University of Alberta

Stochastic Iterative Decoding

by

Anthony Charles Rapley ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2006

Your file Votre référence
ISBN: 978-0-494-22353-6
Our file Notre référence
ISBN: 978-0-494-22353-6

Abstract

This thesis studies a novel approach to iterative decoding of low density parity check codes: stochastic iterative decoders.

Physical implementations of iterative decoders often face a compromise between competing design parameters of decoded error performance, throughput rate, device size, and power consumption. It is shown how the belief propagation decoding algorithm, capable of yielding near-optimal error performance, can be implemented using stochastic computational elements, low complexity digital probability processing circuits. This combination has the potential for achieving a decoder design that yields high error correcting performance while being compact and power efficient.

A primer on error control coding, low density parity check codes, belief propagation decoding, and stochastic computing is first provided. The architecture and operation of an initial stochastic decoder design is then presented. Based on an analysis of deficiencies in this architecture, a number of performance enhancing measures are considered. Employing these enhancements, the stochastic iterative decoder is shown to perform within 0.125dB of the ideal belief propagation algorithm at a bit error rate of 1.2×10^{-5} for a (7,4) Hamming code.

Acknowledgements

I owe many thanks to the people who formed the richly supportive environment in which I have been so privileged to work and live. I wish to acknowledge:

- My supervisor, Dr. Vincent Gaudet, for his guidance, support, and, indeed, his patience in overseeing my work. It has been a rare and fortunate thing to have someone with his professional and personal qualities as a thesis supervisor.
- Anne, my wife and best friend, for her love and encouragement and for allowing me to vent on a fairly regular basis.
- Mom and Dad, for their unfailing love and support and for always letting me know that they're proud of me.
- The Natural Sciences and Engineering Research Council (NSERC), the Alberta Informatics Circle of Research Excellence (iCORE), Micronet R&D, and the University of Alberta for their generous financial support.
- Dr. Christian Schlegel, for his leadership of the world-class iCORE High Capacity Digital Communications (HCDC) laboratory.
- Dr. Chris Winstead, for his friendship, for increasing my knowledge of communications theory beyond the classroom, and for many interesting discussions about engineering things and non-engineering things alike.

- Dr. Ivan Fair, for his assistance during a challenging first semester and for teaching an outstanding course on information theory.
- Aaron Hughes, for his friendship and for being an essential study partner during courses.

I apologize for any omissions. None were intended.

Dedicated to my sweetie Anne, and to the

memory of my grandfather, Charles Maynard Rapley, who passed away

during the writing.

Contents

List of Tables

List of Figures

List of Symbols

Acronyms	Definition
A/D	analog to digital [conversion]
ADC	analog-to-digital converter
ANN	artificial neural network
ASIC	application-specific integrated circuit
AWGN	additive white Gaussian noise
BER	bit error rate
CMOS	complementary metal oxide semiconductor
D/A	digital to analog [conversion]
DAC	digital-to-analog convertor
DSP	digital signal processing / digital signal processor
Eb/N0	information bit signal to noise ratio
ECC	error control coding
FPGA	field-programmable gate array
HDL	hardware description language
IC	integrated circuit
I/O	input/output
LDPC	low density parity check (code)
LFSR	linear feedback shift register
LLR	log-likelihood ratio
PRBS	pseudo-random bit sequence
VLSI	very large scale integration

Chapter 1

Introduction

An error control coding implementation is a critical component of any modern communication system, particularly as the prevalence and ubiquity of digital communications increases. Reliable communication at efficient transmission power is crucial, particularly for mobile data systems.

The discovery of Turbo codes and subsequent rediscovery of low density parity check (LDPC) codes has been revolutionary, making it possible to achieve transmission rates asymptotically close to the Shannon capacity limit. While the theoretical performance of these codes is truly remarkable, several challenges present themselves in making the leap to a physical implementation.

The application of powerful error control codes such as Turbo codes and LDPC codes has become increasingly important due to their incorporation into a number of digital communications standards such as: W-CDMA 3G [2], the 3^{rd} Generation Partnership Project (3GPP) for IMT-2000 [3], Consultative Committee for Space Applications (CCSDS) telemetry channel coding [4], UMTS [5], DVB S-2 [6], DVB-RCS [7], and IEEE 802.x standards [8].

The good properties of powerful codes can be negated by poor decoder implementations. In order to maximize the benefits of error control coding, good implementations are essential.

The design challenge in an error control scheme is primarily in the decoder. An ideal decoder is one that features low complexity, compact size, acceptable power consumption, high throughput, and near optimal error performance in terms of the code being implemented. These goals are in obvious conflict with one another. The prominent engineering compromise is between complexity and error performance.

Most high performance decoders rely on complex algorithms employing 5-8 bit fixed-point computations. The cost of the performance is often seen in terms of larger device size, increased power consumption, and reduced throughput. Size and power consumption are particularly critical commodities in mobile applications.

Within the last few years, pioneering research has been performed on the construction of fully analog decoders, where the internal decoder metrics are represented as continuous analog voltages or currents [9] [10] [11]. Such decoders harness subthreshold conduction in transistors to perform analog computations with extremely low supply voltages. Improvements in speed or power of one to two orders of magnitude have been conjectured for analog decoders compared to their digital counterparts. Furthermore, analog circuitry is well suited to the relatively low precision requirements of iterative decoding algorithms. Potential drawbacks of analog decoders include technology-dependence, power consumption that is largely static, and testability issues.

It is for this reason that there is a high degree of desirability in designing a low-complexity iterative decoder in digital hardware. Digital hardware offers technology-independence through multiple potential platforms ranging from programmable logic such as FPGAs to standard cell-based ASICs to full-custom designs. Power scaling is also possible in digital hardware through clock speed adjustment and potentially through supply voltage scaling. Ultimately, there is a strong motivation to build a decoder that provides the benefits of analog and digital decoders while mitigating their deficiencies. The premise of this thesis is that the construction of the stochastic iterative decoder through the implementation of the belief propagation algorithm in stochastic computational hardware will meet this need.

This thesis is organized as follows: Chapter 2 provides background on the field of error control coding and its applications. The group of codes known as low-density parity-check (LDPC) codes is then discussed, with focus on the code properties, factor graph representation, and the belief propagation decoding algorithm. A summary of recent decoder implementations, both in digital and analog circuitry, is also presented. Chapter 3 describes stochastic computation, including its history, numerical value representation, and computational elements that represent the building blocks of a stochastic computer. A literature review of implementations based on stochastic computation follows, and the chapter then concludes by reviewing properties of stochastic computation that makes it attractive for implementing iterative decoders. Chapter 4 presents a novel implementation of an iterative LDPC decoder using stochastic computation. The architecture and properties of the prototype are first described, and promising early simulation results are provided. Performance problems for larger codes are then investigated and modifications to the decoder to ameliorate these problems are assessed. The chapter closes with a review of two practical methods of generating the stochastic sequences for the decoder. Lastly, Chapter 5 concludes this thesis by identifying its contributions and suggesting potential avenues of future research.

Chapter 2

Error Control Coding

2.1 Introduction

We begin this thesis by discussing the field of error control coding, with specific reference to a class of codes known as low density parity check codes.

This chapter is organized as follows: In Section 2.2 we motivate the discussion of error control coding by introducing Shannon capacity theory and its application to the transmission of digital information. This leads into a discussion of the field of error control coding in section 2.3. Section 2.4 covers several aspects of LDPC codes, including properties of linear block codes (general code family to which LDPC codes belong), properties of LDPC codes, factor graph representation, and the belief propagation decoding algorithm. Several recent digital and analog iterative decoding implementations are presented in Section 2.5. Section 2.6 concludes the chapter.

2.2 Communication and Channel Capacity

Communication is fundamentally a problem of sending a message from a source through a channel and reconstructing it at the destination. Conceptually, a

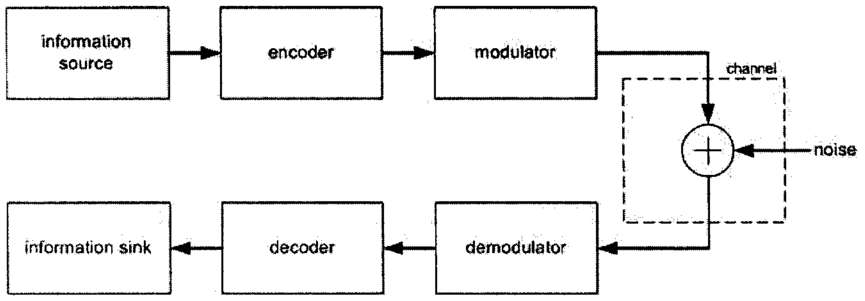


Figure 2.1: Block diagram of basic communications system.

channel is any medium of information transfer [12]. Common examples of channels include wired links such as copper wire carrying electrical impulses and fiber optic cabling conveying photons of light, and wireless links with air or free-space conveying electromagnetic (EM) impulses or photons. The recording channel is a prevalent but less obvious communication channel. The recording channel could be magnetic media (e.g. hard disk drives) or optical media (e.g. compact discs), where information is transferred in time rather than in space. Even water can constitute a channel in the case of sonar-based communication.

The information transmitted over a channel may be represented in a variety of formats, but for the purposes of this thesis it will be assumed that binary digits (bits) are the transmitted quantity.

Figure 2.1 depicts a block diagram of a communication system. Data transmission starts with an information source, such as a computer or a voice encoder, which produces information to be transmitted. The encoder then adds redundant bits to the information bits to assist in reconstituting the data at the receiver. The modulator then converts the data (information plus redundant bits) into a signaling format appropriate for the channel. As the data passes through the channel it is corrupted by noise. Noise can take many forms including thermal noise (e.g. heat from power supplies), man-made noise (e.g.

nearby electrical devices), and environmental noise (e.g. lightning). Channel noise is commonly modeled as additive white Gaussian noise (AWGN). On the receiver side, a demodulator converts the data back from its channel format to a baseband format (bits). The demodulated data is then processed by the decoder, which attempts to detect and correct transmission errors due to the channel noise and outputs a reconstituted version of the information that was originally transmitted. Finally, the information passes to the information sink, where the information can be further processed depending on its content.

Noise corruption and the characteristics of the channel place restrictions on how to transmit information, and on the rate at which information can be transferred.

Information was first quantified mathematically in 1948 by Claude Shannon [13]. Shannon developed a logarithmic measure of the information content of a source and determined the theoretical maximum rate at which information could be transmitted over a channel and be reconstituted at the destination without error. He termed this limit *capacity.* The capacity of a channel is dependent on two parameters: the bandwidth of the channel and the signal-to-noise ratio (SNR), and is expressed as:

$$C = B \cdot \log_2 (1 + S/N)\,. \tag{2.1}$$

A common form of Shannon's famous capacity expression is presented in Equation 2.1. B is the channel bandwidth in Hertz, and S/N is the signal-to-noise ratio expressed as a straight power ratio. Capacity, C, is the theoretical maximum errorless transmission rate for the channel in bits/second. If the rate of information transfer is less than the capacity of the channel, then it is theoretically possible to achieve reliable (i.e. error-free) communication with appropriate encoding of the transmitted data. If the rate of information transfer exceeds capacity, then error-free communication is impossible.

It was from Shannon's work that arose the concept of an encoder and decoder in the communication system. In order to achieve channel utilization that is closer to capacity, it is necessary to intelligently encode transmitted data through the addition of redundancy such that the receiver can compensate for errors due to noise. Shannon proved the existence of capacity-achieving codes in [13].

2.3 Error Control Coding

Error control coding is the branch of information theory concerned with the reliable transmission of information through a channel.

The fundamental principle of error control coding is to add redundancy to a transmitted quantity of information such that the original information can be reconstructed without error, or with fewer errors, at the receiver despite corruption that may occur in the channel due to noise and other factors. Redundancy in a digital communication system takes the form of additional bits transmitted with the information bits. A code is a mapping of information (source) words into longer code words. The source bits are termed *information* bits and the redundant bits are termed *parity* bits.

The *rate* of a code is the ratio of source word bits to code word bits for a given code. A code that maps k-bit source words to n-bit code words has a code rate of k/n. Codes must also specify an inverse mapping such that codewords can be decoded to recover the original source words. To be useful, this process must also facilitate some means of error detection and/or correction.

Coding gain, realized through the decoding process at the receiver, is the decrease in SNR required to achieve a given bit error rate when ECC is em-

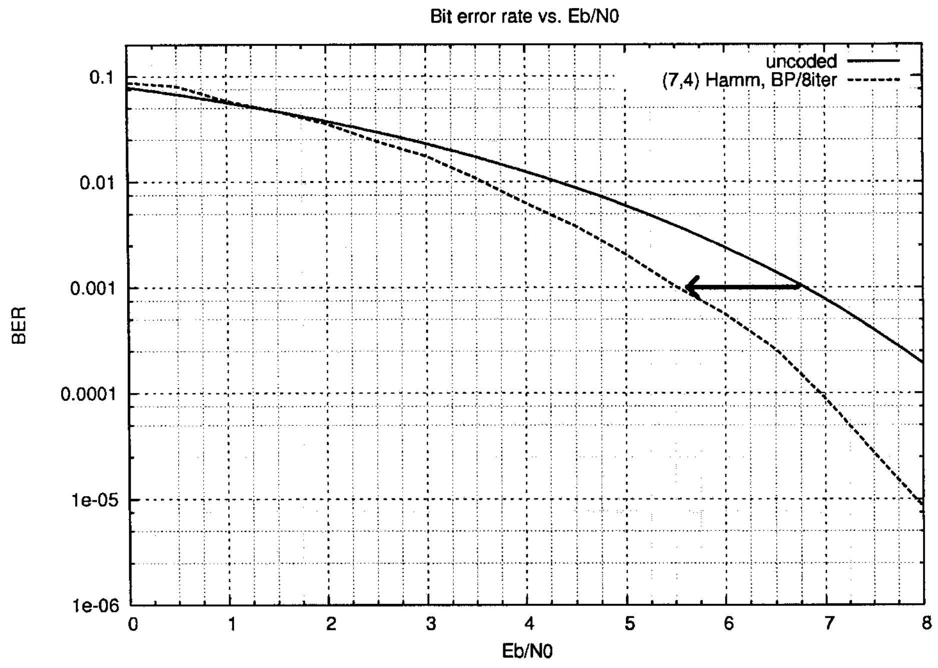


Figure 2.2: Illustration of coding gain

ployed versus uncoded transmission. Figure 2.2 provides an illustration of this concept. The BER performance for a (7,4) Hamming code is compared to the BER for uncoded antipodal transmission (i.e. a binary 1 maps to +1 and a binary 0 maps to -1). At a BER of 1×10^{-3} the Hamming code exhibits a coding gain of approximately 1.25dB compared to uncoded transmission.

We can therefore see that employing an error control code can yield one of two benefits over uncoded transmission:

1. If the same transmission power is used, the bit error rate will be lower.

2. Lower transmission power can be used to achieve the same bit error rate.

There are two major code families: linear block codes and convolutional codes. Low density parity check codes, the main code type targeted for implementation in the decoder presented in this thesis, belong to the family of linear block codes. Convolutional codes are an equally important code family but are generally beyond the scope of this thesis.

2.4 Low Density Parity Check Codes

2.4.1 Overview

Low density parity check codes are a class of linear block codes first described in 1962 by Gallager in [14]. LDPC codes are sometimes referred to as Gallager codes after the name of their inventor. These codes were largely forgotten until the discovery of Turbo codes [15] and iterative decoding techniques. A subsequent rediscovery of LDPC codes [16] [17] occurred when it was determined that LDPC codes have capacity-approaching performance with iterative decoding. There has since been a vigorously renewed interest in researching the design, properties, and practical implementation of these codes.

2.4.2 Linear Block Codes

Linear block codes are an algebraically rich approach to coding and were the first family of codes to be developed.

A linear block code is uniquely defined by an $(n-k) \times n$ parity check matrix, **H**. A binary 0 or 1 is placed in each cell of the matrix. Each binary 1 represents the inclusion of a variable in a parity check constraint.

The number of rows and columns in the parity check matrix provides immediate information about the code: The total number of bits per codeword is n, the number of information bits is k, and the number of parity bits is $n-k$. The parity check matrix is related to the codeword set, C, such that

$$Hx^T = 0, \tag{2.2}$$

where $\mathbf{x}$ is a legal n-bit codeword.

The rate of a code with an $(n-k) \times k$ parity check matrix can be determined as:

$$R = \frac{k}{n}. \tag{2.3}$$

The *rank* of a code is the number of linearly independent rows in the parity check matrix of the code. Any row in the parity check matrix that is a direct copy of any other row or formed by the modulo-2 addition of two or more other rows is said to be linearly dependent. Linearly dependent rows in the parity check matrix do nothing to enhance the error-correcting capabilities of the code.

The *distance* between two codewords is defined as the number of bit positions that differ. For example, the distance between the 7-bit codewords 0100110 and 1110100 is 3. The minimum distance for a code is the smallest distance between any two codewords in the codeword set. Generally speaking, error correcting performance improves with increasing minimum distance in a codeword set.

The modulo-2 sum of any two or more codewords of a linear code where each $x \in C$, is itself a codeword belonging to the set.

2.4.3 Properties of LDPC codes

An LDPC code is simply a linear block code with a sparse parity check matrix. Some LDPC codes have been discovered which approach Shannon Capacity within 0.1dB for a code length of 10^6 and within 1dB for a code length of 10^4 [18].

A *regular* LDPC code is one which has the same number of 1s in each and every row of the parity check matrix, and the same number of 1s in each and every column. The number of 1s in each row need not match the number of 1s in each column for the code to be designated as regular. Conversely, any LDPC whose parity check matrix does not conform to the parameters of a regular code is designated as *irregular.* Given predetermined row and column weightings in the parity check matrix, research indicates that the placement of the 1s in the parity check matrix is best chosen at random [18] [19].

With iterative decoding, regular LDPC codes tend to underperform Turbo codes of similar size. Irregular LDPC codes, however, can outperform with belief propagation decoding Turbo codes of comparable size. [18].

2.4.4 Factor Graph Representation

Decoding algorithms must deal with complex functions involving many variables, a problem space that grows with the code size. It is therefore beneficial to factorize the complex global problem of decoding a large codeword into a set of simpler, localized functions. Such a factorization can be accomplished through the mapping of any linear block code, particularly an LDPC code, into its *factor graph* representation [20]. The factor graph is the essential view of LDPC codes in the context of high-performance forward error correction.

A factor graph is a bipartite graph consisting of two types of nodes: variable nodes and parity check nodes. Either type of node produces outputs based on its inputs according to a specific functional constraint. The size and structure of the factor graph is based directly on the parity check matrix.

The mapping of a parity check matrix to factor graph representation is straightforward, as shown by the parity check matrix for a (7,4) Hamming code in Figure 2.3 and its corresponding factor graph in Figure 2.4. A $(n-k)\times n$ parity check matrix maps to a graph consisting of $n-k$ parity check nodes and

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

Figure 2.3: Parity check matrix for (7,4) Hamming code

n variable nodes. The three rows of the parity check matrix for the sample (7,4) Hamming code translate to three parity check nodes in the factor graph, and the seven columns translate to seven equality nodes. Any row-column position, $h_{i,j}$, of the parity check matrix where a '1' is present signifies an edge connecting the i^{th} parity check node and the j^{th} variable node.

