LDPC codes in 1995 [57, 58], and graph-based codes. Like Turbo codes, LDPC codes were demonstrated to perform very close to the Shannon limit when decoded iteratively on graphs using a *message-passing* algorithm such as the Sum-Product Algorithm (SPA) [20, 56, 58, 75]. It was also shown that iterative LDPC decoding and turbo decoding of turbo codes are instances of the Pearl's *belief*

propagation algorithm [72], collecting LDPC codes and Turbo codes under the same model [57, 58, 62]. In addition, the graph-based code-description became a common way of representing error correcting codes [50].

An LDPC code and its iterative decoding can be graphically represented by a factor graph [50]. A factor graph is a bipartite graph which consists of two different groups of processing nodes: Variable Nodes (VNs) and Parity-check Nodes (PNs). The VN and PN groups in a factor graph are connected to each other by bidirectional edges. The connection between VNs and PNs are defined by the parity-check matrix of the code. Iterative decoding of LDPC codes usually involves message-passing between VNs and PNs over the edges of the factor graph for some number of decoding iterations. These messages represent the VNs' and PNs' beliefs (in the form of probabilities) about the correctness of the received information from the channel. In general, LDPC decoders can be categorized into fully parallel and partially parallel decoders. In a fully parallel decoder the entire factor graph is implemented in hardware, while in a partially parallel decoder a portion of the graph is implemented and, hence, hardware resource sharing and memory blocks are employed to compute, save and pass probability messages between different portions of the factor graph.

Despite the excellent error-correcting performance of LDPC codes, the hardware implementation of LDPC decoders is complex and challenging [13, 24–26, 28, 90, 107], hence, an LDPC decoder is often implemented optionally, as an additional premium, in communication systems (e.g., in [3, 4]). Powerful LDPC codes usually have long code lengths. Also, their parity-check matrix usually imposes random-like/irregular connections between VNs and PNs. In this respect, a fully parallel hardware implementation of a capacity-approaching LDPC decoder usually requires the implementation of thousands

of processing nodes in a silicon chip. In addition, thousands of physical wires are needed in the silicon chip to accommodate message-passing between VNs and PNs. For instance, a decoder for the LDPC code that is considered for the 10Gb/s Ethernet standard [2], has 2048 VNs and 384 PNs. Each VN communicates with six PNs in each decoding iteration. Therefore, by using W-bit quantization to represent probability messages passed between nodes in a fully parallel implementation, a total of $2048 \times 6 \times 2 \times W = 24576 \times W$ physical wires are needed between PNs and VNs (in both the input and output directions). Using 4-bit and 6-bit quantization, the number of physical wires between VNs and PNs in the decoder chip is 98304 and 147456, respectively. The high number of processing nodes and the abundant number of physical wires make the chip consume a large silicon area. In addition, the random-like connections between VNs and PNs result in long and random physical wires and interconnections across the chip, which causes routing congestion. These long wires also limit the clock frequency and the throughput of the decoder and increase its power consumption [13, 25, 26, 28]. In partially parallel decoders, the large and irregular communication network between VNs and PNs results in large memory blocks and address generation units with high power consumption. Another challenge in the design of LDPC decoders is that simplifying the decoder hardware by using a low number of quantization levels can degrade the error-correcting behavior of an LDPC decoder, particularly, in low BER regimes where LDPC codes are usually supposed to operate [111]. Conversely, most modern communication applications require high throughput decoding while demanding low silicon area and power consumption, as well as good decoding performance in low BER regimes. These challenges

have made efficient LDPC decoding a focal point of research at both the theoretical/algorithm level and the hardware implementation level (see Section 2.2).

To address the above-mentioned problems, this work proposes the stochastic decoding of practical capacity-approaching LDPC codes on factor graphs. Stochastic decoding is a new approach for iterative decoding on graphs. This approach is inspired by the method of stochastic computation developed in the 1960's [32]. In stochastic decoding, instead of propagating probabilistic beliefs by exchanging distinct probability messages, as in the conventional message-passing algorithms, beliefs are conveyed in streams of stochastic bits in a sense that the probability of observing a "1" in a stream is equal to the original (encoded) probability. Therefore, VNs and PNs exchange beliefs in a bit-serial manner along the edges of the graph. Stochastic decoding reduces the hardware-complexity of processing nodes in an LDPC decoder and, more importantly, it significantly reduces the number of physical wires between processing nodes. The first stochastic decoding method was proposed in [36, 74]. However, stochastic decoding methods prior to this work (i.e., [36,37,74,95,97]) resulted in significant decoding performance loss compared to the conventional iterative LDPC decoding methods and thus were not practical solutions for decoding LDPC codes (see Section 2.4).

1.2 Objectives

The objectives in this work are to develop stochastic decoding approaches that (a) can decode practical capacity-approaching LDPC codes on factor graphs, (b) have area-efficient hardware implementations, (c) achieve high throughput, and (d) provide good decoding performance, especially in low BER regimes.

1.3 Contributions

This dissertation proposes stochastic decoding as a new competitive approach for decoding state-of-the-art LDPC codes on factor graphs.

We propose edge-based rerandomization using Edge Memories (EMs) [82, 85, 86] as the first successful stochastic approach in the literature for decoding practical capacity-approaching LDPC codes on factor graphs. We also propose a fully parallel decoder hardware architecture for the EM-based stochastic LDPC decoding and discuss its novel architectural features. We apply this architecture to decode an irregular state-of-the-art (1056,528) LDPC code (chosen from the WiMAX standard [3]) on a field-programmable gate-array (FPGA) device. This decoder is the first stochastic LDPC decoder architecture in the literature that decodes a practical LDPC code. The implemented decoder achieves a clock frequency of 222 MHz and a maximum throughput of about 1.66 Gb/s on FPGA. The EM-based stochastic decoder provides good decoding performance behavior at low BERs. We demonstrate the performance of this decoder down to a BER of about 10^{-8} and compare it with other decoding approaches. We show that the proposed decoder provides a performance within 0.5 dB and 0.25 dB of the floating-point SPA with 32 and 16 iterations, respectively. We compare this decoder with other high throughput FPGA-based fully parallel LDPC decoders in detail and demonstrate that this decoder is one of the fastest and most resource-efficient FPGA-based fully parallel LDPC decoders.

We consider the ASIC implementations of stochastic LDPC decoders. We discuss ASIC implementation challenges of EM-based stochastic decoding and propose edge-based rerandomization using Tracking Forecast Memories (TFMs) [87, 88] to significantly reduce the silicon area consumption of

ASIC stochastic decoders. By comparing EM-based decoders with the TFM-based LDPC decoders, we show that TFM-based decoders provide similar or better decoding performance compared to EM-based decoders while having about 40% to 65% less silicon area consumption. The ASIC TFM-based decoder proposed in this dissertation is the first ASIC stochastic LDPC decoder reported in the literature.

We propose node-based rerandomization using Majority-based Tracking Forecast Memories (MTFMs) [89] for area-efficient high throughput ASIC implementation of stochastic LDPC decoders. We apply the MTFM approach for ASIC implementation of a fully parallel stochastic decoder that decodes the (2048,1723) LDPC code from the IEEE 802.3an (10GBASE-T) standard [2]. This stochastic decoder occupies a silicon core area of 6.38 mm^2 in CMOS 90 nm technology, achieves a maximum clock frequency of 500 MHz, and provides a maximum core throughput of 61.3 Gb/s. The decoder also has good decoding performance and error-floor behavior. We investigate and demonstrate its decoding performance down to a low BER of about 4×10^{-13}. We compare this decoder with several recent ASIC LDPC decoders in detail. To the best of our knowledge, the proposed MTFM-based stochastic LDPC decoder is the most silicon area-efficient fully parallel soft-decision LDPC decoder and it is one of the fastest fully parallel LDPC decoders reported in the literature.

In addition to stochastic LDPC decoding, we consider other applications of the stochastic approach. We propose the novel application of stochastic decoding for joint message-passing decoding of LDPC codes and partial-response channels that are considered in practical magnetic recording applications. We propose low hardware-complexity stochastic processing nodes to perform computationally-intensive operations required in partial-response channel detectors. We present decoding performance results for the dicode

partial-response channel and the Extended Class-4 Partial-Response (EPR4) channel, and discuss the throughput and latency of the proposed method.

Finally, we investigate stochastic decoding of linear block codes with high-density parity-check matrices on factor graphs [83]. Stochastic decoding was previously applied to Reed-Solomon (RS) codes in [112]. In this dissertation, we further investigate stochastic RS decoding and extend the application of stochastic decoding to Bose-Chaudhuri-Hocquenghem (BCH) codes and BCH-based turbo block codes. We also propose efficient hardware implementations of high-degree nodes used in the decoding of linear block codes with high-density parity-check matrices on factor graphs. Results demonstrate decoding performance close to floating-point iterative soft-input soft-output (SISO) decoding while offering nodes with considerably lower complexity compared to fixed-point SISO decoding. These results are the first results reported in the literature for stochastic decoding of Bose-Chaudhuri-Hocquenghem (BCH) codes and BCH-based turbo block codes.

1.3.1 List of Publications and Patent Applications

This doctoral research has resulted in the following publications and patent applications:

- Published or Accepted Journal Articles:
  - i S. Sharifi Tehrani, C. Winstead, W. J. Gross, S. Mannor, S. Howard, and V. C. Gaudet, "Relaxation Dynamics in Stochastic Iterative Decoders," IEEE Transactions on Signal Processing, vol. 58, no. 11, November 2010, pp. 5955-5961.
  - ii 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," IEEE Transactions on Signal Processing, vol. 58, no. 9, September 2010, pp. 4883-4896.

iii S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross, "Tracking Forecast Memories for Stochastic Decoding," Invited paper by Journal of Signal Processing Systems, Special Issue on the DISPS Track of IEEE ICASSP 2009, Springer Publishing, To Appear (online publication: January 2010), DOI: 10.1007/s11265-009-0441-5.

iv S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully Parallel Stochastic LDPC Decoders," IEEE Transactions on Signal Processing, vol. 56, no. 11, November 2008, pp. 5692-5703.

v S. Sharifi Tehrani, C. Jego, B. Zhu, and W. J. Gross, "Stochastic Decoding of Linear Block Codes with High-Density Parity-Check Matrices," IEEE Transactions on Signal Processing, vol. 56, no. 11, November 2008, pp. 5733-5739.

vi S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," IEEE Communications Letters, vol. 10, no. 10, October 2006, pp. 716-718.

• Peer-Reviewed Conference Papers:

i S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross, "Tracking Forecast Memories in Stochastic Decoders ," Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), April 2009, Taipei, Taiwan, pp. 561-564.

ii S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "An Area-efficient FPGA based Architecture for Fully-Parallel Stochastic LDPC Decoding," Proceedings of the IEEE International Workshop on Signal Processing Systems (SiPS), October 17-19, 2007, Shanghai, China, pp. 255-260.

iii S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Survey of Stochastic Computation on Factor Graphs," Proceedings of the 37th IEEE International Symposium on Multiple-Valued Logic (ISMVL), May 14-16, 2007, Oslo, Norway, pp. 54-59.

- Workshop Papers:

i S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "A Novel Architecture for Fully-Parallel Stochastic LDPC Decoders," presented at the 7th annual Analog Decoding Workshop (ADW 2008), July 12th, 2008, Logan, USA.

ii S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," presented at the 6th annual Analog Decoding Workshop (ADW 2007), May 24-25, 2007, Montreal, Canada.

- Submitted Patent Applications:

The following patent applications have been filed by the office of technology transfer at McGill University:

i S. Sharifi Tehrani, P. H. Siegel, S. Mannor, and W. J. Gross,"Method for Joint Decoding of LDPC Codes and Partial-Response Channels and Apparatuses Thereof," United States Patent Application 61/433,997, Filed in January 2011.

ii S. Sharifi Tehrani, S. Mannor, and W. J. Gross,"Method and Systems for Improving Iterative Signal Processing," United States Patent Application 12/566,829, Filed in September 2009.

iii S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Methods and Apparatuses of Mathematical Processing," United States Patent Application 12/250,830, Filed in October 2008.

iv S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Method for Implementing Stochastic Equality Nodes," United States Patent Application 12/153,749. Filed in May 2008.

1.4 Dissertation Outline

Chapter 2 provides background materials. It reviews LDPC codes, the SPA, the strategies and challenges of hardware implementation of LDPC decoders, stochastic computation and its benefits, and early stochastic decoding methods. Chapter 2 is in part based on the material in our papers [82, 84].

Chapter 3 proposes the EM approach as the first successful stochastic method for decoding state-of-the-art LDPC codes. It also proposes the first hardware architecture for stochastic decoding of practical Low-Density Parity-Check (LDPC) codes on factor graphs. Chapter 3 is in part based on the material in our papers [82, 85, 86].

Chapter 4 focuses on ASIC implementations of stochastic LDPC decoders. It discusses the silicon area complexity of stochastic decoders and proposes the TFM approach to significantly reduce the hardware-complexity of stochastic decoders for ASIC implementation. It proposes the first ASIC architecture for stochastic decoding of LDPC codes on factor graphs. Chapter 4 is in part based on the material in our papers [87–89]

Chapter 5 continues the theme of Chapter 4 on ASIC implementation of stochastic decoders. It proposes MTFMs for area-efficient high throughput ASIC implementation of stochastic LDPC decoders. The proposed method is applied for the ASIC implementation of a fully parallel stochastic decoder that decodes the (2048,1723) LDPC code from the IEEE 802.3an (10GBASE-T) standard. To the best of our knowledge, this implemented decoder is the most area-efficient and one of the fastest fully parallel soft-decision LDPC decoders reported in the literature. The decoding performance of this decoder has been investigated down to a BER of about 4×10^{-13}. Chapter 5 is in part based on the material in our paper [89].

Chapter 6 proposes the novel application of stochastic decoding for joint decoding of LDPC codes and partial-response channels that are considered in practical magnetic recording applications. This chapter presents hardware architectures for stochastic processing nodes to perform the complex operations required in the partial-response channel detectors. Performance, latency, and throughput of the proposed joint stochastic decoding method are discussed.

Chapter 7 investigates the application of the stochastic decoding approach to decode linear block codes with high-density parity-check matrices on factor graphs. It demonstrates results for decoding the important and popular classes of RS codes, BCH codes, and BCH block turbo codes. Chapter 7 is in part based on the material in our paper [83].

Finally, Chapter 8 concludes the dissertation and provides potential venues for future work.

CHAPTER 2

Background

2.1 LDPC Codes and Iterative Decoding

A binary (n, k) LDPC code is defined as the null space of a sparse $(n - k) \times n$ binary parity-check matrix $\boldsymbol{H}$. This LDPC code consists of codewords $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ such that

$$\boldsymbol{x}\boldsymbol{H}^T = 0, \tag{2.1}$$

where $\boldsymbol{x}$ contains k information bits and $n - k$ parity bits, and $\boldsymbol{x}\boldsymbol{H}^T$ is computed in the Galois field GF(2) [33]. LDPC codes and their iterative decoding process can be graphically represented using bipartite factor graphs [50]. Factor graphs consist of two distinctive groups of processing nodes, VNs and PNs, and edges that connect VNs to PNs. A factor graph for an (n, k) full-rank LDPC code has n VNs and $n - k$ PNs. The i-th VN, v_i, is connected to j-th PN, c_j, if and only if h_{ji}, the entry in $\boldsymbol{H}$ at row j and column i, is 1. The connecting/communication network between VNs and PNs is called *interleaver* (see Figure 2–1). The number of edges connected to a node (in the interleaver) is referred to as the degree of the node and represented as d_v for the VNs and d_c for the PNs. In regular codes, d_v and d_c are fixed for all VNs and PNs, respectively. In irregular LDPC codes, d_v and d_c vary for different nodes. Figure 2–1 also highlights a length-4 *cycle* in the depicted factor graph. A cycle is a closed path in the graph and its length is defined as the

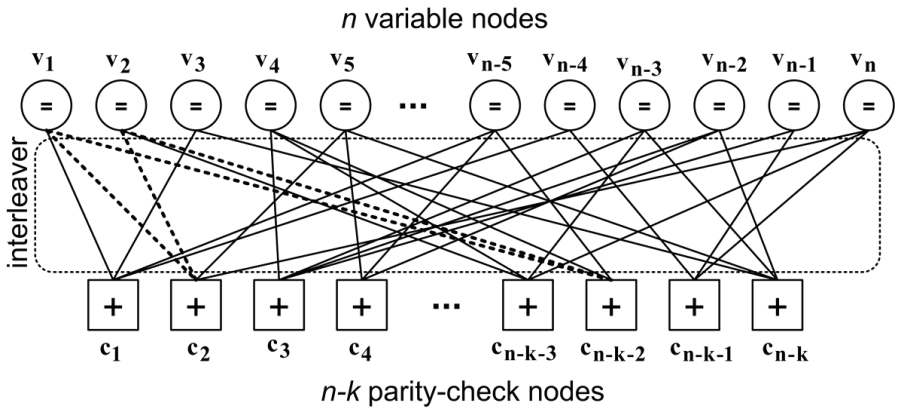


Figure 2–1: A typical factor graph and the interleaver for a full-rank (n, k) LDPC code. A length-4 cycle is dashed. In a conventional implementation with W-bit representation of messages, each edge requires $2W$ wires (for two directions).

corresponding number of path edges. The length of the smallest cycle is the *girth* of the factor graph. While the factor graph of a practical LDPC code can have thousands of cycles, its girth should be more than four to provide a good decoding performance.

LDPC codes are encoded using a $k \times n$ generator matrix, $\boldsymbol{G}$, where

$$\boldsymbol{G}\boldsymbol{H}^T = 0. \tag{2.2}$$

During the encoding process, $n - k$ parity bits are added to $\boldsymbol{b} = (b_1, ..., b_k)$, the information vector, to form the codeword $\boldsymbol{x} = (x_1, ..., x_n)$, where

$$\boldsymbol{x} = \boldsymbol{b}\boldsymbol{G}. \tag{2.3}$$

In a typical LDPC-coded communication system, after encoding the information vector at the transmitter, the codeword $\boldsymbol{x}$ is transmitted through a communication channel. At the receiver, the received vector, $\boldsymbol{y} = (y_1, ..., y_n)$, is passed to the LDPC decoder (see Figure 1–1).

2.1.1 Sum-Product Algorithm

LDPC codes are usually iteratively decoded by means of belief propagation [72] using message-passing algorithms such as the SPA (see [50]) or its less-complex approximation, the Min-Sum Algorithm (MSA) [94]. The SPA is an iterative algorithm for decoding LDPC codes. The SPA uses soft information (probabilities) received from the channel and iteratively processes them using VNs and PNs. The SPA makes decisions by comparing final probabilities to a threshold value (hard-decision) at the end of the decoding process.

Suppose that x_m and y_m, respectively, denote the m-th sample ($1 \leq m \leq n$) in the transmitted vector, $\boldsymbol{x}$, and in the received vector, $\boldsymbol{y}$, in a Binary Phase-Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel with zero mean and a single-sided noise power spectral density of N_0. Let $P_{ml}^{(i)}$ be the probability message from the VN v_m to the PN c_l and $Q_{lm}^{(i)}$ be the probability message from c_l to v_m in the i-th iteration (see Figures 2–2 (a) and (b)). Also, let $N(m)$ be the set of PNs connected to v_m and $M(l)$ be the set of VNs connected to the c_l. The SPA steps in the probability domain can be described as follows (see [50, 79] for details):

I Set the iteration counter to zero ($i = 0$).

II For all VNs, i.e., for $1 \leq m \leq n$, $l \in N(m)$, initialize $P_{ml}^{(0)}$ to P_{CH}^{m}, the *a posteriori* probability (channel probability), computed as:

$$P_{\mathrm{CH}}^{m} = \Pr(x_m = 1|y_m) = \frac{\exp(L_{\mathrm{CH}}^{m})}{\exp(L_{\mathrm{CH}}^{m}) + 1}, \tag{2.4}$$

where L_{CH}^{m} is the log-likelihood ratio (LLR) of y_m and it is computed as:

$$L_{\mathrm{CH}}^{m} = \log\left(\frac{\Pr(x_m = 1|y_m)}{\Pr(x_m = 0|y_m)}\right) = \frac{4y_m}{N_0}. \tag{2.5}$$

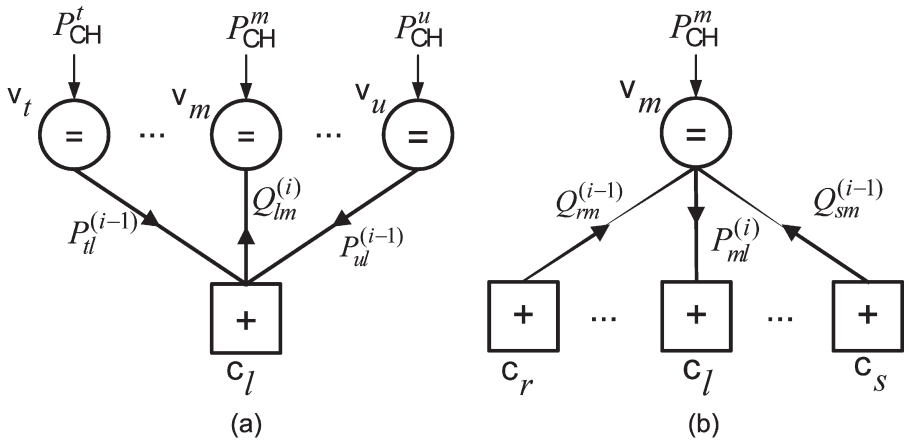


Figure 2–2: Message-passing in the SPA.

III Update all the PNs, i.e., for $1 \leq l \leq n-k$, $m \in M(l)$ compute:

$$Q^{(i)}_{lm} = 0.5 - \left(0.5 \prod_{m' \in M(l) \backslash m} (1 - 2P^{(i-1)}_{m'l}) \right), \tag{2.6}$$

where $M(l)\backslash m$ denotes the set of VNs connected to c_l excluding v_m.

IV Update all the VNs, i.e., for $1 \leq m \leq n$, $l \in N(m)$ compute:

$$P^{(i)}_{ml} = \frac{P^m_{\text{CH}} \prod_{l' \in N(m) \backslash l} Q^{(i)}_{l'm}}{\left(P^m_{\text{CH}} \prod_{l' \in N(m) \backslash l} Q^{(i)}_{l'm} \right) + \left((1 - P^m_{\text{CH}}) \prod_{l' \in N(m) \backslash l} (1 - Q^{(i)}_{l'm}) \right)}. \tag{2.7}$$

V For all VNs, i.e., $1 \leq m \leq n$, $l \in N(m)$, compute P^m_{ext} as:

$$P^m_{\text{ext}} = \frac{P^m_{\text{CH}} \prod_{l' \in N(m)} Q^{(i)}_{l'm}}{\left(P^m_{\text{CH}} \prod_{l' \in N(m)} Q^{(i)}_{l'm} \right) + \left((1 - P^m_{\text{CH}}) \prod_{l' \in N(m)} (1 - Q^{(i)}_{l'm}) \right)}. \tag{2.8}$$

Make the hard-decision to obtain the estimated vector, $\hat{\boldsymbol{x}} = (\hat{x}_1, ..., \hat{x}_n)$, where $\hat{x}_m = 1$ if $P^m_{\text{ext}} > 0.5$, and $\hat{x}_m = 0$, otherwise.

VI Terminate decoding if $\hat{\boldsymbol{x}}\boldsymbol{H}^T = 0$ or if i has reached the maximum number of iterations. Otherwise, set $i = i + 1$ and return to step III.

Because of the high hardware-complexity of VNs' and PNs' operations in the probability domain, the SPA is usually implemented in the log-domain where channel probabilities are considered as LLRs. Using the log-domain conversion, VNs calculate the summation of LLR messages and PNs employ $tanh(\cdot)$ processing to compute their outgoing messages [79]. In the MSA, the $tanh(\cdot)$ processing in PNs is approximated to reduce the complexity, usually at the expense of some decoding performance loss, compared to the SPA [8, 38]. To compensate for some of the loss, different improved methods are suggested in the literature (e.g., see [17, 18, 38, 42, 44]).

2.2 Strategies and Challenges of Hardware Implementations of LDPC Decoders

In general, fully parallel and partially parallel architectures are two main strategies for the implementation of LDPC decoders. In the fully parallel strategy, the entire factor graph is implemented in hardware and all VNs and PNs in the graph are updated concurrently. Fully parallel decoders are usually implemented to achieve high-throughput decoding of a certain LDPC code, usually at the cost of high area consumption. This approach is particularly considered for applications with high-speed requirements such as the IEEE 802.3an (10GBASE-T) standard [2]. The partially parallel approach instantiates a portion of the factor graph. Partially parallel decoders employ memory and hardware resource sharing to manage message-passing between different portions of the factor graph. The main benefits of this approach are to minimize the area and/or to offer the flexibility to support different code lengths and code rates in applications such as IEEE 802.16e (WiMAX) [3] and IEEE 802.11n (WiFi) [4]. However, the partially parallel approach usually has a lower throughput compared to the fully parallel approach. The partially parallel approach is also used for the implementation of LDPC decoders with very long code lengths where the fully parallel approach is not feasible today,

such as the LDPC code for the DVB-S2 standard with a code length of 64800 bits [90].

A major challenge in the implementation of LDPC decoders is the complexity of the interconnections between VNs and PNs. The complexity of the interleaver is due to the random-like locations of ones in the code's parity-check matrix. This problem is acute for practical fully parallel decoders (where the code block length is usually long) and results in routing congestion and interconnection problems [13, 24, 25, 28]. The routing congestion causes high area consumption and low logic utilization in the decoder. For instance, with 4-bit precision of probability messages, the 52.5 mm^2 die size of the (1024,512) decoder in [13] has a logic utilization of 50% in its core; the rest of the core area is occupied by wires. In addition to high area consumption, the presence of long physical wires in the interleaver increases the power consumption and limits the maximum achievable clock frequency and thus the throughput of a fully parallel LDPC decoder (see [13, 25, 26, 28]).

To alleviate these problems, different approaches are investigated in the literature at both the code design and the hardware implementation levels. One approach is to design "implementation-aware" codes. In this approach, instead of randomly choosing the locations of ones in the parity-check matrix (at the code design stage), the parity-check matrix of an LDPC code is designed with constraints allowing for a suitable structure for a decoder implementation and providing acceptable decoding performance [14, 17, 60, 61, 102, 106].

Another approach used to alleviate the routing congestion problem is to use bit-serial or digit-serial architectures to implement LDPC decoders. Examples of this approach are the FPGA implementation of a bit-serial (480,355) LDPC decoder in [27] and the ASIC implementation of a (660,480) LDPC decoder in [28] based on the bit-serial approximate MSA, and also the MSA-based

bit-serial (256,128) LDPC decoder in [15]. Additionally, a message broadcasting technique was suggested in [25] to alleviate the routing congestion by reducing node-to-node communication complexity in LDPC decoders.

Bit-flipping decoding [34] is another approach for low-complexity LDPC decoding at the cost of some performance loss. Bit-flipping methods do not exploit message-passing, they use the knowledge of unsatisfied parity-checks to iteratively correct bit errors. Recently, there has been research interest in various bit-flipping methods such as weighted bit-flipping methods (see [105] and [48]) and a differential binary decoding method based on bit-flipping [63]. Among conventional bit-flipping methods, the weighted bit-flipping method in [48] performs well on many LDPC codes and has a performance loss of about 0.5 to 1 dB, compared to the SPA [48].

The split-row technique [65] is another approach to alleviate the routing congestion problem in LDPC decoders. In the MSA-based split-row technique, the global minimum operation in PNs is partitioned into localized minimum operations. Therefore, the parity-check matrix of an LDPC code is partitioned into multiple blocks which require local routing. However, in the split-row technique, increasing the number of splits/partitions results in decoding performance loss and, possibly, a higher error-floor [65–67]. This is because in this technique, each PN is divided into lower degree PNs (assigned to each block). These lower degree PNs calculate the minimum of only a portion of incoming messages. Therefore, their outputs are not necessarily the global minimum of all incoming messages received by the PN. As the number of splits increases, the approximation made in lower degree PNs becomes less accurate. Recently, MSA-based threshold decoding methods have been proposed for the split-row technique to reduce this performance loss (e.g., see [67]).

LDPC decoders can be implemented with a programmable architecture or processor, which lend themselves to a software-defined radio. Software-defined radio is a programmable hardware platform that consists of multiple processing and memory units. It supports software implementations of wireless communication protocols for physical layers. Software-defined radio offers the flexibility to support codes with different block lengths and rates; however, the throughput of LDPC decoders that are implemented using software-defined radio is usually low (e.g., see [80]).

In addition to digital decoders, continuous-time analog implementations have been considered for LDPC codes [41] and other error-control codes [11, 35, 55, 64, 91, 96]. Compared to their digital counterparts, analog decoders offer improvements in speed and/or power. However, because of the complex and technology-dependent design process, the analog approach has only been considered for short error-correcting codes. The only reported analog LDPC decoder decodes a (32,8) LDPC code [41].

2.3 Stochastic Computation

Stochastic computation was introduced in the 1960's [32]. A significant motivation for considering stochastic computation was the possibility to perform complex computations using only simple circuitry [16, 32]. In stochastic computation, probabilities are represented as streams of random bits using Bernoulli sequences in which the information is contained in the statistics of the bit stream. Using this representation, complex operations on probabilities such as multiplication and division are converted to operations on bits which can easily be manipulated using simple stochastic gates. This allows high clock rates for the stochastic computational elements while requiring a low computation hardware area. In addition, as a result of the bit-serial nature of stochastic computation, communication between computational elements

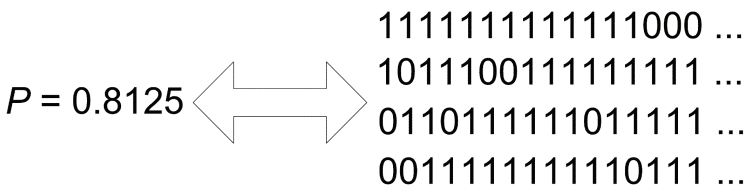


Figure 2–3: Some possible streams for a probability of 0.8125.

requires only one wire per signal. Also, stochastic computation can trade off computation accuracy and time without any change in hardware [16]. Stochastic computation has been considered for different applications such as a field programmable computer in [7], the implementation of artificial neural networks in [16], and a real-time motor controller in [29]. Error-control coding is also a recent application for stochastic computation (see Section 2.4).

2.3.1 Stochastic Representation

In stochastic computation, probabilities are converted to streams of bits called Bernoulli sequences [32]. In this transformation, each bit in a stochastic stream is equal to 1 with the probability represented/convereted. Therefore, the observation of 1's in a stream of bits, $\{a(t)\}$, determines the probability, i.e., $\Pr(a(t) = 1) = P$. The transformation of a probability to a stochastic stream is not unique; therefore, different stochastic streams are possible for a given probability. This also implies that the order/sequence of 1's in a stochastic stream is not important. For example, Figure 2–3 shows some possible streams for a probability of 0.8125. In each stream, 13 bits out of 16 bits are 1, which represents a probability of $13/16 = 0.8125$.

The comparator shown in Figure 2–4 can be used to convert probabilities to stochastic streams [29, 32]. In this figure, P and R are W-bit wide inputs in $[0, 1]$ range. P is fixed and R is a (pseudo) random number with uniform distribution which varies in every clock cycle. In every clock cycle, the output bit of the comparator is 1 if $P > R$, and it is 0, otherwise. Therefore, the

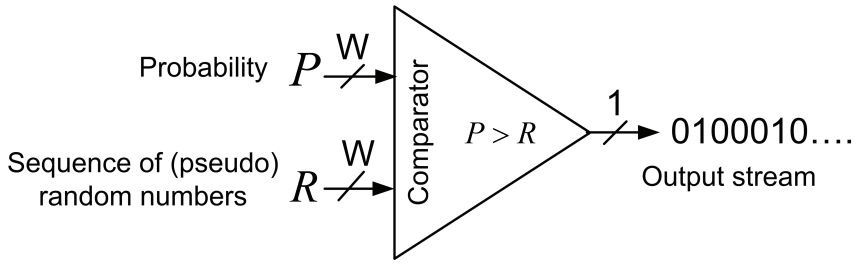


Figure 2–4: Probability to stochastic stream conversion.

likelihood of observing 1 in the output stream is equal to P, representing a probability of P.

2.3.2 Main Stochastic Operations

Using stochastic representation, operations such as multiplication and division on probabilities can be performed using simple hardware structures. It should be noted that stochastic operations discussed in this section are held under the assumption that input stochastic streams are Bernoulli sequences, meaning that the probability of a given bit being equal to 1 is independent of the values of any previous bits. Also, in the case of stochastic operations with multiple input streams, it is assumed that the input streams are uncorrelated with each other.

2.3.2.1 Inversion

Let $P_a = \Pr(a(t) = 1)$ be the input stream of an inverter. The output bit of the inverter, $c(t)$, is 1 when $a(t) = 0$, and it is 0 when $a(t) = 1$. Therefore, $P_c = \Pr(c(t) = 1) = 1 - P_a$ [32].

2.3.2.2 Multiplication

Consider the AND gate shown in Figure 2–5 and its input stochastic streams with $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$ probabilities. The output bit, $c(t)$, is 1 when $a(t) = 1$ and $b(t) = 1$. Therefore, $P_c = \Pr(c(t) = 1) = P_a \times P_b$. Similarly, other boolean operations (such as NAND, XOR, etc.) can be used to implement different operations on probabilities.

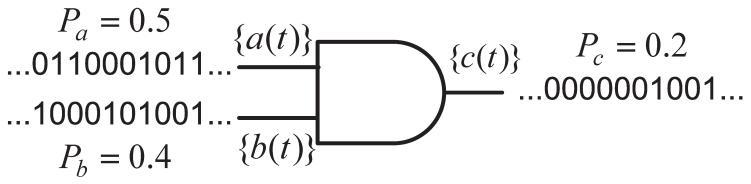
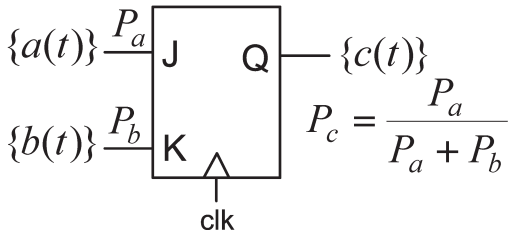


Figure 2–5: Stochastic multiplication [32].



$a(t)$	$b(t)$	$c(t)$
1	0	1
0	1	0
0	0	Hold
1	1	Reverse

Figure 2–6: Stochastic division [32].

2.3.2.3 Division

Consider the JK flip-flop shown in Figure 2–6. This JK flip-flop can be represented as a Markov chain with two states (0 state and 1 state). The probability (transition) matrix of this chain is:

$$\boldsymbol{T} = \begin{bmatrix} 1 - P_a & P_a \\ P_b & 1 - P_b \end{bmatrix},$$

where t_{ij}, the entry at row i and column j in $\boldsymbol{T}$, is the probability of transition from state i to j ($i, j \in \{0, 1\}$). The probability of observing 1 in the steady state of the Markov chain, i.e., $P_c = \Pr(c(t) = 1)$, is obtained based on the eigenvector of $\boldsymbol{T}$ with respect to an eigenvalue of 1. This probability is equal to

$$P_c = \frac{P_a}{P_a + P_b}. \tag{2.9}$$

The operation of (2.9) is an approximation to P_a/P_b, if $P_a << P_b$. Other stochastic division methods exist with more precision [32]. However, as will be discussed in Section 2.4, (2.9) matches the VN operation in the SPA.

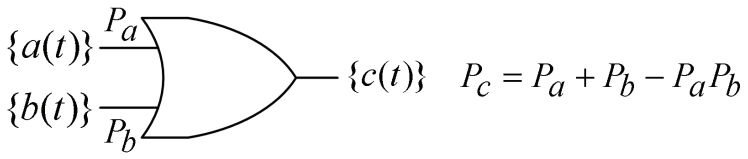


Figure 2–7: Approximate stochastic addition [32].

2.3.2.4 Addition

Consider the OR gate shown in Figure 2–7 and its input stochastic bit streams with $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$ probabilities. The output bit stream of the OR gate represents $P_c = \Pr(c(t) = 1) = P_a + P_b - P_a P_b$. This OR gate can be used as an approximate adder. The approximation made in the OR gate becomes accurate when $P_a P_b$ is small [32].

In general, stochastic addition and substraction are not as straightforward as multiplication and division. This is because they are not closed operations on the probability interval of $[0, 1]$. Therefore, these operations should be combined with a scaling operation to ensure the range of $[0, 1]$ for the outcome [16]. Addition with scaling is performed as $P_c = \sum_{j=1}^{N} S_j P_j$, where $P_j = \Pr(a_j(t) = 1)$ and S_j is the probability of selecting the j-th input bit stream, $\{a_j(t)\}$, such that $\sum_{j=1}^{N} S_j = 1$. The outcome is the scaled sum of the input probabilities. This operation can be implemented in hardware using a multiplexer as shown in Figure 2–8, where RSS refers to the random selection signal supplied by (pseudo) random number generators. Generating RSS is straightforward when the N is a power of two. In a case where N is not a power of two, it is possible to increase N by padding 0 signals to input at the cost of sub-optimality of calculation [16].

2.4 Early Stochastic Decoding Methods

Error-control coding is a recent application of stochastic computation. The idea to use stochastic computation in the SPA-based iterative decoding is first proposed in [36, 74]. In this decoding approach, probabilities received

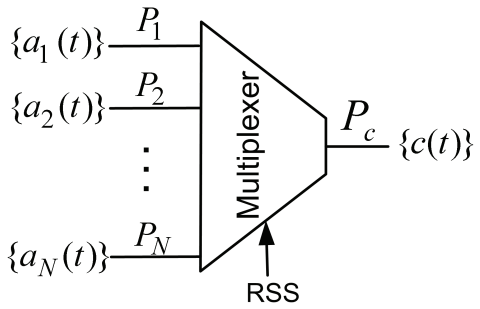


Figure 2–8: Scaled stochastic addition [16].

from the communication channel are converted to streams of stochastic bits. In every stochastic decoding iteration, one bit of each stream is generated and passed to the corresponding VN. Iterative stochastic decoding proceeds by stochastic VNs and PNs exchanging bits until a maximum number of stochastic iterations has been exhausted.

The stochastic representation of probabilities in the code factor graph results in low hardware-complexity bit-serial PNs and VNs. In addition, stochastic computation reduces the routing congestion problem, because only one bit (per direction) is required to represent an edge between a PN and a VN. This implies that in a stochastic decoding iteration (which is usually equal to one clock cycle), decoding proceeds by the VNs and PNs exchanging a bit (per direction) along each edge in the factor graph. The term *Decoding Cycle* or DC is used in the stochastic decoding literature to refer to a stochastic decoding iteration (i.e., the exchange of one bit between stochastic VNs and PNs), and to highlight that a stochastic decoding iteration (or decoding cycle) does not directly correspond to one iteration in the SPA. We stress that in this dissertation, it is assumed that each decoding cycle takes one clock cycle.

This section reviews stochastic decoding methods proposed prior to the work presented in this dissertation (i.e., [36,37,74,95,97]). These early methods were only successful for decoding either short/acyclic codes or some specific

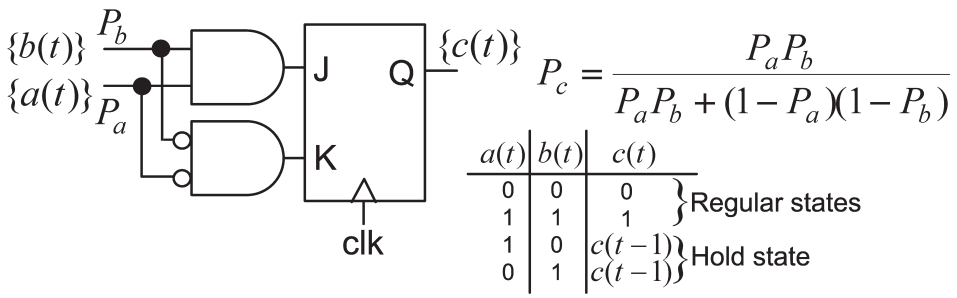


Figure 2–9: The structure of a basic stochastic variable node [36].

error-correcting codes on trellis graphs. They resulted in significant decoding performance loss when used for the decoding of state-of-the-art capacity-approaching LDPC codes on factor graphs.

2.4.1 Basic Stochastic Variable Node

The basic stochastic VN was first proposed in [36]. Let $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$ be the probabilities of two input bit streams, $\{a(t)\}$ and $\{b(t)\}$, in a stochastic VN. In the SPA, the outgoing probability is computed as:

$$P_c = \frac{P_a P_b}{P_a P_b + (1 - P_a)(1 - P_b)}. \tag{2.10}$$

Figure 2–9 shows the stochastic VN's hardware structure proposed in [36] to perform (2.10). It is important to note that in Figure 2–9, the JK flip-flop forces the VN to be in the *hold state* (i.e., $c(t) = c(t-1)$), when the two input bits are not equal ($a(t) \neq b(t)$).

2.4.2 Stochastic Parity-Check Node

The stochastic PN was first proposed in [36]. Let $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$ be the probability of two input bit streams, $\{a(t)\}$ and $\{b(t)\}$, in a stochastic PN. In the SPA, the outgoing probability P_c is computed as:

$$P_c = 0.5 - 0.5(1 - 2P_a)(1 - 2P_b) = P_a(1 - P_b) + P_b(1 - P_a). \tag{2.11}$$

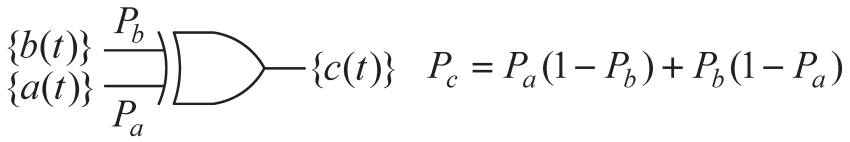


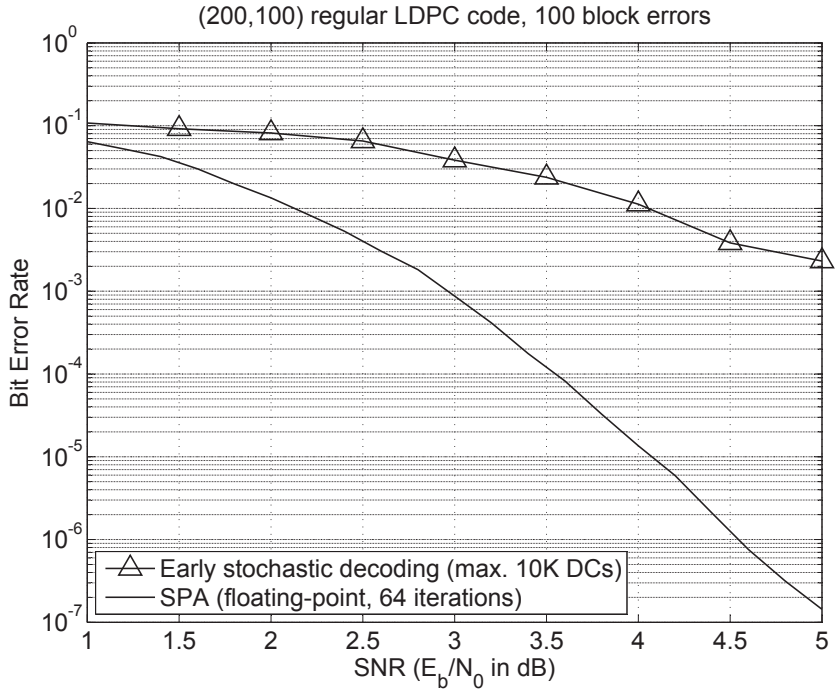
Figure 2–10: The structure of a stochastic parity-check node [36].

Figure 2–10 shows the stochastic PN's hardware structure proposed in [36] to perform (2.11).

