



Algorithms and Architectures for Low-Density Parity-Check Codecs

Chris Howland

A dissertation submitted to the
Department of Electrical and Electronic Engineering,
The University of Adelaide,
Australia
in partial fulfillment for the requirements of the degree of
Doctor of Philosophy
October 10th, 2001



THE UNIVERSITY OF ADELAIDE

Abstract

Low-density parity-check (LDPC) codes have been shown to achieve reliable transmission of digital information over additive white gaussian noise and binary symmetric channels at a rate closer to the channel capacity than any other practical channel coding technique. Although the theoretical performance of LDPC codes is extremely good, very little work to date has been done on the implementation of LDPC codes. Algorithms and architectures for implementing LDPC codes to achieve reliable communication of digital data over an unreliable channel are the subject of this thesis.

It will be shown that published methods of finding LDPC codes do not result in good codes and, particularly for high rate codes, often result in codes with error floors. Short cycles in the bipartite graph representation of a code have been identified as causing significant performance degradation. A cost metric for measuring the short cycles in a graph due to an edge is therefore derived. An algorithm for constructing codes through the minimisation of the cost metric is proposed. The algorithm results in significantly better codes than other code construction techniques and codes that do not have error floors at the bit error rates simulated and measured.

An encoding algorithm for LDPC codes is derived by considering the parity check matrix as a set of linear simultaneous equations. The algorithm utilises the sparse structure of the parity check matrix to simplify the encoding process. A decoding algorithm, relative reliability weighted decoding, is proposed for hard decision decoders. Like all other hard decision decoding algorithms, the proposed algorithm only exchanges a single bit of data between functional nodes of the decoder. Unlike previous hard decision algorithms, the relative probability and reliability of the bits is used to improve the performance of the algorithm.

A parallel architecture for implementing LDPC decoders is proposed and the advantages in terms of throughput and power reduction of this architecture are demonstrated through the implementation of two LDPC decoders in a 1.5V 0.16 μ m CMOS process. The first decoder is a soft decision decoder for a 1024-bit rate 1/2 irregular LDPC code with a coded data throughput of 1Gbs⁻¹. The second code implemented was a 32,640-bit rate 239/255 regular code. Both an encoder and decoder for this code are implemented with a coded data throughput of 43Gbs⁻¹ for use in fiber optic transceivers.

Statement of Originality

This work contains no material which has been accepted for the award of any degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent for this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

Chris Howland

on the 10th day of October 2001.

Acknowledgments

The author would like to acknowledge the guidance, tolerance and liberal approach to supervision of Michael Liebelt. The research contained in this thesis is the result of one year of internship in the DSP & VLSI Research Department of Bell Laboratories and later one year as a member of technical staff with Bell Laboratories and Agere Systems. Mike was gracious and selfless in helping organise the internship and allowing me to change my research topic. Bryan Ackland and Andrew Blanksby, formerly with Bell Laboratories, now with Agere Systems, organised and supervised the internship. Bryan is a vast wellspring of constructive criticism, with the ability to see the big picture and provide an eternally optimistic point of view.

My deepest thanks to Andrew Blanksby for his friendship, guidance, diligence and help with the work contained herein during the past two years at Bell Laboratories and later Agere Systems. During this time we worked closely together on the algorithms, architectures and implementation of the LDPC codes described in this thesis. He undertook all of the tedious back-end place-and-route of the chips which required many new custom CAD algorithms and tools due to the architectures unusual structure and wiring topology. Without his meticulous attention to detail, diligence and persistence the work presented here would not have been possible.

I would also like to thank Douglas Brinhaupt, formerly with Lucent Technologies Microelectronics Division and now with Agere Systems, for his experience, effort and help implementing the 43 Gbs⁻¹ encoder and decoder. His patience with the many last minute design changes and modifications I kept making cannot be understated. Both Andrew and Doug have displayed impressive tolerance for my over optimism and complete disregard of deadlines and schedules.

Lei-lei Song whose patience, help and understanding endured the pain of teaching me a few of the subtleties of communications and information theory. Lei-lei is a wealth of knowledge and experience.

To Eugene Scuteri, formerly with Lucent Technologies Microelectronics Division and now Agere Systems, thank you for allowing the publication of commercially valuable information to allow the completion of this thesis. Gene's trust in new and untested algorithms and architectures are the reason the fiber optic transceiver encoder and decoder were designed.

Kamran Azadet was extremely generous in allowing me a period of extended absence from Agere Systems to return to Australia and write this dissertation.

Errata

page 3, line 7 should read:

“... a code’s information rate...”

page 17, lines 9 & 11 and all subsequent references to:

“gaussian” should be “Gaussian”

page 26, Figure 2.3:

“check 2” should be “check 3”

page 34, first paragraph:

There is a discrepancy between Gallager’s notation, which is followed in Figure 3.1 and the explanatory text, and more recent coding notation. The variable k has been used to both denote the number of set elements in a row of the parity check matrix (Gallager), and to denote the number of uncoded data bits in a codeword. The variable substitution $k = d_c$ on line 3 should not be used, line 6 “... and k ones in every row.” should read “... and d_c ones in every row.”. Line 8 “... columns ik to $(i+1)k$.” should read “... columns id_c to $(i+1)d_c$.”

page 103, Figure 5.11 (a) is missing the x-axis label:

“iteration number”

page 121, third paragraph, line 6:

“Equation equation 6.8 ... ” should read “Equation 6.8 ... ”

page 133 & 134, Figures 6.9 & 6.10, captions should read:

“Packet error rates for a 1024-bit rate 1/2 code decoded using 64 iterations of a double precision floating point and 4-bit fixed point implementation of the sum-product algorithm.”

page 134, line 7:

“... Figure .” should read “... Figure 6.10.”

Figure 5.13 (b) should be added after Figure 5.13, page 110, to show the extrapolated bit error rate performance of the relative reliability weighted decoding algorithm, demonstrating an extrapolated coding gain of 8.4 dB at a BER of 10^{-15} , an increase of 2.2 dB over the (255,239) RS code.

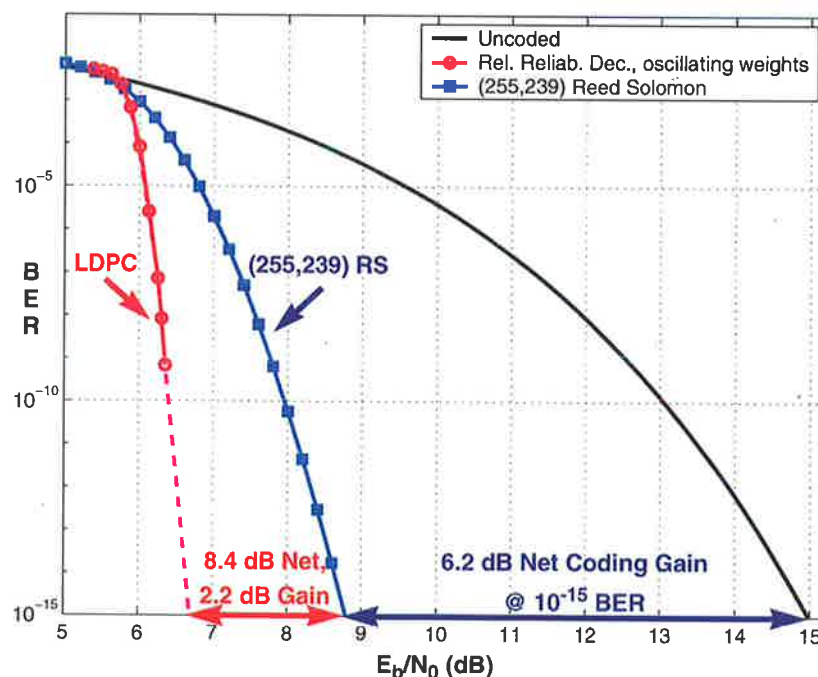


Figure 5.13 (b): Extrapolated performance of the optimised 32,640-bit rate 239/255 code decoded using 51 decoder iterations of the relative reliability weighted algorithm and oscillating received bit and parity check message weights and the (255,239) Reed Solomon code.

Contents

Chapter 1. Introduction	1
1.1 Hamming Codes	3
1.2 Linear Block Codes	5
1.3 Decoding Error Correcting Codes	7
1.3.1 Decoding Linear Block Codes	8
1.4 Turbo Codes	9
1.5 Low-Density Parity-Check Codes	0
1.6 Thesis Overview	11
1.7 Thesis Outline	13
 Chapter 2. Low-Density Parity-Check Codes	 17
2.1 Regular Low-Density Parity-Check Codes	18
2.2 Irregular Low-Density Parity-Check Codes	19
2.3 Code Weight of Low-Density Parity-Check Codes	23
2.4 Encoding Linear Block Codes	24
2.5 Graph Representation of Codes	25
2.6 Decoding Low-Density Parity-Check Codes	28
2.7 Parity Check Matrix Constraints	29
2.8 Generalised Low-Density Codes	30
2.9 Summary	31
 Chapter 3. Code Construction	 33
3.1 Gallager's Code Construction	34
3.2 Random Matrices	35
3.3 Structured Matrices	36
3.3.1 Permutation Matrix Code Construction	36
3.3.2 Convolutional Code Based Low-Density Parity-Check Codes	37
3.3.3 Upper or Lower Triangular Parity Check Matrices	39

3.3.4	Low-Density Generator Matrices	40
3.3.5	Geometric Fields and Steiner Systems	41
3.3.6	Burst Error Protection	42
3.4	Cycle Minimisation	43
3.5	A Minimum Cycle Cost Code Construction Algorithm.	44
3.5.1	Code Comparison	44
3.5.2	Metrics for Cycles Introduced by Edges of a Graph	45
3.5.3	A Minimum Cycle Cost Code Construction Algorithm	46
3.5.4	Initial Graph Edges	47
3.5.5	Variable Node Insertion Order	48
3.5.6	Termination of Edge Addition	49
3.5.7	Final Edge Insertion	49
3.5.8	Graph Refinement	50
3.5.9	Benefit of Metric Based Edge Insertion	50
3.6	A 32,640-Bit Rate 239/255 Regular LDPC Code	51
3.7	A 1024-Bit Rate 1/2 Irregular LDPC Code	55
3.8	Summary.	57

Chapter 4. Encoding Low-Density Parity-Check Codes 59

4.1	Constrained Parity Check Matrix Methods	60
4.2	Cascade Graph Codes.	60
4.3	Linear Simultaneous Equation Based Encoders.	61
4.4	Proposed Encoding Algorithm	64
4.5	Encoder Architectures	72
4.5.1	Encoder Architecture for Solving Simultaneous Equations	76
4.6	A 32,640-Bit Rate 239/255 Encoder	77
4.6.1	VHDL Implementation of the Encoder.	78
4.6.2	Encoder Synthesis	81
4.6.3	Encoder Layout	82
4.6.4	Encoder Timing Analysis and Design Rule Checking	82
4.7	Summary.	82

Chapter 5. Hard Decision Decoding **85**

5.1	Gallager's Algorithm A	86
5.2	Gallager's Algorithm B	90
5.3	Expander Graphs	92
5.3.1	The Binary Erasure Channel and Expander Graphs	92
5.3.2	The Binary Symmetric Channel and Expander Graph Decoding of High Rate Codes	93
5.4	Gallager's Algorithm, Expander Graphs and Erasures	93
5.5	Relative Reliability Weighted Decoding	94
5.5.1	Information Exchange	95
5.5.2	RRWD Algorithm for a 32,640-Bit Rate-239/255 LDPC Code	99
5.5.3	Mitigating the Effect of Graph Cycles	101
5.5.4	Summary of the Relative Reliability Weighted Decoding Algorithm	106
5.5.5	Performance Comparison of 32,640-Bit Rate 239/255 Codes with Relative Reliability Weighted Decoding.	108
5.6	Summary	109

Chapter 6. Soft Decision Decoding **111**

6.1	Gallager's Probabilistic Soft Decision Decoding Algorithm	112
6.2	The Sum-Product Algorithm	117
6.3	Implementation of Soft Decision Decoders	120
6.3.1	The Min-Sum Algorithm	122
6.4	Implementation of a 1024-Bit Rate 1/2 Soft Decision Decoder	125
6.4.1	Performance of a 1024-Bit Rate 1/2 Code Decoded Using a Min-Sum Decoder	126
6.4.2	Performance of a 1024-Bit Rate 1/2 Sum-Product Soft Decision Decoder with 4-Bit Messages	127
6.4.3	Graph Cycles and Finite Precision	133
6.4.4	3rd Generation Wireless 1024-Bit Rate 1/2 Turbo Code	135

6.4.5 Arithmetic Operations Per Bit Per Decoder Iteration	136
6.5 Summary.	138

Chapter 7. Decoder Architectures **139**

7.1 A Decoder Architecture for Wireless Communications.	140
7.2 A Decoder Architecture for Magnetic Recording Channels	141
7.3 General Memory Based Message Exchange Architectures	142
7.4 Parallel Decoders	146
7.5 Message Switching Activity.	149
7.6 A 1024-Bit Rate 1/2 1Gb/s Soft Decision Decoder	150
7.6.1 Parallel Decoder Routing Congestion.	156
7.6.2 Fabricated Chip	157
7.6.3 Measured Power Dissipation.	158
7.7 A 32640-Bit Rate 239/255 43Gb/s Hard Decision Decoder	162
7.8 Event Driven Decoders.	167
7.9 Summary.	168

Chapter 8. Conclusion **171**

8.1 Thesis Contributions.	174
8.2 Further Work	175

Patents **177**

Publications **178**

Bibliography **179**

Chapter 1

Introduction

Continuous advances in very large scale integration (VLSI) technology enable the implementation of increasingly more powerful and complex methods of improving communication reliability. Error correcting codes are a crucial part of modern communications systems where they are used to detect and correct errors introduced during transmission [7]. Low-density parity-check (LDPC) codes were discovered in the early 1960's and have since largely been forgotten due to the inherent complexity of the associated iterative decoding algorithms. With current VLSI integration densities the implementation of low-density parity-check codes has become feasible. Algorithms and architectures for implementing low-density parity-check codes to achieve reliable communication of digital data over an unreliable channel are the subject of this thesis.

Forward error correction (FEC) is the process of encoding blocks of k -bits in a data stream into n -bit codewords of the chosen code. The codewords of an error correcting code are of equal or longer length than the data word they represent,

$$n \geq k \quad (1.1)$$

The rate of information transmission when sending coded data is given by

$$r = \frac{k}{n} \quad (1.2)$$

and is often referred to as the code rate [7].

In many applications a substantial portion of the baseband signal processing is dedicated to the forward error correction encoding and, particularly, decoding of data signals. The coding gain of a forward error correction scheme is the difference between the signal-to noise ratio (SNR) required to achieve a specified bit error rate (BER) at the output of a decoder compared to uncoded transmission. As system designers can trade coding gain for lower transmit power, longer reach and/or higher data throughput, there is an ongoing effort to incorporate increasingly more powerful coding techniques into communications systems [25].

A model of a communications system using forward error correction is shown in Figure 1.1. A vector of data, s , to be sent to the information destination is encoded into a codeword, x . When the codeword x is transmitted over the unreliable channel it is potentially corrupted by noise. The noise added to the codeword can be represented by the error vector, e , and when added to the codeword results in the received vector y . The decoder then estimates the most likely codeword transmitted given the received data. The codeword is then decoded into the data it represents, $\hat{s}$, which is sent to the information destination.

In the late 1940's Shannon studied error correcting codes, investigating and deriving what is now the basis of modern information theory [15]. All forward error correction schemes obey Shannon's channel coding theorem, which states that reliable communication can be achieved provided the information rate does not exceed the capacity of the channel and the code has a sufficiently large block length [56], see [15] for a detailed explanation.

Although Shannon's theory provides a proof that good codes exist and a coding performance limit which codes can be measured against, it does not show how to find good

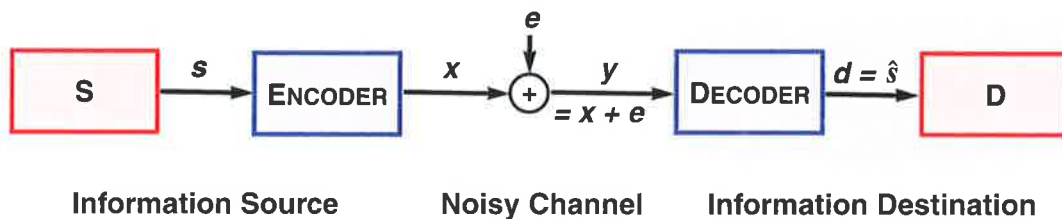


Figure 1.1: Data transmission over an unreliable, noisy channel with forward error correction.

codes that enable reliable information transmission at rates close to the channel capacity. Until recently no capacity achieving codes were known for any channel type and it had been conjectured that achieving capacity required codes with an infinite block length and an infinite complexity.

Recently low-density parity-check codes have been shown to be capacity approaching codes for the binary erasure channel (BEC) [52, 35]. As the difference between a codes information rate and the channel capacity tends to zero the block length of the code required to achieve reliable communication tends to infinity. Although the block length of the code increases the code does not become more complex and can be decoded using a simple iterative decoder.

Linear block codes are a large class of forward error correcting code. Both Hamming codes and Low-density parity check codes are subsets of linear block codes. In the following section Hamming codes will be introduced. Hamming codes are one of the simplest subsets of linear block codes. Following the review of the properties of Hamming codes linear block codes will be examined, before introducing low-density parity-check codes.

1.1 Hamming Codes

Hamming codes are a class of simple error correcting codes [7]. One Hamming code takes groups of four bits and appends three parity check bits to each group. If the four information bits are denoted s_0, s_1, s_2, s_3 and parity bits are denoted p_0, p_1, p_2 then the parity bits added to the information bits are:

$$p_0 = s_0 \oplus s_1 \oplus s_3 \quad (1.3)$$

$$p_1 = s_0 \oplus s_2 \oplus s_3 \quad (1.4)$$

$$p_2 = s_1 \oplus s_2 \oplus s_3 \quad (1.5)$$

The original information to be protected by the code is sent unchanged, except for the appended parity bits. A code which encodes data by appending parity bits to the unchanged original data bits is called a systematic code.

The structure of this $(7,4)$ Hamming code, where the numbers (n,k) denote the coded and uncoded block lengths respectively, can be illustrated as shown in Figure 1.2 [2]. All valid codewords of the code have an even number of ones in each circle, that is each circle must have an even parity.

The code can correct any single incorrect bit in any codeword. The possible cases, and incorrect bit are:

- A single circle has odd parity: The parity bit in this circle is incorrect and requires inversion.
- Two circles have odd parity: The information bit in the intersection of the two circles is the bit in error and requires inversion.
- All of the circles have odd parity: The information bit, s_3 , in the intersection of the three circles is incorrect and requires inversion.

Listing all of the $2^k = 2^4$ valid codewords for the $(7,4)$ Hamming code as:

$$x = \{s_0, s_1, s_2, s_3, p_0, p_1, p_2\} \quad (1.6)$$

gives the set of codewords:

$$C = \begin{array}{cccc} 0000000 & 0100101 & 1000110 & 1100011 \\ 0001111 & 0101010 & 1001001 & 1101100 \\ 0010011 & 0110110 & 1010010 & 1110000 \\ 0011110 & 0111001 & 1011010 & 1111111 \end{array} \quad (1.7)$$

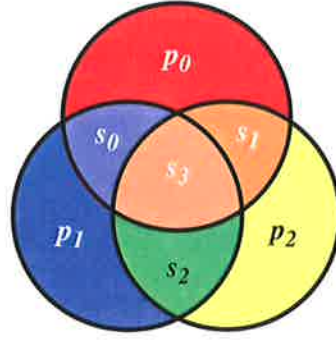


Figure 1.2: Structure of the (7,4) Hamming code, with four systematic data bits, s_0, s_1, s_2, s_3 and three parity bits, p_0, p_1 and p_2 .

From the set of codewords it can be seen that all pairs of codewords differ in at least three positions. The minimum difference between any pair of codewords of a code is called the *Hamming weight*, *minimum distance* or just *weight* of the code [40, 7]. The minimum distance of all Hamming codes is three [30].

1.2 Linear Block Codes

Linear block codes are denoted (n,k) or (n,k,d) codes, where d is the minimum distance of the code. Hamming codes are a subset of linear block codes where the code is constrained such that $H_r = (n,k,d) = (2^r-1, 2^r-1-r, d=3)$. The Hamming code in Section 1.1 is the (7,4) or (7,4,3) Hamming code. All linear block codes can be defined by a parity check matrix, $\mathbf{H}$, or a generator matrix, $\mathbf{G}$. A linear block code contains the set of all codewords, $\mathbf{x}$, spanning the null space of the parity check matrix, $\mathbf{H}$:

$$\mathbf{H} \cdot \mathbf{x}^T = 0, \quad \forall \mathbf{x} \in \mathbf{C} \quad (1.8)$$

where $\mathbf{C} \subset \mathbb{Z}_2^n$ is the set of all n -bit binary codewords belonging to the code.

A parity check matrix for the (7,4,3) Hamming code is:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = [\mathbf{P} \mid \mathbf{I}_3] \quad (1.9)$$

where $\mathbf{P}$ is the parity matrix and $\mathbf{I}_3$ is the 3×3 identity matrix. All matrices spanning the same row space as $\mathbf{H}$ are also valid parity check matrices for the code.

The generator matrix of a block code is used to encode the k information bits, $u = \{s_0, s_1, s_2, \dots, s_{k-1}\}$ into an n -bit codeword with a matrix multiplication. For a parity check matrix of the form:

$$\mathbf{H} = [\mathbf{P} \mid \mathbf{I}_{n-k}] \quad (1.10)$$

the generator matrix is:

$$\mathbf{G} = [\mathbf{I}_k \mid -\mathbf{P}^T] \quad (1.11)$$

Therefore, a generator matrix for the (7,4,3) Hamming code is:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1.12)$$

The generator forms codewords, $x = \{s_0, s_1, s_2, s_3, \dots, s_{k-1}, p_0, p_1, \dots, p_{n-k-1}\}$, from the systematic data bits $u = \{s_0, s_1, s_2, \dots, s_{k-1}\}$ via matrix multiplication:

$$x = u\mathbf{G} \quad (1.13)$$

Codes with an identity matrix as a sub-matrix of the generator matrix, as in equation

(1.11), are systematic codes.

Since the parity check matrix spans the null space of the generator matrix they are dual matrices and

$$\mathbf{H}\mathbf{G}^T = 0 \quad \text{and} \quad \mathbf{G}\mathbf{H}^T = 0 \quad (1.14)$$

Although linear block codes can be constructed over higher order fields, only binary linear block codes will be considered here.

1.3 Decoding Error Correcting Codes

For the majority of practical codes it is not possible to construct simple diagrams such as Figure 1.2 and use an associated set of simple rules for decoding. The task of decoding error correcting codes can be stated as finding the most likely transmitted codeword given the possibly corrupted received data. For each block of received data samples, $\mathbf{y}$, the decoder determines the most likely transmitted codeword, $\hat{\mathbf{x}}$, given the received channel samples.

That is the decoder finds:

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in \mathbf{C}}{\max} P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{x} \in \mathbf{C}}{\max} P(\mathbf{x}|\mathbf{y}) \quad (1.15)$$

Where $\mathbf{C}$ is the set of all valid codewords of the code.

For codes with large block lengths it is often not possible to implement the maximum likelihood decoder specified by equation (1.15). In this case an approximate probabilistic decoding algorithm is used.

1.3.1 Decoding Linear Block Codes

Consider a linear block code defined by a generator matrix, $\mathbf{G}$, and corresponding parity check matrix, $\mathbf{H}$. An n -bit codeword, x , is transmitted over a binary symmetric channel and a vector, y , is received where:

$$y = x + e \quad (1.16)$$

The received vector is corrupted by noise if the error vector, e , contains any non zero elements. The decoder performs a matrix multiplication of the received data vector with the parity check matrix to obtain the $(n-k)$ -bit syndrome vector, s :

$$s = \mathbf{H}y^T = \mathbf{H}(x + e)^T = \mathbf{H}x^T + \mathbf{H}e^T = \mathbf{H}e^T \quad (1.17)$$

using the identity $\mathbf{H} \cdot x^T = 0$ from equation (1.8).

If the syndrome is the all zero vector a valid codeword has been received and the error vector, e , is either all zero, or is itself a valid codeword. The most probable case is that the error vector is all zero and the correct codeword has been received.

If the syndrome, s , contains non zero components then an error vector, e , that results in the same syndrome must be found. This can be done by finding the solution to the $(n-k)$ simultaneous equations specified by equation (1.17). There is not a unique solution to the simultaneous equations specified by equation (1.17), instead 2^k solutions exist [40].

For a binary symmetric channel the most probable solution is the error event with the least errors, or the error vector satisfying equation (1.17) with the least set elements [40]. Maximum likelihood decoding assumes that the lowest weight error vector with the same syndrome as the received data is the error event which has occurred. The error vector is then subtracted from the received data to determine the maximum likelihood transmitted codeword, $\hat{x}$:

$$\hat{x} = y - e \quad (1.18)$$

1.4 Turbo Codes

Recently a new family of codes, known as turbo codes, has been developed which approach the Shannon limit [4]. Turbo codes are based on parallel concatenated, interleaved, convolutional codes and provide excellent coding gain at low code rates¹ [4]. Lin and Costello give a detailed description of convolutional codes, including algorithms and architectures to decode them, in [30]. A related family of codes are block turbo codes, or Tanner codes, which are based on the product of block codes. Block turbo codes yielding very high coding gain at high code rates have been widely investigated following the discovery of the original convolutional turbo codes [63, 11, 50].

Turbo codes achieve very high coding gain and are suitable for implementation. They are therefore a very important class of error correcting code. Another important contribution to coding theory of turbo codes is the demonstration of the significant performance improvement obtained through the use of an iterative decoding process. Each block of data is decoded multiple times, with each iteration further refining the current estimate of the transmitted data. Although Gallager [21], later Tanner [63] and other researchers had previously investigated iterative decoding it was largely ignored due to the inherent complexity of implementing an iterative decoder. This was until the discovery of turbo codes.

Significant implementation challenges still exist for turbo and block turbo codes due to the iterative nature of their respective decoding algorithms. Both turbo and block turbo codes are decoded using sophisticated algorithms iterated multiple times across the data block. Each pass through the data block requires the fetching, computation, and storage of large amounts of state information. Performing multiple iterations to achieve high coding gain necessarily reduces the throughput and increases the power dissipation of the decoder implementation.

1. The rate of a code is defined as the ratio of the number of information bits to the sum of the information and parity bits, see equation (1.2). A low rate code has a large redundant overhead while a high rate code has a small overhead.

1.5 Low-Density Parity-Check Codes

The phenomenal increase in achievable coding gain that the original parallel concatenated convolutional codes and iterative decoding of turbo codes offered triggered interest in other iteratively decoded codes. As a result of this the original work on low-density parity-check (LDPC) codes by Gallager in his paper of 1962 and book in 1963 were rediscovered [21, 22]. Low-density parity-check codes are sometimes called Gallager codes in recognition of his discovery of this class of code.

Unfortunately, at the time of Gallager's original work, the implementation of these iteratively decoded codes was impractical and they were largely forgotten. Only a few papers were published regarding low-density parity-check codes in the period between their discovery and the late 1990's. These were the work of Tanner [63], Margulis [41], Zybalov and Pinsker [72].

Tanner extended the idea of long block length codes based on smaller sub-codes in 1981 [63]. The use of a bipartite graph to represent a code was also introduced by Tanner in this publication. The bipartite graph representation of a code is an important concept, used in the explanation and implementation of decoders for low-density parity-check codes.

The rediscovery of low-density parity-check codes came with the publication of Wiberg's PhD work [67, 68], MacKay and Neal's paper [36], Luby *et. al.*'s papers [32, 33] and Spielman and Sipser's work on expander graphs [57, 61].

Low-density parity-check codes are linear block codes. Thus by equation (1.8) the set of all binary n -bit codewords, $x \in C$, span the null space of a parity check matrix, H , and $H \cdot x^T = 0$, $\forall x \in C$.

Low-density parity-check codes are constructed with a very long block length. The parity check matrix for LDPC codes is a sparse binary matrix, where the set row and column elements are chosen to satisfy a desired row and column weight profile². The

2. Where the column or row weight refers to the number of non zero elements in the column or row.

number of set elements in the parity check matrix of LDPC codes is $O(n)$, while the number of elements in the parity check matrix for random linear block codes is $O(n^2)$.

Due to the very long block length of LDPC codes it is not feasible to solve equation (1.17) and implement a maximum likelihood decoder. The storage or calculation of the minimum weight error vector which results in the syndrome of a received corrupted data vector is prohibitively complex. However, Gallager proposed three simple iterative probabilistic decoders which are well suited to implementation. It will be shown that the performance of Gallager's two hard decision decoding algorithms is poor for high rate codes, for example codes with a rate greater than or equal to $3/4$. Therefore a new hard decision decoding algorithm will be derived. Gallager's soft decision decoding algorithm is a special case of the more general sum-product decoder.

1.6 Thesis Overview

This thesis examines algorithms and implementation architectures for applying low-density parity-check codes to both wireless and fiber optic data transmission. The two applications require very different types of codes. While wireless systems require short block length and low rate codes optical fiber systems require high rate and long block length codes.

An algorithm to construct LDPC codes which perform significantly better than random LDPC codes is derived and used to find two good codes, one for a wireless application and one for a fiber optic transceiver. Existing code construction techniques often result in poor performance and codes with error floors, particularly for high rate codes. The codes designed with the proposed algorithm do not exhibit an error floor at the bit error rates which have been simulated and measured. For the wireless application a 1024-bit rate $1/2$ irregular code is designed. A 32,640 bit rate $239/255$ regular code is designed for the fiber optic application.

Existing algorithms for encoding low-density parity-check codes will be shown to result in an encoder for the fiber optic transceiver which is too complex to be implemented. Therefore a new encoding algorithm is proposed which has been derived specifically for

high rate regular LDPC codes. An architecture for implementing the algorithm is demonstrated through the implementation of an encoder for the 32,640-bit rate 239/255 code in Agere Systems' 1.5V 0.16 μ m CMOS process with 7 layers of metal.

In the case of a wireless communications system low rate short block length codes are considered with soft decision decoding. The coding gain of a 1024-bit rate 1/2 LDPC code is found to be comparable to 1024-bit rate 1/2 turbo codes. A low power and high throughput soft decision decoding algorithm and architecture is developed and demonstrated with the implementation of a 1024-bit rate 1/2 soft decision decoder. The decoder was implemented in Agere Systems' 1.5V 0.16 μ m CMOS process with five levels of metal and dissipates 630 mW while decoding 1Gb of coded data per second.

Due to the poor performance of existing hard decision decoding algorithms when decoding high rate codes a new hard decision decoding algorithm is proposed. Using the proposed relative reliability weighted decoding algorithm to decode the 32,640-bit rate 239/255 code results in a coding gain of greater than 8dB compared to uncoded transmission at a bit error rate of 10^{-15} , representing an improvement of 2dB over the 6dB coding gain of the commonly used (255,239) Reed Solomon code. An architecture for implementing a decoder for this code with a throughput of 43Gbs⁻¹ is described. The code is implemented with the same frame structure, line rate and FEC overhead as the SONET OC-768 standard, often abbreviated 40G, for fiber optic systems. The encoder and decoder pair have been implemented in Agere Systems' 1.5V 0.16 μ m CMOS process with seven levels of metal. This codec operates at full duplex rate as a proprietary FEC replacement for the (255,239) Reed Solomon code.

1.7 Thesis Outline

This thesis is divided into eight chapters whose contents can be summarised as follows:

Chapter 1: Introduction

This chapter.

Chapter 2: Low-Density Parity-Check Codes

Chapter 2 provides an overview of low-density parity-check codes. Simple methods for encoding and decoding LDPC codes are introduced and the constraints on code construction due to the decoding algorithms are examined.

Chapter 3: Code Construction

The constraints on code construction identified in Chapter 2 are used in Chapter 3 to compare construction methods for LDPC codes. Gallager's method of code construction using permutation matrices is reviewed, followed by random and structured matrix construction techniques proposed in the literature. Since the aim when implementing a forward error correction scheme is to obtain the best possible performance, a code construction method is proposed to find codes with good performance.

The performance of the iterative decoding algorithms used to decode LDPC codes is degraded by short cycles in the bipartite graph representation of a code. Hence minimising the number of short cycles in the bipartite graph representing a code improves the performance of the code. The proposed code construction technique is therefore based on minimising a cost metric which measures cycles in the graph.

Two codes are constructed using the metric minimisation technique, a 1024-bit rate $1/2$ code and a 32,640-bit rate $239/255$ code. The benefit of the code construction is demonstrated through comparison of the optimised 32,640-bit code with random and semi-random 32,640-bit codes.

Chapter 4: Encoding Low-Density Parity-Check Codes

Algorithms and architectures for encoding LDPC codes are examined in Chapter 4.

Existing algorithms for encoding linear block codes with long block lengths are reviewed. Most techniques constrain the parity check matrix to simplify the encoding process. Richardson and Urbanke propose a method for encoding random parity check codes that is very good for irregular codes [53]. The 32,640-bit code from Chapter 3 is a regular LDPC code and the encoding algorithm is not efficient for implementing an encoder for this code. A new encoding algorithm suitable for regular LDPC codes is therefore derived. An architecture for implementing the low complexity encoder is developed and demonstrated through the implementation of an encoder for the 32,640-bit rate 239/255 code with a throughput of 43Gbs^{-1} .

Chapter 5: Hard Decision Decoding

Chapter 5 examines hard decision decoding algorithms for LDPC codes. The two hard decision decoding algorithms proposed by Gallager are reviewed, followed by expander graph based decoding algorithms. The performance of these decoding algorithms when decoding the 32,640 bit code is worse than the (255,239) Reed Solomon code. Therefore a new hard decision decoding algorithm is derived specifically for high rate codes. The algorithm results in a 2dB improvement in coding gain compared to the Reed Solomon code and an improvement of 3.6dB over Gallager's Algorithm B at a bit error rate of 10^{-15} .

Chapter 6: Soft Decision Decoding

Soft decision decoding of LDPC codes is considered in Chapter 6. Gallager's soft decision algorithm and the sum-product algorithm are reviewed. The min-sum algorithm, a simplification of the sum-product algorithm is also examined. Although the min-sum algorithm removes some complex logarithms, exponentiation and hyperbolic tangent functions from the decoder it is shown that it does not reduce the number of addition or subtraction operations required by the decoder. It is further shown that these complex functions are easily implemented due to the small number of bits required to represent the quantities to be operated upon. The implementation of fixed point soft decision decoders is examined through the derivation of a decoding algorithm for the 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages between the functional nodes of the decoder. The performance of the decoder when performing 64 decoding iterations is only 0.2dB worse than a sum-product decoder implemented using double precision floating point accuracy

and performing 1000 decoding iterations.

Chapter 7: Decoder Architectures

Architectures for implementing LDPC decoders are considered in Chapter 7. Decoders using memory to exchange messages and a parallel architecture are examined. Two parallel decoder implementations are presented, one is a soft decision decoder for the 1024-bit rate 1/2 code with a throughput of 1 Gbs⁻¹ while performing 64 decoding iterations and one is a hard decision decoder for the 32,640-bit rate 239/255 code with a throughput of 43 Gbs⁻¹ while performing 51 decoding iterations. Measured results for the fabricated 1024-bit soft decision decoder are also presented.

Chapter 8: Conclusion

Chapter 8 concludes this thesis with a summary of contributions and proposals for further work to be considered.

Chapter 2

Low-Density Parity-Check Codes

This chapter introduces the fundamental properties of low-density parity-check codes. The descriptions and results contained here are existing prior work used to introduce low-density parity-check codes. After the introduction provided by this chapter a more detailed analysis of particular problems will be undertaken in later chapters. Algorithms for encoding and iterative decoding of low-density parity-check codes are introduced. Constraints on the construction of low-density parity-check codes due to the iterative decoding algorithms will then be examined.

All of the work in this thesis will assume either a memory-less additive white gaussian noise (AWGN) channel or a binary symmetric channel (BSC). It will be assumed throughout that the channel noise is independent and identically distributed (iid) and is a zero-mean gaussian sequence with noise power level N_0 . Independence of the noise is defined as all channel samples being uncorrelated and the expected value of the correlation of any sequence of noise samples from the channel with any other distinct sequence of noise samples from the channel being zero. It is further assumed that the noise is identically distributed and all noise events arise from a random process which has the same variance and zero mean. Unless otherwise noted all of the performance results for the codes examined will be given relative to the signal-to-noise ratio of the energy per information bit, E_b , to the noise power level, N_0 , in decibels, that is E_b/N_0 in dB.

While not considered in this thesis, low-density parity-check codes have been investigated as forward error correction for other channel types, in particular, channels with memory. The channel types studied have included partial response channels associated with magnetic storage medium [42, 43, 69] and Rayleigh fading channels associated with

wireless transmission [59, 60]. Low-density parity-check (LDPC) codes have also been concatenated with a trellis code to achieve very good performance over a channel with memory [66]. Although the results obtained in this thesis have been derived for additive white gaussian noise and binary symmetric channels they can easily be generalised to other channel types.

2.1 Regular Low-Density Parity-Check Codes

A regular LDPC code has a parity check matrix in which all columns have the same number of set elements and all rows also have the same number of set elements. Gallager's original work on LDPC codes considered only regular codes [21]. A (d_v, d_c) regular LDPC code has d_v set elements per column and d_c elements per row of its parity check matrix. The general structure of the parity check matrix, $\mathbf{H}$, is illustrated in Figure 2.1. Each row of $\mathbf{H}$ corresponds to a parity check and a set element (i, j) indicates that data symbol j participates in parity check i .

A code specified by an $m \times n$ parity check matrix implies that in a block of n -bits or symbols, there are m redundant parity bits and the code rate, r , is given by:

$$r = \frac{(n - m)}{n} = \frac{k}{n} = 1 - (d_v / d_c) \quad (2.1)$$

Equation 2.1 assumes the parity check matrix is of full rank. If the parity check matrix is not of full rank and contains linearly dependent rows the code rate is lower than the rate determined using equation (2.1).