The *degree* of a node is the number of edges connected to the node.

Every edge in the factor graph represents a pair of unidirectional connections pointing in opposite directions relative to each other. An edge connecting two nodes therefore indicates bidirectional communication between those two nodes.

The straightforward mapping of a parity check matrix to a factor graph is also advantageous for automatically generating a factor graph in a software simulator or hardware description language using a parity check matrix as input.

2.4.5 Belief Propagation Decoding

The description of factor graph code representations motivates the discussion of the belief propagation decoding algorithm, used for decoding LDPC codes. Belief propagation is an iterative decoding algorithm, whereby messages are passed back and forth between the variable nodes and parity check nodes giving an iterative refinement of the decoding result. Iterative decoding was one of the key innovations of Turbo codes that pushed channel utilization closer than ever to capacity.

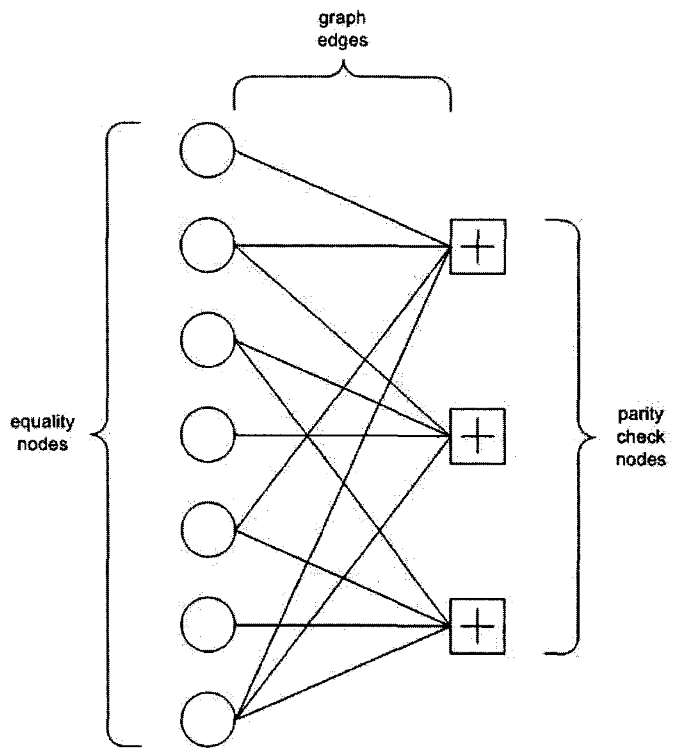


Figure 2.4: Factor graph representation for (7,4) Hamming code

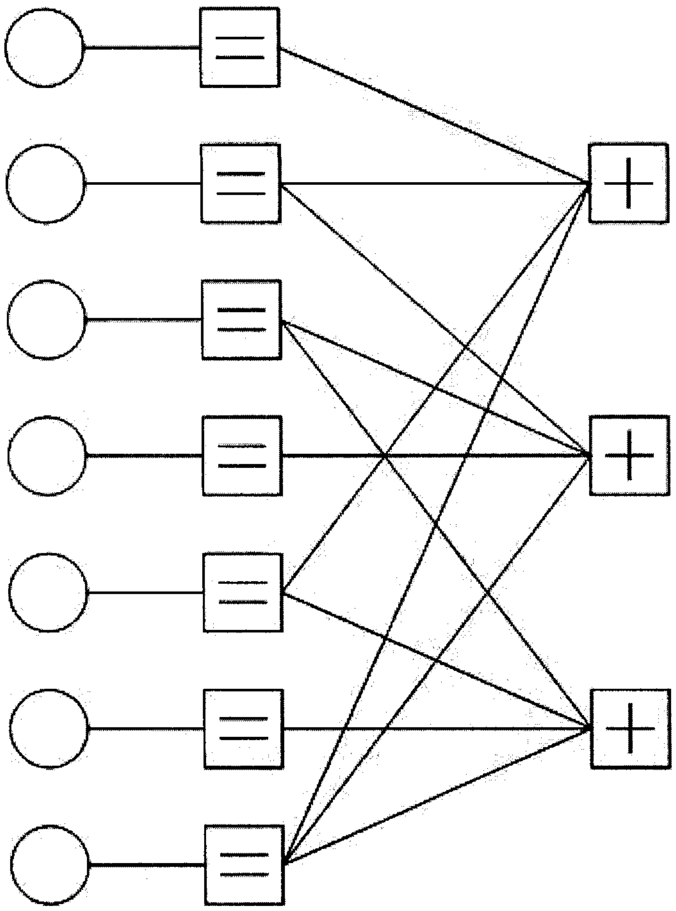


Figure 2.5: Normal factor graph for (7,4) Hamming code

The decoding algorithm utilizes a modified version of the factor graph called a *normal graph.* The normal graph has the requirement that each node has a clearly defined functional constraint and the variable nodes only have one connected edge. To realize a normal graph, *equality nodes* are inserted between the variable nodes and the parity check nodes as depicted in Figure 2.5.

Nodes therefore produce outputs based on functional constraints applied to the inputs. Functional constraints are defined in terms of three-edge nodes. Nodes with a degree greater than three, such as the bottommost equality node and all of the parity check nodes in Figure 2.5, can be created by internally cascading three-edge nodes.

Parity Check Constraint

The generalized functional constraint for a parity check node having inputs A and B and output C is

$$C = A \otimes B, \tag{2.4}$$

where A, B, and C may assume values of 0 or 1. To adapt this constraint for probabilistic computation, let S_1 be the set of all pairs of inputs (A, B) such that (A, B, C) is a valid set for $C = 1$ according to the parity check node constraint. Note that all probabilities represent the probability of a given input or output being a '1'. Then,

$$\Pr(C) = \alpha \sum_{(A,B \in S_1)} \Pr(A)\Pr(B), \tag{2.5}$$

$$\Pr(C) = \alpha\left[\Pr(A)(1-\Pr(B)) + (1-\Pr(A))\Pr(B)\right]. \tag{2.6}$$

Choose α such that

$$\sum_{c} \Pr(C = c \,|\, A, B) = 1. \tag{2.7}$$

This yields

$$\alpha = (1-\Pr(A))(1-\Pr(B)) + (1-\Pr(A))\Pr(B)) + \Pr(A)(1-\Pr(B)) + \Pr(A)\Pr(B). \tag{2.8}$$

Simplifying, $\alpha = 1$ and therefore for every two inputs (A,B) to a parity check node, an output message equal to

$$\Pr(C) = \Pr(A)(1 - \Pr(B)) + (1 - \Pr(A))\Pr(B) \tag{2.9}$$

is produced for the output C.

Equality Constraint

The generalized functional constraint for an equality node having inputs A and B and output C is

$$C = A = B, \tag{2.10}$$

where A, B, and C may take on values of 0 or 1. To adapt this constraint for probabilistic computation, let S_2 be the set of all pairs of inputs (A, B) such that (A, B, C) is a valid set for $C = 1$ according to the equality node constraint. As with the derivation of the parity check node equation, all probabilities represent the probability of a given input or output being a $'1'$. Then,

$$\Pr(C) = \alpha \sum_{(A,B \in S_2)} \Pr(A)\Pr(B). \tag{2.11}$$

Choose α such that

$$\sum_{c} \Pr(C = c\,|A, B) = 1, \tag{2.12}$$

$$\alpha = \frac{1}{\Pr(A)\Pr(B) + (1 - \Pr(A))(1 - \Pr(B))}. \tag{2.13}$$

Therefore, for every two inputs (A, B) to an equality node, an output message equal to

$$\Pr(C) = \frac{\Pr(A)\Pr(B)}{\Pr(A)\Pr(B) + (1 - \Pr(A))(1 - \Pr(B))} \tag{2.14}$$

is produced for the output C.

Algorithm

A message passing decoding algorithm such as belief propagation must observe the *extrinsic information principle*, that a node output produced on a given graph edge cannot depend on a message previously received on that edge. Violating the extrinsic information principle will produce pronounced correlations in the node output messages to the detriment of the accuracy of the calculations.

A *cycle* is a path over connected nodes that begins and terminates at the same node without traversing any edge more than once. Short cycles in a graph violate the extrinsic information principle to the detriment of decoding performance. For this reason cycles of length shorter than a desired parameter are disallowed when designing an LDPC code parity check matrix, from which the structure of the factor graph flows. A graph with no cycles is called a *tree.*

The belief propagation decoding algorithm proceeds on a factor graph as follows:

1. Each variable node is loaded with $r_i = \Pr(x_i = 1|y_i)$, where x_i denotes the i^{th} transmitted symbol and y_i is the physical estimate made at the output of the channel. All other messages in the graph are reset to 0.5.

2. Each variable node forwards its value to the equality node to which it is connected.

3. Messages are passed between the equality nodes and the parity check nodes according to their functional constraints given by Equation 2.14

and Equation 2.9, respectively, for a fixed number of iterations or until some other stopping criterion is satisfied.

4. The received message m_i at each variable node is sampled. If $r_i \cdot m_i > 0.5$, output a 1 for the i^{th} hard bit decision. Otherwise, output a 0.

2.5 VLSI Iterative Decoder Implementations

2.5.1 Digital Decoders

Table 2.1 summarizes recent implementations of digital decoders for LDPC and Turbo codes. Progress has been made in increasing the throughput of the decoders while striving for power efficiency. LDPC decoders, owing largely to their parallel structure, have seen throughput rates push into the multi-gigabit per second range. Nonetheless, the number of published digital decoder implementations appears to be low considering the number of important communications standards in which Turbo and LDPC codes are incorporated. Research into higher throughput, higher performance, and lower power consumption decoder implementations is only expected to expand.

2.5.2 Analog Decoders

Analog decoding [9] [10] [11] is a novel approach to the design of iterative decoders using analog voltages or currents to represent the internal metrics or values that are passed between functional elements. These decoders are an interesting mating of probability theory with transistor physics. Analog decoders can provide improvements in speed, power efficiency, or even both over their digital counterparts, but they should be viewed as a complementary technology to digital decoders rather than an orthogonal development.

Table 2.1: Published digital decoder implementations

Developer	Code Len.	Throughput	Power @ Supply	Energy/Bit	Process	Dimensions	Type
Blanksby, Howland [21]	1024	512Mb/s	690mW @ 1.5V	1.3nJ	0.16μm CMOS, 5m layers	7.5 × 7.5 mm^2	digital LDPC
Mansour, Shanbhag [22]	2048	800Mb/s	760mW @ 1.8V	0.95nJ	0.18μm CMOS	3.1 × 4.2 mm^2	digital LDPC
Zhang, Parhi [23]	9216	54Mb/s	—	—	Xilinx Virtex-E XCV2600E FPGA	—	digital LDPC
Mansour, Shanbhag [24]	2048	320Mb/s	787mW @ 1.8V	2.46nJ	0.18μm CMOS, 6m layers	14.3 mm^2 (chip)	digital LDPC
Lin, Lin, et al. [25]	1200	3.33Gb/s	644mW @ 1.8V	0.19nJ	0.18μm CMOS	25 mm^2 (chip)	digital LDPC
Darabiha, Carusone, et al. [26]	2048	1.6Gb/s	—	—	0.18μm CMOS	4.2 × 4.2 mm^2	digital LDPC
Bougard, Giulietti, et al. [27]	432	75.6Mb/s	658mW @ 1.8V	8.7nJ	0.18μm CMOS	7.16 mm^2 (core), 14.7 mm^2 (chip)	digital Turbo
Bickerstaff, Garrett, et al. [28]	5114 (max)	2Mb/s	292mW @ 1.8V	146nJ	0.18μm CMOS, 6m layers	9 mm^2 (core)	digital Turbo
Lee, Shanbhag, et al. [29]	—	27.6Mb/s	330mW @ 1.8V	12nJ	0.18μm CMOS, 6m layers	3.467 × 2.513 mm^2 (chip)	digital MAP
Bekooij, Dielissen, et al. [30]	—	2Mb/s	35mW @ 1.8V (estimate)	17.5nJ	Altera FLEX 10K130E FPGA	—	digital Turbo
Berrou, Combelles, et al. [31]	—	40Mb/s	1.6W/iter @ 5V	160nJ (4 iters)	0.8μm CMOS, 2m layers	8.9 × 8.8 mm^2	digital Turbo

The speed improvement potential of analog decoders stems primarily from two design characteristics: continuous iterations and the removal of the need for an ADC.

The current or voltage metrics are allowed to circulate freely in an analog decoder and settling to stable result values over time. Value variations decrease on each "iteration" and therefore iteration times progressively decrease, leading to higher speeds. The continuous iterations also eliminate the area and timing requirements of a clock tree.

Analog decoders can accept analog demodulated channel metrics, and therefore do not require ADC at their inputs. The removal of the ADC provides a speed increase as well as power and area savings.

Analog decoders have been shown to provide convergence properties comparable to their digital counterparts, and provide robust operation despite transistor nonidealities and mismatch.

Iterations in an analog decoder are continuous, where the voltage or current-based values are allowed to settle over time.

Analog decoders have been implemented for both Turbo and LDPC codes. Table 2.2 summarizes recent analog decoder implementations.

2.6 Summary

This chapter has provided an introduction to communication systems and to the field of error control coding. Low density parity check codes have been discussed in more detail, including their properties as linear block codes, their factor graph representation, and iterative decoding through belief propagation. Numerous recent digital and analog decoder implementations were then summarized.

Digital and analog decoder implementations continue to improve in performance, but they both have particular drawbacks. Digital decoders may consume more power and area than their analog counterparts, whereas analog decoders may be hindered by technology-dependence, power consumption that is largely static, and intractable testability. There remains a motivation to look beyond these technologies to new approaches that combine the strengths of digital and analog decoders while mitigating their weaknesses. Stochastic decoding may represent such an approach.

We now proceed to a discussion of stochastic computation.

Table 2.2: Published analog decoder implementations

Developer	Code Len.	Throughput	Power @ Supply	Energy/Bit	Process	Dimensions	Type
Gaudet, Gulak [32]	48	13.3Mb/s	185mW @ 3.3V	13.9nJ	0.35μm CMOS, 3m layers	1.1312 × 1.2579mm^2	analog Turbo
Moerz, Gabara, et al. [33]	16	160Mb/s	20mW @ 3.3V	0.125nJ	0.25μm BiCMOS	1.680mm^2	analog tail-biting MAP
Winstead, Dai, et al. [34]	8	2Mb/s	16μW (at 20kbps), 1mW (at 1Mb/s)	0.8nJ (at 20kbps), 1nJ (at 1Mb/s)	0.5μm CMOS	1.5 × 1.5 mm^2	analog tail-biting MAP
Winstead, Gaudet, *et* al. [35]	256	4.84Mb/s	861μW	0.178nJ	TSMC 0.18 μm CMOS	2.3 × 2.4 mm^2	analog Turbo product
Vogrig, Gerosa, et al. [36]	120	2Mb/s	10.3mW @ 3.3V,7.6mW @ 2V	5.15nJ (at 3.3V), 3.8nJ (at 2V)	0.35μm CMOS, 3m layers	3.7 × 1.1 mm^2 (core), 4.5 × 2.0 mm^2 (chip)	analog Turbo
Nguyen, Winstead, et al. [37]	8	444kb/s	283μW @ 0.8V (simulated)	0.64nJ/bit	0.18μm CMOS	0.158 × 0.276 mm^2 (core)	analog Hamming
Lustenberger, Helfenstein, et al. [38]	18	100Mb/s	50mW @ 5V	0.5nJ	0.8μm BiCMOS, 2m layers	2.8 × 2.6 mm^2 (chip), 1.7 × 0.7 mm^2 (core)	analog tail-biting MAP
Lustenberger [39]	44	150Mb/s (simulated)	1.0W @ 5V (simulated)	6.7nJ	0.8μm 2M2P BiCMOS	5.28 × 5.45 mm^2 (chip), 2.7 × 2.5 mm^2 (core)	analog tail-biting LDPC
Winstead, Dai, et al. [40]	8	20Mb/s	3.3mW	0.165nJ	0.5μm CMOS	—	analog tail-biting MAP
Shakiba, Johns, et al. [41]	—	200Mb/s	30mW @ 3.3V	0.15nJ	0.8μm BiCMOS	0.5 mm^2 (core)	class-IV analog Viterbi
Hemati, Banihashemi, et al. [42]	32	80Mb/s	5mW @ 1.8V	0.06nJ	0.18μm CMOS	0.630 × 0.910 mm^2 (core)	analog LDPC

Chapter 3

Stochastic Computation

3.1 Introduction

The concept of stochastic computation is not new, having first been considered in the late 1960s [43] as an alternative to conventional digital computers. At the time, engineers had recognized the utility of performing low-level human mental processes using digital computers. Such low-level processes include arithmetic operations and storage or, in the colloquial, "number-crunching". Already, pioneering minds were considering performing the high-level functions of the human mind using computers as well. Pattern-recognition, learning, and decision-making are the prominent high-level functions.

Artificial neural networks (ANNs) are the hardware implementations of parallel computing structures that mimic the functioning of the human brain. To achieve powerful computational ability in an ANN, parallelism must be created on a massive scale. In the late 1960s, digital hardware was substantial in area, power inefficient, and somewhat unreliable [44]. An enabling technology was required to ameliorate these issues and make practical the creation of ANNs.

Stochastic computation was discovered and recognized as an attractive new computer architecture for ANN implementation because of the simple nature of the processing elements. Stochastic computation is innately suited for the computation of probability values. That is, computation where the operands are all real numbers in the range $[0, 1]$.

Today, modern IC processes make it possible to produce, at a relatively low cost, conventional integrated circuits that are compact and reliable with millions of transistors on a single die. Stochastic computation is nonetheless still relevant, finding application in ANNs where the ease of achieving the required massive parallelism offsets the decreased precision of the computational elements.

This chapter is organized as follows: In Section 3.2, the value representation employed by stochastic computers is introduced and properties of stochastic sequences are described. Simulation results for the accuracy of stochastic sequences in isolation relative to the sequence length are also presented. Some examples of stochastic computational elements are then presented in Section 3.3. Section 3.4 reviews applications of stochastic computation in recent literature. Section 3.5 concludes the chapter by summarizing the properties of stochastic computers and asserting their viability as a modern technology. This concluding section establishes the premise of this thesis: that high performance decoders can be built using stochastic computational elements.