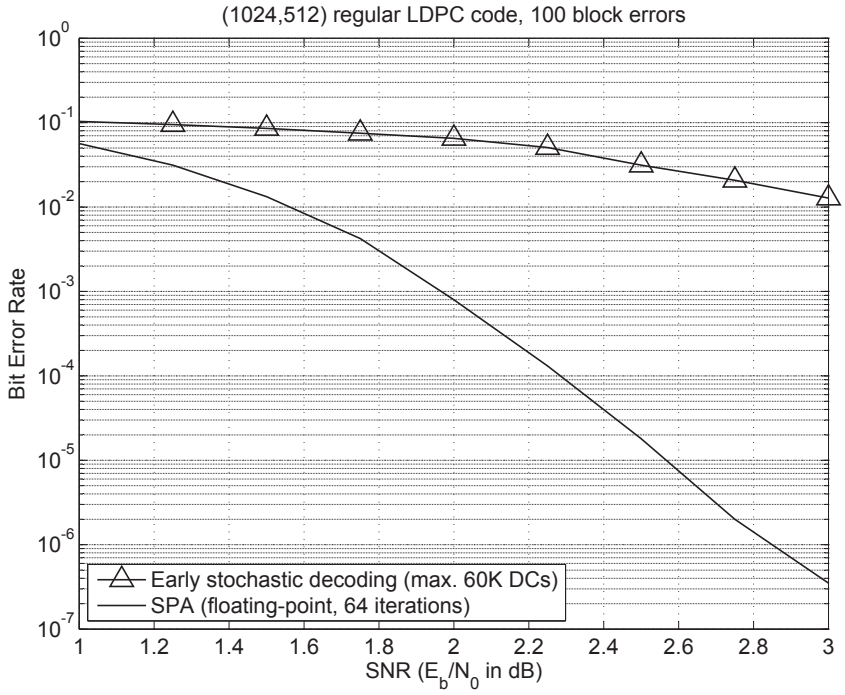
The VN and PN structures shown in Figures 2–9 and 2–10 are used in [74] and [36] for decoding a (7,4) Hamming and a (16,8) LDPC code, respectively. In [36], the decoder had about 0.15 dB decoding performance loss at a Bit Error Rate (BER) of 10^{-4} with respect to the SPA decoding. However, because of severe decoding performance loss, this method cannot be directly used to decode long (practical) LDPC codes. Figures 2–11(a) and (b), respectively, show the decoding performance of this method when applied to a (200,100) regular LDPC code and a (1024,512) regular LDPC code in a BPSK transmission over an AWGN channel. In these figures, E_b is the energy per bit and N_0 is the single-sided noise power spectral density of the AWGN channel. As shown, even by using several thousands of decoding cycles the performance loss is severe compared to the SPA.

2.4.3 The Latching Problem

As mentioned before, stochastic operations rely on the assumption that input bit streams are uncorrelated Bernoulli sequences. This assumption does not hold in graphs with cycles. In addition, a major difficulty observed in stochastic decoding is the sensitivity to the level of random switching activity (bit transition) for a proper decoding operation [95]. The problem of *latching* is described in [82,97] for stochastic decoding on graphs with cycles. The latching problem refers to the case in which a cycle in the code graph causes a group of nodes to lock into a fixed state which is solely maintained by the correlated bit



(a)



(b)

Figure 2–11: Decoding performance of the early scholastic decoding method for decoding (a) a (200,100) LDPC code and (b) a (1024,512) LDPC code.

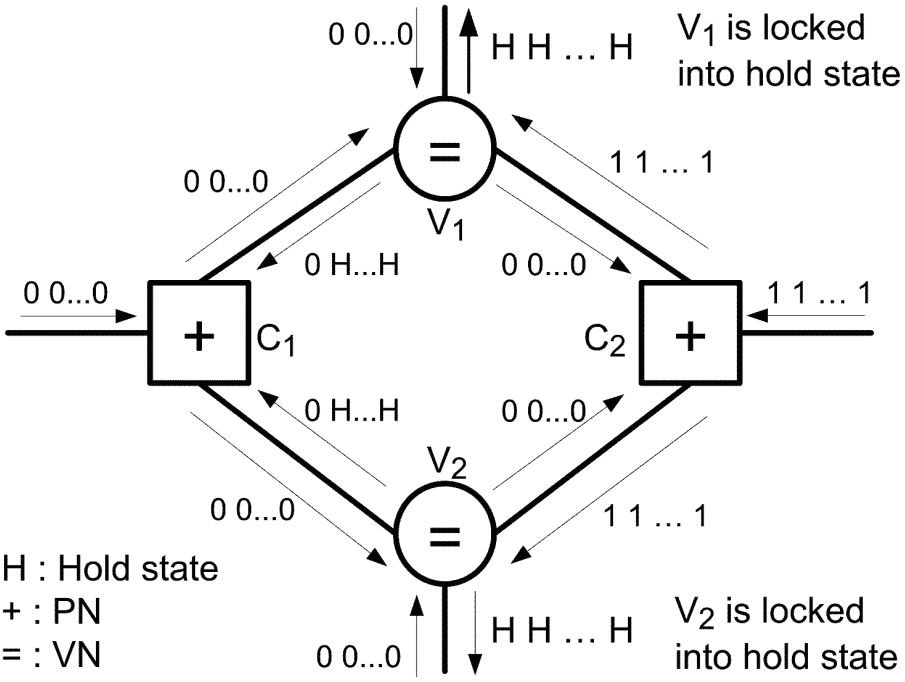


Figure 2–12: An example of latching within a length-4 cycle in a factor graph.

streams within the cycle [82,97]. A simple latching example is the all-zero state in a factor graph. If all internal messages between VN and PN are zero, then they will be held permanently at zero, regardless of any activity from the VNs. This condition occurs because of the JK flip-flop memory used in a stochastic VN (see Figure 2–9). Figure 2–12 shows another example of the latching problem. It illustrates how the lack of switching activity within a length-4 cycle can cause VNs to lock into a fixed state ("hold" state in this example) for several decoding cycles. The latching problem is particulary acute in LDPC decoders, whose corresponding codes' factor graphs have many cycles, and causes severe decoding performance loss [82]. Note that latching can be worse at high Signal-to-Noise-Ratios (SNRs) in which channel probabilities approach zero (or one). In this case, bits in stochastic sequences are mostly 0 (or 1), hence, random switching events become too rare for proper decoding [95].

2.4.4 Supernodes

The idea of *supernodes* was proposed in [97]. Supernodes are special structures which reduce the latching problem by regenerating new messages based on the probabilities of incoming stochastic messages. A supernode tabulates incoming messages in histograms to estimate their probabilities and regenerates new stochastic messages. In [97], supernodes are used as a special VN which can be placed in critical parts of the graph (e.g., where short cycles exist). Figure 2–13(a) shows the structure of supernodes used in [97] for trellis decoding of a (256,121) product Turbo code constructed based on acyclic (16,11) Hamming component decoders. Supernodes in this trellis decoder [97] are used instead of VNs shown in Figure 2–9. These supernodes were "packetized" in a sense that they are using the conventional SPA calculation (i.e., (2.7)) after a time-step to calculate the probabilities of the new outgoing probability messages and regenerate new stochastic streams. Figure 2–13(b) shows another structure of supernodes suggested in [97]. In this structure, the input messages are fed directly to a counter to tally the number of ones for a given number of samples. This count is then used to generate new probabilities. This structure is used in [37] for hardware implementation of a decoder that decodes a specially-constructed tail-biting (16,8) LDPC code with an acyclic factor graph. Supernodes in this implementation were placed between VNs and PNs of the decoder.

2.4.5 Scaling Channel Reliabilities

Scaling methods have been previously suggested in the literature for performance improvement of the SPA (e.g., see [100] for details). As mentioned before, the latching problem can be worse at high SNRs due to the lack of switching activity. In stochastic decoding, scaling channel reliability is used in order to increase the switching activity in the decoder. This idea is first

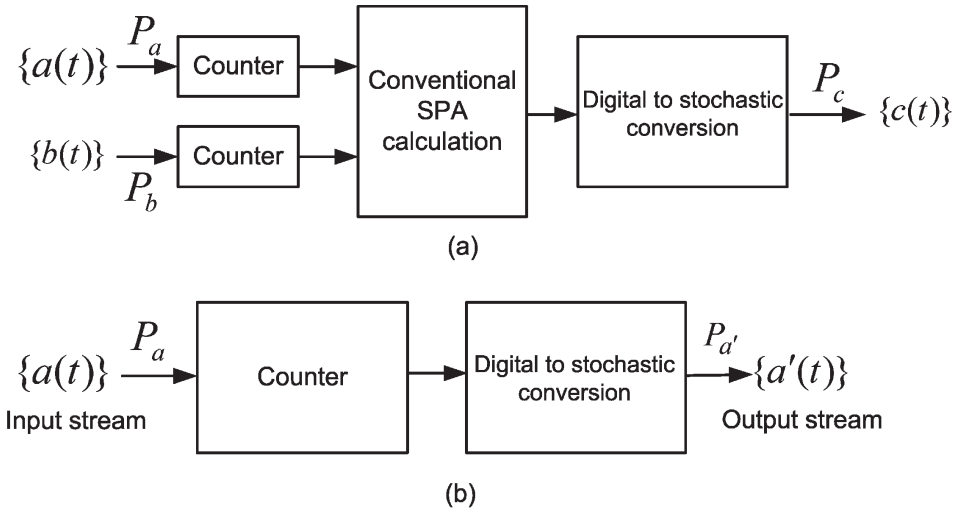


Figure 2–13: Structure of supernodes used (a) in [97] and (b) in [37].

suggested in [95] and used in stochastic decoding of a (16,11) Hamming code. In this method, every block of channel reliabilities received is scaled to a maximum value to ensure the same level of switching activity for each block.

The above-mentioned early stochastic methods were applied for decoding some short Hamming and LDPC codes, a specially-constructed tail-biting (16,8) LDPC code with an acyclic factor graph, and trellis decoding a (256,121) product Turbo code constructed based on acyclic (16,11) Hamming component decoders. As a result of their significant decoding performance loss and/or high error-floors compared to conventional LDPC decoding algorithms, these methods were not practical solutions for decoding state-of-the-art LDPC codes on factor graphs. The next chapter proposes the first successful stochastic LDPC decoding approach.

CHAPTER 3

Edge-Based Rerandomization Using Edge Memories

In this chapter, we propose the rerandomization of stochastic bit streams using EMs and a new method for scaling channel reliabilities to address the latching problem in stochastic LDPC decoders. We discuss the architecture of high-degree stochastic VNs and propose Internal Memories (IMs) to improve the decoding performance of stochastic LDPC decoders. We also discuss the hardware architecture and implementations of EM-based stochastic LDPC decoders and provide comparison with practical LDPC decoding approaches.

3.1 Edge Memories and Regenerative Bits

EMs are memory-based rerandomization units that are assigned to edges in the factor graph. They replace the JK flip-flop used in the basic stochastic VN structure (shown in Figure 2–9). The principal function of EMs is to disrupt correlations among the stochastic streams within cycles by rerandomization. This is effectively accomplished by time-interleaving the output stream of stochastic VNs. In this respect, stochastic bits generated by a VN are categorized into two groups: *regenerative* bits and *conservative* bits. Conservative bits are output bits which are produced in the hold state and regenerative bits are output bits which are produced in nonhold (regular) states. The essentials of the operation of EMs are twofold:

i EMs are only updated with the regenerative bits. Therefore, when a VN is not in the hold state, the newly produced regenerative bit is used as the outgoing bit of the edge and the EM is updated with this new bit. When the VN is in the hold state for an edge,[1] a bit is randomly chosen from the corresponding EM and is used as the outgoing bit. This mechanism breaks the correlation of stochastic streams by rerandomizing stochastic bits and also reducing the correlation caused by the hold state in a stochastic stream. The reason is that every time the hold state happens, a bit is randomly chosen from previous regenerative bits (which are not generated in the hold state).

ii In order to facilitate the convergence of the decoder, EMs need to have a time-decaying reliance (forgetting mechanism) on previous regenerative bits and only rely on the most recent regenerative bits.

Figure 3–1 shows the structure of an EM. An EM is implemented as an M-bit shift register with a single selectable bit. In this implementation, the shift register is updated only when U = 1, which indicates that the VN output bit, $r(t)$, is regenerative. In the case that the VN is in the hold state (U = 0), a bit is (pseudo) randomly chosen from the shift register using a (pseudo) randomly generated address, $R(t)$. Clearly, the length of the shift register, M, guarantees the time-decaying reliance mechanism needed for an EM. As will be shown in this chapter, the shift register-based architecture of EMs results

[1] We recall that, in general, a stochastic VN is in the hold state for edge e, when the input bits of the VN (excluding the input bit received from edge e) are not equal. For a VN that is constructed based on lower degree subnodes (discussed in Section 3.2), the VN is called in the hold state for edge e, when the input bits of the final/exiting subnode for edge e are not equal.

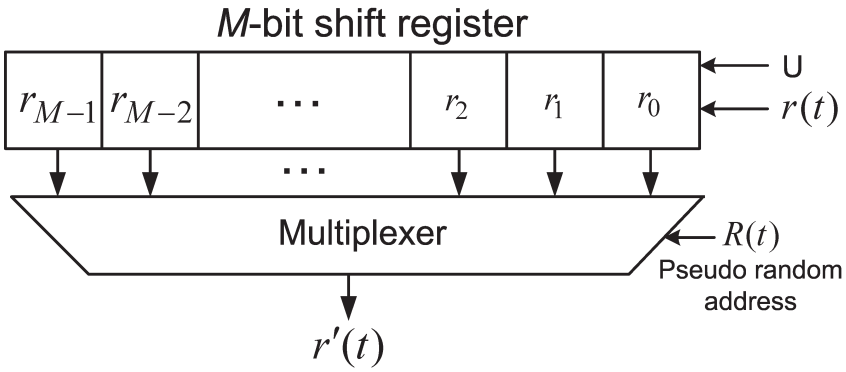


Figure 3–1: An EM is implemented as an M-bit shift register with a single selectable bit.

in their resource-efficient FPGA implementation using shift register look-up tables.

3.2 High-Degree Stochastic Nodes and Internal Memories

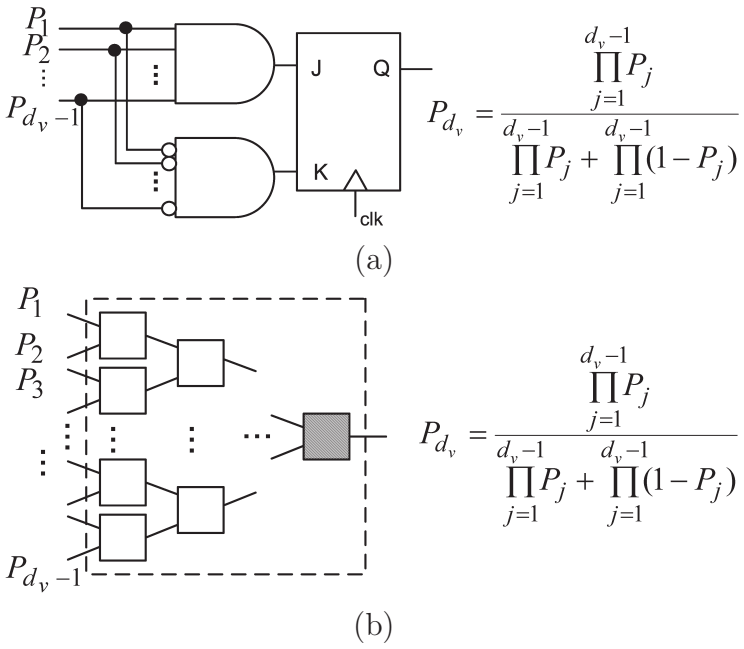
Regenerative bits are important for the proper operation of a stochastic decoder. The lack of enough regenerative bits propagating in the decoder results in low switching activity and a high possibility of latching. A frequent occurrence of the hold state and latching in VNs can severely affect the decoding performance. In this respect, the structure used for VNs is crucial for the decoding performance of stochastic LDPC decoders.

In general, a high-degree node can be constructed based on subgraphs of low-degree subnodes [36,50]. We show that for the case of stochastic decoding, high-degree VNs should be constructed based on low-degree subnodes (usually, $d_v \leq 4$ subnodes can be used). To elaborate, consider the structure of the stochastic VN in Figure 3–2(a) with an arbitrary d_v. This stochastic structure is not suitable for high-degree VNs because it is entirely in the hold state when any two input bits are not equal; therefore, the chance of being in a hold state increases as d_v increases. An increased chance of hold state for VNs reduces the propagating of regenerative bits in the graph and results in less switching activity within the graph, and thus degraded stochastic decoding

performance. Note that this phenomenon can be very destructive when bits in input stochastic streams of VNs are mostly 0 (or 1), for instance at high SNRs where corresponding probability messages are either close to 0 (or 1), or during the convergence to the right codeword where most PNs are satisfied and only a few PNs remained unsatisfied. We propose that these problems can be significantly alleviated by using the VN structure shown in Figure 3–2(b) (only one output and corresponding inputs are shown). In this figure, a high-degree stochastic VN is constructed based on subgraphs of low-degree subnodes with memory. In this structure, by having stochastic input bits which are either mostly 0 (or 1), the chance of a hold state for the (highlighted) exit subnode is much lower. Therefore, the entire node is less likely to be in the hold state. Note that in Figure 3–2(b), an EM is used only for the exit output edge. To show the difference between the two structures of Figures 3–2(a) and (b), Figure 3–3 compares the averaged percentage of hold state occurrence for $d_v = 9$ with P_1 varying and $P_2 = P_3 = ... = P_8 = 0.9$. As shown, the percentage of the hold state decreases significantly when the VN is constructed based on subgraphs of low-degree subnodes.

We also propose the use of IMs for each subnode in high-degree VNs to further decrease the chance of being in the hold state in a high-degree VN. This structure is shown in Figure 3–4, where each IM is assigned to one subnode. The operation of IMs is similar to EMs, but the IM length, L, is much shorter than the EM length, M (it is only a few bits). An IM is updated with regenerative bits produced by the subnode and in the case of the hold state for a sub-node a bit is randomly chosen as the outgoing bit of the subnode.

A straightforward way to implement all the outputs of a VN, is to have d_v instances of the structure shown in Fig 3–2(b). Using this method the entire VN requires $d_v(d_v - 1)$ subnodes. We propose that the complexity of a



$$P_{d_v} = \frac{\prod_{j=1}^{d_v-1} P_j}{\prod_{j=1}^{d_v-1} P_j + \prod_{j=1}^{d_v-1} (1-P_j)}$$

Figure 3–2: (a) A structure which is not suitable for high-degree VNs. (b) An example of constructing a high-degree VN based on low-degree subnodes.

high-degree stochastic VN can be significantly reduced by efficient sharing of subnodes. Figure 3–5 shows the efficient implementation for a degree-16 VN ($d_v = 16$). VNs with arbitrary degrees can be similarly implemented. Based on properties of binary trees, it can be shown that using this structure a degree $d_v > 2$ VN can be efficiently implemented by $3d_v - 6$ subnodes.

The construction of high-degree stochastic PNs is based on XORing input bits. It can be shown than a degree d_c PN requires $2d_c - 1$ two-input binary XORs to compute all the outputs. Figure 3–6 shows the structure of a degree-16 stochastic PN ($d_c = 16$).

3.3 Scaling Channel Reliabilities

Scaling methods have been previously suggested in the literature for performance improvement of iterative decoders (e.g., see [31,100,101] for details). For stochastic decoders, we use a new scaling method. This method is employed to provide a similar level of switching activity over different ranges of SNRs, which results in improved BER decoding performance for stochastic

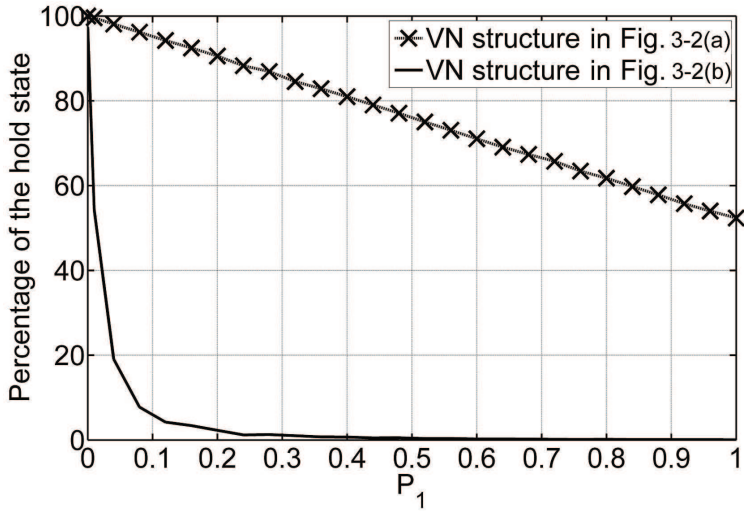


Figure 3–3: (a) Percentage of holds on the output of two $d_v = 9$ VNs based on structures in Figure 3–2(a) and Figure 3–2(b), P_1 is varying and $P_2 = ... = P_8 = 0.9$.

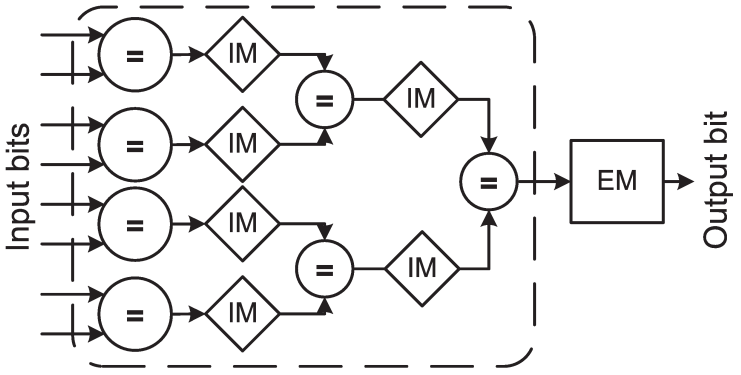


Figure 3–4: The construction of a VN based on IMs for low-degree sub-VNs. An EM is only used for the exit edge.

decoders. In this scaling method, received channel reliabilities are scaled by a factor that is proportional to the noise level in the channel. The scaled reliabilities are, however, independent of channel noise and thus the decoder does not need to estimate the noise in the AWGN channel.

Assume that $L_{\mathrm{CH}}^{m} = \frac{4y_m}{N_0}$ is the channel LLR for y_m, the m-th symbol in the received vector, in a BPSK transmission over an AWGN channel as defined in (2.5). The scaled LLR, $L_{\mathrm{CH}}^{\prime m}$, is computed as:

$$L_{\mathrm{CH}}^{\prime m} = (\gamma N_0) L_{\mathrm{CH}}^{m} = 4\gamma y_m, \tag{3.1}$$

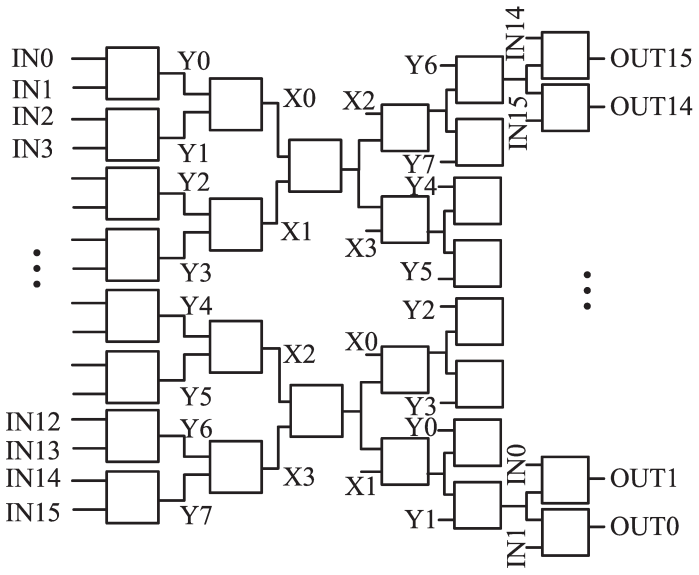


Figure 3–5: The proposed low complexity structure for the implementation of a degree-16 stochastic VN.

where γ is a fixed empirical factor whose value is chosen based on the best BER performance of the stochastic decoder. The input channel probabilities in stochastic decoding is computed based on the scaled LLRs as:

$$P_{\text{CH}}^{m} = \frac{\exp(L_{\text{CH}}^{\prime m})}{\exp(L_{\text{CH}}^{\prime m}) + 1}. \tag{3.2}$$

3.4 Effects on the Decoding Performance

Figures 3–7 (a) and (b) show the BER performance of the EM stochastic decoding approach for a (200,100) and a (1024,512) regular LDPC code with degree-3 VNs and degree-6 PNs. Note that in these figures, the maximum number of decoding cycles used for stochastic decoding is set to a high number to show that the significant performance loss (for the case in which EMs and scaling are not used) is not improved by increasing the decoding latency. As will be shown in the rest of this dissertation, the maximum number of decoding cycles used by stochastic decoding (to provide a competitive performance with state-of-the-art LDPC decoders) is less than the values used in these figures by about two orders of magnitudes. We used $M = 25$ and maximum 10K

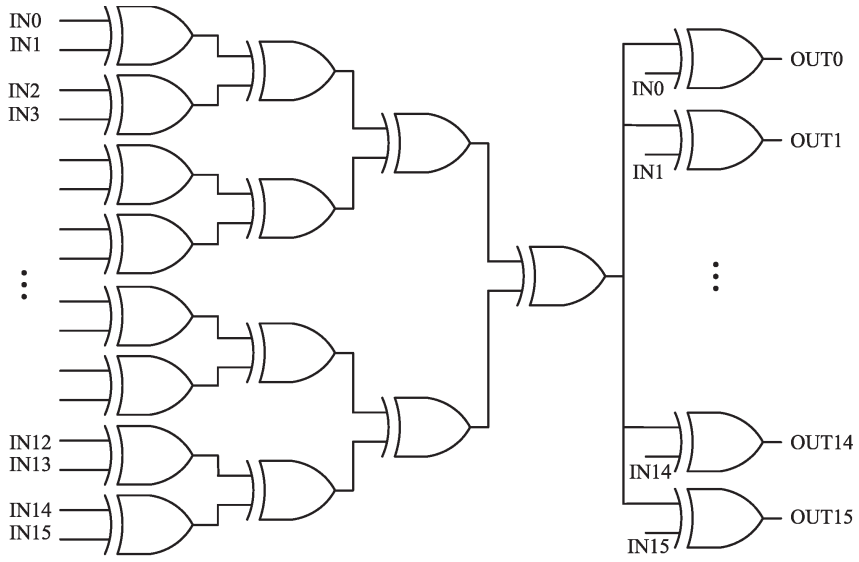
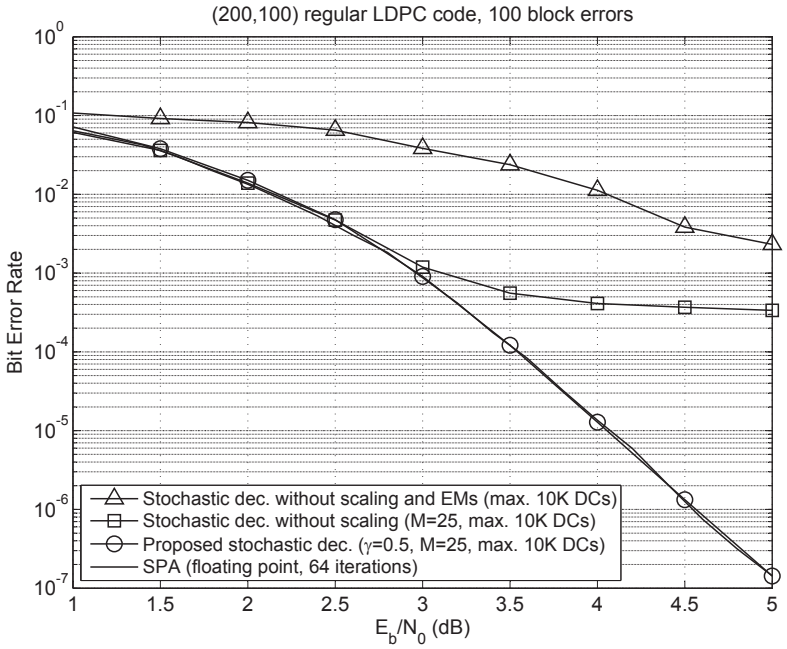


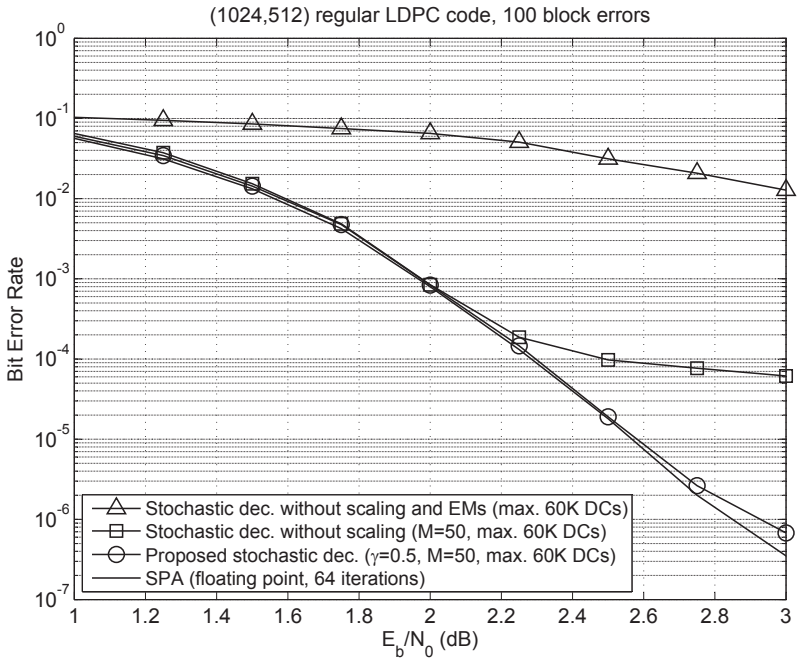
Figure 3–6: The structure of a degree-16 stochastic PN based on 2-input binary XORs.

decoding cycles for decoding the (200,100) LDPC code and, $M = 50$ and maximum 60K decoding cycles for the (1024,512) code. A $\gamma = 0.5$ is used for both codes. As shown, with respect to the floating-point SPA with 64 iterations,[2] the proposed method provides comparable BER performance for the (200,100) code and near-optimal performance for the (1024,512) code. An SNR loss of less than 0.1 dB is observed for the latter code at the BER of 10^{-6}. To show the performance contribution of scaling and EMs, results for (i) decoding without scaling and EMs and, (ii) decoding with EMs but without scaling are also depicted in Figures 3–7 (a) and (b). The contribution of EMs can be observed by comparing results for case (i) and (ii). Also, the contribution of scaling at higher SNRs can be easily seen by comparing the results of case (ii) and stochastic decoding with EMs and scaling.

[2] No major BER improvement is observed after the 64th iteration.



(a)



(b)

Figure 3–7: Performance of the EM-based approach for decoding (a) a (200,100) code and (b) a (1024,512) code. A high maximum number of decoding cycles is used to show that the significant performance loss (for the case in which EMs and scaling are not used) is not improved by increasing the decoding latency.

3.5 A (1056,528) Fully Parallel EM-based LDPC Decoder

This section presents the hardware architecture of fully parallel EM-based stochastic LDPC decoders in detail. Although the focus of this section is on EMs and FPGA implementation, the proposed architecture and its features form the basis of the structure of ASIC stochastic decoders that are explained in the following chapters.

The stochastic decoding operation can be summarized as follows. Upon receiving a vector from the AWGN channel, channel reliabilities are scaled and then transformed into stochastic streams. Each VN receives one bit per decoding cycle and propagates its outgoing 1-bit messages to the connected PNs. PNs check the parities and send their 1-bit messages to VNs. The output of each VN at the end of a decoding cycle is passed to an up/down counter in which its sign-bit determines the hard-decision. This exchange of bits between VNs and PNs will be stopped as soon as all the parity-checks are satisfied or a maximum number of decoding cycles is exceeded.

Table 3–1 summarizes the characteristics of two LDPC codes considered in this section. Both codes are irregular and belong to the IEEE 802.16e (WiMAX) standard [3]. The code used for implementation is the (1056,528) code. The (1056,704) code is only used to study performance behavior. The reader should note that the main reason to choose these codes was to show the applicability of the stochastic approach to decode state-of-the-art irregular LDPC codes with high-degree nodes designed for recent applications (using the fully parallel design approach). In Section A.1 of Appendix A, we provide the performance results of the EM approach for decoding various LDPC codes.

3.5.1 Decoder Specifications and Architecture

This section discusses specifications and the hardware architecture of the implemented (1056,528) EM-based stochastic LDPC decoder.

Table 3–1: Irregular LDPC codes chosen from the IEEE 802.16e standard.

(n,k)	d_v distribution	d_c distribution
(1056,528)	(2,3,6)=$\{11/24, 1/3, 5/24\}$	(6,7)=$\{2/3, 1/3\}$
(1056,704)	(2,3,4)=$\{7/24, 1/24, 2/3\}$	(10,11)=$\{7/8, 1/8\}$

3.5.1.1 Scaling

We used look-up-tables to apply scaling to the symbols received from the AWGN channel. The input of each look-up-table is a 6-bit received symbol and the output is the corresponding probability, represented in 7 bits. Probabilities in each look-up-table are calculated according to (3.2). Note that because of the symmetry in (3.2), a look-up-table can store only half of the probabilities. For example, it is possible to only store probabilities for positive y_m's (i.e. probabilities ≥ 0.5). When a y_m is negative, an additional NOT operation can be performed on the stochastic stream (during probability to stochastic stream conversion). Using this scheme, the size of each look-up-table is $2^{6-1} \times 7$ bits or 28 bytes. The implemented stochastic decoder employs 44 look-up-tables to apply scaling, and each look-up-table serially generates probabilities for $1056/44 = 24$ VNs. This uses $T_{\text{IO}} = 24$ decoding cycles,[3] where T_{IO} is the number of decoding cycles the decoder spends to input channel reliabilities, to apply scaling, and to output the decoded bits.

3.5.1.2 Probability to Stochastic Stream Conversion

The conversion of each P_{CH}^m to the corresponding stochastic stream is done by employing a 7-bit comparator as shown in Figure 3–8. In this structure, P_{CH}^m is fixed during the decoding operation and is compared to a (pseudo) random number, R, which changes at every decoding cycle. The output bit of the comparator is 1 when $P_{\text{CH}}^m > R$, and it is 0, otherwise. The random number in

[3] We recall and highlight that each decoding cycle takes one clock cycle.

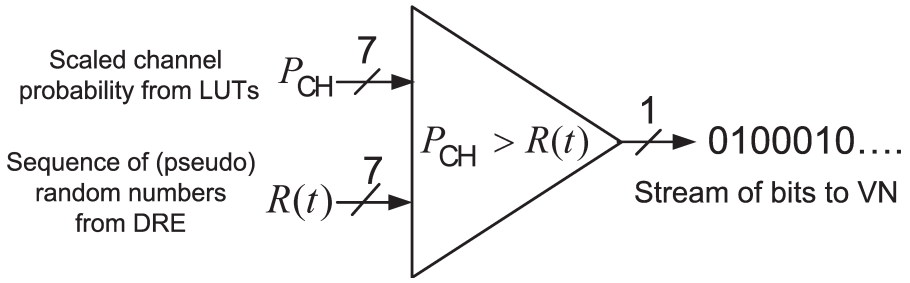


Figure 3–8: Conversion of channel probabilities to stochastic streams.

the figure is generated by a Distributed Randomization Engine (DRE) which is described in Section 3.5.1.7. The output of each comparator is fed to one VN in each decoding cycle. The decoder, hence, needs one comparator per VN.

An attractive advantage of using look-up-tables to apply scaling in stochastic decoders is that the precision of the look-up-tables' output probabilities can be increased without a significant change in the decoder complexity. This is because the precision of probabilities does not affect the interleaver, VNs or PNs. It only affects the size of look-up-tables used for scaling, the comparators and the DRE. This, however, is not the case for SPA-based or MSA-based decoders in which changing the precision means significantly increasing the number of wires in the interleaver. For the case of bit-serial SPA-based or MSA-based decoders, increasing the precision increases the latency of each iteration and hence reduces the throughput, because more clock cycles are needed to bit-serially send messages between nodes.

3.5.1.3 Architecture of Variable Nodes

Figure 3–9 depicts the architecture of VNs in the (1056,528) stochastic decoder (only one output and its corresponding inputs are shown). EM lengths of $M = 32$, $M = 48$, and $M = 64$ bits are used for $d_v = 2$, $d_v = 3$, and $d_v = 6$ VNs, respectively. The architecture of $d_v = 3$ VNs is based on two $d_v = 2$ subnodes. The architecture of $d_v = 6$ VNs is based on two $d_v = 3$ and one

$d_v = 2$ subnodes. IM lengths of $L = 1$ and $L = 2$ bits are used for $d_v = 3$ and $d_v = 6$ VNs, respectively. For FPGA implementations of EMs we used shift register look-up-tables. Many FPGA architectures allow to utilize small look-up tables as shift register look-up tables and to access a single bit in the register (e.g., [98,99]). It is also possible to cascade any number of shift register look-up tables to form a shift register of arbitrary size. These features exactly match the operation of EMs.

The proposed EM-based VN structure has two modes of operation:

- Partial Initialization Mode: Prior to the decoding operation and when the channel probabilities are all loaded into the decoder, VNs start to initialize their EMs according to the received probability. It is possible for EMs to start from zero state, however, the initialization of EMs improves the convergence of the stochastic decoder. In the implemented decoder, we consider partially initializing the EMs to 16 bits. During the partial initialization, the EMs of each VN are bit-serially updated with the output of the channel comparator for $T_{\mathrm{LOAD}_{\mathrm{EMs}}} = 16$ decoding cycles.
- Decoding Mode: After the partial initialization phase, the decoding operation starts. Each VN in the Figure 3–9, uses a signal U to determine if the VN is in the hold state (U = 0) or not (U = 1). When the VN is not in the hold state, the new regenerative bit is used as the output bit and also to update the EM. In the case of the hold state, a bit is randomly chosen from the EM. This scheme is also employed in each subnode to update the IMs. The random selection of bits in EMs and IMs are done by (pseudo) random addresses which vary in each decoding cycle. These addresses are also provided by the DRE in Section 3.5.1.7.

Because of the partial initialization scheme used at the beginning of the decoder operation, the range of (pseudo) random addresses is limited to 4 bits (i.e., 0 to 15) for 40 decoding cycles. This ensures that during the hold state, a valid bit is picked from EMs. When decoding proceeds for 40 decoding cycles and EMs are updated, the DRE produces full range addresses for EMs.

3.5.1.4 Hard-Decision using Saturating Up/Down Counters

The output bit of each VN at the end of every decoding cycle is passed to an up/down counter. Each counter is incremented when receiving 1 and decremented when receiving a 0 bit. The counters are implemented as saturating counters which stop incrementing/decrementing when they reach their maximum/minimum limits. For this implementation, we used 4-bit saturating counters that count from -7 to 7. The sign-bit of each counter determines the hard-decision, i.e., in a BPSK transmission a 0 sign-bit of the counters determines a "+1" decoded bit and a 1 sign-bit determines a "-1" decoded bit.

Based on our observation, we discovered that for the case of stochastic LDPC decoders, up/down counters are mostly effective at low SNRs (high BERs). At high SNRs, up/down counters can be neglected and replaced with 1-bit flip-flops. In this case, the last output bit of each VN directly determines the hard-decision. The reason is that at high SNRs where convergence of stochastic decoders is fast, the counters easily become saturated (i.e., high reliability) which implies that they mostly receive constant output bits from VNs. The output bits of VNs at low SNRs are, however, less reliable and are more varying.

3.5.1.5 Architecture of Parity-Check Nodes

The construction of a PN is based on XORing the input bits received from VNs. Figure 3–10 shows the structure of a $d_c = 7$ PN used in the implemented stochastic decoder. The construction of $d_c = 6$ PNs in the decoder is similar.

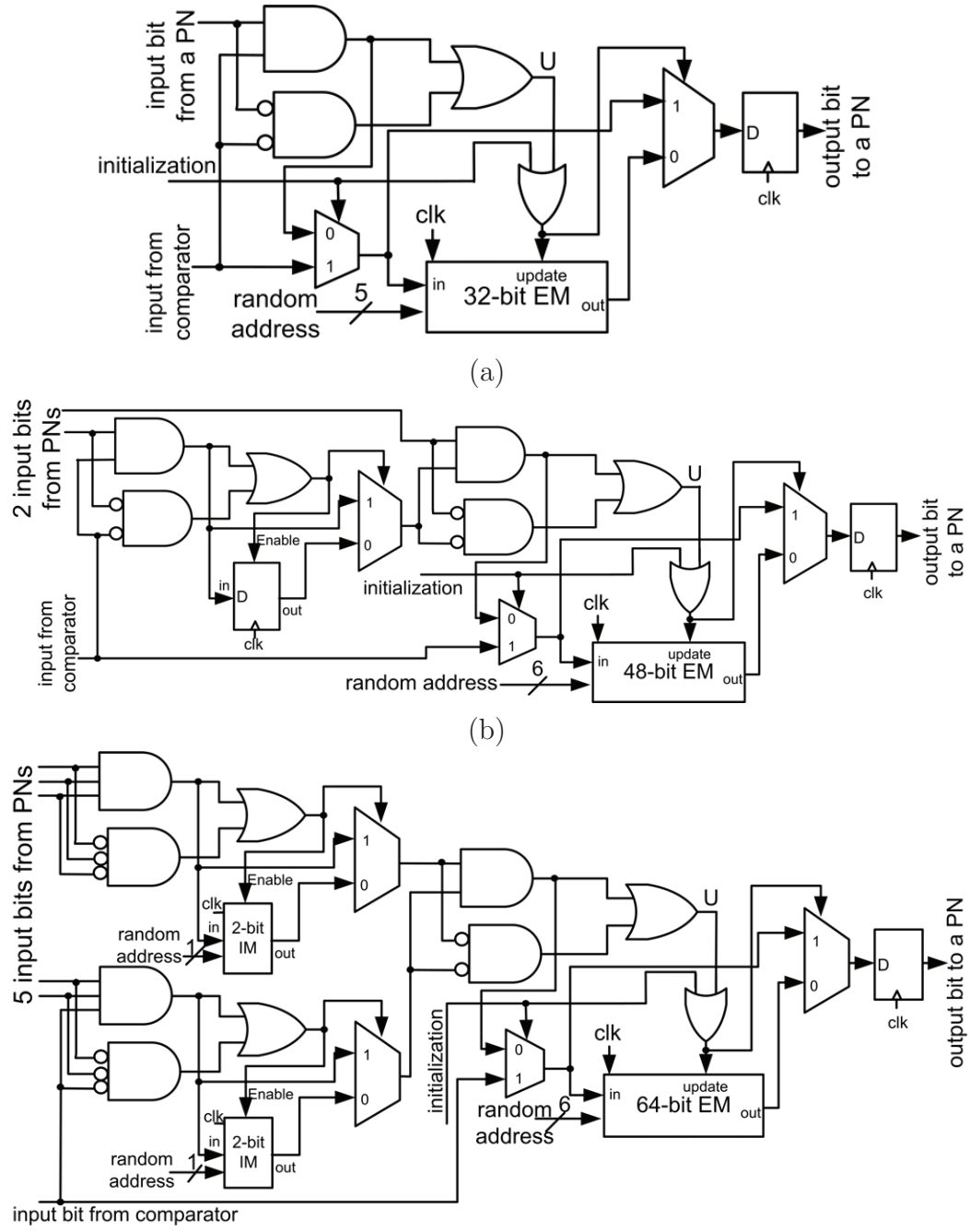


Figure 3–9: Architectures of (a) a degree-2 VN, (b) a degree-3 VN, and (c) a degree-6 VN based on IMs and an EM (in each figure, only one output and its corresponding inputs are shown).