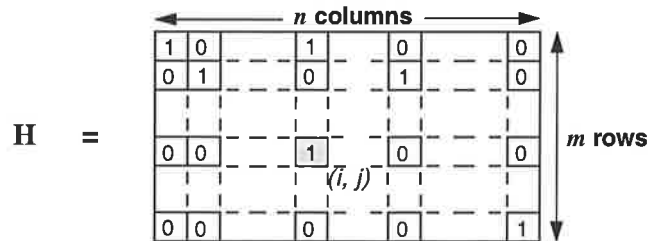


Figure 2.1: General structure of a low-density parity-check matrix.

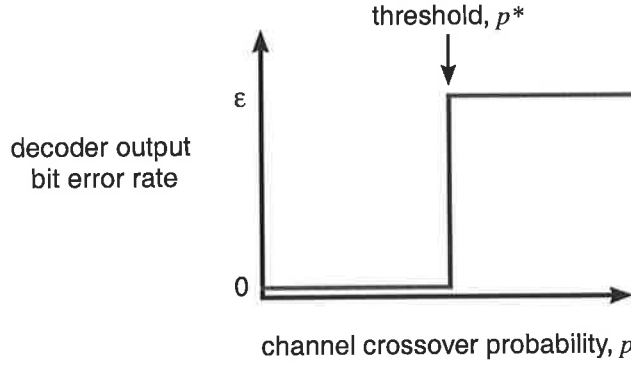


Figure 2.2: Probability of error versus channel crossover probability for an infinite block length LDPC code.

Gallager used the values d_v and d_c to calculate channel thresholds for regular LDPC codes. If the channel crossover probability, p , or standard deviation, σ , is greater than the threshold he showed the probability of error at the output of a decoder remains at a fixed value. If the channel parameter is less than the threshold the probability of error can be made arbitrarily small by selecting a sufficiently long block length code. In the limit of the block length tending to infinity the probability of error versus channel parameter becomes a step function at the code threshold [22], as shown in Figure 2.2. A simplified explanation of the threshold for a code is the channel parameter at which the decoder changes from not working, and being unable to correct errors, to working and able to correct errors to any desired output error rate.

2.2 Irregular Low-Density Parity-Check Codes

Although Gallager proposed the use of parity check matrices with all rows and columns of the same weight¹, it is possible to construct codes with varying numbers of set elements per column and row. Codes constructed in this way were first investigated by Luby *et al.* [33]. Simulation results presented by Luby *et al.* for regular and irregular

1. The ‘weight’ of a column or row refers here to the number of non zero entries in the row or column.

LDPC codes of the same rate and block size showed irregular codes have a higher coding gain [33]. It is possible to design irregular LDPC codes of almost any rate and block length.

Irregular codes can be defined by two vectors, (λ_i) and (ρ_i) , where λ_i and ρ_i are the fraction of edges belonging to columns and rows of weight i respectively [34]. Column weights in the parity check matrix are from the set $\{2, \dots, d_l\}$, where d_l is the maximum column weight in the code and row weights are from the set $\{2, \dots, d_r\}$, where d_r is the maximum row weight.

The set of values (λ_i) and (ρ_i) are generator functions and are constrained such that:

$$\lambda_i \geq 0, \quad \forall i \quad (2.2)$$

$$\rho_i \geq 0, \quad \forall i \quad (2.3)$$

$$\sum_{i=2}^{d_l} \lambda_i = 1 \quad (2.4)$$

$$\sum_{i=2}^{d_r} \rho_i = 1 \quad (2.5)$$

The average column and row weights of the code are given by:

$$d_v = \frac{1}{\left(\sum_i \lambda_i / i \right)} \quad (2.6)$$

$$d_c = \frac{1}{\left(\sum_i \rho_i / i \right)} \quad (2.7)$$

Another two generator functions are also defined and used in the derivation of good weight profiles, or degree sequences, for irregular codes and the channel threshold of the code [54]. The generator functions introduce a continuous real valued variable, x , which can be used in deriving properties of the code. The functions are:

$$\lambda(x) = \sum_{i=1}^{d_l} \lambda_i \cdot x^{i-1} \quad (2.8)$$

$$\rho(x) = \sum_{i=1}^{d_r} \rho_i \cdot x^{i-1} \quad (2.9)$$

The average column and row weight of the code can be found by integrating the generator functions, equation (2.8) and equation (2.9), from zero to one. The code rate of an irregular LDPC code is given by:

$$r = 1 - d_v / d_c = 1 - \frac{\int_0^1 \rho(x) dx}{\int_0^1 \lambda(x) dx} \quad (2.10)$$

The generator functions are used to derive an equation that expresses the probability of error at any iteration as a function of the probability of error at the previous iteration and the error probability of the received data. The probabilities used in the recursive bit error rate update as a function of the iteration number only remain uncorrelated during the iterative update when the code has an infinite block length². The updated probabilities are either a new crossover probability at the l^{th} iteration, p_l , or a probability density function, $g_l(x)$, for hard and soft decision decoding algorithms respectively. The changing of the probability density distribution during decoding has been named density evolution by Richardson, Shokrollahi and Urbanke [54]. This has also been used by Chung [11, 13]. The method of finding a good weight profile for a code is to select a value of $\rho(x)$ and find the

2. The requirement of an infinite graph will be explained in Chapter 5 where decoding algorithms are examined in detail.

set of values, (λ_i) , which maximises σ^* or p^* such that the probability update function, p_i or $g_i(x)$, is a strictly decreasing function with $p \in (0, p^*]$ or $\sigma \in (0, \sigma^*]$. This finds the code with the largest initial error probability that will, with high probability, decode correctly [51, 3].

The performance improvement of irregular LDPC codes when decoded with a soft decision decoder was proven by Richardson, Shokrollahi and Urbanke [51]. Irregular codes have also been shown to have thresholds very close to the channel capacity for a binary input AWGN channel. The theoretical threshold for a random, irregular, one million bit rate 1/2 code with the derived column and row weight profile in [52] was only 0.06 dB from the Shannon limit. A weight profile for a ten million bit rate 1/2 code with a threshold only 0.0045 dB from the Shannon limit has also been published by Chung, Forney, Richardson and Urbanke [14]. The same papers simulated codes designed with these block sizes and the derived column and row weight profiles and have achieved results 0.13 dB and 0.04 dB from channel capacity at a bit error rate (BER) of 10^{-6} respectively.

An important result from the derivation of good weight profiles is the theorem of concentration of row weights, ρ , derived by Richardson and Urbanke [52].

Theorem 2.1: *The threshold of a code with rate greater than 0.4 can be maximised by using row weights concentrated at a single value, $i = \rho_{av}$, if the average row weight is an integer, or consecutive row weights $i = \lfloor \rho_{av} \rfloor$ and $i + 1 = \lceil \rho_{av} \rceil$ if the average weight is not an integer. (Proof: [52]).*

Although irregular codes are optimal with soft decision decoding, for hard decision decoding Bazzi, Richardson and Urbanke have proven that regular codes are optimal for codes of rate greater than 0.4 and hard decision decoding using Gallager's first proposed algorithm, often called Gallager's Algorithm A. The proof derives the probability of error for each bit in the code as a function of the iteration number, column and row weights and the maximum initial error probability that can be corrected. The maximum initial error rate that can be corrected was shown to be maximised by using a code with all columns of the same weight and rows of the same weight.

2.3 Code Weight of Low-Density Parity-Check Codes

The number of places in which two valid codewords of a code differ, often referred to as the *distance*, *weight* or *Hamming distance*, of the code [5], is important in determining the maximum number of errors that can be corrected by a maximum likelihood decoder. In general a decoder for a code with minimum distance d can correct a maximum of $\lfloor (d - 1) / 2 \rfloor$ errors [40]. The Hamming code from Section 1.1 has a minimum distance of three and can correct any $\lfloor (3 - 1) / 2 \rfloor = 1$ error in a codeword.

Since the all zero codeword is always a valid codeword for linear block codes, the distance of the code can be determined by finding the number of non zero entries in the codeword with the minimum number of set elements [23, 40]. The distance of a linear block code can also be determined using a parity check matrix for the code.

Theorem 2.2: [30, 23, 40] *Let $\mathbf{H}$ be a parity check matrix for a linear code $\mathbf{C}$. Then $\mathbf{C}$ has distance d if and only if any set of $d-1$ rows of $\mathbf{H}$ are linearly independent, and at least one set of d rows of $\mathbf{H}$ are linearly dependent.*

The distance of LDPC codes increases linearly with the block length [21, 52]. When the initial input error rate is below the threshold of a code the result is the probability of a decoding error decreases exponentially with the block length. It was also noted by Gallager that although this is the upper bound, experimental results show better improvement of the error correction capability as a function of the block length of a code [22]. It is noted here though that due to computational limits Gallager's experimental tests were on very short block lengths, and thus the bound may still apply for large block lengths.

The Gilbert-Varshamov bound provides an asymptotic bound on the ratio of the minimum distance to block length for all randomly chosen linear block codes as the block length tends to infinity. The bound can be used to compare the performance of linear block codes.

Theorem 2.3: [40] (*The Gilbert-Varshamov bound*) *There exists a linear binary code of length n , with at most k parity checks and minimum distance at least d , provided:*

$$\sum_{i=0}^{d-2} \binom{n-1}{i} < 2^k \quad (2.11)$$

2.4 Encoding Linear Block Codes

As introduced in Section 1.2, the general method of encoding a linear block code with a random parity check matrix is to use a generator matrix, $\mathbf{G}$, corresponding to the parity check matrix. The generator matrix is any matrix which spans the dual vector space of the parity check matrix, $\mathbf{H}$. If $\mathbf{H}$ is an $m \times n$ matrix, then $\mathbf{G}$ is an $n \times (n - m)$ matrix satisfying equation (1.14), i.e. $\mathbf{H}\mathbf{G}^T = \mathbf{0}$ or $\mathbf{G}\mathbf{H}^T = \mathbf{0}$.

Then by taking any $k = (n-m)$ element row vector of uncoded data, s , and multiplying it to the right by $\mathbf{G}$ yields a valid codeword, x , s.t.:

$$s\mathbf{G}^T = x \quad \Leftrightarrow \quad \mathbf{H}x^T = \mathbf{0} \quad (2.12)$$

A generator matrix for a parity check matrix may be found by performing Gaussian elimination on the parity check matrix to obtain a matrix $\mathbf{H}'$, spanning the same row space as $\mathbf{H}$ of the form:

$$\mathbf{H}' = \left[\mathbf{P} \mid \mathbf{I}_m \right] \quad (2.13)$$

Where $\mathbf{I}_m$ is an $m \times m$ identity matrix and $\mathbf{P}$ is an $m \times k$ parity matrix. Since only the row space spanned by $\mathbf{H}'$ is important, arbitrary row permutations during the Gaussian elimination are inconsequential and reduce the complexity of finding a suitable $\mathbf{H}'$ for large matrices.

The generator matrix is then given by:

$$\mathbf{G} = \left[\mathbf{I}_k \mid -\mathbf{P}^T \right] \quad (2.14)$$

An encoder for a low-density parity-check code based on a generator matrix is extremely complex for two reasons:

Firstly the parity check matrix, $\mathbf{H}$, is sparse. Therefore the gaussian elimination performed to get $\mathbf{H}'$ into the form given by equation (2.13) results in the sub matrix $\mathbf{P}$ being dense. Thus, the generator, $\mathbf{G}$, will also be dense. A dense generator requires a large number of exclusive-or (XOR) operations to implement the encoding matrix multiplication and will require a large amount of memory or dedicated gates to implement. This is a major problem limiting the utility of LDPC codes. Codes used in practical applications generally have very simple encoders which require very few gates to implement [30, 7].

The second problem is in performing Gaussian elimination to find the generator matrix $\mathbf{G}$. Gaussian reduction of a matrix is an $O(n^3)$ operation. Although the reduction only needs to be performed once it can take significant amounts of time for codes with very long block lengths.

However, it is possible to consider the encoding of linear block codes as the process of finding the solution to a set of simultaneous equations specified by the parity check matrix. The known variables in the equations are the systematic data bits and the unknown variables are the corresponding parity check bits. Finding the parity bits becomes a simple back substitution calculation if the parity check matrix is either an upper or lower triangular matrix. Although constraining the parity check matrix to be either upper or lower triangular simplifies the encoding of LDPC codes it degrades the performance of the code.

2.5 Graph Representation of Codes

Tanner introduced the use of bipartite graphs to represent codes in 1981 [63]. It is possible to represent any binary matrix as a bipartite graph, with one type of node corre-

spending to the columns of the matrix and another type corresponding to the rows. Every column of the parity check matrix is represented by a variable node. The variable nodes represent a data variable or symbol in the received block to be decoded. Similarly, every row of the matrix is represented by a check node. The check nodes represent a parity check constraint on the data block. Each non zero entry in the matrix is represented as an edge in the graph and connects a variable node to a check node.

The graph in Figure 2.3 represents a parity check matrix with a block length $n = 12$ which has $m = 6$ parity check constraints. The 6×12 parity check matrix can be represented by a bipartite graph with 12 variable nodes and 6 check nodes. A parity check node is only connected to the variable nodes representing bits participating in the parity check the node represents, or the columns that are non zero in the row of the matrix the node represents. Similarly, a variable node is only connected to the parity check nodes corresponding to the parity checks it is involved in, or the rows with non zero elements in the column of the matrix that the node represents. Hence, for every element $h_{i,j} = 1$ of $\mathbf{H}$ there is one edge in the graph connecting variable node j to check node i .

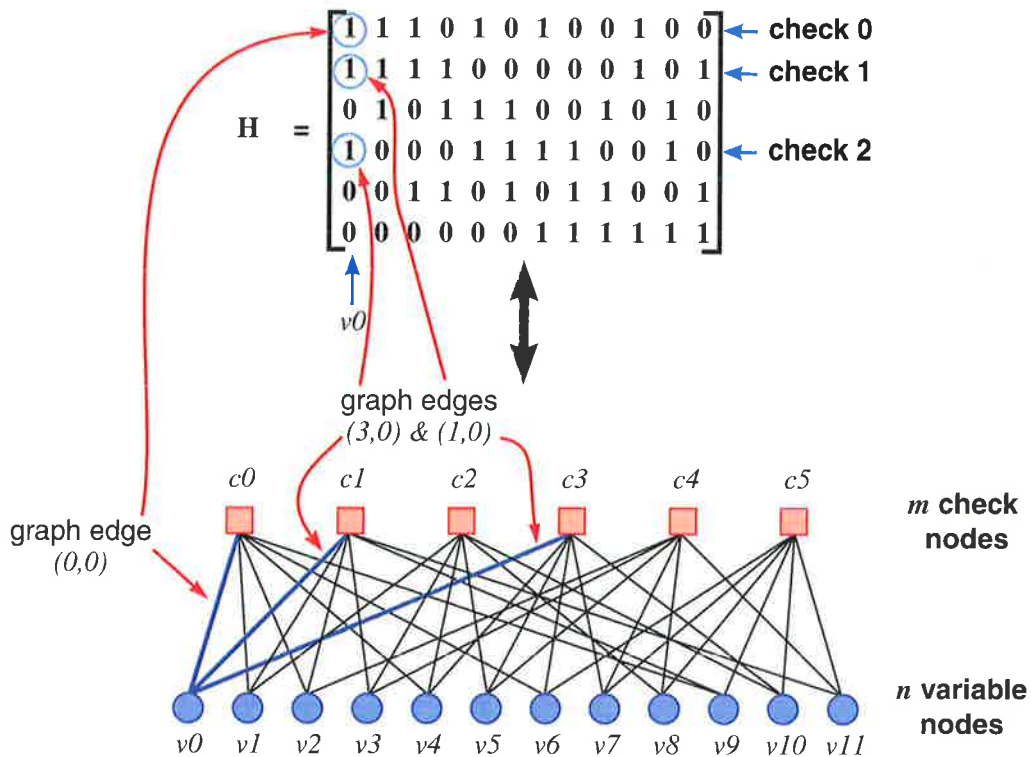
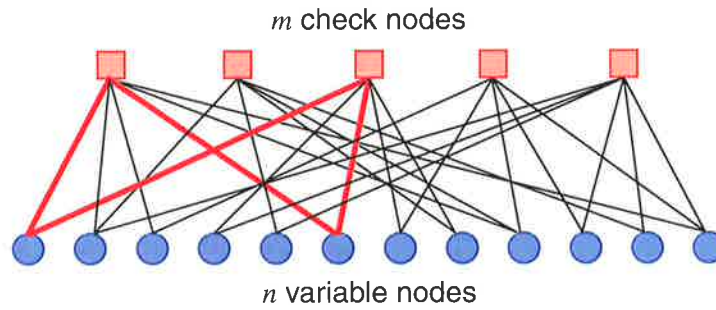
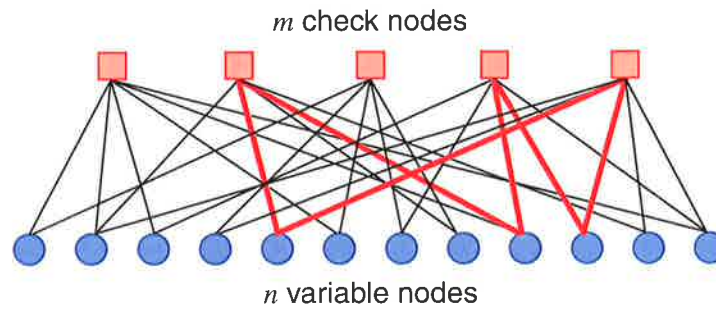


Figure 2.3: Bipartite graph representation of a 12-bit (3,6) regular LDPC code, or a (3,6,12) code and the corresponding parity check matrix.

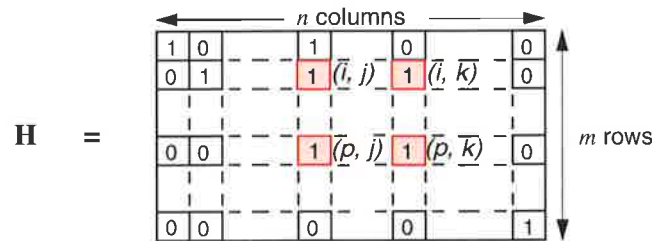
In Figure 2.4 short cycles in a graph are highlighted. Figure 2.4 part (a) shows a cycle containing four graph edges, a cycle of length four. Figure 2.4 part (b) shows a cycle of length six and Figure 2.4 part (c) shows the parity check matrix structure resulting in a length four cycle, where two columns contain two common elements. To prevent length four cycles in a code no column (or row) can have more than one non zero row (or column) in common with another column (or row). The length of the shortest cycle in a graph is referred to as the girth of the graph.



(a) a length 4 cycle



(b) a length 6 cycle



(c) matrix structure resulting in a length 4 cycle

Figure 2.4: Short cycles in a bipartite graph, of (a) length 4, (b) length 6 and (c) matrix structure resulting in a length 4 cycle.

2.6 Decoding Low-Density Parity-Check Codes

Low-density parity-check codes can be decoded using simple iterative decoding algorithms, best understood with reference to the graph representation of the code. When a block of data is received the value associated with each variable node in the block is stored at the node. Each variable node sends the value associated with it to all of the check nodes connected to it by graph edges. The check nodes then perform the parity checks that the code specifies and send the results back to the connected variable nodes.

At the variable nodes an update rule is applied using the received bits and the results of the parity checks. The update can simply be a vote for the value of the decoded bit, where an unsatisfied parity check is counted as a vote for the received value being incorrect, thus requiring inversion. The updated values are then sent back to the check nodes and the iterative decoding process continues.

Decoding continues until all of the parity checks are satisfied, indicating that a valid codeword has been decoded, or until a maximum number of decoder iterations is reached and a block error declared.

All of the iterative decoding algorithms, both hard decision and soft decision, for low-density parity-check codes can be considered as variations of this iterative message passing between the variable and check nodes. For this reason decoders for low-density parity-check codes are often referred to as message passing decoders. The differences between the various decoding algorithms are the update rules applied at the variable and check nodes, which determine the value of the messages in the next decoding iteration.

Decoding algorithms for LDPC codes can perform optimal variable and check node updates while all of the inputs in the update remain uncorrelated. Information in the decoder remains uncorrelated while the number of iterations is less than half the girth of the graph. Once the number of decoder iterations is greater than half the girth of the graph, information can travel around a cycle and contribute to the update of a node which has already been used in determining the message value arriving at the node. If the block length of the code is infinite the graph representing the code can have an infinite girth and the simple iterative decoder remains an optimal decoder for the code.

2.7 Parity Check Matrix Constraints

The rows of a low-density parity-check code are required to be linearly independent, thus the parity check matrix has full rank. If the parity check matrix does not have full rank the actual code rate will be lower than that indicated by equation (2.1) [22] and the redundant row(s) can be removed [53].

A further constraint on the parity check matrix will be imposed by the decoding algorithms presented later. The decoding algorithms require the column and row overlap of the matrix to be minimised. When representing the code as a graph this constraint corresponds to maximising the graphs girth or minimum cycle length. The decoding algorithms can be optimal for a number of iterations equal to less than half of the minimum girth of the graph. Constructing graphs with large girth requires very long block lengths. Higher rate codes also require longer block lengths than lower rate codes to achieve the same minimum graph girth. Irregular LDPC codes containing columns with many set elements result in highly connected graphs and therefore require longer block lengths to increase the girth of the graphs. Therefore both high rate and irregular codes require very long block lengths to improve the girth of the graph associated with the code.

Constructing a code with no column or row overlap greater than one element requires a minimum block length which will be a function of the code rate. The minimum block length required to construct a code as a function of code rate was studied by MacKay and Davey in terms of Steiner systems [37]. High rate codes require very long block lengths to prevent length four cycles and to increase the minimum length of graph cycles. The result is intuitive because a high rate code has relatively few parity checks across a large number of data bits. A high rate code therefore has a highly connected graph and minimising short cycles leads to considerably longer block lengths than lower rate codes with the same minimum girth.

Another motivation for increasing the block length of a low-density parity-check code is due to theorem 2.2, the distance of a linear block code is equal to the smallest number of rows in the parity check matrix which are linearly dependant. Increasing the minimum distance of a code will result in lowering the possibility of a decoder error and

improve the performance of the code. The code distance can be increased linearly as a function of the block length [22].

Determination of a good weight profile for an irregular code is the first step in finding a good code. A good parity check matrix meeting the row and column overlap and linear independence constraints must then be found. Both Richardson and Chung found very good irregular rate 1/2 code weight distributions using the optimisation technique described in Section 2.2. Specific matrices used in simulation were found by randomly adding edges from a list of available variable node sources and check node targets. A column overlap constraint was enforced such that no pair of weight two columns contained the same two rows.

2.8 Generalised Low-Density Codes

Tanner generalised low-density parity-check codes by using codes other than a simple $(k-1, k)$ parity check as the constituent code for each row of the parity check matrix [63]. The construction includes the product codes now referred to as block turbo codes. Generalised low-density (GLD) codes are sometimes called Tanner codes or Tanner graphs in recognition of his pioneering work in the area. These codes may be useful in reducing the complexity of implementing low-density parity-check codes, which will be discussed in Chapter 7. Generalised low-density codes were constructed by Boutros, Pothier and Zemor of rate 0.677 and block length 65,534-bits, from constituent (31,26) Hamming codes. The codes result in large coding gains which approach the channel capacity. In their paper they claim the code achieves zero error probability at 1.8dB, only 0.72 dB from the Shannon limit [8]. It is noted here though that any finite code has a finite probability of error at any signal-to-noise ratio.

Lentimaier and Zigangirov proved that generalised low-density parity-check codes have a minimum distance which grows linearly in block length [29], as Gallager proved for low-density parity-check codes [22]. Additionally, for GLD codes the ratio of minimum distance to block length is closer to the Gilbert-Varshamov bound³ than for the low-density parity-check codes that Lentimaier and Zigangirov considered. In this work the construc-

tion of LDPC codes was restricted to a method first described by Gallager. The construction is simple, but does not yield codes with graphs of the largest possible girth, see Chapter 3, which can impact the codes minimum distance and performance.

Another generalisation of LDPC codes is the construction of codes over higher order fields, in particular $GF(2^m)$, was investigated by Davey and MacKay [16]. The results showed improvements in simulation performance for codes with an average of 2.3 set elements per column, but none for codes with 3 set elements per column.

2.9 Summary

Low-density parity-check codes can be constructed for almost any desired code rate, particularly when using irregular LDPC codes. Gallager proved that the performance of a low-density parity-check code is improved through the use of a very long block length. Implementing the encoder of a block code with a very long block length is potentially a considerable problem using traditional encoding algorithms for linear block codes.

Tanner showed that LDPC codes can be represented as a bipartite graph by considering the parity check matrix as an incidence matrix for the two opposing node types representing the columns and rows of the code [63]. The graph representation of LDPC codes is extremely useful in understanding the decoding algorithms for LDPC codes.

Cycle length constraints are placed on the construction of a graph representing an LDPC code by the iterative decoding algorithms for the codes. Methods for constructing codes which maximise the minimum cycle length in the graph of the code have not been published.

Open research topics therefore include methods of finding good LDPC codes, efficient encoding algorithms and methods to implement LDPC encoders and decoders. These topics will therefore be addressed in the subsequent chapters of this thesis.

-
3. The Gilbert-Varshamov bound is the asymptotic bound on the ratio of minimum distance to block length for all randomly chosen linear parity-check codes as the block length tends to infinity, see theorem 2.3.

Chapter 3

Code Construction

When implementing a forward error correction scheme the best possible coding performance subject to the constraints of the application is sought. However, finding the best possible low-density parity-check code subject to the constraints of practical block length and code rate has not been widely investigated.

Many papers report results for ensembles of random codes rather than individual codes [34, 51]. Bounds for decoding thresholds are also normally derived assuming a code with an infinite block length [54]. The theoretical results derived using these assumptions and random codes are extremely important in understanding LDPC codes but do not show how to find a code with the best possible performance.

This chapter contains a review of existing techniques for constructing low-density parity-check codes. Following the review a method of code construction is proposed based on minimising the short cycles in the code. A metric is introduced to calculate the cost of inserting a new edge into a partially complete graph in terms of the cycles the new edge will introduce. The algorithm proposed builds a code by inserting edges into a partially complete graph which minimise the cost metric. The minimum cost code construction is used to design a 1024-bit rate $1/2$ code for wireless applications and a 32,640-bit rate $239/255$ code for fiber optic applications.

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

Figure 3.1: An example of a low-density parity-check code with $n=20$, $j=3$ and $k=4$ constructed using Gallager's method, with rate $r=1-j/k = 1/4$ [22].

3.1 Gallager's Code Construction

Gallager described a method for constructing regular (n, j, k) codes. Where n is the codes block length, $j = d_v$ is the number of set elements per column of the parity check matrix and $k = d_c$ is the number of set elements in each row. The parity check matrix of the code will be an $m \times n$ matrix, where $m = n - k$. The construction divides the parity check matrix into j sub matrices, each an $(m / j) \times n$ matrix, as shown in Figure 3.1. Each of the sub matrices has a single one in every column and k ones in every row. The first sub matrix is constructed using a regular ordering, with all of the ones in descending order. The construction results in the i 'th row containing ones in columns ik to $(i+1)k-1$ [22]. The remaining $(j-1)$ sub matrices are column permutations of the first sub matrix.

Gallager's construction will not yield a graph with the maximum possible girth for a given block size, nor does it guarantee that no columns or rows have overlaps, but it is very simple. One improvement proposed by Gallager is to prevent column overlaps greater than one element between any pair of columns corresponding to cycles of length four while performing the permutations, but this still does not optimise the graphs girth.

3.2 Random Matrices

Although Gallager used random permutations of the columns of the first sub matrix to generate low-density parity-check codes there is still a significant amount of structure imposed by this construction technique.

Luby *et. al.* have proposed a completely random construction technique to obtain a code satisfying the desired row and column weight profiles [33]. It is possible to construct a random code by taking the set of all edges in a graph and ordering them $\{0, \dots, e-1\}$. All possible variable node connections for the edges, named sockets by Luby *et. al.*, are then listed. Each variable node will have the same number of sockets as the desired weight of the column represented by the node. A list of sockets for check nodes is also constructed. A random graph can be created by connecting the graph edges from the variable node sockets to a random permutation of the check node socket ordering. Provided multiple edges between any pair of nodes is avoided, the random connection of graph nodes results in codes satisfying the desired column and row weight profile. The random permutation can also be constrained such that the resulting graph has no short cycles. Any permutation resulting in a column overlap greater than one element can be rejected.

MacKay and Neal constructed random codes by starting with the all zero $m \times n$ matrix and for each column randomly flipping d_v entries [36]. This construction yields columns with weights $\{d_v, d_{v-2}, \dots, 0\}$ (d_v even) or $\{d_v, d_{v-2}, \dots, 1\}$ (d_v odd) and rows with a random number of entries. The construction can be modified so that the bit flipping does not flip any element which has already been flipped and avoids column overlaps greater than one element, at the cost of extra computational effort during the code design. However, Bazzi, Richardson and Urbanke have proven that the performance of a code is optimised through the concentration of the row weights at a single weight or two consecutive weights, Theorem 2.1. Therefore this construction is sub-optimal.

3.3 Structured Matrices

Many methods of constructing low-density parity-check matrices in a structured manner have been proposed. The methods are used to simplify one or more of the following:

- finding a code,
- encoding codewords of the code, or
- reducing the memory required to store a code.

3.3.1 Permutation Matrix Code Construction

Gallager's code construction was based on cyclic permutation of the columns of sub matrices. MacKay and Neal used variations of the idea of permutations [36], including the design of codes with half of the parity bits, $m/2$ columns, with weight two. Another method examined was the construction of a matrix for a (j,k) regular code where k is an integer multiple of j , that is $k = c.j$ and c is an integer, from a grid of $j \times k$ identity matrices with each identity matrix of size $(m/j) \times (m/j)$ and permuting the sub matrices. The construction of a regular $(3,6)$ LDPC code using this method is shown in Figure 3.2. The construction requires every parity check to contain an element from every group of m/j bits in the block. This does not yield the longest possible minimum cycle length for a given block size.

To improve the performance of the random codes that MacKay and Neal generated they also proposed the removal of all columns which form length six cycles¹. The removal of these columns results in a code with a lower rate than initially designed for and one that has a sub-optimal irregular row weight profile [36].

Irregular codes based on the permutation of sub-matrices were also investigated by MacKay, Wilson and Davey with error floors occurring in some of the codes due to short cycles in the graph representation of the codes [38].

1. see Figure 2.4 part (b) for an illustration of a length six cycle.

$$\mathbf{H} = \begin{bmatrix} \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) \\ \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) \\ \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) & \pi(\mathbf{I}_{m/3}) \end{bmatrix}$$

Figure 3.2: A regular $(3,6)$ parity check matrix constructed from column permutations of $(m/3) \times (m/3)$ identity matrices. Each function $\pi()$ is a unique random permutation of the columns of the matrix it operates on.

3.3.2 Convolutional Code Based Low-Density Parity-Check Codes

Convolutional codes are commonly used for applications requiring low rate codes and low complexity implementation, such as wireless systems [7]. The encoder for a convolutional code is very simple to implement and consists of a shift register and a number of exclusive-or (XOR) gates to implement a parity function.

A convolutional code is specified by a set of generator sequences, which specify the encoders output for an impulse input. For example a systematic rate $1/2$ convolutional code can be constructed using the generator sequences:

$$g^{(0)} = (1 \ 0 \ 0 \ 0) \quad (3.1)$$

and

$$g^{(1)} = (1 \ 1 \ 0 \ 1) \quad (3.2)$$

resulting in a code with the transfer function:

$$G(D) = [1 \ 1+D+D^3]. \quad (3.3)$$

The convolutional code specified by equation (3.3) and with the encoder shown in Figure 3.3 has a constraint length, or memory, of three and can be implemented using three shift registers.

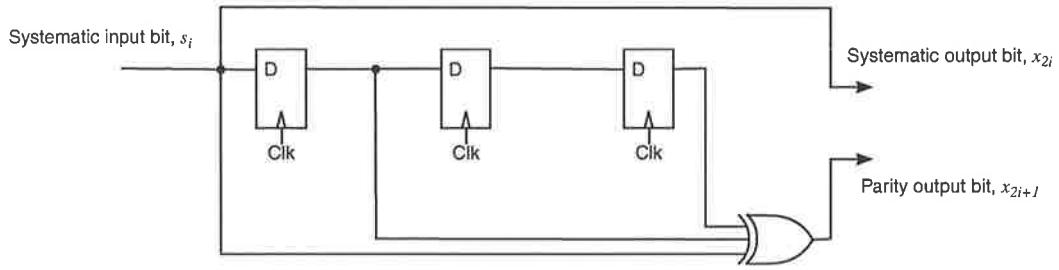


Figure 3.3: Encoder for a simple rate 1/2 systematic convolutional code with transfer function $G(D) = [1 \quad 1+D+D^3]$.

Convolutional codes can be specified by an infinite generator matrix. If the code is terminated and the data stream broken into blocks the generator matrix becomes finite. The convolutional code specified by equation (3.3) with the encoder shown in Figure 3.3 has the generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ & & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ & & & \ddots & & & & & & \ddots \end{bmatrix} \quad (3.4)$$

A terminated convolutional code has a sparse parity check matrix. It is therefore possible to decode codes based on convolutional codes, such as turbo codes, using a decoder for low-density parity-check codes.

The complexity of traditional decoders for convolutional codes, such as a Viterbi decoder, is exponential in the memory of the code [7, 30]. Therefore commonly used convolutional codes have relatively short memory. The short memory of the convolutional codes used in constructing turbo codes results in their associated parity check matrices containing many short cycles. Due to the large number of short cycles turbo codes do not perform well when decoded using an LDPC decoder.

The length of cycles in the parity check matrix for a convolutional code can be increased by increasing the constraint length of the code. Therefore using a low-density

parity-check decoder to decode a convolutional code with a very long memory combines the simplicity of encoding a convolutional code with the very good performance of a low-density parity-check decoder. Felstrom and Zigangirov investigated and improved upon this idea by using very long constraint length convolutional codes with time-varying generator functions [18]. The use of a time varying generator function further improves the length of cycles in the parity check matrix for the convolutional code.

3.3.3 Upper or Lower Triangular Parity Check Matrices

Parity check matrices with the parity bits in either upper or lower triangular form enable simplified encoding. Encoding can be considered as the solution of a set of linear simultaneous equations, where the solution required is the value of the m parity bits for a given block of $(n-m)$ data bits. With an upper or lower triangular matrix this can be done simply using back-substitution.

Codes constructed in this way were proposed by Ping, Leung and Phamdo [47]. The systematic section of the parity check matrix, columns $(m, \dots, n-1)$, is constructed using Gallager's sub matrix permutation method on $(m/j) \times (n-m)$ matrices. The parity check section, columns $(0, \dots, m-1)$, is constructed with a one on every element of the diagonal and a one directly below all of these set elements, as shown in Figure 3.4. The result is weight columns $(0, \dots, m-2)$ of weight two and column $m-1$ having weight one. With soft decision decoding the weight two columns are acceptable if the number of edges with

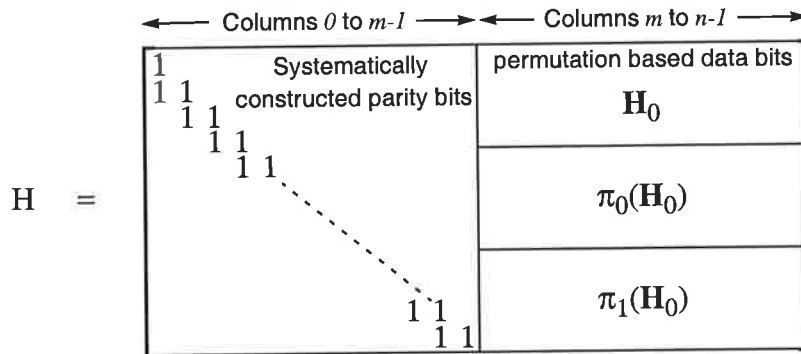


Figure 3.4: Construction of a lower triangular low-density parity-check code, with weight three columns for data bits, π_0 and π_1 are column permutation functions.

this weight is below a stability threshold for the code rate and column weight profile [54]. The weight one column will result in some lower weight codewords, potentially causing an error floor.

3.3.4 Low-Density Generator Matrices

Cheng proposed codes based on sparse systematic generator matrices which enable efficient encoding in his thesis [11]. The codes are called low-density generator matrix (LDGM) codes. Each systematic bit is involved in a fixed number of parity check equations, χ . The code can be represented by a generator matrix, $\mathbf{G}$ such that:

$$\mathbf{G} = [\mathbf{I}_m | \mathbf{P}] \quad (3.5)$$

Where $\mathbf{I}_m$ is an $m \times m$ identity matrix and $\mathbf{P}$ is an $m \times (n - m)$ parity generator with χ set elements per column. Cheng's proposed decoding algorithm also acts on this generator matrix. Although the proposed decoding algorithm is different than that for a low-density parity-check code, the generator matrix here can be considered as a parity check matrix with columns representing parity bits with weight one. Although the code construction simplifies encoding the weight one columns lead to low weight codewords. A characteristic of the LDGM codes is the presence of error floors due to low weight codewords.

Oenning and Moon constructed high rate codes from a parallel concatenation of permuted parity checks [43]. All of the columns corresponding to systematic data have a fixed weight. Oenning and Moon used weight four columns, and all of the columns corresponding to parity bits are of weight one. The structure is also that of a systematic linear block code, $\mathbf{H} = [\mathbf{I}_m \ \mathbf{P}]$. Matrices with columns of weight one, such as the parity bits in this construction, result in low weight codewords [63, 54] and the low weight codewords lead to error floors [36]. The paper shows simulation results with the same coding gain for the low density generator matrix codes and LDPC codes. The low-density parity-check codes used for comparison were constructed following MacKay's sub-optimal method of random parity check matrix construction [39].