3.2 Stochastic Value Representation

The defining characteristic of stochastic computation is the unique value representation that it employs. In a stochastic computer, values are encoded as a Bernoulli sequence of bits. A Bernoulli sequence is a sequence of independent

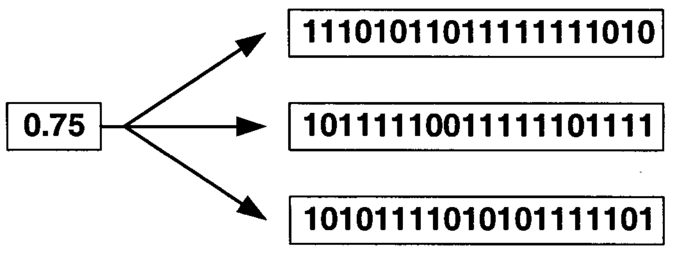


Figure 3.1: Possible mappings of a probability value to stochastic sequences

random binary variables $\alpha_1, \alpha_2, \alpha_3, ...$ where the probability that any element in the sequence is 1 is the same for each and every element. For an unsigned number N in the range $[0, 1]$, the probability that any bit d_i in a Bernoulli sequence is a binary '1' is given by

$$\Pr(d_i = 1) = \frac{N}{N_{\max}}. \tag{3.1}$$

For probability values, $N_{max} = 1$. Thus, the probability of any given bit being a binary '1' in a sequence of stochastic bits is equal to the real number probability value being represented.

Encoding of stochastic sequences is probabilistic. That is, there is no fixed mapping between a real number probability value and its representation as a sequence of stochastic bits. Many different stochastic representations are possible for the same value. Figure (3.1) shows three possible mappings of a probability value to a stochastic sequence.

The precision of stochastic sequences can be controlled in the time domain. Precision can be improved by increasing the length of a stochastic sequence used to represent a probability value. As the length of the stochastic sequence approaches infinity, the difference between the stochastic value and the real number value it represents approaches zero. That is, if we let e represent the absolute error between a probability value and its stochastic representation,

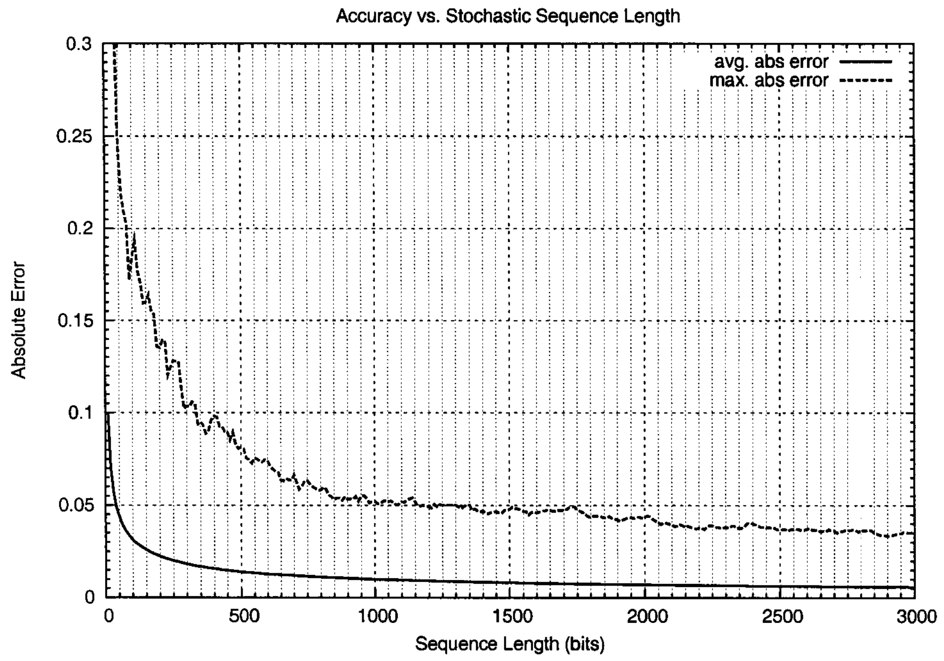


Figure 3.2: Average and Absolute Error in Stochastic Sequence Accuracy

and let N represent the length of the stochastic sequence in bits, then:

$$\lim_{N \to \infty} e = 0. \qquad (3.2)$$

Figure 3.2 depicts the simulated accuracy of a stochastic sequence versus its length. The plotted curves were obtained as follows: 10,000 tests were run. Each test involved generating a rational number p between 0 and 1, then generating stochastic bits from the rational number. The stochastic sequence was converted back to a rational number at regular length intervals and compared to the original number. The average absolute error is an indication of the average +/- deviation of a stochastic sequence at a given sequence length compared to the rational number the sequence represents. The maximum absolute error is the maximum +/- deviation observed across the set of tests.

Figure 3.3 depicts the simulated accuracy, relative to the probability being

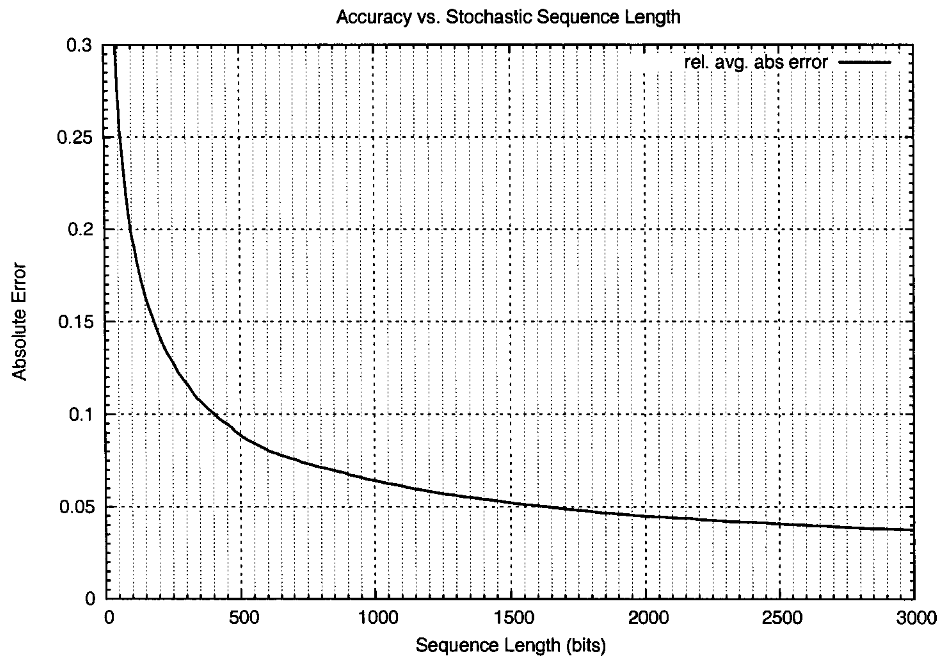


Figure 3.3: Error in Stochastic Sequence Accuracy Relative to Represented Probability

represented, of a stochastic sequence versus its length. The plotted curve was obtained through a procedure similar to the procedure described above for obtaining the average absolute error curve in Figure 3.2. 10,000 tests were run. Again, for each test a rational number p between 0 and 1 was generated, then stochastic bits were generated from this number. At each interval where the stochastic sequence was converted back to a rational number the absolute error was computed. The absolute error was then divided by $(1-p)$ when $p \geq 0.5$, and was divided by p when $p < 0.5$. The resulting scaled absolute error value was then averaged across the set of tests.

3.3 Stochastic Computational Elements

Table 3.1 presents some common stochastic computational elements. It is remarkable that complex functions such as multiplication and division can be implemented with a single gate. This simplicity stands in contrast to the relatively large number of gates required to implement multiplication, division, or even addition functions in conventional digital circuitry.

Please refer to [43] and [44] for an expanded listing of stochastic computational elements.

3.4 Applications

Currently, stochastic computation is primarily targeted for pulsed neural network implementations. Papers on stochastic neural networks abound in the literature. However, papers on specific implementations using Gaines' stochastic computational elements are sparsely distributed:

1. A stochastic neural network with in situ learning capabilities is described in [45]. A hardware implementation was produced in 1.2μm CMOS, with approximately 200 synapses on the chip.

2. A stochastic computational implementation of a optical character recognition (OCR) system for the E-13B MICR (Magnetic Ink Character Recognition) font is proposed and investigated in [46]. The system was simulated in a bit-true C++ program.

3. It is shown how stochastic computing can be used to implement complex analog controllers in [47]. A stochastic computation-based nonlinear dissipative controller for a series resonant converter was implemented in

a Xilinx 4000XL FPGA and its output compared very closely to the theoretical expected output.

4. The application of stochastic computing to the signal processing of information from parallel microsensor arrays and VLSI imagers is investigated in [48]. The computations described were not implemented in hardware, but the authors suggest ready portability to an FPGA. One specific potential application is the creation of a coprocessor for handling the digital signal processing of visual sensor data on mobile robots.

5. The authors of [49] present approaches for designing stochastic neurons that can be used to construct very large scale neural networks.

6. An implementation of a multilayer neural network in a 1.5μm process is described in [50].

7. A lookup table-based artificial network architecture using stochastic computation and suitable for implementation in FPGAs is discussed in [51].

8. A flexible architecture for realizing neural networks with arbitrary topology and dimensions using stochastic computation is presented in [52]. Two general-purpose neural network ICs were fabricated as a proof of concept.

Other papers provide analysis and insight into the properties of stochastic computation without describing implementations, thus serving as technology-enablers:

1. The effects of refractory pulse counting processes in stochastic computers are studied in [53].

2. A space-efficient technique for the parallel summation of weighted input sequences in a stochastic computer is described in [54].

3. Generation of weighted random sequences suitable for input to a stochastic computer is studied in [55] [56] [57] [58] [59]. Stochastic sequence generation is addressed in this thesis in Section 4.4.

3.5 Summary

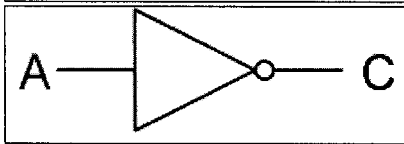
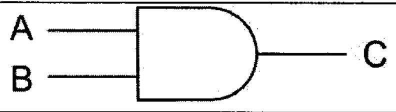
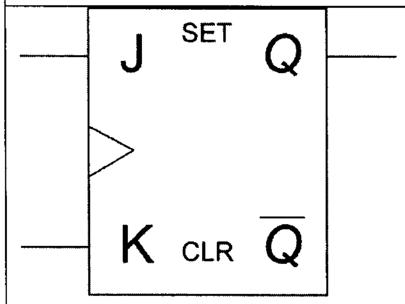
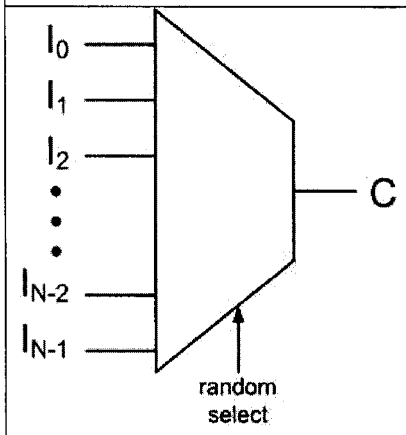
In this chapter we have introduced stochastic computation, including its developmental background, value representation, and computational elements. The utility of employing stochastic computation is clear where the operands are probability values. Stochastic computation offers five significant advantages over conventional digital hardware [44]:

1. Small die area in implementation
2. Robustness in low precision operation
3. Bit-serial single-wire communication between elements
4. Short critical paths through simple elements, enabling extremely high clock rates
5. Control of precision in the time domain, allowing accuracy to be traded for computational time with the same hardware

Stochastic computation is a technology currently targeted primarily to neural network implementations. A factor graph based decoder may be viewed as an instance of a neural network. Together, the functional nodes in the factor graph form a massively-parallel (at least for larger codes) network much like the array of neurons in a neural network. Also like neurons, the factor graph nodes maintain internal learning states that are updated through inter-nodal communication.

A stochastic computational network implementation of an iterative decoder would seem ideal for resolving the apparently competing design goals of a compact, power-efficient decoder that also yields high throughput and error performance. Thus, the premise of this thesis is established.

Table 3.1: Common stochastic computational elements

Circuit	Type	Function
	Probability Inversion	$C = 1 - A$
	Multiplication	$C = A \times B$
	Division	$Q = \frac{J}{J+K}$
 I_0 I_1 I_2 I_{N-2} I_{N-1} C random select	Weighted Summation	$C = \sum S_i I_i$

Chapter 4

Implementation Approaches

4.1 Introduction

We now examine the application of stochastic computing to iterative decoding. Stochastic decoding was shown in Chapter 3 to have several advantages over conventional digital hardware which could make possible efficient, high performance decoders. We consider the design requirements of a stochastic decoder and investigate its performance.

This chapter is organized as follows: Section 4.2 details the architecture and operation of an early stochastic iterative decoder design. The architecture is first overviewed. Following this overview is a discussion of the construction and performance of stochastic parity and equality circuits, and the construction of larger functional nodes using these circuits. The issue of converting output stochastic sequences to decoded codewords is considered. The operating algorithm of the decoder is then described, and first performance results for the decoder are supplied.

Section 4.3 deals with performance issues encountered with the decoder for larger codes. A baseline decoder architecture is established for comparing

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

Figure 4.1: Parity check matrix for irregular (16,8) LDPC code used in prototype stochastic decoder

performance improvements resulting from modifications. The issue of lockup in the equality nodes is discussed, then several potential improvements are proposed and examined: broadcast initialization, parity check node output randomization, log-likelihood ratio scaling, factor graph layering, and supernodes.

Section 4.4 covers two approaches to the implementation of stochastic sequence generators for VLSI implementations. Section 4.5 concludes the chapter.

4.2 Initial Decoder

This section describes the construction, operation, and performance characteristics of the early stochastic iterative decoder presented in [1].

4.2.1 Architectural Overview

A simple irregular (16,8) LDPC code was used as the basis for the initial stochastic decoder. The parity check matrix for this code is supplied in Figure 4.1.

Figure 4.2 shows two equivalent factor graph representations of the code. Note that the maximum degree of the equality nodes is 3 (including the channel

I/O) and that the parity check nodes uniformly have a degree of 3.

Only a few types of functional units are required to implement the factor graph as a stochastic decoder in hardware: equality nodes, parity check nodes, stochastic sequence generators, and threshold converters.

Both equality nodes and parity check nodes are comprised of an interconnection of equality circuits or parity circuits, respectively, as described in Section 4.2.6. The stochastic parity check circuit is presented in Section 4.2.2 with a subsequent analysis of its computational accuracy in Section 4.2.3. Likewise, the stochastic equality circuit is presented in Section 4.2.4 along with a computational accuracy analysis in Section 4.2.5.

Stochastic sequence generators convert probability values of noisy channel estimates into stochastic sequences for input to the equality nodes. Because the primary focus of this thesis is to explore the use of stochastic computational elements for performing iterative decoding calculations, the discussion of stochastic sequence generation is deferred to Section 4.4.

Threshold converters are employed to convert the output stochastic sequence from each equality node into a single decoded bit. The use of threshold converters for this purpose is elaborated in Section 4.2.7.

Factor graph edges simply translate to wires in hardware. Communications are bit-serial and bidirectional, meaning that each graph edge actually represents a pair of wires conveying digital sequences in opposite directions. The bidirectionality of the links is normally abstracted out in typical representations of factor graph-based decoders.

4.2.2 Stochastic Parity Circuit

By inspection, it can be seen that the logical function represented by the parity check constraint of Equation 2.9 is exclusive-OR (XOR). Consequently,

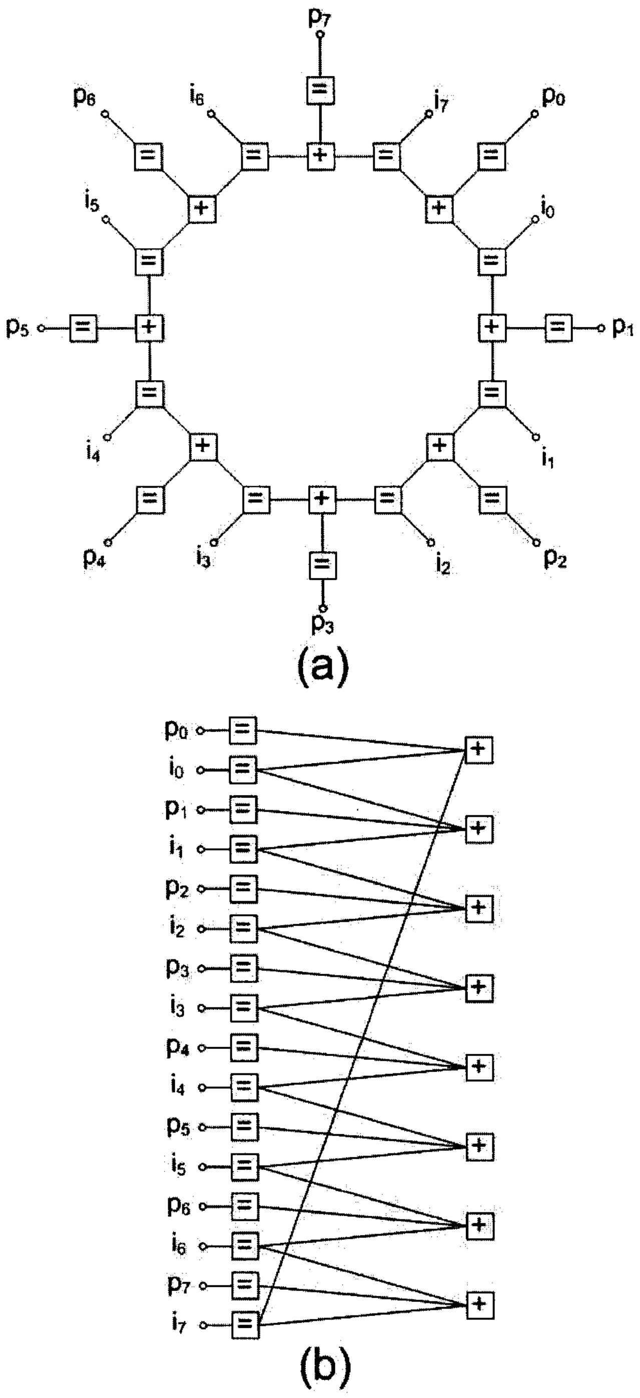


Figure 4.2: Factor graph representation of irregular (16,8) LDPC code used for stochastic decoder prototype. i_n indicate information bits and p_n indicate parity bits. (a) Ring structure of code. (b) Familiar two-column representation of the same code.