PNs send their output bits to VNs. In addition, each PN produces a "parity-check satisfied" output signal which determines if the corresponding parity-check is satisfied. This signal is used to terminate the decoding process, as will be discussed in Section 3.5.1.8.

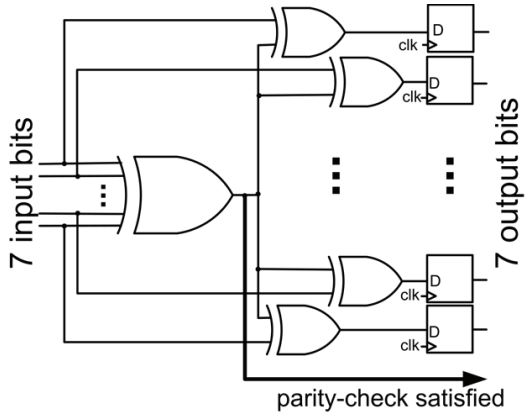


Figure 3–10: Architecture of a degree-7 stochastic PN. The "parity-check satisfied" signal is used for termination criteria.

3.5.1.6 Asynchronous Pipelining and Interleaver Design

As mentioned earlier, the structure of interleavers in LDPC decoders results in (long) wires and forces a bottleneck on the speed and throughput of decoders. For this reason, fully parallel architectures can use pipelining to break the wires and increase the speed/throughput. However, pipelining the interleaver in conventional SPA-based or MSA-based decoders has a drawback: it increases the number of clock cycles required per iteration by a factor of S, the number of pipeline stages. The reason is that in the SPA and the MSA, there is a data dependency between iterations and the output of nodes at each iteration depends on their outputs at the previous iteration. For instance, assume that the non-pipelined decoder runs for I iterations and uses 1 clock cycle per iteration. In the decoder with a pipelined interleaver, S clocks are needed to pass messages generated in the previous iteration. Therefore, although the pipelined decoder is faster than the non-pipelined decoder, it needs $S \times I$ clock cycles to provide the same BER decoding performance. To reduce this inefficiency and to increase the utilization of decoder, the pipelined LDPC decoders need to decode more than one codeword in the pipelined interleaver at the expense of more hardware-complexity [15]. Another suggested

technique in the literature is block interlacing [25]. This technique is used to increase the throughput of the decoder by processing two consecutive blocks simultaneously. In [25], the block interlacing technique together with a message broadcasting technique provided high throughput for ASIC LDPC decoders using 16 iterations of the MSA and 32 iterations of the hard-decision message-passing decoding algorithm [76]. In addition to these methods, the flooding-type update-schedule algorithm is suggested in [70]. This algorithm allows limited partitioning of some of the long wires in the decoder using flip-flops [70] without affecting the required clock cycles per iteration. This relies on the similarity of time-consecutive messages which limitedly let nodes tolerate operating with messages produced in recent iterations. However, in this algorithm the degree of freedom for partitioning wires is limited. In [70], only messages from two consecutive iterations are used at VNs.

We claim that the above-mentioned drawbacks and limitations do not apply to stochastic decoders. The operation of stochastic nodes does not depend on the output bits produced in the previous decoding cycle. In fact, the order of bits in stochastic streams is not important for the nodes. That is why EMs with random bit selection and different lengths can be used at VNs. Therefore, if a stochastic decoder needs to operate for D decoding cycles to decode a codeword, the S-stage pipelined stochastic decoder needs $D + S$ decoding cycles to decode the codeword. This interesting characteristic introduces a high degree of freedom for partitioning wires in stochastic decoders, which is especially advantageous for ASIC implementations:

i In principle, an arbitrary number of pipeline stages can be used in the interleaver to break the wires and increase the clock rate to a "desired" speed.

ii Pipelining in a stochastic decoder does not need to be uniform in the entire factor graph. Different stages of pipelining can be used for different edges. It is also possible to only pipeline some (critical) wires in the interleaver with an arbitrary number of pipeline stages.

It should be noted that because stochastic decoders (and other bit-serial approaches) require fewer wires to represent the factor graph, pipelining the interleaver in stochastic decoders requires fewer hardware resources (registers) compared to the conventional SPA-based or MSA-based decoders. For the implemented stochastic decoder we used a 4-stage pipeline interleaver.

3.5.1.7 Distributed Randomization Engine

The randomization engine is responsible for providing random numbers in the decoder. In the proposed architecture, (pseudo) random numbers are used in comparators and also as the addresses of EMs and IMs. Although this amount of random numbers for the entire decoder might seem high, (pseudo) random numbers can be significantly shared at two levels without having a considerable impact on the decoding performance of the decoder: (i) different EMs can share the same random address and, (ii) random numbers used in comparators and random numbers used as the addresses of EMs and IMs can share bits. Sharing random numbers significantly reduces the complexity of the randomization engine.

We propose a distributed architecture to generate random numbers. The DRE consists of 48 independent randomization engines. Each randomization engine generates the required random numbers for a portion of the factor graph and consists of only two 10-bit Linear Feedback Shift Registers (LFSRs) associated with prime polynomials. Random bits in each randomization engine are generated by XORing different bits of the two LFSRs. The main reason to use a distributed structure is to reduce the routing required by DRE. Note that

by using the asynchronous pipelining technique, the interleaver is no longer a bottleneck for the speed of a stochastic decoder. This is because non-uniform levels of registers can be used to break long wires in a pipelined stochastic interleaver. In this case, the routing required by randomization engines becomes a limiting factor for the speed and hence using a distributed architecture for generating random numbers becomes essential. It should be noted that the asynchronous pipelining technique is also applicable for DRE because the sequence/order of random numbers is not important for comparators, EMs and IMs.

3.5.1.8 Termination Criteria

The stochastic decoder checks two criteria in each decoding cycle to terminate the decoding operation: (i) it checks if all the PNs are satisfied or (ii) if a maximum number of decoding cycles has been exceeded. As soon as one of the criteria is satisfied, the decoder outputs the sign-bit of each saturating up/down counters as the decoded codeword and starts loading the probabilities for the next received block. Checking the first criterion is done by NORing "parity-satisfied" signals from all PNs (i.e., decoding is terminated if all the 528 parity-checks are satisfied). This is implemented as a 3-stage pipelined NOR tree. The latter criterion is checked using a counter.

3.5.1.9 Input/Output Unit

As mentioned previously, the decoder uses $T_{\mathrm{IO}} = 24$ decoding cycles to load 1056 received symbols (each with 6-bit precision) into the decoder and apply scaling. To do so, the decoder employs 264 input pins. While loading the probabilities, the decoder also outputs the previous 1056 bit decoded codeword using 44 pins (in $1056/44 = 24$ decoding cycles). Therefore, the total IO overhead is $T_{\mathrm{IO}} = 24$ decoding cycles.

Table 3–2: Decoding parameters used.

Code	EM length (M)	IM length (L)	γ	Max. DCs
(1056,528)	$\{32, 48, 64\}$ for $d_v = (2, 3, 6)$	$\{1, 2\}$ for $d_v = (3, 6)$	0.5	700
(1056,704)	$\{32, 48, 48\}$ for $d_v = (2, 3, 4)$	$\{1, 1\}$ for $d_v = (3, 4)$	0.75	700

3.5.2 Performance and Tradeoffs

Table 3–2 lists the parameters used for each code. To obtain the characteristics of the proposed architecture, the (1056,528) irregular LDPC decoder is implemented on a Xilinx Virtex-4 XC4VLX200-11FF1513 device [98] using Xilinx ISE 9.2 tool. The following sections discuss the performance of the decoder.

3.5.2.1 Decoding Performance

Figures 3–11 and 3–12 show the decoding performance. These figures also depict the performance of the floating-point SPA. Also depicted in Figure 3–11 is the performance of the decoder in [104] and a (1024,512) EM-based stochastic decoder whose implementation values will be discussed in Section 3.7. Compared to the floating-point SPA with 32 and 16 iterations, the irregular stochastic decoders only have a loss of about 0.5 dB and 0.25 dB, respectively, at low BERs. It should be highlighted that, in the BER region shown, a similar error-floor behavior to that of the floating-point SPA is observed. Note that the floating-point implementation usually outperforms the fixed-point implementation which is considered in hardware implementations. In fact, because of complexity/area concerns, in most fully parallel decoders, fixed-point implementation with limited precision (usually ≤ 4 bits) is considered, which may cause additional decoding loss and/or higher error-floors.

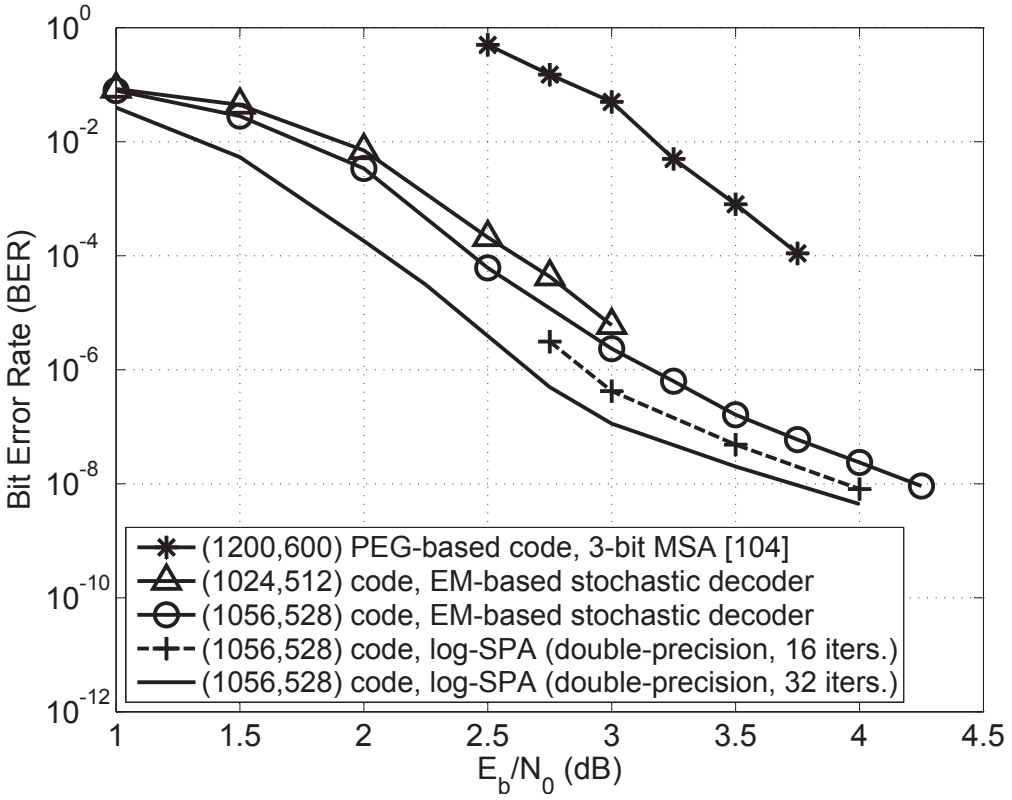


Figure 3–11: Decoding performance of the implemented (1056,528) irregular stochastic decoder.

Table 3–3: Xilinx Virtex-4 XC4VLX200-11FF1513 device utilization (LUT: 4-input look-up-table, FF: flip-flop).

Resources	Occupied	Available	Utilization
Slice LUTs	68163	178176	38%
Slice FFs	44502	178176	24%
IOBs	308	960	32%
Slices	46097	89088	51%

3.5.2.2 Area and Clock Frequency

Table 3–3 summarizes the area consumption of the (1056,528) decoder on the FPGA device. The decoder occupies about 38% of the 4-input look-up-tables and 24% of the flip-flops available on the device. These occupied resources are distributed in 51% of the device slices. The decoder uses one clock cycle per decoding cycle and achieves a clock rate of 222 MHz after place-and-route.

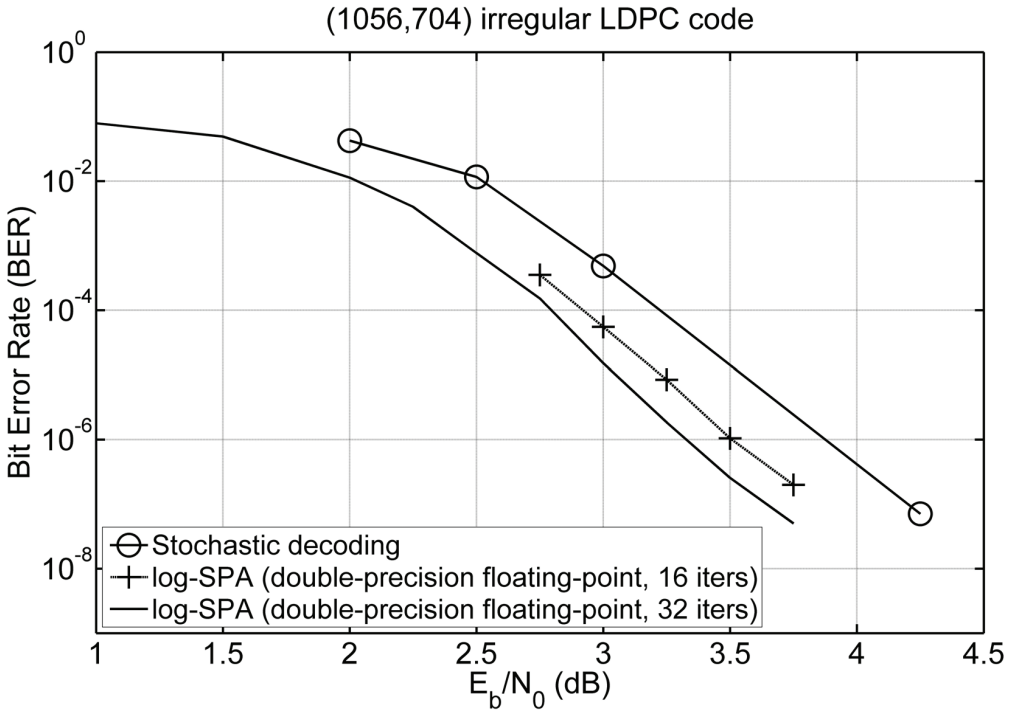


Figure 3–12: Decoding performance of the (1056,704) irregular stochastic decoder.

3.5.2.3 Throughput

As mentioned in Section 3.5.1.8, the decoder terminates the decoding and starts loading the next codeword when all the parity check signals are satisfied or when a maximum number of decoding cycles has been exceeded. Because of these termination criteria, T_{AVG}, the average number of decoding cycles used to load, decode and output codewords determines the throughput of the decoder. For the sake of brevity, we refer to T_{AVG} as the average number of decoding cycles. T_{AVG} is equal to

$$T_{\mathrm{AVG}} = T_{\mathrm{AVG_{CORE}}} + T_{\mathrm{IO}} + T_{\mathrm{LOAD_{EMs}}}, \tag{3.3}$$

where $T_{\mathrm{AVG_{CORE}}}$ is the average number of decoding cycles used by the core decoder to decode codewords and, as mentioned before, $T_{\mathrm{IO}} = 24$ and $T_{\mathrm{LOAD_{EMs}}} = 16$. It should be noted that at high SNRs (low BERs), T_{AVG} is much less than the maximum number of decoding cycles used by the core decoder ($T_{\mathrm{MAX_{CORE}}} =$

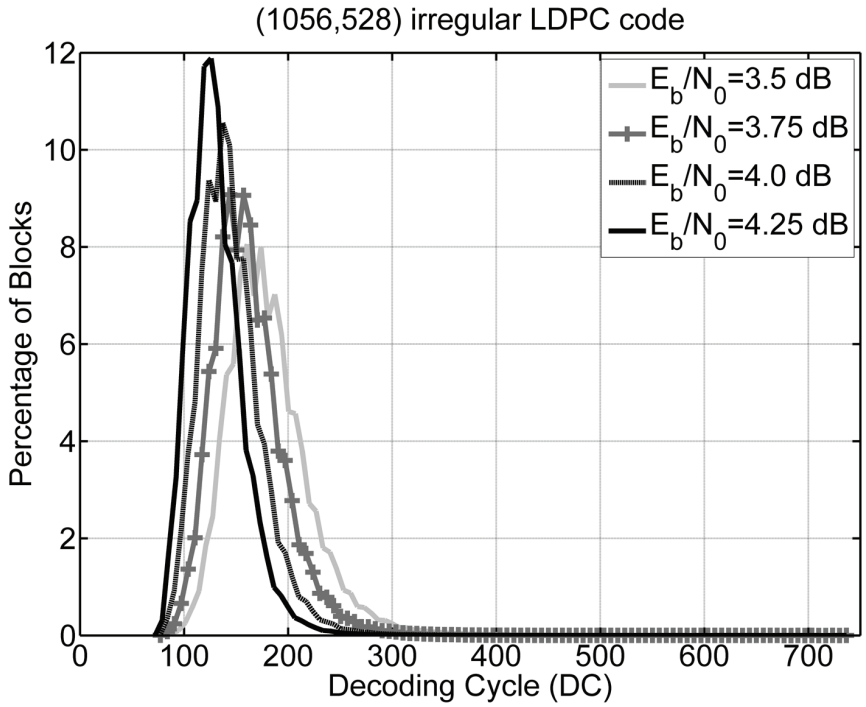


Figure 3–13: Histograms of T_{AVG} at different SNRs (based on 1 million blocks). Each decoding cycle takes one clock cycle.

700 decoding cycles). In fact at low BERs, only a few codewords require a high number of decoding cycles to decode. This is shown in Figure 3–13 in which the histograms of T_{AVG} over different SNRs are depicted. These histograms are based on observation of 1 million blocks.

Figure 3–14 shows the observed T_{AVG} over different SNRs. It also shows the throughput of the decoder based on T_{AVG} at different SNRs for the achieved clock rate of 222 MHz. T_{AVG} and the throughput of the decoder vary at different BERs. As Figure 3–14 shows, at high SNRs (low BERs) the throughput of the decoder is higher than the requirements of many applications. The decoder provides a throughput of more than 1 Gb/s for $E_b/N_0 > 3.5$ dB. The throughput of the decoder at $E_b/N_0 = 4.25$ is about 1.66 Gb/s.

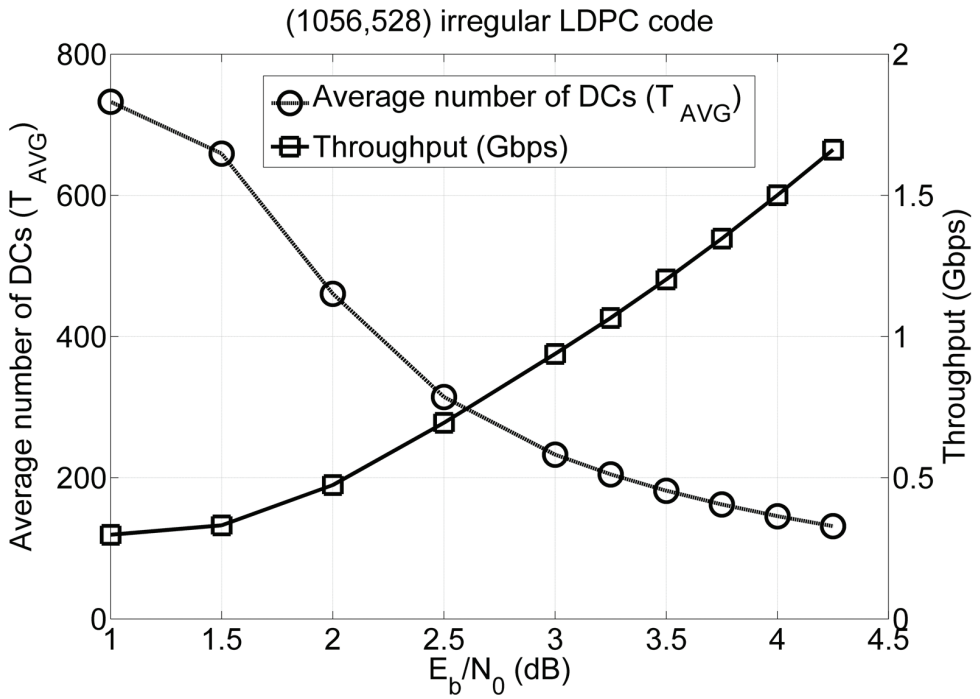


Figure 3–14: T_{AVG} and throughput of the decoder at different SNRs (based on 1 million blocks). Each decoding cycle takes one clock cycle.

3.5.2.4 Latency

Figure 3–15 shows the BER performance of the stochastic decoder versus decoding cycle. The maximum number of decoding cycles used for decoding the (1056,528) code was $T_{\text{MAX}_{\text{CORE}}} = 700$ decoding cycles. Because of the termination criteria of the decoder, $T_{\text{MAX}_{\text{CORE}}}$ only influences the latency of the decoder. The maximum latency of the decoder is determined by T_{MAX} which is calculated as

$$T_{\text{MAX}} = T_{\text{MAX}_{\text{CORE}}} + T_{\text{IO}} + T_{\text{LOAD}_{\text{EMs}}}. \tag{3.4}$$

For the (1056,528) decoder, T_{MAX} is 740 decoding cycles. With the achieved clock rate of 222 MHz, this results in a maximum latency of 3.3 μs which is in an acceptable range for most applications such as the IEEE 802.16e (WiMAX) standard. In addition, as Figure 3–15 suggests, for applications which have

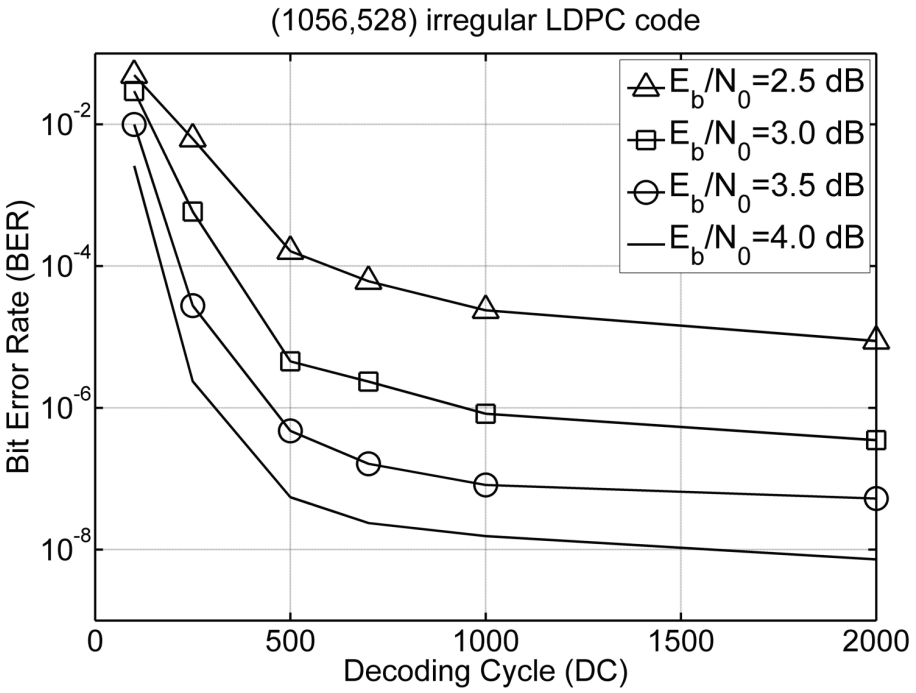


Figure 3–15: Decoding performance of the (1056,528) stochastic decoder over decoding cycles. Each decoding cycle takes one clock cycle.

a strict latency requirement, it is possible to trade the latency with decoding performance.

3.6 A (1024,512) Fully Parallel EM-based LDPC Decoder

In this section, we report an FPGA EM-based stochastic decoder that decodes a (1024,512) regular LDPC code with $d_v = 3$ and $d_c = 6$. The implementation values obtained for this decoder enable us to better compare the EM-based stochastic decoders with state-of-the-art FPGA LDPC decoders in Section 3.7. Because all the building blocks of this EM-based decoder are similar to those in the (1056,528) decoder discussed in the previous section, we briefly mention the implementation characteristics and values of this decoder.

The (1024,512) decoder uses 8-bit input channel probabilities and a $\gamma = 0.5$ for scaling. All VNs have 64-bit EMs, 1-bit IMs, and 6-bit saturating up/down counters to make the hard-decision. The decoder uses a 2-stage

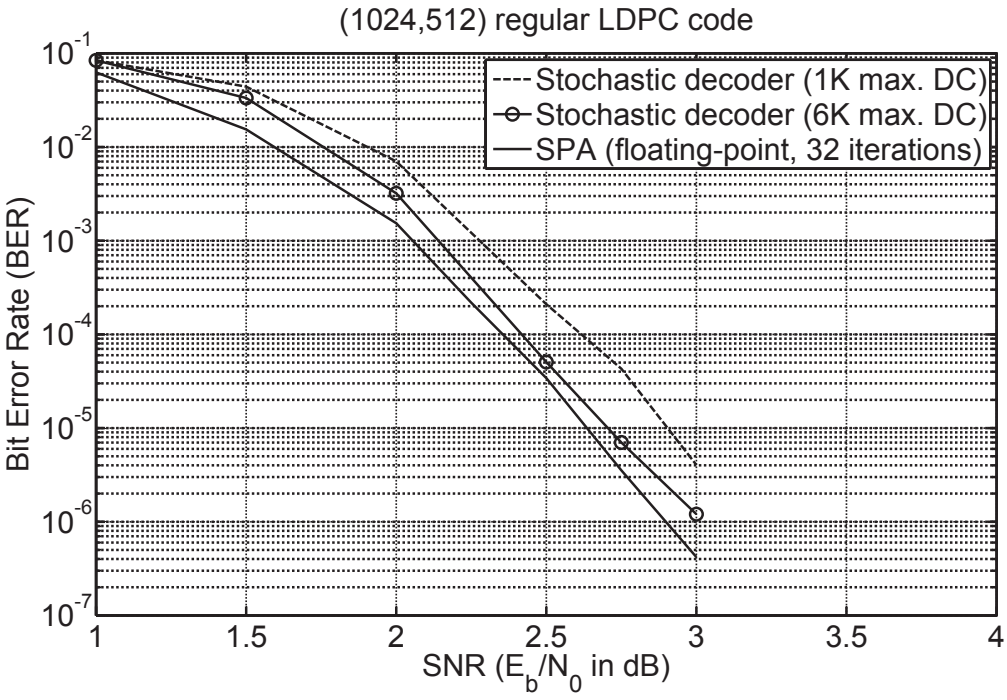


Figure 3–16: Decoding performance of the (1024,512) stochastic decoder.

pipeline interleaver and its DRE generates 32 8-bit (pseudo) random numbers every decoding cycle. Each (pseudo) random number is shared between $1024/32 = 32$ VNs. The (1024,512) decoder employs the termination criteria described in the previous section. The decoder is implemented on a Xilinx Virtex-4 XC4VLX200-11FF1513 FPGA device. It occupies 32875 slices of the device (i.e., 36% of the total available slices). It achieves a maximum clock frequency of 212 MHz and a throughput of 706 Mb/s at a BER of 10^{-6}. The BER performance of the decoder is shown in Figure 3–16 for 6K and 1K maximum decoding cycles. Each decoding cycle takes one clock cycle.

3.7 Comparison

This section compares the characteristics of the implemented EM-based stochastic LDPC decoders with some recent high throughput FPGA-based LDPC decoders.

3.7.1 Comparison with FPGA Fully Parallel Decoders

Table 3–4 compares different aspects of the most recent FPGA-based fully parallel LDPC decoders. To our knowledge, the decoders in [104] and [27] are among the fastest (non-stochastic) FPGA-based fully parallel LDPC decoders. The decoder in [104] decodes a (1200,600) regular code which is constructed based on the progressive-edge-growth method [43]. The throughput of the decoder is 12 Gb/s. This throughput was achieved by employing 3-bit fixed-point MSA with 10 iterations. The decoder in [27] decodes a (480,355) regular code with a throughput of 650 Mb/s using the bit-serial approximate MSA with 15 iterations. The table also shows the implementation characteristics of the (1056,528) and (1024,512) EM-based stochastic LDPC decoders. Note that compared to the (1024,512) decoder, the (1056,528) decoder has a much more complex structure because of its irregularity and high-degree nodes.

Table 3–4 gives the throughput efficiency per information bit for each decoder. As discussed in Section 3.5.2.3, the throughput of the (1056,528) decoder varies at different SNRs. For example, the decoder provides a throughput of 694 Mb/s at a $E_b/N_0 = 2.5$ dB (BER $\simeq 10^{-4}$) and at $E_b/N_0 = 4.25$ dB (BER $\simeq 10^{-8}$) the throughput is about 1.66 Gb/s. Compared to [104] and [27], the stochastic decoder has a higher latency. This latency is however within an acceptable range for many applications. Usually, a latency limit of about 6 μs is assumed for channel decoders in applications such as WiMAX. In addition, as mentioned before, for applications with stricter latency requirements, it is possible to trade the decoding performance with the latency (see Figure 3–15).

Table 3–4 also gives the absolute area as well as area efficiency based on the number of 4-input look-up-tables and flip-flops per coded bit, and slices per coded bit. Note that a Logic Element in the Altera Stratix architecture has one 4-input look-up-table and one flip-flop [6] which is half of the resources of a

slice in a Xilinx Virtex-4 architecture [98]. Since the number of look-up-tables and flip-flops were not reported in [27], the comparison with this decoder is based on the approximate slice per coded bit efficiency. The area efficiency of the stochastic decoder is better than the bit-serial decoder in [27]. Compared to [104], the stochastic decoder needs more look-up-tables and flip-flops per coded bit (but offers about 1.3 dB decoding gain as shown in Figure 3–11). The majority of this difference is because of the higher degree of VNs. As shown, the (1024,512) EM-based stochastic decoder, with the same rate and node degrees as in [104], needs much fewer resources and offers a better slice per coded bit efficiency compared to [104].

Compared to other fully parallel approaches, an important advantage of the stochastic approach is its good decoding performance and error-floor behavior. Figure 3–11 compares the performance of the (1024,512) stochastic decoder with the decoder in [104]. Both codes are regular and have the same rate and node degree. As shown, even though the (1200,600) LDPC code in [104] is longer, stochastic decoders outperform this decoder by more than 1 dB. It should be noted that the reported area efficiency for stochastic decoders is for providing a performance close to the floating-point SPA. The stochastic decoding approach is able to easily trade the hardware-complexity with decoding performance. For example, if performance close to the fixed-point MSA with limited precision is required, it is possible to significantly increase the area efficiency and/or reduce the latency of the stochastic decoder by using much shorter EMs/IMs, simpler DRE, and by reducing the precision of comparators/counters.

3.7.2 Comparison with FPGA Partially Parallel Decoders

As mentioned before, partially parallel decoders use memory and share hardware to trade area/flexibility with throughput. Fully parallel decoders,

however, occupy a much larger area but provide much higher throughput. In this respect, partially parallel and fully parallel decoders occupy a different place on the trade-off curve. This is also the case for the proposed fully parallel architecture. Compared to the recent FPGA partially parallel decoders, the (1056,528) stochastic decoder occupies much more (absolute) area but corrects more errors at a much higher speed. For example, the multi-rate partially parallel decoder in [40] occupies 1640 to 6568 slices and uses more than 60K bits of RAM, and provides a throughput of 41 to 278 Mb/s on a Xilinx Virtex-II 2V8000 device. Also, the partially parallel (8176,7154) decoder in [92] uses about 23K to 27K slices and 128 block RAMs of a Xilinx VirtexII-6000 FPGA device and, provides a throughput of 172 Mb/s with 15 decoding iterations.

3.8 Conclusion

This chapter proposed EMs for stochastic decoding of state-of-the-art LDPC codes on factor graphs. It presented a novel decoder architecture for fully parallel EM-based stochastic LDPC decoders. The proposed decoder architecture was used for the FPGA implementation of a fully parallel stochastic LDPC decoders that decodes a state-of-the-art irregular (1056,528) code. It was also applied for the FPGA implementation of an EM-based stochastic decoder that decodes a (1024,512) regular LDPC code. Both decoders were implemented on Xilinx Virtex-4 LX200 FPGA devices. The (1056,528) decoder exploits several novel architectural techniques, provides a throughput of 1.66 Gb/s at $E_b/N_0 = 4.25$ dB (BER of 10^{-8}), and achieves decoding performance within 0.5 dB and 0.25 dB loss of the floating-point SPA with 32 and 16 iterations, respectively. It was shown that this decoder provides similar error-floor behavior as the floating-point SPA with 32 iterations. The LDPC decoders proposed in this chapter are the first stochastic decoders that decode

state-of-the-art LDPC codes, and they are among the fastest and the most area-efficient fully parallel LDPC decoders implemented on FPGAs.

In the next chapter, we focus on the ASIC implementation of stochastic LDPC decoders. We propose a new approach for decorrelating and rerandomizing stochastic streams. This approach significantly increases the silicon area efficiency of stochastic decoders and thus facilitates their ASIC implementations.

Table 3–4: Comparison of FPGA-based fully parallel LDPC decoders (LUT: 4-input look-up-table, FF: flip-flop, LE: Logic Element).

	Fast non-stochastic FPGA-based decoders		Stochastic decoders	
	[27]	[104]		
Code	(480,355)	(1200,600)	(1024,512)	(1056,528)
Code structure	Regular, RS-based	Regular, PEG-based	Regular	Irregular, WiMAX code
Max. (d_v,d_c)	(4,15)	(3,6)	(3,6)	(6,7)
Decoding	Bit-serial approx. MSA	3-bit fixed-point MSA	Stochastic	Stochastic
Iterations or DCs	15 iters.	10 iters.	6K and 1K (max DCs)	700 (max DCs)
Input quantization	3 bits	3 bits	8 bits	6 bits
FPGA device	Stratix EP1S80	Virtex-4 XC4VLX200	Virtex-4 XC4VLX200	Virtex-4 XC4VLX200
Max. clock	61 MHz	100 MHz	212 MHz	222 MHz
Clocks per iter. or DC	3	1	1	1
Max. latency (μs)	0.73	0.1	28.30 (for 6K max. DCs) 4.71 (for 1K max. DCs)	3.3
Throughput	650 Mb/s	12 Gb/s	706 Mb/s	1.66 Gb/s
Throughput per information bit	1.83 Mb/s	10 Mb/s	1.38 Mb/s (at E_b/N_0=3 dB)	3.14 Mb/s (at E_b/N_0=4.25 dB)
Absolute area (in slices/LEs)	66588 LEs ($\simeq$33294 slices)	40613 slices	32875 slices	46097 slices
4-input LUTs and FFs per coded bit	not reported	57.5 LUTs and 15.7 FFs	46.0 LUTs and 20.1 FFs	64.5 LUTs and 42.1 FFs
Slices/LEs per coded bit	138.7 LEs ($\simeq$69.3 slices)	33.8 slices	32.1 slices	43.6 slices
Relative decoding gain (at BER=10^{-4})	not comparable	–	$\simeq$ 1.1 dB gain, compared to [104]	$\simeq$ 1.3 dB gain, compared to [104]
Hardware decoding loss (at BER=10^{-4})	not reported	$\simeq$ 0.25 dB loss from floating-point MSA (10 iters.)	$\simeq$ 0.2 dB loss from floating-point SPA (32 iters.)	$\simeq$ 0.4 dB loss from floating-point SPA (32 iters.)

CHAPTER 4

Edge-Based Rerandomization Using Tracking Forecast Memories

As discussed in the previous chapter, EMs can be efficiently implemented using shift register look-up-tables available in FPGAs. This implementation makes EM-based stochastic LDPC decoders resource-efficient in FPGAs. However, EMs consume considerable silicon area in ASIC. For example, a 64-bit EM has about 821 two-input NAND gate-count complexity and occupies about 4506 μm^2 silicon area in CMOS 90nm technology (when synthesized for maximum possible speed). Because EMs are assigned to each outgoing edge of VNs in a factor graph (see Figure 4–1), a practical ASIC stochastic LDPC decoder requires thousands of EMs whose silicon area consumption becomes the bottleneck of the overall hardware-complexity of the decoder. In this respect, less-complex solutions that can provide similar or better decoding performance are important.

In this chapter, we propose TFMs to replace EMs in ASIC stochastic decoders. We discuss various hardware architectures for ASIC implementation of TFMs and their effects on the complexity of stochastic VNs. We also provide examples of ASIC stochastic decoders that decode a (1056,528) LDPC code chosen from the IEEE 802.16 (WiMAX) standard. We show that TFMs can provide similar or better decoding performance compared to EMs, while having

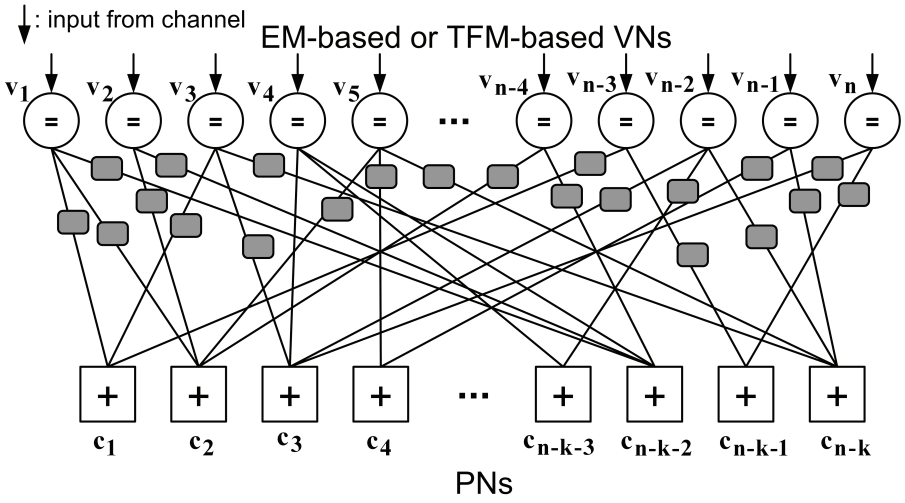


Figure 4–1: EMs or TFMs are used for rerandomization/decorrelation of stochastic streams and are assigned to each outgoing edge of stochastic VNs.

much lower hardware-complexity. We also investigate the impact of TFMs on the overall area of ASIC stochastic decoders.

4.1 Tracking Forecast Memories

TFMs replace EMs in stochastic VNs. Similar to EMs, TFMs are used to alleviate the latching problem and increase the switching activity by rerandomizing/decorelating stochastic streams. In this respect, EMs and TFMs can be considered as rerandomization units in stochastic decoders as shown in Figures 4–1 and 4–2. A TFM efficiently extracts the moving average probability of a stochastic stream based on the method of successive relaxation [49, 71, 103]. Let $r(t) \in \{0, 1\}$ be the input bit of a TFM in a stochastic VN and $P(t)$ be the probability extracted by the TFM at time t for the corresponding stochastic stream ($0 \leq P(t) \leq 1$). The TFM updates $P(t)$ in nonhold (regular) states as follows:

$$P(t+1) = P(t) + \beta(t)\left(r(t) - P(t)\right), \tag{4.1}$$

where $\beta(t)$ is the relaxation coefficient and usually $0 < \beta(t) < 1$. In stochastic decoding, β is considered as an empirical factor whose value (for decoding a

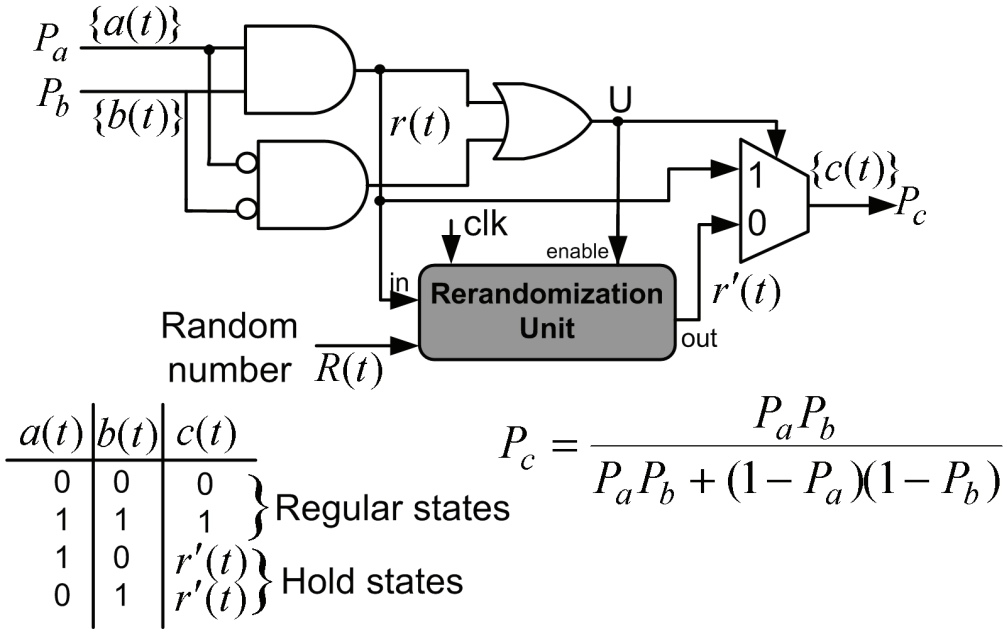


Figure 4–2: Structure of a degree-2 stochastic VN (only one output and its corresponding inputs are shown). An EM or a TFM can be used as a rerandomization unit.

specific code) can be chosen by simulation (i.e., based on the best decoding performance obtained for different β). This memory-based mechanism is named TFM since it is based on tracking the past observations while emphasizing recent outcomes (see [59]).

As shown in Figure 4–2, the operation of a TFM-based VN is similar to an EM-based VN (discussed in Section 3.5). In regular states, the TFM is updated as in (4.1) and $c(t) = r(t)$. When the VN is in the hold state, the TFM is not updated and $c(t) = r'(t)$. To generate $r'(t)$, $P(t)$ is compared against a (pseudo) random number, $R(t)$, as follows:

$$r'(t) = \begin{cases} 1 & P(t) > R(t), \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

The TFM update criterion provides an exponential time-decaying dependence on input bits. By unrolling (4.1) we have:

$$\begin{aligned} P(t+1) = & \\ & P(0)\left(\prod_{j=0}^{t}(1-\beta(j))\right) \\ & +\beta(t)r(t)+\sum_{j=0}^{t-1}\left(\left(\prod_{u=j+1}^{t}(1-\beta(u))\right)\beta(j)r(j)\right). \end{aligned} \tag{4.3}$$

Also, when $\beta(t)$ is a constant value equal to β, we have:

$$P(t+1) = P(0)\,(1-\beta)^{t+1} + \beta\sum_{j=0}^{t}(1-\beta)^{t-j}r(j). \tag{4.4}$$

Figure 4–3 (a) and (b), respectively, depict the dependence of $P(t+1)$ on previous input bits in a TFM with $\beta(t) = 2^{-5}$ and in an EM with a length of $M = 32$ bits. As shown, the dependence in the TFM exponentially decreases over time as $\beta, (1-\beta)\beta, (1-\beta)^2\beta, ...$, but the dependence in the EM is equal to $1/M$ for the last M input bits and is zero for the prior input bits. Figure 4–3 (c) shows the dependence in a bit-serial TFM which is presented in Section 4.2.3.

The strong dependence on recent input regenerative bits and gradual forgetting of older input bits enable TFMs to track changes in $P_r(t) = \Pr(r(t) = 1)$. The value of $\beta(t)$ determines the speed and the accuracy of the convergence/response of TFMs. Figure 4–4(a) shows the convergence of a TFM for different values of $\beta(t)$. The input stream of the TFM is generated based on input stream probability of $P_r(t) = 0.8$ and the TFM is initialized to $P(0) = 0$ for each case. As shown, as the value of $\beta(t)$ decreases, TFM converges more conservatively toward $P_r(t)$, but after convergence, $P(t)$ approximates $P_r(t)$ more accurately and with less fluctuations. This can be also seen in Figure 4–4(b), in which the estimation error of TFMs, $E(t) = |P(t) - P_r(t)|$, is depicted.