3.3.5 Geometric Fields and Steiner Systems

Code construction techniques have also been proposed using the results of geometry and graph theory. MacKay and Davey used Steiner systems to prove limits on the maximum block length of a code with a fixed number of parity check equations that will not violate a maximum column overlap constraint [37]. Although this will prove the existence of a matrix it may be difficult to find a particular code. The use of algebraic or projective geometry to construct graphs with large girth such as Steiner systems and Ramanujan graphs has been studied but has not been widely used to construct codes [41]. This is a possible area for further research.

Parallel to this work a number of people have investigated the geometric construction of low-density parity-check codes. The work was published in the Proceedings of the International Symposium on Information Theory 'ISIT2001' which took place in June 2001 [31, 65, 55, 62].

Lin, Tang and Kou used euclidean and projective geometries to form codes from the intersection of μ -flats and $(\mu-1)$ -flats² [31]. The construction uses the set of all μ -flats as columns and $(\mu-1)$ -flats as rows of the parity check matrix. Set elements in the graph correspond to where a μ -flat intersects a $(\mu-1)$ -flat. The rate, block length and number of set elements per column and row of codes constructed using this technique are not very flexible.

Vontobel and Tanner used finite generalized quadrangles to similarly construct codes [65]. A generalised n -gon has the property that its graph has diameter n and girth $2n$. Generalised n -gon's are only known to exist for $n = 2, 3, 4, 5, 6, 8$. These graphs are constructed from point-line incidence matrices, as in Lin, Tang and Kou's work.

Ramanujan graphs were used in two papers, following work first done by Margulis in 1982 [41]. Rosenthal and Vontobel also introduce an algebraic construction of irregular graphs in their paper [55]. Sridhara and Fuja examine codes modulated with higher order constellations, using algebraic LDPC codes as the component codes [62].

2. With $\mu = 0, 1$ and 2 the resultant μ -flats are: a zero-flat which is a point, a one-flat which is a line, a two-flat which is a two dimensional plane etc.

One apparent problem with using algebraic constructions are constraints on the block size, code rate and column weights which can be constructed using the techniques. It is possible to construct a longer block length code than required and then remove the extra columns, but this will lead to irregular row weights and affect the performance of the code. For this reason heuristic approaches to code construction may remain useful even after algebraic code constructions have been more completely investigated.

3.3.6 Burst Error Protection

Although low-density parity-check codes are very well suited to correcting errors on an AWGN channel there exists the possibility of correcting bursts of errors also. Traditionally codes for protection against burst errors have been designed by interleaving algebraic codes, for example Reed-Solomon or BCH codes [7]. The technique involves using an interleaver to spread bursts of errors over multiple code blocks. The ability of the component algebraic codes to correct groups of bit errors is then efficiently exploited. Low-density parity-check codes are also inherently good candidates for burst error protection due to their very long block length. Provided the length of a burst of errors is small relative to the block length the distribution of the errors has very little effect on the error correction capability of the code, as the particular error distribution is less important than the percentage of errors in any block.

The error correcting performance of low-density parity-check codes for channels with bursts of errors can be improved through enforcing some structure on the parity check matrix. Unlike interleaved algebraic codes where the goal is the distribution of the errors over as many codes as possible, better performance for low-density parity-check codes will be obtained through concentrating the errors into a smaller number of parity checks. This can be understood from the fact that error correction requires a large percentage of correct parity checks and grouping many of the errors into a single parity check results in a lower percentage of the total number of parity checks being incorrect in a block of data with burst errors. Grouping a burst of errors into a single parity check can be understood by considering the decoding algorithm based on expander graphs, discussed later in Chapter 5 [57].

The required structure of a parity check matrix which results in improved burst error protection is to have as many parity checks as possible consisting of sequential columns of the parity check matrix. This is exactly how the first sub-matrix of Gallager's code construction method is organised. However, in the applications considered here burst error protection is not required and will not be further considered.

3.4 Cycle Minimisation

When Gallager first studied low-density parity-check codes he devised decoding algorithms which assume that the parity check matrix is infinite, with no cycles present in the graph of the code. Cycles in the graph result in the information used to update the value of a data bit or parity check not being independent and uncorrelated. The algorithm for updating values is only optimal while its inputs remain uncorrelated. Although a formal proof does not exist many researchers have reported significant degradation of performance and error floors due to the presence of short cycles in codes.

Due to the strong relationship between short cycles and degraded code performance Gallager enforced a constraint requiring that no pair of columns have an overlap greater than one element during the construction of his codes, preventing cycles of length four [22]. This constraint was also used for weight two columns by Richardson, Shokrollahi and Urbanke during the construction of their one million bit code [54], and Chung, Forney, Richardson and Urbanke's construction of a ten million bit code [14]. Mackay and Neal also used this column overlap constraint in some of their constructions [36]. They also examined the refinement of codes after construction by the removal of columns until no length six cycles remain either.

Short cycles in the graph of a code have been identified as characteristic of a code with poor coding gain and a code which potentially has an error floor. The removal of short cycles after a code has been constructed does not solve the problem of how to find a good code without short cycles. A method of minimising the number of short cycles introduced during the construction of a code is therefore required.

3.5 A Minimum Cycle Cost Code Construction

Algorithm

An algorithm will be developed which constructs a code by adding edges to a graph which starts with no edges. The addition of edges to the graph will be constrained such that introduction of short cycles during the construction is minimised. A cost metric which measures the number of short cycles introduced when adding an edge to a graph will be developed. By selecting the edge which minimises the cost metric at each step of the code construction a minimum cost code will be obtained. To compare the performance of the codes designed using the algorithm it will be shown that simulations of the coding performance using only the all zero codeword is characteristic of the codes performance with all other codewords for any symmetric channel. Two codes will be designed using the minimum cycle cost code construction technique, a 1024-bit rate $1/2$ irregular code and a 32,640 bit rate $239/255$ regular code.

3.5.1 Code Comparison

When designing a code it is necessary to compare codes based on measurable metrics. Since the output bit error rate at a given signal-to-noise ratio (SNR) or the SNR required to achieve a target bit error rate are the major design targets, it is desirable to be able to compare two codes on this basis.

For codes designed through a parity check matrix a corresponding generator matrix or alternative encoder must be found to allow simulation of the error correcting performance of the code. However, generating an encoder for a low-density parity-check code with a very long block length is both difficult and time consuming.

It is well known that for a linear block code all codewords have the same distance properties [40]. Gallager showed that all codewords for a LDPC code also have the same distance properties [22]. Richardson and Urbanke further showed that given any symmetric channel and decoders with symmetric update rules this property applies to many classes of message passing decoders operating on the bipartite graph representation of a low-density parity-check code [52]. Given the distance properties proven by Gallager and later Rich-

ardson and Urbanke, it is possible to simulate LDPC codes using the all zero codeword and compare codes based on the simulation results. Once a specific low-density parity-check code for an application has been found an encoder corresponding to the parity check matrix of the code can be derived.

The ability to simulate the performance of a code without finding a generator matrix for the code will be used to reduce the time taken to compare a group of codes and find the particular code in the group with the best performance.

3.5.2 Metrics for Cycles Introduced by Edges of a Graph

Efficient optimisation algorithms require the choice of a good cost function. During the construction of a low-density parity-check code a cost metric is required which calculates the increase in the number of short cycles in a graph when an edge is added to the graph. Given a partially complete graph for a low-density parity-check code the cost of inserting further edges into the graph for a particular variable node or check node can be evaluated by testing the number of times nodes are repeated in the tree of nodes rooted at the node in question. Repeated nodes in the tree only occur when a cycle is present in the graph and the number of repeated nodes is a measure of the number of short cycles. The proposed cost metric is:

$$C(d) = \sum_{j \in T_d} [w(\lambda_j) \times (m_j - 1)]^{\beta} \quad (3.6)$$

Where:

- d is the depth of the tree for which the cost is being evaluated,
- $j \in T_d$ is a node from the set T_d of all variable nodes in the final level of the depth d tree,
- λ_j is the weight of column j ,
- $w(\lambda_j)$ is a penalty function of the column weight, for example low weight columns are more sensitive to cycles and require a higher penalty than high weight columns,

- m_j is the multiplicity of node j , *i.e.* the number of times it occurs in the final level of the tree, and
- β is an exponent which causes individual nodes appearing many times to be more heavily penalised than a large number of nodes each occurring fewer times.

3.5.3 A Minimum Cycle Cost Code Construction Algorithm

An overview of the algorithm will be presented first. Refinements to the algorithm to improve the performance of the code and enable efficient implementation of the algorithm will be presented in the following sections. Given a cost metric, such as the one in equation (3.6), and a variable node it is possible to find a check node with the minimum cost edge connecting the two nodes. The minimum cost edge can be found by evaluating the metric over all possible check nodes. If an edge with a zero cost metric is found during the search then the search can be terminated early and the zero cost edge can be inserted. The minimum cost edge can then be inserted between the variable node and the corresponding check node. This leads to the following algorithm for finding a low-density parity-check code:

1. Determine the required column weights of the code. For a code to be decoded using hard decisions the code should be a regular code. For a code to be decoded using a soft decision decoder the code should be irregular. The column weights of an irregular code can be optimised using techniques proposed by Richardson, Shokrollahi and Urbanke [54], and Chung, Richardson and Urbanke [13]. It should be noted that these derivations are for infinite codes and the performance of finite codes optimised through the performance comparison of codes with different weight profiles. Determine the average row weight, ρ_{av} , for the desired column weights and code rate, finding the number of rows of weight $\lfloor \rho_{av} \rfloor$ and $\lceil \rho_{av} \rceil$ required.
2. Create an empty $m \times n$ matrix.
3. Insert edges into the graph by selecting variable nodes and then choosing the check node which minimises the cost of inserting an edge for the variable node.

4. When a variable node or check node is connected to a number of edges equal to its weight, remove it from the list of available variable or check nodes.
5. Repeat steps 4 and 5 until all nodes have the selected weights and the graph is complete.

Specific details regarding the initial edge selection, choosing the variable node at each iteration for which the next edge will be added, early termination of the edge addition and refinement of the code designed through the algorithm will be examined in the following sections.

3.5.4 Initial Graph Edges

Lemma 3.1: Any node connected to a single graph edge cannot introduce a cycle in a graph.

It is therefore possible to insert one edge for all variable nodes without testing any cycle constraints. This allows faster insertion of the initial n edges in the graph. It is also possible to introduce some structure into the graph at this point without introducing any cycles in the code.

For channels which are potentially bursty a good initial edge insertion rule, as derived in Section 3.3.6, is to set sequential columns in the graph to belong to the same parity check. This is the same construction Gallager used for the upper most sub matrix, $\mathbf{H}_0$ see Section 3.1. The parity checks assigned this way will require modification if the code is to be made systematic³, such as a column permutation resulting in each parity check constraint containing at least one parity bit.

A second option is to select an ordering which will help the implementation of the code, either reducing the routing complexity or memory requirements for storing part of the graph.

3. A *systematic code* is a code where the original data is transmitted unchanged, except having m parity check bits appended to the uncoded data.

Another option is the insertion of elements (i,i) for $i \in [0, m)$, which forms an $m \times m$ identity matrix in the first section of the parity check code. This can be useful in helping to form a systematic code and deriving a corresponding encoder.

3.5.5 Variable Node Insertion Order

At any stage of the code construction through the addition of edges to a partially completed graph a rule is required for choosing the next variable node which will have an edge added. The order in which the variable nodes are chosen for edge addition has a large effect on the cycles and metrics introduced when adding the final group of edges, as any cycles that have been introduced earlier cannot be removed and can only be made worse by subsequent edge insertion.

When reaching the final stage of edge addition it is important that the number of parity check nodes and variable nodes remaining incomplete is relatively balanced, to give as much flexibility as possible for the remaining edge insertions. If only a small number of variable or parity check nodes remain some edges may be inserted with very large metrics during the final stage of the construction and result in a graph with a large number of short cycles.

Edges should be added to variable nodes sequentially, adding one edge to each variable node before adding another edge to any individual variable node. That is all variable nodes should be connected to two edges before adding a third edge to any variable node and all variable nodes should be connected to three edges before adding a fourth edge to any variable node. When constructing an irregular code all lower weight variable nodes should be completed before higher weight variable nodes. This minimises cycles formed between lower weight variable nodes, which degrade performance more significantly than cycles between higher weight variable nodes.

The worst order for variable node selection is the completion of the edge addition for one node before selecting another node for edge addition. When the final variable nodes are reached there are few parity check nodes of the graph remaining incomplete, reducing the possible choices of where an edge can be inserted for a variable node. The result is many short cycles and very high cost metrics for the final edges inserted into the graph.

3.5.6 Termination of Edge Addition

Before all edges in the graph have been inserted it may become impossible to add any edge for a remaining variable node that will not introduce a column overlap less than one, *i.e.* it will form a length four cycle. As the graph construction continues the calculation of the edge metrics involves searching a greater number of nodes in a more highly connected graph. Therefore, another possible reason to terminate the minimum cost based edge addition is that the calculation time for each subsequent edge has risen beyond a threshold. In either of these circumstances the metric based edge insertion can be terminated.

If the graph being constructed is for an irregular code the code can be considered as finished when a termination condition is reached. The addition of further edges may not improve the performance of the code, or worse be detrimental, if the addition of more edges introduces short cycles in the code. By varying the threshold for edge termination a number of irregular codes can be constructed and compared through simulation results to obtain a code with the desired performance.

When constructing a regular code the addition of the remaining edges can be done using a different cost metric which is faster to test. The method proposed to improve the speed of finding a good code is described in the next section.

3.5.7 Final Edge Insertion

Given an almost complete graph and a set of remaining variable and check nodes it is possible to complete the parity check matrix construction by testing edge insertion to a small depth, for example only testing that no length four cycles are introduced. A large number of codes can be generated using the partially complete graph as a starting point. The completed codes can then be compared using the sum of the square of the metrics for all variable nodes in the graph to a greater depth than the simpler metric was tested to. The metric proposed to compare the codes is:

$$\tau = \sum_{i=0}^n \left\{ \sum_{j \in T_d^i} [w(\lambda_j) \times (m_j - 1)]^\beta \right\}^2 \quad (3.7)$$

where T_d^i is the depth d tree of nodes rooted at node i .

Using a two-pass metric evaluation to different tree depths greatly improves the speed of finding a good permutation of the remaining edges of a code. The use of squared metrics reduces the incidence of nodes with significantly higher metrics than the mean metric in the final graph. This effectively distributes graph cycles across all nodes and prevents a graph with many short cycles involving a small subset of the nodes.

3.5.8 Graph Refinement

A refinement stage can be undertaken for completed parity check matrices in which the edges in the graph with the worst metrics are removed and permutations of them are tested to find variable to check node connections with a lower total squared cost metric. The number of edges in this refinement stage can be iteratively reduced and a final code selected based on the minimum total squared edge metric found in all of the refinement steps.

3.5.9 Benefit of Metric Based Edge Insertion

For codes with relatively short block lengths at a given code rate short cycles are inevitable, for instance length six cycles in 32,640-bit rate 239/255 code, or a 512-bit rate 1/2 code. The advantage of the minimum cost based code generation described above is that any short cycles occur as far from each other in the graph as possible. Evenly distributing short cycles in the graph reduces the effect they have on correlating noise when updating a node of the graph during the iterative decoding. The cycles can also be forced to affect higher weight columns rather than lower weight columns through the penalty function $w(\lambda_j)$. Higher weight columns are less adversely affected by the cycles as they receive information from a greater number of sources.

3.6 A 32,640-Bit Rate 239/255 Regular LDPC Code

The first code considered here is a 32,640-bit rate 239/255 (~ 0.937) regular LDPC code. The code is of the same block size and rate as the 16 way interleaved (255,239) Reed Solomon code used in fiber optic communications [58, 74]. It has been implemented as a non-standard higher coding gain replacement for the interleaved Reed Solomon code for OC-192 and OC-768 SONET transceivers with uncoded data rates of approximately 10 and 40 Gbs⁻¹ respectively [58, 74].

Regular codes have been shown to have superior performance with hard decision decoding [3]. Currently no analog-to-digital (A/D) converter exists for the required 43Gbs⁻¹ throughput of the SONET OC-768 standard, hence a hard decision code was designed.

A comparison of 32,640-bit regular LDPC codes with column weights 3,4,5 and 6 was performed. The results of the comparison are shown in Figure 3.5. All of the codes were decoded using the relative reliability weighted decoding algorithm proposed in Chapter 5. The codes were constructed randomly and, for all but the weight six code, no length four cycles were introduced. For the weight six code the number of length four cycles was minimised but not reduced to zero.

The code with weight five columns resulted in the best performance in the simulation results shown in Figure 3.5. Bazzi, Richardson and Urbanke have also proven that the performance of regular LDPC code decoded with Gallager's Algorithm A is optimised by using a weight five code [3]. Simulation results presented in Chapter 5 for Gallager's Algorithm A also have the best performance when using a weight five code. Therefore a weight 5 code was selected for use in the fiber optic transceiver.

An optimised weight five graph was then constructed using minimum cycle cost code construction described in Section 3.5. All columns in the code were designed with a weight of five. The average row weight of a rate 239/255 code with weight 5 columns can be calculated using equation (2.1) to be 79.6875, or 640 weight 79 rows and 1408 rows of weight 80.

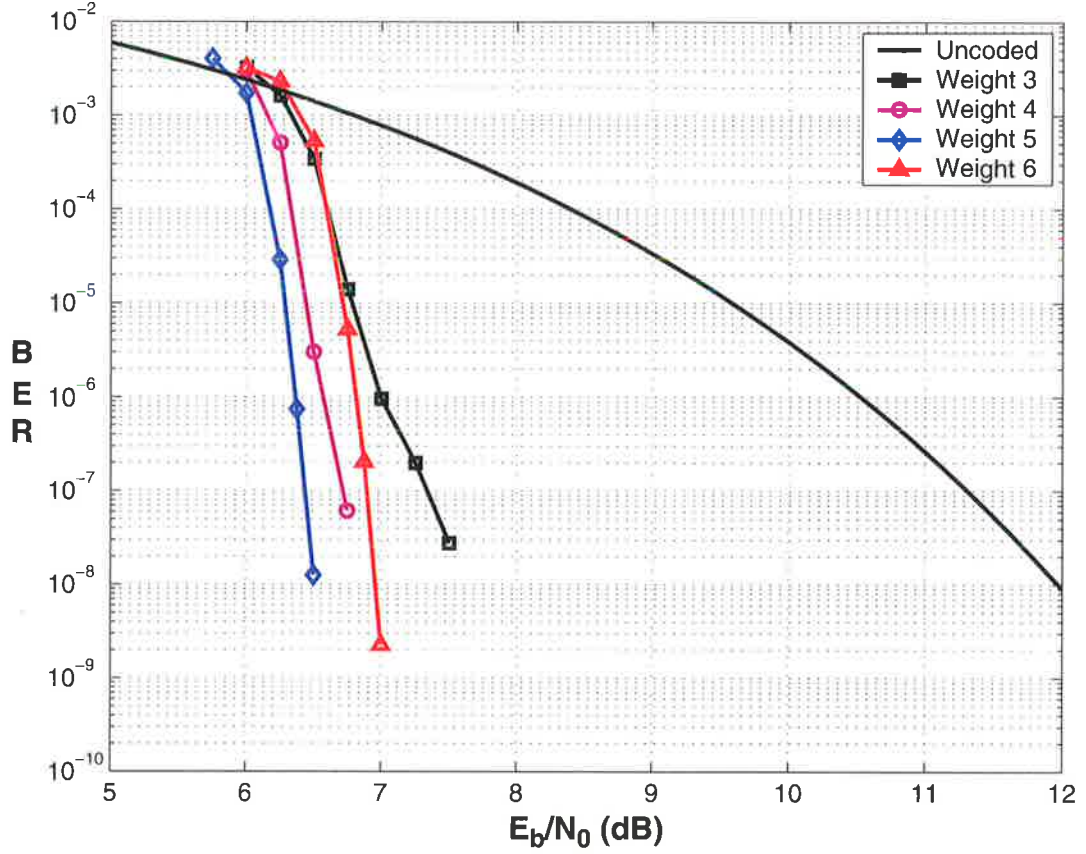


Figure 3.5: Comparison of 32,640-bit rate 239/255 regular codes with column weights 3,4,5 and 6 decoded using 51 decoder iterations of the relative reliability weighted decoding algorithm which is described in Section 5.5.

The insertion metric used in the initial edge insertion was:

$$C(6) = \sum_{j \in T_6} (m_j - 1)^3 \quad (3.8)$$

where $w(\lambda_j) = 1$ since all columns are of the same weight. Construction using this metric was terminated after approximately 90,000 edges were inserted into the graph. The simpler metric which is faster to test:

$$C(4) = \sum_{j \in T_4} (m_j - 1)^3 \quad (3.9)$$

was used to complete the edge insertion. It is possible to calculate the expected number of variable nodes connected at a given depth using the average row and column weights. After traversing from a variable node to all of its connected check nodes and then back to all of the connected variable nodes it is expected that a weight λ_j variable node, on average, is connected to

$$p(1) = \lceil \lambda_j \times (\rho_{av} - 1) \rceil \quad (3.10)$$

other variable nodes. Provided the block length, n , is greater than $p(1)$, then the code can be constructed without length four cycles. At the first level of nodes connected to a variable node in the 32,640-bit rate 239/255 weight five code on average there are

$$p(1) = \lceil 5 \times (79.6875 - 1) \rceil = 394 \quad (3.11)$$

variable nodes. Since $p(1)$ is less than the block length of the code it is possible to construct the code without length four cycles. The number of nodes at the second level of connected nodes can be calculated as

$$\begin{aligned} p(2) &= \lceil \lambda_j \times (\rho_{av} - 1) \times (\lambda_{av} - 1) \times (\rho_{av} - 1) \rceil \\ &= \lceil \lambda_j \times (\rho_{av} - 1)^2 \times (\lambda_{av} - 1) \rceil \end{aligned} \quad (3.12)$$

The expected number of connected variable nodes at the second level of this code is

$$p(2) = \lceil 5 \times (79.6875 - 1)^2 \times (5 - 1) \rceil = 123,835 \quad (3.13)$$

variable nodes, which is greater than the block length of the code. It is therefore not possible to construct this code without length six cycles.

The calculation of the sum of the squared depth six metrics for the 32,640-bit code involves the searching of $32,640 \times p(2)$ nodes which is

$$32,640 \times \lceil 5 \times (79.6875 - 1)^2 \times (5 - 1) \rceil \cong 4.04 \times 10^9 \quad (3.14)$$

nodes. Calculating the cost metric with a search involving 4 billion nodes takes a significant amount of time.

The metric in equation (3.9) only prevents length four cycles, formed by pairs of columns with an overlap greater than one. A number of refinement steps were then applied, initially removing 3000 edges with the worst metrics and then generating a number of codes with no length four cycles based on the partially complete code. The codes were then compared based on the sum of the squared metric with a tree of depth six for all nodes in the code. Each time a code with a new minimum total squared edge cost at a depth of six was found the worst edges of the new code were identified and removed and the search repeated. Approximately 40 refinement steps were undertaken. The number of edges removed was gradually decreased until only a few hundred were removed in the final refinement steps. The refinement results in the minimum metric code in Figure 3.6 becoming the optimised minimum metric code in the Figure 3.6. The effect of short cycles in the random code is also evident in the error floor of the code. The error floor is much higher than the target bit error rate for the code of 10^{-15} and results in the code construction techniques reviewed in Sections 3.1 to 3.3 being unsuitable for use in fiber optic transceivers. The refinement steps increase the coding gain of the minimum metric constructed code by approximately 0.25 dB at a bit error rate of 10^{-15} . This improvement in coding gain is extremely important and significant for this application.

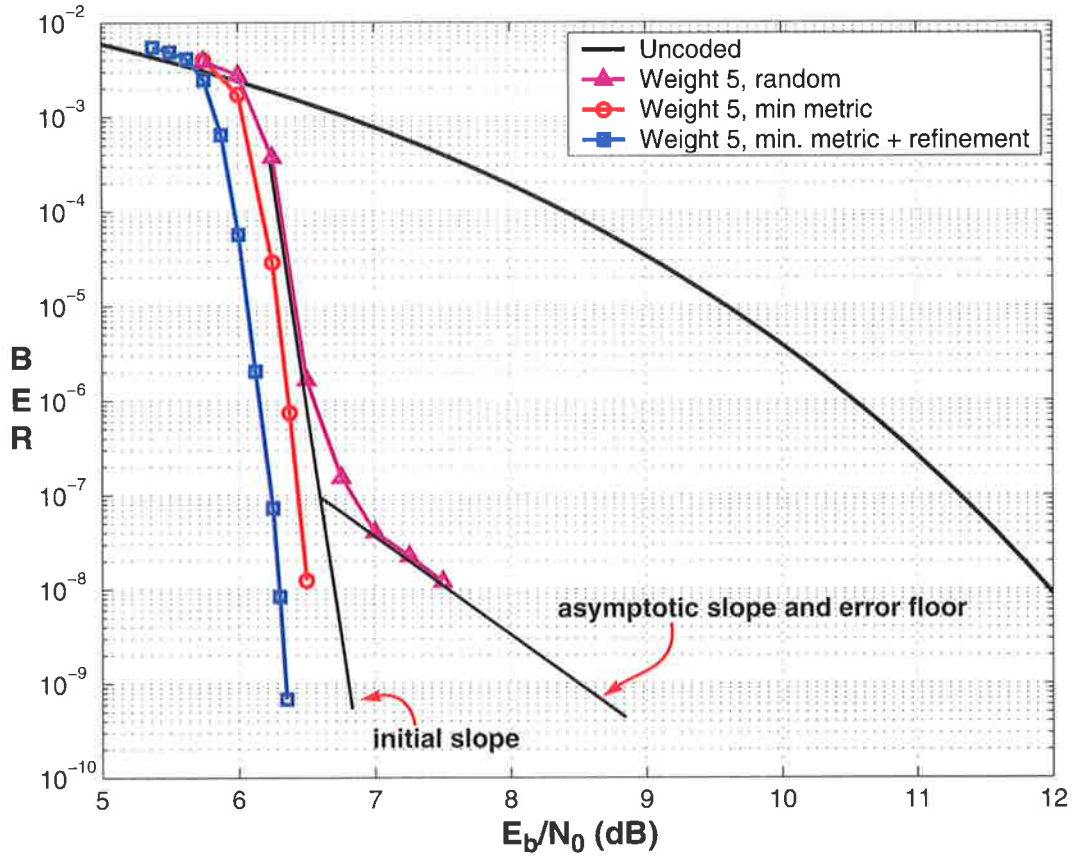


Figure 3.6: Comparison of 32,640-bit rate 239/255 regular codes with column weight 5, one random code, one constructed using metric minimisation and a refined version of the metric minimisation code, all decoded using 51 decoder iterations of the algorithm proposed in Section 5.5.

3.7 A 1024-Bit Rate 1/2 Irregular LDPC Code

The second code designed using minimum cost code construction described in Section 3.5 was a 1024-bit rate 1/2 irregular code [25]. Through simulations of a large number of codes designed using different column weight profiles it was decided to use columns of weight 3, 6, 7 and 8. The total number of edges in the graph is 3328, corresponding to $\lambda_{av} = 3.25$ and $\rho_{av} = 6.5$. The parity checks were divided into 256 weight 6 checks and 256 weight 7 checks. The code was designed using the cost function from equation (3.8), $C(6) = \sum_{j \in T_6} (m_j - 1)^3$, where $w(\lambda_j) = 1$ has been used since it will

be shown that it is possible to make the cost function, $C(6)$, zero for all nodes in this code. In the case of this 1024-bit code with $\lambda_{av} = 3.25$ and $\rho_{av} = 6.5$, the largest number of nodes in the first level of nodes will be connected to columns of weight 8, for which equation (3.10) can be calculated to be:

$$p(1) = \lceil 8 \times (6.5 - 1) \rceil = 44 \quad (3.15)$$

other variable nodes, which is less than the block length. Therefore this code can also be constructed without length four cycles. The number of nodes at the second level of connected nodes can be calculated using equation (3.12). For this code the maximum expected number of connected nodes at the second level is:

$$p(2) = \lceil 8 \times (6.5 - 1)^2 \times (3.25 - 1) \rceil = 545 \quad (3.16)$$

variable nodes, which is also less than the block length. It is therefore possible to construct this code with no cycles of length six, hence the sum of the squared metric in equation (3.8) over all variable nodes can be made equal to zero.

The calculation of the sum of the squared depth six metrics for the 1024-bit code involves the searching of $1024 \times p(2)$ nodes which is

$$1024 \times \lceil 7 \times (6.5 - 1)^2 \times (3.25 - 1) \rceil = 487,872 \quad (3.17)$$

variable nodes, significantly less than the 4 billion nodes the calculation for the 32,640-bit code involves.

The performance of the code when decoded using 64 iterations of a 4-bit message passing fixed point implementation of the sum-product algorithm, which is derived in Section 6.4 is shown in Figure 3.7. The performance of this code will be compared to the performance of 1024-bit rate 1/2 turbo codes in Section 6.4.4.

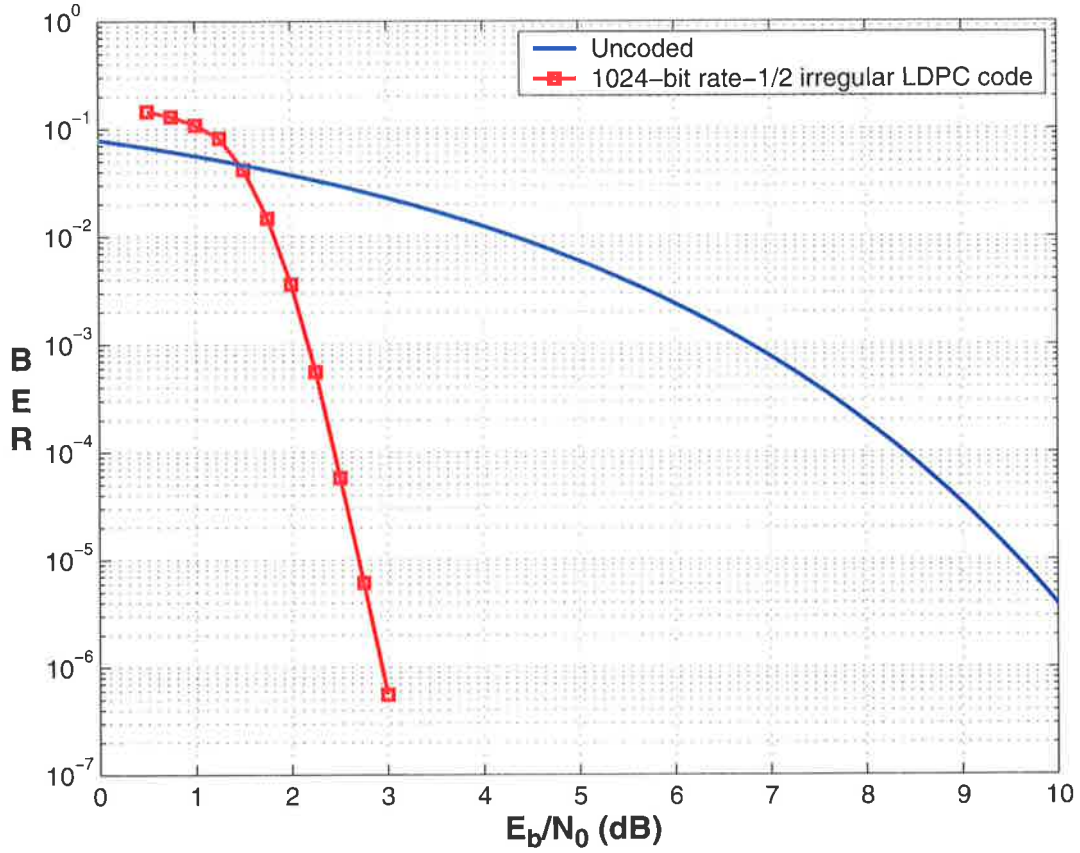


Figure 3.7: Bit error rate performance of a 1024-bit rate 1/2 irregular LDPC code with an average column weight of 3.25 when decoded using 64 iterations of a 4-bit message passing soft decision decoder.

3.8 Summary

In this chapter published methods for constructing low-density parity-check codes have been reviewed. Although Richardson and Urbanke have proven that given a long enough block length the performance of random LDPC codes converges to the average code performance [52] for relatively short block lengths random codes can perform significantly worse than carefully constructed codes. Due to the number of permutations possible it is not feasible to generate a large number of random codes and then select the one with the best performance. Therefore an algorithm or heuristic for constructing good codes is required.

Due to the poor performance of existing code construction techniques a novel code construction technique has been proposed based on the minimisation of a cost metric. A very good cost metric which measures short cycles in the graph of the code has been found for use with the proposed algorithm. The method was used to design two codes, a 1024-bit rate $1/2$ irregular code and a 32,640-bit rate $239/255$ regular code. Although the optimised 32,640-bit code constructed using the proposed algorithm takes significantly more time and effort to construct than a random code with no short cycles the effort is very important for a code used in a commercial application where a performance edge is crucial. All existing code construction techniques resulted in error floors when used to construct a 32,640-bit rate $239/255$ code. Without using the proposed algorithm it would not be possible to design a useful 32,640-bit rate $239/255$ code for optical applications.

Chapter 4

Encoding Low-Density Parity-Check Codes

The majority of work regarding low-density parity-check codes has focused on decoding algorithms and their performance. Encoding low-density parity-check codes is not trivial and has been suggested by some as the barrier to LDPC codes becoming practical for real world applications [36, 59, 60].

Shorter block length turbo codes have equivalent coding gain to LDPC codes [52]. Encoding convolutional or block turbo codes is very simple; the encoder consists of memory for the block of data, an interleaver and one or more small shift register based encoders. The problem faced when encoding a low-density parity-check code is the encoding of a random block code with a very large block length. Existing methods for encoding random block codes become extremely complex for long block lengths. Unless low complexity encoding algorithms can be developed for LDPC codes, turbo codes will remain the more practical implementation choice.

Encoders for systematic LDPC codes will be examined here. A systematic code is a code which does not change the uncoded data and the encoding process merely appends parity bits to the block of uncoded data bits. Existing algorithms for encoding LDPC codes will be examined and an efficient algorithm for encoding LDPC codes is developed which has been specifically designed for high rate regular codes, an application where existing encoding algorithms do not result in an efficient encoder implementation.

4.1 Constrained Parity Check Matrix Methods

A number of methods to simplify encoding have been proposed through constraining the structure of the parity check matrix. Oenning and Moon used a parity check matrix of the form¹ [43]:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_m & \mathbf{P} \end{bmatrix} \quad (4.1)$$

Using a parity check matrix of this form makes encoding very simple, as the parity bits appended to the systematic data bits are just the row parities of the systematic data bits in the parity check matrix. Unfortunately all parity bits participate in only a single parity check equation and are not well protected against errors, therefore also degrading the error protection of the systematic data bits.

Ping, Leung and Phamdo studied an improved graph construction, similar to Oenning and Moon's [47]. A lower triangular parity check matrix was used and the encoding can be performed using back substitution starting at row 0 of the parity check matrix². The proposed construction requires the parity check bit $(m-1)$ be involved in only a single parity check equation and all other parity check bits to be used in two parity check equations. A parity check matrix with columns of weight 2 can perform well with soft decision decoding, but is not suitable for hard decision decoding.

4.2 Cascade Graph Codes

Rather than construct a code from a single large matrix, which can be represented as a bipartite graph, cascaded simpler codes forming a longer length code have been proposed

1. This paper and code construction were also discussed in Section 3.3.4.

2. See Section 3.3.3 for a description of the parity check matrix construction used in this paper.

that are linear time encodable and decodable [35, 61, 49, 57]. This follows on from the original work of Tanner on graph based codes [63]. The problem with this approach is that the minimum distance of the component codes is necessarily lower due to their shorter block lengths. The lower minimum distance of the constituent codes results in a lower minimum distance for the overall code and reduced coding gain compared to an equivalent random code of the same block length.

4.3 Linear Simultaneous Equation Based Encoders

The parity check matrix for a code can be considered as a set of linear simultaneous equations which are satisfied by valid codewords. The systematic data bits are known variables and the parity bits are the unknown variables for which the parity check equations must be solved. A code with an $m \times n$ parity check matrix contains m unknown variables. If the code has rate:

$$r = \frac{n - m}{n} = \frac{k}{n} \quad (4.2)$$

there is a unique solution to the m unknowns. If there is not a unique solution, the parity check matrix is not of full rank and has a higher code rate.

Richardson and Urbanke have proposed encoders based on solving the simultaneous equations that the parity check matrix represents [53]. The algorithm involves reordering the columns and rows of the parity check matrix to obtain an approximate lower triangular matrix. As shown in Figure 4.1 the parity section of the codeword is then divided into two parts and the constraints for each part are solved separately. The algorithm requires finding the column and row permutation of the parity check matrix resulting in the minimum number of rows and columns which cannot be made into an upper triangular form, called the ‘gap’, g . Along with the encoding algorithm Richardson and Urbanke also describe a number of heuristics for generating good approximate lower triangular representations of the parity check matrix.

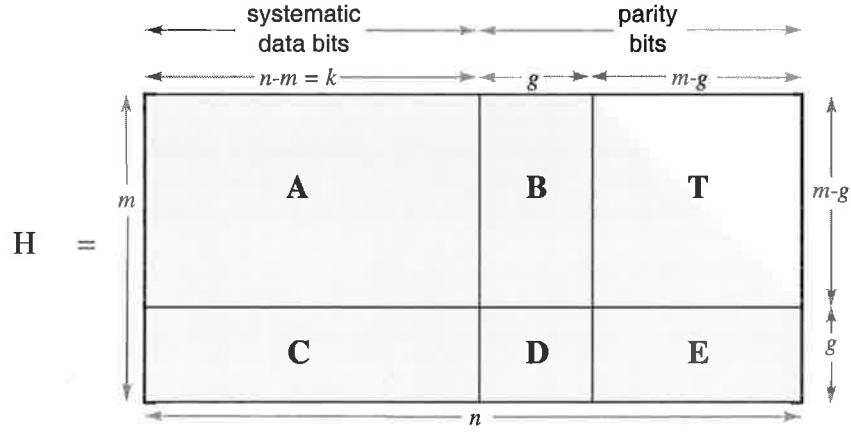


Figure 4.1: Parity check matrix form required by Richardson and Urbanke's encoding algorithm.