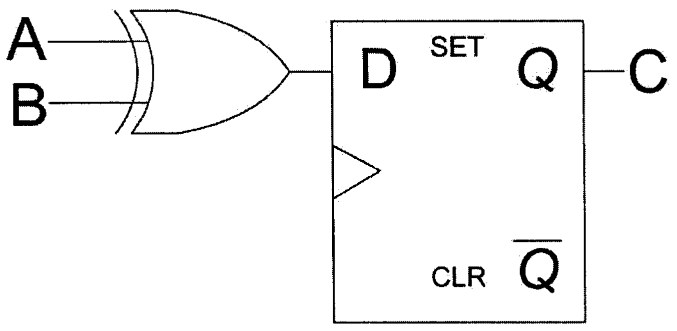


Figure 4.3: Stochastic Parity Circuit

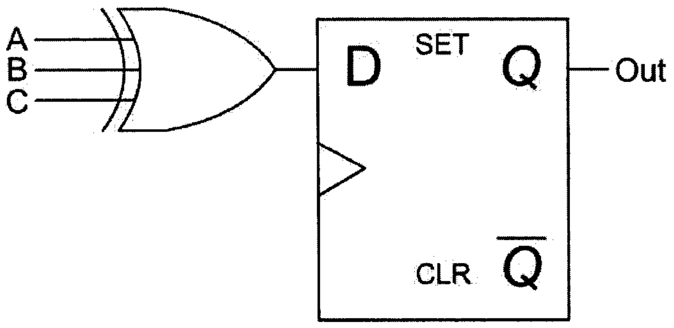


Figure 4.4: Three input stochastic parity check circuit using higher fan-in XOR

the parity check constraint can be implemented with a simple XOR gate. Figure 4.3 depicts a stochastic parity circuit. The D flip-flop in this figure has been added to support synchronous decoder operation. It does not affect the computational logic of the circuit. The initialization scheme described in Section 4.3.3 proposes the complete removal of the D flip-flop from the parity circuit.

The number of inputs to the parity check circuit can be increased by either increasing the fan-in of the XOR gate or by cascading a series of XOR gates. Figure 4.4 depicts a three-input parity check circuit constructed using the former approach. The choice of approach depends on the engineering constraints of the implementation.

4.2.3 Feed-forward Accuracy of Stochastic Parity Circuit

A bit-true model of the stochastic parity circuit was written in C++ to determine through simulation the computational accuracy of the circuit. This model was designed to have run-time configurability of its degree, such that it can be used to simulate any size of parity circuit.

The results presented in Figure 4.5 were obtained by generating 10^4 sets of random floating-point input value sets for the circuit. For each set of input values, the expected floating-point output was first computed according to Equation 2.9. The floating-point input values were then used to generate input sequences of 2000 stochastic bits per input. These sequences of stochastic bits were passed through the circuit model and the sequence of output bits was summed.

After the passing of the first 50 stochastic bits, and every 50 stochastic bits thereafter, the output count was converted to a floating-point value by dividing it by the total number of stochastic bits passed. The absolute difference between this floating-point value and the expected value was computed, and the mean of the absolute differences across all 10^4 input sets formed one datum point on the plot.

The output bits from the circuit were not fed back to the circuit inputs in any way, hence the designation of this test as a feed-forward accuracy test.

These results provide some insights into the accuracy of the stochastic parity circuit: First, as expected, the accuracy of the computations improves with increasing numbers of stochastic bits passed per computation. Second, the nearly overlapping curves indicate that there is only a minute reduction in computational accuracy for circuits of increasing degree. Third, it appears that the accuracy of the circuit is fundamentally limited by the accuracy of

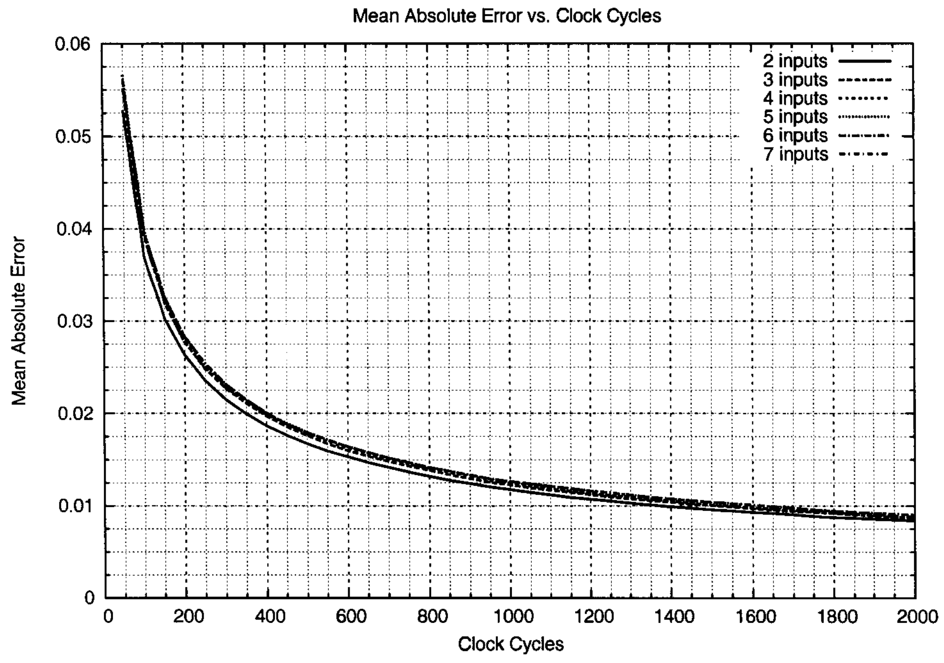


Figure 4.5: Accuracy of parity check circuit in feed-forward configuration

the individual stochastic sequences passing through it. Referring back to the accuracy plot of an individual stochastic sequence in Figure 3.2, it is shown that an individual stochastic sequence will have an absolute error of roughly 0.0100 at 1000 clocks, and 0.0070 at 2000 clocks. This is closely tracked by the parity circuit, which has an absolute error of about 0.0125 at 1000 clocks and 0.0090 at 2000 clocks.

The second and third points above confirm that the stochastic parity circuit implements a very close approximation of the parity constraint given by Equation 2.9. The plot presented in Figure 4.5 may be used as a guide for a system integrator to determine the number of clock cycles required to achieve the desired level of accuracy.

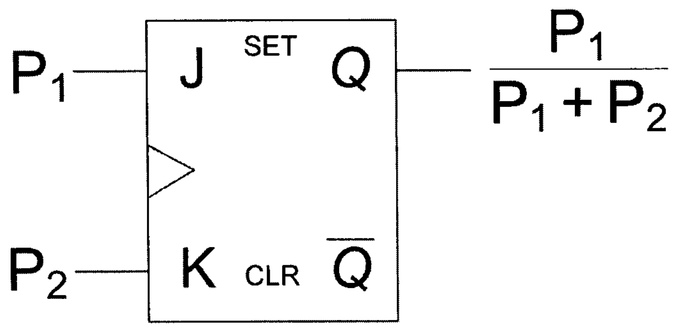


Figure 4.6: Stochastic division using a JK flip-flop.

4.2.4 Stochastic Equality Circuit

The stochastic circuit implementation of the equality constraint given by Equation 2.14 is not as straightforward as the implementation of the stochastic parity circuit from the parity check constraint. Stochastic division is particularly difficult to implement.

Fortunately, Gaines shows in [43] that a JK flip-flop can be used to implement a stochastic divider whose generalized output maps conveniently to Equation 2.14. Depicted in Figure 4.6, a JK flip-flop with an input value, P_1, on its J input and an input value, P_2, on its K input produces the output result

$$\frac{P_1}{P_1 + P_2}. \tag{4.1}$$

By substituting $\Pr(A)\Pr(B)$ for P_1 and $(1 - \Pr(A))(1 - \Pr(B))$ for P_2 in Equation 4.1, Equation 4.1 is transformed into Equation 2.14.

A JK flip-flop can therefore be used to implement the division in Equation 2.14. AND gates are used to implement the multiplication in the numerator and denominator. Inverters are used to implement the probability inversion. The resulting stochastic equality circuit is shown in Figure 4.7.

The number of inputs to the equality circuit can be increased by either

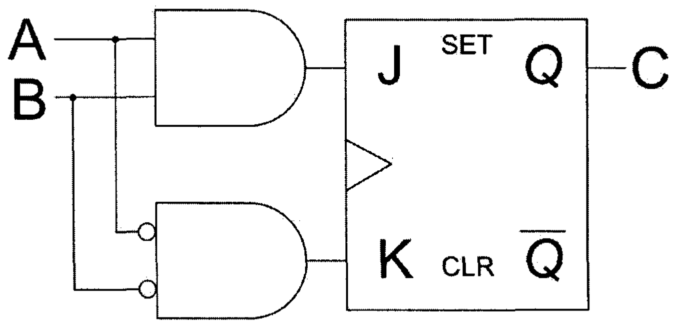


Figure 4.7: Stochastic Equality Circuit

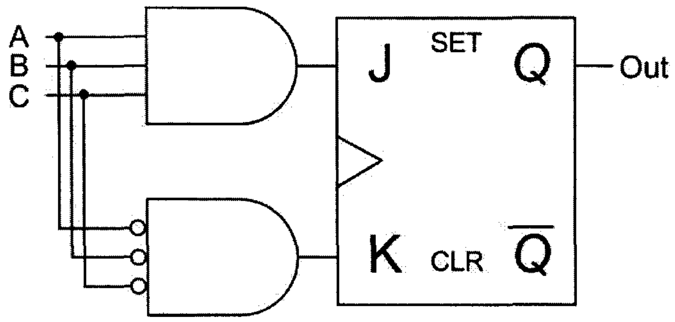


Figure 4.8: Three input stochastic equality circuit using higher fan-in AND gates

increasing the fan-in of the AND gates or by cascading a series of AND gates at the J and K inputs. Figure 4.8 depicts a three-input equality circuit constructed using the former approach. The choice of approach depends on the engineering constraints of the implementation.

4.2.5 Feed-forward Accuracy of Stochastic Equality Circuit

As with the stochastic parity circuit, a bit-true model of the stochastic equality circuit with run-time degree configurability was written in C++ to determine through simulation the computational accuracy of the circuit.

The results presented in 4.9 were obtained through a test identical to the

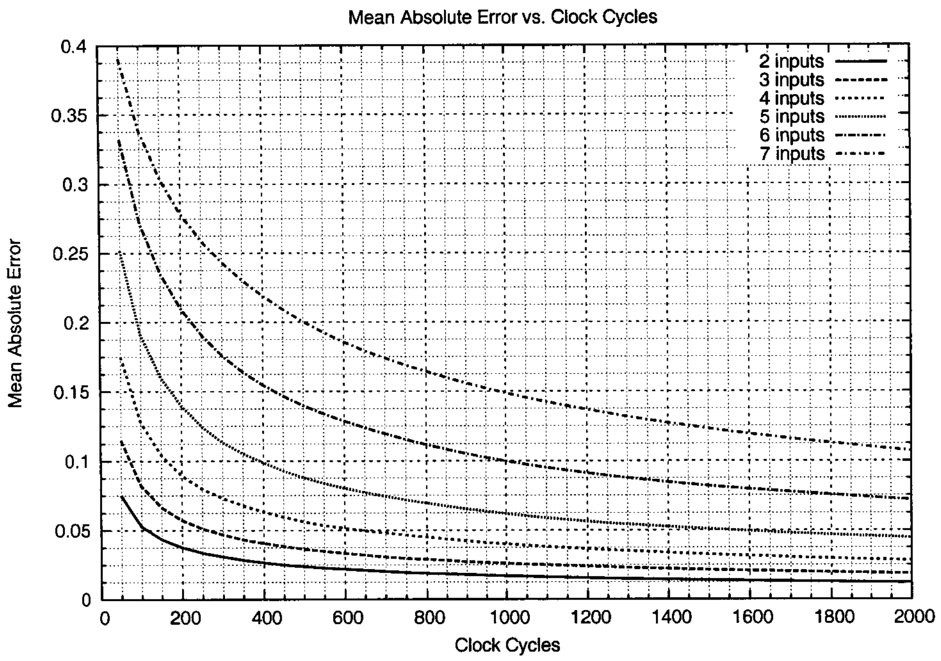


Figure 4.9: Accuracy of equality circuit in feed-forward configuration

one described in Section 4.2.3, except that the expected values were calculated according to the equality constraint given by Equation 2.14.

These results provide some insights into the accuracy of the stochastic equality circuit: First, as expected, the accuracy of the computations improves with increasing numbers of clock cycles. Second, the curves show a distinct reduction in accuracy for increasing numbers of inputs to the circuit at all numbers of clock cycles. This indicates that the stochastic approximation of the equality constraint does introduce an error term, and this error compounds as the number of inputs to the circuit increases.

Given a sufficient number of clocks, the stochastic equality circuit should be capable of providing enough accuracy to make it amenable to integration in an iterative decoder. The plot presented in Figure 4.9 may be used as a guide for a system integrator to determine the number of clock cycles required to achieve the desired level of accuracy.

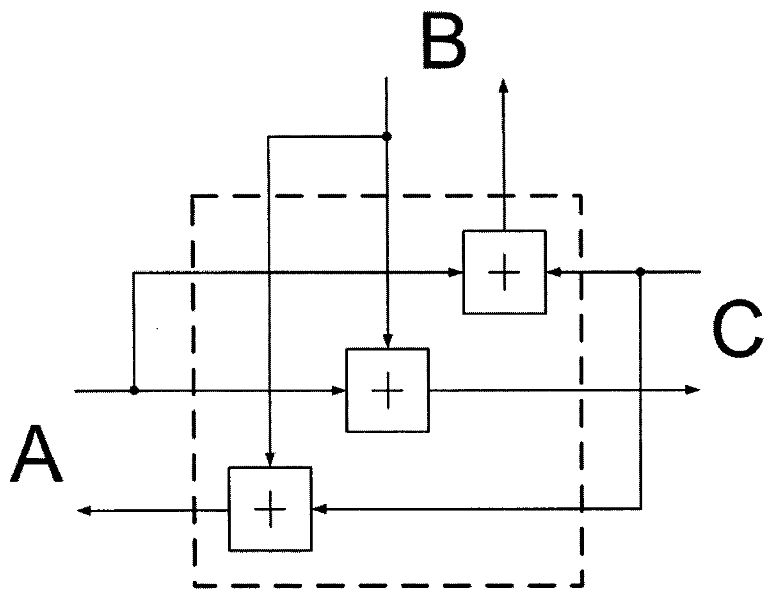


Figure 4.10: Three-edge parity check node (dashed line) with three, three-input constituent parity check circuits. Node edges are labeled A, B, C.

4.2.6 Node Construction

An output must be produced on each edge of a node based on inputs on the other edges. Functional nodes in the factor graph are therefore constructed by an interconnection of constituent circuits. Parity check nodes are comprised of instances of the parity check circuit presented in Section 4.2.2. Likewise, equality nodes are comprised of instances of the equality circuit presented in Section 4.2.4.

The construction of parity check nodes is straightforward. The output produced for any given edge is based on the inputs on all of the other edges. An n-edge parity check node will consist of n constituent parity circuits, each having $n-1$ inputs. Figure 4.10 depicts a three-edge parity check node with three constituent parity check circuits.

Equality nodes are constructed similarly to parity check nodes, with a slight variation due to the channel edge. The output produced for any given edge

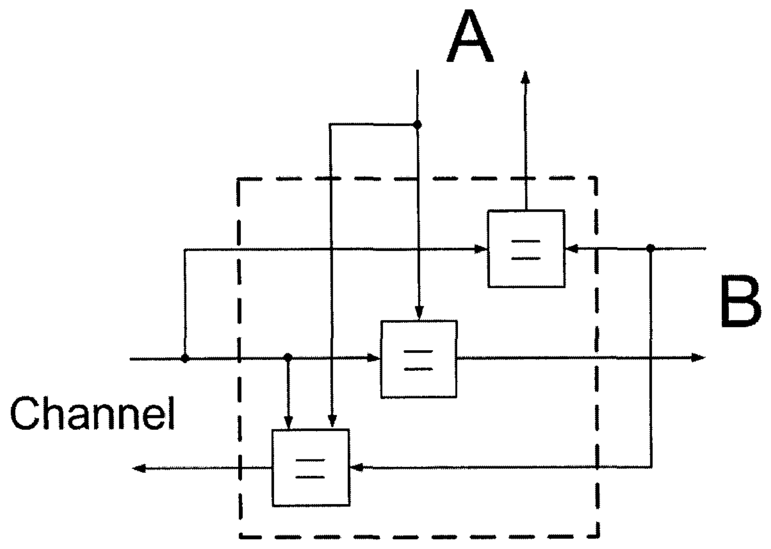


Figure 4.11: Three-edge equality node (dashed line) with three constituent eqaulity circuits. Channel edge is labelled "channel" and factor graph edges are labeled A and B.

connected to the factor graph is based on the inputs on all of the other edges. The output on the channel edge, however, is based on the inputs on all node edges including the channel edge input. Therefore, an n-edge equality node will consist of n constituent equality circuits, with $n-1$ having $n-1$ inputs and one having n inputs. Figure 4.11 depicts a three-edge equality node with three constituent equality circuits.

It is evident that the stochastic parity check and equality nodes are deterministically scalable to any node degree. This fact may make it possible to automatically generate hardware description language (HDL) code for a stochastic decoder based only on a parity check matrix.

4.2.7 Threshold Conversion

To obtain a result from the decoder, it is not necessary to convert the output stochastic sequences back to binary numeric representation. All that is re-

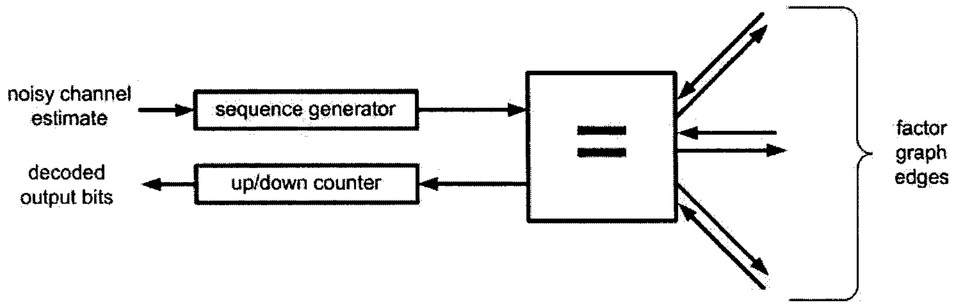


Figure 4.12: Equality node with up/down counter connected to the output and sequence generator connected to the input. Bidirectionality of connections is shown explicitly.

quired is to perform a threshold conversion for each codeword decoding cycle on the output sequence of each equality node, such that a single-bit result is produced. If 50% or more of the output bits are '1', then the threshold converter output bit should be a '1'. Otherwise, the output bit should be a '0'.

The task of threshold conversion can be performed through the use of an up/down counter with a sign bit. The up/down counter must be reset at the beginning of the decoding cycle for each new noisy received codeword. At the end of the decoding cycle for a given codeword the sign bit is taken as the decoding result.

Care must be taken not to confuse the meaning of the sign bit. In typical signed binary representation a sign bit of '0' indicates a positive number. In this application a positive number shall be threshold converted to a '1', and a negative number shall be converted to a '0'. It is therefore recommended that the sign bit be inverted inside each up/down counter, thereby avoiding confusion in implementing a stochastic decoder in the framework of a larger communications system.

Figure 4.12 depicts an up-down counter attached to the output wire of an equality node. The bidirectionality of the connections to the equality node are explicitly shown in this figure, whereas they are normally abstracted as a single wire. A stochastic sequence generator is connected to the input of the equality node. Methods for implementing stochastic sequence generators are discussed in Section 4.4.