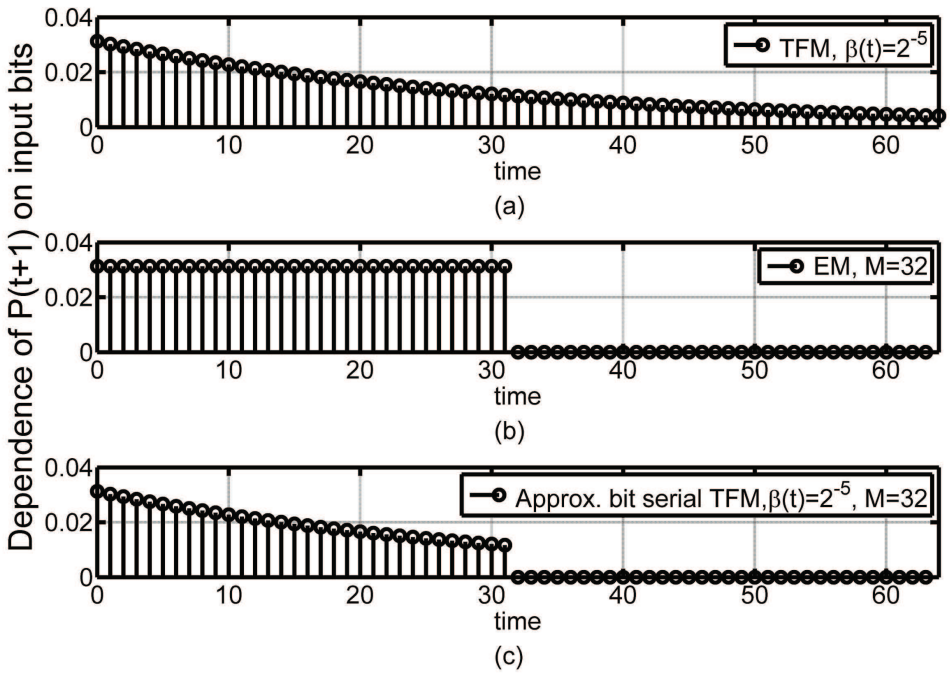


Figure 4–3: The dependence of output probability on previous input bits in (a) TFM with $\beta = 2^{-5}$, (b) EM with $M = 32$ bit length and, (c) approximate bit-serial TFM with $M = 32$ bit length and $\beta = 2^{-5}$.

4.2 Hardware Realization of TFMs

This section discusses different variants of TFMs and their hardware realizations.

4.2.1 General Architecture

Figure 4–5 shows the general architecture of a TFM. In this architecture, it is assumed that $\beta(t)$ can vary over time. We recall that the signal U in the figure determines if the VN is in hold state (U= 0) or if it is in a nonhold state (U = 1) and hence $r(t)$ is regenerative. When U = 1, $P(t)$ is updated and VN directly uses $r(t)$ as the output bit. When U= 0, $P(t)$ does not change and the VN uses $r'(t)$ as the output bit. This architecture requires the use of one multiplier, two adders, one comparator and one register.

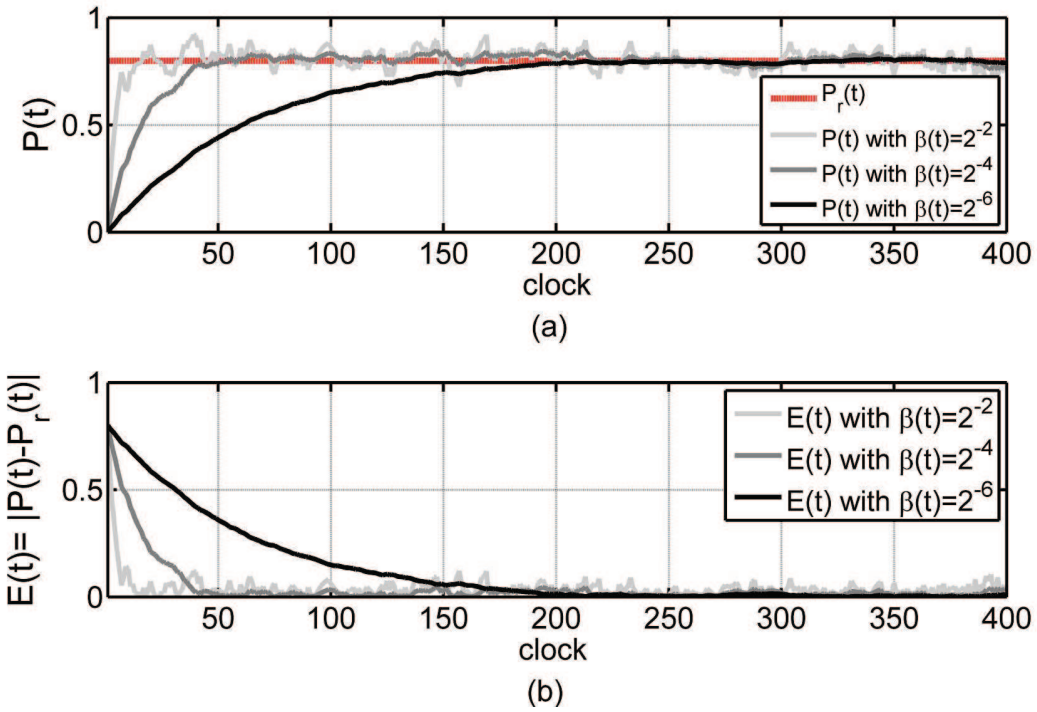


Figure 4–4: (a) The convergence speed and (b) the corresponding estimation error of a TFM for different values of $\beta(t)$.

4.2.2 Reduced-Complexity Architecture

The complexity of a TFM is significantly reduced when β is chosen as a negative power of 2. In this case, the multiplication involved in the TFM operation can be replaced by shifting bit wires of $P(t)$. We also propose that when $P(t)$ is represented as an unsigned integer, $1 - P(t)$ is equal to $\bar{P}(t)$, the complement of $P(t)$; therefore, the two adders in Figure 4–5 can be replaced by one adder/subtractor unit because

$$P(t+1) = \begin{cases} P(t) - \beta(t)P(t) & r(t) = 0, \\ P(t) + \beta(t)\bar{P}(t) & r(t) = 1. \end{cases} \tag{4.5}$$

Figure 4–6 shows the proposed reduced-complexity architecture for a TFM. Compared to the general architecture, this architecture does not use any multiplier and uses one fewer adder. In our simulations we observed that

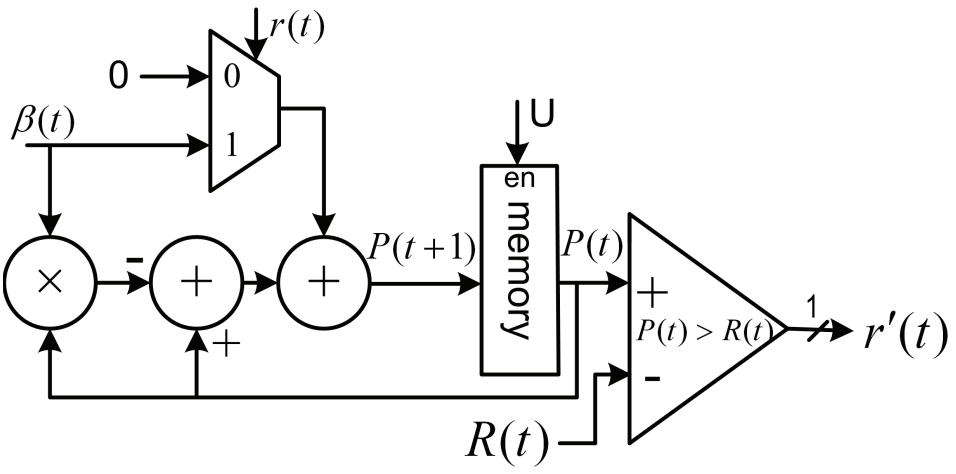


Figure 4–5: General architecture of a TFM. $\beta(t)$ can change and take any value in the [0,1] interval.

by using the reduced-complexity architecture the decoder is able to provide similar decoding performance as the general architecture.

4.2.3 Approximate Bit-Serial Architecture

As shown in Figure 4–3, a TFM provides an exponential time-decaying dependence on the past input regenerative bits. In approximate bit-serial TFMs, the TFM operation is approximated by using only the last M regenerative input bits as:

$$P(t+1) = P(0)\,(1-\beta)^{M+1} + \beta \sum_{j=t-M+1}^{t} (1-\beta)^{t-j} r(j-1). \tag{4.6}$$

Figure 4–3(c) depicts the dependence on previous input bits in an approximate bit-serial TFM with $\beta(t) = 2^{-5}$ and $M = 32$. It can be seen that for the last M bits, the dependence is the same as a TFM, while for the prior input bits the dependence is zero.

Figure 4–7 depicts the proposed architecture for bit-serial approximate TFM. In this architecture, the last M regenerative bits are stored in an M-bit shift register and upon receiving a new regenerative bit a single shift operation is performed. The TFM operation is performed by a series of concatenated

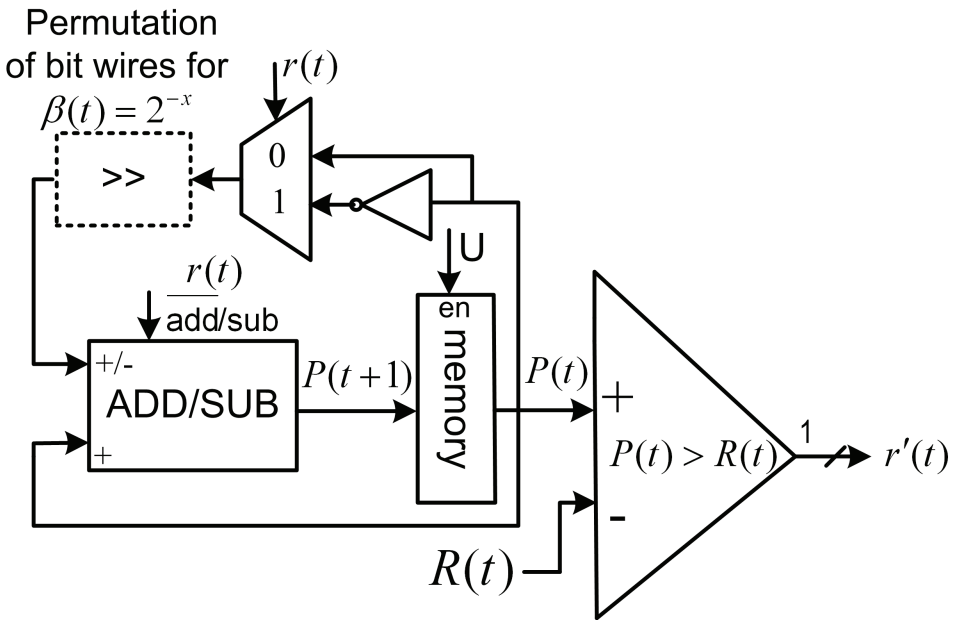


Figure 4–6: Architecture of a reduced-complexity TFM. $\beta(t)$ is a negative power of 2.

multiplexers. The selection line of each multiplexer is an independent stochastic stream with a probability of $P_s = \beta$ where $0 \leq \beta \leq 1$. This means that at each multiplexer stage, the bit in the shift register is (directly) selected with a probability of P_s and the bit from the previous stage is passed through with a probability of $1 - P_s$. To provide an (initial) estimation for (residual) regenerative input bits that has been neglected (i.e., $r(j)$ with $j < t - M$), the generated stochastic stream, $\{\text{in}_{\text{CH}}\}$, from the received channel probability, $P_{\text{CH}} = \Pr(\text{in}_{\text{CH}} = 1)$, can be connected as the input bit to the last multiplexer. In this case, the probability of $r'(t)$ being equal to 1 is:

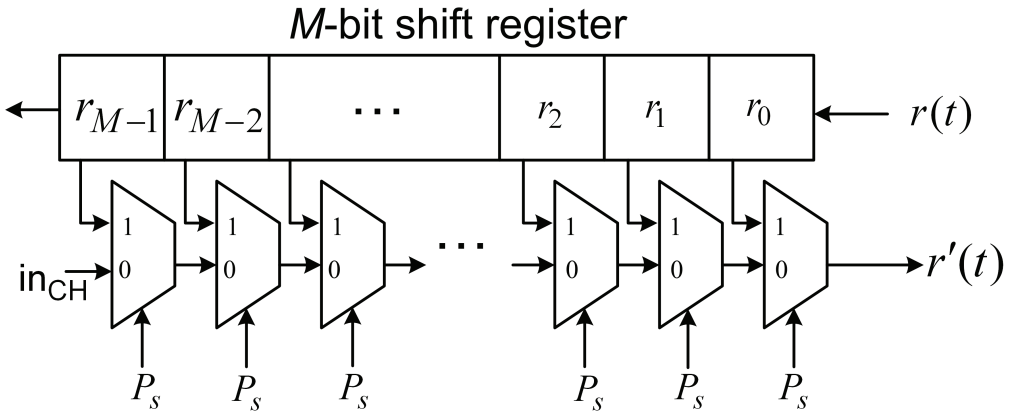


Figure 4–7: Architecture of an approximate bit-serial TFM. in_{CH} is the input stochastic bit from the channel and $P_s = \beta$

$$\begin{aligned}\Pr(r'(t)=1) = \\ &\beta\Pr(r_0=1) + \beta(1-\beta)\Pr(r_1=1) + \cdots \\ &+ \beta(1-\beta)^{M-1}\Pr(r_{M-1}=1) \\ &+ (1-\beta)^M P_{\text{CH}} \\ &= \beta\left(\sum_{j=0}^{M-1}(1-\beta)^j\Pr(r_j=1)\right) + (1-\beta)^M P_{\text{CH}}.\end{aligned} \tag{4.7}$$

It should be noted that compared to the shift register used in an EM (shown in Figure 3–1), the shift register in the bit-serial TFM is less complex, because it does not provide a single selectable output bit and hence, does not require the use of an address decoder. However, compared to EMs and non-bit-serial architectures for TFMs, the random number generation for bit-serial TFMs is more complex and, depending on the length of the shift registers used, can require more physical wires.

4.2.4 Approximate Counter-Based Architecture

Another method for approximating the TFM operation, for the sake of lower hardware-complexity, is to approximate the addition/subtraction in

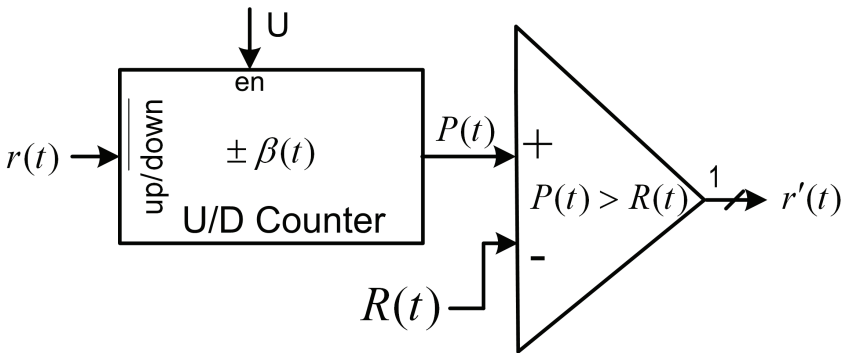


Figure 4–8: Architecture of an approximate counter-based TFM.

(4.1). Since the probability stored in a TFM memory is in $[0, 1]$ interval, the TFM update rule in (4.1) implies that the maximum absolute change in the value of $P(t)$ is $\beta(t)$ in other words, $|P(t+1) - P(t)| \leq \beta(t)$. Therefore, for the sake of lower hardware-complexity, it is possible to approximate $P(t+1)$ by an up/down counter (see Figure 4–8) with a step-size equal to $\beta(t)$ which operates as follows:

$$P(t+1) = \begin{cases} P(t) - \beta(t) & r(t) = 0, \\ P(t) + \beta(t) & r(t) = 1. \end{cases} \tag{4.8}$$

4.3 Comparison of TFM-based and EM-based Variable Nodes

Figure 4–9 compares the performance of EM and TFM approaches for decoding a (2048,1723) LDPC code with degree-6 VNs and degree-32 PNs chosen from the 10 Gb/s Ethernet (10GBASE-T) standard [2]. The figure shows the decoding performance of 32-bit EMs, 64-bit EMs, 12-bit reduced-complexity TFMs, 12-bit approximate bit-serial TFMs, and 12-bit counter-based TFM approaches. For all of these approaches, symbols received from the channel are quantized to 6 bits and a scaling factor of $\gamma = 1.33$ is used. Also, an early decoding termination (based on syndrome checking) until a

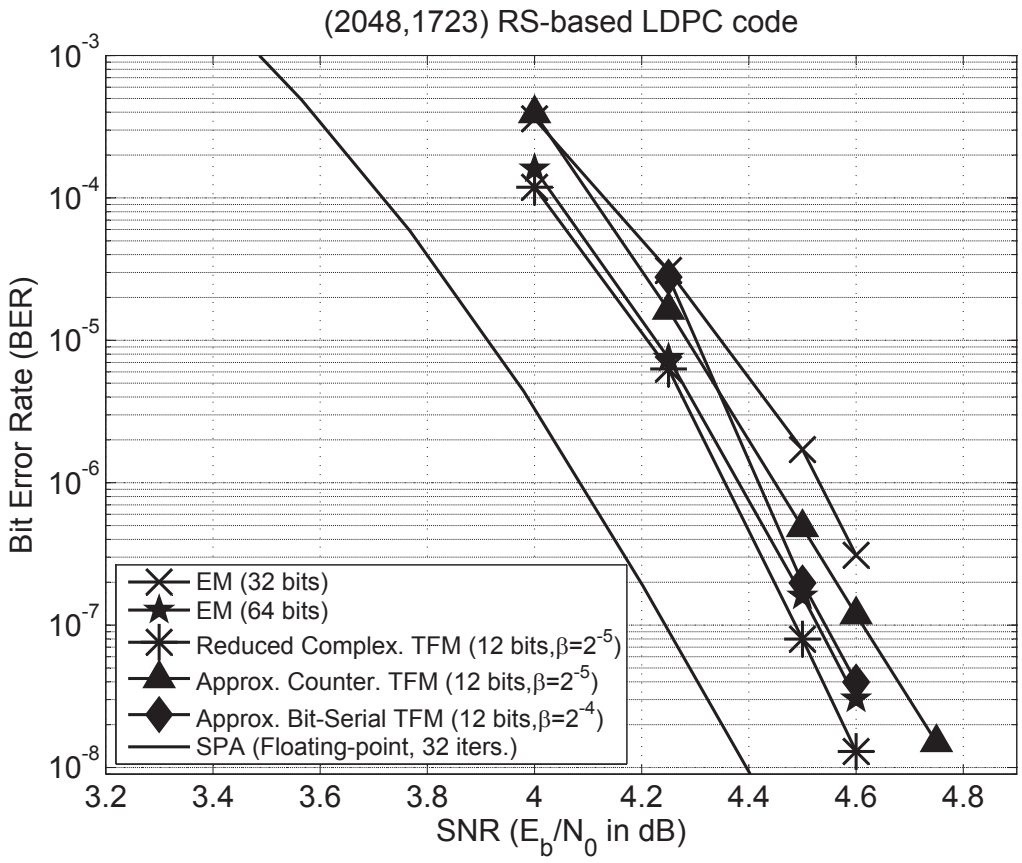


Figure 4–9: Comparison of decoding performance of EMs and TFMs.

maximum number of 400 decoding cycles is used. As shown, the reduced-complexity 12-bit TFM approach outperforms 64 bits and 32 bits EMs for decoding the (2048,1723) LDPC code.

Table 4–1 shows the silicon area consumption, 2-input NAND gate count, and the maximum achievable clock period for degree-6 stochastic VNs in CMOS 90nm technology. The table is divided into two sections. Results shown in the first (left) section are obtained for synthesizing for the maximum possible speed. Therefore, as shown in the second (right) section of the table, by synthesizing the modules for a lower target clock frequency, lower silicon area consumption can be obtained. The first section of the table confirms that stochastic VNs are able to operate very fast, with clock frequencies beyond 2.5

Table 4–1: Hardware-complexity of TFM-based and EM-based degree-6 VNs in CMOS 90nm technology.

Module and Architecture	Area in μm^2 & Gate count	Clock period in ps	Area in μm^2 & Gate count at 500 MHz
EM-based VN (32 bits)	(13860, 2525)	283	(7133, 1299)
EM-based VN (64 bits)	(22575, 4112)	287	(12255, 2232)
TFM-based VN (12 bits bit-serial)	(9223, 1670)	263	(4352, 793)
TFM-based VN (12 bits counter-based)	(9111, 1660)	312	(5318, 969)
TFM-based VN (12 bits reduced-complexity)	(14989, 2730)	354	(5924, 1079)

GHz. With respect to the area consumption, as the second section of the table shows, a reduced-complexity TFM-based VN consumes about 48% of the silicon area of a 64-bit EM-based VN and about 83% of the silicon area of a 32-bit EM-based VN. Also, it is possible to further reduce the area consumption of a TFM-based VN by about 10% to 26% by using the approximate counter-based and the approximate bit-serial TFM architectures.

4.4 Decoding Performance and Hardware-Complexity

This section investigates the effects of the TFM approach on the overall hardware-complexity and decoding performance of ASIC stochastic decoders. We compare the decoding performance and ASIC implementations of a TFM-based decoder and two EM-based decoders. All decoders decode a (1056,528) irregular LDPC code that is chosen from the IEEE 802.11n (WiMAX) standard [3]. In Section A.2 of Appendix A, we provide the performance results of the TFM approach for decoding various LDPC codes.

4.4.1 Decoding Performance

Figure 4–10 shows the BER decoding performance of the proposed TFM method for decoding a (1056,528) LDPC code. To show the effects of probability quantization in the TFM architecture on decoding performance, the

figure shows the performance of floating-point TFM implementation as well as 9-bit, 8-bit, and 7-bit fixed-point TFM implementations. Also, for the sake of comparison, the figure shows the decoding performance of the floating-point SPA, the floating-point MSA, and the floating-point offset MSA. In addition, the figure shows the performance of two EM-based stochastic decoders that decode the same (1056,528) LDPC code: (a) the EM-based decoder discussed in Section 3.5 which uses 64-bit, 48-bit, and 32-bit EMs for degree-6, degree-3 and degree-2 VNs, respectively, and (b) an EM-based stochastic decoder which uses 32-bit EMs for every VNs. All stochastic decoders in the figure use early termination, a maximum of 700 decoding cycles, and a scaling factor of $\gamma = 0.5$. Note that for the case of "ideal" stochastic decoding, floating-point implementation is used and random numbers in the stochastic decoder are not shared. For true-bit stochastic simulations, fixed-point implementation of TFMs is considered and symbols received from the channel are quantized to six bits as in Section 3.5. Also, random numbers are generated by a DRE identical to the one used in the (1056,528) EM-based stochastic decoder in Section 3.5. This DRE consists of 48 independent random number generators (randomization engines). Each random number generator consists of two 10-bit LFSRs and produces a (pseudo) random number which is shared between 22 VNs (by XORing different bits of the LFSRs). Therefore, in total, $1056/22 = 48$ random numbers are procured for the entire decoder in every decoding cycle.

As shown in Figure 4–10, the floating-point TFM outperforms the EM-based decoder at low SNRs. The decoding performance of stochastic decoders with 9-bit fixed-point TFMs and 8-bit fixed-point TFMs (with shared random numbers) are similar to the performance of the floating-point TFMs at low BERs (high SNRs). The decoder with 9-bit fixed-point TFMs exhibits a performance similar to the performance of the EM-based decoder in Section 3.5

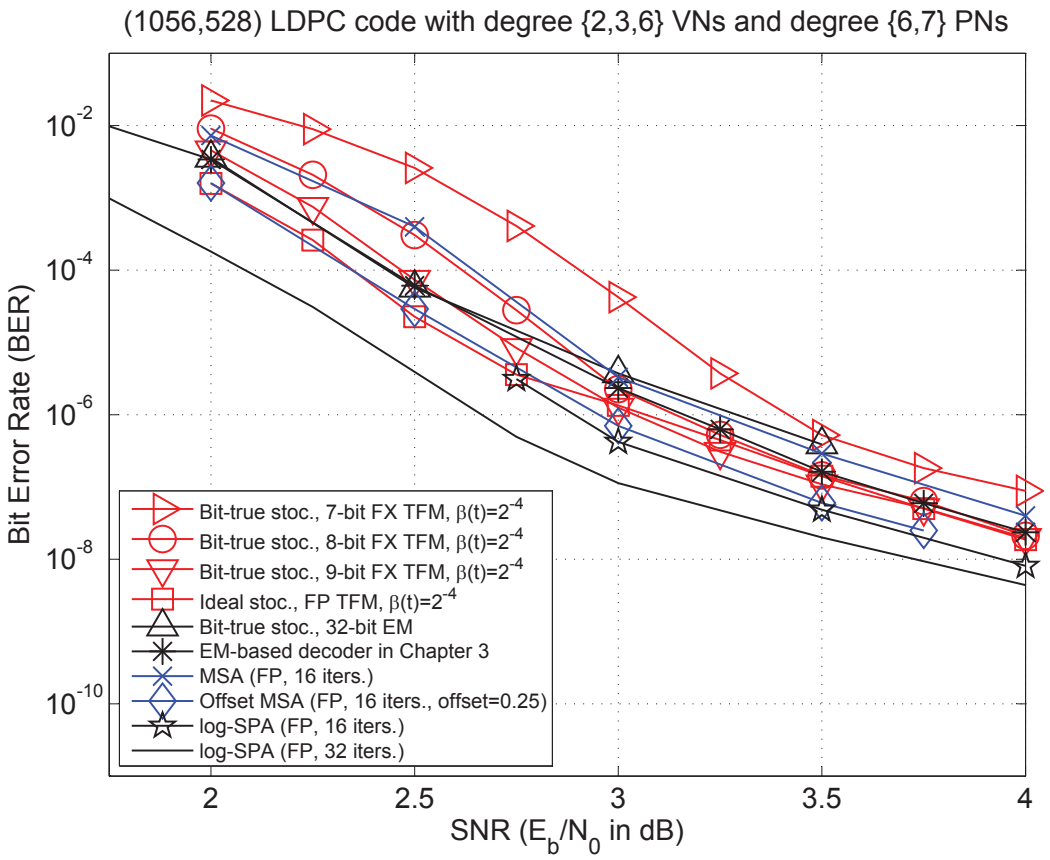


Figure 4–10: Decoding performance results for a (1056,528) LDPC code (FP:floating-point, FX:fixed-point).

and outperforms the 32-bit EM-based stochastic decoder. It also outperforms the floating-point MSA with 16 iterations and, at low SNRs, it has a performance comparable to the performance of the floating-point offset MSA with 16 iterations. The performance loss of the decoder with 9-bit TFMs is about 0.5 dB and 0.25 dB, when compared to the floating-point SPA with 32 and 16 iterations, respectively. In summary, it can be concluded that 8 or 9-bit fixed-point TFMs are sufficient to provide similar decoding performance as the EMs used in Section 3.5 for stochastic decoding of the (1056,528) irregular LDPC code. However, since the decoding performance of 8-bit and 9-bit TFMs are similar in low BER (high SNR) regimes, where practical LDPC decoders are expected to be used, we consider 8-bit fixed-point TFMs for the

implementation of the (1056,528) stochastic LDPC decoder in the following section.

4.4.2 Hardware-Complexity Comparison

To study the effects of TFMs on the overall area of stochastic decoders, we implemented two EM-based and one TFM-based fully parallel stochastic LDPC decoders which decode the (1056,528) irregular LDPC code. The TFM-based decoder uses 8-bit TFMs and the EM-based decoders use 64-bit EMs and 32-bits EMs. All decoders are synthesized in the ST Microelectronics 90 nm 1V CMOS technology and are clocked at 500 MHz. Note that, as shown in Chapter 3, with this clock frequency a fully parallel stochastic LDPC decoder is able to provide multi Gb/s throughput at low BERs (high SNRs). Table 4–2 summarizes the synthesis results for area and 2-input NAND gate count of these decoders. As shown, TFMs significantly reduce the hardware-complexity of the stochastic decoder. The area and the gate count of the TFM-based decoder is about 60% of the area and the gate count of the 32-bit EM based decoder and it is about 35% of the area and the gate count of the 64-bit EM based decoder. It is also possible to approximately compare the area efficiency of the (1056,528) TFM-based decoder with other decoders in the literature. For example, the 8-bit TFM-based decoder has 517K/1056 $\simeq$ 489 gate-count-per-coded-bit complexity (after synthesis) which is much less than the 2230K/2048 $\simeq$ 1088 gate-count-per-coded-bit complexity (after synthesis) reported for the bit-serial MSA-based (2048,1723) LDPC decoder in [28].

4.5 Conclusion

This chapter proposed TFMs for efficient rerandomization and decorrelation of stochastic bit streams in stochastic channel decoders. Various hardware architectures for ASIC implementation of TFMs were discussed. It was shown that TFMs are able to provide similar or better BER decoding performance as

Table 4–2: Synthesis results for EM-based and TFM-based (1056,528) stochastic LDPC decoders in CMOS 90nm technology. All decoders are synthesized for 500 MHz clock frequency.

Decoder	Area (mm^2)	Gate count
EM-based decoder (64-bit EMs)	7.890	1437 K
EM-based decoder (32-bit EMs)	4.604	838 K
TFM-based decoder (8-bit TFMs)	2.841	517 K

EMs for decoding state-of-the-art LDPC codes while having much lower silicon area consumption. This chapter also showed that TFMs significantly reduce the overall area of ASIC implementations of stochastic LDPC decoders. The silicon area consumption of the proposed ASIC TFM-based (1056,528) LDPC decoder is 40% and 65% less than the area consumption of ASIC EM-based LDPC decoders with 32-bit EMs and 64-bit EMs, respectively.

CHAPTER 5

Node-Based Rerandomization Using Majority-Based Tracking Forecast Memories

The TFM approach is much more area-efficient compared to the EM approach. However, both EM and TFM approaches rely on the edge-based rerandomization where a rerandomization unit is assigned to each outgoing edge of a stochastic VN (see Figure 4–1). Therefore, the total number of TFMs in a stochastic LDPC decoder is equal to the number of edges in the corresponding LDPC code's factor graph, and even though the hardware-complexity of a TFM is much less than an EM in ASIC, the total number of TFMs is still the bottleneck of the overall hardware-complexity of ASIC TFM-based decoders. In this chapter, we propose node-based rerandomization using the MTFM stochastic decoding approach. In this approach, instead of assigning one TFM per outgoing edge, each VN uses only one MTFM as its rerandomization unit (see Figure 5–1). This significantly reduces the total number of rerandomization units used in a stochastic LDPC decoder. For example, the number of rerandomization units in a decoder that decodes a regular (n, k) LDPC code with degree-d_v VNs, reduces from $n \times d_v$ to n. For the case of the (2048,1723) LDPC code from the 10 Gb/s Ethernet (10GBASE-T) standard [2], the number of rerandomization units reduces from $2048 \times 6 = 12288$ to 2048.

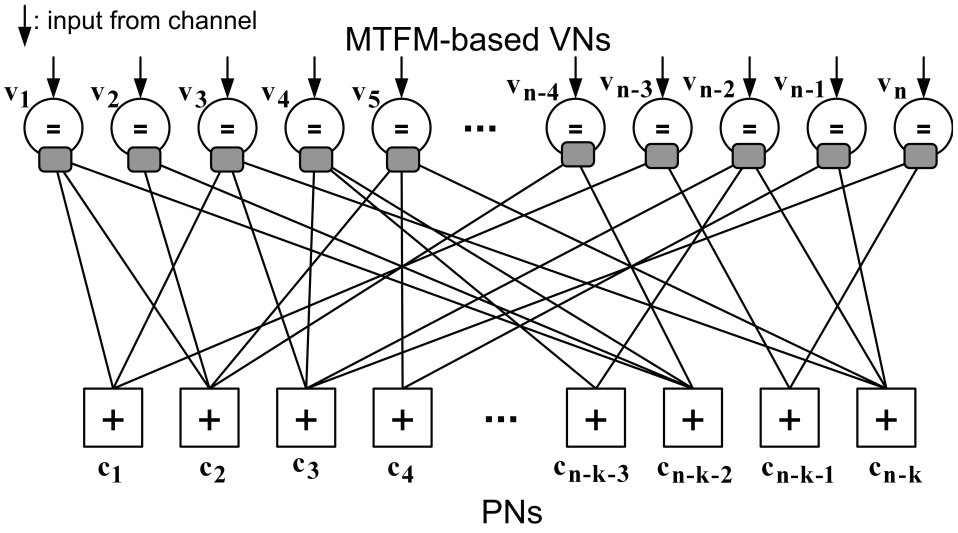


Figure 5–1: An MTFM-based stochastic decoder uses one MTFM per VN.

We refer to this approach as the MTFM approach, since it relies on the TFM approach, but it has a different update rule based on the majority of outgoing regenerative bits in a stochastic VN. As will be shown in this chapter, the MTFM approach significantly reduces the hardware-complexity of stochastic LDPC decoders. We apply this approach to implement an ASIC MTFM-based stochastic decoder that decodes the (2048,1723) LDPC code from the 10 Gb/s Ethernet standard [2]. To the best of our knowledge, this MTFM-based stochastic decoder is the most silicon area-efficient fully parallel soft-decision LDPC decoder reported in the literature.

5.1 Majority-Based Tracking Forecast Memories

To explain the concept of node-based rerandomization, Figure 5–2 depicts the block diagram of a degree-6 MTFM-based stochastic VN. This VN receives the in_{CH} bit (from a comparator that converts the corresponding channel probability to a stochastic stream) and 6 input bits, in_0 to in_5, from the six connected PNs. The corresponding output bits are out_{CH} and out_0 to out_5. The final output of the VN, out_{CH}, is determined by the majority of bits received from connected PNs. The structure of the VN is based on the

cascaded subgraphs of degree-3 and degree-2 subnodes in which each degree-3 subnodes uses 2-bit IMs. The VN uses one MTFM. The (final) input stream of the MTFM is the majority of the VN's outgoing regenerative bits for all edges. It is important to note that at a given time the VN might be in a nonhold state for some of its edges and be in the hold state for the rest of the edges.[1] Therefore, some of the input bits of the MTFM might be regenerative and the rest are conservative bits. For this reason, each degree-2 subnode sends two bits, $r_i(t)$ and $s_i(t)$, to the MTFM, where $0 \leq i \leq d_v - 1$. $s_i(t)$ determines if the VN is in the hold state or nonhold (regular) state for edge i. Also, $r_i(t)$ is the output bit of the subnode which can be regenerative or conservative.

The MTFM of the degree-6 VN computes the majority of $r_0(t)$ to $r_5(t)$ bits if they are regenerative. The MTFM-based VN operates as follows:

- When the VN is in a nonhold (regular) state for edge i, it directly uses the corresponding regenerative bit as the outgoing bit of the edge.
- In case that the VN is in the hold state for the edge i, it refers to the MTFM and uses its output bit as the outgoing bit.

Different thresholds might be exploited for (i) updating an MTFM and (ii) for calculating the majority of regenerative bits in a MTFM. For example, the MTFM might be updated only when at least a certain percentage of its input bits are regenerative, and the majority criteria might be set to "$> 50\%$" or "$> 75\%$" etc.

[1] Note that a VN is in the hold state for an edge, when the input bits of the final/exiting subnode for that edge are not equal. Therefore, the VN in Figure 5–2 is in the hold state for edge i when the two input bits of the degree-2 subnode i are not equal.

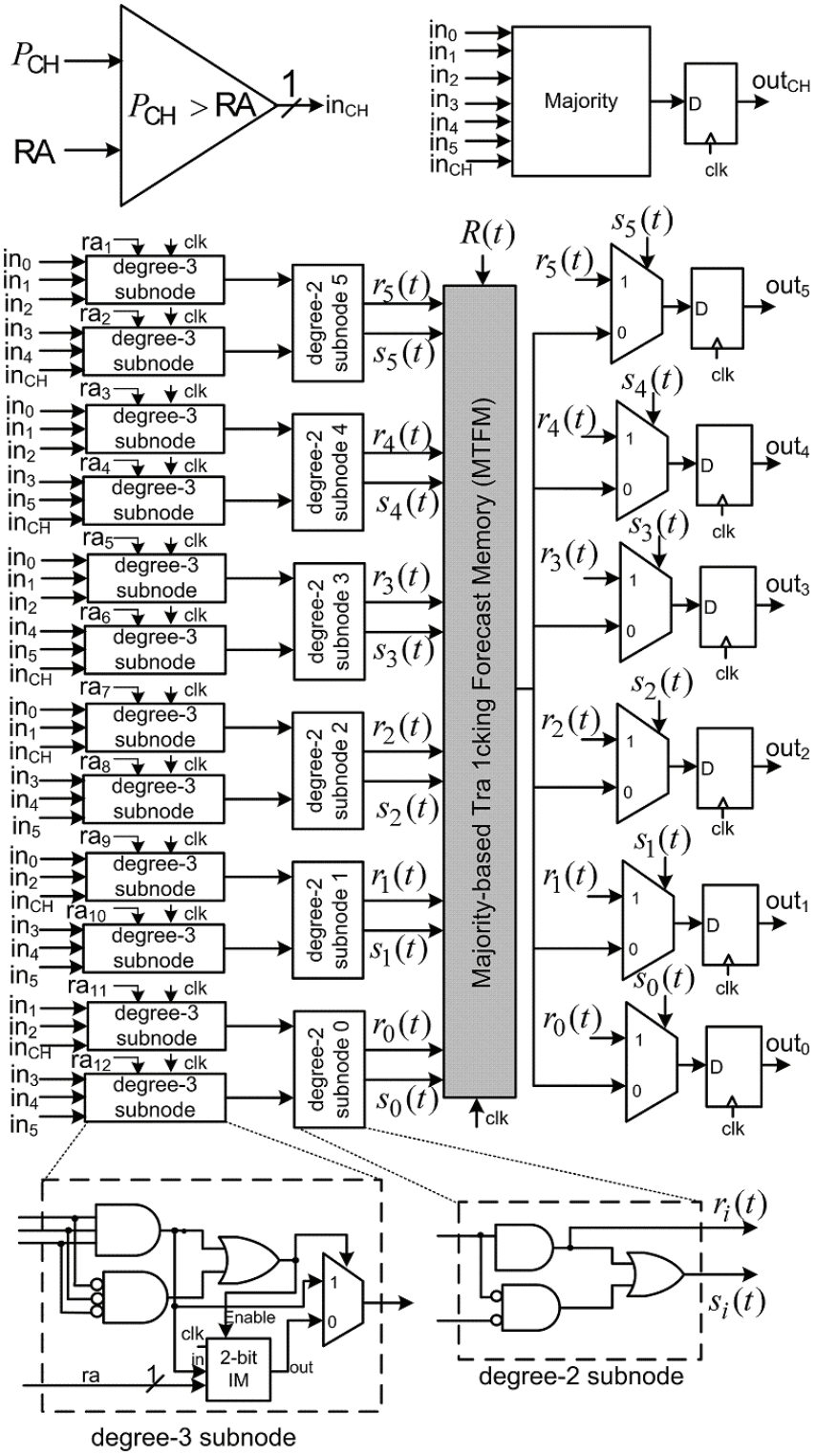


Figure 5–2: The structure of a degree-6 MTFM-based stochastic VN. P_{CH} is the channel probability, $R(t)$ and RA are (pseudo) uniform random numbers, ra_i is a random bit, and IM refers to internal memory.

5.2 Hardware Realization of MTFMs

This section discusses the hardware realization of MTFMs and how the choice of majority and update thresholds can affect the hardware-complexity of MTFMs.

5.2.1 General Architecture

Figure 5–3 shows the general architecture of MTFM for a degree-d_v VN. Note that the TFM block in the figure can use any TFM architecture discussed in Section 4.2. We recall that $r_i(t)$ is the output bit of the VN for the i-th edge. Also, $s_i(t)$ determines the state of the VN for the i-th edge: $s_i(t) = 0$ means that the VN is in the hold state, and $s_i(t) = 1$ means that the VN is in a nonhold state for the i-th edge and $r_i(t)$ is regenerative.

The MTFM architecture shown in the figure uses T_m as a majority threshold and it uses T_u as a (fixed) threshold for updating the TFM. The MTFM calculates $S(t) = \sum_{i=1}^{d_v} s_i(t)$ which determines the number of input bits that are regenerative. It also computes $X(t) = \sum_{i=1}^{d_v} s_i(t)r_i(t)$ which determines how many of these regenerative bits are equal to 1 (note that when $s_i(t) = 0$, $r_i(t)$ is not regenerative and it is forced to be 0 by using an AND gate at the input). The comparator in this architecture applies the majority criterion according to the majority threshold, T_m. In this respect, T_m is usually set to $S(t)/2$ which implies that the output bit of the comparator, $r(t)$, is equal to 1 when more than half of the regenerative bits at time t are 1, and it is equal to 0, otherwise.

The TFM is only updated when U = 1 which indicates that $S(t) > T_u$, i.e., the number of input regenerative bits is greater than T_u. In general, T_u is a fixed integer ($0 \leq T_u \leq d_v$) whose value can be chosen based on the decoding performance of the decoder for different T_u. As shown in Figure 5–2, the MTFM-based VN uses the output bit of the MTFM, $r'(t)$, as the outgoing

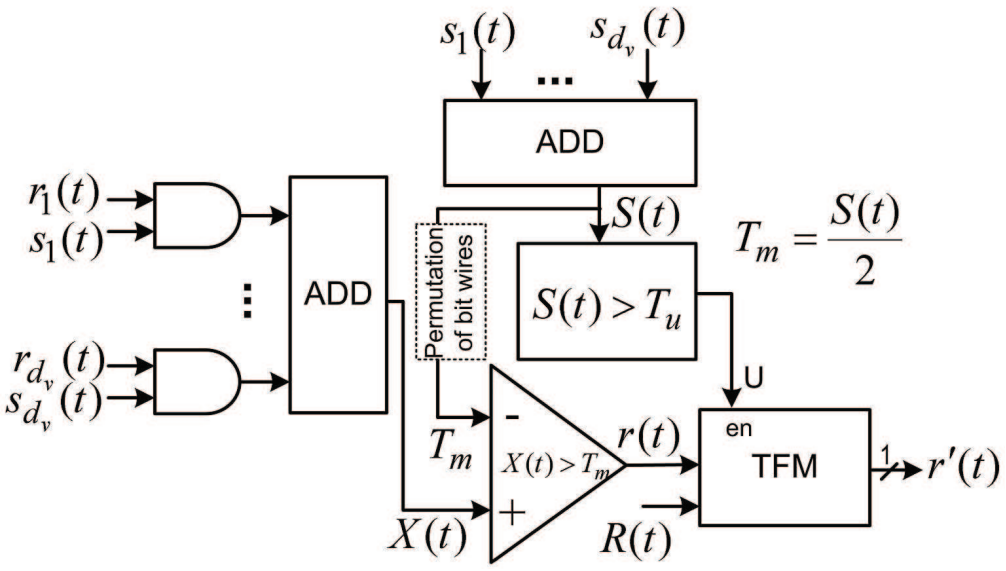


Figure 5–3: General architecture of an MTFM. T_u is a fixed threshold for updating the TFM and T_m is the majority threshold.

bit for any edge that is in the hold state; however, if edge i is in a nonhold state it directly uses $r_i(t)$ as the outgoing bit for that edge.

5.2.2 Reduced-Complexity Architecture

The hardware-complexity of an MTFM can be significantly reduced by properly adjusting T_m and T_u. Figure 5–4 shows the reduced-complexity implementation of architecture of an MTFM. In this structure, the TFM is only updated when all the input bits are regenerative (i.e., $T_u = d_v$); therefore, the update signal for the TFM, signal U, can be determined by a d_v-input AND gate (instead of a comparator as in Figure 5–3). Also, T_m is set to $d_v/2$, hence, the most significant bit of $X(t)$ directly determines the majority and if the input bit of the TFM, $r(t)$, is 0 or 1. Compared to the general architecture of MTFMs in Figure 5–3, the reduced-complexity MTFM uses one fewer adder and two fewer comparators. The reduced-complexity MTFM architecture is used for ASIC implementation of the (2048,1723) LDPC stochastic decoder, which is discussed in Section 5.4.