The desired form of the parity check matrix is shown in Figure 4.1 and given by:

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix} \quad (4.3)$$

Where: $\mathbf{A}$ is $(m - g) \times k$,

$\mathbf{B}$ is $(m - g) \times g$,

$\mathbf{T}$ is $(m - g) \times (m - g)$,

$\mathbf{C}$ is $g \times k$,

$\mathbf{D}$ is $g \times g$ and

$\mathbf{E}$ is $g \times (m - g)$

The matrix $\mathbf{T}$ is also a lower triangular matrix with ones along the diagonal. All of the sub matrices are sparse. Following the derivation given in [53] the parity check matrix is multiplied from the left by the $m \times m$ matrix:

$$\begin{bmatrix} \mathbf{I}_{m-g} & \mathbf{0} \\ (-\mathbf{E}\mathbf{T}^{-1}) & \mathbf{I}_g \end{bmatrix} \quad (4.4)$$

where $-\mathbf{ET}^{-1}$ is a $g \times (m - g)$ matrix. Resulting in the matrix:

$$\mathbf{H}' = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ -\mathbf{ET}^{-1}\mathbf{A} + \mathbf{C} & -\mathbf{ET}^{-1}\mathbf{B} + \mathbf{D} & \mathbf{0} \end{bmatrix} \quad (4.5)$$

The codeword can then be divided into three parts, $x = \{s, p_1, p_2\}$, where s is the systematic data section of the codeword and the parity section is divided into p_1 of length g and p_2 of length $(m-g)$. Applying the constraint $\mathbf{H}x^T = 0$ from equation (1.8) to the codeword then yields two separate constraint equations:

$$\mathbf{A}s^T + \mathbf{B}p_1^T + \mathbf{T}p_2^T = 0 \quad (4.6)$$

which is an $(m-g)$ element parity constraint and

$$(-\mathbf{ET}^{-1}\mathbf{A} + \mathbf{C})s^T + (-\mathbf{ET}^{-1}\mathbf{B} + \mathbf{D})p_1^T = 0 \quad (4.7)$$

which is a g element parity constraint. The derivation thus defines the $g \times g$ matrix

$$\phi \equiv -\mathbf{ET}^{-1}\mathbf{B} + \mathbf{D} \quad (4.8)$$

With the correct column and row ordering the matrix ϕ will be non-singular. Since $\mathbf{H}$ is of full rank it is always possible to find a column and row ordering which results in a non-singular ϕ . Using the constraints of equation (4.7) p_1 can be found,

$$p_1^T = -\phi^{-1} \cdot (-\mathbf{ET}^{-1}\mathbf{A} + \mathbf{C}) \cdot s^T \quad (4.9)$$

The algorithm requires the precomputation of the dense $g \times k$ matrix

$$\zeta = -\phi^{-1}(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C}) \quad (4.10)$$

The computation of p_1 can be done in $O(g \times (n - m)) = O(g \times k)$ steps as a dense matrix multiplication. Once p_1 is known the final section of the codeword, p_2 , can then be found using equation (4.6). Calculating p_2 requires two sparse matrix multiplications to find $\mathbf{A}s^T$ and $\mathbf{B}p_1^T$ followed by an $(m-g)$ element modulo 2 vector addition to calculate $\mathbf{A}s^T + \mathbf{B}p_1^T$, and a back substitution step using the matrix $\mathbf{T}$ to find the vector p_2 satisfying equation (4.6).

The encoding algorithm is extremely well suited to graphs constructed with irregular column weight profiles containing many low weight columns, particularly weight two columns. Graphs containing a high fraction of columns with weight two can be permuted to have a gap of only a single element [53]. The dense matrix $-\phi^{-1}(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C})$ is problematic when the gap and code block length are both large. For the 32,640-bit code constructed in Section 3.6, the minimum gap found was $g = 198$ parity check constraints. The uncoded data vector length is $k = 30,592$ -bits. Therefore the matrix ζ is a $198 \times 30,592$ dense matrix. Due to the complexity of this algorithm for the 32,640-bit encoder, particularly storing and performing the $198 \times 30,592$ dense matrix multiply, an alternative algorithm was derived.

4.4 Proposed Encoding Algorithm

With high probability there will be some parity check equations in a code which involve only a single parity bit. Each of these bits can be uniquely found using the systematic data bits. If row $m-1$ has only a single parity bit and this is in column $m-1$, then the bit can be found using:

$$p_{m-1} = \sum_{i=m}^{n-1} h_{m-1,i} \cdot s_i \quad (4.11)$$

Solving parity bit values can be continued for all parity constraints involving only one unknown. Each parity bit found in this way reduces the number of unknowns remaining. Parity check equations with two or more parity bits may then have a single remaining unknown, which can also be determined by back substitution using the results of previously determined parity bits.

Solving for some parity bit values leads to an approximate upper triangular matrix with fewer unknowns than the original problem. Using a general random parity check matrix it is not possible to solve for all of the unknown parity bits using this back substitution method. This is because with a random matrix it is not possible to perform column and row permutations yielding a completely upper triangular structure. This is most easily seen by considering the first few columns of the parity check matrix. Formation of the upper triangular matrix structure and maximisation of the girth of the graph are conflicting requirements for the code. The first column of an upper triangular matrix can only contain a single set element. Similarly, the second column can only contain two set elements. Set elements in the first few columns in a dense pattern also form short cycles in the matrix.

In general any parity check matrix can be permuted into the form shown in Figure 4.2, with u unknowns remaining. The sub matrix U is an upper triangular matrix. The back substitution method of solving parity bit values stops at a point where no parity check equation remains with only a single unknown. The solution of the remaining parity check equations must be performed using a different method.

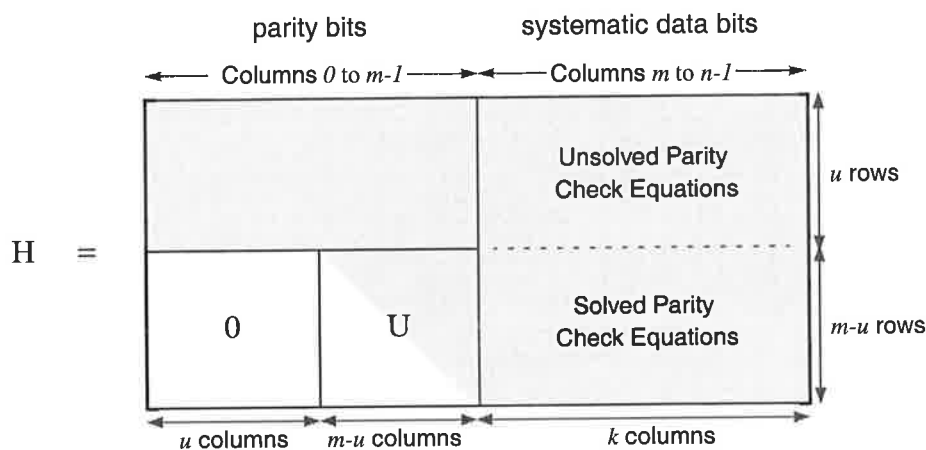


Figure 4.2: Parity check matrix with approximate upper triangular form.

If all columns of the parity check matrix are considered equivalent, it is possible to perform row and column permutations to arrive at a parity check matrix in an approximate upper triangular form such that the number, u , of unknown variables remaining is not a function of the block length, but is a function of the form:

$$u \propto f(r, d_v') \quad (4.12)$$

where r is the minimum girth of the graph, and d_v' , is the average column weight of the u unknown columns. The number of unknown variables remaining is proportional to these values because the sub matrix defined by the first u rows and u columns must have a girth at least equal to that of the entire graph. The number of rows and columns required to form this graph will be a function of the number of set elements in the columns of the sub matrix.

After all parity bits that can be determined in a single back-substitution stage are separated the remaining sub matrix is further partitioned as shown in Figure 4.3 and given by:

$$\mathbf{H} = \begin{bmatrix} \mathbf{T} & \mathbf{B} & \mathbf{F} & \mathbf{A} \\ \mathbf{E} & \mathbf{D} & \mathbf{G} & \mathbf{C} \\ \mathbf{0} & \mathbf{0} & \mathbf{U} & \mathbf{W} \end{bmatrix} \quad (4.13)$$

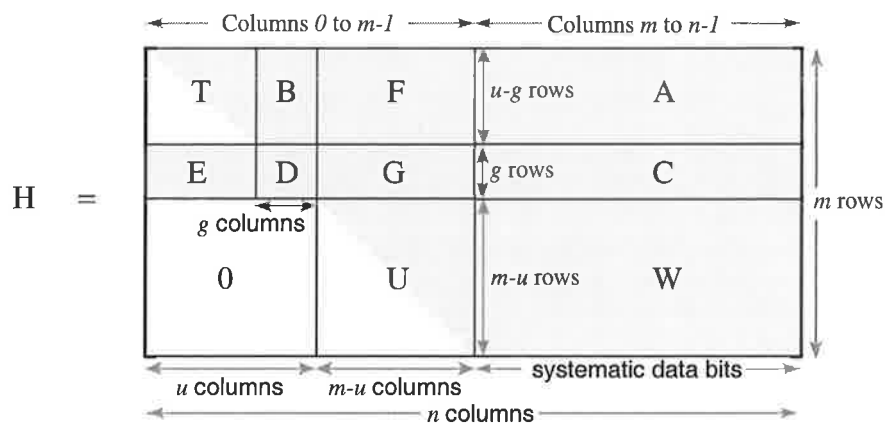


Figure 4.3: Parity check matrix after further row and column permutations to enable efficient encoding.

Where $\mathbf{A}$ is $(u - g) \times (n - m)$,

$\mathbf{B}$ is $(u - g) \times g$,

$\mathbf{T}$ is $(u - g) \times (u - g)$,

$\mathbf{C}$ is $g \times (n - m)$,

$\mathbf{D}$ is $g \times g$,

$\mathbf{E}$ is $g \times (u - g)$,

$\mathbf{F}$ is $(u - g) \times (m - u)$,

$\mathbf{G}$ is $g \times (m - u)$,

$\mathbf{U}$ is $(m - u) \times (m - u)$ and

$\mathbf{W}$ is $(m - u) \times k$.

Both $\mathbf{U}$ and $\mathbf{T}$ are upper triangular matrices with ones along the diagonal. Again, as in equation (4.3), all of the sub matrices are sparse.

Following the reasoning regarding the size of the set of unknown parity equations, u , the gap, g , will also be of a size that is a function of the girth of the matrix and the average number of set elements in columns 0 to $u-1$, as in equation (4.12).

Encoding can then be considered as finding three distinct sets of unknown variables, such that the entire codeword is of the form:

$$x = [p_a \ p_b \ p_c \ s] \quad (4.14)$$

Where s is the systematic data vector, p_c is the $(m-u)$ -bit vector of parity bits which can be determined using a single back substitution pass of the rows of $\mathbf{U}$ and $\mathbf{W}$, p_a is the vector of parity bits found using the first $(u-g)$ rows of $\mathbf{H}$, and p_b is a vector of the remaining g parity bits, associated with the gap.

Multiplying $\mathbf{H}$ to the left by the $m \times m$ matrix,

$$\begin{bmatrix} \mathbf{I}_{u-g} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E}\mathbf{T}^{-1} & \mathbf{I}_g & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{m-u} \end{bmatrix} \quad (4.15)$$

where $-\mathbf{E}\mathbf{T}^{-1}$ is a $g \times (u - g)$ matrix, results in the matrix,

$$\mathbf{H}' = \begin{bmatrix} \mathbf{T} & \mathbf{B} & \mathbf{F} & \mathbf{A} \\ \mathbf{0} & (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}) & (-\mathbf{E}\mathbf{T}^{-1}\mathbf{F} + \mathbf{G}) & (-\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C}) \\ \mathbf{0} & \mathbf{0} & \mathbf{U} & \mathbf{W} \end{bmatrix} \quad (4.16)$$

which spans the same row space as the original parity check matrix, and therefore is an alternative parity check matrix for the code. The encoding algorithm derived later in this section only requires the $g \times g$ sub matrix $(-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D})$ of this alternative representation of the parity check matrix. This matrix can be used to calculate the required value of the parity check bits p_b from the syndrome vector $\hat{s}$ formed assuming the codeword is of the form $x = [p_a \ 0 \ p_c \ s]$. The initial row and column permutation of $\mathbf{H}$ must result in the sub matrix $(-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D})$ being invertible. Again this is always possible if $\mathbf{H}$ is of full rank. Denote the sub matrix as

$$\Phi^{-1} = (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}) \quad (4.17)$$

It should be noted that the matrix Φ^{-1} is different to the matrix ϕ^{-1} used by Richardson and Urbanke. Encoding can begin by finding the vectors of row parities due to the systematic data bits

$$v = \mathbf{A} \cdot s \quad (4.18)$$

$$y = \mathbf{C} \cdot s \quad (4.19)$$

$$z = W \cdot s \quad (4.20)$$

All of these vectors are efficiently calculated since the matrices involved in the multiplications are all sparse. Finding the elements of p_c can easily be done with back-substitution by solving

$$U \cdot p_c + z = 0 \quad (4.21)$$

using

$$p_c(t - u) = \sum_{j=m}^{n-1} h_{t,j} \cdot s_{j-m} + \sum_{j=u}^{m-1} h_{t,j} \cdot p_c(j - u) \quad (4.22)$$

Where $t \in [u, \dots, m - 1]$. Which is equivalent to

$$p_c(t - u) = z_{t-u} + \sum_{j=u}^{m-1} h_{t,j} \cdot p_c(j - u) \quad (4.23)$$

Again this computation is efficient since the matrix U is also sparse.

The next step is to calculate a preliminary estimate of p_a , denoted $\hat{p}_a$, which is the required parity vector to satisfy the first $(u-g)$ constraints of $\mathbf{H}$ assuming p_b is the all zero vector. Using another back-substitution step, $\hat{p}_a$ is found such that:

$$\mathbf{T} \cdot \hat{p}_a + \mathbf{F} \cdot p_c = v \quad (4.24)$$

A g -element syndrome vector, $\hat{s}$, is then found:

$$\hat{s} = \mathbf{E} \cdot \hat{p}_a + \mathbf{G} \cdot p_c + y \quad (4.25)$$

Using the parity constraints and the parity check matrix representation from equation (4.16):

$$(-\mathbf{ET}^{-1}\mathbf{B} + \mathbf{D})p_b + (-\mathbf{ET}^{-1}\mathbf{F} + \mathbf{G})p_c + (-\mathbf{ET}^{-1}\mathbf{A} + \mathbf{C})s = 0 \quad (4.26)$$

Multiplying equation (4.24) by $\mathbf{ET}^{-1}$ gives:

$$\mathbf{E} \cdot \hat{p}_a + \mathbf{ET}^{-1}\mathbf{F} \cdot p_c = \mathbf{ET}^{-1}\mathbf{A}s \quad (4.27)$$

Using the definition of $\hat{s}$ and equation (4.27):

$$\hat{s} = \mathbf{E} \cdot \hat{p}_a + \mathbf{G} \cdot p_c + \mathbf{C} \cdot s = \mathbf{ET}^{-1}\mathbf{A}s - \mathbf{ET}^{-1}\mathbf{F} \cdot p_c + \mathbf{G} \cdot p_c + \mathbf{C} \cdot s \quad (4.28)$$

Therefore:

$$\hat{s} = (\mathbf{ET}^{-1}\mathbf{A} + \mathbf{C})s + (-\mathbf{ET}^{-1}\mathbf{F} + \mathbf{G})p_c \quad (4.29)$$

Using the definition of Φ^{-1} from equation (4.17) and $\hat{s}$ from equation (4.29), then equation (4.26) can be written as:

$$\Phi^{-1}p_b + \hat{s} = 0 \quad (4.30)$$

Therefore p_b can be found using:

$$p_b = -\Phi\hat{s} \quad (4.31)$$

Finally the exact p_a can be determined by using back-substitution to solve:

$$\mathbf{T}p_a = \mathbf{B}p_b + \mathbf{F}p_c + v \quad (4.32)$$

The only operation in the encoding process which does not act on sparse matrices is the multiplication in equation (4.30), which is a dense matrix multiply by the $g \times g$ matrix, Φ .

The encoding process can be summarised by the following steps:

1. Let s be the vector of systematic data bits. Compute all of the systematic row parities, v , y and z , due to the systematic data bits using equations (4.18), (4.19) and (4.20).
2. Using back-substitution find the first set of parity bits, p_c , using equation (4.23).
3. Set $p_b = 0$ and use back-substitution to find an initial estimate $\hat{p}_a$ which satisfies the first $u-g$ parity constraints using equation (4.24).
4. Find $\hat{s}$, the g -bit syndrome vector of rows $u-g-1$ to $u-1$, when $p_a = \hat{p}_a$, $p_b = 0$, p_c and s are as above using equation (4.25).
5. Set $p_b = -\Phi \hat{s}$
6. Find the final exact p_a using back-substitution to solve equation (4.32).

The major difference between this algorithm and the one derived by Richardson and Urbanke is the use of a repeated back-substitution step to reduce the size of the dense matrix used in the dense matrix multiply of both algorithms. For codes that cannot be reduced to have a small gap this algorithm leads to a more efficient encoder.

For regular codes a small gap cannot be found. Codes with relatively high column weights, like the five elements per column in the 32,640-bit code being examined, have larger gaps. For irregular codes that can be permuted into a form with a very small gap Richardson and Urbanke's algorithm will result in a more efficient encoder implementation.

Using the proposed encoding algorithm an encoder for the 32,640-bit rate 239/255 code requires a dense 198×198 matrix multiply, compared to the $198 \times 30,592$ dense matrix multiply required to implement Richardson and Urbanke's encoding algorithm.

Assuming both matrices are equally dense the encoding algorithm derived here reduces the number of exclusive-or operations by a factor of more than 150. The cost of obtaining the reduction in the size of the dense matrix is the performance of a repeated back-substitution. For the 32,640-bit code the repeated back substitution is over a $590 \times (590 + 198)$ sparse matrix. Assuming a density of 40% for the dense matrices and four set elements per column in the back-substitution operation the number of exclusive-or operations to implement Richardson and Urbanke's encoding algorithm is:

$$40\% \times 198 \times 30,592 \approx 2.4 \times 10^6, \quad (4.33)$$

compared to the proposed algorithm:

$$40\% \times 198^2 + 4 \times (590 + 198) = 18,834, \quad (4.34)$$

representing a reduction by more than a factor of 125.

4.5 Encoder Architectures

It will be assumed that the uncoded data has been rate matched to the coded data of the encoder's output, hence the entire encoder can operate using a single clock. The rate matching consists of taking the serial uncoded data and breaking it into blocks of $(n-m)$ -bits, which are equal to the length of the systematic data vector in a codeword, and appending a sequence of zeros with a length equal to the number of parity bits the encoder will insert for each codeword, m -bits. Thus the encoder will replace the padding bits with the correct parity bits as shown in Figure 4.4.

Two methods for generating the row parities, v , y and z for the systematic data using equations (4.18), (4.19) and (4.20) will be considered. The choices are either generating the systematic row parities on-the-fly as the systematic data is being transmitted, or latching an entire data block and generating the systematic parity bits from the block.

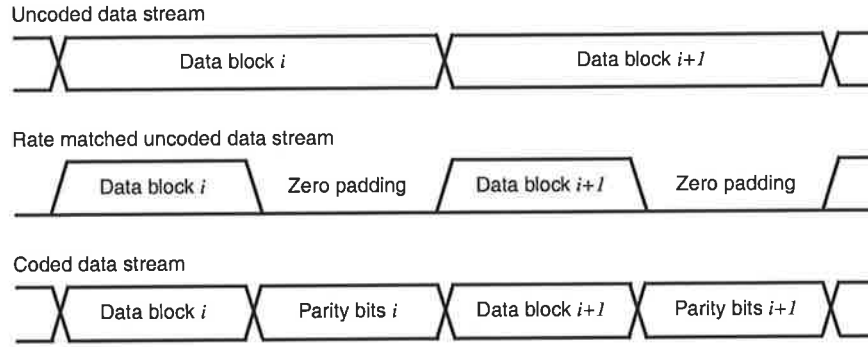


Figure 4.4: Insertion of parity bits and rate matching the input and output of the encoder. Delay between uncoded data, rate matched uncoded data and coded data is not shown.

For very high throughput applications it is not feasible to encode the data from a serial source, therefore an encoder must act on a bus of data bits. With an arbitrary parity check matrix this means each parity bit could be a function of none, one or multiple data bits at every time step. This leads to very complex routing and bit selection problems in the design of an on-the-fly encoder for a random parity check matrix.

The encoder architecture shown in Figure 4.5 shows a possible implementation for an encoder which generates the systematic parity bits, v , y and z using equations (4.18), (4.19) and (4.20), on-the-fly as the systematic bits are transmitted. The serial-to-parallel demultiplexer in Figure 4.5 must provide buffering and rate matching functions. The parallel-to-serial multiplexer is required to delay the systematic bits for a number of clock cycles to enable completion of the parity bit calculation using equations (4.22) to (4.29) in the encoder matrix operation block. Both the routing of the signals and generation of the bit select signals, $Sel(i,j,t)$, are very complex in this encoder architecture. The number of and-gates and exclusive-or (XOR) gates required to implement an on-the-fly encoder will be equal to the number of parity bits multiplied by the bus width of the encoder. In general this makes on-the-fly encoding impractical for codes with a long block length.

An alternative approach is to load and store an entire data block in a series of parallel shift registers. The parity network generating the systematic parity bits, v , y and z , will then be explicitly wired using only the outputs of the required systematic bits in the calculation. The output of every flip-flop will then be connected to exactly λ_j exclusive-or gate inputs, where λ_j is the column weight of the column the flip-flop (F/F) represents.

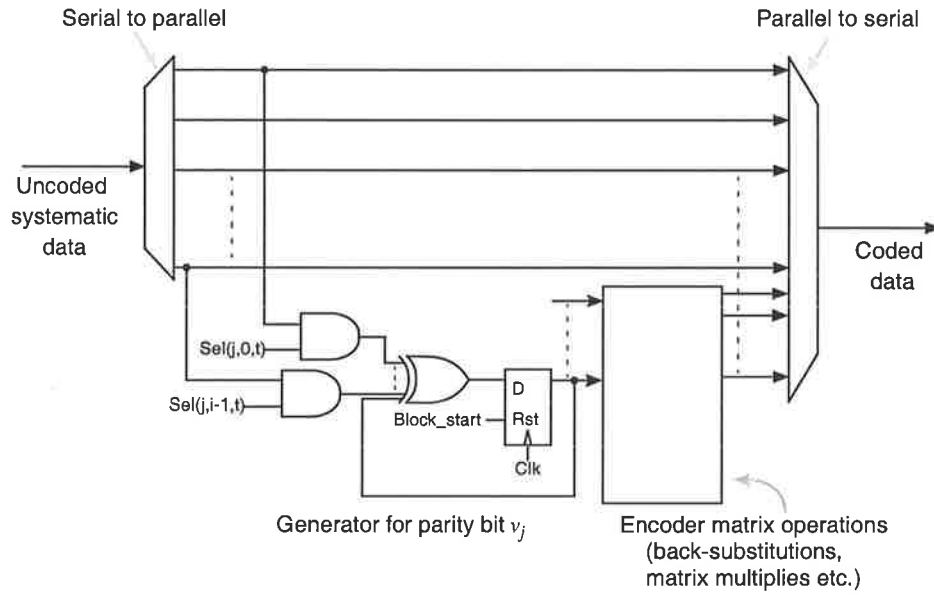


Figure 4.5: Encoder architecture using on-the-fly parity generation for the systematic data row parities, v_j and z from equations (4.18), (4.19) and (4.20) with only a single parity generator shown.

The network generating the systematic parity bits is only required to produce a result when an entire block of data has been shifted into the correct location in the scan chain network. Using a scan chain which is i flip-flops deep the parity bits will only require generation every i cycles. During the other $(i-1)$ cycles the circuits generating the parity results will be switching and dissipating power unnecessarily. Further increasing unnecessary switching will be glitches in the XOR network as signals being XORED together arrive at the gate inputs at different times from different length paths across a large area of flip-flops. For high rate codes with very high fan-in parity checks this glitching could represent a large dynamic power dissipation. The 32,640-bit rate 239/255 code with weight five columns has parity checks with 79 and 80 inputs. Therefore a systematic parity generation network may contain as many as 79 inputs.

To avoid continuous data glitches rippling through the parity generation network a second flip-flop for each bit can be used which is only updated once the entire data block is in the correct position in the scan chain network, as shown in Figure 4.6.

The scan chain element in Figure 4.6 includes buffers on the input and output of the register-register scan path to avoid set up and hold time violations for the flip-flops, which

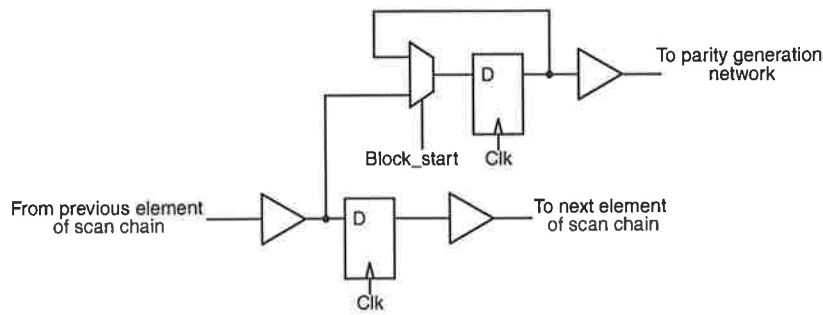


Figure 4.6: Single scan chain element, using a separate data flip-flop to reduce glitching.

can be particularly problematic when crossing from one clock domain to another in a clock distribution tree. Using this architecture all of the systematic bits in the data block must be latched before the encoding begins, which occurs in the clock cycle when the control signal `Block_start` is active.

Although no select logic is required to implement an encoder using the hard wired approach, shown in Figure 4.7, a large number of XOR functions need to be performed, equal to the number of set elements in the $(n-m)$ columns of the systematic section of the parity check matrix.

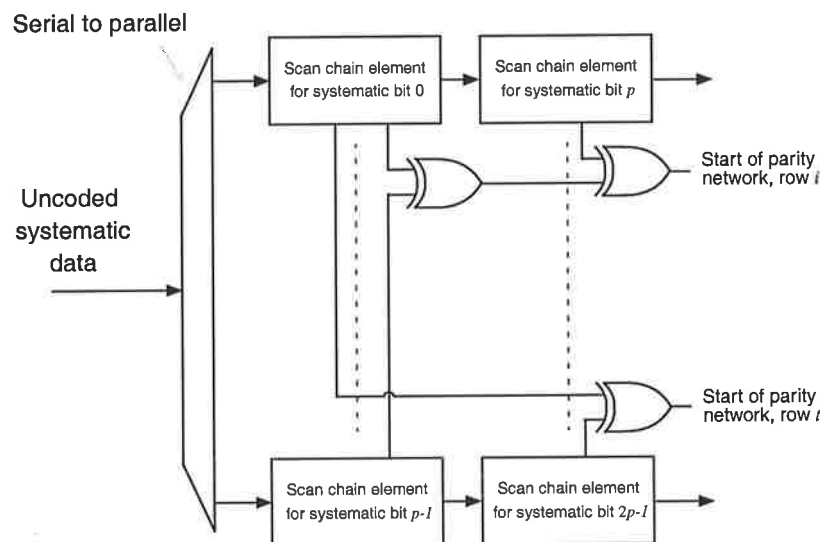


Figure 4.7: Arrangement of scan chain units in a systematic parity generator for an encoder with a hard wired parity network.

4.5.1 Encoder Architecture for Solving Simultaneous Equations

The matrix equations required to calculate the parity bits are all modulo 2 operations which can be easily implemented as a network of exclusive-or gates. For high rate codes the number of inputs to this network, $(m+1)$, is significantly less than when calculating the row parities due to the $(n-m)$ systematic data bits.

Due to the iterative dependencies of the back-substitution operations used to calculate p_a and p_c it is not possible for the evaluation to ripple through the parity calculation of each row and complete the operation in a single clock cycle. The calculations must therefore be pipelined and ripple up the rows of the matrix over multiple clock cycles.

When evaluating parity bits using back-substitution there are three sets of inputs which can be used in the calculation of any bit, the original systematic parity bits, the unlatched evaluated parity bits and the latched parity bits. It is possible to attempt the entire

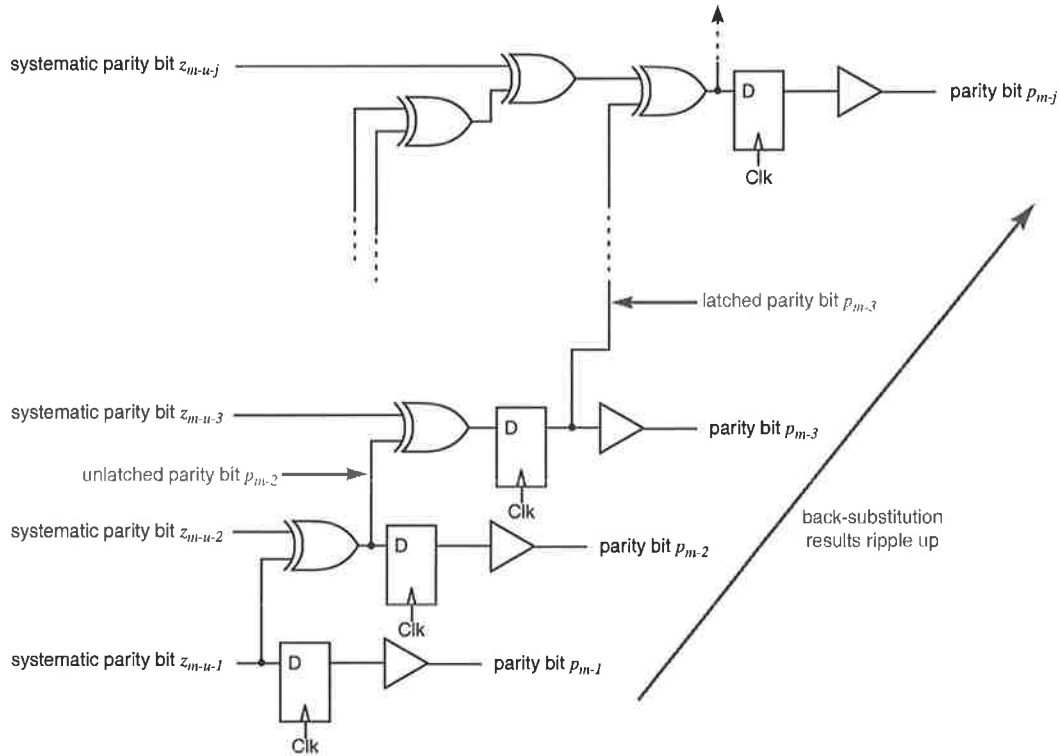


Figure 4.8: Back substitution network for calculating encoder parity bits p_a or p_c . Nearby results ripple through without latching, far away results are used only after latching.

back-substitution in a single clock cycle by using only unlatched parity bits in the iterative calculation of the subsequent parity bits. The use of only unlatched parity bits is infeasible due to the very long critical path of such an operation. Alternatively, using only latched parity bits, results in the $(m-u)$ back-substitution operations in calculating p_c potentially taking $(m-u)$ clock cycles to complete. Using only latched parity bits in the iterative calculation therefore results in a latency which is too large for many practical applications.

For most applications it would seem that a trade-off somewhere between the two extremes is required, balancing the critical path and latency of the calculation. The use of both latched and unlatched parity bits in a back-substitution calculation is shown in Figure 4.8.

4.6 A 32,640-Bit Rate 239/255 Encoder

An encoder for the 32,640-bit code from Section 3.6 was designed using the architecture shown in Figure 4.9. The uncoded systematic data was rate matched, zero padded and parallelised to arrive at the encoder input on a 640-bit bus with a 67MHz clock frequency. Using a 640-bit bus the `block_start` signal has a period of $(32,640/640) = 51$ clock cycles.

The entire encoder is a network of pipelined exclusive-or gates. To avoid glitches propagating through the network it is possible to pipeline successive stages using delayed block start signals. The circuit used to perform this function can be similar to the one used in the systematic bit scan chain, a flip-flop with a feedback multiplexer from its output and a delayed block start signal as the multiplexer's select signal as shown in Figure 4.6.

A block start signal delayed by x clock cycles is denoted as `block_start_cx` in Figure 4.9, Figure 4.10 and Figure 4.11. The systematic data scan chain block and the three back substitution blocks also include internal flip-flops clocked on the global clock.

The 2048 parity check bits were divided into three groups, $\{p_a, p_b, p_c\}$, following the proposed encoding algorithm. Using a search for the minimum gap, g , starting with each of the 2048 parity check columns in turn, the smallest gap found was 198-bits. With the minimum gap column and row ordering, u was found to be 788-bits. Resulting in:

- p_a comprising of 590 bits,
- p_b comprising of 198 bits, and
- p_c comprising of 1260 bits.

The encoder has approximately 65,000 flip-flops and more than 250,000 equivalent 2 input exclusive-or gates. Many of the XOR gates actually used were 3 or 4 input gates. Parts of the encoder netlist were synthesised from VHDL. The blocks not synthesised were the systematic scan chains and systematic parity XOR network. Perl scripts and C programs were used to build the netlist of the scan chains and systematic parity network in a systematic way using an understanding of the relative physical placement of the gates, greatly reducing routing congestion, signal delay and timing violations.

4.6.1 VHDL Implementation of the Encoder

The C program used to design and simulate the code was annotated to automatically generate the VHDL description of the encoder for a parity check matrix representation read from a text file. The structure of the VHDL required to implement the encoder is quite simple and very easily automated.

Automating the VHDL generation allowed rapid changes in the netlist and made it possible to change the parity check matrix structure a number of times as different column or row orderings were compared based on the routing congestion of the corresponding decoder. The changes in routing congestion were examined and compared based on the decoder since it is a much larger and more complex block than the encoder. Any change in the parity check matrix of the decoder had to be replicated in the encoder. The scripted generation of the VHDL description, netlist and cell placement of the encoder greatly simplified this task.

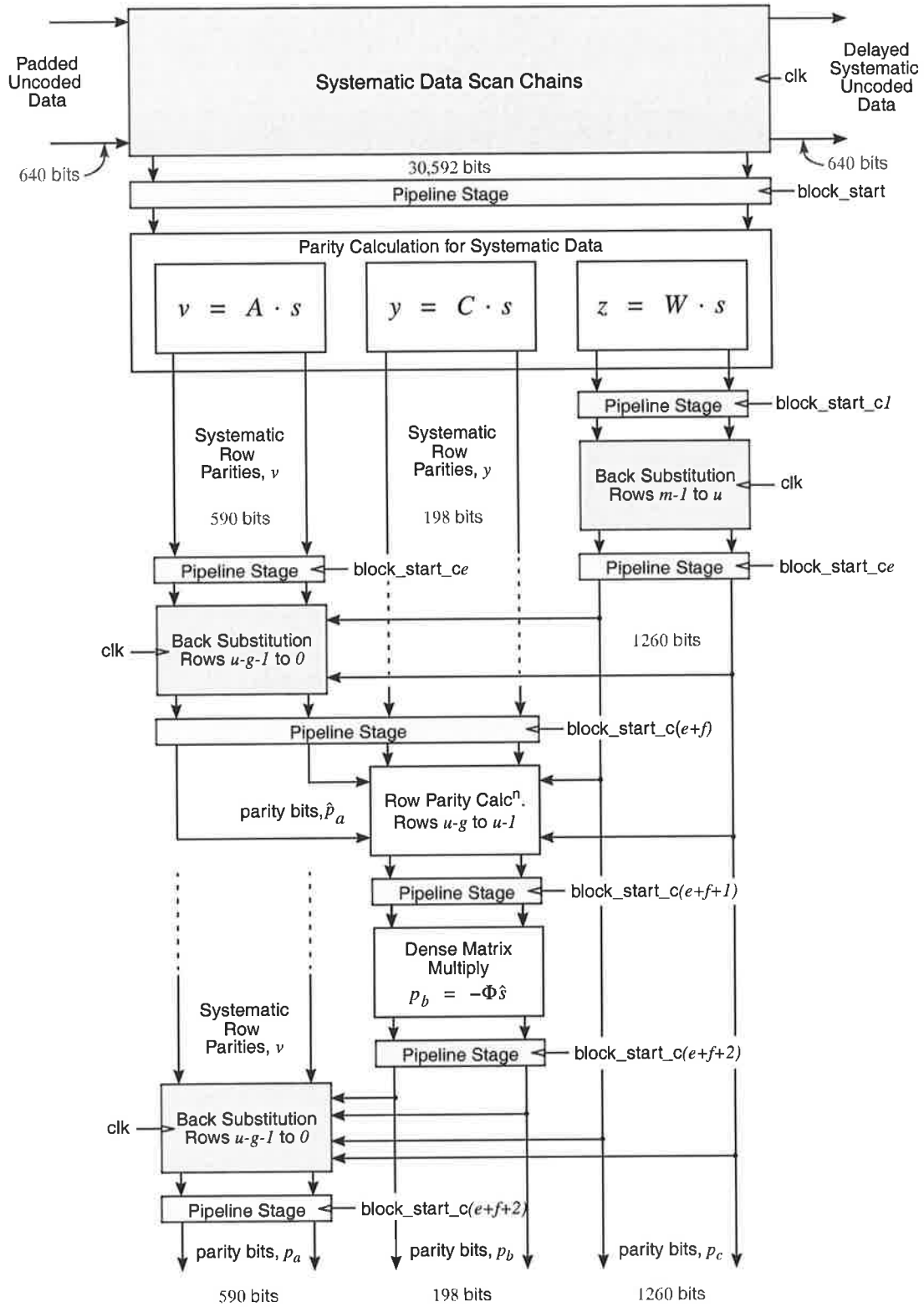


Figure 4.9: Encoder architecture for parity calculation network including glitch propagation control using delayed block start signals.