4.2.8 Decoder Operation

The decoder operates as follows:

Initialization is performed for each new noisy received codeword by clearing all of the flip-flops in the factor graph and clearing the up-down counters attached to the equality nodes. The probabilities of each received codeword bit being a '1', the noisy channel estimates, are loaded into the stochastic sequence generators.

The decoder is clocked. On each clock edge each equality node receives a new bit from the attached stochastic sequence generator. A new output bit is produced by the nodes on every edge in the factor graph (in both directions of each edge).

The decoder is clocked for T_INIT clock cycles with the up-down counters held in reset. That is, for the first T_INIT clock cycles, the up-down counters do not count the output bits. This constitutes a training period for the decoder.

After T_INIT clock cycles, the decoder is clocked until all up-down counters reach an absolute count of T_CHECK. At this point, the decoder checks to ensure that the sign bits of the up-down counters indicate a valid codeword. If the codeword is valid, the sign bits are used to form the decoded codeword and a new noisy received codeword can be loaded into the decoder. Otherwise, the decoder is allowed to continue operating.

A codeword can be determined to be valid when, for each and every parity check node, the XOR sum of all bits coming into the parity check node is '0'.

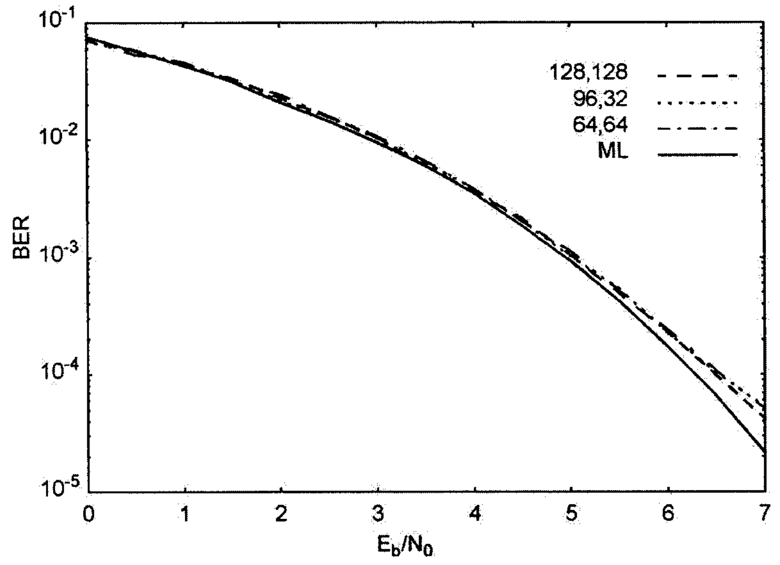


Figure 4.13: BER plot for initial decoder [1]

4.2.9 Performance Results

Figure 4.13 presents BER performance results for the LDPC code shown in Figure 4.2. The solid curve is the maximum likelihood (ML) decoding result for the code, which was possible to obtain due to the code's simplicity. The dashed lines represent different tests of the stochastic decoder with the number of cycles for the T_INIT and T_CHECK phases varied as indicated by the paired numbers in the graph legend. At a BER of 10^{-4} the stochastic decoder was only about 0.15dB from the ML curve.

4.3 Implementation Challenges and Solutions

While the initial results of the stochastic decoder were extremely promising, severe performance degradation was observed with slightly larger codes with higher node degrees. This thesis extends the original work by examining methods of improving performance. The following sections examine problems and potential solutions.

4.3.1 Baseline Decoder

It is useful to establish a baseline version of the stochastic decoder in order to compare the relative merit of different attempts at improving performance.

The baseline stochastic decoder is constructed identically to the initial decoder described in Section 4.2. The baseline decoder uses a simpler operating algorithm, differing from that of the initial decoder as follows:

- The node outputs are reset to random values.
- The up/down counters count every bit, eliminating the concept of T_INIT and T_CHECK phases.
- The decoder operates for a fixed duration for each codeword.

Figure 4.14 shows the BER plot for the baseline stochastic decoder operating on a (7,4) Hamming code. The stochastic BER curve is compared against the output of the ideal belief propagation algorithm operating on the same code for 8 iterations. It is apparent that the baseline decoder exhibits severe deficiencies. It is interesting to note that the BER performance of the stochastic decoder actually degrades above an SNR of approximately 5.5dB.

4.3.2 Equality Node Lockup

A serious problem affecting the performance of the baseline stochastic decoder for more complex codes is a phenomenon that can be termed "equality node lockup".

In high-level terms, equality node lockup occurs when the transitions in the channel input stochastic sequence cannot cause the outputs of any equality

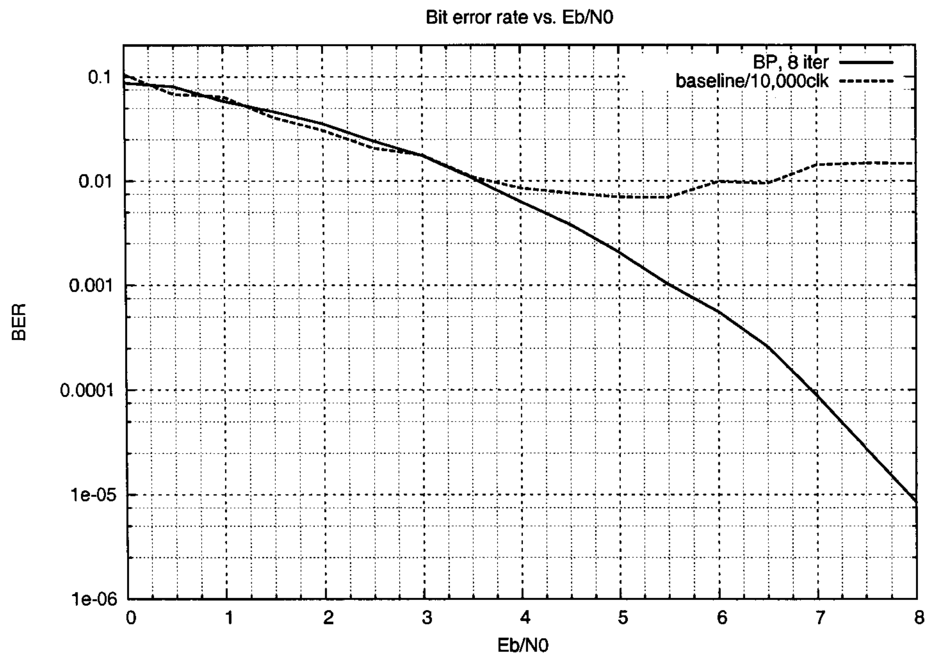


Figure 4.14: BER plot for (7,4) Hamming code with baseline stochastic decoder

node to change. The actual lockup condition is preceded by equality node outputs that are static for a few or several clock cycles across the entire set of equality nodes. When the decoder reaches a locked-up state, the output decision bits do not change and iterative decoding effectively ceases.

The decoder can potentially lock onto the correct codeword. The probability of this happening, though, decreases for larger codes. The more likely outcome is that the decoder will lock into some set of suboptimal local minima, which may not even comprise a valid codeword let alone the correct codeword. Recently there have been investigations into these suboptimal minima, "trapping sets", or "stopping sets" with an aim to mitigate their impact [60] [61] [62] [63] [64].

The key difference between the (16,8) LDPC code, for which the stochastic decoder worked well, and the (7,4) Hamming code, for which the stochastic decoder performed poorly, is that the Hamming code includes an equality node with degree 4. All of the equality nodes for the LDPC code have degree 3.

To understand why a higher degree in an equality node could lead to a problem, one needs to re-examine the behaviour of the equality circuit presented in Section 4.2.4. On each clock cycle, the equality circuit evaluates its inputs. If all of the inputs are '0', the output will be '0'. Likewise, if all of the inputs are '1', the output will be '1'. If the inputs are neither all '0' nor all '1', the equality circuit will hold its output value from the previous clock cycle.

As the degree of an equality node increases, so do the number of inputs to the constituent equality circuits. With an increased number of inputs, the probability that all of them will agree is decreased. Consequently, higher degree equality nodes are more likely to continue asserting a stored output bit.

For every input that is added to an equality circuit, the probabilities of the inputs agreeing and the output bit inverting both decrease by up to a factor of two. For example, assuming completely random inputs the probability that the inputs of a two-input equality circuit will agree is 0.5. The probability that inputs that are in agreement will cause a transition at the output is 0.25 because the inputs may be agreeing on the previously held output value. If the number of equality circuit inputs is increased to three, the probability of the inputs being in agreement drops to 0.25 and the probability of a transition at the output drops to 0.125.

It is shown in Section 4.2.5 that the stochastic equality circuit has good computational accuracy in a feed-forward configuration. That study, however, did not take into account the operational characteristics of the equality circuit in a feedback configuration.

Figure 4.15 shows the stochastic decoder for a (7,4) Hamming code in a state of partial lockup. All graph edges are conveying '0' values in both directions. Many other states of partial lockup are possible.

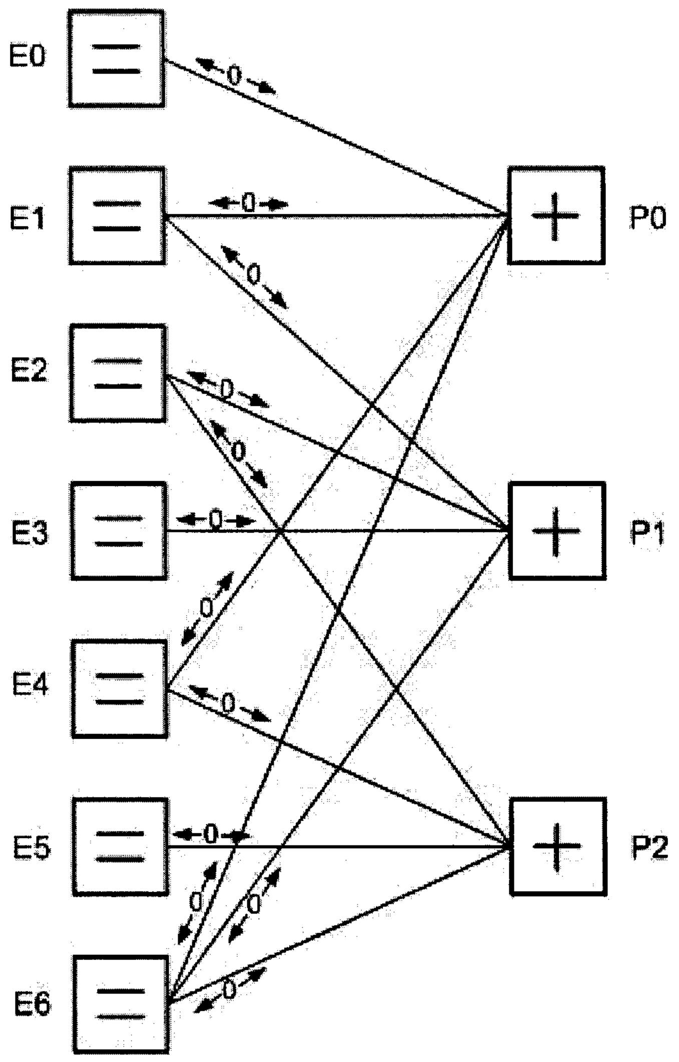


Figure 4.15: Partial lockup state in stochastic decoder for (7,4) Hamming code.

The term partial lockup is used in reference to Figure 4.15 because the two-edge equality nodes in the graph always have the potential to invert the outputs of the parity check nodes to which they are connected. It is nonetheless difficult for the decoder to have switching activity in a partial lockup state.

For example, the channel input bits on nodes E0 and E1 would both have to be '1' on the same clock cycle in order for node E1 to output a '1' to parity check node P1. To have E1 output a '1' to its attached up-down counter, the channel inputs on E0, E1, and E3 would all have to be '1' on the same clock cycle.

Because E6 has a degree of four, it is even more difficult for this node to output a '1' on any of its output edges. Nodes E0, E3, and E6 would all have to have a '1' on their channel inputs on the same clock cycle in order for E6 to output a '1' to P2. Even worse, to have E6 output a '1' to its attached up-down counter, the channel inputs on E0, E3, E5, and E6 would all have to have a '1' on the same clock cycle.

The fact that it is difficult, but not impossible, to maintain switching activity in the (7,4) Hamming code decoder provides an explanation for the poor to mediocre performance of the stochastic decoder for this code.

Figure 4.16 depicts a situation of complete lockup. For illustrative purposes, the same (7,4) Hamming code decoder is shown with an additional parity check that was specified by adding a linearly-dependent row to the parity check matrix. This addition was made to increase to three the minimum degree of any given equality node. In this situation even if all of the channel inputs to the decoder are '1', none of the equality node outputs will change.

The situation presented in Figure 4.16 also highlights an important problem: Initializing the outputs of all of the flip-flops in the graph to '0' before starting a new decoding cycle as described in Section 4.2.8 will initialize the

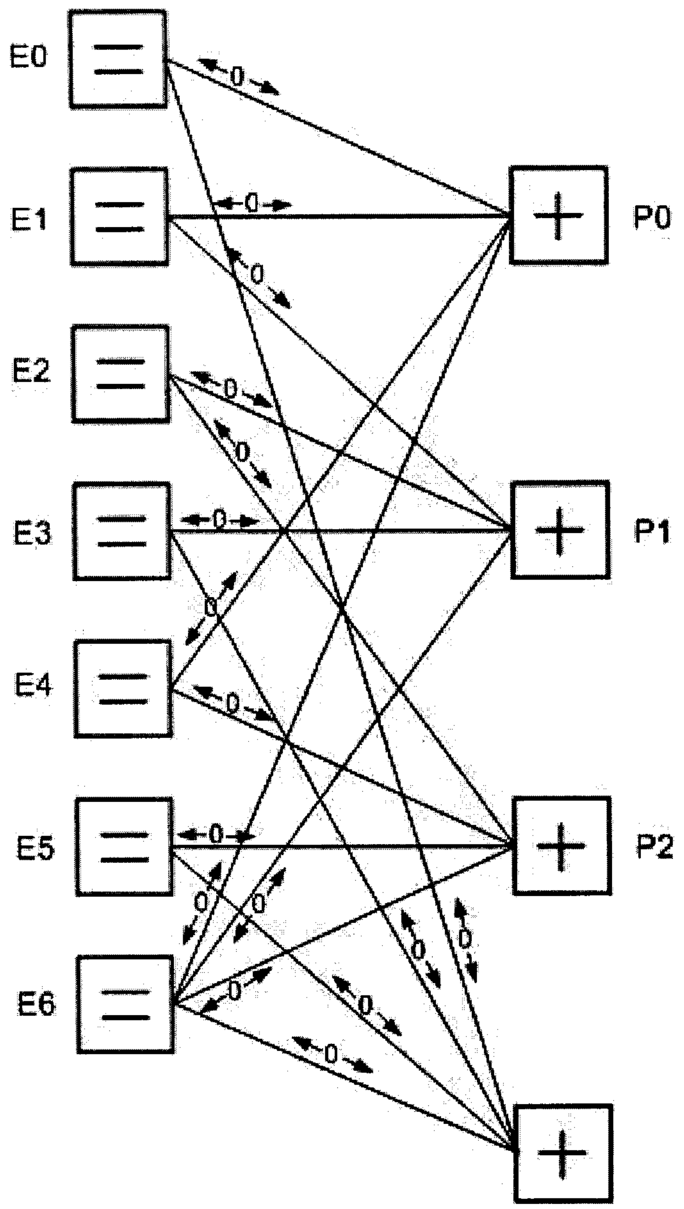


Figure 4.16: Complete lockup state in stochastic decoder for (7,4) Hamming code with additional parity check.

decoder to a locked-up state if the minimum degree of any equality node is three or greater. Using broadcast initialization as an improvement to the initialization process is discussed in Section 4.3.3.

4.3.3 Design Improvements

Obviously equality node lockup is detrimental to the performance of the stochastic decoder. Indeed, the reduction in uncertainty of received codeword bit probabilities and a corresponding reduction in toggling of stochastic sequence inputs may explain why the stochastic decoder actually performs worse at higher SNR values. This section describes attempts that were made to mitigate the lockup problem. The common thread in these modifications is that they are primarily aimed at increasing the switching activity in the graph in order to keep the equality node outputs toggling periodically.

Broadcast Initialization

Broadcast initialization is proposed as a means of intelligently priming the decoder at the beginning of each codeword decode process. As has been shown in Section 4.3.2, initializing all node outputs to 0 can actually cause an immediate lockup condition. Randomizing the initial node outputs potentially avoids this situation, but broadcasting a set of "best guess" bits on the first decoding iteration may be a better solution.

Broadcast initialization is implemented on top of the baseline decoder by specifying that, for each codeword bit, a stochastic bit will be clocked into the associated equality node and driven out on all of its inputs for the first iteration. The D flip-flop is also removed from the parity check circuits. This change is made because the initial values of the D flip-flops would otherwise propagate back to the equality nodes and negate the benefit of the broadcast

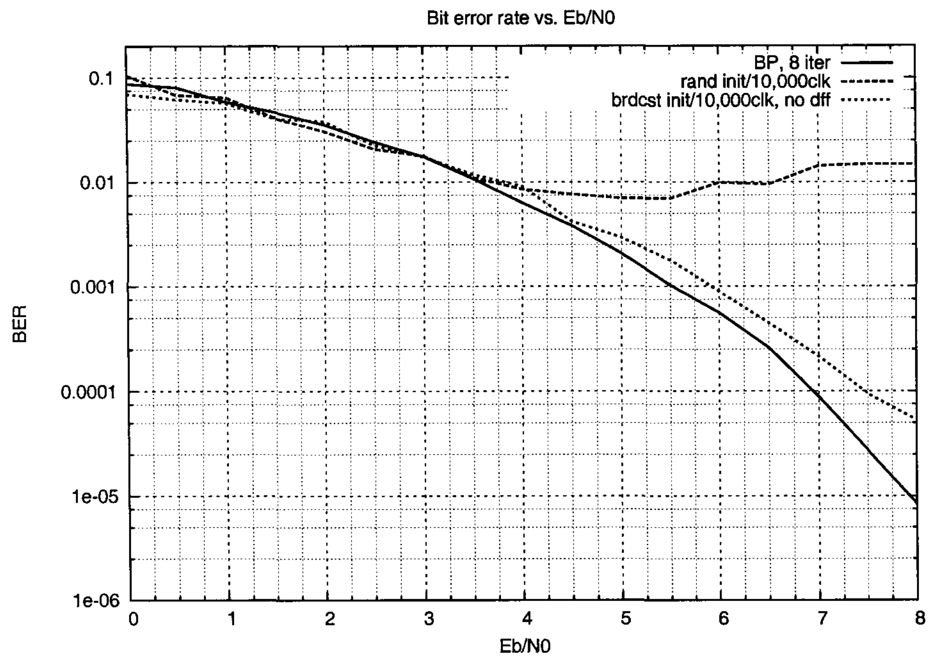


Figure 4.17: Effect of broadcast initialization with no D flip-flop in parity circuit

initialization.