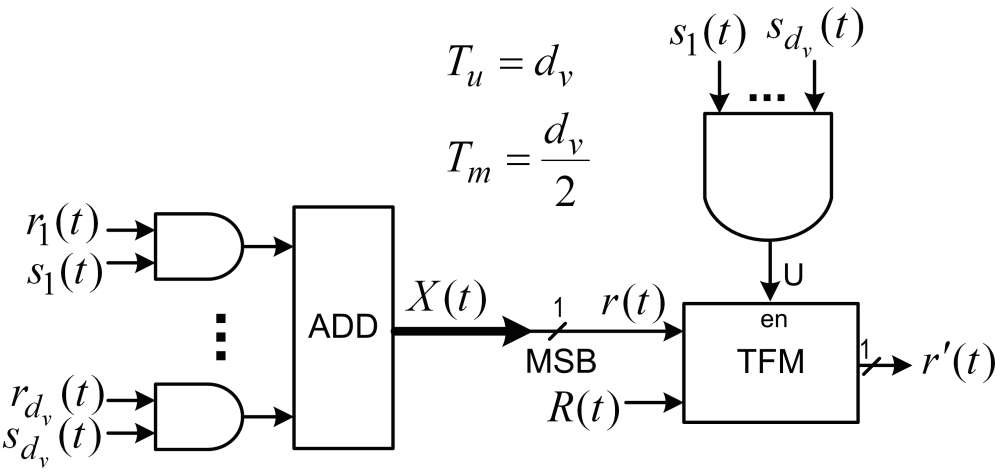


Figure 5–4: Architecture of a reduced-complexity MTFM. $r(t)$ is the most significant bit (MSB) of $X(t)$.

5.3 Comparison of the Hardware-Complexity and Decoding Performance of MTFMs with EMs and TFMs

In an MTFM-based stochastic VN, the output bit for an edge is determined by the MTFM only when the edge is in the hold state. In nonhold (regular) states, the output bit for an edge is directly determined by the newly regenerative bit in both TFM and MTFM approaches. In this regard, the majority approximation made in the MTFM approach is only effective when an edge is the hold state, and no approximation is made in nonhold (regular) states. The majority approximation used in MTFM approach is precise when the degree of the VN is high (usually, a degree of 4 or more). For instance, Figure 5–5 shows the output probability of an edge in degree-6 TFM-based and MTFM-based VNs and compares them with the ideal target output probability (computed according to the floating-point SPA). Both VNs receive the same input stream. As shown, the extracted output probabilities in both approaches closely follow the SPA's output probability.

Figure 5–6 compares the performance of the MTFMs with EMs and TFMs for decoding the (2048,1723) LDPC code with degree-6 VNs and degree-32

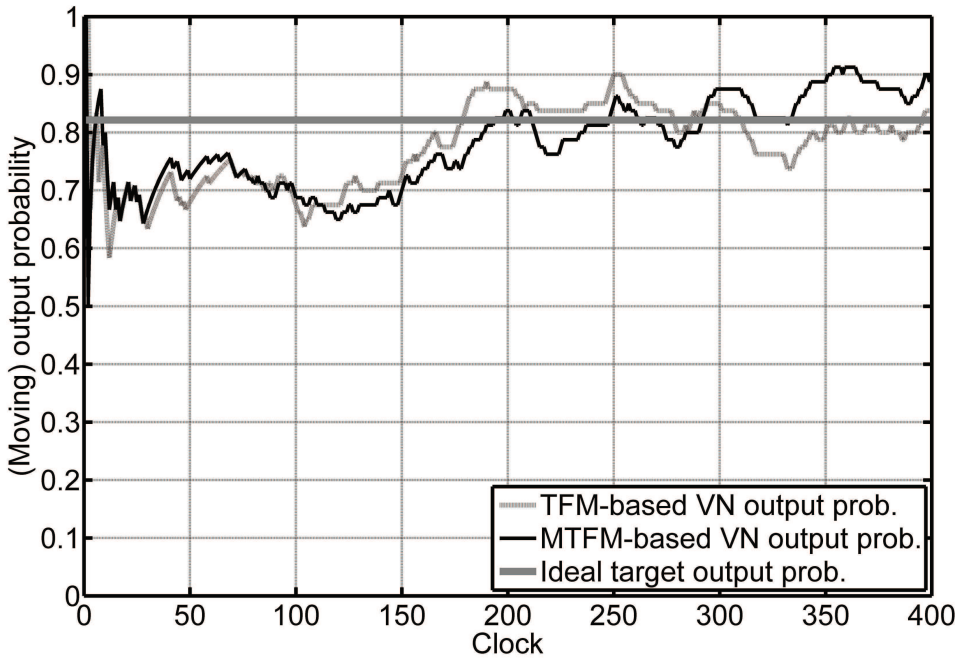


Figure 5–5: Extracted output probability of an edge in degree-6 TFM-based and MTFM-based VNs. Both VNs receive the same input stream.

PNs. Note that results reported for the MTFM approach in this section are based on the reduced-complexity MTFM architecture that uses a reduced-complexity TFM (see Figure 5–4). For all of these approaches, symbols received from the channel are quantized to 6 bits and an early decoding termination (based on syndrome checking) until a maximum number of 400 decoding cycles is used. As shown, the reduced-complexity MTFM approach provides similar decoding performance compared to the TFM approach which outperforms 64-bit and 32-bit EMs for decoding the (2048,1723) LDPC code. The performance loss of the MTFM approach compared to the SPA with floating-point implementation and 32 iterations is about a 0.2 dB loss.

Table 5–1 shows the silicon area consumption, 2-input NAND gate count, and the maximum achievable clock period for degree-6 stochastic VNs and a degree-32 stochastic PN in CMOS 90nm technology. Results shown in the first section of the table are obtained for synthesizing for the maximum possible

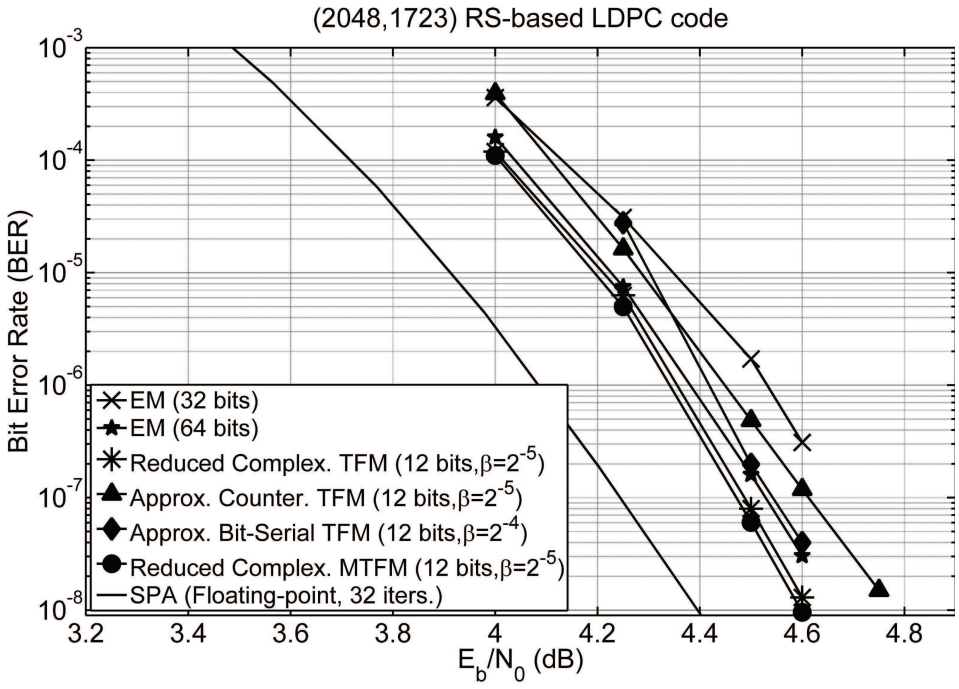


Figure 5–6: Comparison of decoding performance of EM-based, TFM-based, and MTFM-based stochastic decoding approaches.

speed. These results confirm that stochastic nodes are able to operate very fast, with clock frequencies beyond 2.5 GHz. As shown in the second section of the table, by synthesizing the modules for a lower target clock frequency, lower silicon area consumption can be obtained. With respect to the area consumption, the second section of the table shows that the reduced-complexity MTFM approach results in significant area reduction when compared to the TFM and EM approaches. The area of a degree-6 MTFM-based VN is about 32% of the area of a reduced-complexity TFM-based VN, 15% of the area of a 64-bit EM-based VN, and 27% of the area of a 32-bit EM-based VN. The hardware-complexity of an MTFM-based VN and a stochastic PN operating at 500 MHz (2 ns clock period) is equivalent to the complexity of 351 and 79 two-input NAND gates, respectively.

Since the main difference between stochastic approaches is in the VN architecture, it is possible to approximately compare the area efficiency of

Table 5–1: Hardware-complexity of degree-6 VNs and degree-32 PN in CMOS 90nm technology.

Module and Architecture	Area in μm^2 & Gate count	Clock period in ps	Area in μm^2 & Gate count at 500 MHz
EM-based VN (32 bits)	(13860, 2525)	283	(7133, 1299)
EM-based VN (64 bits)	(22575, 4112)	287	(12255, 2232)
TFM-based VN (16 bits bit-serial)	(9223, 1670)	263	(4352, 793)
TFM-based VN (12 bits counter-based)	(9111, 1660)	312	(5318, 969)
TFM-based VN (12 bits reduced complex.)	(14989, 2730)	354	(5924, 1079)
MTFM-based VN (12 bits reduced complex.)	(4329, 789)	350	(1927, 351)
Stochastic PN	(1415, 258)	283	(431, 79)

LDPC decoders implemented using these approaches from Table 5–1. For example, we can estimate that the implementation of a (2048,1723) LDPC decoder using the 12-bit MTFM approach in CMOS 90 nm technology and with maximum 500 MHz clock frequency results in saving of about $2048 \times (5294 - 1927) = 6.89$ mm^2 silicon area compared to 12-bit reduced-complexity TFM approach, and about $2048 \times (12255 - 1927) = 21.15$ mm^2 silicon area compared to the 64-bit EM approach.

5.4 A (2048,1723) Fully Parallel MTFM-based Stochastic LDPC Decoder

This section discusses the ASIC implementation of a fully parallel MTFM-based stochastic decoder that decodes the (2048,1723) LDPC code from the IEEE 802.3an standard. This LDPC code is a regular RS-based code [30] with degree-6 VNs and degree-32 PNs. It is adopted for the standard to provide enough coding gain to allow for a BER level of 10^{-12} or less. To demonstrate the applicability of the MTFM approach to decode other LDPC codes, we also

provide the performance of the MTFM approach for decoding a (1057,813) LDPC code at low BERs in Section A.3 of Appendix A.

5.4.1 Decoder Architecture and Specifications

The implemented fully parallel stochastic decoder instantiates 2048 MTFM-based degree-6 VNs and 384 degree-32 PNs based on the partitioned design shown in Figure 5–7. The decoder uses a flooding-schedule for updating VNs and PNs. The binary parity-check matrix of the (2048,1723) LDPC code is based on the permutation of 64×64 sub-matrices. This parity-check matrix is not full-rank and has 384 degree-32 PN. Each PN has exactly one connection to a VN in every 64 columns. Therefore, it is possible to partition the whole parity-check matrix into 32 VN blocks in which each block has 64 degree-6 VNs (see Figure 5–7). In this configuration, each block receives $64 \times 6 = 384$ input bits from each one of its neighbor blocks and outputs 384 bits to each of them. To form the parity-check equation, each VN inside a block XORs its output bit with the input received from the neighboring block and passes it to the next neighboring block. The VN also XORs the input bits received from the neighbor blocks to from its input bit. This method of partitioning relies on the split-row technique for MSA-based decoders first introduced in [65]. However, we note that compared to the split-row technique for the MSA it has the following major benefits:

- In the split-row technique, increasing the number of splits/partitions results in decoding performance loss and, possibly, a higher error-floor [65–67]. Recently, MSA-based threshold decoding methods have been proposed for the split-row technique to reduce this performance loss (e.g., see [67]). In stochastic decoders, however, partitioning PNs does not affect the decoding performance or the error-floor. This is because in the split-row technique for the MSA, each PN is divided into lower degree

PNs (assigned to each partition). These lower degree PNs calculate the minimum of only a portion of incoming messages. Therefore, their outputs are not necessarily the absolute minimum of all incoming messages received by the PN. As the number of splits increases, the approximation made in lower degree PNs becomes less accurate. In stochastic decoding, the PN operation is XOR-based, hence, as shown in Figure 5–7, each lower degree PN can send its 1-bit outcome to neighboring PNs and; therefore, no approximation is made in the PN operation.

- Increasing the number of partitions/splits results in long physical wires between VN blocks. These wires can become the bottleneck of the clock frequency and throughput. For instance in Figure 5–7 the input signal of the VNs in the block number 32 starts from block 1 and passes through 30 VN blocks before reaching block 32. Stochastic decoding benefits from asynchronous pipelining (discussed in Section 3.5.1.6). Asynchronous pipelining enables stochastic decoders to pipeline long wires with negligible effect on the average number of decoding cycles and throughput. By relying on this useful feature, non-uniform levels of registers can be inserted to pipeline the signals between VN blocks and break long wires into small pieces to increase the clock frequency and throughput. In the implemented decoder, three levels of flip-flops (i.e., one level of flip-flop after every 8 VN block) are used to break these wires.

The decoder receives 6-bit input symbols from the channel. It applies scaling (with a scaling factor of $\gamma = 1.33$) and converts the input symbols to 7-bit probabilities using $2^6 \times 7$ bits (56 bytes) look-up tables. These probabilities are converted to stochastic streams using 7-bit comparators. The MTFM resolution in VNs is 11 bits and all the MTFMs are initialized to the corresponding channel probabilities prior the start of the decoding operation.

5.4.1.1 Random Number Generation

The stochastic decoder requires (pseudo) random numbers at input (channel) comparators to convert probabilities to stochastic streams. Random numbers are also required in MTFMs to convert probabilities stored in MTFMs to stochastic streams. Random numbers used in the decoder are generated using a DRE architecture which consists of 64 independent random engines. Every two random engines are assigned to a VN block. Each random engine consists of four 16-bit LFSRs associated with different prime polynomials and generates an 11-bit random number which is shared among $2048/64 = 32$ VNs.

5.4.1.2 Early Decoding Termination Criterion

The VNs and PNs exchange bits until decoder output bits satisfy all the parity checks (i.e., syndrome checking) or a maximum number of 400 decoding cycles is exhausted (see Section 5.4.1.3). In the implemented decoder, each decoding cycle takes one clock cycle. The final output of each VN is determined by the majority of bits received from connected PNs. The syndrome checking is performed in every clock and hence the decoder stops decoding as soon as it finds a valid codeword. The syndrome check is performed by XORing the output bits of VNs that are connected to the same PNs. If the outcome of all of these XOR gates are zero, decoding terminates. The early termination logic consists of 384 32-input XOR gates whose 384 output bits are passed through a 384-input NOR gate or equivalently 383 2-input NOR gates in a tree configuration. Because of the early termination criterion, the throughput of the decoder is determined by the average number of decoding cycles used and the decoding latency is determined by the maximum number of decoding cycles used.

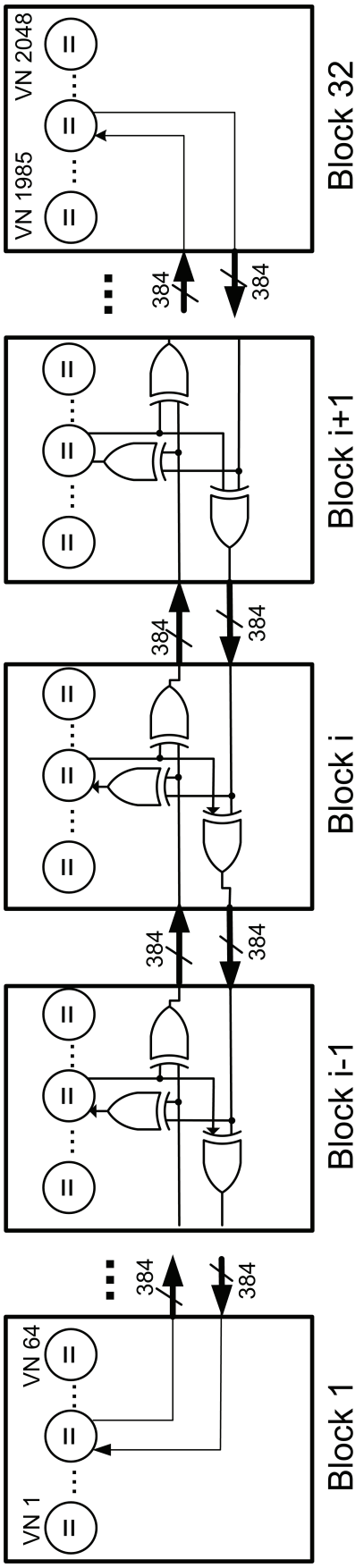


Figure 5–7: The (2048,1723) stochastic decoder is implemented using 32 VN blocks in which each block contains 64 degree-6 VNs. Each block receives 384 input bits from each one of its neighbor blocks and outputs 384 bits to each of them. To form the parity-check equation, each VN inside a block XORs its output bit with the input receives from the neighboring block and passes it to the next neighboring block. The VN also XORs the inputs received from neighbor blocks to from its input bit. One level of flip-flops is used after every 8 VN blocks to break long wires.

5.4.1.3 Redecoding and Postprocessing

The idea of combining different decoding algorithms, for the sake of better performance or less latency, has been used in the literature (e.g., see [10]). The ASIC stochastic decoder uses a combined redecoding [53] and postprocessing scheme which lowers the error-floor of the 10GBASE-T LDPC code and enables the decoder to achieve a good BER decoding performance with less decoding latency. Redecoding [53] is an interesting feature of stochastic decoders which is useful for lowering the error-floor of LDPC codes. In stochastic decoding the decoding trajectory depends on the stream of random numbers generated for conversion of probabilities to stochastic bit streams. Consequently, by using different sequences of random numbers, different decoding trajectories are possible. Therefore, if the decoding outcome does not converge to a codeword after some decoding cycles, it is possible to restart the decoding operation with different random numbers (a different decoding trajectory) to possibly converge to a codeword in the new round. For the case of the (2048,1723) LDPC code, it is known that the dominant error events in the error-floor region are due to (8,8) absorbing sets (e.g., see [108–110]). Redecoding in this respect helps to reduce these events by randomizing the decoding trajectory.

The ASIC stochastic decoder uses 4 rounds of decoding for $E_b/N_0 \geq 5$ dB in which each decoding round uses a maximum number of 100 decoding cycles (i.e., a maximum of 400 decoding cycles including redecoding and postprocessing). In rounds 1 to 3 of decoding, the stochastic decoding is performed for 92 decoding cycles. In the last 8 decoding cycles of these decoding rounds, the ASIC decoder uses a postprocessing scheme to correct the remaining bit errors. During the postprocessing mode, the output bit of each VN is directly sent to PNs. The PNs perform the parity-check operation and send back their bit

messages to VNs. This postprocessing scheme can efficiently correct remaining bit errors provided they are few. When the number of bit errors are high, this scheme may result in propagation of errors in the entire graph; therefore, the ASIC decoder only uses this postprocessing scheme at the end of decoding rounds 1, 2 and 3 and does not use it for the last round. If the syndrome check is not satisfied during decoding rounds 1 to 3, all MTFMs are reset to the corresponding channel probabilities and the next round of decoding is started. In the last round of decoding (round 4), stochastic decoding is performed for a maximum of 100 decoding cycles.

5.5 Performance and Tradeoffs

This section discusses the decoding performance as well as various characteristics of the implemented MTFM-based stochastic LDPC decoder.

5.5.1 Decoding Performance

Figure 5–8 depicts the BER and Frame-Error-Rate (FER) decoding performance of the MTFM-based stochastic decoder. The BERs of the MTFM-based decoder at $E_b/N_0 = 5.00$ dB and $E_b/N_0 = 5.15$ dB are obtained by counting 30 and 10 frame errors, respectively. For $E_b/N_0 \leq 4.75$ dB at least 100 frame errors are counted.[2] For the sake of comparison, the figure shows the decoding performance of MTFM-based stochastic decoding without postprocessing and redecoding (with the same maximum number of decoding cycles), the floating-point sum-product with 32 iterations, 6-bit fixed-point sum-product with 50 iterations from [108], and 4-bit fixed-point offset min-sum from [108]. Also shown in the figure are the decoding performance of the

[2] To simulate a very low BER, we used two Canadian clusters (namely, WestGrid and CLUMEQ) and a local cluster available at our research group. Our simulations took about two months and were distributed on about 1000 CPUs.

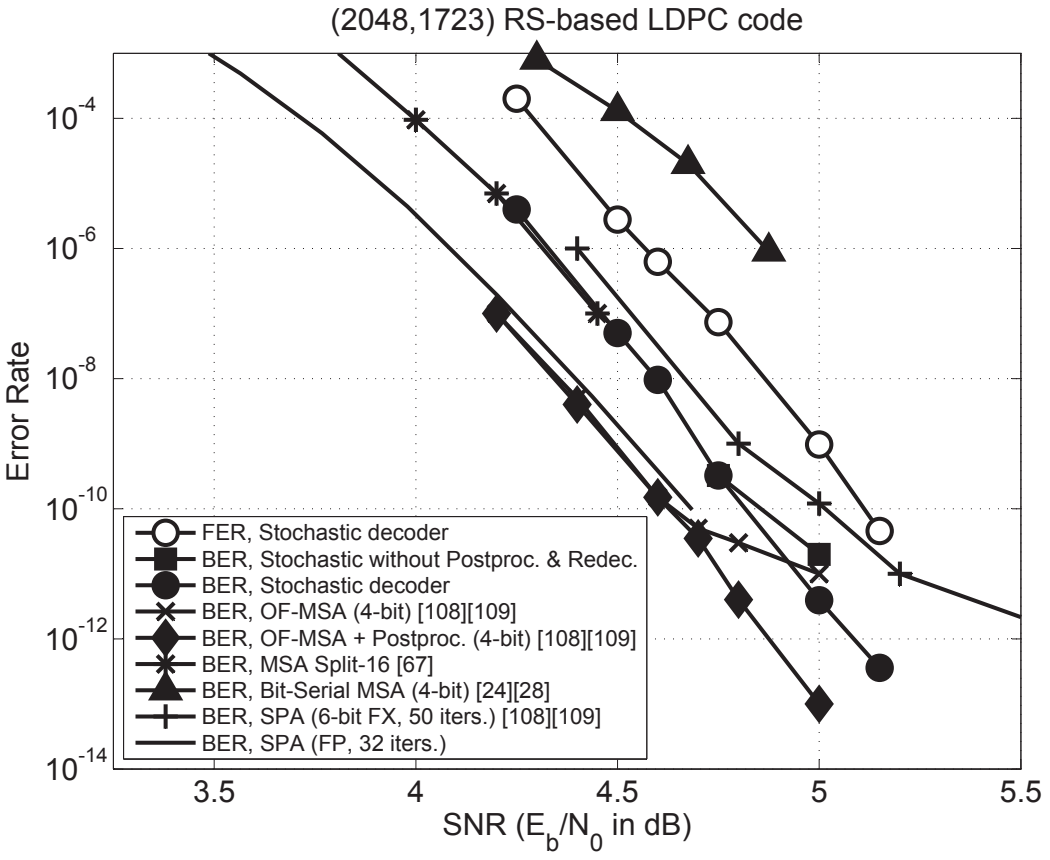


Figure 5–8: Decoding performance of the MTFM-based stochastic decoder. The stochastic decoder uses early termination until a maximum of 400 decoding cycles has been exhausted.

(2048,1723) ASIC LDPC decoder in [24] which uses the approximate bit-serial MSA, the (2048,1723) MSA Split-16 decoder in [67], and the (2048,1723) ASIC LDPC decoder in [108, 109]. The decoder in [67] is a recently proposed fully parallel decoder which provides a maximum throughput of 92.8 Gb/s by using an early termination criterion. The decoding performance of this decoder is demonstrated down to a BER of about 10^{-7} in [67]. The decoder in [108, 109] is also a recently proposed partially parallel decoder that relies on the 4-bit fixed-point offset MSA and a special postprocessing technique [110] to lower the error-floor of the (2048,1723) LDPC code. It also uses an early termination criterion to increase the decoding throughput.

Table 5–2: Summary of the ASIC implementation results for the (2048,1723) MTFM-based stochastic LDPC decoder.

CMOS technology	CMOS90nm-GPSVT (7 metal layers)
Supply voltage	1 V
Clock frequency	500 MHz
Decoding latency	800 ns
Throughput	61.3 Gb/s at $E_b/N_0 = 5.5$ dB
Core area	6.38 mm^2
Logic utilization	86% (initial) and 95% (final)

As shown Figure 5–8, the proposed MTFM-based stochastic decoder is able to provide a BER of 4×10^{-13} at $E_b/N_0 = 5.15$ dB. Compared to the 4-bit offset MSA (without postprocessing) and the 6-bit SPA, the stochastic decoder shows superior error-floor behavior since no error-floor is observed down to the BER of 4×10^{-13}. This decoder outperforms the bit-serial decoder in [24, 28] by about 1 dB and 6-bit sum-product decoding algorithm by about 0.4 dB. Compared to the decoder in [108, 109] with postprocessing, the MTFM-based decoder has about 0.2 dB loss. The stochastic decoder and the decoder in [67] have similar decoding performance down to a BER of about 10^{-7} in the water fall region.

5.5.2 Implementation Characteristics and Hardware-Complexity

Table 5–2 summarizes the ASIC implementation characteristics of the MTFM-based stochastic decoder. The decoder is implemented using GP 90nm CMOS technology (standard Vt) from STM with 1 V supply voltage. It is synthesized using Cadence RTL-Compiler in nominal process corner and place and route was done by using Cadence Encounter. The decoder achieves a maximum clock frequency of 500 MHz after the place and route step and its core occupies 6.38 mm^2 silicon area with a high final logic utilization of 95%. Figure 5–9 shows the chip layout after the place and route step.

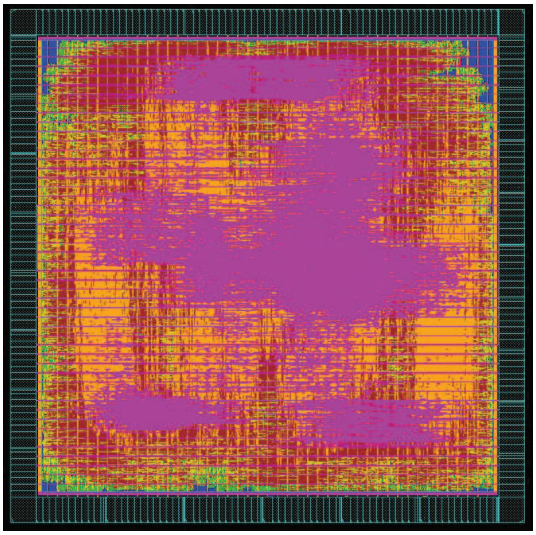


Figure 5–9: The stochastic decoder chip layout.

5.5.3 Throughput

Figure 5–10 depicts the average number of decoding cycles used (left y-axis) for decoding at different E_b/N_0 and the corresponding throughput (right y-axis) with the achieved clock frequency of 500 MHz after the place and route step. As mentioned before, the stochastic decoder uses early termination criterion and the core throughput of the decoder is determined by the average number of decoding cycles used. At $E_b/N_0 = 5.50$ dB the MTFM-based stochastic decoder uses an average of 16.7 decoding cycles which results in a core throughput of 61.317×10^9 b/s or 61.3 Gb/s (each decoding cycle takes one clock cycle). The throughput of the decoder at $E_b/N_0 = 5.15$ dB is about 49.4 Gb/s. Also, as Figure 5–10 shows, the decoder is able to provide throughput of more than 23 Gb/s for $E_b/N_0 > 4.5$ dB (BERs less than 10^{-7}, approximately). Figure 5–11 shows histograms of the decoding cycles used for decoding at different E_b/N_0. The figure also shows the corresponding mean (μ) and the standard deviations (σ) associated with each histogram. Each histogram is based on observation of one million transmitted codewords. As shown, at high E_b/N_0 the majority of transmitted codewords are decoded very

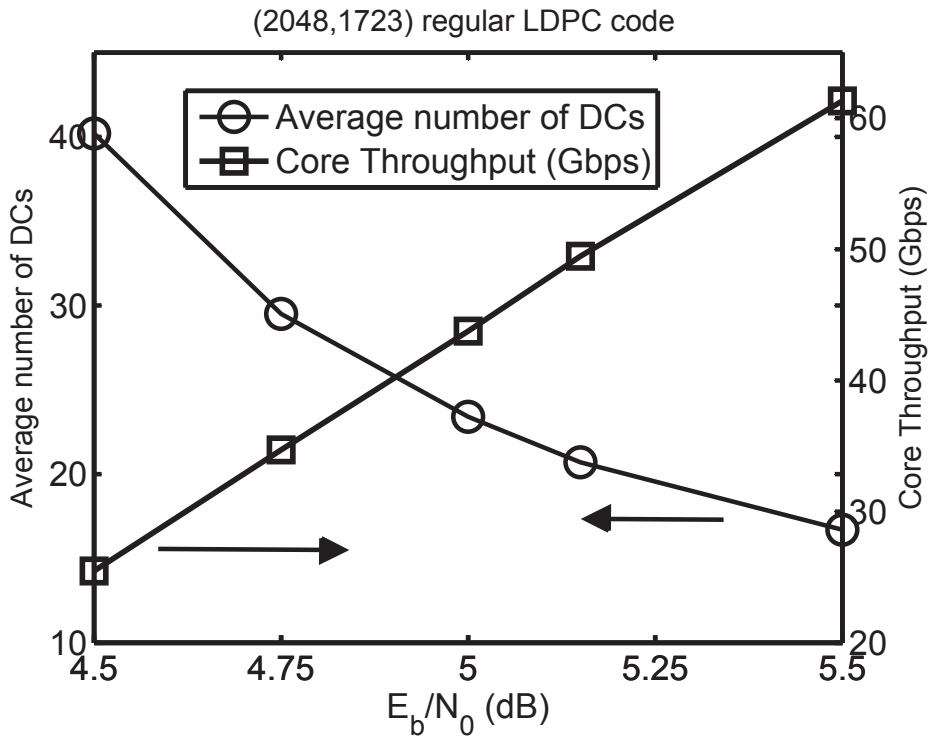


Figure 5–10: Average number of decoding cycles used (left y-axis) for decoding at different SNRs and the corresponding core throughput (right y-axis) for the achieved clock frequency of 500 MHz. Each decoding cycle takes one clock cycle.

fast and hence the histograms become highly concentrated near the average number of decoding cycles used.

5.5.4 Latency

The latency of the MTFM-based decoder is determined by the maximum number of decoding cycles used. As mentioned before, the decoder uses a maximum of 400 decoding cycles (including postprocessing), hence, with the clock frequency of 500 MHz, the decoder maximum latency is 800 ns. If lower latency is required, it is possible to trade BER with the maximum number of decoding cycles used. For example, Figure 5–12 shows the BER/FER versus latency (in nanoseconds) tradeoff at $E_b/N_0 = 5.15$ dB. As shown, at this SNR a BER of about 10^{-12} can be achieved with 580 ns decoding latency (i.e., maximum 290 decoding cycles).

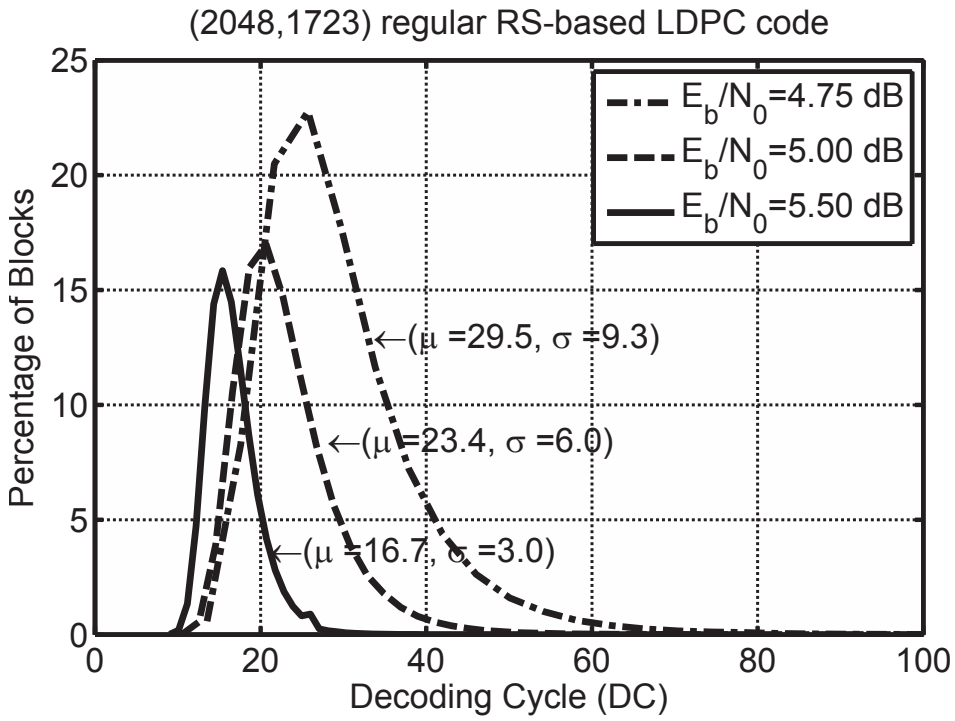


Figure 5–11: Histograms of decoding cycles used for decoding codewords at different SNRs. One million codewords used for each histogram. μ is the average number of decoding cycles and σ is the standard deviation. Each decoding cycle takes one clock cycle.

5.5.5 Input and Output Buffer Requirements

Because of the early termination used in the decoder, the proposed stochastic decoder needs to use input and output buffers to accommodate the difference between the variable number of decoding cycles used and the maximum number of decoding cycles. The distribution of the number of decoding cycles used changes with E_b/N_0. Based on the histograms of the number of decoding cycles used (see Figure 5–11), it is possible to determine the buffer requirements of the stochastic LDPC decoder for a specific operating E_b/N_0. In particular, we would like to determine such requirements for low BER regimes (high E_b/N_0) where the decoder is supposed to operate.

The input buffer of the decoder receives a codeword from the channel every T_{IN} decoding cycles. Each received codeword occupies 2048×6 bits or 1.5K

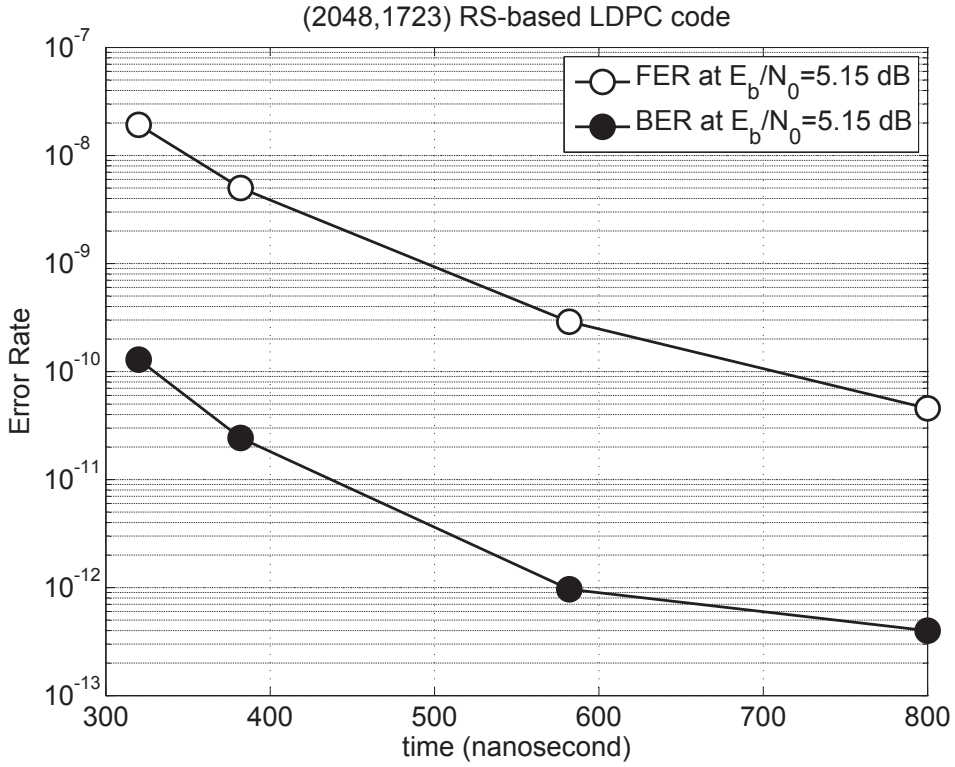


Figure 5–12: Decoding performance versus latency (in nanoseconds) at $E_b/N_0 = 5.15$ dB. A BER of about 10^{-12} is achieved with about 580 ns maximum decoding latency (i.e., maximum 290 decoding cycles). The shown 800 ns latency corresponds to the maximum 400 decoding cycles with the achieved 500 MHz clock frequency. Each decoding cycle takes one clock cycle.

bytes. The number of decoding cycles used by the decoder for decoding, T_D, is a random variable where $0 < T_D \leq 400$ and its probability distribution function is obtained by the histograms shown in Figure 5–11. Every T_D decoding cycles, the decoder takes a codeword from the input buffer. On the other hand, the output buffer of the decoder receives a decoded codeword from the decoder every T_D decoding cycles and outputs a codeword every T_{OUT} decoding cycles. Note that each decoded codeword occupies 2048 bits or 256 bytes which is much less than a received codeword. Figure 5–13 shows the simulation results for the probability of codeword overflow at $E_b/N_0 = 5.15$ dB for different input and output buffer sizes ($T_{\text{IN}} = 128$ and $T_{\text{OUT}} = 16$). As shown, the codeword overflow probability sharply decreases by increasing the size of the input and

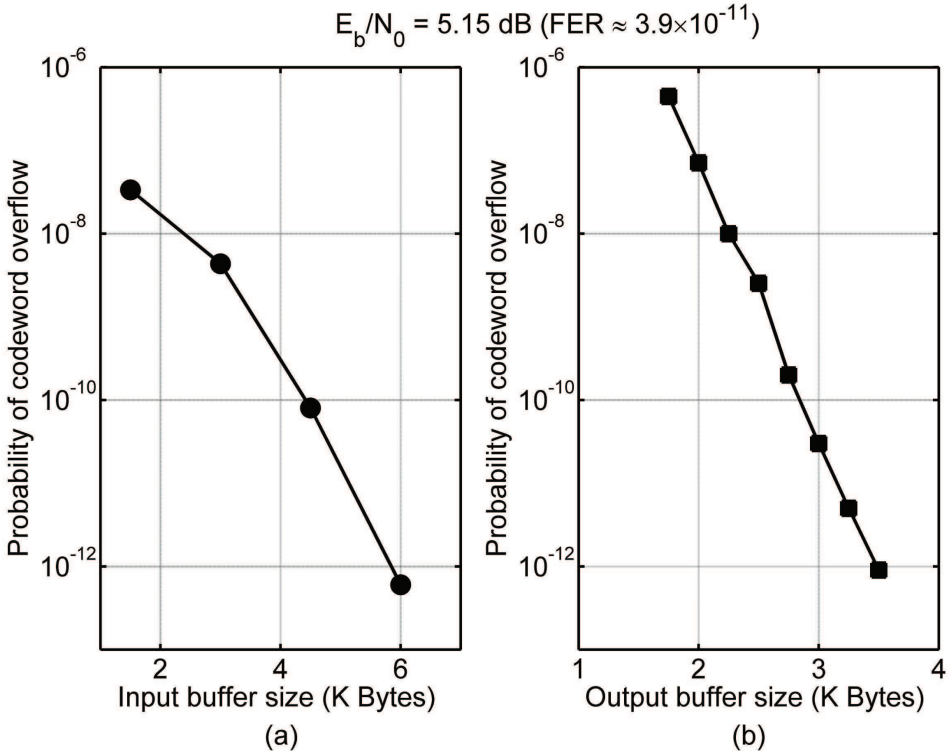


Figure 5–13: Probability of codeword overflow for different sizes of (a) input buffer and (b) output buffer at $E_b/N_0 = 5.15$ dB.

output buffers. This is mainly because the probability distribution function of the number of decoding cycles used is highly concentrated near the average. For example, at $E_b/N_0 = 5.15$ dB with a FER of about 3.9×10^{-11}, the input and output buffers require about 10K bytes in total (roughly about 0.28 mm^2 in CMOS 90nm technology) to have a codeword overflow probability that is below the FER, induced by the channel, by more than an order of magnitude.

5.6 Comparison with State-of-the-Art ASIC LDPC Decoders

Table 5–3 compares the implemented MTFM-based stochastic decoder with some state-of-the-art high-throughput soft-decision ASIC LDPC decoders. The table is partitioned in two parts: first, ASIC decoders that decode the (2048,1723) LDPC code from the IEEE 802.3an standard and second, ASIC decoders that decode other LDPC codes.

With respect to the decoding latency, the stochastic decoder requires 800 ns latency which is higher than other (2048,1723) LDPC decoders in the table, but it is within an acceptable latency range (< 1 μs) for a channel decoder. Also, if lower latency is desired, it is possible to trade latency with decoding performance as shown in Figure 5–12. The stochastic decoder occupies a core area of 6.38 mm^2 in CMOS 90nm technology. To compare with the area of the decoders that are implemented in different technologies, the table also shows the scaled-area-per-coded-bit (in CMOS 90 nm technology). As shown, this factor for the stochastic decoder is 3115 μm^2, which indicates the most area efficiency among the soft-decision decoders in the table. The decoder also achieves a maximum core throughput of 61.3 Gb/s which is the second highest throughput among the decoders. To bring the effect of technology and code block length into account, the table shows the throughput-per-coded-bit-per-scaled-area (in CMOS 90nm technology) for each decoder. This factor for the stochastic decoder is 4.69 Mb/s/mm^2 which is about twice that of for the decoder in [108] and it is about 4% less than that of for the decoder in [67].

5.7 Conclusion

This chapter proposed the node-based rerandomization of stochastic streams using MTFMs for area-efficient and high throughput ASIC implementation of stochastic LDPC decoders. It presented a (2048,1723) fully parallel MTFM-based ASIC LDPC decoder. The implemented decoder provides a maximum throughout of 61.3 Gb/s and occupies a 6.38 mm^2 core area in CMOS 90 nm technology. To the best of our knowledge, this decoder is the most area-efficient soft-decision fully parallel LDPC decoder and it is one of the fastest fully parallel LDPC decoders reported in the literature.

Table 5–3: Comparison with some state-of-the-art high throughput soft-decision ASIC LDPC decoders.