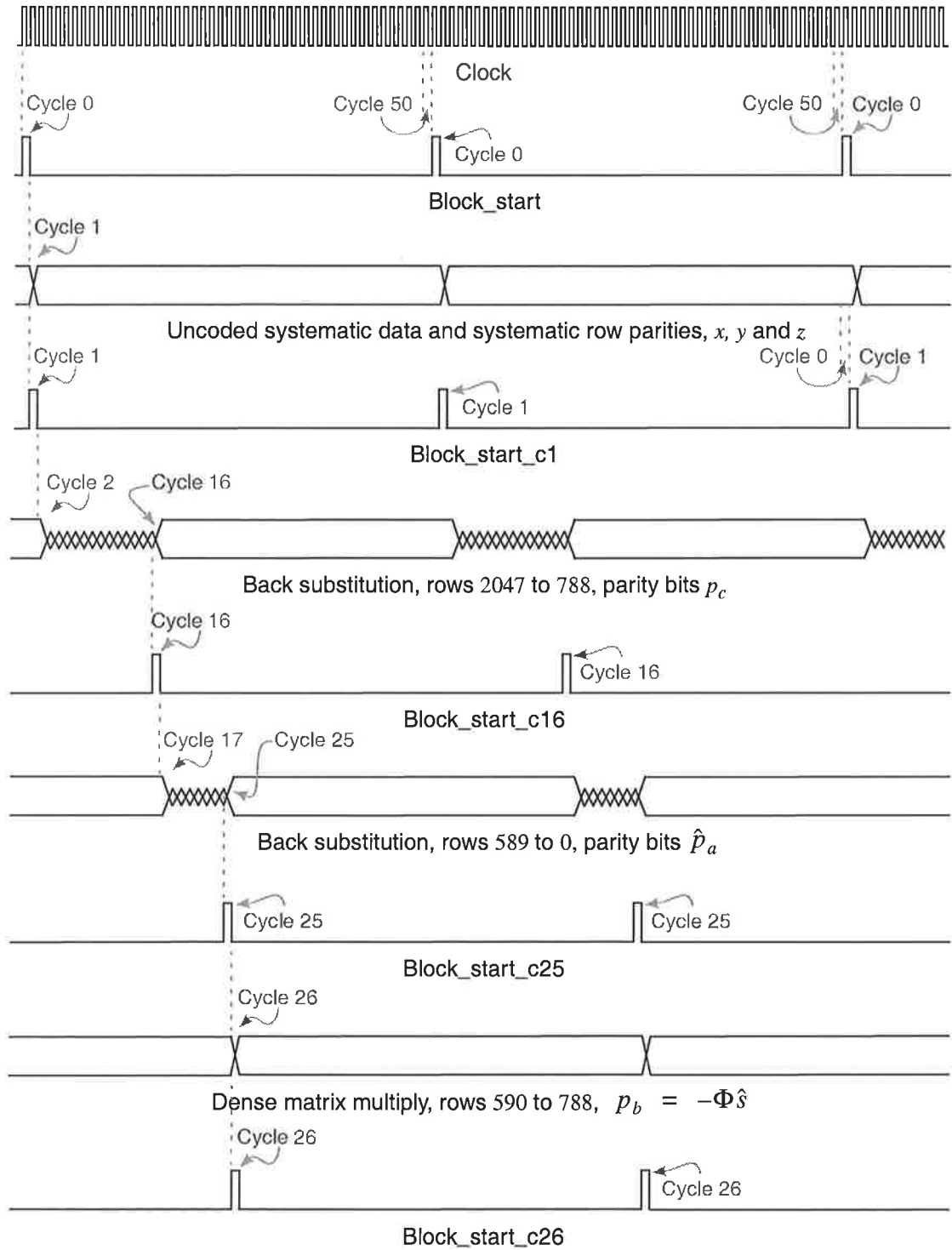


Figure 4.10: Timing diagram for the 32,640-bit rate 239/255 encoder.

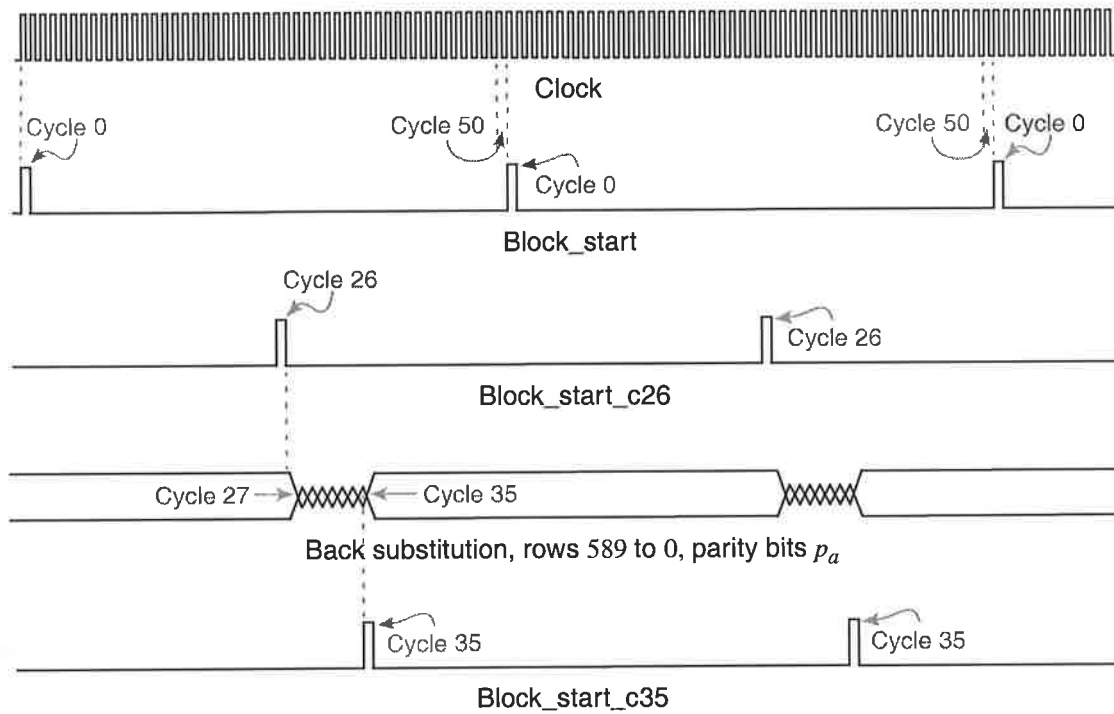


Figure 4.11: Timing diagram for the 32,640-bit rate 239/255 encoder (continued).

The encoder was implemented in collaboration with Andrew Blanksby, Member of Technical Staff, formerly with Bell Laboratories, Lucent Technologies, now with Agere Systems and Douglas Brinthaup, Consulting Member of Technical Staff, formerly with Lucent Technologies Microelectronics Division, now with Agere Systems.

The VHDL description of the interface, control and testing logic for the encoder was all written by Douglas Brinthaup. The author wrote the VHDL description of the encoder and the C-program generation of parts of the VHDL.

4.6.2 Encoder Synthesis

The encoder was synthesised from the VHDL description to a netlist of gates from a standard cell library. All synthesis scripts were written by Douglas Brinthaup. Optimisation and formal verification of the synthesised netlist was also performed by Douglas Brinthaup.

4.6.3 Encoder Layout

The netlist of standard cells for the encoder was placed-and-routed using both standard and custom CAD tools. The custom CAD tools were written by Andrew Blanksby who also performed the place-and-route layout of the encoder. Due to routing congestion in the layout due to the synthesised netlist parts of the netlist were custom generated using an algorithm which utilised information about the location of flip-flops in the scan chains. The custom netlist was verified against the synthesised netlist and greatly reduced the routing congestion of the encoder.

4.6.4 Encoder Timing Analysis and Design Rule Checking

All timing analysis and design rule checks (DRC) were performed by Andrew Blanksby and Douglas Brinthaup. The encoder has been successfully implemented with a throughput of 43Gbs^{-1} of coded data and passed all design rule checks.

4.7 Summary

Encoding low-density parity-check codes using generator matrices is impractical due to the large block length of the codes. Two solutions to this have been examined. The first involves modifying the structure of the code to enable single pass encoding using the parity check matrix. Modifying the parity check matrix to simplify encoding results in loss of coding gain and is therefore undesirable. The second is to encode LDPC codes using an algorithm which solves the simultaneous equations the parity check matrix represents.

Considering the parity check matrix as a set of linear simultaneous equations and solving for the unknown parity bits results in practical and efficient encoders for random LDPC codes. Encoders based on this method can take advantage of the sparse structure of the parity check matrix and reduce the amount of logical operations or hardware required to encode the code. Richardson and Urbanke have proposed a low complexity encoder for LDPC codes [53]. Although this algorithm is very good for irregular codes it is not efficient

for regular codes. A low complexity encoding algorithm for regular codes was therefore developed.

Using the proposed encoding algorithm an encoder architecture was also proposed and demonstrated through the implementation of an encoder for the 32,640-bit rate 239/255 code first introduced in Section 3.6. The use of the proposed algorithm reduces the number of XOR operations required to implement the dense matrix multiply operation in the algorithm by a factor of 125 when compared to the encoding algorithm derived by Richardson and Urbanke. The encoder has a throughput of 40Gbs^{-1} of uncoded systematic data, 43Gbs^{-1} of coded data, and was implemented in Agere Systems' 1.5V 0.16 μm CMOS process with 7 layers of metal in an area of 27mm^2 .

Chapter 5

Hard Decision Decoding

Hard decision decoding refers to the process of decoding a code using only thresholded, hard decisions as input to the decoder and as information exchanged in the decoder. A hard decision decoder operates on a channel modelled as a binary symmetric channel (BSC), which can be represented as shown below in Figure 5.1. The channel is characterised by its crossover probability, p_0 . When a bit is transmitted over a binary symmetric channel it is received correctly with probability $(1-p_0)$, and incorrectly with probability p_0 .

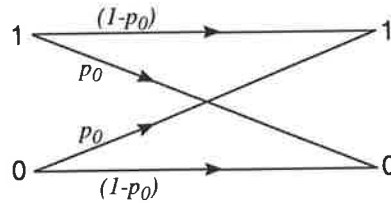


Figure 5.1 : A binary symmetric channel model with crossover probability p_0 .

All decoding algorithms for LDPC codes are iterative, with decoding proceeding using multiple iterations rather than the single pass decoding of simpler codes. When Gallager originally proposed LDPC codes he published two iterative hard decision decoding algorithms [21], which have become known as Gallager's Algorithm A and Gallager's Algorithm B.

Any code with rate equal to or greater than $3/4$ will be considered as a high rate code in the following analysis of hard decision decoding algorithms. It will be shown that existing hard decision decoding algorithms, such as Gallager's Algorithm A, Gallager's Algorithm B and expander graph decoding, do not perform well for high rate codes. The

performance of the 32,640-bit rate 239/255 code from Section 3.6 when decoded with these existing algorithms is inferior to the (255,239) Reed Solomon code. Due to the lack of a suitable hard decision decoding algorithm for high rate codes a new algorithm, relative reliability weighted decoding, is proposed. Relative reliability weighted decoding of the rate 239/255 code will be shown to provide a significant coding gain improvement of 2 dB when compared to the (255,239) Reed Solomon code at a bit error rate of 10^{-15} .

5.1 Gallager's Algorithm A

The simplest algorithm for decoding LDPC codes is Gallager's Algorithm A. When the decoder has received an entire block of data it calculates all of the parity checks the code specifies. Any bit that is involved in more unsatisfied parity checks than a fixed threshold is '*flipped*', or inverted. The parity checks are then re-evaluated, and the results again used to determine if any bits require inversion. The iterative parity and bit update is performed until either all parity checks are satisfied, indicating a valid codeword has been decoded, or until an iteration limited is reached. The threshold for bit inversion is generally made equal to the number of parity checks a bit is involved in. With this threshold a bit is only flipped if all of the parity checks it is involved in are unsatisfied.

The local parity checks can quickly use information from the entire block of data to correct any remaining incorrect bits. This can be seen when the information used in deciding the correct value for a bit is drawn in a tree structure, with the bit in question as the root of the tree. Each tier in the tree represents the information available when decoding the bit at a different iteration. Figure 6 from [22], used by Gallager to explain the algorithm is reproduced below in Figure 5.2. Gallager did not use the bipartite graph representation of low-density parity-check codes, which was introduced later by Tanner and appears in Section 2.5 [63]. The diagram used by Gallager indicates the tree of bits connected to bit d after the second decoding iteration of a (j,k) regular LDPC code. The nodes of the tree all represent bit variables in the data block. Edges in the tree represents parity checks on variables. Each parity check consists of one bit at tier t and all other bits at tier $t+1$. The difference in tiers for the bits in the parity check indicates that the value of the one bit in tier t is updated using values in the parity check group from the decoding iteration of tier $t+1$.

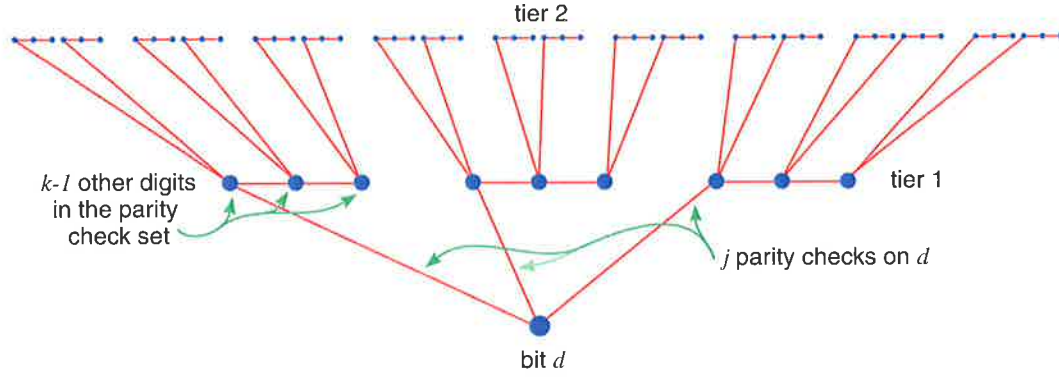


Figure 5.2 : Parity check set tree used by Gallager to describe the information flow during decoding of a $(3,4)$ regular LDPC code.

All of the bits involved in the same parity check constraints as bit d are contained in tier 1. Each bit in tier 1 is also used in other parity check constraints. The bits involved in these parity checks are connected to the first tier variables and are all located in tier 2.

In the first decoding iteration bits from tier 2 are used to attempt to correct any incorrect bits in tier 1. In the second iteration the possibly corrected bits in tier 1 are used to check the value of bit d , potentially correcting any error in the received value. An important part of the algorithm is that information only flows down the tiers, from the top of Figure 5.2 to the bottom. The information sent to bit d from the first tier does not include the value of bit d in any form.

Information exchange during decoding can also be understood by examining the bipartite graph representation of a LDPC code, shown in Figure 5.3. In the first decoding iteration of the code in Figure 5.3 each variable node, v_i sends its received value to the check nodes, c_j , connected to it via the graph edges. Each check node performs a parity check on the incoming values, or messages, arriving on the graph edges connected to it. The result of the parity check is then sent along the graph edges back to the connected variable nodes. At the variable nodes the decoded value for the current iteration is the received value, unless all parity checks connected to a variable node are not satisfied, in which case the decoded bit is ‘flipped’ and declared to be the inverse of the received value.

The values sent from the variable nodes to the check nodes at the start of the second decoding iteration are not evaluated using the same update rule as the decoded values. The

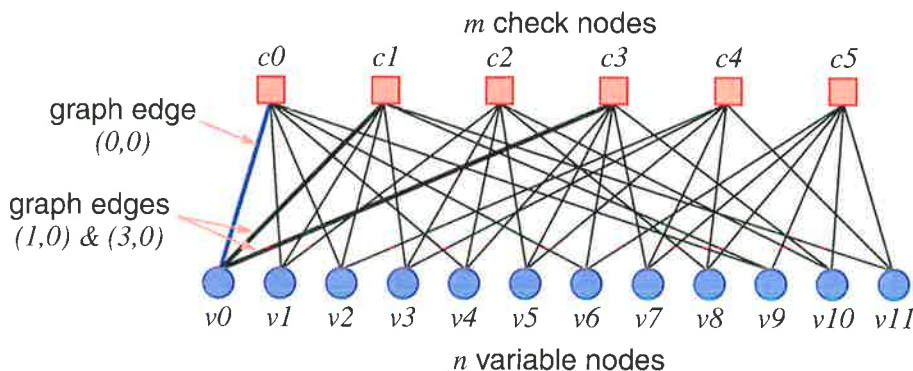


Figure 5.3 : Bipartite graph of a 12 bit $(3,6)$ regular LDPC code, or a $(3,6,12)$ code.

information arriving on an edge should not be used in calculation of the value to be sent back along the edge. In Figure 5.3 the message from variable node v_0 to check node c_0 , along edge $(0,0)$, in the second iteration is equal to the value of the bit received by v_0 , unless both parity check c_1 and c_3 were not satisfied in the first iteration, then the value sent is inverse of the received value. The update does not depend on the value of the parity check c_0 .

Before any cycles in the graph are encountered the tree of bits whose values are used in updating any element of the data block contains only distinct variable and check nodes, with no node repeated in the tree. In a decoding tree without cycles all of the information used in the update originates from uncorrelated received data bits. Once a cycle is encountered and a received bit's variable node occurs more than once in the tree the errors remaining in each tier of the decoding tree become correlated [22] and can degrade the performance of the decoder [54]. For this reason codes are designed with the largest possible girth for a given block size and code rate.

Gallager's Algorithm A is better suited for low rate codes with few set elements per row of the parity check matrix [22]. In the case of a low rate code it is highly probable that all of the parity checks that an incorrect received bit is involved in will have incorrect parity [22]. Conversely, Gallager's Algorithm A does not perform well when decoding high rate codes, such as the rate $239/255$ code from Section 3.6. The poor performance is due to each parity check involving a large number of variable bits. With weight five columns the 32,640-bit code has 79 and 80 input parity checks.

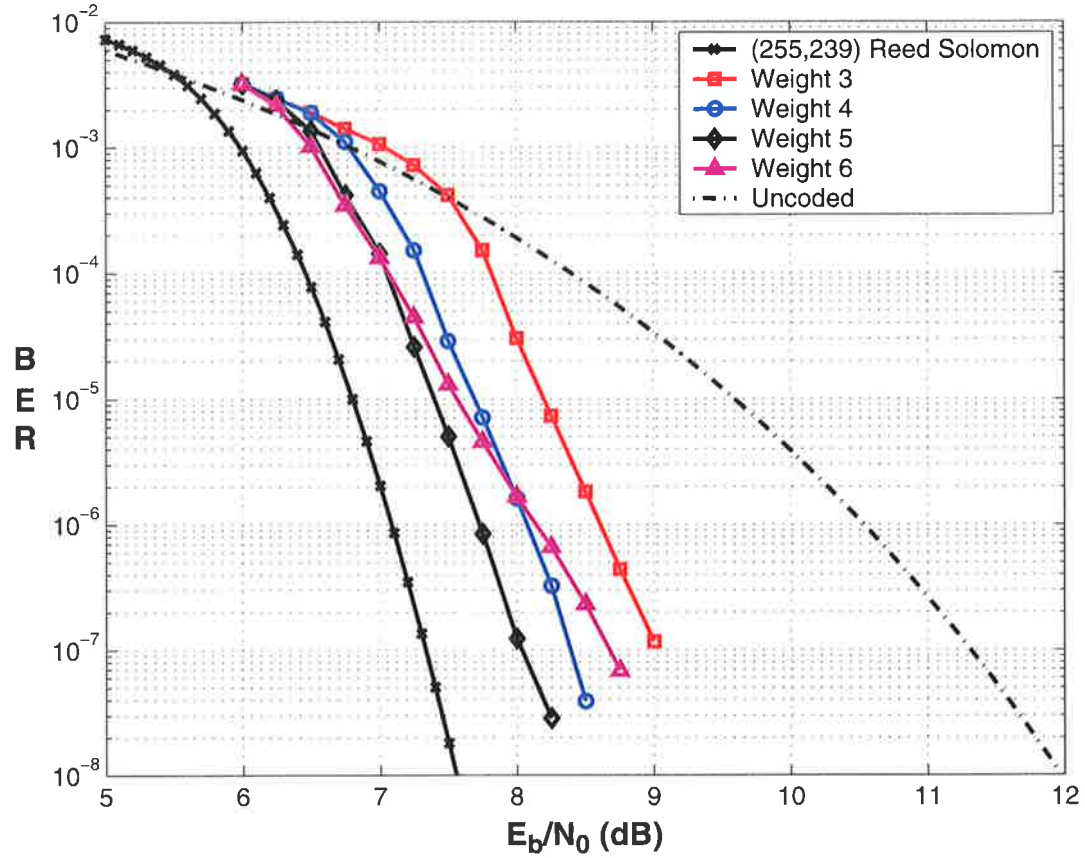


Figure 5.4 : Bit error rate versus E_b/N_0 for 32,640-bit rate 239/255 regular LDPC codes with column weights 3, 4, 5, and 6 decoded using Gallager's Algorithm A and 20 decoder iterations.

Figure 5.4 compares 32,640-bit rate 239/255 codes with column weights 3, 4, 5, and 6 decoded with Gallager's Algorithm A and the (255,239) Reed Solomon code. All of the LDPC codes decoded with Gallager's Algorithm A perform significantly worse than the (255,239) Reed Solomon code. It is therefore not possible to replace the Reed Solomon code with a low-density parity-check code decoded using Gallager's Algorithm A. All of the results in Figure 5.4 were obtained after 20 decoder iterations. After performing 20 decoder iterations any errors remaining in a block of data are highly correlated. Many of the blocks that fail to converge contain parity checks that contain two incorrect bits, therefore incorrectly satisfying the parity check. With the parity check satisfied the errors cannot be corrected. Therefore increasing the number of iterations does not result in any performance improvement.

5.2 Gallager's Algorithm B

Along with the simple algorithm described above Gallager also proposed a second more complex algorithm with better performance known as Gallager's Algorithm B. It can be observed that during the message passing algorithms iterative decoding the reliability of the information exchanged improves, since fewer and fewer bits remain in error as the algorithm corrects incorrectly received bits at each iteration. Exploiting this, it is possible to modify the threshold value required to 'flip' a bit at any iteration by making the threshold a function of the iteration number. Gallager showed that the optimal threshold value as a function of the iteration number can be found by minimising the probability of error at each iteration as a function of the initial received error probabilities [21, 22]. The result is a threshold which is a monotonically decreasing function of the iteration number.

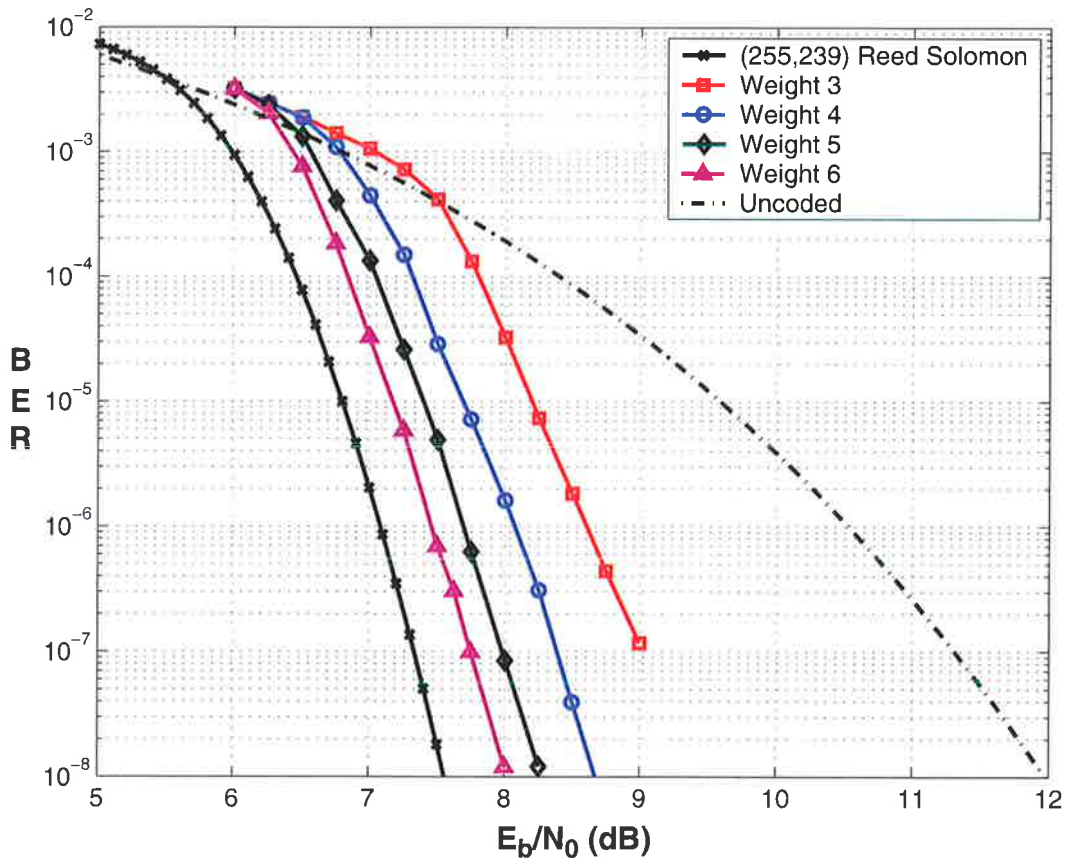


Figure 5.5 : Bit error rate versus E_b/N_0 for 32,640-bit rate 239/255 regular LDPC codes with column weights 3, 4, 5, and 6 decoded using Gallager's Algorithm B with 30 decoder iterations.

When deriving the threshold value as a function of the iteration number an infinite block length code is assumed. For a finite code the actual variance of the error probability at each iteration is not the same as predicted by the derivation for an infinite graph. The increased variance results in considerably worse performance than the derived probability of error as a function of the iteration number since the threshold values are reduced prematurely for a finite code [35]. Luby *et. al.* alleviated the effect of the finite block length by delaying the threshold reductions resulting in improved performance of the algorithm.

A comparison of 32,640-bit LDPC codes decoded using Gallager's Algorithm A, Algorithm B and the (255,239) Reed Solomon code is shown in Figure 5.6. The simulation results for Gallager's Algorithm B used 30 decoder iterations. Performing more iterations and further delaying the threshold reductions does not improve the decoders performance.

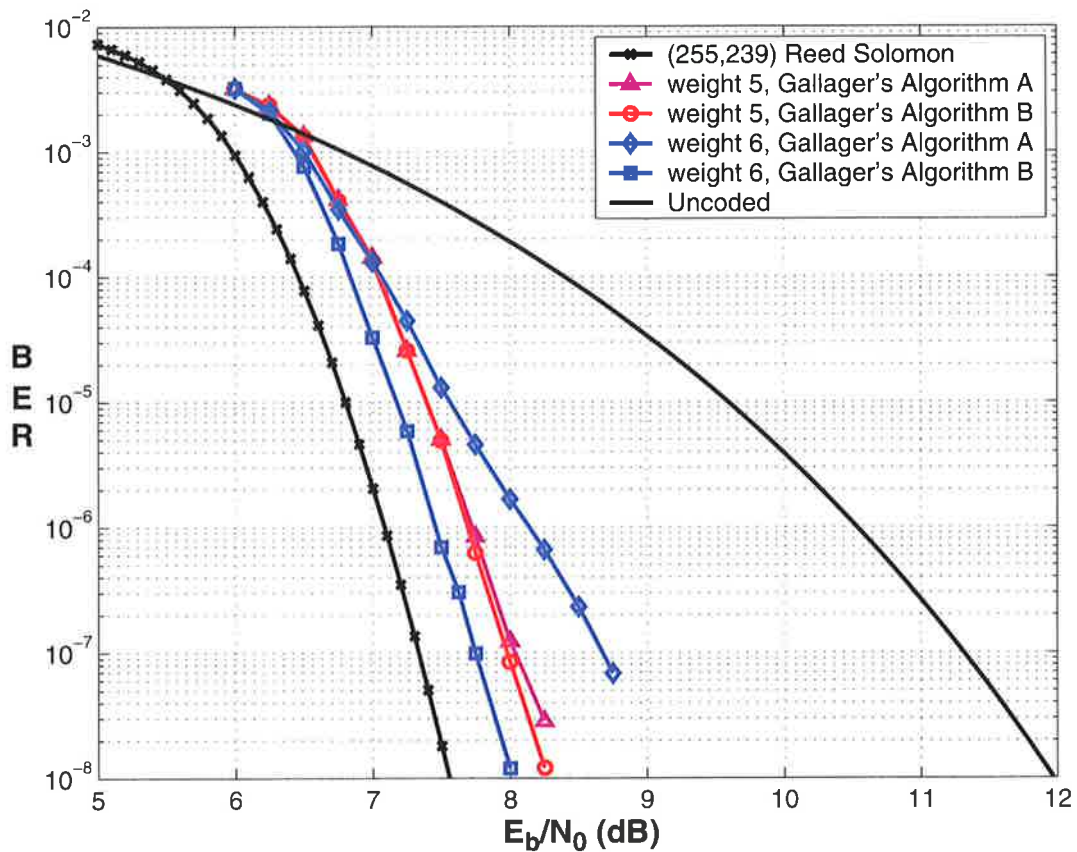


Figure 5.6 : Bit error rate versus E_b/N_0 for 32,640-bit rate 239/255 regular LDPC codes with column weights 5 and 6 decoded using 20 decoder iterations of Gallager's Algorithm A and Gallager's Algorithm B.

The bit error rate performance of the LDPC codes decoded using Gallager's Algorithm B is still worse than the bit error rate of the (255,239) Reed Solomon code. Although the weight six graph decoded with Gallager's Algorithm A performed significantly worse than the weight five graph, when decoded using Gallager's Algorithm B it is the code with the best performance, as shown in Figure 5.6. The improved performance Gallager showed for his second decoding algorithm for low rate codes is not observed for all of the high rate codes considered here with column weights less than six. The weight six graph was the only code which improved significantly when decoded using Gallager's Algorithm B compared to Gallager's Algorithm A.

5.3 Expander Graphs

Sipser and Spielman examined graph based decoding using a property of the graphs expansion, naming the method "*Expander Codes*" [57]. The expansion of a graph is a property of the expected fraction of edges crossing the boundary of any subset of nodes in a graph. Expander based graph codes include the more general graph based codes of Tanner [63, 70], and cascade graph constructions [61] in addition to low-density parity-check codes.

For a binary symmetric channel the expander based decoding algorithm is very similar to Gallager's Algorithm B. At each iteration the algorithm inverts the value of any variable node with more than some fraction β of its neighbouring parity checks unsatisfied. In the original work of Sipser and Spielman β was equal to one half [57].

5.3.1 The Binary Erasure Channel and Expander Graphs

The binary erasure channel (BEC) is a channel where the value of samples at the output of the channel are either declared to be known exactly or declared to be unknown, an erasure. Decoding a low-density parity-check code for a BEC starts by dividing all nodes in the graph into one of two subsets, known or unknown nodes. Any parity check which involves only a single unknown can be used to determine the value of the previously

unknown node. The decoder continues finding the value of unknown nodes until either all of the values are known, or no parity check remains with only a single unknown.

The expansion property of a graph has been used to show that LDPC codes can achieve channel capacity on a binary erasure channel as the block length of the code tends to infinity [35, 52].

5.3.2 The Binary Symmetric Channel and Expander Graph Decoding of High Rate Codes

The expansion property of graphs features in the proofs of many theorems regarding LDPC codes and the ability to achieve channel capacity on a BEC. However, this does not translate into good performance when decoding high rate codes transmitted over a binary symmetric channel with an expander graph based decoder. The performance of an expander graph based decoder for a BSC is between the performance of Gallager's Algorithm A and Algorithm B for a the 32,640-bit weight 6 graph examined. Again this algorithm is not suitable for use as a decoder for a low-density parity-check code replacing the (255,239) Reed Solomon code for a binary symmetric channel.

5.4 Gallager's Algorithm, Expander Graphs and Erasures

Once a decoder has iterated a number of times greater than half of the girth of the underlying graph the messages exchanged by the decoder become highly correlated. This can be seen by examining the ratio of parity checks with a single incorrect bit to parity checks with two incorrect bits. As decoding progresses this ratio falls significantly. The remaining errors in the data become grouped with the probability of multiple errors occurring in the same parity check equation increasing. The correlation of the error locations often results in the decoder being unable to correct the remaining errors. When this happens the decoder has found a local minimum from which it cannot escape and converge to the final correct codeword. Failure to converge happens frequently for high rate codes, resulting in error floors, as the uncorrectable error sequences are frequently of low weight¹.

The cause of the problem is the particular location of the error events in relation to the graphs structure and the resultant ratio of parity checks with one and two incorrect inputs. This can be also be considered from the point of view of the expansion of the group of nodes in error. The expansion of the group is insufficient to correct the errors, too few correct messages are received by the nodes in error.

One method proposed to improve convergence is to change decoding algorithms after a fixed number of decoding iterations [34, 52]. The different decoding algorithm is unlikely to have a local minima with the same error pattern that had caught the initial decoding algorithm and the decoder can then converge to the correct codeword from an error pattern the first decoder could not correct. Changing decoding algorithms during decoding has been examined by Luby *et. al.* [34], and by Richardson and Urbanke [52].

Burshtein & Miller used the expansion of a graph to prove that swapping decoding algorithms from a parallel expander graph decoder to a serial form of Gallager's Algorithm A after the number of errors remaining has been reduced to a small fraction guarantees convergence. However, due the finite block length of practical codes a fraction of the node sets do not have the required expansion to ensure convergence, resulting in error sequences which are unable to be corrected by the decoder [9]. The finite graph of practical codes greatly reduces the practical use of many theoretical results for LDPC codes. Combining or swapping algorithms for decoding high rate LDPC codes does not result in sufficient performance to replace the (255,239) Reed Solomon code.

5.5 Relative Reliability Weighted Decoding

To overcome the inherent problems of finite graphs, expander graph decoders and Gallager's Algorithm A and Algorithm B, a new decoding algorithm is proposed here. The algorithm has been named relative reliability weighted decoding (RRWD) and uses the reliability of the parity result to infer greater information transfer between the check nodes and

-
1. A low weight error sequence refers to a small number of errors in the received data.

variable nodes. The increased information transfer is then used to improve the performance of the decoding algorithm.

None of the existing hard decision decoding algorithms have resulted in a performance improvement compared to the (255,239) Reed Solomon code when decoding a 32,640-bit low-density parity-check code. As a result a new decoding algorithm optimised for high rate codes is developed here. Although it has been shown that a binary symmetric channel approximation for fiber optic communications results in an underestimation of the channel capacity [10], a more complex model will not be considered here as the potential gain in performance is small in comparison to the added complexity of implementing a decoder for an asymmetric channel.

5.5.1 Information Exchange

If a decoder operates on random data the messages sent from the variable nodes to the check nodes in the first decoder iteration will be equally distributed and carry no further information content other than their value. However, this does not apply to the result of the parity check performed on the variable messages. The probability that all messages are received correctly at the input to a check node with k inputs and that the parity is therefore correctly satisfied is:

$$P(\text{All correct}) = (1 - p_0)^k, \quad p_0 = \text{channel crossover probability} \quad (5.1)$$

The probability of one input being inverted and in error at the input to a parity check and hence that the parity check has incorrect parity is:

$$P(1 \text{ inv}) = \binom{k}{1} \cdot (1 - p_0)^{k-1} \cdot p_0 = k \cdot (1 - p_0)^{k-1} \cdot p_0 \quad (5.2)$$

The probability of two inputs to a parity check being incorrect and the parity check being satisfied by incorrect inputs is:

$$P(2 \text{ inv}) = \binom{k}{2} \cdot (1 - p_0)^{k-2} \cdot (p_0)^2 = \frac{k \cdot (k-1)}{2} \cdot (1 - p_0)^{k-2} \cdot (p_0)^2 \quad (5.3)$$

If the channel crossover probability, p_0 , is small then:

$$(1 - p_0) \gg p_0 \quad (5.4)$$

which results in the probability of no errors being significantly greater than one error, which in turn is also significantly greater than the probability of two errors:

$$P(\text{All correct}) \gg P(\text{one incorrect}) \gg P(\text{two incorrect}) \gg \dots \quad (5.5)$$

The 32,640-bit rate 239/255 code with columns of weight five from Section 3.6 has an average row weight of 79.6875. The probability of no input errors in an 80 input parity check versus SNR is shown in Figure 5.7, and is compared to the probabilities of one and two errors in Figure 5.8. It will be shown in Section 5.5.2 that the intended target operating region of the decoder is an input signal-to-noise ratio of approximately 6.5dB, as highlighted in Figure 5.7 and Figure 5.8.