Figure 4.17 shows the BER curve for the stochastic decoder with broadcast initialization (including the removal of the D flip-flops in the parity circuits) operating on the (7,4) Hamming code. This curve is contrasted with the output of the baseline decoder and of the ideal belief propagation algorithm operating for 8 iterations. The simple change of adding broadcast initialization produces marked improvement in the output. The BER curve now monotonically decreases, and differs from the belief propagation output at most by 0.75dB at a BER of 5×10^{-5}.

Parity Check Node Output Randomization

One means of proactively avoiding lockup in the equality nodes involves a low probability randomization of the parity check node outputs. Because the

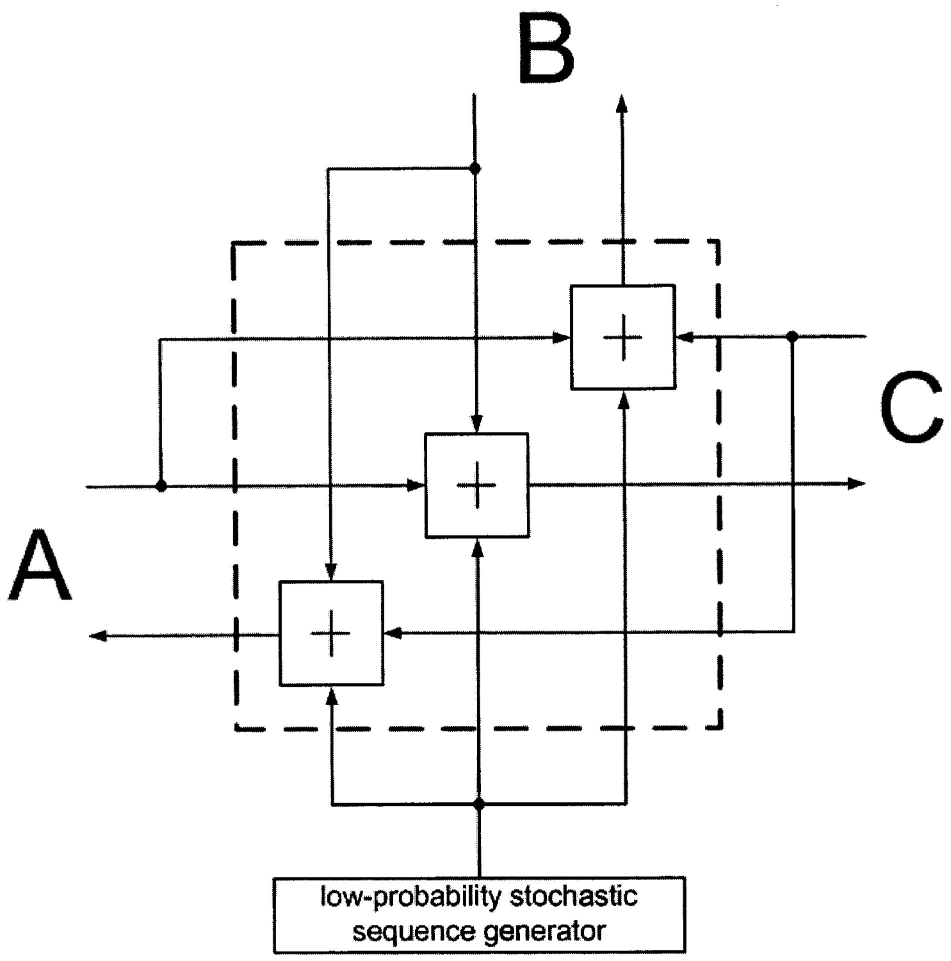


Figure 4.18: Parity check node with additional input edge for output randomization.

parity check node outputs directly feed back to the equality node inputs, such an approach would consequently randomize the equality inputs.

Introducing some artificial randomness to the parity check node outputs can be achieved by adding an input edge to the node. This additional input edge is then driven by a stochastic sequence generator. Figure 4.18 depicts a parity check node with an additional randomizing input edge.

The new input edge feeds into all of the constituent parity check circuits. On each clock cycle a new random bit is driven into the input by a stochastic sequence generator. If the bit is a '0', the outputs from the parity check node will not be affected. That is, the outputs will be the same as they would be with no additional randomizing input. If the bit is a '1', the outputs from the parity check node will all be inverted compared to their values without the additional randomizing input. The net effect is that the parity check node inverts its single-bit decision on all output edges when the input bit on the randomizing edge is a '1'.

Log-likelihood Ratio Scaling

Log-likelihood ratio scaling represents an effort to increase switching activity by preprocessing the channel-noise-added output values from the demodulator before converting these values to probabilities of being 1s and passing them to the decoder.

The scaling operation is accomplished by first selecting a scaling factor, β. All noisy values, n_i, from the demodulator that constitute a received codeword are the scaled in accordance with

$$n_i = \frac{n_i \beta}{\max\{|n_i|\}}. \tag{4.2}$$

That is, each noisy value is multiplied by the scaling factor then divided by the absolute value of the largest noisy value in the noisy value set. Selecting smaller values of β causes the codeword bit probabilities to be compressed closer to 0.5 when the noisy values are subsequently converted to probabilities of being 1s. More transitions will occur in the input stochastic sequences to the decoder when the probability values are all compressed closer to 0.5.

Figure 4.19 shows the BER curve for the baseline stochastic decoder with LLR scaling with β separately set to 0.8 and 0.9 and operating on the (7,4) Hamming code. These two curves are contrasted with the output of the base-

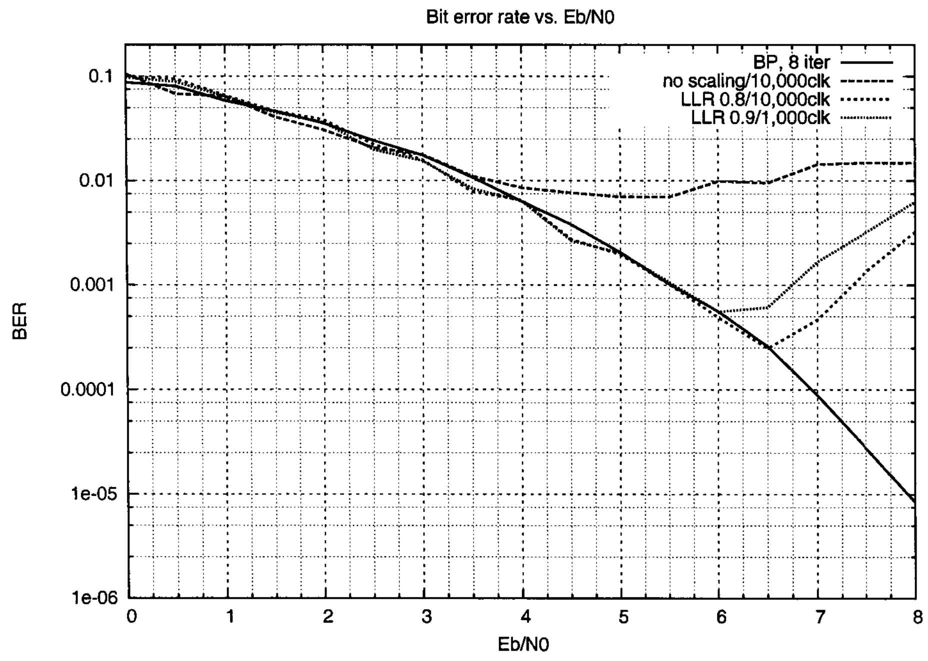


Figure 4.19: Effect of LLR scaling

line decoder and of the ideal belief propagation algorithm operating for 8 iterations. The notable feature is that employing LLR scaling delays the onset of the performance degradation of the decoding result at higher SNR values. It would appear that the increased switching activity has the desired effect of keeping the decoder from locking at higher SNR values.

Factor Graph Layering

Even though stochastic computation is essentially serial, at least in terms of its metric representations, this does not preclude a "layered" implementation of the decoder. Layering involves the instantiation of L identical factor graphs in one decoder. In such an arrangement every node is duplicated L times and edge connections are made between nodes exactly as in the single-layered factor graph with the exception that the connections are randomly permuted amongst the layers. Thus, a form of spatial diversity is realized.

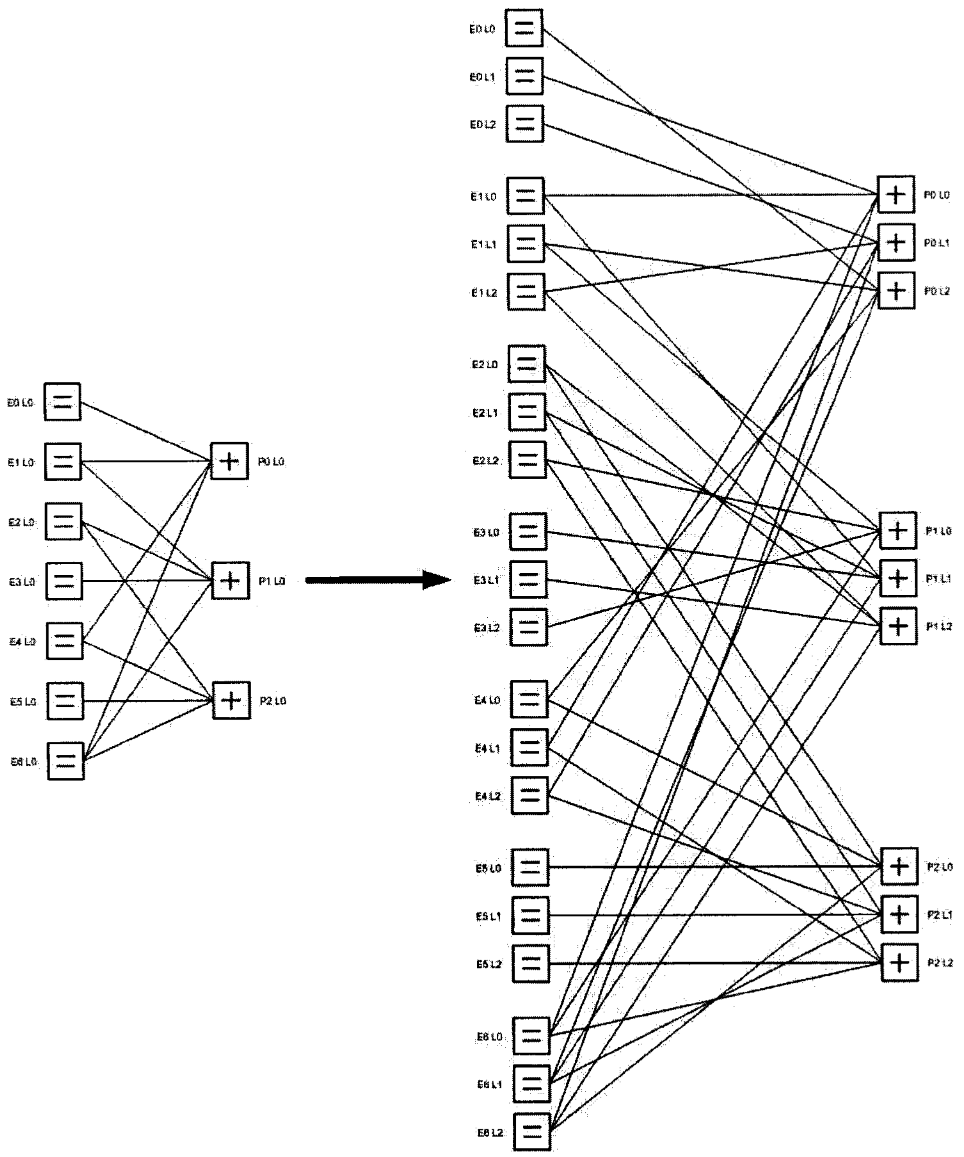


Figure 4.20: Single-layer factor graph (left) and three-layer factor graph (right) for a (7,4) Hamming code. The node labels indicate the node number (E{x}, P{x}) and the layer of the node (L{x})

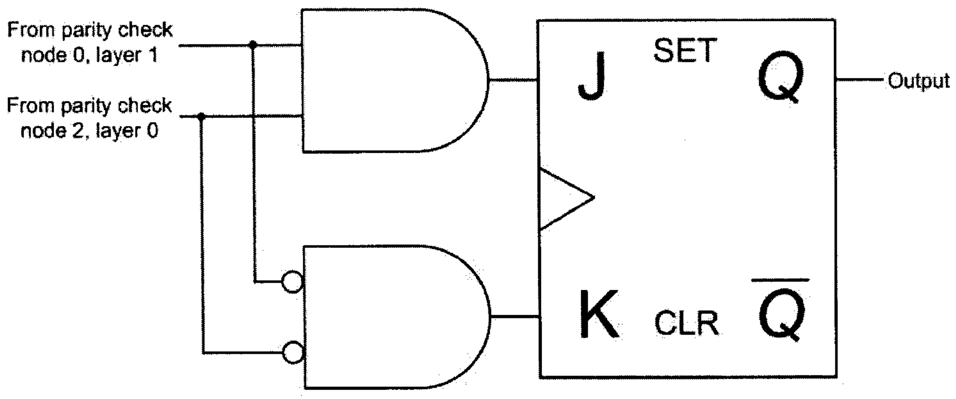
The left side of Figure 4.20 depicts a standard single-layered factor graph for a (7,4) Hamming code, while the right side of the figure depicts one possible three-layered graph for the same code. The labels E{x} and P{x} enumerate the equality nodes and parity check nodes, respectively, while the L{x} labels indicate the layer of the node.

Note that the equality nodes and parity check nodes are connected in fundamentally the same way in the single-layered graph and the three-layered graph. It is only the connections between layers in the three-layered graph that are randomized. For example, the "E0" equality node(s) connect to the "P0" parity check node(s) in both graphs. In the three-layered graph, however, equality node E0L0 connects to parity check node P0L2, equality node E0L1 connects to parity check node P0L0, and equality node E0L2 connects to parity check node P0L1.

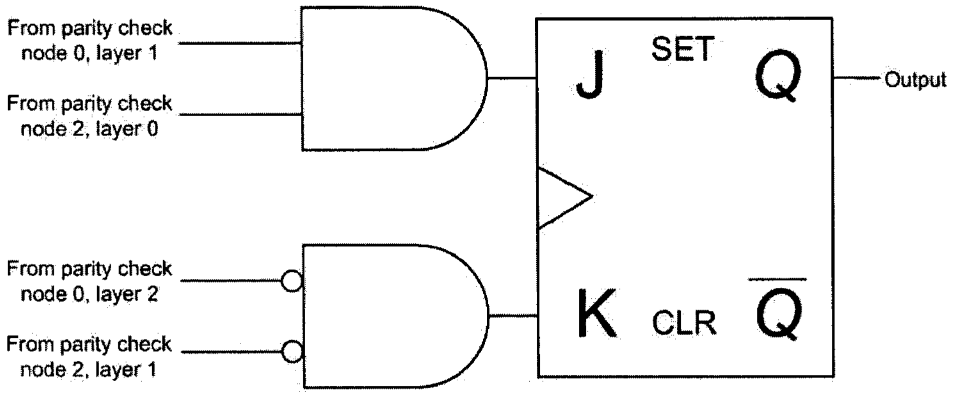
There are two degrees to which spatial interleaving can occur: lower-intricacy and higher-intricacy. Lower intricacy interleaving is achieved by permuting connections between layers as in the three-layered factor graph of Figure 4.20.

Higher intricacy interleaving is achieved by separately routing edges to the two AND gates in each equality circuit. In the conventional construction of the equality circuit, the same set of inputs is fed into both AND gates. The difference between lower intricacy and higher intricacy interleaving at the level of the equality circuit is highlighted in Figure 4.21.

One upshot of higher intricacy interleaving is that the inputs to the JK flip-flop are decorrelated. Another effect is that the toggle operation of the JK flip-flop can potentially occur. It is possible for the J and K inputs to both be '1' because the sets of inputs to the two AND gates in the equality circuit are different. The toggle operation is not possible in the conventional equality


From parity check node 0, layer 1
From parity check node 2, layer 0
J
SET
Q
Output
K
CLR
$\overline{Q}$

(a)


From parity check node 0, layer 1
From parity check node 2, layer 0
From parity check node 0, layer 2
From parity check node 2, layer 1
J
SET
Q
Output
K
CLR
$\overline{Q}$

(b)

Figure 4.21: (a) Equality circuit with lower-intricacy interleaving between factor graph layers. (b) Equality circuit with higher-intricacy interleaving between factor graph layers.

circuit because the inputs to the two AND gates are the same. The specific effects of the toggle operation on the computation of equality node constraint as well as on switching activity are worthy of further study.

A layered implementation of a stochastic iterative decoder has interesting performance implications. The original motivation for layering was speed. It was expected that decoding using L layered stochastic decoders could be terminated after $1/L$ the number of clock cycles normally required by a single decoding layer. After the equality node lockup problem was discovered, layering was seen as being more important as a means of promoting a heightened degree of randomness in the decoder and assisting in increasing switching activity.

Layering can further have the desirable effect of increasing the length of the shortest cycles in the factor graph. Recall that a cycle is a path that begins and terminates at the same node without traversing any edge more than once. If the edge connections between layers are made appropriately, the shortest cycle in a single-layer factor graph can generally be expected to increase in length by a factor of at least L.

The obvious detraction of a layered decoder implementation is the consumption of at least L times as much die area, and likely more because of the non-linear increase in the routing requirements between the nodes. Another drawback is that L times as many stochastic sequence generators are required for a layered decoder. It is vital that each equality node has its own independent (ie. uncorrelated with other input sequences) input sequence. This problem may be marginalized by the first implementation approach for a stochastic sequence generator described in Section 4.4.

Outputs from the equality nodes, however, can still be summed in a single layer of up-down counters.

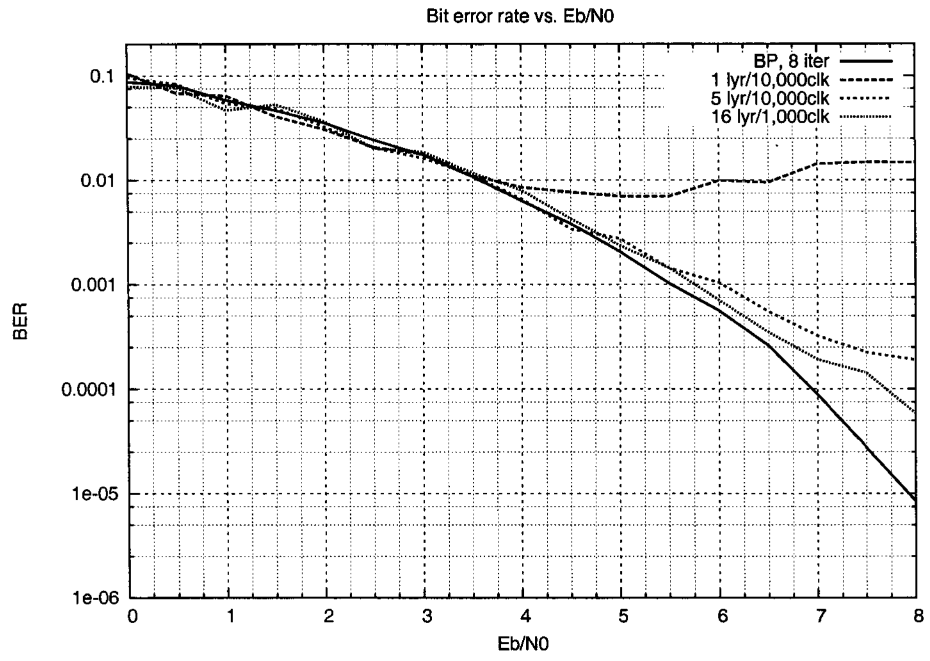


Figure 4.22: Effect of layering

Figure 4.22 shows the BER curve for the baseline stochastic decoder separately with 5 layers and 16 layers, operating on the (7,4) Hamming code. These two curves are contrasted with the output of the baseline decoder (the 1 layer curve) and of the ideal belief propagation algorithm operating for 8 iterations. With layering, the BER curve decreases monotonically. With 16 layers, a loss of approximately 0.75dB at a BER of 6×10^{-5} is observed.