	Decoders for the (2048,1723) LDPC code from IEEE 802.3an					Other decoders	
	this work	[24, 28]	[108, 109]	[54]	[67]	[13]	[15]
Code	(2048,1723)	(2048,1723)	(2048,1723)	(2048,1723)	(2048,1723)	(1024,512)	(256,128)
Code structure	regular, RS-based	regular RS-based	regular RS-based	regular RS-based	regular RS-based	irregular	regular
Maximum (d_v,d_c)	(6,32)	(6,32)	(6,32)	(6,32)	(6,32)	not reported	(3,6)
Decoding algorithm	MTFM-based stochastic	bit-serial approx. MSA	offset MSA + postprocessing	SPA	MSA Split-16	SPA	bit-serial MSA
Implementation strategy	fully parallel	fully parallel	partially parallel	partially parallel	fully parallel	fully parallel	fully parallel
CMOS technology	90 nm	90 nm	65 nm	90 nm	65 nm	160 nm	180 nm
Input quantization	6 bits	4 bits	4 bits	5 bits	5 bits	4 bits	4 bits
Clock frequency	500 MHz	250 MHz (before place & route)	700 MHz	207 MHz	195 MHz	64 MHz	250 MHz
Iterations or decoding cycles	max. 400 (with postproc.)	8	$8+4$ postproc.	16	11	64	32
Clocks per iteration or decoding cycle	1	4	12	5	1	1	8
Decoding latency	800 ns	128 ns	206 ns	386 ns	54 ns	1000 ns	1024 ns
Throughput	61.3 Gb/s at E_b/N_0=5.5 dB	max. 16 Gb/s	47.7 Gb/s at E_b/N_0=5.5 dB	5.3 Gb/s	92.8 Gb/s at $E_b/N_0\approx$4.55 dB	1 Gb/s	500 Mb/s
Area	6.38 mm^2	9.8 mm^2 (before place & route)	5.35 mm^2	14.5 mm^2	4.84 mm^2	$\simeq$40 mm^2	6.96 mm^2
T/P per coded bit per scaled area (in 90 nm)	4.69 Mb/s/mm^2	0.80 Mb/s/mm^2	2.27 Mb/s/mm^2	0.18 Mb/s/mm^2	4.88 Mb/s/mm^2	0.08 Mb/s/mm^2	1.12 Mb/s/mm^2
Scaled area per coded bit (in 90 nm)	3115 μm^2	4785 μm^2 (before place & route)	5008 μm^2	7080 μm^2	4530 μm^2	12360 μm^2	6797 μm^2

CHAPTER 6

Joint Stochastic Decoding of LDPC Codes and Partial-Response Channels

This chapter proposes the application of the stochastic decoding approach for joint decoding of LDPC codes and partial-response channels. Our proposed method relies on the joint message-passing algorithm in [51, 52] for decoding of LDPC codes and partial-response channels.

6.1 System Model

Figure 6–1 depicts the system model considered in this chapter. In this model, it is assumed that an independent and identically distributed binary vector $\boldsymbol{x} = (x_1, ..., x_n)$, where $x_i \in \{0, 1\}$, is passed through a partial-response channel. The transfer polynomial of the partial-response channel is $h(D) = \sum_{j=0}^{d} h_j D^j$, where d is the channel degree, and h_j is a real number. The output vector of the partial-response channel is $\boldsymbol{y} = (y_1, ..., y_{n+d})$, and its output alphabet set is A, where $y_i \in A$. The vector $\boldsymbol{y}$ is passed through an AWGN channel with zero mean and a single-sided noise power spectral density of N_0. The joint decoder receives the vector $\boldsymbol{r} = (r_1, ..., r_{n+d})$ from the AWGN channel.

Similar to [51], we consider two types of partial-response channels in this chapter: the dicode channel and the EPR4 channel. The transfer polynomial of the dicode channel is

$$h(D) = 1 - D. \tag{6.1}$$

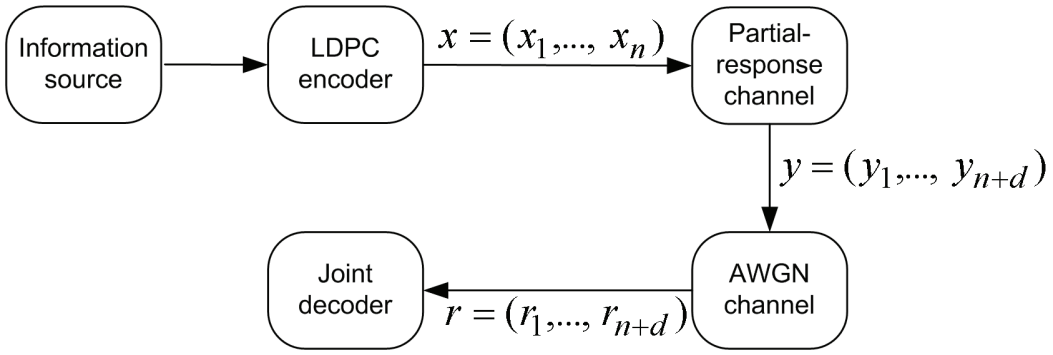


Figure 6–1: System model.

In the dicode channel, $d = 1$ and the channel's output alphabet set is $A = \{-1, 0, +1\}$. The EPR4 channel model is a practical partial-response channel model considered in magnetic recording applications. This channel model has a transfer polynomial of

$$h(D) = 1 + D - D^2 - D^3. \tag{6.2}$$

In the EPR4 channel, $d = 3$ and $A = \{-2, -1, 0, +1, +2\}$. We assume that the partial-response model starts from the zero state, and it is terminated at the zero state.

6.2 Overview of Joint Message-Passing Decoding

This section provides a brief overview of the bit-based joint message-passing decoding of LDPC codes and partial-response channels which was proposed in [51, 52].

Figure 6–2 depicts the block diagram of the joint message-passing decoding of LDPC codes and partial-response channels. The joint message-passing decoder is comprised of a partial-response channel detector and an LDPC decoder. Similar to LDPC codes, the message-passing detector for the partial-response channel detector is represented by a bipartite graph. This graph has two types of nodes: triangle nodes which receive the noisy samples, r_i, from the AWGN channel, and bit nodes. In Figure 6–2, triangle nodes are depicted

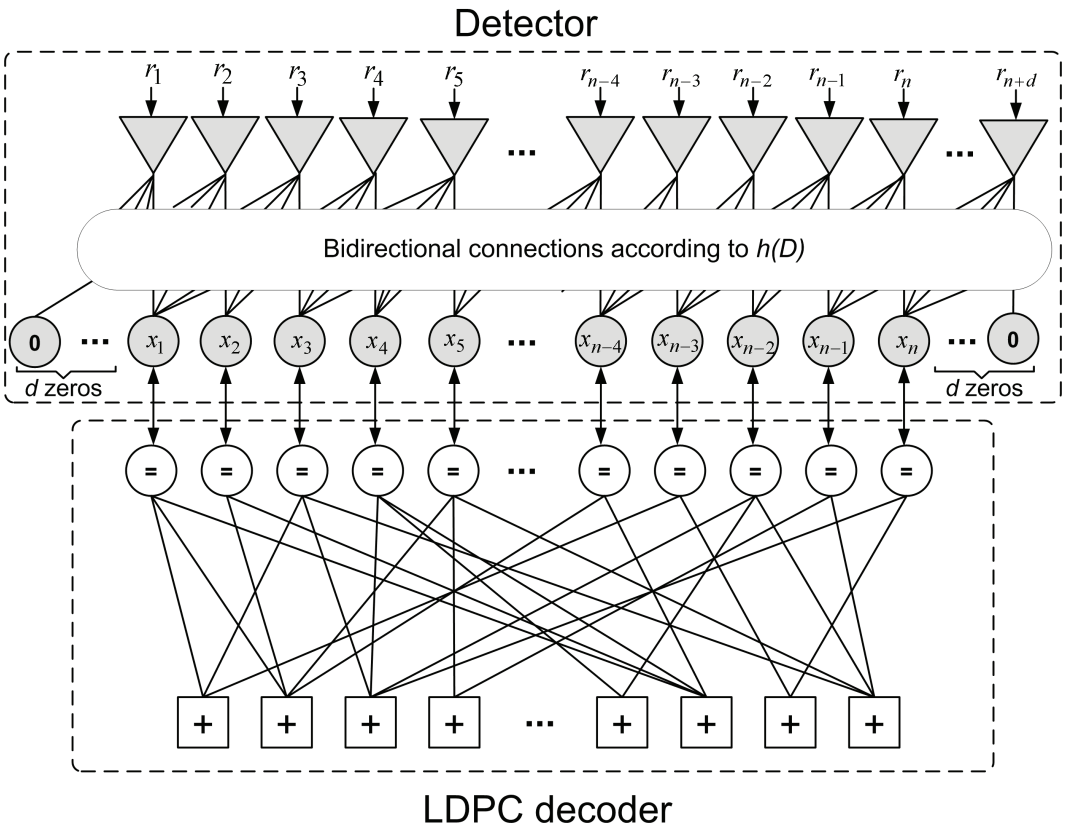


Figure 6–2: Joint message-passing diagram for decoding LDPC codes and partial-response channels. d is the degree of the partial-response channel.

as grey triangles and bit nodes are depicted as grey circles. The connection between triangle nodes and bit nodes is determined by $h(D)$, i.e., a triangle node is connected to a bit node if and only if $h(D)$ indicates a direct dependence between the input and the corresponding output of the partial-response channel. The detector operates for T iterations on the received samples from the channel, then it passes its soft outputs to the LDPC decoder. The LDPC decoder runs for S iterations and its soft outputs are passed back to the detector. This scheme is repeated for U global/turbo iterations [51, 52].

Figures 6–3 (a) and (b) show the message-passing graph for the dicode channel and the EPR4 channel, respectively. Note that the message-passing graph for the dicode channel is acyclic, but the graph for the EPR4 channel has many length-4 cycles. A length-4 cycle is highlighted in the latter graph.

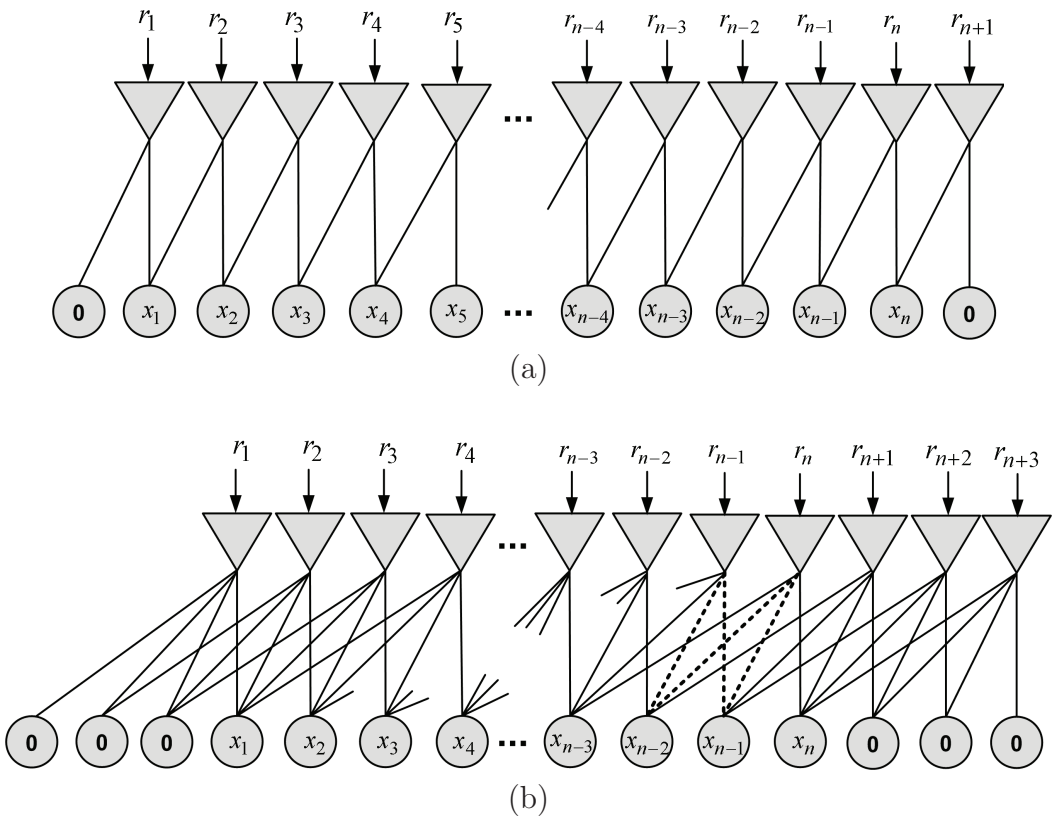


Figure 6–3: (a) The message-passing diagram for the dicode channel with $h(D) = 1 - D$. The p-th triangle node is connected to bit nodes numbered p and $p-1$. (b) The message-passing diagram for the EPR4 channel with $h(D) = 1 + D - D^2 - D^3$. The p-th triangle node is connected to bit nodes numbered p, $p-1$, $p-2$, and $p-3$. A length-4 cycle is highlighted in the graph.

6.2.1 Operation of Triangle Nodes

Triangle nodes and bit nodes in the detector exchange probability messages that represent $\Pr(x_i = 1)$. Their outgoing messages are produced according to the SPA rule, in which the outgoing message for an edge is based on all incoming messages, excluding the message received from that edge. Let $\boldsymbol{x}_{p-d}^{p} = (x_{p-d}, ..., x_p)$ and let $\boldsymbol{x}_{p-d}^{p\backslash m}$ be a vector for the same bits excluding bit x_m for $p - d \leq m \leq p$. Let $\boldsymbol{b}_0^{d\backslash m-p+d} = (b_0, ..., b_d)$ be a vector of binary inputs to the partial-response channel except for b_{m-p+d}, and let $B_{m,p,d}$ be the set of all such binary inputs. The probability message from the p-th triangle node to the m-th bit node is $R_{pm}(1)$, and the message from the m-th bit node to the

p-th triangle node is $Q_{mp}(1)$. $R_{pm}(\cdot)$ and $Q_{mp}(\cdot)$ are defined as functions of j, where $j \in \{0, 1\}$. The p-th triangle node computes $R_{pm}(j)$ as follows [51, 52]:

$$
\begin{aligned}
R_{pm}(j) = & \\
& \Pr(x_m = j | r_p) = \\
& \sum_{a \in A, \boldsymbol{b}_0^{d \backslash m-p+d} \in B_{m,p,d}} \Pr(x_m = j, \boldsymbol{x}_{p-d}^{p \backslash m} = \boldsymbol{b}_0^{d \backslash m-p+d}, y_p = a | r_p) = \\
& \sum_{a \in A, \boldsymbol{b}_0^{d \backslash m-p+d} \in B_{m,p,d}} \Pr(x_m = j, \boldsymbol{x}_{p-d}^{p \backslash m} = \boldsymbol{b}_0^{d \backslash m-p+d}, y_p = a) \times \\
& \frac{\Pr(r_p | y_p = a) \Pr(y_p = a | \boldsymbol{x}_{p-d}^{p \backslash n} = \boldsymbol{b}_0^{d \backslash m-p+d}) \Pr(\boldsymbol{x}_{p-d}^{p \backslash m)} = \boldsymbol{b}_0^{d \backslash m-p+d})}{\Pr(r_p)} = \\
& \sum_{a \in A, \boldsymbol{b}_0^{d \backslash m-p+d} \in B_{m,p,d}} \Pr(x_m = j, \boldsymbol{x}_{p-d}^{p \backslash n} = \boldsymbol{b}_0^{d \backslash m-p+d}, y_p = a) \times \\
& \frac{\Pr(r_p | y_p = a) \Pr(y_p = a | \boldsymbol{x}_{p-d}^{p \backslash m} = \boldsymbol{b}_0^{d \backslash m-p+d}) \prod_{u=p-d}^{p \backslash m} Q_{up}(b_{u-p+d})}{\Pr(r_p)}.
\end{aligned}
\tag{6.3}
$$

In the above equation, the term $\Pr(x_m = j, \boldsymbol{x}_{p-d}^{p \backslash n} = \boldsymbol{b}_0^{d \backslash m-p+d}, y_p = a)$ is either zero or one, and the term $\Pr(y_p = a | \boldsymbol{x}_{p-d}^{p \backslash m})$ is equal to 0.5 for all nonzero terms in the summation. Also, the term $\Pr(r_p | y_p = a)$ is the channel probability, which is calculated using the knowledge that the channel is AWGN. Finally, $\Pr(x_m = j, \boldsymbol{x}_{p-d}^{p \backslash n} = \boldsymbol{b}_0^{d \backslash m-p+d}, y_p = a)$ are the prior probabilities which are factored into individual probability messages sent by the connected bit nodes to the p-th triangle node [51, 52].

6.2.2 Operation of Bit Nodes

The operation of bit nodes is the same as the operation of VNs in the SPA. The m-th bit node computes its outgoing message for the p-th triangle node, $Q_{mp}(j)$, as follows [51, 52]:

$$Q_{mp}(j) = \frac{\prod_{u=m}^{m+d\backslash p} \Pr(x_m = j|r_u)}{\prod_{u=m}^{m+d\backslash p} \Pr(x_m = 1|r_u) + \prod_{u=m}^{m+d\backslash p} \Pr(x_m = 0|r_u)} = \frac{\prod_{u=m}^{m+d\backslash p} R_{um}(j)}{\prod_{u=m}^{m+d\backslash p} R_{um}(1) + \prod_{u=m}^{m+d\backslash p} R_{um}(0)}. \tag{6.4}$$

6.3 The Proposed Method

The triangle node operation in the joint message-passing decoding is a computationally-intensive operation which requires the division, multiplication, and summation of probabilities. It is possible to perform this operation in the log-domain to avoid division and multiplication, but even by using the LLR transformation, the triangle node's operation requires the evaluation of complex nonlinear functions (see [21]). In this section, we propose hardware architectures that perform the triangle node operation for the dicode and the EPR4 channels using the stochastic approach. Because the operation of bit nodes in a message-passing channel detector is the same as VNs in a LDPC decoder, stochastic VNs discussed in previous chapters can be used to perform bit node operations in a stochastic partial-response channel detector. In this regard, we do not discuss the hardware architectures of stochastic bit nodes in this chapter.

In joint stochastic decoding, channel probabilities, $\Pr(r_p|y_p \in A)$, are transformed into stochastic bit streams. Similar to stochastic LDPC decoding, this transformation is done by comparing each channel probability to a (pseudo) random number that changes in every decoding cycle. The output bit stream of the comparator represents the corresponding channel probability.

In the dicode channel, the alphabet set A has three elements; therefore, each triangle node transforms three channel probabilities to stochastic streams, i.e., $\Pr(r_p|y_p = -1)$, $\Pr(r_p|y_p = 0)$, and $\Pr(r_p|y_p = +1)$. Similarly, in the EPR4 channel, A has five elements and each triangle node transforms five channel probabilities to stochastic streams, i.e., $\Pr(r_p|y_p = -2)$, $\Pr(r_p|y_p = -1)$, $\Pr(r_p|y_p = 0)$, $\Pr(r_p|y_p = +1)$, and $\Pr(r_p|y_p = +2)$.

Each triangle node in the stochastic detector receives one bit from each of its (channel) comparators in every decoding cycle.[1] The stochastic channel detector operates by stochastic triangle nodes and bit nodes exchanging bits for T_{SD} decoding cycles. The detector then passes its soft output (extracted by TFMs) to the stochastic LDPC decoder which runs for a maximum of S_{SD} decoding cycles. The stochastic LDPC decoder performs syndrome checking in every decoding cycle to terminate the joint decoding process as soon as all the parity-checks are satisfied. If this termination criterion is not satisfied within S_{SD} decoding cycles, the stochastic LDPC decoder passes back its soft outputs (extracted by TFMs) to the channel detector. This scheme is repeated for at most U global/turbo iterations.

6.3.1 Stochastic Triangle Nodes for the Dicode Channel Detector

As mentioned previously, the message-passing graph of the dicode channel is acyclic. In acyclic graphs, the latching problem does not hold. In this respect, the proposed stochastic triangle node architecture for the dicode channel does not use rerandomization units.

[1] Similar to the definition of a decoding cycle or DC (i.e., a stochastic decoding iteration) in Section 2.4 for stochastic LDPC decoding, in a stochastic partial-response channel detector, a decoding cycle refers to the exchange of one bit between triangle nodes and bit nodes.

From (6.3), it follows that we can write $R_{pm}(1)$ in the form of

$$R_{pm}(1) = \frac{P_1}{P_1 + P_0}, \tag{6.5}$$

where, for the case of the dicode channel,

$$\begin{aligned} P_1 &= \Pr(x_p = 1, r_p) = \\ &\quad 0.5 \times \Pr(r_p|y_p = +1)(1 - Q_{(m-1)p}) + 0.5 \times \Pr(r_p|y_p = 0)Q_{(m-1)p}, \end{aligned} \tag{6.6}$$

and

$$\begin{aligned} P_0 &= \Pr(x_p = 0, r_p) = \\ &\quad 0.5 \times \Pr(r_p|y_p = 0)(1 - Q_{(m-1)p}) + 0.5 \times \Pr(r_p|y_p = -1)Q_{(m-1)p}. \end{aligned} \tag{6.7}$$

Figure 6–4 depicts the proposed hardware architecture to compute $R_{pm}(1)$ in a stochastic triangle node for the dicode channel detector. A similar hardware architecture is used to compute $R_{p(m-1)}(1)$. In this architecture, the inverse operation on $Q_{(m-1)p}$ is performed using a NOT gate, and the multiplication of probabilities is performed using AND gates. The output of each AND gate in the figure forms a term for the summation in (6.6) and (6.7). The stochastic summation is performed by two 2-input OR gates. As mentioned in Chapter 2, an OR gate can be used as an approximate stochastic adder. The output streams of the OR gates shown in the figure represent $P_1' \approx 2P_1$ and $P_0' \approx 2P_0$. Finally, the stochastic streams representing P_1' and P_0' are passed to a JK flip-flop that performs a division and its output bit stream represents $P_1'/(P_1' + P_0')$, which approximates $R_{pm}(1) = P_1/(P_0 + P_1)$.

The structure of degree-2 stochastic bit nodes used in the dicode channel detector is based on the basic stochastic VN structure, where a JK flip-flop is used to perform division (see Figure 2–9 in Chapter 2).

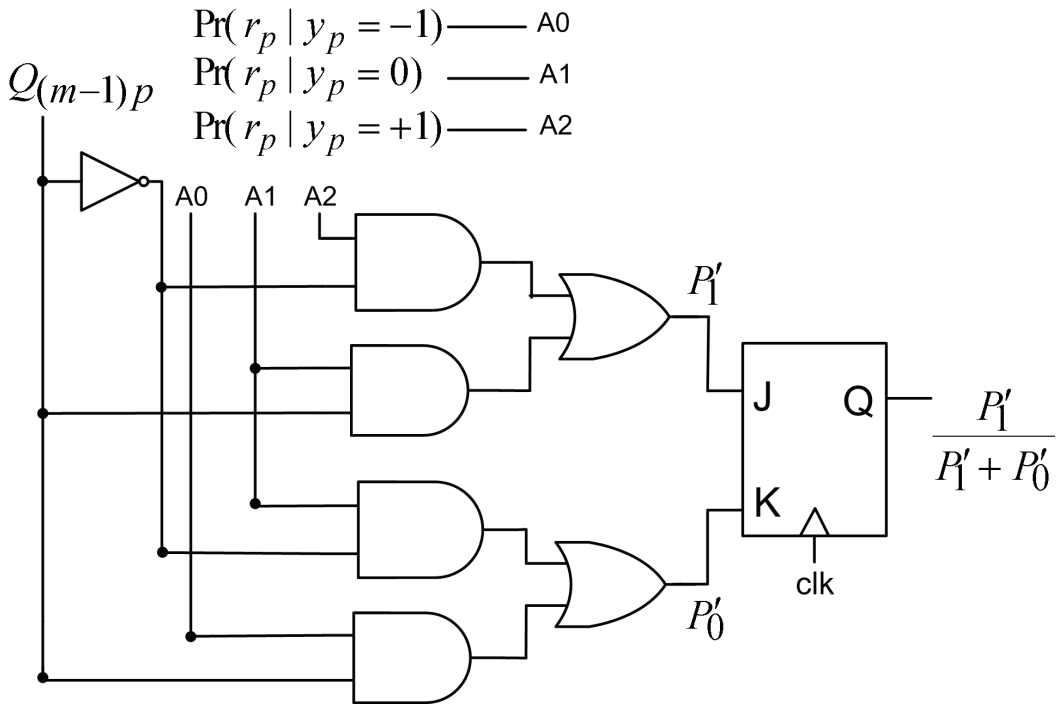


Figure 6–4: The hardware architecture of a stochastic triangle node for the dicode channel (only one output and its corresponding inputs are shown).

6.3.2 Stochastic Triangle Nodes for the EPR4 Channel Detector

The message-passing graph for the EPR4 channel has many length-4 cycles (see Figure 6–3(b)). These short cycles severely intensify the latching problem in the stochastic channel detector and deteriorate the BER decoding performance of joint stochastic decoding. Moreover, triangle nodes and bit nodes in the EPR4 channel detector have higher node degrees and perform more complex operations compared to the triangle nodes and bit nodes in the dicode channel detector. In this respect, the proposed architecture for stochastic triangle nodes in the EPR4 channel detector relies on TFMs, as rerandomization units, to alleviate the latching problem and to increase the switching activity in the joint decoder.

In the EPR4 channel, the term P_1 in (6.5) is computed as:

$$
\begin{aligned}
P_1 = \Pr(x_p = 1, r_p) = \\
& 0.5 \times \Pr(r_p|y_p = +1)(1 - Q_{(m-1)p})(1 - Q_{(m-2)p})(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = 0)(1 - Q_{(m-1)p})(1 - Q_{(m-2)p})Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = 0)(1 - Q_{(m-1)p})Q_{(m-2)p}(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = -1)(1 - Q_{(m-1)p})Q_{(m-2)p}Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = +2)Q_{(m-1)p}(1 - Q_{(m-2)p})(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = +1)Q_{(m-1)p}(1 - Q_{(m-2)p})Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = +1)Q_{(m-1)p}Q_{(m-2)p}(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = 0)Q_{(m-1)p}Q_{(m-2)p}Q_{(m-3)p}.
\end{aligned}
\tag{6.8}
$$

Also, P_0 in (6.5) is computed as follows:

$$
\begin{aligned}
P_0 = \Pr(x_p = 0, r_p) = \\
& 0.5 \times \Pr(r_p|y_p = 0)(1 - Q_{(m-1)p})(1 - Q_{(m-2)p})(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = -1)(1 - Q_{(m-1)p})(1 - Q_{(m-2)p})Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = -1)(1 - Q_{(m-1)p})Q_{(m-2)p}(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = -2)(1 - Q_{(m-1)p})Q_{(m-2)p}Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = +1)Q_{(m-1)p}(1 - Q_{(m-2)p})(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = 0)Q_{(m-1)p}(1 - Q_{(m-2)p})Q_{(m-3)p} + \\
& 0.5 \times \Pr(r_p|y_p = 0)Q_{(m-1)p}Q_{(m-2)p}(1 - Q_{(m-3)p}) + \\
& 0.5 \times \Pr(r_p|y_p = -1)Q_{(m-1)p}Q_{(m-2)p}Q_{(m-3)p}.
\end{aligned}
\tag{6.9}
$$

Figure 6–5 depicts the proposed hardware architecture to compute $R_{pm}(1)$ in a stochastic triangle node for the EPR4 channel. Similar hardware architectures are used to compute $R_{p(m-1)}(1)$, $R_{p(m-2)}(1)$, and $R_{p(m-3)}(1)$. In the architecture shown, the network of AND gates computes the terms that are needed for

summations in (6.8) and (6.9). The stochastic summation is performed by two 8-input OR gates. The output streams of the OR gates shown in the figure represent $P_1' \approx 2P_1$ and $P_0' \approx 2P_0$. The division required to compute $R_{pm}(1)$ in (6.5) is performed by a TFM-based stochastic divider. The operation and the update rule of the TFM-based divider is the same as a JK flip-flop divider, however, instead of a flip-flop, a TFM is used to efficiently rerandomize the output stochastic bit stream. The output stream of the TFM-based divider represents $P_1'/(P_1' + P_0') \approx 2P_1/(2P_1 + 2P_0) = P_1/(P_1 + P_0)$. In this divider, the TFM is updated when $J \neq K$. The output bit of the divider is 1 when $J = 1$ and $K = 0$, and it is 0 when $J = 0$ and $K = 1$. Also, when $J = K = 0$ the output bit of the TFM is directly used as the output of the divider (i.e., hold state), and when $J = K = 1$ the inverse of the output bit of the TFM is used as the output of the divider (i.e., reverse state).

The structure of degree-4 stochastic bit nodes used in the EPR4 channel detector is based on the MTFM-based stochastic VN structure (see Section 5.1). We used reduced-complexity MTFMs with $T_u = d_v = 4$ and $T_m = d_v/2 = 2$.

6.4 Decoding Performance Results

Figure 6–6 shows the BER decoding performance of the stochastic approach for joint decoding of a (2000,1000) LDPC code and the dicode partial-response channel.[2] For the sake of comparison, the figure also shows the decoding performance results obtained for the floating-point joint message-passing decoding (with a floating-point channel detector and a floating-point

[2] The energy per bit in the simulation results reported in this chapter is defined as $E_b = E_y/R$, where $R = k/n$ is the rate of the LDPC code, and E_y is the average energy of y_i's. E_y is calculated by considering the probabilities of occurrence of $y_i \in A$, given equiprobable channel inputs $x_i \in \{0, 1\}$.

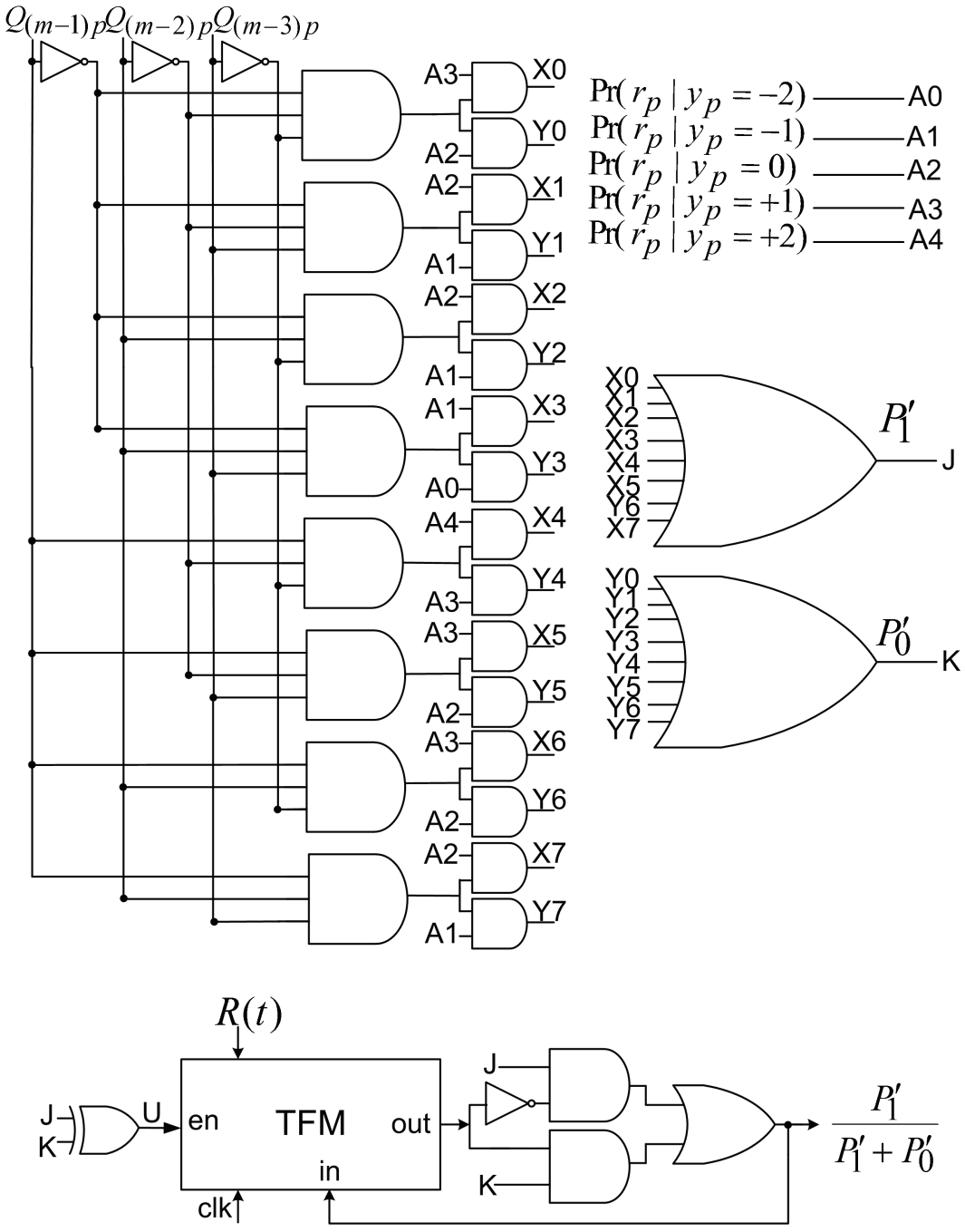


Figure 6–5: The hardware architecture of stochastic triangle node for the EPR4 channel (only one output and its corresponding inputs are shown). $R(t)$ is a (pseudo) random number varying in every decoding cycle.

SPA-based LDPC decoder), and the dicode channel truncated union bound from [51, 93]. We used $U = 16$, $T = 3$, and $S = 3$ for the floating-point SPA-based message-passing decoding. Also, for joint stochastic decoding, we used

$U = 16$, $T_{\text{SD}} = 100$ decoding cycles for detection, and $S_{\text{SD}} = 100$ decoding cycles for stochastic LDPC decoding. The (2000,1000) stochastic LDPC decoder used in joint decoding relies on TFMs with a relaxation coefficient of $\beta = 2^{-4}$. The LDPC decoder applies syndrome checking to terminate the joint decoding process as soon as all the parity-checks are satisfied. As shown, at a BER of about 10^{-8}, the proposed joint stochastic decoder is able to provide a decoding performance within 0.4 dB of the floating-point joint message-passing.

Figure 6–7 shows the BER decoding performance of the stochastic approach for joint decoding of a (2000,1000) LDPC code and the EPR4 partial-response channel. Also shown in the figure are the decoding performance of floating-point joint message-passing decoding using a floating-point channel detector and a floating-point SPA-based LDPC decoder, and the EPR4 channel truncated union bound from [51, 93]. To show the effects of quantization in the EPR4 channel detector, the figure also depicts performance results for joint message-passing decoding using an 8-bit channel detector and a floating-point SPA-based LDPC decoder. We used $U = 16$, $T = 8$, and $S = 8$ in both joint message-passing decoding schemes. For joint stochastic decoding, we used $U = 16$, $T_{\text{SD}} = 200$ decoding cycles for detection, and $S_{\text{SD}} = 200$ decoding cycles for TFM-based stochastic LDPC decoding. Results demonstrate the applicability of the proposed stochastic approach for joint decoding of LDPC codes and the EPR4 channel. Despite the existence of a high number of length-4 cycles in the detector graph of the EPR4 channel, which severely intensify the latching problem, no error-floor is observed down to a BER of about 10^{-8}. Compared to the dicode channel, more decoding loss with respect to the floating-point joint message-passing is observed. A comparison of joint message-passing decoding with floating-point and 8-bit channel detectors reveals the sensitivity of the BER decoding performance to the number

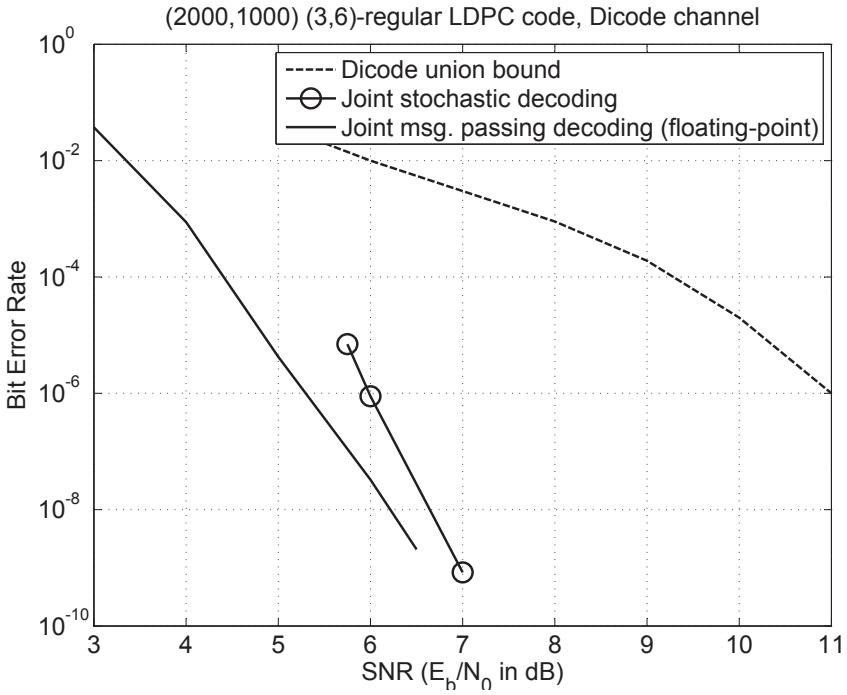


Figure 6–6: Decoding performance of the stochastic approach for joint decoding of a (2000,1000) LDPC code and the dicode partial-response channel.

of quantization levels used in the EPR4 channel detector. The decoding performance of the joint stochastic decoding is within about 0.6 dB of the joint message-passing decoding with an 8-bit channel detector and a floating-point SPA-based LDPC decoder.

6.5 Estimation of Decoding Latency and Throughput

Although the hardware implementations of the proposed joint stochastic decoders are not considered in this chapter, based on the results presented in previous chapters, it is possible to investigate/approximate the (core) decoding latency and throughput of the joint stochastic decoders. As mentioned in previous chapters, one of the main challenges in hardware implementations of LDPC decoders is the random-like/irregular connections between VNs and PNs. These irregular connections result in long physical wires across the decoder chip, which limit the speed and increase the area and power consumptions of LDPC decoders. It is worth noting that, compared to LDPC codes,

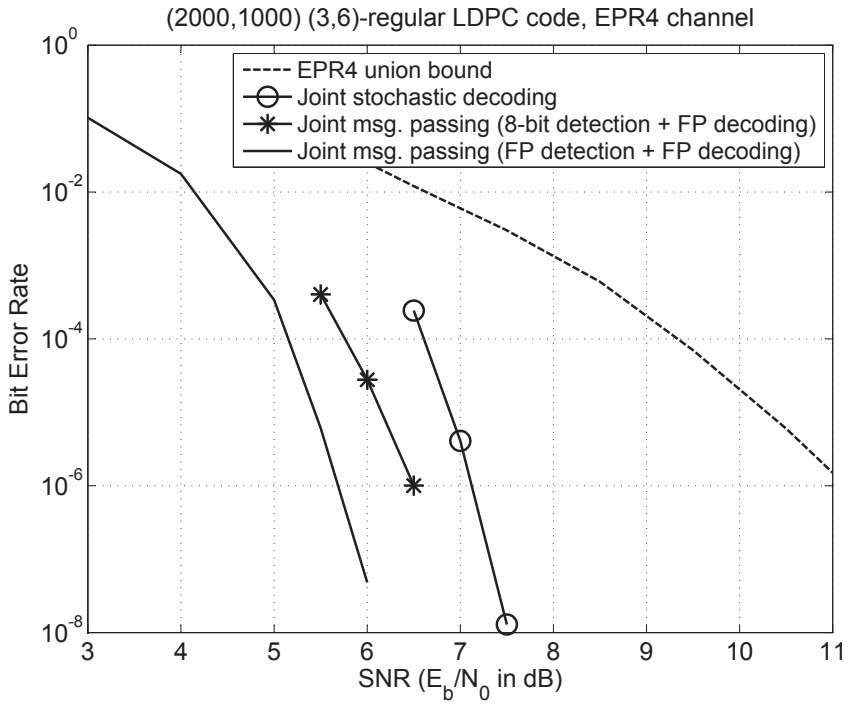


Figure 6–7: Decoding performance of the stochastic approach for joint decoding of a (2000,1000) LDPC code and the EPR4 partial-response channel (FP: floating-point).

connections between triangle nodes and bit nodes in a message-passing graph for partial response channels are regular. This feature is favorable for hardware implementations, because physical wires between triangle nodes and bit nodes become local/short, which potentially results in a higher speed as well as higher area and power efficiency.

ASIC implementations of stochastic LDPC decoders in previous chapters showed that clock frequencies in the order of 500 MHz can be achieved for fully parallel ASIC stochastic LDPC decoders (in CMOS 90nm technology). The decoding latency of the joint stochastic decoder is determined by $U(T_{\mathrm{SD}}+S_{\mathrm{SD}})$. For the dicode channel and the EPR4 channel, the decoding latencies are 3200 decoding cycles and 6400 decoding cycles, respectively, where each decoding cycle takes one clock cycle. Figure 6–8 shows the decoding latency of the joint decoder for different clock frequencies ranging from 100 MHz to 500 MHz. It

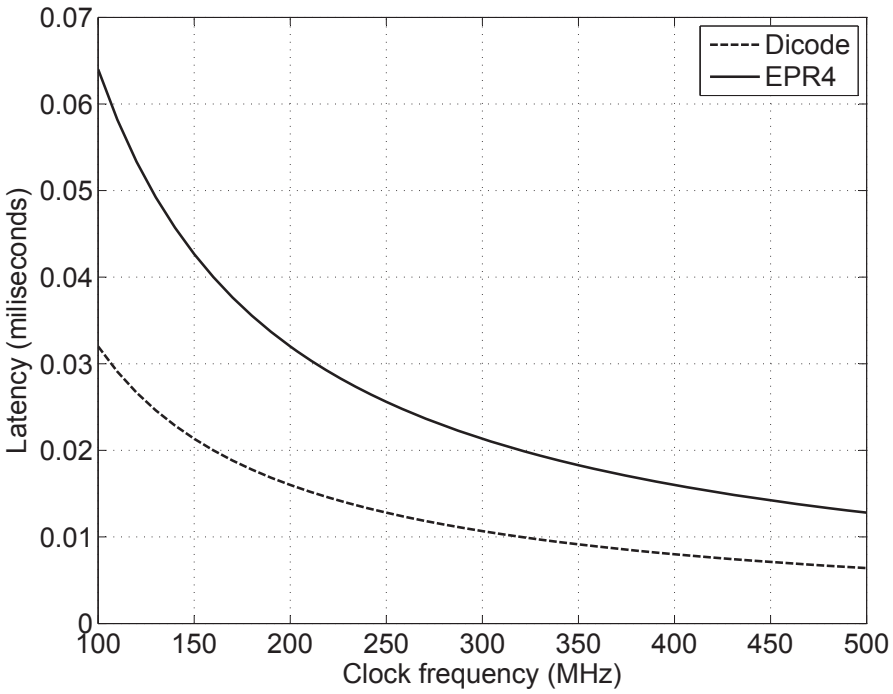


Figure 6–8: Estimated latency of joint stochastic decoding for different clock frequencies.

should be noted that compared to digital wireless and wireline communications applications, magnetic recording applications have higher decoding latency requirements (usually, in the order of milliseconds). As shown, the decoding latency for the case of the dicode channel ranges from 0.0320 down to 0.0064 milliseconds, and for the case of the EPR4 channel, it ranges from 0.0640 down to 0.0128 milliseconds.