The probability of a parity check being satisfied is given by:

$$P(\text{Parity} = 0) = (1 - p_0)^k + \binom{k}{2} \cdot (p_0)^2 \cdot (1 - p_0)^{k-2} + \dots + (p_0)^k \quad (5.6)$$

if k is even, or by

$$P(\text{Parity} = 0) = (1 - p_0)^k + \binom{k}{2} \cdot (p_0)^2 \cdot (1 - p_0)^{k-2} + \dots \quad (5.7)$$

$$\dots + k \cdot (1 - p_0)(p_0)^{k-1}$$

if k is odd. The probability of the parity check being satisfied can alternatively be written as:

$$P(\text{Parity} = 0) = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{i} \cdot (1 - p_0)^{k-2i} \cdot (p_0)^{2i} \quad (5.8)$$

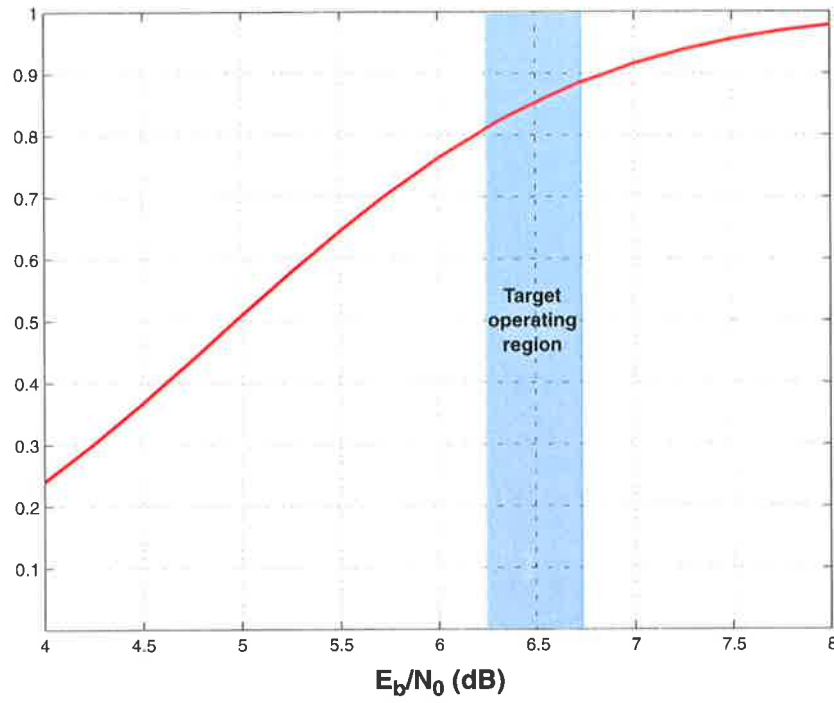


Figure 5.7 : Probability of an 80-input parity check having no incorrect inputs versus E_b/N_0 in dB.

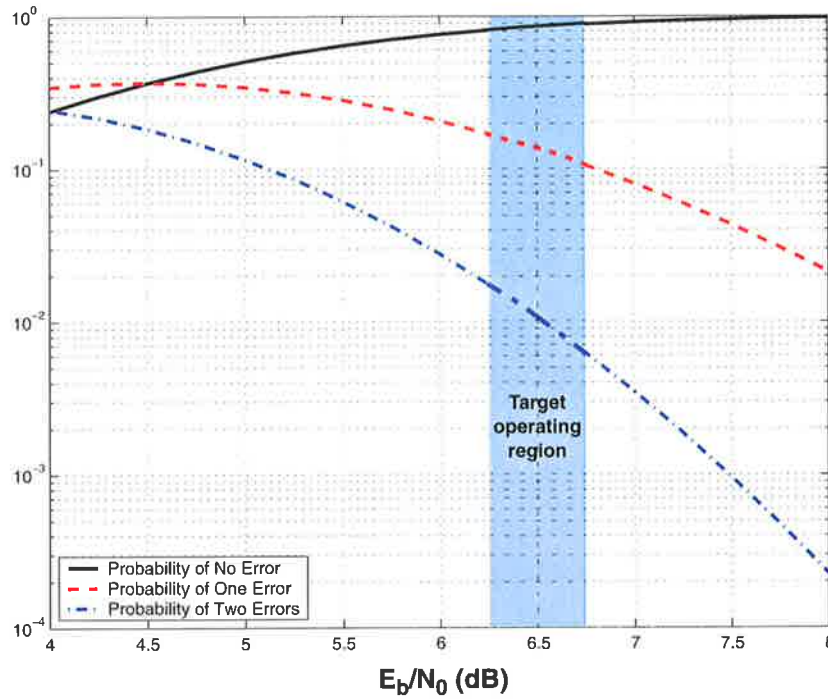


Figure 5.8 : Probabilities of an 80-input parity check having zero, one or two incorrect inputs versus E_b/N_0 in dB.

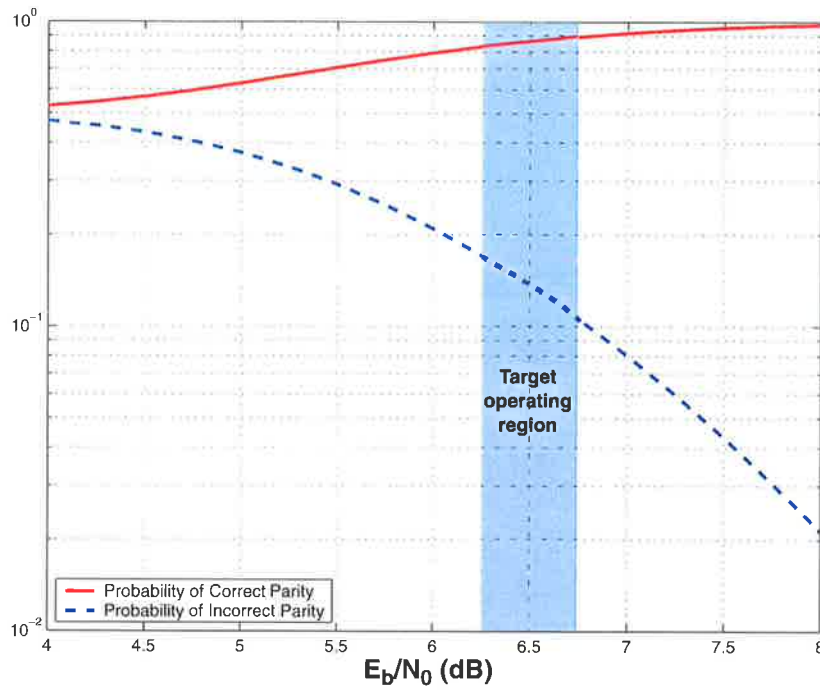


Figure 5.9 : Probabilities of an 80-input parity check being satisfied or unsatisfied versus E_b/N_0 in dB.

The probability that an 80 input parity check is satisfied or unsatisfied in the rate 239/255 code with column weight five is shown in Figure 5.9.

When a check node sends the parity check result to a variable node it is most likely satisfied, and with a very high probability it is correctly satisfied, with no errors in the input bits for the check node. The voting schemes of Gallager's algorithms and the expander graph decoding algorithms may therefore be considered biased towards the unsatisfied parity checks, as they are both given an equally weighted vote. If the probability of the message being correct is included in weighting the message votes at the variable node, a correct parity result can be considered to be more reliable and of a higher weight vote than an incorrect parity result.

Each iteration of the decoder reduces the number of errors remaining in the partially decoded block. With less incorrect values remaining the results of the parity checks become more reliable, since the probability that a parity check contains multiple incorrect inputs becomes decreasingly likely. Therefore as the iterative decoding of a low-density parity-check code proceeds the reliability of the messages sent from the parity check nodes

to the variable nodes increases. It is therefore possible to weight the information from check nodes used in the decoder updates with increasing reliability compared to the received data from the channel, or equivalently to reduce the weighting of the received values compared to the messages. This is the motivation for reducing the threshold for bit inversion in Gallager's Algorithm B.

5.5.2 RRWD Algorithm for a 32,640-Bit Rate-239/255 LDPC Code

A modified hard decision decoding algorithm can be constructed in which the weight of check message votes in the variable node updates is a function of the parity check result and the iteration number. From an implementation perspective it is easier to scale the vote of the received bit as a function of the iteration number rather than scale the weight of all of the parity check messages.

The maximum number of decoder iterations for the 32,640-bit code was chosen to be fifty one. Using more iterations does not significantly improve the performance of the decoder. The choice of this number also reflects a constraint imposed by the architecture used to implement the decoder, described in Chapter 7, where data is loaded into the decoder using a 640-bit bus and shift registers or scan chains of $(32640 / 640) = 51$ elements.

To simplify the implementation of this decoder the weights of the check node message votes were restricted to be small three or four bit integers. The performance of the decoder was compared when using various ratios of the check message weight for satisfied and unsatisfied parity checks. Based on the simulation results it was decided to weight the message ± 3 if the parity check was satisfied and ± 2 if the parity check was unsatisfied. Where the message is weighted:

- +3 if the variable message to the check node was a zero and the parity check was satisfied,
- -3 if the variable message to the check node was a one and the parity check was satisfied,

- -2 if the variable message to the check node was a zero and the parity check was not satisfied, and
- +2 if the variable message to the check node was a one and the parity check was not satisfied.

The received value for the variable nodes was given a weight of ± 8 initially and reduced linearly down to ± 1 from the first to last decoder iteration. This reduction is similar to the reduction of the threshold in Gallager's Algorithm B and reflects the increasing reliability of the messages compared to the received value as the decoder iterates.

The algorithm described so far is initialised by propagating the value of the received bit each variable node represents to all of the parity checks. The result of the parity checks are used at the variable nodes in a weighted summation to compute the group sum,

$$S_j = w_{iter} \times (1 - 2y_j) + \sum_{i|h_{i,j}=1} (3 - p_i) \cdot (-1)^{(v_{ij} \oplus p_i)} \quad (5.9)$$

Where w_{iter} is the weight of the received value at the current iteration, y_j is the received value of the j^{th} variable $y_j \in \{0, 1\}$, p_i is the parity of the i^{th} row, v_{ij} was the message value sent from variable node j to parity check i and $h_{i,j}$ is an element of the parity check matrix. The sign of S_j is taken as the estimate of the j^{th} decoded bit in the current iteration, unless S_j is zero, in which case the received value is used as the current estimate of the decoded bit. The messages from the variable node to the check node are updated using

$$S_{ij} = w_{iter}(1 - 2y_j) + \sum_{l|h_{l,j}=1, l \neq i} (3 - p_l) \cdot (-1)^{(v_{lj} \oplus p_l)} \quad (5.10)$$

which is the group sum excluding the input from the edge currently being updated. The updated message, v_{ij}' , is taken as the sign of S_{ij} unless the sum is zero in which case the received value is used as the next message value.

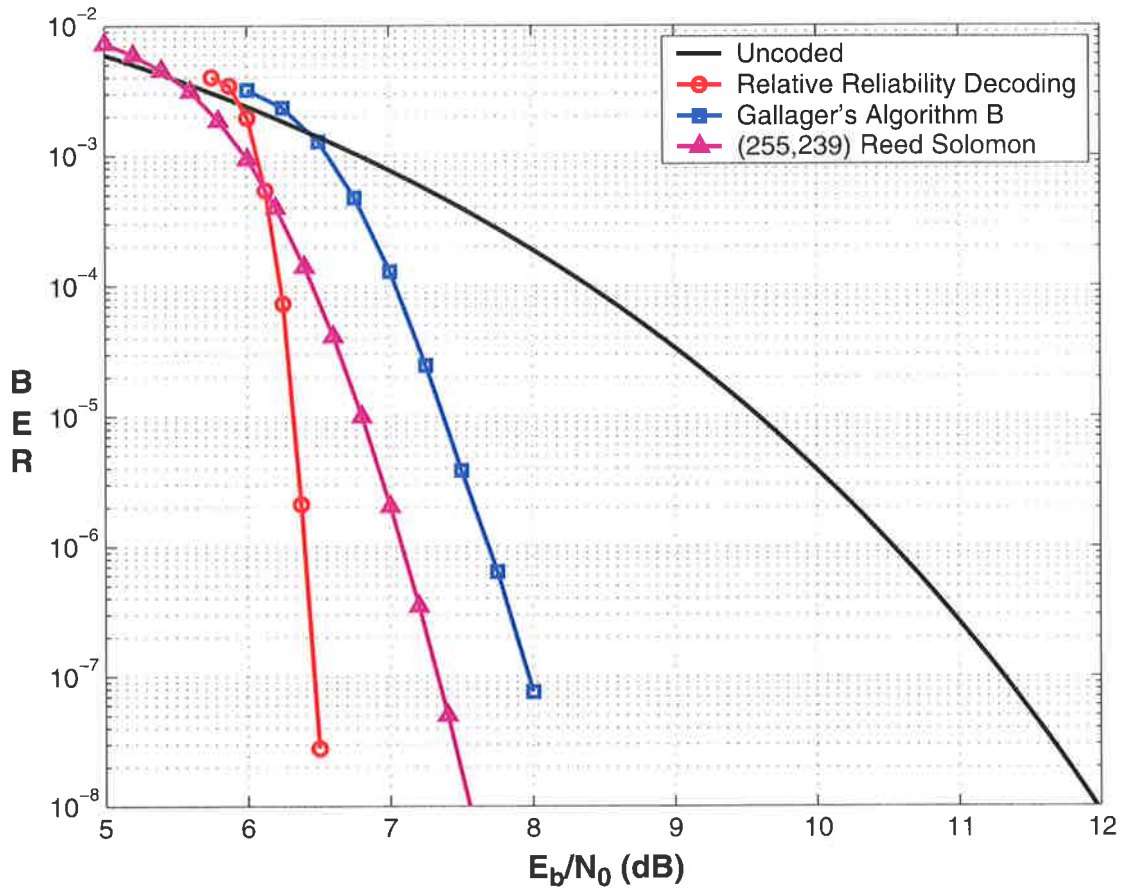


Figure 5.10 : Simulation performance of the optimised 32,640-bit rate 239/255 code using weight 5 columns decoded with 30 iterations of Gallager's Algorithm B and 51 iterations of relative reliability weighted decoding.

The performance of this decoder is compared with Gallager's Algorithm B in Figure 5.10. Relative reliability weighted decoding provides 1.75dB of improved performance when compared to the (255,239) Reed Solomon code at an output bit error rate of 10^{-15} and in the following sections it will be further modified to increase the coding gain by an additional 0.25dB.

5.5.3 Mitigating the Effect of Graph Cycles

After a number of decoder iterations equal to half of the girth of the graph has been completed any remaining errors in the partially decoded block become correlated. The optimised 32,640-bit code with weight five columns contains cycles of length six. Therefore after only three decoder iterations the information used in performing variable node

updates has become correlated. The small number of iterations for which the errors in the decoder remain uncorrelated results in the decoder being unable to correct some error events involving very few bit errors. The correlation of error locations causes the decoder to become caught in a loop following a cycle in the underlying graph where the remaining errors cannot be corrected.

The decoder can be considered as a damped feedback system with the received bit weighting acting as the damping factor. If the weighting of the received bit is never reduced to a low weight and is instead held at high weights the system is over-damped and often will not converge and decode a valid codeword. Conversely, if the weighting of the received bit is reduced rapidly to a value which over estimates the reliability of the messages the system becomes under-damped and unstable. When the weighting of the received bit is reduced too rapidly the decoder becomes completely unstable and introduces errors. It is possible to make the decoder become so unstable as to introduce errors on fifty percent of the codeword.

However, instability introduced by under damping a feedback system can be very useful in escaping local minima which have trapped a system and are preventing convergence to the global minimum. An example is the perturbation of a system during simulated annealing to escape local minima which have trapped a gradient descent algorithm. In the case of the relative reliability weighted decoder this property can be used to escape error sequences which trap the decoder in cycles it cannot break. Rather than reducing the weighting of the received bit monotonically, it is possible to oscillate the weighting and alternate decoding iterations between under and over-damped.

The received bit weights as a function of the iteration number in Figure 5.11 were obtained through the collection of a large number of received vectors which the decoder was unable to correct. The error events were sorted according to the number of bits which were in error. The weights as a function of the iteration number were changed and the minimum number of errors the decoder could correct using the weights was recorded. The number of errors remaining as a function of decoder iteration was also examined to determine the effect of changing the weighting of the received bit in a particular iteration. The final weights in Figure 5.11 were used in a simulation of more than 12 million blocks of data

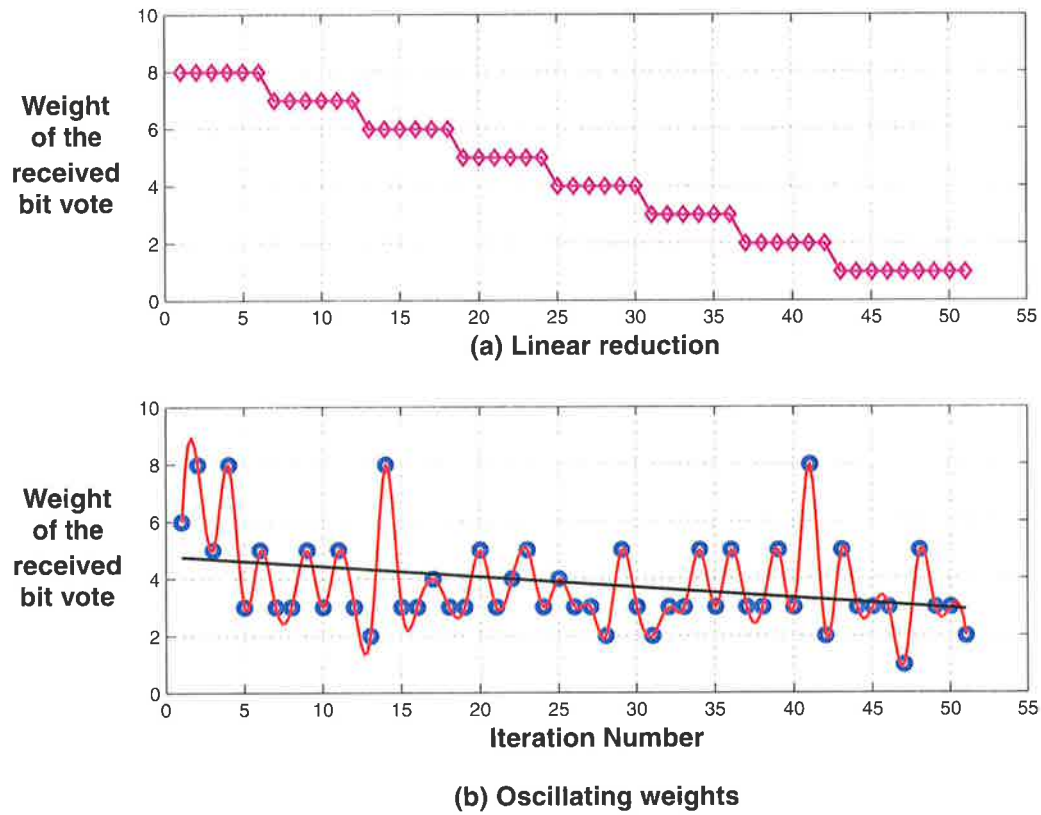


Figure 5.11 : Weight of the received bits vote as a function of iteration number, for
 (a) the original linear reduction and (b) oscillating weights.

with the decoder never failing to correct an error event involving fewer than 98 bit errors in a block. Using the initial linear reduction of weights the decoder made errors with as little as 55 bit errors. Using Gallager's Algorithm B to decode the 32,640-bit optimised weight five code the decoder failed to converge with error events containing as few as 13 errors in a block.

Oscillating the vote weights of the received bits, as shown in Figure 5.11, is not enough to escape all forms of cycles which can trap the decoder in local minima. As the noise in the decoder becomes correlated the ratio of parity checks with one incorrect input to parity checks with two incorrect values becomes significantly lower than expected with the original random channel samples. The result of this is that a large number of incorrectly satisfied parity checks are being assigned a high weight vote since the parity is satisfied by two incorrect inputs. This results in a decoder with oscillating received bit weights which performs slightly worse than one with linearly decreasing weights, as shown in Figure 5.12.

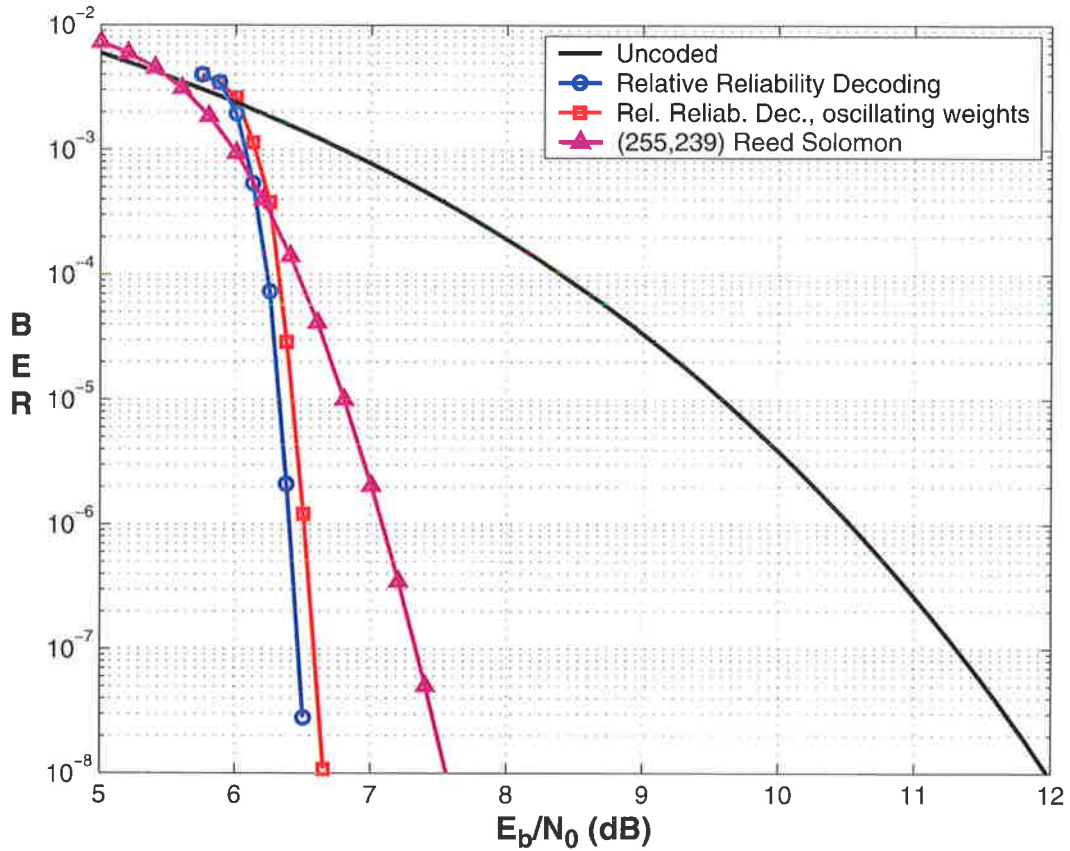


Figure 5.12 : *Simulation performance of the optimised 32,640-bit rate 239/255 code with weight 5 columns decoded and 51 iterations of relative reliability weighted decoding using linear and oscillating received bit weights.*

One method of breaking cycles resulting from incorrect weighting of the messages with correct parity and incorrect parity is to change the ratio of the message weighting for correct parity and incorrect parity results. The ratio of three to two works very well and corrects a large fraction of the initial errors in the block. It would therefore seem good to keep this initial ratio and examine changing the ratio towards the end of decoding, a method somewhat similar to changing decoding algorithms as examined in Section 5.4.

Two possible changes of the received bit weighting were examined for breaking cycles caused by parity checks with two incorrect inputs. The first was to remove the weighting ratio completely and weight all messages with a weight of three. The second was to reverse the weighting, and weight correct messages with a weight of two and incorrect messages with weight three. The second change causes a great instability in the feedback

system and as such is only suitable for use when very few errors remain in the block being decoded.

When implementing this decoder, distributing control signals to indicate that the relative weights of the messages should be changed is inefficient. It was decided that the method used for indicating the change should be based on the weight of the received bits, which already must be distributed as a control signal. The change implemented was to make the message weights equal to three for both correct and incorrect parity when the received bit weight is two and to reverse the weights when the received bit weight is one. In Figure 5.11 it can be seen that the check message weights are only reversed once during decoding at iteration number 47 when the received bit weight is one. The swapping of check message weights is only performed after the decoder has had many iterations to reduce the number of remaining bit errors to be a very small fraction of the total bits in the

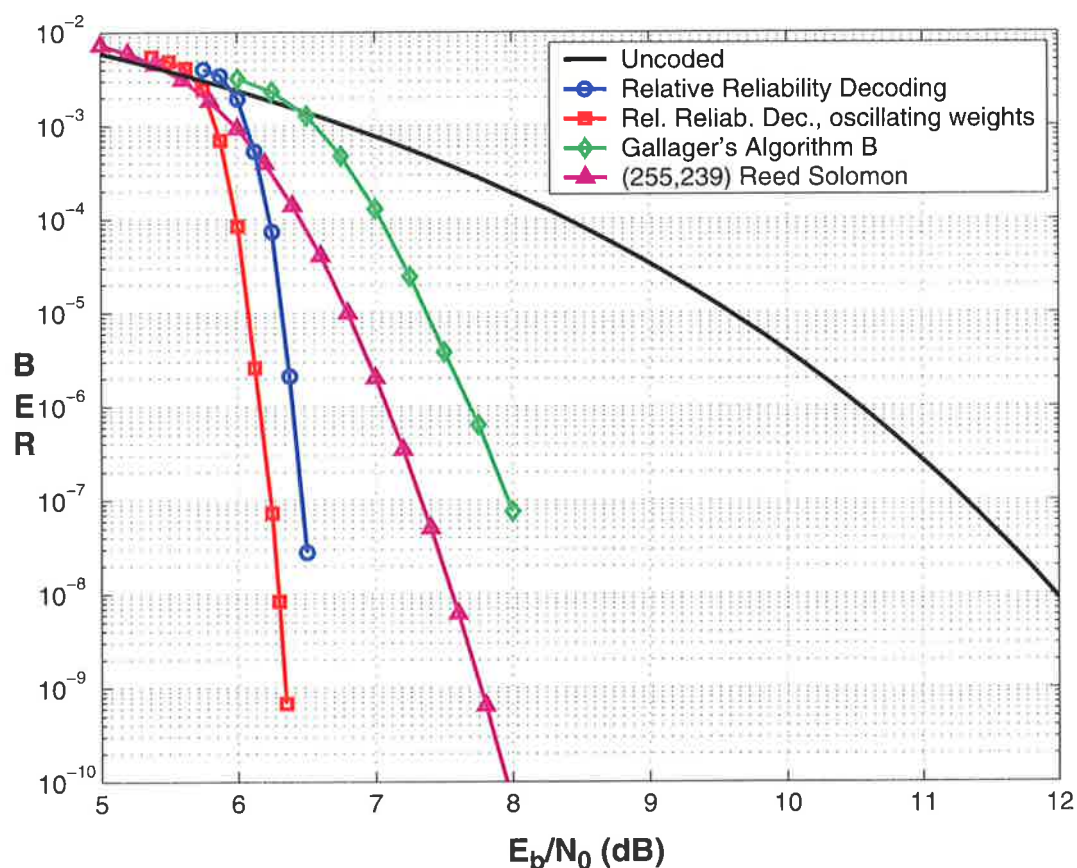


Figure 5.13 : Performance of the optimised 32,640-bit rate 239/255 code using decoded with 51 decoder iterations of relative reliability weighted decoding using linear and oscillating received bit and parity message weights.

code. The performance of the decoding algorithm with oscillating received bit weights and changes in the check parity message weights is shown in Figure 5.13. Combining both oscillating received bit weights and changing parity message weights results in a decoder with an increase in coding gain of 0.25dB compared to the original RRWD algorithm proposed in Section 5.5.2 and 2dB compared to the (255,239) Reed Solomon code at an output bit error rate of 10^{-15} .

Implementing the changes to gain the performance increase of relative reliability weighted decoding does not require the distribution of more control signals or information between the check nodes above that required to implement Gallager's Algorithm B, which also requires a threshold value distributed as control for the variable node update. When comparing the implementation complexity of Gallager's Algorithm B and RRWD it can be seen that they both require the distribution of the same number of control signals to indicate the threshold value or received bit vote weight.

5.5.4 Summary of the Relative Reliability Weighted Decoding Algorithm

1. Initialise all variable nodes with the received value corresponding to the bit they represent. Send these initial values from the variable nodes to the check nodes.
2. Perform a parity check at all parity check nodes using the incoming variable messages. At each check node send the result of the parity check to all connected variable nodes.
3. At the variable nodes calculate the sum of the weighted check messages and received bit. The weight of the check messages is given in Table 5.1. The weight of the received bits is given in Table 5.2. If the received bit was zero the value it contributes to the sum is the value of the weight, if it was one the value it contributes is minus one times the weight. The sign of the summation can be used as the estimate of the decoded bit value for the variable node in the current iteration, except in the case the group sum is zero, then value of the received bit is used as the decoded bit.
4. For each edge connected to a variable node subtract the incoming message value from the group sum formed in step 3 and use the sign of the result as the next mes-

sage value for the edge. If the result of the group sum minus the message value is zero, the received bit is used as the next message value.

5. Repeat steps 2 to 4 until all parity checks are satisfied or a maximum number of decoder iterations is reached.

The check message weights used at the variable nodes to decode the 32,640-bit rate 239/255 code as shown in Figure 5.13 are given in Table 5.1 and the weight of the received bit in the sum is given in Table 5.2.

Variable Message	Check Message (parity)	Received Bit Weight	Check Message Value at Variable Node
0	0	not 1	+3 = 011
0	0	1	+2 = 010
0	1	greater than 2	-2 = 110
0	1	1 or 2	-3 = 101
1	1	1 or 2	+3 = 011
1	1	greater than 2	+2 = 010
1	0	1	-2 = 110
1	0	not 1	-3 = 101

Table 5.1: Message weights for relative reliability weighted decoding.

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
Weight	6	8	5	8	3	5	3	3	5	3	5	3	2	8	3	3	4	3	3	5	3	4	5	3	4	
Iteration	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Weight	3	3	2	5	3	2	3	3	5	3	5	3	3	5	3	8	2	5	3	3	3	1	5	3	3	2

Table 5.2: Received weight versus iteration number.

Implementing the calculation of the three bit binary message values, $m_2m_1m_0$, from Table 5.1 requires very few gates. Denoting the variable message sent to the check node as v , the parity as p , the received weight is one or two by r_1 and r_2 then the sign bit of the message is the exclusive-or of the outgoing variable message and the incoming parity result from the check node:

$$m_2 = v \oplus p \quad (5.11)$$

The least significant bit of the message value is not set if the parity check result is zero and the received bit weight is one or if the parity check is not satisfied and the received bit weight is not one or two,

$$m_0 = (r_1 \oplus p) \vee (r_2 \wedge p) \quad (5.12)$$

The second bit of the message value is zero if the sign bit and the least significant bit are both one,

$$m_1 = \overline{m_2 \wedge m_0} \quad (5.13)$$

5.5.5 Performance Comparison of 32,640-Bit Rate 239/255 Codes with Relative Reliability Weighted Decoding

A comparison of the simulated performance of the 32,640-bit rate 239/255 codes with column weights three, four, five and six decoded using the relative reliability weighted decoding algorithm is shown in Figure 5.14.

The weight three code, and to a small extent the weight four code, decoded using relative reliability decoding show an error floor effect similar to turbo codes [4]. Initially the slope of the bit error rate versus signal to noise ratio is steep and at higher signal to noise ratios the slope is lower, forming the error floor. The codes with column weight five and six do not exhibit any reduction of the bit error rate slope for the bit error rates simulated.

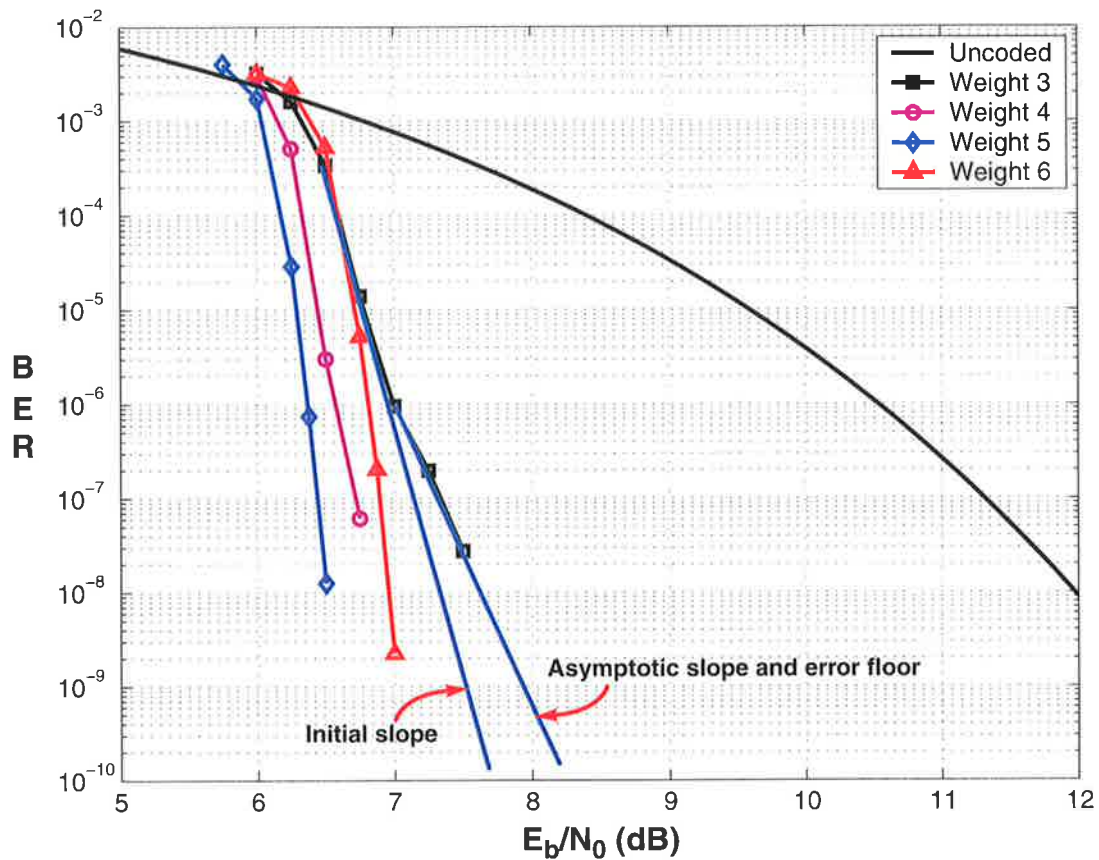


Figure 5.14 : Simulation performance of 32,640-bit rate 239/255 codes with column weights 3,4,5 and 6 using 51 iterations of relative reliability weighted decoding.

5.6 Summary

Gallager's Algorithm's A and B, expander graph decoding and combinations of them have been examined as potential decoding algorithms for a 32,640-bit rate 239/255 LDPC code to be used in a fiber optic transceiver. The decoding algorithms have been shown to have worse performance than the (255,239) Reed Solomon codes and do not perform adequately for this application

By taking into consideration the relative probability of a parity check being satisfied or unsatisfied and the relative reliability of the messages in the decoder compared to the reliability of the received data, a new decoding algorithm called relative reliability weighted decoding has been developed with significantly better performance than existing hard decision decoding algorithms.

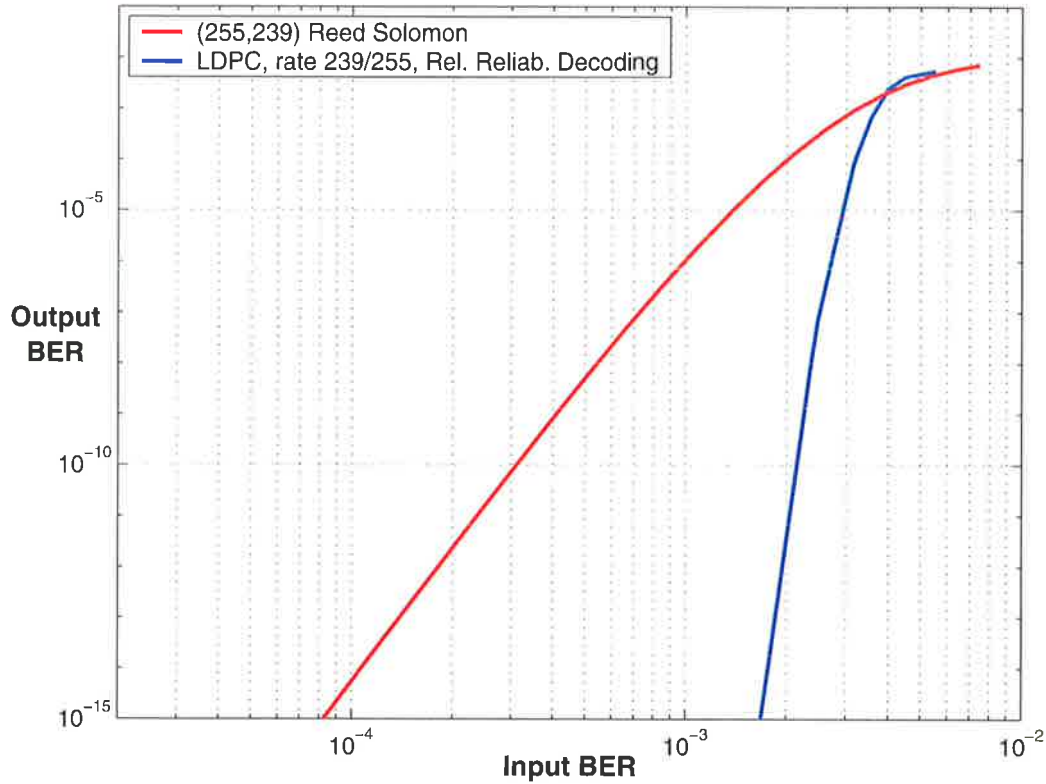


Figure 5.15 : Output versus input bit error rates for the (255,239) Reed Solomon code and the 32,640-bit LDPC code decoded using 51 iterations of RRWD.

Relative reliability weighted decoding requires no additional information exchange between the variable and check nodes of the bipartite graph of a decoder for low-density parity-check codes and does not require the distribution of control information above that required to implement Gallager's Algorithm B. The only change required to implement the relative reliability weighted decoder is a change to the update rules used at the variable nodes in the decoder.