Combined Techniques

Prior to proceeding to the last performance improving technique, supernodes, we examine the combination of the best of the previously analyzed techniques.

Figure 4.23 shows the BER curve for the baseline stochastic decoder with LLR scaling ($\beta = 0.8$), 16 layers, and broadcast initialization operating on the (7,4) Hamming code. This curve is contrasted with the curves for LLR scaling on its own, broadcast initialization on its own, and the belief propagation al-

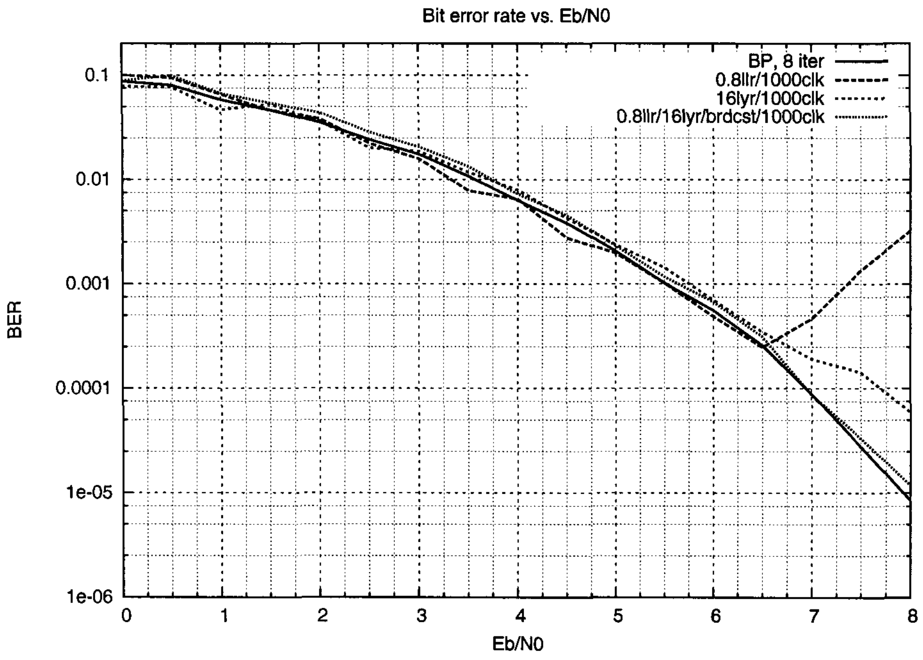


Figure 4.23: Effect of combining broadcast initialization with LLR scaling and layering

gorithm operating for 8 iterations. The combined feature decoder outperforms the other features in isolation. Moreover, the combined feature decoder closely tracks the belief propagation curve and shows a minute loss of only 0.125dB at a BER of approximately 1.2×10^{-5}.

Confident in the capabilities of the stochastic decoder that combines LLR scaling, layering, and broadcast initialization, we attempt to tackle a much larger code. Figure 4.24 shows the BER curve for the combined feature decoder operating on a rate 1/2, 1024-bit (3,6) regular LDPC code. The BER curve is contrasted with the output of the belief propagation algorithm operating for 8 iterations. When the belief propagation algorithm reaches a BER of 3×10^{-7} at an SNR of 3.5dB, the stochastic decoder shows little error correction with a BER of 7.5×10^{-2}. Clearly more investigation is required in the future to make the stochastic decoder work with large codes.

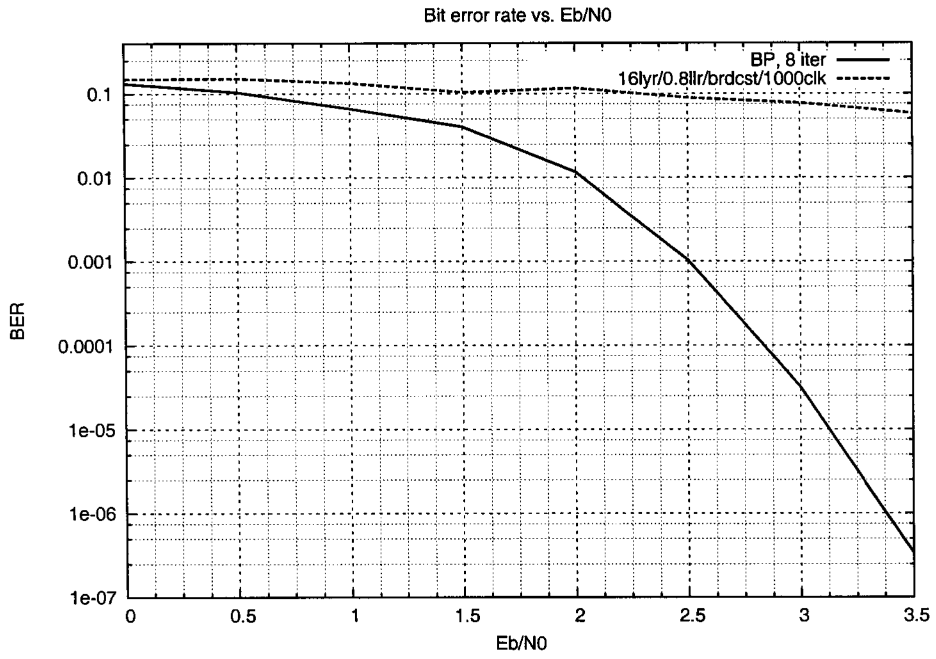


Figure 4.24: Effect of combining broadcast initialization with LLR scaling and layering for rate 1/2 1024-bit LDPC code

Supernodes

A final performance-enhancing measure studied in this thesis is the use of "super" equality nodes, or supernodes. A supernode receives and outputs stochastic sequences, but internally performs conventional computations. This approach therefore represents the most significant departure from the baseline decoder.

The supernode decoder differs architecturally from the baseline decoder in that the equality nodes are replaced with supernodes, the stochastic sequence generators are moved inside the supernodes, and the D flip-flops are removed from the parity check circuits. Figure 4.25 shows the block diagram of a three-input supernode. In reference to this figure, the supernode decoding algorithm proceeds as follows:

1. The input accumulators within the supernodes are reset to 0.

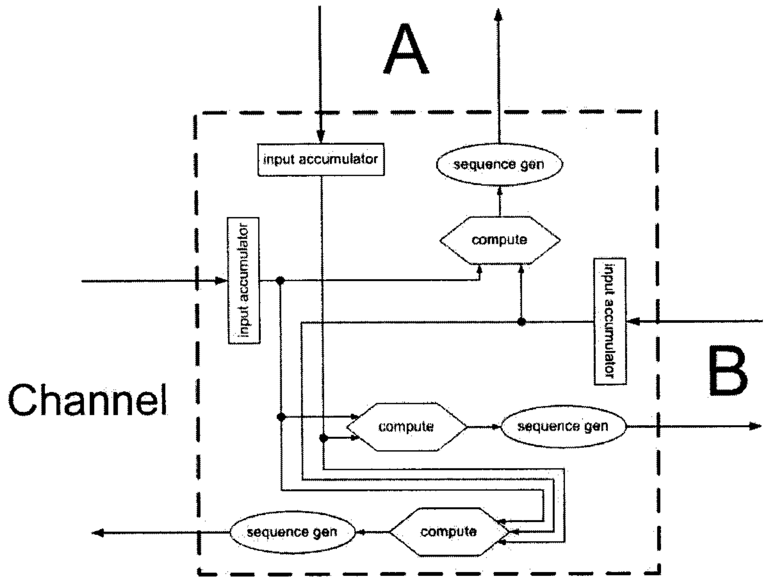


Figure 4.25: Block diagram of supernode

2. For each codeword bit, the channel probability value (i.e. the probability that the bit is a 1) is loaded directly into each stochastic sequence generator within the associated supernode. This is a broadcast initialization.

3. The decoder is clocked. The stochastic generators in the supernode produce a new output on every clock edge. The parity check nodes, without D flip-flops, pass bits back to the inputs of the supernodes. These bits are summed in the input accumulators.

4. After a specific number of clock cycles, n_c, the accumulator values are forwarded to the compute blocks. Here they are converted into probabilities by dividing by n_c. Each compute block calculates a new output value from the input probabilities according to the equality node constraint. The computation is performed in conventional fixed bit arithmetic. This output value is loaded into the attached stochastic sequence generator. The input accumulators are cleared. The stochastic sequence generators

produce stochastic bits based on their updated values. After another n_c cycles, the update procedure is repeated.

5. The decoder is allowed to operate for a fixed number of clock cycles before threshold conversion is performed in the up/down counters to obtain a hard-decision decoding result.

A notable feature of the above algorithm is that the supernodes packetize the information transferred between themselves and the parity check nodes. The baseline stochastic decoder transfers a continuous stream of bits between the nodes without any update interval, in much the way that currents flow freely between nodes in an analog decoder. The decoder with supernodes, however, effectively creates discrete iterations where each iteration is marked by the transfer of n_c stochastic bits between the nodes. The packetized nature of the supernode decoder makes it more like a conventional decoder with a sequence of stochastic bits used in place of a single real number as the message passed between the nodes.

Figure 4.26 shows the BER curve for the supernode decoder operating on the (7,4) Hamming code. The internal computations in the compute blocks were performed in floating-point arithmetic for a proof of concept of the supernode. Four curves are shown for the supernode decoder with the paired numbers in the graph legend indicating n_c and the total number of clocks per codeword, respectively. These curves are contrasted with the output of the belief propagation algorithm operating for 8 iterations. It can be seen that all of the supernode curves are monotonically decreasing, and that both a higher number of clocks per iteration and a higher total number of clocks improve performance. The supernode decoder operating for 20,000 clock cycles with 2000 cycles per iteration shows a loss of 1dB at a BER of 1×10^{-4}. The

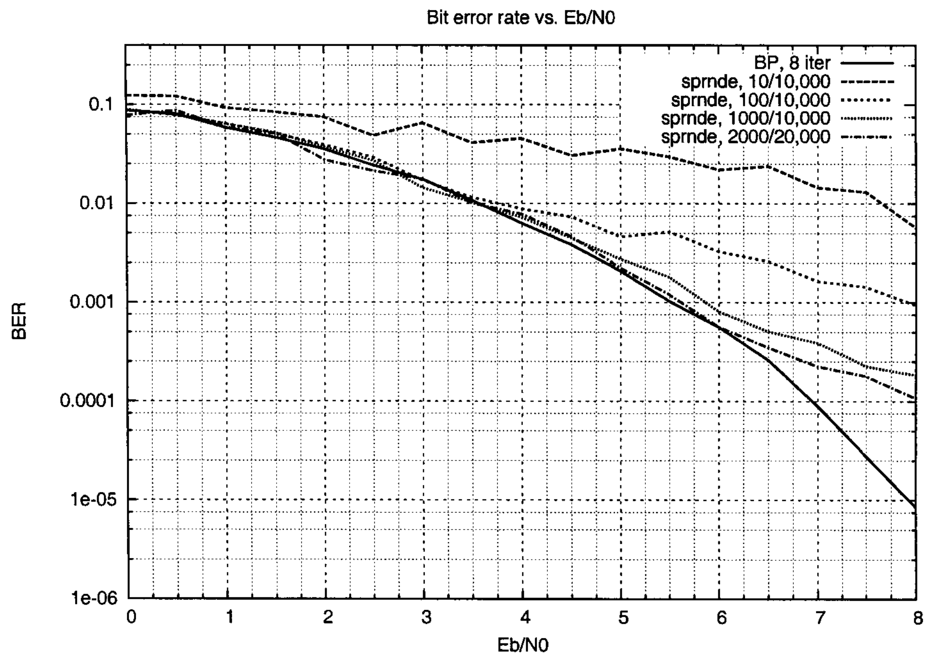


Figure 4.26: Effect of supernode

16-layer stochastic decoder outperforms the supernode decoder at comparable clock counts, but LLR scaling has not been applied to the supernode decoder.

4.4 Stochastic Sequence Generation

Aside from the implementation of the factor graph decoder itself, one of the most important hardware considerations is the generation of weighted stochastic sequences for the channel inputs of the equality nodes.

Generation of the stochastic sequences can be accomplished through the use of a weighted linear feedback shift register (LFSR) at each required generation point. This approach is infeasible, however, because of the requirement for an independent stochastic generator for every equality node. Most of the silicon area would be consumed with circuitry involved in the generation of stochastic sequences. Other researchers have encountered this inefficiency in their own

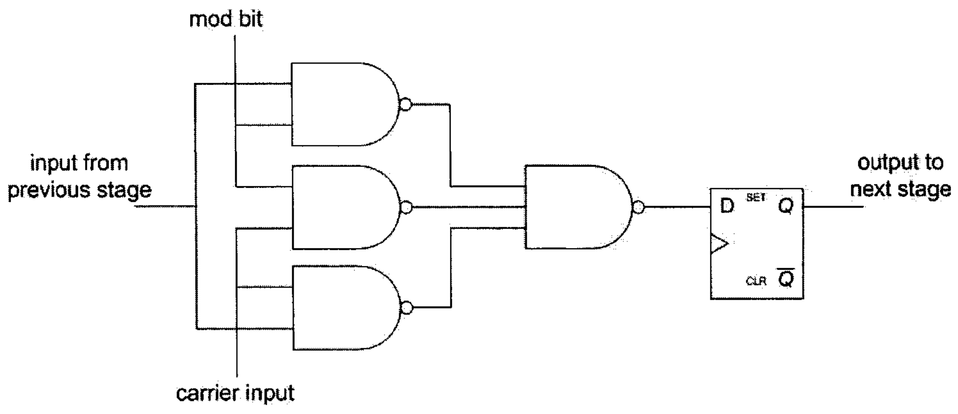


Figure 4.27: Modulator for pipelined stochastic sequence generation

VLSI stochastic computer implementations [59].

This section will focus on two proposals from the literature for improving the silicon area efficiency of stochastic sequence generation: pipelined modulators and CMOS ring oscillators.

Pipelined Modulators

The approach of using pipelined modulators is architecturally described in [57] and mathematically proven in [58]. Figure 4.27 depicts an instance of what the authors term a "modulator", which forms one stage of a generation pipeline.

The number, k, of cascaded modulators in the pipeline is equal to the number of bits of resolution required to represent the probability value being encoded as a stochastic sequence. The output of each modulator is gated with a D flip-flop to facilitate pipelining.

The probability value, represented as a binary value of length k, is supplied to the pipeline through the mod bit inputs of the modulators. The most significant bit of the probability value is supplied to the first modulator in the pipeline and the least significant bit is supplied to the last modulator in the pipeline.

The carrier input is supplied with a "carrier stream", a stochastic sequence having a probability of 0.5 and changing values synchronously with the pipeline.

The input of the first pipeline stage is the all-zero sequence. The output of the last pipeline stage is the weighted stochastic sequence ready for input into the stochastic computer. In the case of a stochastic iterative decoder, the last pipeline stage would be connected to the channel input of an equality node.

The modulator performs one of two operations depending on the value of the mod bit. When the mod bit is '0', the output of the modulator is the bitwise AND of the input and carrier streams. In this case, the relationship of the modulator output probability to its input probability is given by

$$Pout_{\text{mod}=0} = \frac{1}{2}Pin. \tag{4.3}$$

When the mod bit is '1', the output of the modulator is the bitwise OR of the input and carrier streams. In this case, the relationship of the modulator output probability to its input probability is given by

$$Pout_{\text{mod}=1} = \frac{1}{2}Pin + \frac{1}{2}. \tag{4.4}$$

The resolution, R, of a k-bit modulator pipeline is given by

$$R = \frac{1}{2^k}. \tag{4.5}$$

Variance of the output sequence is highest when the probability value to be encoded is 0.5. The number of bits that must be produced by the pipeline to achieve maximum accuracy is given by

$$n = 2^{2\nu-2}, \tag{4.6}$$

where ν is the number of bits in the binary representation of the probability value being encoded.

The pipelined modulators require supporting hardware to produce the mod bits and the carrier streams. Depending on whether the probability value to be encoded is initially represented as a binary value or an analog voltage, a respective quantizer or A/D converter is required to produce the mod bits. In the case of the stochastic decoder, the production of the mod bits could be made an integral part of the demodulator supplying noisy channel estimates to the decoder.

A device utilizing n bit stream generators, each consisting of k pipeline stages, requires the production of kn statistically independent carrier streams. Because it is impractical to achieve the ideal of a true random source for each carrier stream, the authors describe a method by which a single LFSR configured to generate a maximal-length pseudo-random bit sequence (PRBS) can be used to supply all of the required carrier streams in the device.

The generation of numerous independent sequences from one PRBS generator is described in [56]. The procedure described therein derives multiple random streams from a relatively small number of well-spaced taps on the shift register. The authors of [57], however, assert that this approach could still easily lead to a high degree of correlation between the derived sequences, which would be detrimental for the operation of a stochastic computer. They instead propose tapping successive bits of the PRBS generator. Sequences derived in this manner have overlap but almost no correlation [58].

To ensure that the carrier stream inputs to successive modulators along the pipeline of a given stochastic sequence generator are not coincident, the PRBS shift register is shifted in a direction opposite to the direction of the modulator pipeline. For sufficiently large values of the PRBS shift register size, n, the carrier streams will be highly uncorrelated. The minimum length of the PRBS shift register is given by the inequality

$$n \geq 2k, \tag{4.7}$$

where k is the number of modulators in a stochastic sequence generator pipeline.

Pipelining allows for a high rate of operation, and the authors posit that 50MHz should be easily attained for the pipelined modulator approach. Years later, Brown and Card described an almost identical stochastic sequence generator architecture in [44].

CMOS Ring Oscillator

A more elegant, albeit dependent on some custom layout, solution to the problem of generating weighted stochastic sequences is presented in [59]. The authors propose the high-frequency oscillator with a lower rate sampling flip-flop depicted in Figure 4.28.

The oscillator consists of five CMOS inverters chained together, with the output of the last inverter fed back into the input of the first inverter, thus forming a ring oscillator circuit. If the input capacitance of the circuit is sufficiently low the circuit will be highly sensitive to noise, which is a desirable trait in this application.