The throughput of the joint decoder is determined by the average number of decoding cycles used. This is because the stochastic LDPC decoder uses an early decoding termination criterion that stops the joint decoding process as soon as all the parity-checks are satisfied. Figure 6–9 (a) depicts the average number of decoding cycles used for joint stochastic decoding at different SNRs for the dicode and the EPR4 channels. Also, Figures 6–9 (b) and (c) show the corresponding (core) throughput for different clock frequencies ranging from 100 MHz to 500 MHz. As shown, the average number of decoding cycles

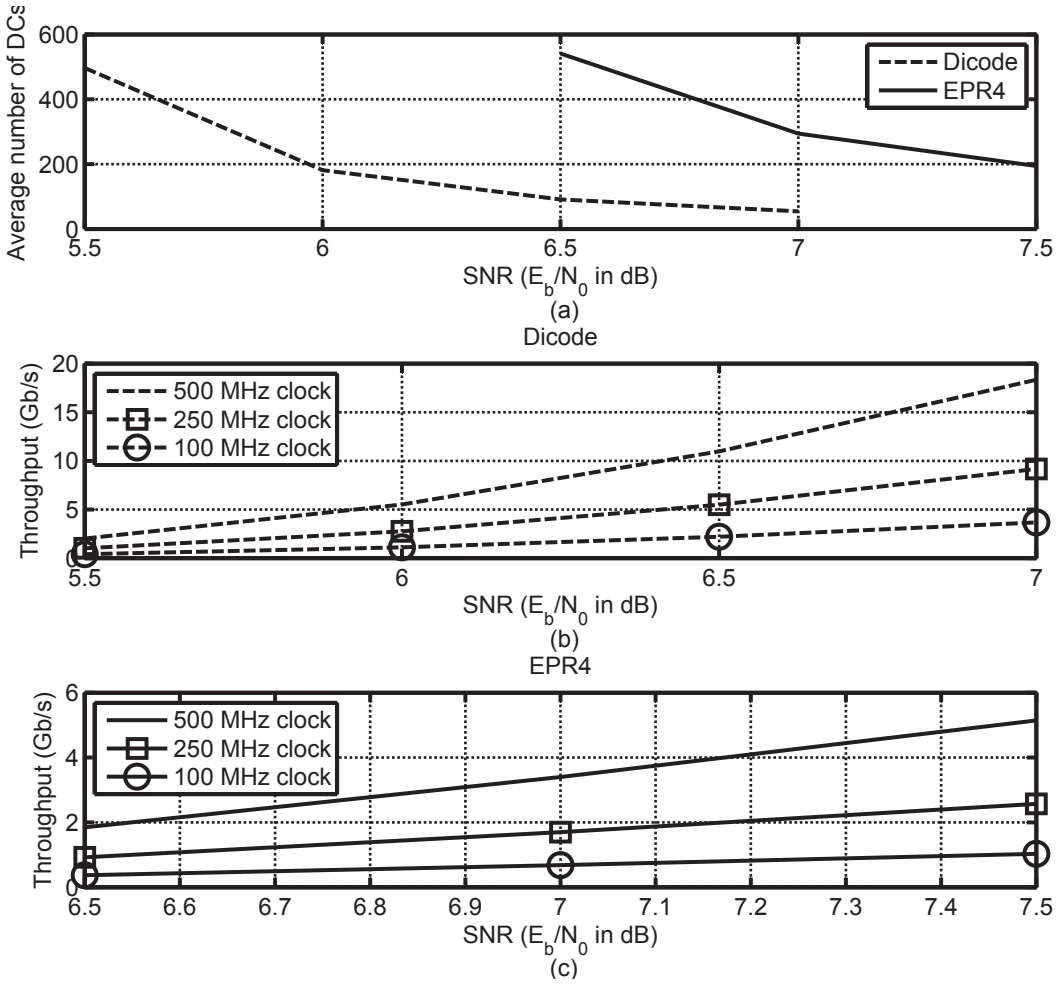


Figure 6–9: (a) Average number of decoding cycles used for joint stochastic decoding at different SNRs. (b) Estimated (core) throughput for joint stochastic decoding of the (2000,1000) LDPC code and the dicode channel. (c) Estimated (core) throughput for joint stochastic decoding of the (2000,1000) LDPC code and the EPR4 channel.

decreases significantly in low BER regimes (high E_b/N_0), which enables the joint stochastic decoder to provide multi-Gb/s throughput.

6.6 Stochastic Channel Detection and Log-Based LDPC Decoding

Although the focus of this chapter was on joint stochastic decoding (using stochastic channel detection and stochastic LDPC decoding), it should be noted that it is feasible to use stochastic channel detection with LDPC decoding algorithms that operate in the log-domain. Recently, it was shown that the concept of TFMs is applicable in the log-domain [53]. An LLR-based TFM

receives an input stochastic bit stream and tracks the corresponding LLR of the bit stream, at the cost of more hardware-complexity compared to regular TFMs. It is therefore possible to have a stochastic channel detector whose final soft outputs (passed to an LDPC decoder) are in the form of LLRs. In this respect, stochastic partial-response channel detection can be used jointly with LDPC decoding algorithms such as the MSA, the offset MSA, etc.

6.7 Conclusion

This chapter proposed the novel application of joint stochastic decoding of LDPC codes and partial-response channels that are considered in practical magnetic recording applications. It proposed low hardware-complexity architectures for stochastic triangle nodes to perform computationally-intensive operations required in the dicode and the EPR4 partial-response channel detectors. The decoding performance, latency, and throughput of the proposed joint stochastic decoding method are discussed. Results demonstrated the applicability of the stochastic approach for joint decoding of LDPC codes and practical partial-response channels.

CHAPTER 7

Stochastic Decoding of Linear Block Codes with High-Density Parity-Check Matrices

This chapter investigates stochastic decoding of linear block codes with high-density parity-check matrices on factor graphs. As mentioned in Chapter 1, stochastic decoding was first applied to RS codes in [112]. Our contributions in this chapter will be to further investigate stochastic RS decoding, to extend the application of stochastic decoding to BCH codes and BCH-based block turbo codes, and to investigate efficient hardware implementations of high-degree nodes used in the decoding of linear block codes with high-density parity-check matrices on factor graphs. This chapter is in part based on the material in our paper [83]. It shows how the stochastic approach, despite its bit-serial nature, can integrate in the Adaptive Belief Propagation (ABP) [47] and the Turbo-oriented Adaptive Belief propagation (TAB) [45, 46]. Although the focus of this chapter is on the EM-based stochastic decoding, it should be noted that TFMs and MTFMs are also applicable for this purpose. To the best of our knowledge, results provided in this chapter are the first results reported in the literature for stochastic decoding of BCH codes and BCH-based turbo block codes on factor graphs.

7.1 Overview

Despite the excellent performance of the SPA (also referred to as belief propagation or BP) for LDPC decoding, the SPA/BP algorithm is not suitable for decoding codes with non-sparse parity-check matrices such as BCH and RS codes. This problem was investigated in [47] and a new ABP was suggested for RS decoding. It was shown that when BP is applied to a code with high-density parity-check matrix, it is likely that BP gets stuck at some local minimum points that correspond to some unreliable symbols. Therefore, at each iteration of ABP, the parity-check matrix of the code is adapted according to the bit reliabilities to sparsify those columns associated with unreliable bits. The ABP offers a decoding gain of more than 3 dB over hard-decision RS decoding. However, the parity-check matrix adaptation step in the ABP is complex. Inspired by the ABP, a novel method for the TAB was proposed for turbo decoding of product codes [45, 46]. In the TAB, the parity-check matrix adaptation is performed before the BP process and thus, the parity-check matrix is fixed during the BP process in the component decoder. This feature significantly decreases decoding complexity. In addition, the TAB outperforms the ABP and provides a performance close to the Chase-Pyndiah algorithm [73].

7.1.1 Adaptive Belief Propagation

This section provides an overview of the ABP for SISO RS decoding over GF(2^q) [47]. The same approach is applicable for binary BCH codes.

BCH and RS codes are important classes of linear cyclic error-correcting codes with multiple error detection and correction capability. The parity-check

matrix of an (n, k) RS code over GF(2^q) can be represented by

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \alpha^{(n-k)} & \cdots & \alpha^{(n-k)(n-1)} \end{bmatrix}, \tag{7.1}$$

where α is the primitive element in GF(2^q). In the ABP, $\boldsymbol{H}$ is expanded to its binary representation, $\boldsymbol{H}_b$, by substituting each codeword in the GF(2^q) with its equivalent binary representation [47]. This representation transforms the problem of RS decoding to the general problem of decoding of an (N, K) binary block code with $N = q \times n$ and $K = q \times k$.

Let i denote the iteration step in the ABP. Also, let $\boldsymbol{L}^{(i)}$ be the vector of LLRs and $\boldsymbol{H}_b^{(i)}$ be the adapted binary parity-check matrix at the i-th iteration. At $i = 0$, the ABP starts with the channel LLRs ($\boldsymbol{L}^{(0)} = \boldsymbol{L}_{\mathrm{CH}}$) and the binary parity-check matrix ($\boldsymbol{H}_b^{(0)} = \boldsymbol{H}_b$). At each iteration of ABP, two steps are performed: the reliability-based adaptation of $\boldsymbol{H}_b^{(i)}$ and the generation of extrinsic information using BP [47]:

- Reliability-based Adaptation: In this step, the LLRs are sorted based on their absolute value in an ascending manner and ordering indices are stored. Let $\boldsymbol{L}^{(i)} = \{L_{m_1}^{(i)}, ..., L_{m_N}^{(i)}\}$ be the sorted list of LLRs and $\{m_1, ...m_N\}$ be the stored indices. The first LLR, $L_{m_1}^{(i)}$, corresponds the least reliable bit (the m_1-*th* bit in the block) and the last LLR, $L_{m_N}^{(i)}$ corresponds to the most reliable bit (the m_N-*th* bit in the block). After this phase, starting from $j = m_1$ to m_N, row operations are performed to systematize the j-*th* column of $\boldsymbol{H}_b^{(i)}$ to form a unity weight column $e_j = [0..010..0]^T$, in which the only non-zero element is placed at the j-*th* position. If such systematization is not possible for a column, the algorithm proceeds to the next column in $\boldsymbol{L}^{(i)}$. This procedure can be

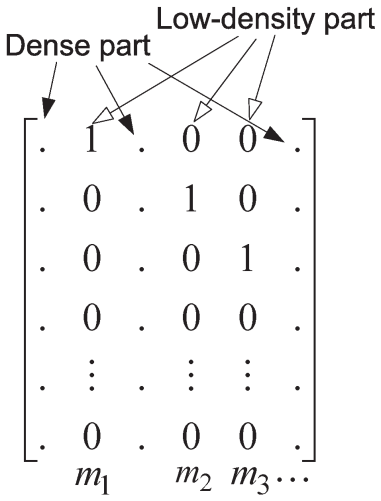


Figure 7–1: Form of an adapted parity-check matrix in the ABP [47].

done using the Gaussian elimination method. Figure 7–1 shows the form of an adapted parity-check matrix which is decomposed into dense and low-density parts.

- Generation of Extrinsic Information: After the adaptation step, the BP is applied on the sorted LLRs, $\boldsymbol{L}^{(i)}$, based on the adapted parity-check matrix, $\boldsymbol{H}_b^{(i)}$, to generate the extrinsic LLRs, $\boldsymbol{L}_{\text{ext}}^{(i)}$. The LLR $\boldsymbol{L}^{(i+1)}$ is then updated according to:

$$\boldsymbol{L}^{(i+1)} = \boldsymbol{L}^{(i)} + \lambda \times \boldsymbol{L}_{\text{ext}}^{(i)}, \tag{7.2}$$

where $0 < \lambda < 1$ is a damping coefficient. The algorithm returns to the adaptation step unless it has been run for a fixed maximum number of iterations, $i_{\max}$, or all the parity-checks are satisfied.

The ABP can be exploited with variants of BP such as the MSA and the offset MSA [44]. In addition, the Hard-Decision Decoding (HDD) step at the end of each iteration of ABP may be used to improve the decoding performance [47]. The HDD variant consists of (a) performing hard decisions on LLRs at the end of each iteration of ABP to obtain a codeword and, (b)

selecting the most likely codeword at the end of the decoding process. This variation improves the convergence and the performance of the iterative RS decoder [47].

7.1.2 Turbo-Oriented Adaptive Belief Propagation

The TAB is a new method for block turbo decoding using BP-based decoders as elementary SISO component decoders [45, 46]. The TAB is inspired by the ABP but it is less complex than the ABP and offers better decoding performance. Figure 7–2 shows the principles of SISO block turbo decoding. It includes sequential decoding of rows and columns of the component codes and the iterative process. The "global/turbo" iterations, i, have to be distinguished from the iterations of the BP process that we call "local" iterations. As mentioned earlier, the ABP requires the adaptation of $\boldsymbol{H}_b$ and the generation of extrinsic information using BP in each local iteration. The Gaussian elimination used in the adaptation step of the ABP is a computationally-expensive process. In the TAB, the adaption step is only performed at the beginning of each global iteration (before the BP process). This means that the parity-check matrix is the same during the BP process and no damping coefficient or matrix adaptation is necessary during local iterations. Instead, the LLR update is performed in the global iteration of turbo process using μ coefficient. This feature significantly reduces the complexity. In addition, the TAB outperforms the ABP for block turbo decoding and provides performance close to the Chase-Pyndiah algorithm [45, 46].

7.2 The Stochastic Decoding Method

This section discusses the stochastic method for decoding high-density linear block codes.

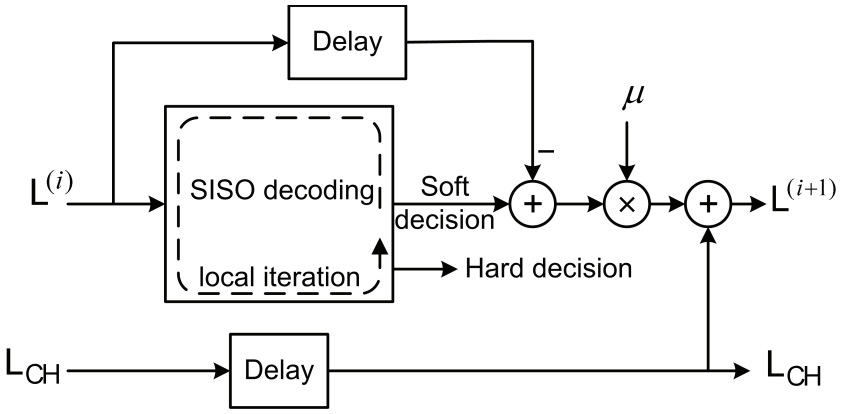


Figure 7–2: Block turbo decoding.

7.2.1 High-Degree Stochastic Nodes

The parity-check matrix of an RS or BCH code is a dense matrix which results in a factor graph with high-degree nodes. For instance, the factor graph of the (63,55) RS code used in this chapter has 378 VNs and 48 PNs over GF(2^6). The maximum degrees of VNs and PNs are 34 and 184, respectively, and about 77% of VNs have a $d_v \geq 20$. In this respect, it is essential to construct high-degree stochastic VN based on the method proposed in Section 3.2 to provide enough switching activity in the decoder and to significantly alleviate the latching problem.

7.2.2 Representing Soft-Output Information

The ABP and the TAB rely on the parity-check matrix adaptation as well as the reliability update scheme using BP. Both of these steps use the soft-output information provided by BP. Since the BP-based decoders inherently operate on reliabilities, they can be easily incorporated into the adaptation and the update scheme. However, the situation is different for stochastic decoding methods. Stochastic methods convert the channel reliabilities to stochastic bit streams and the decoding proceeds entirely in a bit-serial fashion. To incorporate the reliability update and adaptation steps, it is essential that the stochastic decoding method provides soft-output information. For this

purpose, we use the value of saturating up/down counters to represent soft-output information [112]. In this technique, a counter is assigned to each VN. The counter can be initialized to contain zero value. The counter is incremented if the corresponding VN output is 1, unless the counter has reached its maximum limit ($+U$). Similarly, when the VN output is 0 the counter is decremented, unless it has reached its minimum limit ($-U$). After a number of decoding cycles, the contents of the counters can be converted to soft information and the LLR update and the parity-check matrix adaptation steps can be performed. Let V be the value of a saturating up/down counter associated with a VN ($-U \leq V \leq +U$). This value can be transformed into extrinsic soft-information in the probability domain as [112]:

$$P_{\text{ext}} = \frac{V + U}{2U}. \tag{7.3}$$

Consequently, the corresponding extrinsic LLR of the VN is

$$L_{\text{ext}} = \log\left(\frac{P_{\text{ext}}}{1 - P_{\text{ext}}}\right) = \log\left(\frac{U + V}{U - V}\right), \tag{7.4}$$

where $\log(\cdot)$ indicates the natural logarithm operation.

To increase the efficiency of the counters, we update counters only with regenerative (output) bits. This means that the counters are not updated when their corresponding VNs are in the hold state. This technique prevents a counter from being updated with the same bits when its VN is in the hold state. Also, it provides a faster convergence for counters (i.e., output reliabilities) and makes it possible to use counters with a smaller size. Note that as (7.4) shows, the size and the value of a counter affects the reliabilities. In this respect, a large size counter requires more decoding cycles to converge.

It should also be noted that the sign-bits of the up/down counters can be used for the HDD method in RS decoding. In this case, a hard-decision is

applied to V. Thus, a 0 sign-bit and a 1 sign-bit, respectively, indicate a -1 and a $+1$ decision, as discussed in Section 3.5.

7.2.3 Summary of the Stochastic Decoding Method

The stochastic decoding method starts with $\boldsymbol{H}_b$ and $\boldsymbol{L}_{\mathrm{CH}}$. At the adaptation step, the parity-check matrix adaptation is performed to obtain $\boldsymbol{H}_b^{(i)}$ as discussed in Section 7.1.1. After the adaptation step, LLRs are scaled and transformed into stochastic bit streams as in (3.2). Stochastic decoding is then applied to the adapted parity-check matrix $\boldsymbol{H}_b^{(i)}$ and proceeds by VNs and PNs exchanging bits for a fixed number of decoding cycles. At the end of the last decoding cycle, the contents of up/down counters are transformed into the extrinsic LLRs, $\boldsymbol{L}_{\mathrm{ext}}^{(i)}$, according to (7.4). The $\boldsymbol{L}_{\mathrm{ext}}^{(i)}$ is then used to update LLRs based on the ABP (for RS and BCH codes) or the TAB (for BCH block turbo codes). For the case of RS decoding, the HDD is also applied to the contents of counters. The decoding process is terminated as soon as all parity checks are satisfied or if a maximum number of iterations, $i_{\max}$, has been exhausted.

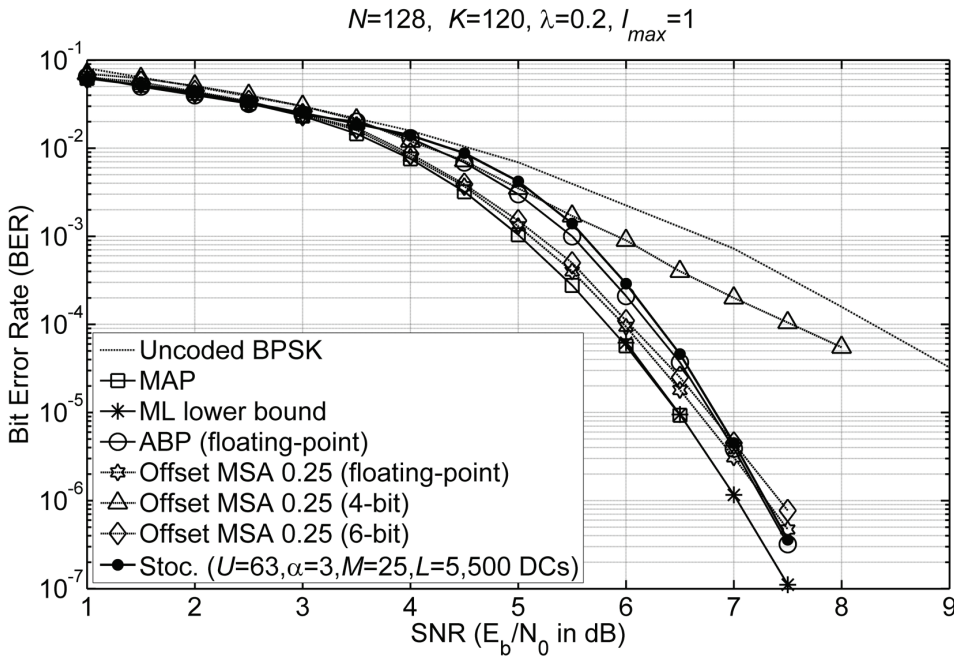
7.3 Decoding Performance Results

Classes of BCH, RS and BCH block turbo codes are considered for simulation. Concerning BCH codes, extended BCH codes are considered since they are more efficient than non-extended codes. A BPSK transmission with an average bit energy of E_b over an AWGN channel is assumed for each simulation. The parity-check matrix adaption step for BCH and RS codes is done based on the ABP. This step for BCH block turbo codes is performed based on the TAB.

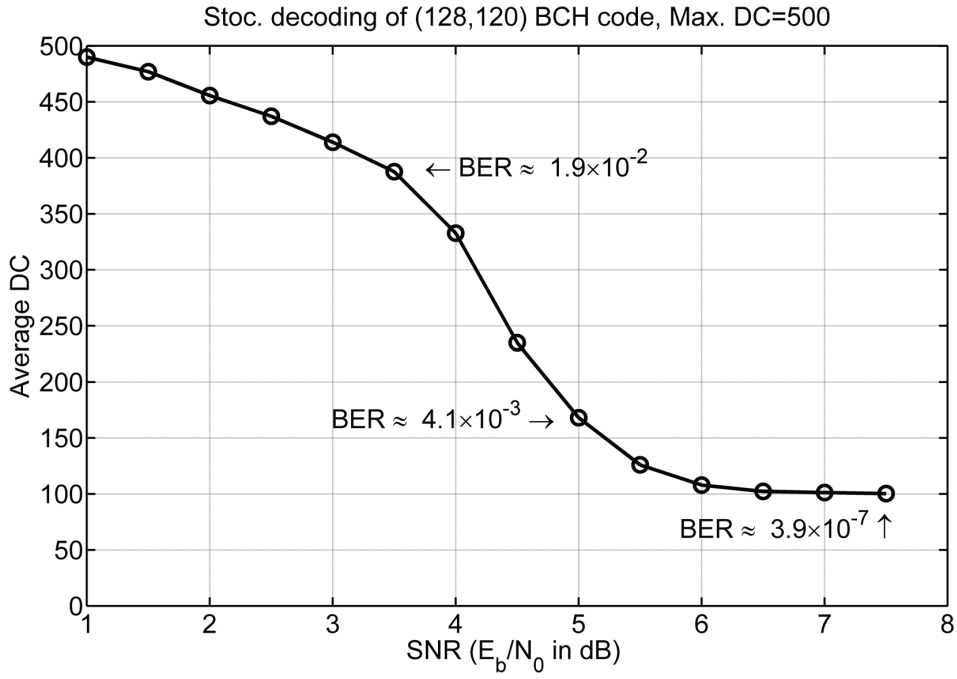
The decoding performance of a (128,120) BCH code is depicted in Figure 7–3(a). For this code, several decoding methods are employed: the Maximum A Posteriori (MAP) probability decoding, the ABP, the adaptive offset MSA and stochastic decoding. For comparison, the uncoded BPSK and the

Maximum-Likelihood (ML) lower bound [19] curves are also plotted. The values of the damping coefficient and number of adaptations are $\lambda = 0.2$ and $i_{\max} = 1$. The EM and IM lengths of $M = 25$ and $L = 5$ are used for the stochastic decoding method. Stochastic decoding runs for a fixed number of 500 decoding cycles. No significant BER deviation is observed for (128,120) BCH codes between the stochastic decoding and the floating-point ABP. The floating-point offset MSA outperforms the ABP at low SNRs. As shown, to achieve the same performance with the fixed-point adaptive offset MSA at least 6-bit precision is needed. The MAP decoding outperforms other methods by about 0.25 dB at a BER of 10^{-6} and achieves the asymptotic bound. However, the MAP decoding of this BCH code requires a trellis with 256 states and 128 sections which is too complex for implementation.

Figure 7–4 shows the results obtained for (31,25) and (63,55) RS codes over GF(2^5) and GF(2^6). In this figure, FER performance for Algebraic hard-decision decoding, the ABP, the adaptive-MSA and, stochastic decoding are shown. The values $\lambda = 0.05$ and $i_{\max} = 20$ are used for the (31,25) RS code. For the (63,55) RS code these values are $\lambda = 0.115$ and $i_{\max} = 5$. A scaling parameter of $\gamma = 1.33$ and the EM and IM lengths of $M = 50$ and $L = 8$ are used for RS codes. Stochastic decoding runs for 500 decoding cycles and 750 decoding cycles, between each adaptation, for the (31,25) and the (63,55) RS code, respectively. Similar to [47], after each iteration, i, the HDD is applied. Stochastic decoding provides performance close to the ABP for both codes and as shown in Figure 7–4(a), it outperforms the adaptive-MSA by more than 0.5 dB. To show the effect of EMs, Figure 7–4(b) depicts the performance of stochastic decoding with $M = 10$. As shown, an early error-floor is observed for short EM length.



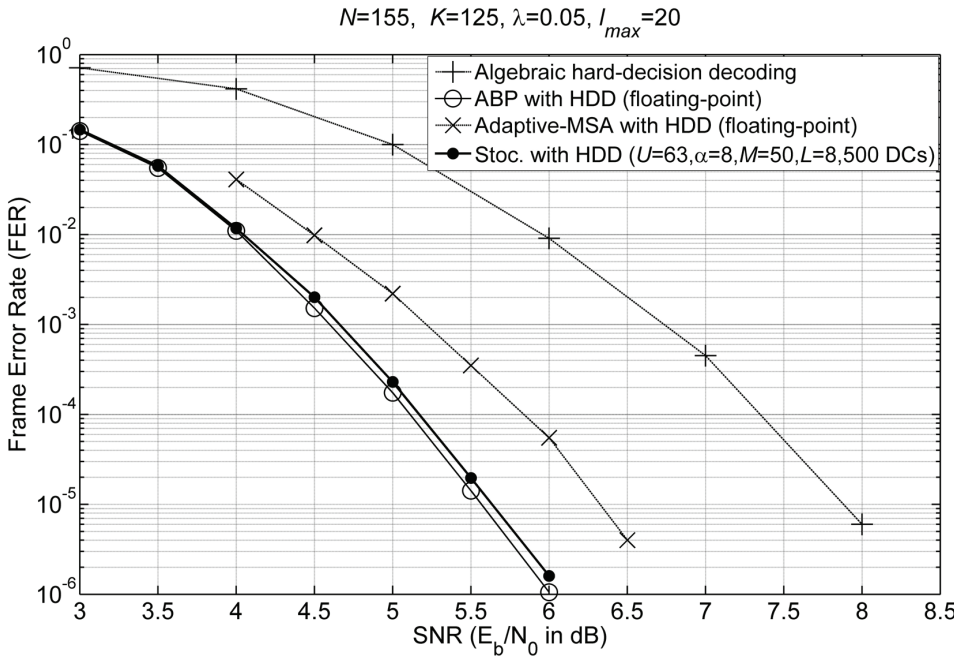
(a)



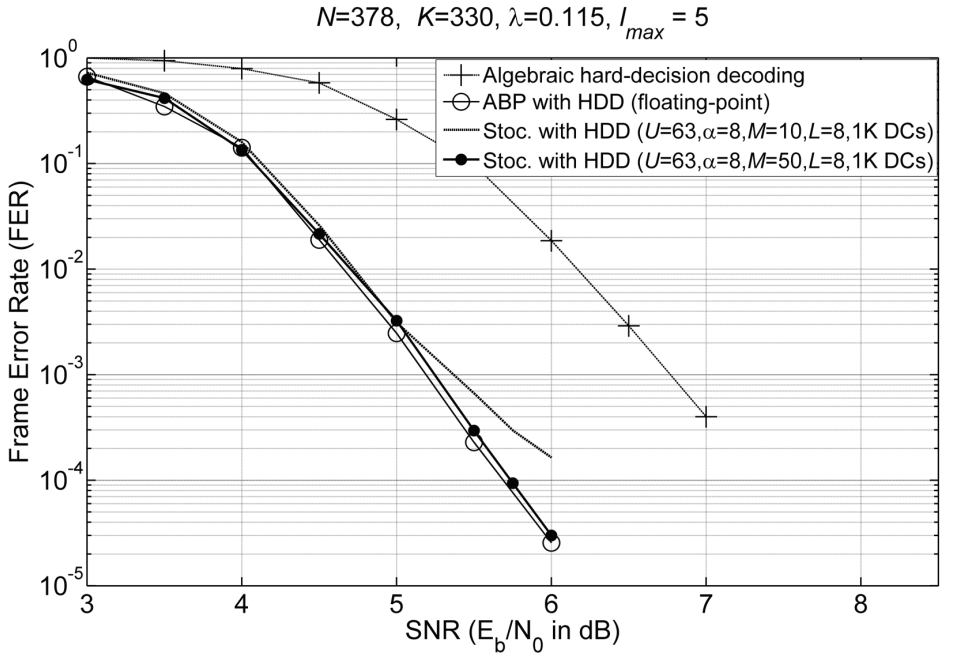
(b)

Figure 7–3: (a) Simulation results for a (128,120) BCH code. (b) Average number of decoding cycles for stochastic decoding of (128,120) BCH code.

Results for a (256,121) block turbo code based on (16,11) BCH component decoders and a (1024,676) block turbo code based on (32,26) BCH component decoders are shown in Figure 7–5. For turbo decoding of these codes,



(a)



(b)

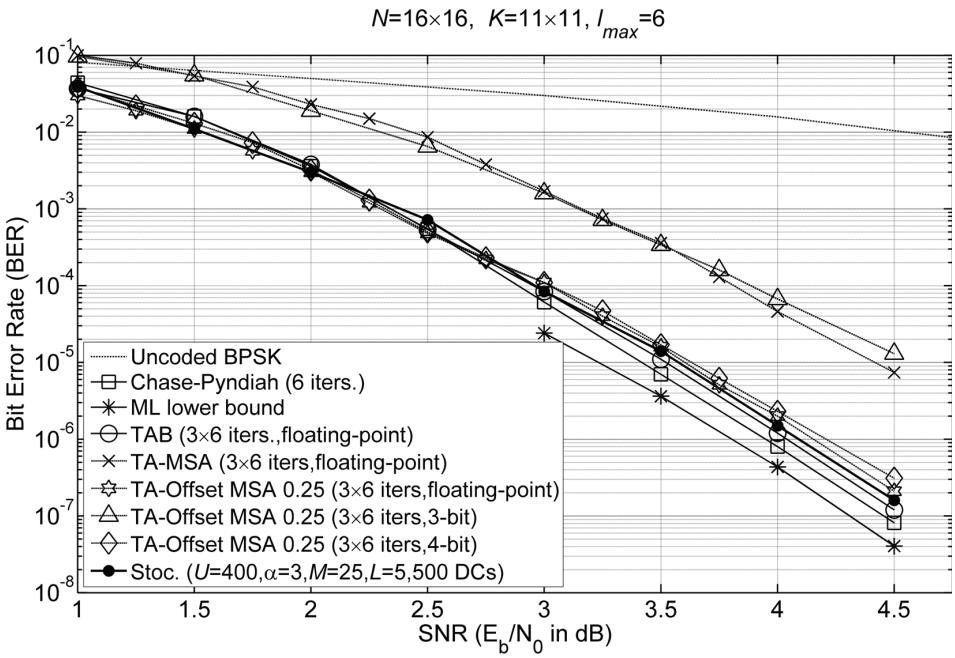
Figure 7–4: Simulation results for (a) a (31,25) RS code over GF(2^5) and (b) a (63,55) RS code over GF(2^6).

the traditional Chase-Pyndiah algorithm (with 6 global iterations and 16 error

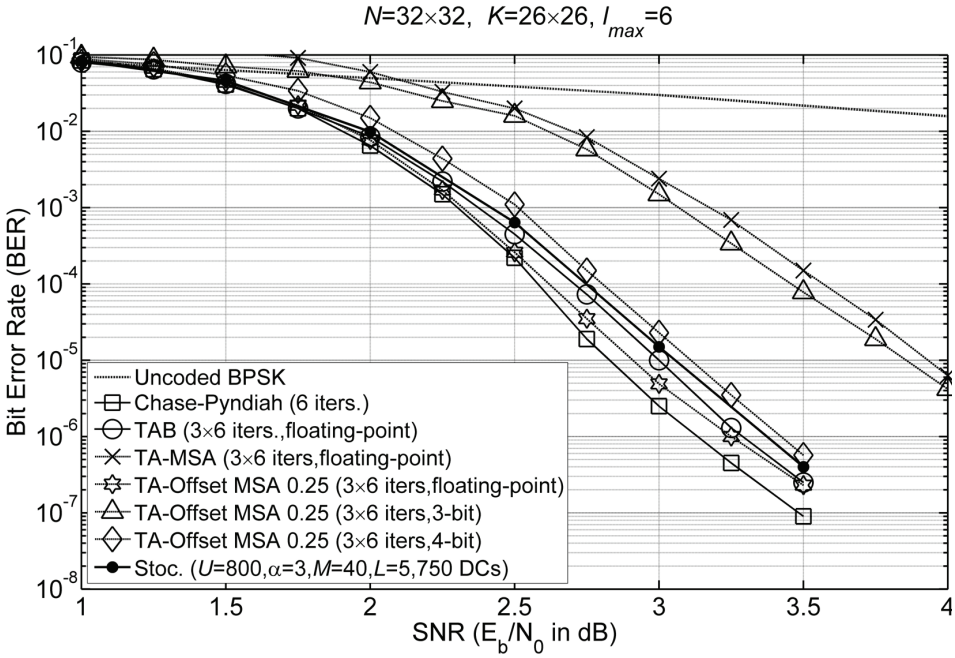
patterns), the TAB, the Turbo-oriented Adaptive MSA (TA-MSA), the Turbo-oriented Adaptive Offset MSA (TA-Offset MSA), and the stochastic decoding method (applied to the TAB algorithm) are employed for the SISO decoding algorithm during the iterative process. Note that only 3 local iterations are necessary during the BP process of the TAB algorithm and 6 global iterations are sufficient for the two decoding methods based on the TAB algorithm. In addition, no damping coefficient is necessary for the TAB algorithm. Instead, the reduction of the extrinsic information effect is done during the soft information computation [45, 46]. The EM and IM lengths of $M = 25$ and $L = 5$, and a fixed number of 500 decoding cycles are used for the (256,121) turbo code. These parameters for the (1024,676) turbo code are $M = 40$, $L = 5$, and 1K decoding cycles. For the (256,121) turbo code, the results for the floating-point TAB and the stochastic decoding method are close and show a decoding loss of about 0.1 dB compared to the classical Chase-Pyndiah decoding at 10^{-6} BER. For the (1024,676) turbo code, the decoding loss of stochastic decoding at 10^{-7} BER is about 0.1 dB and 0.3 dB, compared to the floating-point TAB and Chase-Pyndiah decoding, respectively. As shown, the TA-MSA results in about 1 dB loss compared to stochastic decoding, and the fixed-point TA-Offset MSA requires at least 4-bit precision to provide performance close to stochastic decoding.

7.4 Complexity Comparison and Trade-Offs

This section compares the complexity of nodes in stochastic decoding with their equivalent fully parallel fixed-point offset MSA implementation. As results in the previous section show, the MSA has a performance loss of more than 0.5 dB compared to stochastic decoding. Also, the performance loss of algebraic hard-decision decoding is more than 1.5 dB. Therefore, to have a fair complexity comparison, we compare the offset MSA with stochastic decoding.



(a)



(b)

Figure 7–5: Simulation results for (a) a (256,121) BCH block turbo code and (b) a (1024,676) BCH block turbo code.

As simulation results show, to have a performance close to stochastic decoding, the offset MSA needs at least 4 bits of precision.

A VN in the MSA obtains the sum of the all inputs and then subtracts each input from this value to obtain the output for each edge. The VN operation is usually done using two's complement arithmetic. A PN in the MSA is more complex and for each edge it needs to obtain the minimum absolute values of all inputs except the input received from that edge. A PN also calculates the sign-bit of each output by XORing all sign-bits of inputs except the sign-bits of the input of that edge. In the Offset MSA, an offset value is also added to each output. To reduce the hardware-complexity, the PN operation is usually done using sign-magnitude arithmetic; therefore, both two's complement to sign-magnitude (T's-SM) conversion and sign-magnitude to two's complement (SM-T's) conversion units are needed in VNs (or PNs). An efficient method to implement a PN in the MSA is to only find the first and the second minimum of all inputs. This method requires much fewer operations when compared to the conventional implementation of a PN (see [39] for details) and, hence, it is considered in this chapter. Table 7–1 shows the 2-input operations needed in a VN and a PN. As mentioned in previous chapters, a stochastic PN needs to calculate the XOR of all inputs and then XOR this value with the input bit of each edge to obtain the output bit for that edge. Therefore, the complexity of the entire stochastic PN is equivalent to only the sign-bit calculation in the MSA's PN. Using the proposed structure in Figure 3–5(b), stochastic VNs can be implemented based on $3d_v - 6$ subnodes where each subnode has one AND gate, one NOR gate, and one IM (except for exit subnodes). Stochastic VNs also use one comparator (to transform probabilities to stochastic streams [82]), one up/down counter, and d_v EMs.

Based on the maximum node degrees in the (63,55) RS code (i.e., d_v=34 and d_c=184) and the (1024,676) BCH turbo code (i.e., d_v=5 and d_c=16), we compared the FPGA implementation of the stochastic nodes with their

Table 7–1: Basic 2-input resources in the fixed-point Offset MSA and stochastic nodes (FX: Fixed-point, ADD: adder, SUB: subtractor, CMP: comparator, CNT: u/d counter).

Method	VN (degree d_v)	PN (degree d_c)
Offset MSA [39]	$d_v - 1$ FX ADDs, d_v FX SUBs, d_v T's-SM conversion	$d_c + \log_2 d_c - 2$ FX CMPs, $d_c + 4$ FX ADDs, $2d_c - 1$ binary XORs, 2 SM-T's conversion
Stochastic	1 FX CMP, 1 CNT, d_v EMs, $3d_v - 6$ AND and NOR, $2d_v - 6$ IMs	$2d_c - 1$ binary XORs

equivalent 4-bit offset MSA. Table 7–2 shows the implementation results on a Xilinx Virtex-4 LX200 device based on occupied 4-input look-up-tables and flip-flops. EMs and IMs with M=48 and L=8 were implemented using shift register look-up-tables as in Chapter 3. As shown, stochastic VNs have less complexity compared to VNs in the 4-bit offset MSA. For $d_v = 34$, a stochastic VN needs about 70% fewer look-up-tables and 81% fewer flip-flops. Stochastic PNs have much less complexity compared to PNs in the offset MSA. A $d_c = 16$ stochastic PN uses about 93% fewer look-up-tables than a $d_c = 16$ offset MSA PN. The complexity of a $d_c = 184$ stochastic PN node is even less than a $d_c = 16$ PN in the 4-bit offset MSA. For this reason, the complexity advantage over a $d_c = 184$ PN in the offset MSA follows and we did not consider the implementation. Such a node should calculate the first and the second minimum of 184 inputs and have a very high complexity. It is also important to note that an appealing complexity advantage of stochastic decoding is that it needs $\frac{W-1}{W}$ fewer wires in the interleaver, compared to the W-bit fixed-point MSA. As mentioned in previous chapters, this advantage is important because in the implementation of factor graphs, the number of (interleaver) wires directly translates to the area complexity and, in fact, it becomes the bottleneck of the overall hardware-complexity [13].

Table 7–2: Implementation comparison on a Xilinx Virtex-4 XC4VLX200-10FF1513 FPGA device (LUT: look-up-table, FF: flip-flop).

Method	VN d_v=5	PN d_c=16	VN d_v=34	PN d_c=184
4-bit Offset MSA	106 LUTs 48 FFs	325 LUTs 0 FFs	1363 LUTs 280 FFs	not considered
Stochastic	90 LUTs 24 FFs	22 LUTs 0 FFs	401 LUTs 52 FFs	245 LUTs 0 FFs

Since the parity-check matrix adaptation hardware in both stochastic and MSA decoding is similar, to provide an estimation of the order of area complexity and operating clock frequency for the stochastic decoding method, we instantiated all VNs, PNs and the interleaver of the (128,120) BCH code on the Virtex-4 LX200 FPGA device. This implementation occupied 11438 4-input look-up-tables (6% of available look-up-tables), 3184 flip-flops (1% of available flip-flops), and achieved a clock frequency of 180 MHz after place-and-route. In general, compared to the adaptive offset MSA or the ABP, the stochastic approach needs more clock cycles for decoding and therefore has a longer latency. However, resulting from their low hardware-complexity nodes and much alleviated routing problem, stochastic decoders can achieve higher clock frequency which helps them to provide an acceptable throughput. In addition, in stochastic decoders, the required average number of decoding cycles is usually much less than the maximum number of decoding cycles, especially at low BERs. The reason is that at low BERs there are only a few codewords that require a large number of decoding cycles to decode and thus the histograms of the number of decoding cycles used for decoding at low BERs are highly concentrated around the average number of decoding cycles. Therefore, instead of operating for a maximum number of decoding cycles for all received codewords, early termination methods can be exploited to terminate decoding as soon as a codeword is found for the sake of increasing the throughput.

As demonstrated in previous chapters, such a termination method has a significant impact on the throughput of a stochastic decoder and it also has a straightforward hardware implementation. For example, syndrome checking can be done based on XORing the sign-bits of up/down counters (i.e., hard-decisions). Figure 7–3(b) depicts the average number of decoding cycles for decoding the BCH code when syndrome checking is used as the termination criteria. As shown, at BERs less than 4.1×10^{-3}, the average number of decoding cycles is about 100 (i.e., a throughput of about 230 Mb/s with a 180 MHz clock frequency).

7.5 Conclusion

This chapter investigated the application of stochastic decoding to the important classes of RS, BCH, and block turbo codes. Simulation results demonstrated decoding performance close to the floating-point ABP and TAB. This chapter also discussed the hardware-complexity and the throughput of the stochastic approach and compared it with low-complexity fixed-point implementations of the ABP. It was shown that the hardware-complexity of stochastic decoding on factor graphs with nodes of high-degree is significantly lower than that of the offset MSA.

CHAPTER 8

Conclusion and Future Work

8.1 Advances

The edge-based rerandomization approach using EMs and TFMs, and the node-based rerandomization approach using MTFMs are proposed for stochastic decoding of state-of-the-art LDPC codes. The proposed approaches are the first stochastic approaches in the literature for stochastic decoding of state-of-the-art LDPC codes. They rely on the concept of regenerative and conservative bits, and efficient rerandomization of stochastic streams to alleviate the latching problem. It was shown that these approaches are able to decode state-of-the-art LDPC codes with competitive performance compared to the practical LDPC decoding approaches.

We proposed hardware architectures and discussed FPGA and ASIC implementations of the stochastic decoders. A (1024,512) and a (1056,528) EM-based LDPC decoder are implemented in FPGA. The (1056,528) EM-based decoder achieves a clock frequency of 222 MHz and a throughput of about 1.66 Gb/s at $E_b/N_0 = 4.25$ dB (a BER of 10^{-8}). The decoder latency is 3.3 μs. It provides decoding performance within 0.5 dB and 0.25 dB of the floating-point SPA with 32 and 16 iterations, respectively, and similar error-floor behavior. The decoder uses less than 40% of the look-up-tables, flip-flops and IO ports available on a Virtex-4 XC4VLX200 FPGA device. The proposed EM-based

stochastic decoders are among the fastest and most resource-efficient FPGA LDPC decoders reported in the literature.

The EM approach is resource-efficient in FPGAs and thus it is suitable for FPGA implementations of stochastic LDPC decoders. However, this approach consumes a considerable silicon area when implemented in ASIC. The TFM approach is therefore proposed to significantly reduce the silicon area consumption of ASIC stochastic LDPC decoders. Various hardware architectures for the implementation of TFMs are discussed. The TFM approach is applied for ASIC implementation of a (1056,528) TFM-based LDPC decoder. It was shown that the (1056,528) decoder with 8-bit TFMs occupies 40% and 65% less silicon area compared to (1056,528) decoders with 32-bit and 64-bit EMs, respectively. Additionally, it was demonstrated that TFMs are able to provide similar or better decoding performance compared to EMs.

Both EM and TFM approaches are based on the edge-based rerandomization in which rerandomization units are assigned to each outgoing edge of VNs. As a result, the decoder uses a high number of EMs or TFMs. We proposed the node-based rerandomization approach using MTFMs in which one rerandomization unit in each VN is used. This approach significantly reduces the number of rerandomization units used in a stochastic decoder and thus the overall hardware-complexity of the stochastic decoder. The MTFM approach is applied for ASIC implementation of a fully parallel stochastic decoder that decodes the (2048,1723) RS-based LDPC code from the IEEE 802.3an (10GBASE-T) standard. The decoder occupies a silicon core area of 6.38 mm^2 in CMOS 90 nm technology, achieves a maximum clock frequency of 500 MHz, and provides a maximum core throughput of 61.3 Gb/s. The decoder latency is 800 ns. The decoder has good decoding performance and error-floor behavior and provides a BER of about 4×10^{-13} at $E_b/N_0 = 5.15$ dB. The

decoder's area-per-coded-bit efficiency is 3115 μm^2 and its throughput-per-coded-bit-per-area efficiency is 4.69 Mb/s/mm^2 (in CMOS 90 nm technology). To the best of our knowledge, this decoder is the most area-efficient fully parallel soft-decision LDPC decoder and it is one of the fastest fully parallel soft-decision LDPC decoders reported in the literature.