The large performance improvement obtained by using relative reliability weighted decoding of LDPC codes compared to the (255,239) Reed Solomon code can be seen in Figure 5.15. To obtain an output bit error rate of 10^{-12} the LDPC decoder can correct a raw input bit error rate ten times higher than the Reed Solomon code. The coding gain at an output bit error rate of 10^{-15} of the LDPC code when decoded using the relative reliability decoding algorithm proposed here is 8dB, compared to a coding gain of 6dB for the (255,239) Reed Solomon code, representing an improvement of 2dB.

Chapter 6

Soft Decision Decoding

Soft decision decoders use sample reliability information from the channel they are operating on. Included in this class of decoder are those operating on real valued input or using quantised input from an analog-to-digital converter. The reliability information improves the performance of soft decision decoders compared to hard decision decoders.

Soft decision decoders often work with log-likelihood ratios (LLR's). The log-likelihood is defined as the logarithm of the probability a received sample is due to the transmission of a one divided by the probability the sample is due to the transmission of a zero:

$$\Lambda = \log\left(\frac{P(y|x = 1)}{P(y|x = 0)}\right) \quad (6.1)$$

Soft decision LDPC decoders pass reliability information along with sign information as messages between the variable and check nodes of the decoder. The reliability values of the channel samples are used with the message reliabilities to form soft updates of the message values for use in the next iteration. Soft decision decoders can also produce a reliability measure for the decoded bits.

Two soft decision decoding algorithms for low-density parity-check codes will be examined, Gallager's soft decision algorithm and the sum-product algorithm. Two important variations of the sum-product algorithm, the min-sum algorithm and MacKay and Neal's derivation of the sum-product algorithm, will also be reviewed. A fixed point simplification of the sum-product algorithm to enable efficient implementation will be proposed in this chapter. The fixed point algorithm is demonstrated using the 1024-bit rate 1/2 code

described in Section 3.7. The performance of this algorithm when exchanging 4-bit messages is compared with a sum-product decoder implemented using double precision floating point accuracy. It will be shown that the performance of 64 iterations of the fixed point decoding algorithm is only 0.2dB worse than performing 1000 decoder iterations of the floating point sum-product decoder.

6.1 Gallager's Probabilistic Soft Decision Decoding

Algorithm

Gallager used the parity check set tree¹ for a digit d in the received block to derive the probability that the transmitted digit in position d is a one, conditional on the set of received symbols $\{y\}$ and on the event S that the transmitted digits satisfy the j parity check equations, each with k inputs, on the digit d [22]. Gallager denoted this as

$$Pr[x_d = 1 | \{y\}, S] \quad (6.2)$$

The conditional probability for a regular (j,k) LDPC code can be expressed using Theorem 4.1 from [22]:

Theorem 6.1: [22] *Let P_d be the probability that the transmitted digit in position d is a 1 conditional on the received digit in position d , and let P_{il} be the same probability for the l^{th} digit in the i^{th} parity-check set of the first tier in Figure 6.1. Let the digits be statistically independent of each other, and let S be the event that the transmitted digits satisfy the j parity-check constraints on digit d . Then*

$$\frac{Pr[x_d = 0 | (\{y\}, S)]}{Pr[x_d = 1 | (\{y\}, S)]} = \frac{1 - P_d}{P_d} \prod_{i=1}^j \left[\frac{1 + \prod_{l=1}^{k-1} (1 - 2P_{il})}{1 - \prod_{l=1}^{k-1} (1 - 2P_{il})} \right] \quad (6.3)$$

Proof: [22], page 41.

1. This diagram was first described in Section 5.1 as part of the derivation of Gallager's hard decision decoding algorithms.

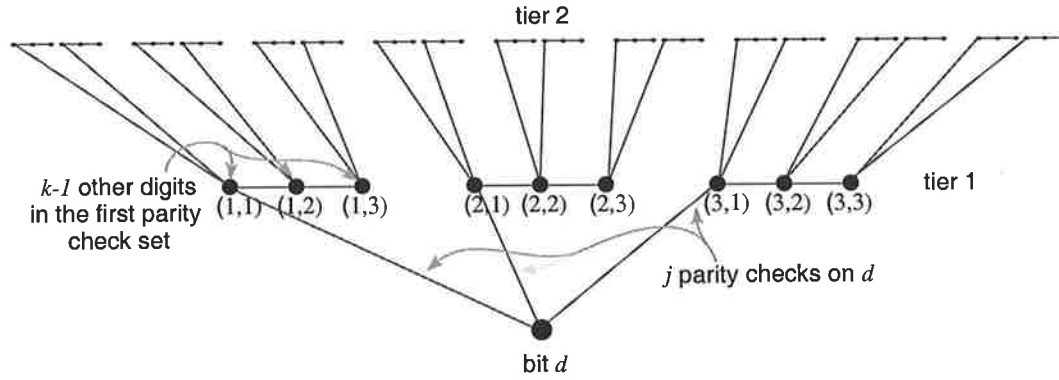


Figure 6.1 : Parity-check set tree of a (3,4) regular LDPC code [22].

Due to the complexity of the equation for the conditional probability Gallager showed that the probability can be extended using an iterative technique, rather than through an explicit calculation for more than one tier of the decoding tree of each bit.

Gallager's argument considers the two-tier decoding tree case: Calculate the probability that each of the digits in the first tier is a one conditional on the received digits in the second tier using Theorem 6.1:, equation (6.3). The only modification to the equation is that the first product is taken over $(j-1)$ terms since the parity check set containing the digit d is not included. These conditional probabilities can then be used in equation (6.3) to find the conditional probability that digit d was a one.

The decoding procedure for the entire block can be stated as:

1. For each variable node with column weight j and each combination of $(j-1)$ parity check sets containing the variable node, that is for each graph edge connected to each variable node, use equation (6.3) to calculate the probability of a transmitted one conditional on the received symbols in the $(j-1)$ parity check sets. There are j different probabilities associated with each variable node, one for each edge connected to the node.
2. The probabilities found in the previous iteration are used in equation (6.3) to calculate a second set of probabilities. The probability to be associated with any one digit in the computation of another digit, d , is the probability from the previous iteration omitting the parity check set containing d .
3. Repeat step 2.

If the decoding is successful the probabilities associated with each digit approach zero or one, depending upon the transmitted digit, as the number of iterations increases [22]. Equation (6.3) can only be used to exactly calculate the conditional probabilities while the independence assumption of Theorem 6.1: remains valid. The assumption breaks down once the parity check set tree for any node is not a perfect tree and contains cycles or repeated nodes [22]. Each tier of the tree contains $(j - 1) \times (k - 1)$ more nodes than the previous tier. Therefore the number of decoding iterations for which the conditional probability can be calculated exactly is small. However, the iterative update can be continued ignoring the lack of independence since the effect is, in general, small [22, 52].

Gallager observed an important property of this decoding algorithm, namely the computation per digit per iteration is independent of the block length [22]. He also showed that the average number of decoder iterations required to decode the code is $O(\log(\log(n)))$, where n is the block length of the code.

To simplify the computation of the conditional probabilities Gallager re-defined the conditional probability update of equation (6.3) in terms of log-likelihoods. Defining

$$\alpha_d \beta_d = \ln \left(\frac{1 - P_d}{P_d} \right) \quad (6.4)$$

$$\alpha_{il} \beta_{il} = \ln \left(\frac{1 - P_{il}}{P_{il}} \right) \quad (6.5)$$

$$\alpha'_{il} \beta'_{il} = \ln \frac{\Pr[x_d = 0 | \{y\}, S]}{\Pr[x_d = 1 | \{y\}, S]} \quad (6.6)$$

where $\alpha \in \{-1, 1\}$ is the sign and β is the magnitude of the log-likelihood ratio [22]. Then the iterative conditional probability update of equation (6.3) becomes

$$\alpha'_{il} \beta'_{il} = \alpha_d \beta_d + \sum_{i=1}^j \left\{ \left(\prod_{l=1}^{k-1} \alpha_{il} \right) f^{-1} \left[\sum_{l=1}^{k-1} f(\beta_{il}) \right] \right\} \quad (6.7)$$

where

$$f(x) = f^{-1}(x) = \ln \frac{e^x + 1}{e^x - 1} \quad (6.8)$$

Gallager's soft decision decoding algorithm can be applied to either a binary symmetric channel or an additive white gaussian noise channel using the appropriate initial reliability measure, P_d , for each.

The algorithm can be implemented as a bipartite graph decoder with functional nodes representing variables and parity check constraints. Decoding starts when the variable nodes are initialised with the channel sample reliability and sign, $\alpha_d \beta_d$ of the received bit they represent. During the first iteration of the decoder the channel reliabilities, with signs, are propagated from each variable node to the connected parity check nodes. At the check nodes the parity check result is formed,

$$p = \prod_{l=1}^k \alpha_l \quad (6.9)$$

over the entire k inputs of the parity constraint. If the decoder is operating on hard inputs from a binary symmetric channel the sign bits are mapped such that

$$\{0, 1\} \Rightarrow \{1, -1\} \quad (6.10)$$

Each variable node requires the parity all other connected variables in the check imply it should take,

$$\hat{\alpha}_l = p \oplus \alpha_l, \quad l \in [1, k] \quad (6.11)$$

An associated reliability for the parity is also calculated for each connected variable node using the group reliability,

$$\Gamma = \sum_{l=1}^k f(\beta_l) \quad (6.12)$$

and the contribution to the group reliability of each connected variable node,

$$\hat{\beta}_l = f[\Gamma - f(\beta_l)] , \quad l \in [1, k] \quad (6.13)$$

where the variables $\hat{\beta}_l$ and $\hat{\alpha}_l$ denote values sent back from the check nodes to the variable nodes. The updated check node reliability and parity values, $\hat{\beta}_l$ and $\hat{\alpha}_l$, are then propagated to the corresponding connected variable nodes. The variable nodes can then complete the decoding iteration by updating the parity and reliability values for the decoded data variables and the graph edges connected to the variable nodes. Initially a group log-likelihood is calculated,

$$\alpha' \beta' = \alpha_d \beta_d + \sum_{i=1}^j \hat{\alpha}_i \hat{\beta}_i \quad (6.14)$$

The group log-likelihood values are the conditional probabilities at the current iteration and the sign can be used as the current estimate of the transmitted bit. The log-likelihood values to be propagated from the variable nodes to the check nodes in the next decoder iteration can then be found using the group log-likelihood and message value arriving from each of the connected parity check nodes,

$$\alpha_i \beta_i = \alpha' \beta' - \hat{\alpha}_i \hat{\beta}_i , \quad i \in [1, j] \quad (6.15)$$

Decoding continues until all of the group parity check constraints, equation (6.9), are satisfied and equal to one or a maximum number of decoder iterations is reached.

6.2 The Sum-Product Algorithm

Gallager's soft decision decoding algorithm is a special case of the sum-product algorithm. The sum-product algorithm is itself a special case of the *BCJR* algorithm of Bahl, Cocke, Jelinek and Raviv, also called the *forward-backward* algorithm or *maximum a-posteriori probability* (MAP) algorithm [67, 1]. Gallager's soft decision decoding algorithm is a special case of the algorithm. Kschischang, Frey and Loeliger showed the sum-product algorithm acting on a bipartite graph can describe many signal processing algorithms with marginal a-posteriori probabilities, either exactly or approximately [28, 27]. The algorithms include Viterbi decoding, turbo decoders, MAP decoders, Kalman Filters, and Perl's belief propagation algorithm for Bayesian networks and some FFT algorithms.

MacKay and Neal derived a soft decision decoding algorithm based on the sum-product algorithm [36]. The notation used in the algorithm begins with denoting the set of bits, l , which participate in parity check m by

$$\mathcal{L}(m) \equiv \{l \mid h_{ml} = 1\} \quad (6.16)$$

where h_{ml} is an element of the parity check matrix. Similarly, the set of checks in which bit l participates is denoted as

$$\mathcal{M}(l) \equiv \{m \mid h_{ml} = 1\} \quad (6.17)$$

and the set $\mathcal{L}(m)$ excluding bit l is given by

$$\mathcal{L}(m) \setminus l \quad (6.18)$$

The decoding algorithm consists of iteratively updating two quantities q_{ml} and r_{ml} associated with each non zero element of the parity check matrix. The values q_{ml}^x denotes the probability that bit l of the transmitted codeword, $\mathbf{x}$, was the value x conditional on the parity checks bit l is involved in other than m . The value r_{ml}^x denotes the probability of parity check m being satisfied if bit l of $\mathbf{x}$ is considered fixed at x and the other bits in the

parity check have a separable distribution given by the probabilities $\{q_{mi} | i \in \mathcal{L}(m) \setminus l\}$ [36].

The decoder is initialised using the probabilities $p_l^0 = P(x_l = 0)$ and $p_l^1 = P(x_l = 1) = 1 - p_l^0$ given the received data. For all of the set elements of $\mathbf{H}$ the variables q_{ml}^0 and q_{ml}^1 are initially set to p_l^0 and p_l^1 respectively.

MacKay and Neal define the algorithm in terms of a horizontal and vertical update on the parity check matrix, equivalent to the check node and variable node updates in a bipartite graph implementation of Gallager's soft decision decoding algorithm.

The horizontal update of the parity check reliabilities is performed using all variables involved in the parity check, $l \in \mathcal{L}(m) \quad \forall m$. For each variable involved in the parity check two probabilities r_{ml}^0 and r_{ml}^1 are calculated. The probabilities calculated are the probability of the current row parity, z_m , when $x_l = 0$ and $x_l = 1$ respectively. The probabilities are conditional on the other bits in the parity check, which have probabilities $\{q_{mi}^0, q_{mi}^1\}$. The probability updates are defined as

$$r_{ml}^0 = \sum_{\{x_i | i \in \mathcal{L}(m) \setminus l\}} P(z_m | x_l = 0, \{x_i | i \in \mathcal{L}(m) \setminus l\}) \prod_{i \in \mathcal{L}(m) \setminus l} q_{mi}^{x_i} \quad (6.19)$$

and

$$r_{ml}^1 = \sum_{\{x_i | i \in \mathcal{L}(m) \setminus l\}} P(z_m | x_l = 1, \{x_i | i \in \mathcal{L}(m) \setminus l\}) \prod_{i \in \mathcal{L}(m) \setminus l} q_{mi}^{x_i} \quad (6.20)$$

MacKay and Neal suggest calculating the conditional probabilities by considering the parity check as a Markov chain with states 0 and 1, starting in state 0 and undergoing transitions corresponding to the addition of the values x_i , with transition probabilities $\{q_{mi}^0, q_{mi}^1\}$ [36]. This can be implemented using the maximum a-posteriori probability (MAP) algorithm first derived by Bahl, Cocke, Jelinek and Raviv [1]. MacKay and Neal

also derive an alternative update rule for the parity constraints using the product of differences, $\delta q_{ml} = q_{ml}^0 - q_{ml}^1$, and $\delta r_{ml} = r_{ml}^0 - r_{ml}^1$ then

$$\delta r_{ml} = (-1)^{z_m} \prod_{i \in \mathcal{L}(m) \setminus l} \delta q_{mi} \quad (6.21)$$

Reference [36] contains a complete derivation of equation (6.21). Using equation (6.21) the values r_{ml}^0 and r_{ml}^1 can be determined using

$$r_{ml}^0 = (1 + \delta r_{ml}) / 2 \quad (6.22)$$

and

$$r_{ml}^1 = (1 - \delta r_{ml}) / 2 \quad (6.23)$$

The vertical update step computes

$$q_{ml}^0 = \alpha_{ml} p_l^0 \prod_{i \in \mathcal{M}(l) \setminus m} r_{il}^0 \quad (6.24)$$

$$q_{ml}^1 = \alpha_{ml} p_l^1 \prod_{i \in \mathcal{M}(l) \setminus m} r_{il}^1 \quad (6.25)$$

where α_{ml} is a normalisation constant such that the two probabilities, q_{ml}^0 and q_{ml}^1 , sum to one.

The conditional probabilities of each bit in the data block can also be calculated using

$$q_l^0 = \alpha_l p_l^0 \prod_{m \in \mathcal{M}(l)} r_{ml}^0 \quad (6.26)$$

and

$$q_l^1 = \alpha_l p_l^1 \prod_{m \in \mathcal{M}(l)} r_{ml}^1 \quad (6.27)$$

which are used to form a tentative decoding, $\hat{x}$, with $x_l = 1$ if $q_l^1 > 0.5$. Again the probabilities have been normalised using the quantity α_l .

Mackay and Neal's decoding algorithm requires more arithmetic operations than Gallager's algorithm, due to the separation of the probability updates for q_{ml}^0 and q_{ml}^1 and the normalisation required at the end of each iteration.

6.3 Implementation of Soft Decision Decoders

Implementing soft decision decoders requires the representation of continuous real valued probabilities by finite precision fixed point numbers. Incorrect quantisation of variables in a decoder can lead to significant performance loss and numerical instability. It is therefore essential that the dynamic range of the quantisation is efficiently utilised.

Gallager's algorithm is more easily implemented than MacKay and Neal's algorithm since it requires fewer arithmetic operations per iteration and does not require a normalisation step. Division or normalisation of fixed point numbers can lead to large round off errors and a loss of performance. This may be a contributing factor in the error floors of the fixed point decoders simulated by Ping and Leung [48]. Using 12-bit quantisation their implementation of MacKay and Neal's algorithm resulted in an error floor for which the error rate did not improve at all with increasing signal to noise ratio once the bit error rate was approximately 5×10^{-6} . Although an alternative function for the iterative update is derived in this paper, reducing the required number of quantisation bits to 6, the error floor is still present due to the decoding algorithm.

Zhang, Wang and Parhi simulated the performance of soft decision decoders based on MacKay and Neal's decoding algorithm [71]. In the paper they concluded 6-bit quantisation was required for passing messages between the variable and check nodes.

Another problem faced when implementing soft decision decoders is the complex arithmetic operations required. The hyperbolic trigonometric functions used in the check node reliability update of Gallager's algorithm are very complex to implement unless the number of quantisation bits is small. MacKay and Neal's algorithm is also extremely complex to implement since there are many multiplications and normalisation divisions. The complexity of these operations may be reduced by considering performing part of the decoding in the log domain, similar to the log-MAP implementation of MAP decoders, where multiplications become additions and divisions become subtractions.

Another approach was introduced by Forney who derived a method of translating the constraints of a code between the code and its dual code [19]. Using the dual code constraints the parity check can be updated using only additions. The complex operation required to convert between the two code representations is a Fourier transform and inverse Fourier transform at the input and output of the check node.

If the messages exchanged between check and variable nodes are quantised to only a few bits the simplest method of reducing the number of complex operations is to consider performing Gallager's parity check update in the logarithmic domain [26, 22, 12, 52]. With the parity updates implemented in the log domain Gallager's soft decision decoding algorithm can be implemented using only additions, subtractions and merged hyperbolic tangent, exponentiation and logarithm functions. Equation (6.8) can be written as

$$f(\beta) = \ln \frac{e^\beta + 1}{e^\beta - 1} = \ln \left[\coth \left(\frac{\beta}{2} \right) \right] \quad (6.28)$$

and directly implemented as combinatorial logic when β is quantised to a few bits, as shown later in Section 6.4 for a related function. With the hyperbolic trigonometric functions and logarithms involved in equation (6.28) implemented efficiently by combinatorial logic Gallager's soft decision decoding algorithm can be implemented using only additions, subtractions and combinatorial logic.

When the sum-product algorithm probability update for the check node is evaluated using the MAP, BCJR or forward-backward algorithm, the update rule becomes

$$\beta'_j = 2 \operatorname{atanh} \left\{ \prod_{m \in \mathcal{M}(l) \setminus j} [\tanh(\beta_m / 2)] \right\} \quad (6.29)$$

The reliability update can be performed in the log domain as

$$\beta'_j = 2 \operatorname{atanh} \left\{ \exp \left(\sum_{m \in \mathcal{M}(l) \setminus j} \ln [\tanh(\beta_m / 2)] \right) \right\} \quad (6.30)$$

The parity check reliability update function of Gallager's soft decision algorithm and the sum-product algorithm using the forward-backward algorithm are equivalent. One difficulty exists when implementing Gallager's update in software due to a discontinuity in $f(\beta)$ at $\beta = 0$ when the inner function is evaluated before taking the logarithm. This is an important numerical problem when writing a software simulator for the algorithm and must be treated as an exception.

6.3.1 The Min-Sum Algorithm

The min-sum algorithm is a simplification of the sum-product algorithm. The intention of the algorithm is to reduce the arithmetic complexity of implementing a soft decision decoder. It will be shown though that there is a significant performance loss due to simplification of the algorithm. Further, it will be shown that implementing the min-sum algorithm is no simpler than implementing the sum-product algorithm. The parity check log likelihood calculation for the sum-product algorithm, equation (6.30) can be simplified using

$$\exp(x + y) \approx \exp(\max(x, y)) \quad (6.31)$$

Approximating the arguments to the exponential function results in the update rule

$$\beta'_j \approx \min_{m \in \mathcal{M}(l) \setminus j} \left\{ \beta_m \right\} \quad (6.32)$$

The use of this approximation is the basis of the Viterbi algorithm [67], and the max-log-MAP simplification of the MAP algorithm [45, 46, 64]. The simplification can be improved using a correction function [46], since

$$\ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{|x-y|}) = \max(x, y) + f(z) \quad (6.33)$$

where $z = |x - y|$ and $f(z) = \ln(1 + e^z)$. The correction term, $f(z)$, can be approximated to yield the desired decoder performance. The approximation can be applied recursively to the summation in equation (6.30) in any order [46, 64].

Using the simplest approximation, equation (6.32), results in a parity reliability update which is the minimum reliability of all other bits in the parity update. The decoding algorithm can then be described as:

1. Initialise all variable nodes with their associated log-likelihood of the channel sample and all outgoing variable messages in the graph with the same values.
2. Perform the parity update for each check message as the parity result for the check excluding the incoming parity of the variable message for each graph edge. Update the reliability associated with each check message using the minimum reliability of all incoming variable messages in the parity check excluding the variable message of the edge being updated.
3. Update the current estimated decoded bit as the sign of the sum of all check messages arriving at a variable node and the initial channel sample log-likelihood associated with the node. For each edge update the variable message as the sum of all other incoming check messages and the initial channel sample log-likelihood associated with the node.
4. Repeat steps 2 and 3 until all parity checks are satisfied, or a maximum number of decoder iterations is reached.

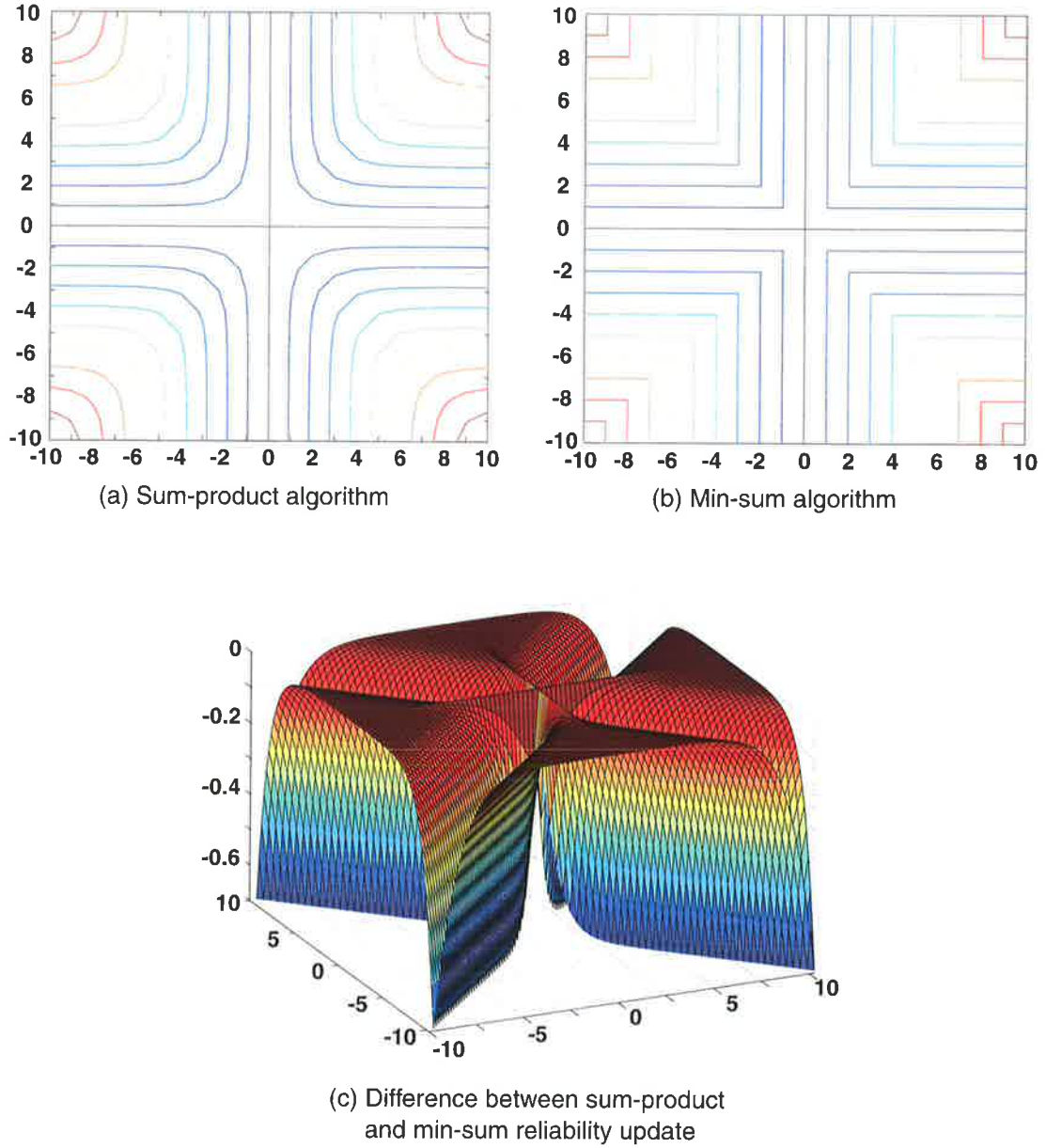


Figure 6.2 : Plot of reliability contours for the output message of a degree 3 check node using (a) sum-product algorithm, (b) min-sum algorithm as a function of the two other input reliabilities and (c) reliability of the sum-product update minus the reliability of the min-sum update.

The parity check reliability update using the min-sum algorithm is compared to the sum-product algorithm update in Figure 6.2². The contours in Figure 6.2 (a) and (b) are of equal magnitude. When the magnitude difference between the two reliabilities is large or one input is almost zero there is very little difference between the algorithms, however

when the difference is small and neither input is approximately zero the min-sum algorithm overestimates the reliability. When the magnitude of the two input reliabilities are large and equal the min-sum algorithm overestimates the reliability by a factor of $\ln(2) \approx 0.693$

6.4 Implementation of a 1024-Bit Rate 1/2 Soft Decision Decoder

A fixed point soft decision decoding algorithm for the 1024-bit rate 1/2 code designed in Section 3.7 is designed here. The code was investigated for use with wireless data transmission. For data transmission a *packet*, *frame* or *block* of data cannot be used if it has any bit errors, therefore the packet error rate (PER) is considered in this section as the design performance measure. The target packet error rate of interest here is one error in a hundred packets, a high error rate due the low signal to noise ratios frequently encountered in wireless communications. It is unlikely that an implementation of a decoder can perform an arbitrary number of decoder iterations, therefore an upper bound on the number of iterations is required.

The performance a sum-product decoder implemented using double precision floating point numbers with 64 and 1000 decoder iterations is compared in Figure 6.3. At a PER of 10^{-2} the performance degradation when performing 64 decoder iterations rather than 1000 is less than 0.08dB. Therefore the upper bound on the number of iterations that will be considered for the decoder implementation will be 64. All further simulation of the 1024-bit rate 1/2 code will therefore be performed using a maximum of 64 decoder iterations.

-
2. Section (a) and (b) of Figure 6.2 are similar to Figure 5-5 in [11], where it was used in relation to the derivation of decoding thresholds for the sum-product and min-sum algorithms.

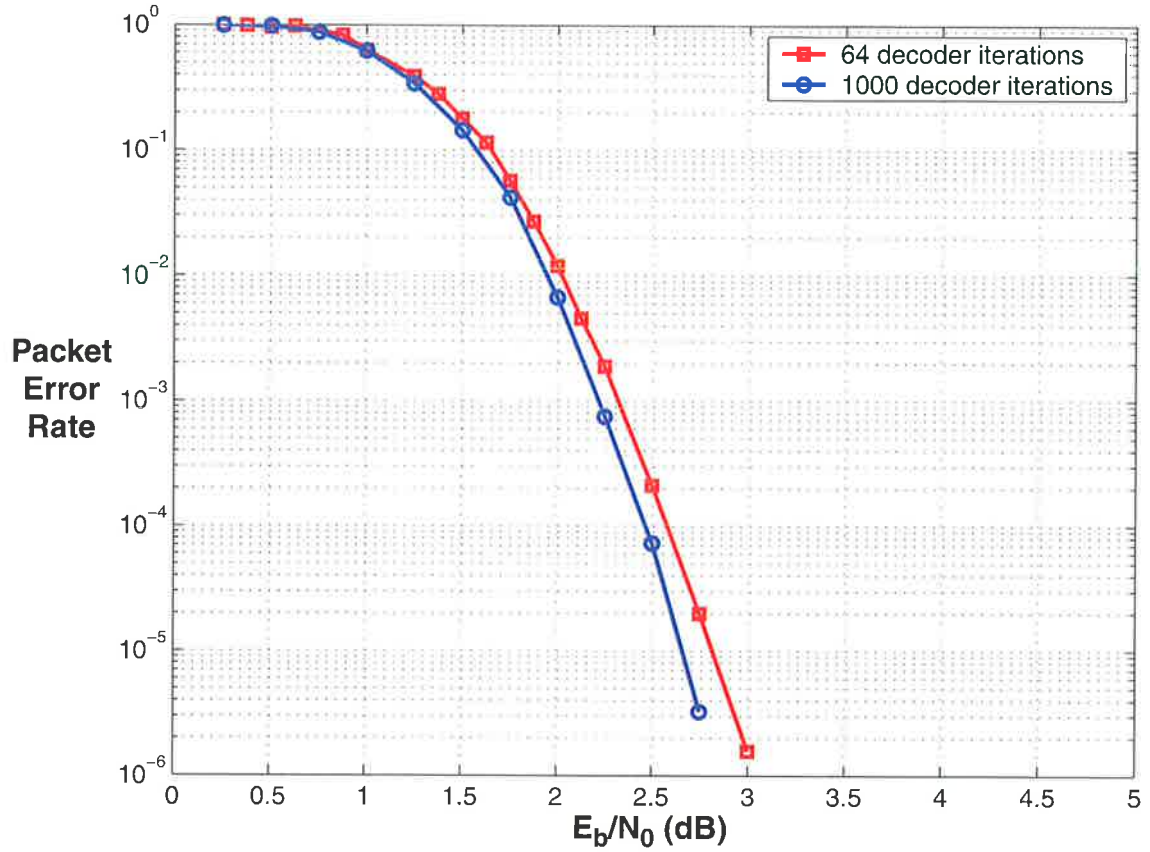


Figure 6.3 : Packet error rates for a 1024-bit rate 1/2 code decoded using 64 and 1000 iterations of a sum-product soft decision decoder.

6.4.1 Performance of a 1024-Bit Rate 1/2 Code Decoded Using a Min-Sum Decoder

The performance of a min-sum and sum-product decoder, both performing 64 decoder iterations, are compared in Figure 6.4. When the 1024-bit rate 1/2 code is decoded using the min-sum decoding algorithm the loss of the coding gain at a packet error rate of 10^{-2} is 0.48 dB compared to a sum-product decoder. The loss of coding gain is significant, especially considering the complexity of implementing a min-sum decoder compared to a sum-product decoder is very similar. Both decoders require a similar number of addition operations, one finding a group sum and set of differences, the other finding the two minimum reliability inputs from the group of inputs. It was therefore decided to implement a sum-product decoder rather than a min-sum decoder.

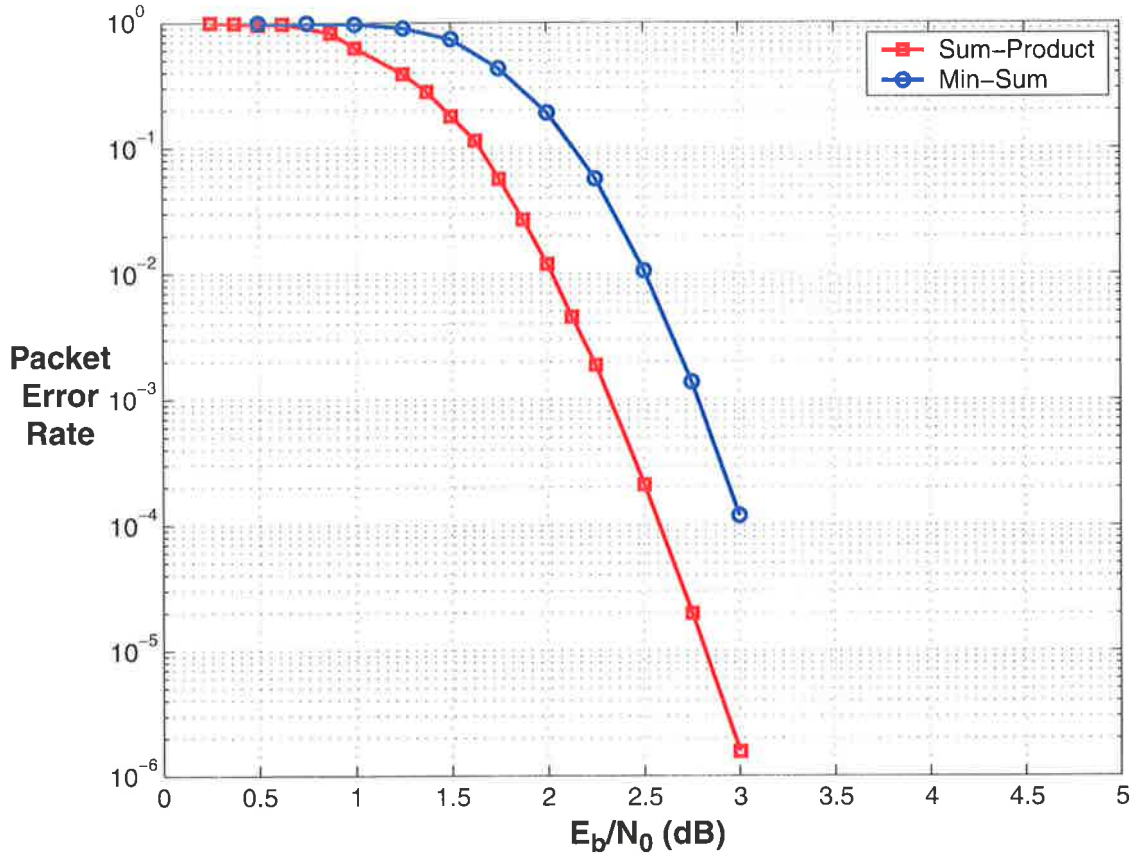


Figure 6.4 : Packet error rates for a 1024-bit rate 1/2 code decoded using 64 iterations of a sum-product and min-sum soft decision decoder.

6.4.2 Performance of a 1024-Bit Rate 1/2 Sum-Product Soft Decision Decoder with 4-Bit Messages

A fixed point decoder was designed using 4-bit initial channel sample log-likelihood values in sign-magnitude representation, $y_i \in \{-7, \dots, \pm 0, \dots, 7\}$. The received samples used as decoder input can be normalised such that the samples efficiently utilise the available dynamic range. The fixed point decoding algorithms considered here all used a fixed channel reliability scaling factor, up to an input signal-to-noise ratio of 3dB, such that the input to the decoder was given by:

$$y_i = \text{sgn}(\hat{y}_i) \min(\lfloor 4.5 \times |\hat{y}_i| \rfloor, 7) \quad (6.34)$$

where $\hat{y}_i$ is the received value arising from a transmitted value $x_i = \pm 1$. The

scaling factor, 4.5, can be varied to reflect the channel reliability if it is known or has been estimated. In all of the simulation results presented here the scaling factor is fixed and does not assume the channel reliability or signal to noise ratio is known, except for simulations with a signal to noise ratio of 3dB or greater, when the scaling factor of 4.5 in equation (6.34) is *reduced* to 4.0. However, all floating point simulations use the exact standard deviation of the noise to scale the channel reliability.

The decoding algorithm chosen for implementation was the sum-product algorithm implemented in the log domain and exchanging log-likelihood messages. The algorithm is the same as that investigated by Richardson *et. al.* [52, 51, 54] and Chung *et. al.* [13, 14, 12]. Using log-likelihood messages greatly simplifies the implementation of the decoder and removes multiplications in the variable node update and the probability normalisation of MacKay and Neal's algorithm.

The check nodes implement the reliability update from equation (6.30), which is

$$\beta'_{ij} = 2 \operatorname{atanh} \left\{ \exp \left(\sum_{m | h_{im} = 1; m \neq j} \ln [\tanh (\beta_{im} / 2)] \right) \right\} \quad (6.35)$$

for the parity check corresponding to row i of the parity check matrix and using the 3 bit reliability values from the variable node messages as inputs to the function

$$\ln [\tanh (\beta_{im} / 2)] \quad (6.36)$$

approximated assuming input reliabilities of

$$\frac{3}{4}(\beta_{im} + 1) \text{ , } \beta_{im} \in \{0, \dots, 7\} \quad (6.37)$$

resulting in a scaled input message mapping function of

$$-124.792 \times \ln \left[\tanh \left(\frac{3}{8}(\beta_{im} + 1) \right) \right] \quad (6.38)$$

Message value	0	1	2	3	4	5	6	7
exact mapping	128	56.614	26.391	12.430	5.868	2.771	1.309	0.618
approximation	128	64	24	12	4	2	1	0
error	0	7.4	-2.4	-0.4	-1.9	-0.8	-0.3	-0.6

Table 6.1: Check node input message mapping.