The authors assert that phase uncertainty in the oscillator must be greater than or equal to 2π in order to ensure complete spatial and temporal randomness in the output sequence. In this context, spatial randomness means low cross-correlation between sequences produced by separate stochastic sequence generators. Temporal randomness means low autocorrelation amongst the output bits of a single stochastic sequence generator.

Because the feedback voltage is indeterminate, the switching times of the

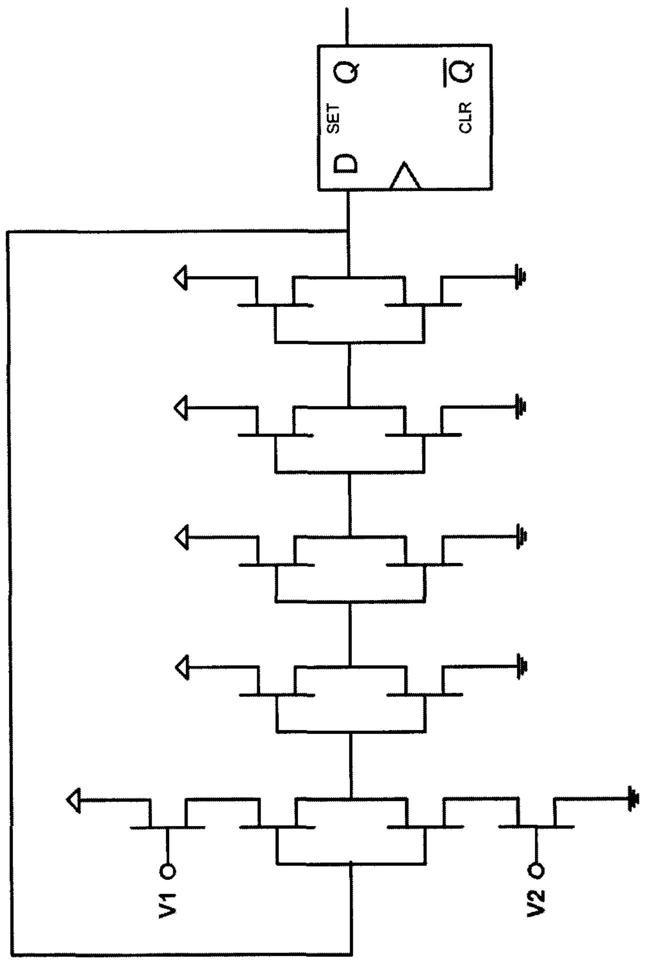


Figure 4.28: CMOS oscillator with sampling flip-flop for stochastic sequence generation

oscillator will also be indeterminate. To provide an output with a fixed period, a D flip-flop is additionally connected to the output of the fifth CMOS inverter. If the clock supplied to the D flip-flop is sufficiently slow relative to the aperiodic oscillation frequency of the oscillator then the sampled output bit stream will be random.

The ratio of the period of time that the oscillator output is high to an arbitrary sampling period is $\frac{T_{on}}{T}$. Thus, the probability value to be encoded is given by $\frac{T_{on}}{T}$. The utility of this circuit is that the weight of the output stochastic stream is easily voltage-controlled by the inputs V1 and V2.

The regulation relationship between the inputs V1 and V2 and the output $\frac{T_{on}}{T}$ ratio is determined experimentally. The results produced by the authors are for an implementation in an outmoded 1.5μm process and are for $\frac{T_{off}}{T}$, the inverse period of $\frac{T_{on}}{T}$.

Again, because a stochastic decoder would be preceded by a demodulator, the output of channel estimates mapped to V1 and V2 could be made an integral part of the demodulator to reduce signal conversion overhead.

Experimental results from an IC with 8 on-board stochastic sequence generators show that $\frac{T_{off}}{T}$ values between in the range $[0, 1]$ can be produced by varying V1 and V2.

A study of cross-correlation between sequence generators based on their physical proximity on the die is also performed with a sampling frequency of 100KHz, 1000-bit sequence lengths, and $\frac{T_{off}}{T}$ fixed to 0.5. It is proven that neighbouring sequence generator circuits may exhibit unacceptably high cross-correlation when placed too close to one another, but will be independent and uncorrelated otherwise. The minimum distance for acceptable cross-correlation is not quantified, so it would remain a point of study if this approach to stochastic sequence generation were used in a stochastic decoder.

Increasing the sampling frequency of the D flip-flop does not increase the level of cross-correlation between generation circuits.

A higher switching frequency in the oscillator, however, will increase the autocorrelation of the output sequence of an individual sequence generator. The authors note that autocorrelation is not usually a concern for stochastic applications where stochastic sequences are being combined through logic gates. The only requirement is that the sequences have low cross-correlation.

The authors identify a reduction of autocorrelation in the sequence generator output as future work. The two identified methods of improvement are to add a noise source to the inverter chain, and to decrease the input capacitance of the circuit by using a finer CMOS process.

4.5 Summary

This chapter has described the architecture and operation of the stochastic decoder presented in [1], then expanded on this initial work. A baseline stochastic decoder was described against which performance enhancing changes could be measured. The problem of equality node lockup was elaborated, and means to ameliorate this problem were examined. These approaches included broadcast initialization, parity check node output randomization, log-likelihood ratio scaling, factor graph layering, and supernodes. Last, two approaches to generating stochastic sequences were described.

Chapter 5

Conclusions and Future Directions

5.1 Contributions and Conclusion

In this thesis we have presented the stochastic iterative decoder, a novel application of stochastic computing to the design of iterative decoders for low density parity check codes. By combining the power and area efficiency of stochastic computational elements with the error performance of the belief propagation decoding algorithm, it may be possible to build more efficient iterative decoders.

The stochastic decoder design techniques described herein provide a foundation for future research into stochastic decoders.

The following subsections summarize the contributions of this thesis:

Stochastic Computation Tutorial

A tutorial of stochastic computing technology has been provided. Stochastic computing does not appear to be well known outside of the neural networks design community, and even within this community its mention is intermittent. Stochastic computing is an important technology for efficient probabilistic processing. The applicability of stochastic computing to iterative decoding has been demonstrated.

Deficiency Analysis

Performance impediments in the stochastic decoder have been identified. Chief among these is the problem of equality node lockup, whether due to inputs with few transitions or equality nodes with higher degrees. The accuracy of stochastic sequences, stochastic parity check nodes, and stochastic equality nodes has been simulated.

Performance Enhancements

Performance enhancing measures for the stochastic decoder were proposed and studied. These included broadcast initialization, parity check node output randomization, log-likelihood ratio scaling, factor graph layering, and supernodes. Simulation results were provided for each of these approaches.

5.2 Future Directions

The following subsections outline suggested avenues of continued research into stochastic iterative decoders:

Mathematical Analysis

It is important to understand the mathematical underpinnings of the stochastic decoder. Density evolution techniques should be employed to develop a more mathematical understanding of the decoder. Such procedures should also help to define fundamental decoder limitations from a mathematical perspective.

Code Design

It is possible that some LDPC codes are more well suited to implementation in a stochastic decoder than others. The application of recent work on stopping set analysis, also known as trapping sets, may provide insights into the design of codes that are well tailored for stochastic decoders.

Applicability to Turbo Codes

Thus far, stochastic decoder research has focused on LDPC codes. The applicability to Turbo and other codes should be investigated [65].

Implementations

In addition to the simulations presented in this thesis, a stochastic decoder has already been implemented in an FPGA [66]. ASIC implementations are the next logical step. Subsequent implementations should provide a complete solution, with sigma-delta modulators at the front end of the decoder and hardware stochastic sequence generation.

Power and Clock Rate Analysis

To better understand the physical performance of stochastic decoders, power consumption should be analyzed. Maximum clock rates should be characterized through static timing analysis. Power and achievable clock rates should be considered for a range of CMOS processes.

Applicability to Fault-Tolerant Computing

Many nanotechnology devices being proposed exhibit stochastic properties. It should be investigated whether these devices would be suitable for higher-level systems integration.

Bibliography

[1] V.C. Gaudet and A.C. Rapley. Iterative decoding using stochastic computation. *Electronics Letters*, 39(3):299–301, February 2003.

[2] S.G. Glisic. *Adaptive WCDMA: Theory and Practice.* John Wiley & Sons, New York, 2003.

[3] Japan's proposal for candidate radio transmission technology on imt-2000: W-CDMA [online]. Available: http://www.arib.or.jp/IMT-2000/proponent.

[4] Consulatative Committee for Space Data Systems (CCSDS). *Telemetry Channel Coding*, May 1999. Blue book 101.0-B-4.

[5] Multiplexing and channel coding (FDD), universal mobile telecommunications systems (UMTS), 3G TS2 5.212 version 3.3.0 release 1999 [online]. Available: http://www.etsi.org.

[6] A. Morello and V. Mignone. DVB-S2: The second generation standard for satellite broad-band services. *Proceedings of the IEEE*, 94(1):210–227, January 2006.

[7] Digital video broadcasting return channel via satellite (DVB-RCS) background book [online]. Avail-

able: http://www.dvb.org/technology/white_papers/Tech-Papers_RCS_nera.pdf.

[8] IEEE 802 standards [online]. Available: http://standards.ieee.org/getieee802.

[9] J. Hagenauer and M. Winklhofer. The analog decoder. In *IEEE International Symposium on Information Theory*, page 145, Cambridge, Massachusetts, August 1998.

[10] H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy. Probability propagation and decoding in analog VLSI. *IEEE Transactions on Information Theory*, 47(2):837–843, February 2001.

[11] V.C. Gaudet. *Architecture and Implementation of Analog Iterative Decoders.* Doctor of philosophy dissertation, University of Toronto, Toronto, Canada, 2003.

[12] J.G. Proakis. *Digital Communications.* McGraw-Hill, New York, 4^{th} edition, 2001.

[13] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[14] R.G. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, 8(1):21–28, January 1962.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *IEEE International Conference on Communications*, volume 2, pages 1064–1070, Geneva, Switzerland, May 1993.

[16] D.J.C. MacKay and R.M. Neal. Good codes based on very sparse matrices. In *Cryptography and Coding. 5th IMA Conference*, number 1025, pages 100–111. Springer, Berlin, 1995.

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

[18] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, February 2001.

[19] T.J. Richardson and R.L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, February 2001.

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

[21] A.J. 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.

[22] M.M. Mansour and N.R. Shanbhag. A 1.6 Gbit/s 2048-bit programmable and code-rate tunable LDPC decoder chip. In 3^{rd} *International Symposium on Turbo Codes*, pages 137–140, Brest, France, September 2003.

[23] T. Zhang and K. Parhi. A 54 Mbps (3,6)-regular FPGA LDPC decoder. In *IEEE Workshop on Signal Processing Systems*, pages 127–132, San Diego, CA, October 2002.

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

[25] C.-C. Lin, K.-L Lin, H.-C. Chang, and C.-Y. Lee. A 3.33Gb/s (1200,720) low-density parity check code decoder. In *31st European Solid-State Circuits Conference*, pages 211–214, Grenoble, France, September 2005.

[26] A. Darabiha, A.C. 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*, volume 5, pages 5194–5197, Kobe, Japan, May 2005.

[27] B. Bougard, A. Giulietti, V. Derudder, J.-W. Weijers, S. Dupont, L. Hollevoet, F. Catthoor, L. Van der Perre, H. De Man, and R. Lauwereins. A scalable 8.7nJ/bit 75.6Mb/s parallel concatenated convolutional (Turbo-) codec. In *IEEE International Solid-State Circuits Conference*, volume 1, pages 152–484, San Francisco, CA, February 2003.

[28] M.A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L.M. Davis, G. Woodward, C. Nichol, and R.-H. Yan. A unified Turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 37(11):1555–1564, November 2002.

[29] S.-J. Lee, N.R. Shanbhag, and A.C. Singer. A 285-MHz pipelined MAP decoder in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 40(8):1718–1725, August 2005.

[30] M. Bekooij, J. Dielissen, F. Harmsze, S. Sawitszki, J. Huisken, A. van der Weri, and J. van Meerbergen. Power-efficient application-specific VLIW

processor for Turbo decoding. In *IEEE International Solid-State Circuits Conference*, pages 180–181, San Francisco, CA, February 2001.

[31] C. Berrou, P. Combelles, P. Penard, and B. Talibart. An IC for Turbo-codes encoding and decoding. In *IEEE International Solid-State Circuits Conference*, pages 90–91, San Francisco, CA, February 1995.

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

[33] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25 μm biC-MOS tailbiting MAP decoder. In *IEEE International Solid-State Circuits Conference*, pages 356–357, San Francisco, CA, February 2000.

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

[35] C. Winstead, V. Gaudet, and C. Schlegel. A CMOS analog $(16,11)^2$ Turbo product decoder. In *3^{rd} Analog Decoding Workshop*, Banff, AB, June 2004.

[36] D. Vogrig, A. Gerosa, A. Neviani, A.G. 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.

[37] N. Nguyen, C. Winstead, V.C. Gaudet, and C. Schlegel. A 0.8V CMOS analog decoder for an (8,4,4) extended Hamming code. In *2004 International Symposium on Circuits and Systems*, volume 1, pages I – 1116–19, Vancouver, BC, May 2004.

[38] F. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarköy, and G.S. Moschytz. All-analog decoder for a binary (18,9,5) tail-biting trellis code. In *ESSCIRC'99*, pages 362–365, Duisburg, Germany, September 1999.

[39] F. Lustenberger. *On the Design on Analog Iterative VLSI Decoders.* Doctor of technical sciences dissertation, ETH Zürich, Zürich, Switzerland, 2000.

[40] C. Winstead, J. Dai, W.J. Kim, S. Little, and Y.-B. Kim. Analog MAP decoder for (8,4) Hamming code in subthreshold CMOS. In *Advanced Research in VLSI Conference*, pages 132–147, Salt Lake City, UT, March 2001.

[41] M.H. Shakiba, D.A. Johns, and K.W. Martin. An integrated 200-MHz 3.3-V biCMOS class-IV partial-response analog Viterbi decoder. *IEEE Journal of Solid-State Circuits*, 33(1):61–75, January 1998.

[42] S. Hemati, A.H. Banihashemi, and C. Plett. An 80-Mb/s 0.18-μm CMOS analog min-sum iterative decoder for a (32,8,10) LDPC code. In *IEEE 2005 Custom Integrated Circuits Conference*, pages 243–246, Ottawa, Canada, September 2005.

[43] B.R. Gaines. *Advances in Information Systems Science*, volume 2, chapter 2. Stochastic Computing Systems, pages 37–172. Plenum, New York, 1969.

[44] B.D. Brown and H.C. Card. Stochastic neural computation I: Computational elements. *IEEE Transactions on Computers*, 50(9):891–905, September 2001.

[45] J.A. Dickson, R.D. McLeod, and H.C. Card. Stochastic arithmetic implementations of neural networks with in situ learning. In *IEEE International*

Conference on Neural Networks, volume 2, pages 711–716, San Francisco, CA, March-April 1993.

[46] B.D. Brown and H.C. Card. Stochastic neural computation II: Soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, September 2001.

[47] S.L.T. Marin, J.M.Q. Reboul, and L.G. Franquelo. Digital stochastic realization of complex analog controllers. *IEEE Transactions on Industrial Electronics*, 49(5):1101–1109, October 2002.

[48] D.K. McNeill, D. Zhao, C. Shafai, N. Chadha, A. Cuhadar, and H.C. Card. Processing noisy analog signals from microsensor arrays and VLSI imagers using stochastic binary computataions. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 986–990, Winnipeg, MB, May 2002.

[49] M. van Daalen, P. Jeavons, and J. Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, Napa, CA, April 1993.

[50] J.M. Quero, J.G. Ortega, C.L. Janer, and L.G. Franquelo. VLSI implementation of a fully parallel stochastic neural network. In *IEEE International Conference on Neural Networks*, volume 4, pages 2040–2045, Orlando, FL, June-July 1994.

[51] S.L. Bade and B.L. Hutchings. FPGA-based stochastic neural networks - implementation. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 189–198, Napa Valley, CA, April 1994.

[52] M.S. Tomlinson, D.J. Walker, and M.A. Sivilotti. A digital neural network architecture for VLSI. In *IJCNN International Joint Conference on Neural Networks*, volume 2, pages 545–550, San Diego, CA, June 1990.

[53] D.K. McNeill and H.C. Card. Refractory pulse counting processes in stochastic neural computers. *IEEE Transactions on Neural Networks*, 16(2):505–508, March 2005.

[54] C.L. Janer, J.M. Quero, J.G. Ortega, and L.G. Franquelo. Fully parallel stochastic computation architecture. *IEEE Transactions on Signal Processing*, 44(8):2110–2117, August 1996.

[55] J. Alspector, J.W. Gannett, S. Haber, M.B. Parker, and R. Chu. Generating multiple analog noise sources from a single linear feedback shift register with neural network applications. In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 1058–1061, New Orleans, LA, May 1990.

[56] J. Alspector, J.W. Gannett, S. Haber, and R. Chu. A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks. *IEEE Transactions on Circuits and Systems*, 38(1):109–123, January 1991.

[57] M. van Daalen, P. Jeavons, J. Shawe-Taylor, and D. Cohen. Device for generating binary sequences for stochastic computing. *Electronics Letters*, 29(1):80–81, January 1993.

[58] P. Jeavons, D.A. Cohen, and J. Shawe-Taylor. Generating binary sequences for stochastic computing. *IEEE Transactions on Information Theory*, 40(3):716–720, May 1994.

[59] J.G. Ortega, J.M. Quero, C.L. Janer, and L.G. Franquelo. Synaptic weight generation in VLSI stochastic neural networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 179–182, Perth, WA, November-December 1995.

[60] A. Orlitsky, K. Viswanathan, and J. Zhang. Stopping set distribution of LDPC code ensembles. *IEEE Transactions on Information Theory*, 51(3):929–953, March 2005.

[61] M. Schwartz and A. Vardy. On the stopping distance and the stopping redundancy of codes. *IEEE Transactions on Information Theory*, 52(3):922–932, March 2006.

[62] T. Tian, C.R. Jones, J.D. Villasenor, and R.D. Wesel. Selective avoidance of cycles in irregular LDPC code construction. *IEEE Transactions on Communications*, 52(8):1242–1247, August 2004.

[63] S. Lander and O. Milenkovic. Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes. In *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, pages 630–635, Maui, HI, June 2005.

[64] E. Cavus and B. Daneshrad. A performance improvement and error floor avoidance technique for belief propagation decoding of LDPC codes. In *IEEE* 16^{th} *International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 4, pages 2386–2390, Berlin, Germany, September 2005.

[65] C. Winstead, V.C. Gaudet, A. Rapley, and C. Schlegel. Stochastic iterative decoders. In *International Symposium on Information Theory, 2005*, pages 1116–1120, Adelaide, Australia, September 2005.

[66] W.J. Gross, V.C. Gaudet, and A. Milner. Stochastic implementation of LDPC decoders. In *Record of the Thirty-Ninth Asilomar Conference on Signals, Systems, and Computers*, pages 713–717, Pacific Grove, California, October-November 2005.