In addition to LDPC decoding, we proposed the novel application of stochastic decoding for joint decoding of LDPC codes and partial-response channels. We considered the dicode partial-response channel and the EPR4 partial-response channel, which is a practical channel model considered in magnetic recording applications. The hardware architectures of stochastic dicode and EPR4 channel detectors were presented. We demonstrated that in the case of the dicode channel whose corresponding message-passing graph is acyclic, it is possible to perform stochastic detection without using rerandomization units. For the case of the EPR4 channel, whose message-passing graph has a high number of length-4 cycles, TFMs are used to efficiently rerandomize stochastic streams and alleviate the latching problem. Results demonstrated the applicability of the stochastic approach for joint decoding of LDPC codes and partial-response channels.

Finally, we investigated the application of the stochastic approach for decoding linear block codes with high-density parity-check matrices on factor graphs. We considered the stochastic decoding of RS codes, BCH codes and BCH-based block turbo codes. Results showed that the stochastic approach can be exploited in SISO decoding based on the ABP and the TAB. They also demonstrated decoding performance close to floating-point iterative SISO decoding while offering nodes with considerably lower complexity compared to fixed-point SISO decoding.

8.2 Future Work

Inspired by the results and based on intuitions gained from this work on stochastic decoding of LDPC codes, several related research projects have begun. In [78], non-binary EMs were used and stochastic decoding was extended for decoding non-binary LDPC codes over GF(q). In [77], non-binary versions of TFMs were used to reduce the decoding latency and increase the throughput in stochastic decoding of non-binary LDPC codes over GF(q). In [53], it was shown that the concept of TFMs can be applied in the log-domain. In this respect, half-stochastic LDPC decoding was proposed in which VNs are based on the SPA (in the log-domain) while PNs are in the stochastic domain. Also, the concept of redecoding in stochastic LDPC decoders was introduced in [53]. It was shown that as a result of the random decoding trajectory in stochastic decoding, it is possible to repeat the decoding experiment for several rounds in order to improve the decoding performance and lower the error-floors of LDPC codes.

In addition to the above-mentioned ongoing research projects, there are other research possibilities that can be considered as related future work. These potential research projects are briefly discussed as follows.

8.2.1 Power-Efficient Stochastic LDPC Decoders

The main focus of this dissertation was on the decoding performance, silicon area consumption, throughput, and decoding latency of stochastic decoding. However, because the stochastic decoding approach has low silicon area consumption, fast decoding convergence, and uses fewer physical wires in the decoder chip (compared to conventional decoding approaches), it also has a high potential for power-efficient LDPC decoding. In this respect, the power-consumption analysis and comparison of EM, TFM, and MTFM stochastic approaches would be valuable research work.

8.2.2 Reduced-Latency Stochastic LDPC Decoders

In general, stochastic decoding has a longer decoding latency compared to other decoding approaches. However, as histograms of the number of decoding cycles used for decoding show, there are only few codewords that require a long decoding latency; the majority of codewords decode very fast. In Chapter 5, we used a postprocessing technique to enable the (2048,1723) MTFM-based LDPC decoder to achieve a good BER decoding performance with less decoding latency (i.e., 800 ns). Although this order of latency is acceptable for many applications, there might be room for more improvement. In addition, our understanding of the effectiveness of the postprocessing technique was based on heuristics and BER simulations. A better understanding can be developed to both qualitatively and quantitatively justify the effectiveness of this technique.

8.2.3 Reconfigurable Stochastic LDPC Decoders

The FPGA and ASIC implementations presented in this dissertation are based on the fully parallel design approach. In this approach, the whole factor graph is implemented in hardware; therefore, this approach is usually suitable for applications where a fixed LDPC code is used, such as the 10Gb/s Ethernet (10GBASE-T) standard. There are applications that are required to support different LDPC codes (with different lengths and rates) depending on the desired decoding performance and the channel condition (e.g., WiFi and WiMAX). In such applications, reconfigurable stochastic decoders are required whose architectures and implementations need to be investigated.

One possible approach to support reconfigurability is based on the partially parallel LDPC decoding, where only a portion of the factor graph is implemented in hardware. In applications such as WiMAX and WiFi, multiple LDPC codes are designed in a way that allows for unified partially parallel LDPC decoding. In a partially parallel stochastic LDPC decoder, memory

blocks and the required control logic should be used to save the state of rerandomization units and manage message-passing between different portions of the factor graph. Also, different decoding schedules can be used in partially parallel decoders whose effects on the latency and throughput of stochastic decoding need to be investigated.

Another approach which can be suitable for some applications is to build a fully parallel decoder that support different LDPC codes. For example, in the WiMAX standard [3] there are multiple LDPC codes with different code rates (which range from 1/2 to 5/6) and different code lengths (which range from 576 to 2304). All the LDPC codes in this standard can be generated by removing some rows and/or columns of the parity check matrix of the (2304,1152) LDPC code (with rate 1/2). Therefore, it is possible to implement a fully parallel stochastic LDPC decoder that decodes the (2304,1152) LDPC code and to add extra control logic to deactivate some VNs, PNs and edges according to the parity-check matrix of the target LDPC code. In [23], a bidirectional interleaver architecture was proposed that can be used in fully parallel stochastic LDPC decoders that support multi LDPC codes.

8.2.4 Different Channel Models

The AWGN and partial-response channels are the main channel models used in this dissertation for stochastic decoding. Investigating the decoding performance and hardware-complexity tradeoffs for communications systems that have different channel characteristics (e.g., fading channels) would be an interesting contribution.

8.2.5 Asynchronous Stochastic Decoding

The Muller's C-element (also known as C-gate) is an asynchronous logic component whose output reflects the inputs when the states of all inputs

agree [68]. The output then remains in this state until all the inputs transit to another state. The operation of a C-element is similar to the basic operation of a stochastic VN and the concept of regenerative and conservative bits introduced in this dissertation. In a stochastic VN, if all the input bits agree, the outgoing bit is regenerative and it is equal to the input bits. Otherwise, the VN remains in a hold state. In addition, PNs in stochastic decoding are based on combinational logic (XOR gates); therefore, they can be implemented as asynchronous components. In this regard, asynchronous stochastic decoding, at the both algorithm and hardware implementation levels, can be considered as possible future work.

8.2.6 Quantum Stochastic Decoding

In stochastic computation, probabilities are encoded into streams of stochastic bits and computation is performed on stochastic bit streams. In other words, the statistic of a bit stream represents the original (encoded) probability. In quantum computation, a quantum bit (referred to as "qubit") can be 0, 1, or a superposition of both [69]. It is interesting to study the feasibility of stochastic representation using quantum bits and to investigate the expression of stochastic VNs' and PNs' operations using quantum logic gates [69]. Such a research study can potentially open doors for the application of quantum computation for iterative decoding on graphs.

APPENDIX A

Decoding Performance Results for Various LDPC Codes

This appendix reports BER decoding performance results for stochastic decoding of different LDPC codes. The reader should note that these results are presented to demonstrate the applicability of the proposed stochastic approaches for decoding various LDPC codes with different lengths, rates, and node degrees. In this regard, the parameters used for stochastic decoding of these LDPC codes are not necessarily optimized for the best possible decoding performance and/or lowest possible decoding latency.

A.1 Results for EM-Based Decoding

Figures A–1 to A–4 show the decoding performance of the EM approach. In all simulations, an early termination criterion (based on syndrome checking) is used until a maximum number of decoding cycles has been exhausted. The (2000,1000) LDPC code used in simulations is a regular code with a girth of 8. The (1536,1024) LDPC code and the (576,288) LDPC code are irregular codes that belong to the WiMAX standard [3]. Also, the (648,540) LDPC code is an irregular code from the WiFi standard [4].

A.2 Results for TFM-Based Decoding

Figures A–5 and A–6 show the decoding performance of the TFM approach. In all simulations, an early termination criterion (based on syndrome

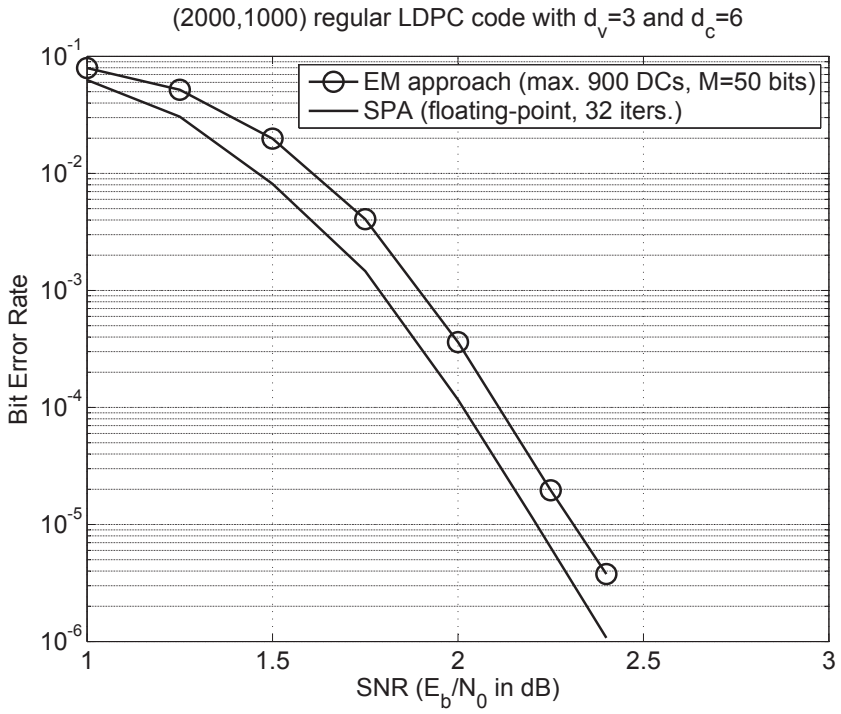


Figure A–1: Performance of the EM approach for decoding a (2000,1000) LDPC code.

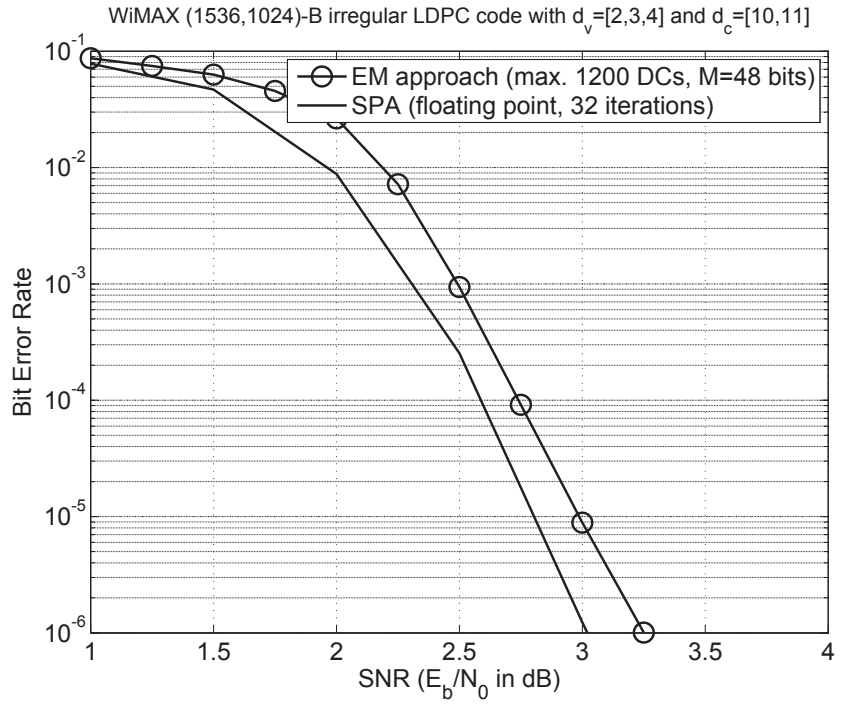


Figure A–2: Performance of the EM approach for decoding a (1536,1024) LDPC code.

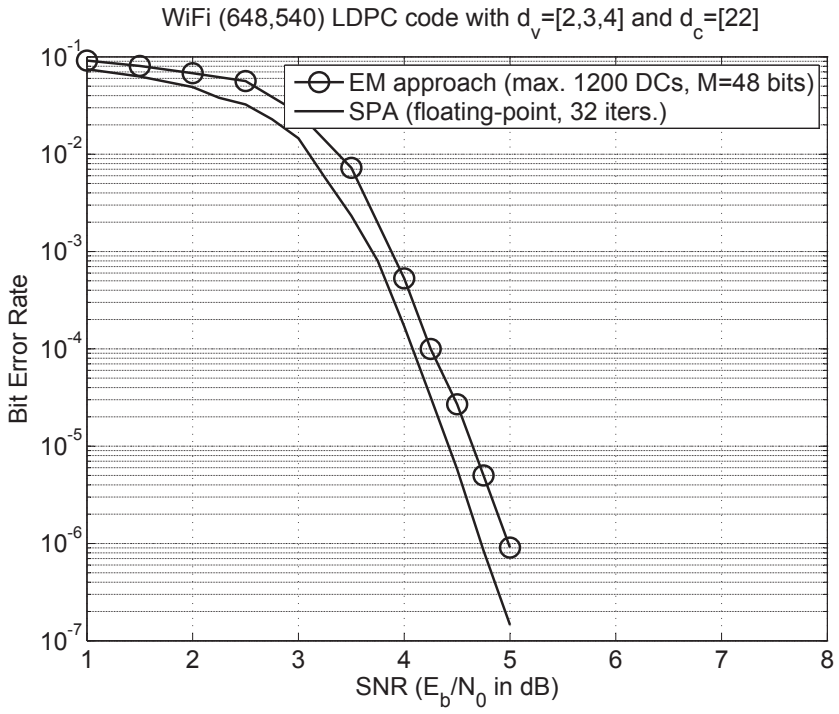


Figure A–3: Performance of the EM approach for decoding a (648,540) LDPC code.

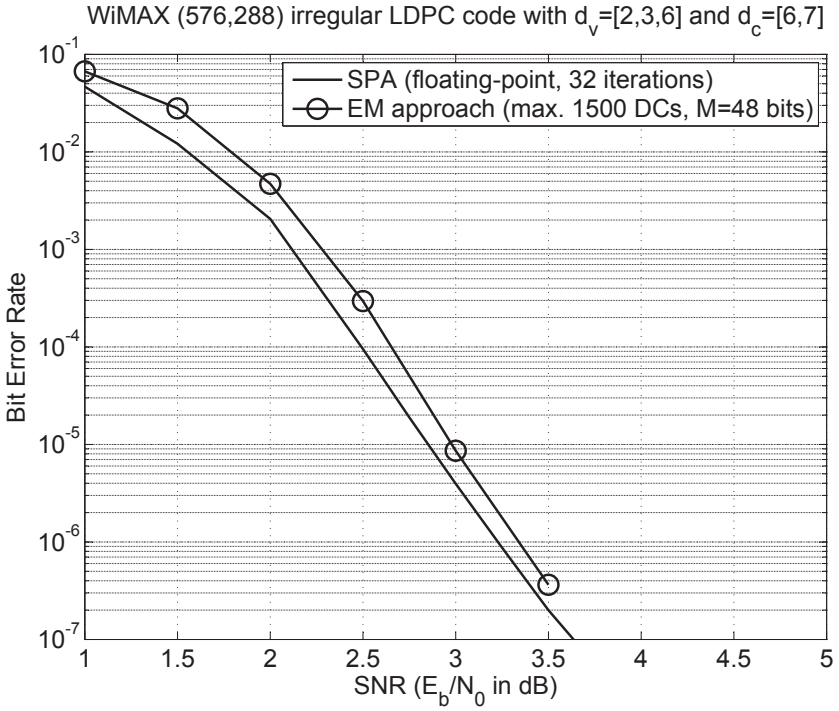


Figure A–4: Performance of the EM approach for decoding a (576,288) LDPC code.

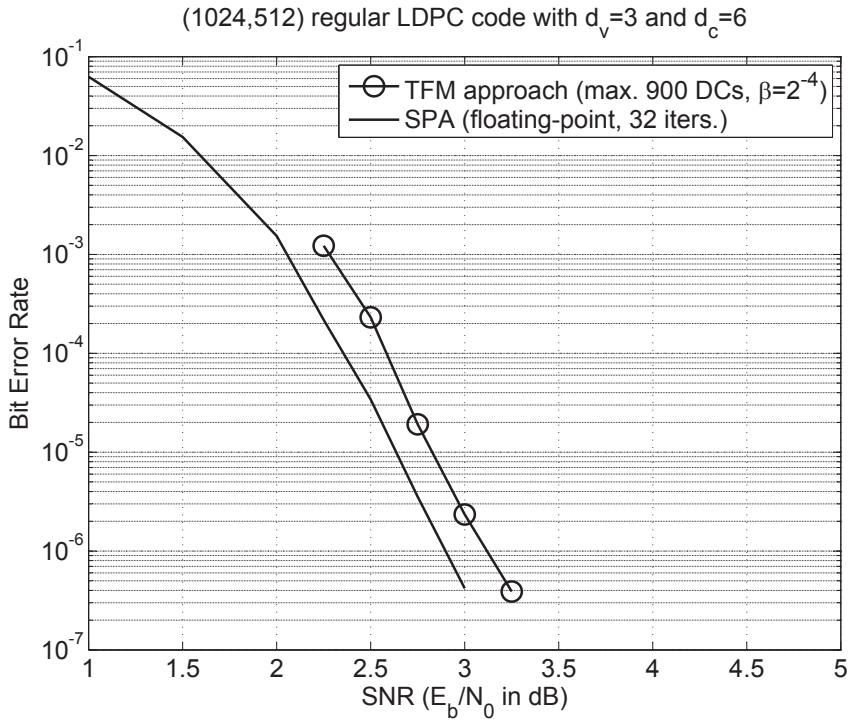


Figure A–5: Performance of the TFM approach for decoding a (1024,512) LDPC code.

checking) is used until a maximum number of decoding cycles has been exhausted. The (1024,512) LDPC code is a regular code with a grith of 8. The (648,324) LDPC code is an irregular LDPC code that belongs to the WiFi standard [4].

A.3 Results for MTFM-Based Decoding

To demonstrate the applicability of the MTFM approach for decoding other LDPC codes, Figure A–7 depicts the performance of the MTFM approach for decoding a (1057,813) LDPC code chosen from [5]. This LDPC code has maximum PN and VN degrees of 18 and 4, respectively. Figure A–7 also shows the decoding performance of the floating-point SPA with 32 and 16 iterations. At least 40 frame errors were counted for BERs less than 10^{-9}.

Similar to the proposed (2048,1723) stochastic LDPC decoder in Chapter 5, we used 6-bit input probabilities, 12-bit reduced-complexity MTFMs (with

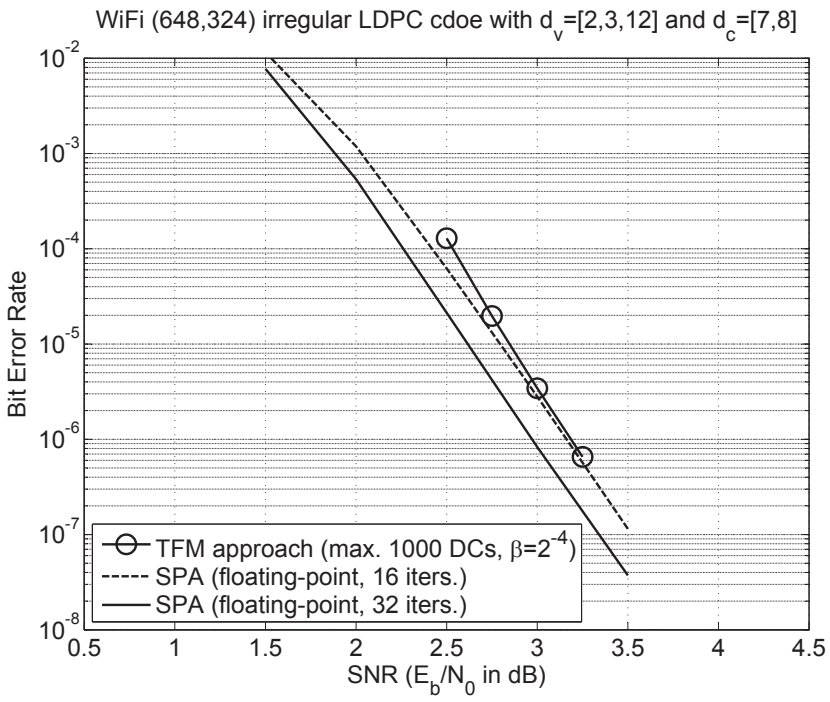


Figure A–6: Performance of the TFM approach for decoding a (648,324) LDPC code.

$T_u = d_v$ and $T_m = d_v/2$), and the same postprocessing and redecoding scheme in our simulations. Also, an early termination criterion is used until a maximum of 400 decoding cycles has been exhausted. As shown, the proposed MTFM approach (with postprocessing and redecoding scheme) has good decoding performance behavior in low BER regimes and at $E_b/N_0 = 5.75$ dB it outperforms the SPA with 32 iterations.

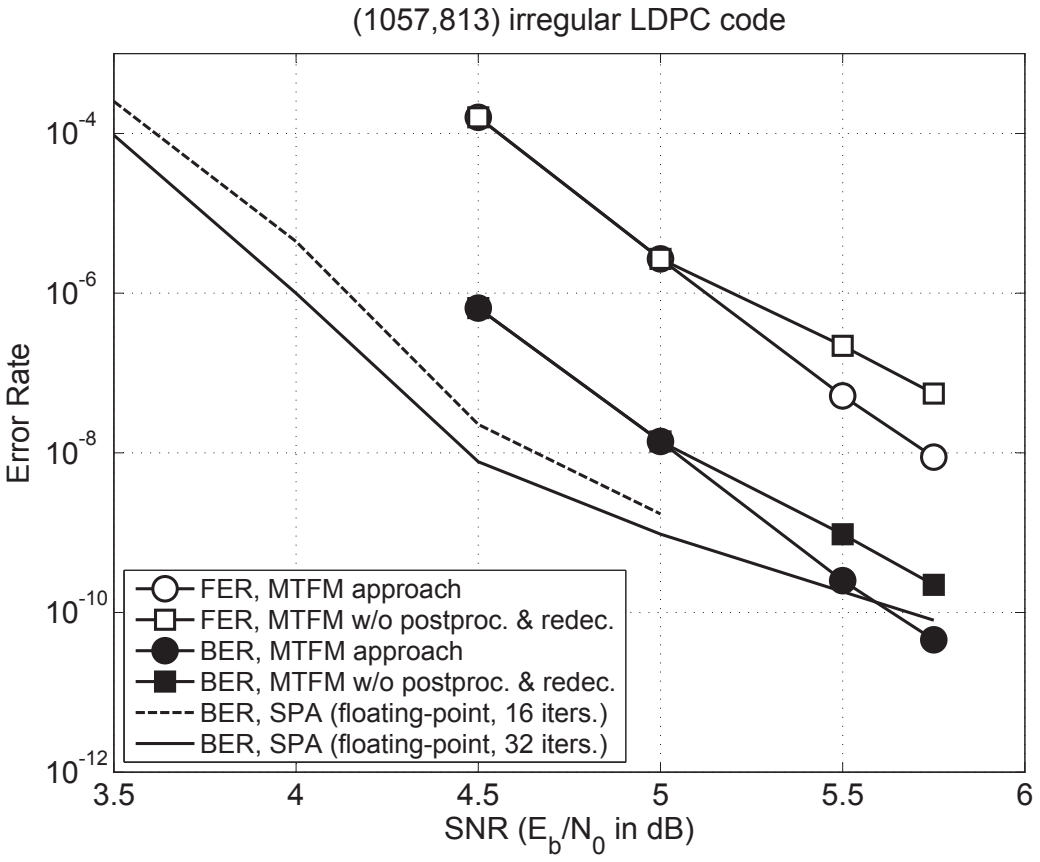


Figure A–7: Performance of the MTFM approach for decoding a (1057,813) LDPC code. An early termination criterion until a maximum of 400 decoding cycles is used.

REFERENCES

[1] The Digital Video Broadcasting standard, www.dvb.org.

[2] The IEEE P802.3an 10GBASE-T Task Force, www.ieee802.org/3/an.

[3] The IEEE 802.16 Working Group, http://www.ieee802.org/16/.

[4] The IEEE 802.11n Working Group, http://www.ieee802.org/11/.

[5] D. J. C. MacKay. Encyclopedia of Sparse Graph Codes, http://www.inference.phy.cam.ac.uk/mackay/codes/.

[6] Altera Corporation. *Stratix Device Handbook*. www.altera.com.

[7] R. Ananth. *A Field Programmable Stochastic Computer for Signal Processing Applications*. M.A.Sc. Thesis, University of Toronto, Canada, 1992.

[8] A. Anastasopoulos. A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution. In *IEEE Global Telecomm. Conference*, volume 2, pages 1021–1025, Nov. 2001.

[9] K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones, and F. Pollara. The development of turbo and LDPC codes for deep-space applications. *Proceedings of the IEEE*, 95(11):2142–2156, Nov. 2007.

[10] M. Ardakani and F. R. Kschischang. Gear-shift decoding. *Communications, IEEE Transactions on*, 54(7):1235–1242, July 2006.

[11] M. Arzel, C. Lahuec, F. Seguin, D. Gnaedig, and M. Jezequel. Analog slice turbo decoding. In *IEEE International Symposium on Circuits and Systems*, pages 332–335, May 2005.

[12] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding: Turbo codes. In *Proceedings of the IEEE International Conference on Communications*, pages 1064–1070, May 1993.

[13] A. Blanksby and C. J. Howland. A 690-mw 1-Gb/s 1024-b rate-1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37(3):404–412, March 2002.

[14] E. Boutillon, J. Castura, and F. R. Kschischang. Decoder-first code design. In *2nd International Symposium on Turbo Codes and Related Topics*, pages 459–462, Brest, France, Sept. 2002.

[15] T. L. Brandon, R. Hang, G. Block, V. Gaudet, B. F. Cockburn, S. L. Howard, C. Giasson, K. Boyle, S. Sheik Zeinoddin, A. Rapley, S. Bates, D. G. Elliott, and C. Schlegel. A scalable LDPC decoder ASIC architecture with bit-serial message exchange. *Integration, the VLSI Journal*, 41(3):385–398, May 2008.

[16] B. Brown and H. Card. Stochastic neural computation I: Computational elements. volume 50, pages 891–905, Sept. 2001.

[17] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu. Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications*, 53(7):1232–1232, July 2005.

[18] J. Chen, R. M. Tanner, C. Jones, and Yan Li. Improved min-sum decoding algorithms for irregular LDPC codes. In *Proceedings of International Symposium on Information Theory (ISIT)*, pages 449–453, 2005.

[19] F. Chiaraluce and R. Garello. Extended Hamming product codes analytical performance evaluation for low error rate applications. *IEEE Transactions on Wireless Communications*, (6):2353–2361, Nov. 2004.

[20] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5:58–60, Feb. 2001.

[21] G. Colavolpe and G. Germi. On the application of factor graphs and the sum-product algorithm to ISI channels. *IEEE Transactions on Communications*, 53(5):818–825, May 2005.

[22] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker. Applications of error-control coding. *IEEE Transactions on Information Theory*, 44(6):2531 –2560, Oct. 1998.

[23] K. Cushon, W. J. Gross, and S. Mannor. Bidirectional interleavers for LDPC decoders using transmission gates. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 232–237, Tampere, Finland, 2009.

[24] A. Darabiha. *VLSI Architchtures for Multi-Gbps Low-Denisty Parity-Check Decoders*. Ph.D. Thesis, University of Toronto, Canada, 2008.

[25] A. Darabiha, A. Chan Carusone, and F. R. Kschischang. Block-interlaced LDPC decoders with reduced interconnect complexity. *IEEE*

Transactions on Circuits and Systems-II: Express briefs, 55(1):74–78, Jan. 2008.

[26] A. Darabiha, A. Chan Carusone, and F. R. Kschischang. Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity. In *IEEE International Symposium on Circuits and Systems*, pages 5194–5197, San Jose, USA, May 2005.

[27] A. Darabiha, A. Chan Carusone, and F. R. Kschischang. A bit-serial approximate min-sum LDPC decoder and FPGA implementation. In *IEEE International Symposium on Circuits and Systems*, pages 149–152, Greece, May 2006.

[28] A. Darabiha, A. Chan Carusone, and F. R. Kschischang. A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-um CMOS. In *Custom Integrated Circuits Conference*, pages 459–462, USA, Sept. 2007.

[29] A. Dinu, M. N. Cirstea, and M. McCormick. Stochastic implementation of motor controllers. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 639–644, July 2002.

[30] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin. A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols. *IEEE Communications Letters*, 7(7):317–319, July 2003.

[31] M. Fu. Stochastic models for turbo decoding. In *IEEE International Conference on Communications*, volume 1, pages 668–672, May 2005.

[32] B. Gaines. *Advances in Information Systems Science*, chapter 2, pages 37–172. Plenum, New York, 1969.

[33] R. G. Gallager. *Low Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.

[34] R. G. Gallager. Low density parity check codes. *IRE Transactions Information Theory*, 8:21–28, Jan. 1962.

[35] V. Gaudet and G. Gulak. A 13.3-Mb/s 0.35μm CMOS analog turbo decoder IC with a configurable interleaver. *IEEE Journal of Solid-State Circuits*, 38(11):2010–2015, Nov. 2003.

[36] V. Gaudet and A. Rapley. Iterative decoding using stochastic computation. *Electronics Letter*, 39(3):299–301, Feb. 2003.

[37] W. J. Gross, V. Gaudet, and A. Milner. Stochastic implementation of LDPC decoders. In *Proceedings of the 39th Asilomar Conference on*

Signals, Systems, and Computers, pages 713–717, Pacific Grove, CA, Nov. 2005.

[38] F. Guilloud, E. Boutillon, and J.-L. Danger. λ-min decoding algorithm of regular and irregular LDPC codes. In *Proceedings of the 3rd International Symposium on Turbo Codes*, pages 451–454, Brest, France, 1-5 Sept. 2003.

[39] K. K. Gunnam, G. S. Choi, and M. B. Yeary. A parallel VLSI architecture for layered decoding for array LDPC codes. In *Proceedings of the 20th International Conference on VLSI Design*, pages 738–743, Washington, DC, USA, 2007.

[40] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman. VLSI architectures for layered decoding for irregular LDPC codes of WiMax. In *IEEE International Conference on Communications*, pages 4542–4547, June 2007.

[41] S. Hemati, A. Banihashemi, and C. Plett. A 0.18μm analog min-sum iterative decoder for a (32,8) low-density parity-check (LDPC) code. *IEEE Journal of Solid-State Circuits*, 41(11):2531–2540, Nov. 2006.

[42] S. Howard, C. Schlegel, and V. Gaudet. A degree-matched check node approximation for LDPC decoding. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1131–1135, Adelaide, Australia, 4-9 Sept. 2005.

[43] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold. Regular and irregular progressive edge-growth Tanner graphs. *IEEE Transactions on Information Theory*, 51(1):386–398, Jan. 2005.

[44] J. Chen and M. P. C. Fossorier. Density evolution for two improved BP-based decoding algorithms of LDPC codes. *IEEE Communications Letter*, (5):208–210, May 2002.

[45] C. Jego and W. J. Gross. Turbo decoding of product codes based on the modified adaptive belief propagation algorithm. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 641–645, June 2007.

[46] C. Jego and W. J. Gross. Turbo decoding of product codes using belief propagation. *IEEE Transactions on Communications*, (10):2864–2867, Oct. 2009.

[47] J. Jiang and K. R. Narayanan. Iterative soft-input soft-output decoding of Reed-Solomon codes by adapting the parity-check matrix. *IEEE Transactions on Information Theory*, 52(8):3746–3756, Aug. 2006.

[48] M. Jiang, C. Zhao, Z. Shi, and Y. Chen. An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes. *IEEE Communications Letter*, 9:814–816, Sept. 2005.

[49] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Philadelphia, PA: SIAM, 1995.

[50] F. R. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, Feb 2001.

[51] B. M. Kurkoski, P. H. Siegel, and J. K. Wolf. Joint message-passing decoding of LDPC codes and partial-response channels. *IEEE Transactions on Information Theory*, 48(6):1410–1422, June 2002.

[52] B. M. Kurkoski, P. H. Siegel, and J. K. Wolf. Joint message-passing decoding of LDPC codes and partial-response channels (correction). *IEEE Transactions on Information Theory*, 49(8):2076–2076, Aug. 2003.

[53] F. Leduc-Primeau, S. Hemati, W. J. Gross, and S. Mannor. A relaxed half-stochastic iterative decoder for LDPC codes. In *IEEE Global Telecommunications Conference*, pages 1–6, Nov. 30 - Dec. 4 2009.

[54] L. Liu and C.-J. R. Shi. Sliced message passing: High throughput overlapped decoding of high-rate low density parity-check codes. *IEEE Transactions on Circuits and Systems I*, 55:3697–3710, Dec. 2008.

[55] F. Lustenberger, M. Helfenstein, G. Moschytz, H.-A. Loeliger, and F. Tarkoy. All-analog decoder for a binary (18,9,5) tailbiting trellis code. In *European Solid-State Circuits Conference*, pages 362–365, 1999.

[56] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45:399–432, March 1999.

[57] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In *Proceedings of the 5th IMA Conference on Cryprography and Coding*, pages 100–111, Berlin, Germany, 1995.

[58] D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letter*, 32(18):1645–1646, 1996.

[59] S. Mannor, S. J. Shamma, and G. Arslan. Online calibrated forecasts: Memory efficiency versus universality for learning in games. *Machine Learning*, 67(1-2):77–115, 2007.

[60] M. Mansour and N. Shanbhag. High-throughput LDPC decoders. *IEEE Communications Magazine*, 11(6):976–996, Dec. 2003.

[61] M. Mansour and N. Shanbhag. A 640-Mb/s 2048-bit programmable LDPC decoder chip. *IEEE Journal of Solid-State Circuits*, 41(3):684–698, March 2006.

[62] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communications*, 16:140–152, Feb. 1998.

[63] N. Mobini, A. H. Banihashemi, and S. Hemati. A differential binary message-passing LDPC decoder. *IEEE Transactions on Communications*, 57(9):2518–2523, Sept. 2009.

[64] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25 μm BiCMOS tailbiting MAP decoder. In *IEEE Custom Integrated Circuits Conference*, pages 356–357, Feb. 2000.

[65] T. Mohsenin and B. M. Baas. Split-row: A reduced complexity, high throughput LDPC decoder architecture. In *IEEE International Conference of Computer Design*, 2006.

[66] T. Mohsenin and B. M. Baas. High-throughput LDPC decoders using a multiple split-row method. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2007.

[67] T. Mohsenin, D. Truong, and B. Baas. A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders. *IEEE Transcations on Circuits and Systems I*, pages 1048–1061, May 2010.

[68] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of International Symposium Theory of Switching (Part 1)*, pages 204–243, Harvard University Press, 1959.

[69] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge Uinverstiy Press, 2000.

[70] N. Onizawa, T. Ikeda, T. Hanyu, and V. Gaudet. 3.2-Gb/s 1024-b rate-1/2 LDPC decoder chip using a flooding-type update-schedule algorithm. In *Proceedings of the 50th IEEE Midwest Symposium on Circuits and Systems*, pages 217–220, Brest, France, Aug. 2007.

[71] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic, 1970.

[72] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

[73] R. Pyndiah. Near optimum decoding of product codes: Block turbo codes. *IEEE Transactions on Communications*, pages 1003–1010, Aug. 1998.

[74] A. Rapley, C. Winstead, V. Gaudet, and C. Schlegel. Stochastic iterative decoding on factor graphs. In *Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics*, pages 507–510, Brest, France, Sept. 2003.

[75] T. J. Richardson, M. A. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47:619–637, Feb. 2001.

[76] T. J. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47:599–618, Feb. 2001.

[77] G. Sarkis and W. J. Gross. Reduced-latency stochastic decoding of LDPC codes over GF(q). In *Proceedings of the European Wireless Conference*, pages 994–998, April 2010.

[78] G. Sarkis, S. Mannor, and W. J. Gross. Stochastic decoding of LDPC codes over GF(q). In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1–5, Dresden, Germany, June 2009.

[79] C. B. Schlegel and L. C. Perez. *Trellis and Turbo Coding*. IEEE Press, 2004.

[80] S. Seo, T. Mudge, Y. Zhu, and C. Chakrabarti. Design and analysis of LDPC decoders for software defined radio. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 210–215, Shanghai, China, Oct. 2007.

[81] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[82] S. Sharifi Tehrani, W. J. Gross, and S. Mannor. Stochastic decoding of LDPC codes. *IEEE Communications Letter*, 10(10):716–718, Oct. 2006.

[83] S. Sharifi Tehrani, C. Jego, B. Zhu, and W. J. Gross. Stohastic decoding of linear block codes with high-densiy parity-check matrices. *IEEE Transactions on Signal Processing*, 56(11):5733–5739, Nov. 2008.

[84] S. Sharifi Tehrani, S. Mannor, and W. J. Gross. Survey of stochastic computation on factor graphs. In *the 37th IEEE International Symposium on Multiple-Valued Logic*, pages 54–59, Oslo, Norway, May 2007.

[85] S. Sharifi Tehrani, S. Mannor, and W. J. Gross. Fully parallel stochastic LDPC decoders. *IEEE Transactions on Signal Processing*, 56(11):5692–5703, Nov. 2008.

[86] S. Sharifi Tehrani, S. Mannor, and W. J. Gross. An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS)*, pages 255–260, Shanghai, China, Oct. 2007.

[87] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross. Tracking forecast memories in stochastic decoders. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 561–564, Taipei, Taiwan, April 2009.

[88] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross. Tracking forecast memories for stochastic decoding. *Invited paper by Journal of Signal Processing Systems, Special Issue on the DISPS Track of IEEE ICASSP 2009*, Springer Publishing, To appear (online publication: Jan. 2010), DOI: 10.1007/s11265-009-0441-5, Available: http://www.springerlink.com/content/9j386874m12681r0/.

[89] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor S. Hemati, and W. J. Gross. Majority-based tracking forecast memories for stochastic LDPC decoding. *IEEE Transactions on Signal Processing*, 58(9):4883–4896, Sept. 2010.

[90] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta. A 135Mb/s DVB-S2 compliant CODEC based on 64,800b LDPC and BCH codes. *IEEE ISSCC Digest of Technical Papers*, Feb. 2005.

[91] D. Vogrig, A. Gerosa, A. Neviani, A. Graell I Amat, G. Montorsi, and S. Benedetto. A 0.35μm CMOS analog turbo decoder for the 40-bit rate 1/3 UMTS channel code. *IEEE Journal of Solid-State Circuits*, 40(3):753–762, March 2005.

[92] Z. Wang and Z. Cui. Low-complexity high-speed decoder design for quasi-cyclic LDPC codes. *IEEE Transactions on VLSI Systems*, 15(1):104–114, Jan. 2007.

[93] A. D. Weathers, S. A. Altekar, and J. K. Wolf. Distance spectra for PRML channels. *IEEE Transactions on Magnetics*, 33(5):2809–2811, Sept. 1997.

[94] N. Wiberg. *Codes and Decoding on General Graphs*. Ph.D. Thesis, Dept. of Electrical Engineering, Linkoping University, Sweden, 1996.

[95] C. Winstead. Error-control decoders and probabilistic computation. In *Tohoku University 3rd Student Organizing International Mini-Conference on Information Electronics Systems (SOIM-COE)*, pages 349–352, Sendai, Japan, Oct. 2005.

[96] C. Winstead, J. Dai, S. Yu, C. Myers, R. Harrison, and C. Schlegel. CMOS analog MAP decoder for (8,4) Hamming code. *IEEE Journal of Solid-State Circuits*, 39(1):122–131, Jan. 2004.

[97] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel. Stochastic iterative decoders. In *IEEE International Symposium on Information Theory*, pages 1116–1120, Sept. 2005.

[98] Xilinx Corporation. *Virtex-4 User Guide*. www.xilinx.com.

[99] Xilinx Corporation. *Virtex-5 User Guide*. www.xilinx.com.

[100] M. R. Yazdani, S. Hemati, and A. Banihashemi. Improving belief propagation on graphs with cycles. *IEEE Communications Letter*, 8(1):57–59, Jan. 2004.

[101] R. Yazdani and M. Ardakani. Optimum linear LLR calculation for iterative decoding on fading channels. In *IEEE International Symposium on Information Theory*, pages 61–65, June 2007.

[102] E. Yeo, B. Nikolic, and V. Anantharam. Iterative decoder architectures. *IEEE Communications Magazine*, 41(8):132–140, Aug. 2003.

[103] D. M. Young and R. T. Gregory. *A Survey of Numerical Mathematics Reading*. MA: Addison-Wesley, 1973.

[104] R. Zarubica, S. G. Wilson, and E. Hall. Multi-Gbps FPGA-based low density parity check (LDPC) decoder design. In *IEEE Global Telecomm. Conference*, Washington D.C., USA, Nov. 2007.

[105] J. Zhang and M. P. C. Fossorier. A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letter*, 8:165–167, March 2004.

[106] T. Zhang and K. Parhi. Joint (3,k)-regular LDPC code and decoder/encoder design. *IEEE Transactions on Signal Processing*, 52(4):1065–1079, Aug. 2004.

[107] Z. Zhang. *Design of LDPC Decoders for Improved Low Bit Error Rate Performance*. Ph.D. Thesis, University of California at Berkeley, July 2009.

[108] Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic. A 47 Gb/s LDPC decoder with improved low error rate performance,. In *Symposium on VLSI Circuits*, June 2009.

[109] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic. An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors. *IEEE Journal of Solid-State Circuits.*, 45(4):843–855, April 2010.

[110] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright. Lowering LDPC error floors by postprocessing. In *IEEE Global Communications Conference*, pages 1–6, November 2008.

[111] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright. Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices. *IEEE Transactions on Communications*, 57(11):3258–3268, Nov. 2009.

[112] B. Zhu. Adaptive stochastic Reed Solomon decoding. Master of Engineering Project Report, McGill University, Montreal, Canada, April 2007.

KEY TO ABBREVIATIONS

Acronym	Significance
ABP	Adaptive Belief Propagation
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit-Error-Rate
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
CMOS	Complementary Metal-Oxide-Semiconductor
DC	Decoding Cycle (i.e., stochastic decoding iteration)
dB	Decibel
DRE	Distributed Randomization Engine
EM	Edge Memory
FER	Frame-Error-Rate
FF	Flip-Flop
FP	Floating-Point
FPGA	Field-Programmable Gate Array
FX	Fixed-Point
GF	Galois Field
IM	Internal Memory

LDPC	Low-Density Parity-Check
LFSR	Linear Feedback Shift Register
LLR	Log-Likelihood Ratio
LUT	Look-Up Table
MAP	Maximum A Posteriori
ML	Maximum-Likelihood
MSA	Min-Sum Algorithm
MTFM	Majority-based Tracking Forecast Memory
PN	Parity-check Node
RS	Reed-Solomon
SISO	Soft-Input Soft-Output
SM-T's	Sign-Magnitude to Two's Complement
SNR	Signal-to-Noise Ratio
SPA	Sum-Product Algorithm
TAB	Turbo-oriented Adaptive Belief propagation
TA-MSA	Turbo-oriented Adaptive MSA
TA-Offset MSA	Turbo-oriented Adaptive Offset MSA
T's-SM	Two's Complement to Sign-Magnitude
TFM	Tracking Forecast Memory
VLSI	Very-Large-Scale Integration
VN	Variable Node