The actual values were scaled and approximated by seven bit integers such that the representation of any three bit reliability input required at most only two non zero bits in the seven bit result.

Exact mapped reliability values using equation (6.38) and the approximations are shown in Table 6.1 and Figure 6.5. The small error in the mapping has no effect on the decoders performance at the signal to noise ratios of interest, which will be shown when the final fixed point decoding algorithm is compared with a floating point precision implementation of the sum-product algorithm.

At the output of the check node the reliability values must be converted back to log-likelihoods using the function

$$\beta'_{ij} = 2 \operatorname{atanh} \{ \exp(x_{ij}) \} \quad (6.39)$$

This function was approximated using a leading zeros count for the group sum, limited to the integers [0,7]. The exact mapping, equation (6.39), and the leading zeros count are shown in Figure 6.6. The error in the calculation of the message log-likelihood values again does not significantly effect the performance of the decoder at the signal to noise ratios of interest, again this will be shown when the final fixed point decoding algorithm is compared with a floating point precision implementation of the sum-product algorithm.

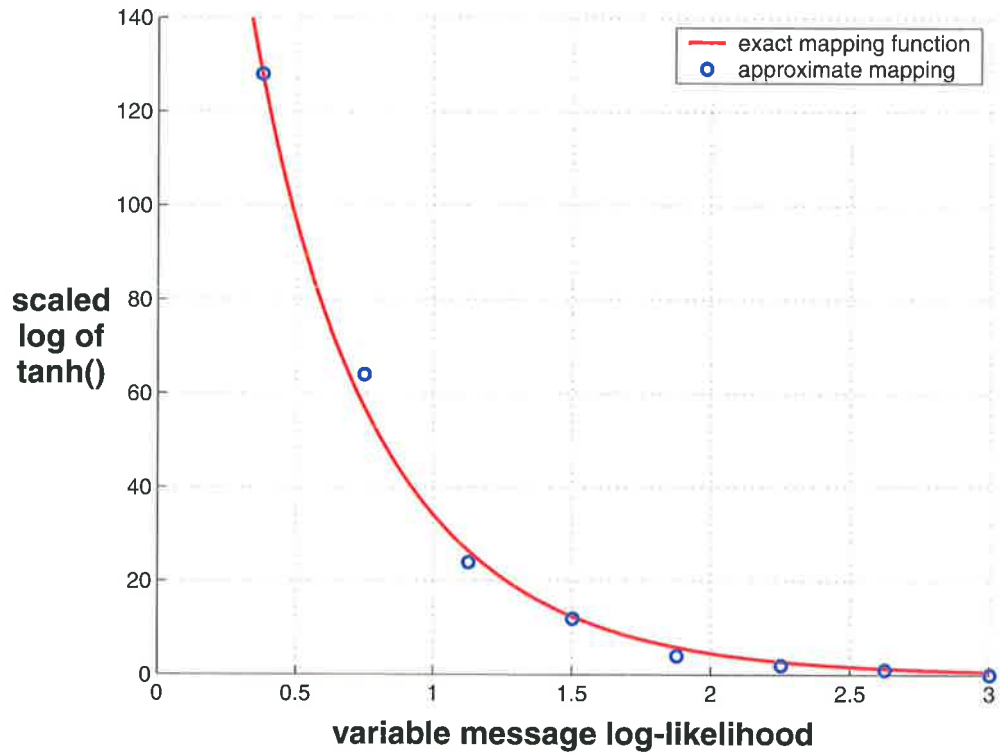


Figure 6.5 : Approximation of the check node input mapping function.

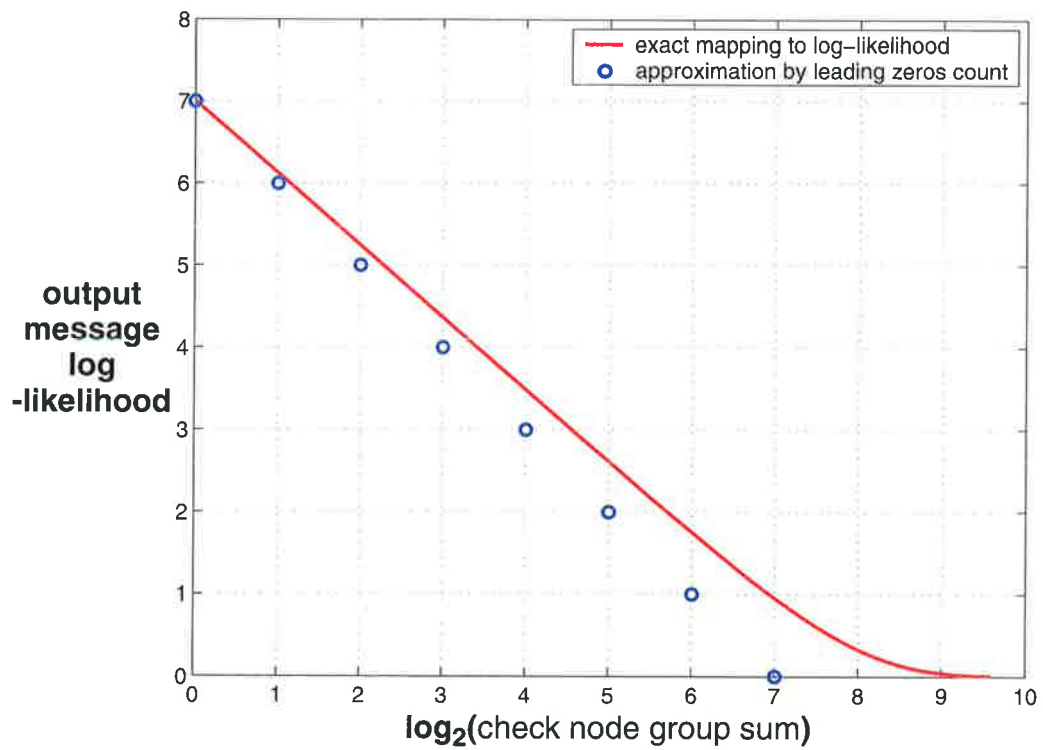


Figure 6.6 : Approximation of the check node output mapping function.

The decoder variable node update was performed as a weighted summation, with the initial channel sample given the weight

$$\text{round}\left(\frac{8.5}{3} \times y_i\right) \quad (6.40)$$

where the $\text{round}()$ function is rounding to the nearest integer. The weighted initial value is added to the sum of all incoming log-likelihood check messages to form a group sum. The sign of the group sum is used as the estimated decoded bit at each iteration and in the final iteration is declared to be the decoded bit for the variable node, except in the case that the sum is zero, in which case the sign of the received bit is then used as the decoded bit. Messages for propagation to the check nodes in the next decoding iteration are formed as the group sum minus the message arriving on each edge with the magnitude of the reliability scaled using the values in Table 6.2 and Figure 6.7. When the group sum minus the weighted message is zero the sign of the received bit is used as the sign of the next message.

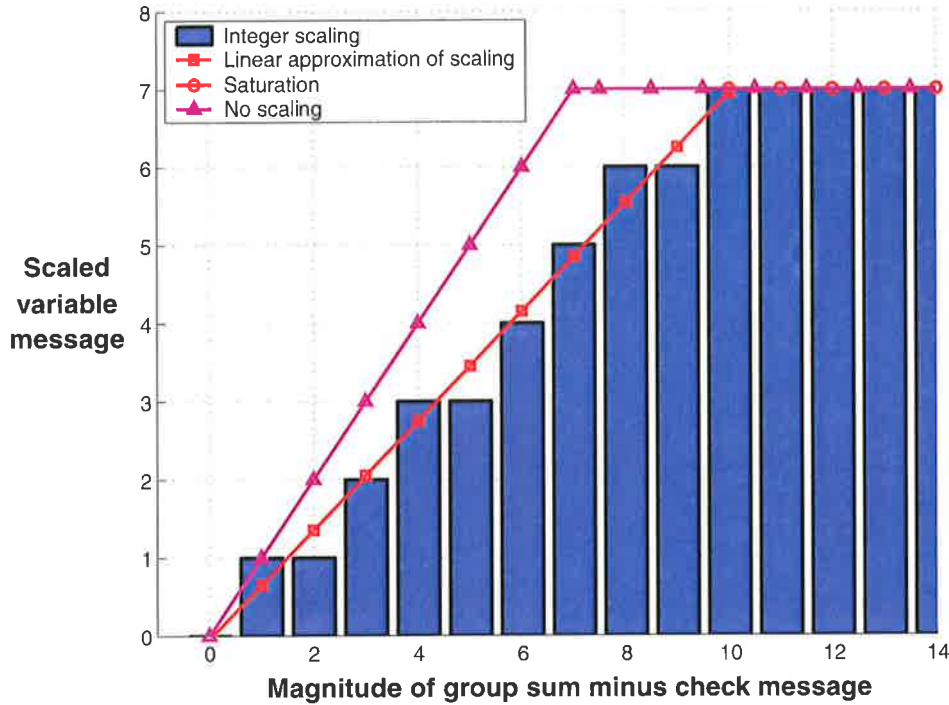


Figure 6.7 : Variable node scaling of out-going check message.

abs (group sum subtract message)	0	1	2	3	4	5	6	7	8	9	≥ 10
variable message	0	1	1	2	3	3	4	5	6	6	7

Table 6.2: Variable node output message scaling.

The effect of the scaling on the decoders performance is shown in Figure 6.8, where the final fixed point decoding algorithm derived here is used with and without the scaling. The scaling function is not applied to the initial channel samples which are used as the variable messages in the first decoding iteration. Scaling the reliability of the variable messages is easily implemented as combinatorial logic due to the small number of bits involved. This results in significant performance improvement.

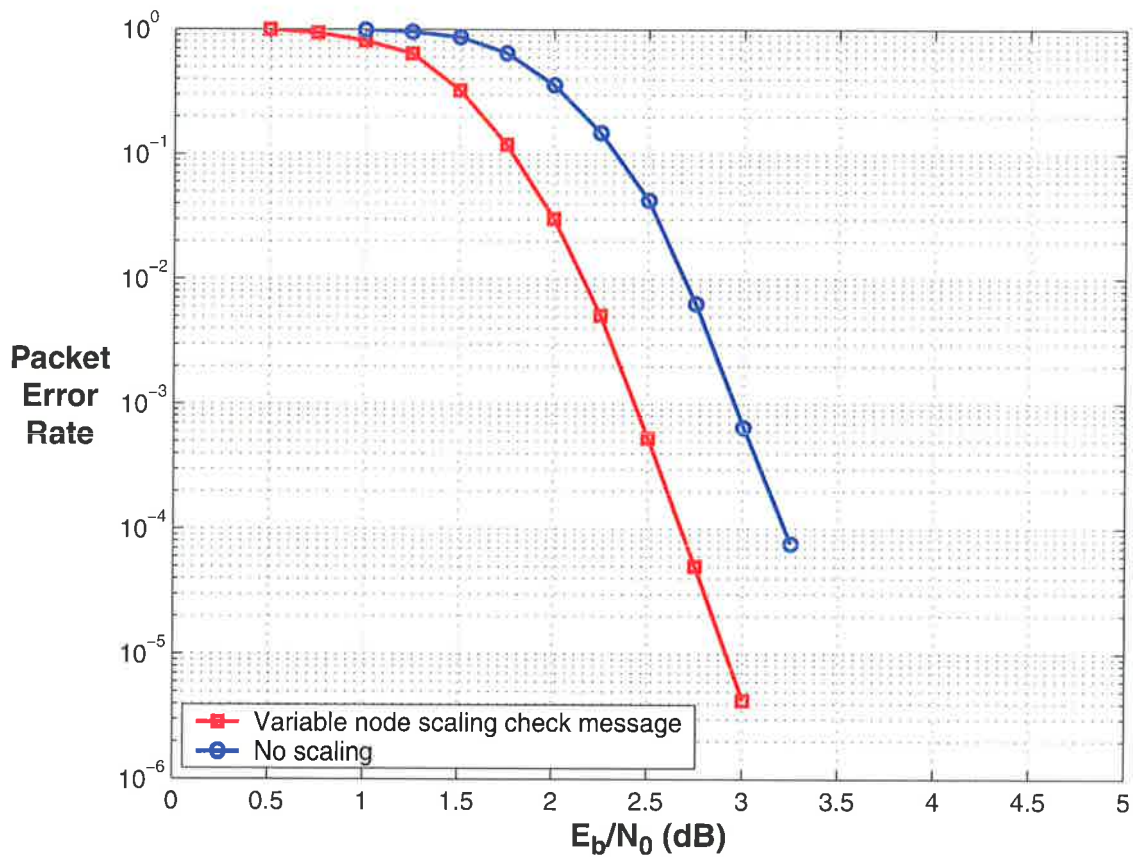


Figure 6.8 : Packet error rates when performing 64 iterations of a fixed point decoder with and without variable message scaling, both hard limited to a maximum variable message reliability of seven.

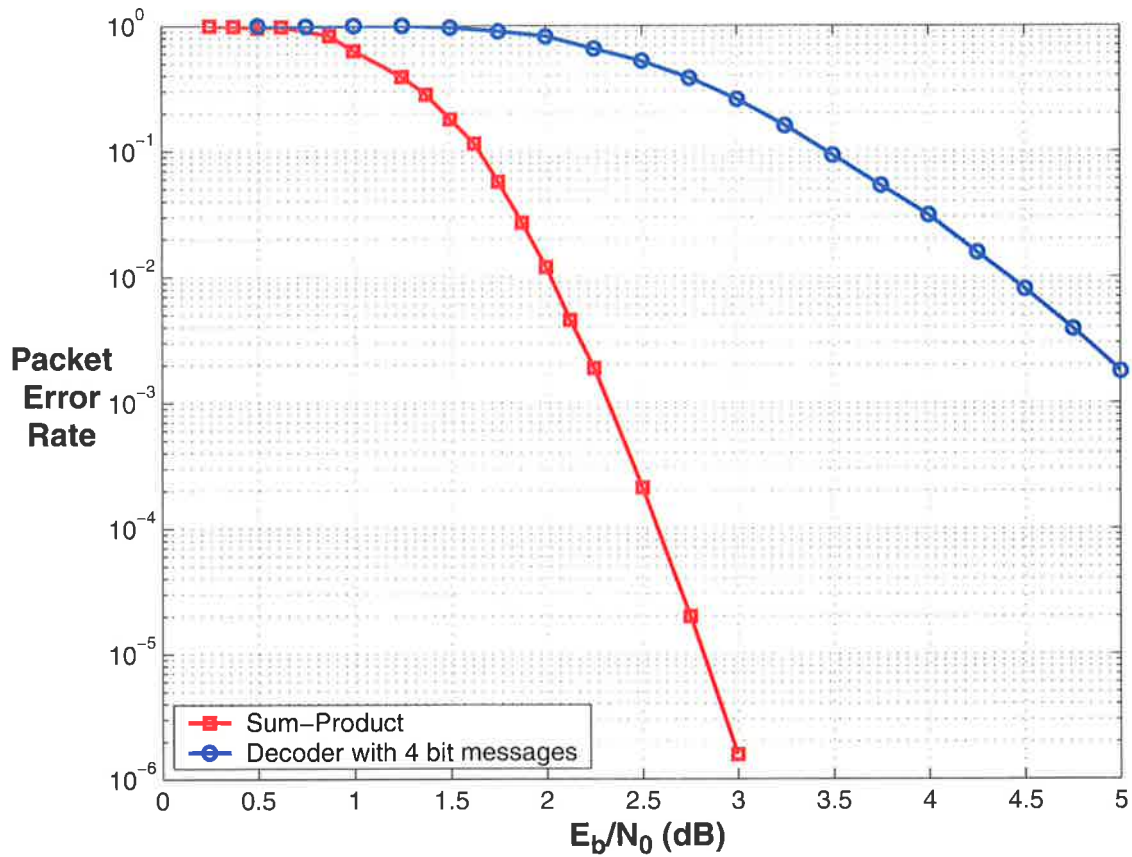


Figure 6.9 : Packet error rates for a 1024-bit rate 1/2 code decoded using 64 iterations of a sum-product soft decision decoder and a fixed point soft decision decoder with 4-bit messages.

6.4.3 Graph Cycles and Finite Precision

The performance of the decoder using the current fixed point approximations is shown in Figure 6.9 compared to a double precision floating point implementation of the sum-product algorithm. It is evident that the current approximations, saturation, scaling factors and fixed point precision result in very poor performance.

Saturation of the message values prevents the decoder from breaking many error cycles that the floating point sum-product decoder can correct. This is a similar situation to the local minima which caused many low weight error events to result in block errors for the 32,640-bit hard decision decoder, described in Section 5.5.3. Using the same reasoning

a perturbation of the state of the decoder was introduced by changing the log-likelihood weighting of the received bit in the variable node update to be the value:

$$\text{round}(1.5 \times y_i) \quad (6.41)$$

replacing the scaling of equation (6.40) after a number of decoder iterations. Through simulation of the decoder it was determined that the weighting change from that of equation (6.40) to equation (6.41) should be made at decoding iteration number ten.

Introducing the weighting change after ten decoder iterations results in the performance shown in Figure . Changing the check and variable node update for a fixed point decoder to remove the error floor was also concurrently proposed by Richardson and Urbanke [52], who changed the non linear mapping of the reliability update for messages at the check nodes after a number of iterations of a 3-bit decoder for (3,6) regular codes.

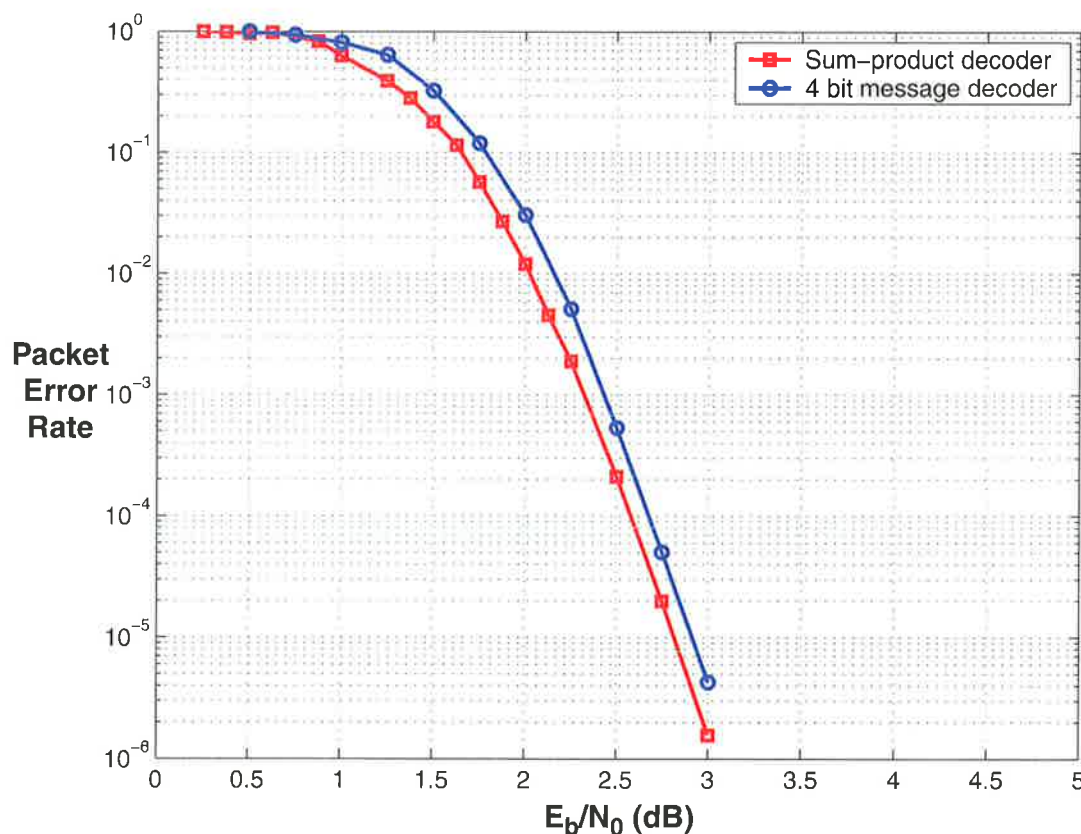


Figure 6.10 : Bit and packet error rates for a 1024-bit rate 1/2 code decoded using 64 iterations of a sum-product decoder and fixed point soft decision decoder with 4-bit messages.

Comparing the fixed point decoder using 4-bit messages and the sum-product decoder implemented using double precision floating point arithmetic at a packet error rate of 10^{-2} it can be seen that the fixed point implementation loses less than 0.14 dB of coding gain. The total loss compared to the sum-product decoder implemented using double precision floating point numbers and performing 1000 iterations is approximately 0.2dB. The extremely small deterioration shows the sum-product algorithm is amenable to implementation using very few quantisation levels. The efficient mapping of the algorithm to the fixed point representation is also evident when compared to the 6-bit quantisation both Zhang, Wang and Parhi [71] and Ping and Leung [48] proposed for use with the sum-product algorithm. The fixed point algorithm is an extremely efficient mapping of an iterative decoding algorithm, both the goal of using few quantisation bits to reduce the complexity of the decoder and a very small coding performance loss have been met.

6.4.4 3rd Generation Wireless 1024-Bit Rate 1/2 Turbo Code

The Third Generation Partnership Project (3GPP) specifies a 1024-bit rate 1/2 turbo code as one channel coding option for data transmission [73]. The coding performance of the 1024-bit rate 1/2 LDPC code decoded using the fixed point decoder will be compared to that of the 1024-bit rate 1/2 turbo code specified in the 3GPP proposal.

The performance of the iterative turbo decoder using both MAP and Soft Output Viterbi Algorithm (SOVA) constituent decoders is shown in Figure 6.11. Both 6 and 20 decoder iterations of a floating point precision implementation of the decoders are shown. It will be shown in Section 7.6 that it is possible to implement a LDPC decoder with 64 iterations with both high throughput and low power dissipation. The throughput achieved with the decoder described in Section 7.6 is far in excess of that of any published turbo decoder [26]. After 20 iterations of the turbo decoder there is very little improvement in the coding gain and results with more iterations are not shown. With 20 iterations of a MAP decoder the turbo code performance is superior to the LDPC decoder by approximately 1 dB. When practical limitations result in a MAP based turbo decoder only able to perform 6 decoder iterations the coding gain of the LDPC and turbo decoder become very similar. The 1024-bit rate 1/2 fixed point LDPC decoder with 4-bit messages can be considered as an alternative choice to a turbo code when considering the coding gain of the two codes.

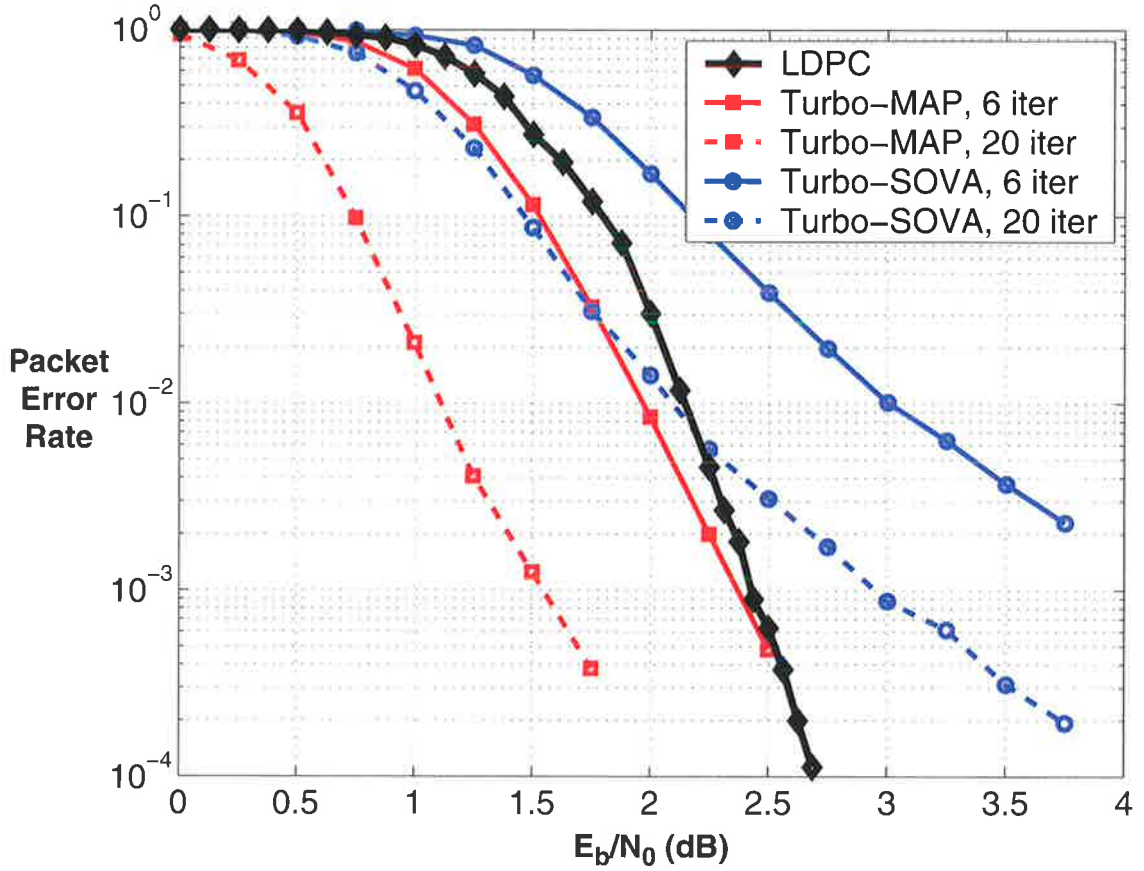


Figure 6.11 : Packet error rate for 1024 bit rate 1/2 LDPC decoded with 64 decoder iterations and Turbo Codes [25].

6.4.5 Arithmetic Operations Per Bit Per Decoder Iteration

To perform one iteration of the 1024-bit fixed point decoder requires 12,800 addition or subtraction operations which is equivalent to 12.5 addition or subtraction operations per bit per decoder iteration.

MacKay calculates the complexity of implementing the MacKay and Neal's derivation of the sum-product algorithm for a (n, j, k) LDPC code in [39]. The complexity to calculate each component of the algorithm is

- δq_{mn} - nj additions
- δr_{mn} - $3n(j-1)$ multiplications
- r_{mn}^x - nj additions

- $f^x \prod_m r_{m'n}^x$ - $2n(3j-4)$ multiplications
- α_{mn} - nj additions and nj divisions
- q_{mn}^x - $2nj$ multiplications
- α_n - n additions and n divisions
- q_n^x - $2n$ multiplications

For a rate 1/2 code the number of operations per bit per decoder iteration is $(11j-9)$ multiplications, $(j+1)$ divisions and $(3j+1)$ additions. For the 1024-bit code considered here with $j = \lambda_{av} = 3.25$ this is 26.75 multiplications, 4.25 divisions and 10.75 additions per bit per decoder iteration.

Fossorier, Mihaljevic and Imai [20] calculate the complexity of implementing a min-sum type decoder derived by MacKay in [39] which loses approximately 0.4dB of coding gain compared to a sum-product decoder at a bit error rate of 10^{-3} . The complexity to calculate each component of the algorithm is

- $\min_{n'} \{ |y_{mn'}| \}$ - $(n/2)(2j + \lceil \lg(2j) \rceil - 2)$ additions
- z_n - nj additions

For the 1024-bit code considered here with $j = \lambda_{av} = 3.25$ this is 17.25 additions per bit per decoder iteration.

The proposed implementation of the fixed point decoder is therefore very efficient compared to other published algorithms. The complexity calculation has ignored the merged logarithm and hyperbolic tangent used at the input of the check nodes and the leading zeros count since these operations are performed with very few bits and do not require many gates to implement. It will be shown in Section 7.6 that both the input function and leading zeros count can be implemented using 11 standard cells from Agere Systems low power COM2 1.5V 0.16 μ m standard cell library.

6.5 Summary

Soft decision decoding algorithms for low-density parity-check codes have been reviewed. The fixed point implementation of the algorithms has been investigated and a 64 iteration decoding algorithm for a 1024-bit rate 1/2 irregular LDPC code has been developed which results in less than 0.2 dB coding gain loss when compared to a double precision floating point decoder performing 1000 decoding iterations. The extremely small loss of coding gain shows the sum-product algorithm is very well suited to implementation using very few quantisation levels. The efficient mapping of the algorithm to the fixed point representation is also evident through this small loss.

The fixed point algorithm has been compared to the 1024-bit rate 1/2 turbo code in the 3G standard proposal. The code performance is similar to 6 iterations of a turbo decoder with constituent decoders implementing the MAP algorithm.

The implementation complexity in terms of arithmetic operations required per bit per decoder iteration has been calculated to be 12.5 fixed point addition or subtraction operations. This compares very well with the 17.25 floating point addition or subtractions required to implement a decoder with approximately 0.4 dB loss compared to a sum-product decoder presented in [20].

Chapter 7

Decoder Architectures

The implementation of LDPC decoders has not been widely investigated and reported. The majority of published research has concentrated on the performance of code design and decoding algorithms for low-density parity-check codes. With respect to implementation issues two papers have been published regarding the fixed point simulation performance of soft decision decoders [48, 71] and two regarding the arithmetic complexity of decoders [39, 20]. In terms of architectures for LDPC decoders two papers have also been published concurrently with this work [60, 69] and will be reviewed in this chapter. Both of the architectures can be considered as instances of a generalised memory based architecture where memory is used to store and exchange message in the decoder. It will be shown that memory based architectures are not suitable for implementing high throughput decoders.

In this chapter a parallel decoder architecture is proposed for supporting very high throughputs. The architecture is demonstrated through the implementation of two decoders. The implementation of a 1Gbs^{-1} 1024-bit rate 1/2 soft decision decoder is described and measurement results from the fabricated ASIC are presented. The implementation of a 43Gbs^{-1} 32,640-bit rate 239/255 hard decision decoder which is part of a commercial product is also described. Both decoders were implemented using Agere Systems' 1.5V 0.16 μm CMOS process, the former using 5 levels of metalisation and the later 7 levels of metalisation.

7.1 A Decoder Architecture for Wireless Communications

Sorokine, Kschischang and Pasupathy described an architecture for implementing MacKay and Neal's sum-product algorithm for use in mobile telephony [59]. One modification to the algorithm was proposed, the use of a matrix of the difference between q_{mn}^0 and q_{mn}^1 rather than using two separate matrices. The architecture uses four functional processing units and three interconnection units. Each of the functional units implements one part of the iterative update:

- two units implementing the separate vertical matrix updates for the probability of a zero and a one,
- one unit performing normalisation and an updated syndrome calculation using a matrix multiply, and
- one unit performing the horizontal matrix update.

Cross-bar switches or Banyan-networks¹ are suggested for implementing the interconnect units.

The architecture described is only a block diagram division of the decoding algorithm without details of pipelining or memory/functional unit contention resolution. No simplification of the algorithm is suggested beyond the use of a difference matrix, resulting in the need to perform many multiplication operations and divisions for normalisation.

1. A Banyan network is a power of 2 divide and conquer type switch similar to the interconnection of the 'butterfly' units in a realisation of a Fast Fourier Transform.

7.2 A Decoder Architecture for Magnetic Recording Channels

Yeo, Pakzad, Nikolic and Anantharam proposed a decoder architecture for decoding an $n = 4608$, $m = 512$ regular (4,36) LDPC code used as forward error correction with the magnetic recording channel of a disk drive [69]. The algorithm considered for implementation is a form of the sum-product algorithm with check node updates performed in the log domain, which does not require multiplications or divisions.

Pipelining and implementation of the functional units is described in detail. The architecture proposes the use of hardware sharing of four parity check update units and four variable update units. An overview of a method using first-in first-out (FIFO) memories and stacks to exchange information between the variable and check update units is described. Each of the functional units is supplied data by a FIFO or stack, without contention. A detailed description of the method used to resolve write conflicts to the FIFOs and stacks is not given, and could constrain the structure of the LDPC code implemented using the architecture.

However, the most problematic feature when implementing the architecture is the proposed use of very high fan-in demultiplexers at the input and output of the FIFOs and stacks supplying data to the functional units, for example a 512 input demultiplexer for the parity check output and a 4608 input multiplexer at the input of the variable node units. The implementation of very high fan-in and fan-out multiplexers is impractical due to the extremely high routing congestion associated with them.

7.3 General Memory Based Message Exchange Architectures

The architectures of Yeo *et. al.* [69] and Sorokine *et. al.* [60] are both examples of memory based message exchange. A general purpose decoder can be built using memory to exchange messages between the variable and check node functional units, as shown in Figure 7.1. The received values for a block of data are stored in the memory and decoding consists of alternating between performing the check and variable updates using the respective functional units. To reduce memory access contention problems the number of functional units in this architecture would generally be much smaller than the number of rows and columns in the matrix.

For a random and unstructured parity check matrix the decoder cannot commence the variable update until the check update is completed, and vice-versa. To increase the utilisation of the functional units the decoder could operate on two blocks of data at a time, with the check node processors operating on one block while the variable node processors operate on the other block.

Control logic is required to generate the correct memory addresses for the parity check matrix the decoder is operating upon. Scheduling the functional unit updates such that there is no contention between read and write memory accesses and is extremely

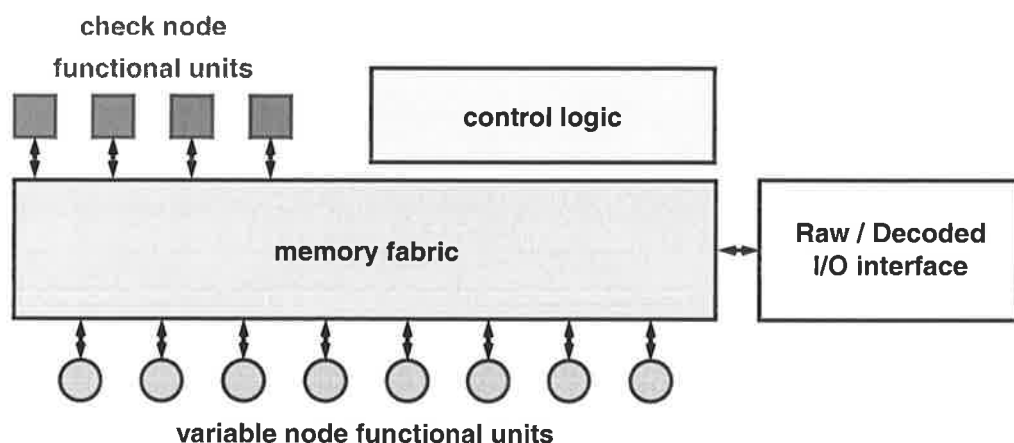


Figure 7.1: Decoder using memory based message exchange.

complex. Memory contention can be reduced using hierarchical memory structures and enforcing a structure on the parity check matrix to control the pattern of memory access, potentially at the expense of coding gain.

A further limitation of memory based message exchange is the memory bandwidth required to support high throughput decoders. The memory bandwidth required to implement memory based message exchange is given by

$$\text{av. column weight} \times \text{no. of bits/message} \times \# \text{ of iterations} \times 2 \times \text{throughput} \quad (7.1)$$

for both read and write operations, where the factor of two is to count both variable and check messages. The memory bandwidth required to implement the 1024-bit rate 1/2 soft decision decoding algorithm designed in Section 6.4 with a throughput of 1Mbs^{-1} is therefore

$$3.25 \times 4 \times 64 \times 2 \times 10^6 \approx 1.66 \text{ Gbits/s} = 208 \text{ MB/s} \quad (7.2)$$

The high bandwidth memory access will result in a decoder which dissipates a lot of power.

Implementing the relative reliability weighted decoding algorithm from Section 5.5 for the 32,640-bit rate 239/255 code with a throughput of 43Gbs^{-1} requires a memory based decoder with a memory bandwidth of

$$5 \times 1 \times 51 \times 2 \times 43 \times 10^9 \approx 21.9 \text{ Tbits/s} = 2.7 \text{ TB/s} \quad (7.3)$$

It is also noted that these bandwidth calculations are lower bounds, higher bandwidth access may be required if the location of the messages in the memory results in multiple read and write operations to the same memory rows during a single decoder iteration. Therefore memory based message exchange is infeasible for the 43Gbs^{-1} throughput requirement of the optical transceiver.

Some of the memory contention and bandwidth problems associated with memory based message exchange can be illustrated with the simple example of a soft decision

decoder exchanging t -bit messages for a 12-bit regular (3,6) LDPC code whose graph is shown in Figure 7.2 part (a). The messages exchanged by the decoder can be stored in a $3t$ -bit wide memory with 12 rows, as shown in Figure 7.2 part (b). Each row of the memory can store the three messages associated with a variable node, for example the three messages for variable node 0 in row 0, down to the three messages for row 11 stored in row 11 of the memory.

When the variable nodes are being updated the decoder can read a single row at a time to update a single variable node at a time. If more parallelism is required two or more rows of memory can be read at a time and the decoder can perform the updates in parallel. Problems with the memory architecture arise when processing of the check node updates commences. The locations of the data required to perform each check update and then write the results back to are not organised in the memory. Instead the messages are distributed randomly according to the graph structure. If the message associated with the left most edge incident on a variable node of Figure 7.2 part (a) is stored in the left most t -bits of the memory, the message associated with the centre edge incident on each node is stored in the centre t -bits of the memory, and message associated with the right most edge incident on a variable node is stored in the right most t -bits of the memory, then the messages associated with each check node are as shown in Figure 7.2 part (c).

Although it is relatively straight forward to build a memory architecture which enables high bandwidth reading and writing of the messages for the variable nodes, which only access the memory row-wise, it is very difficult to build a high bandwidth memory architecture to access the messages required by the check nodes. Using significant amounts of parallelism to increase the throughput of a decoder is also extremely difficult, as multiple check nodes may need access to the same row or column of the memory at the same time.

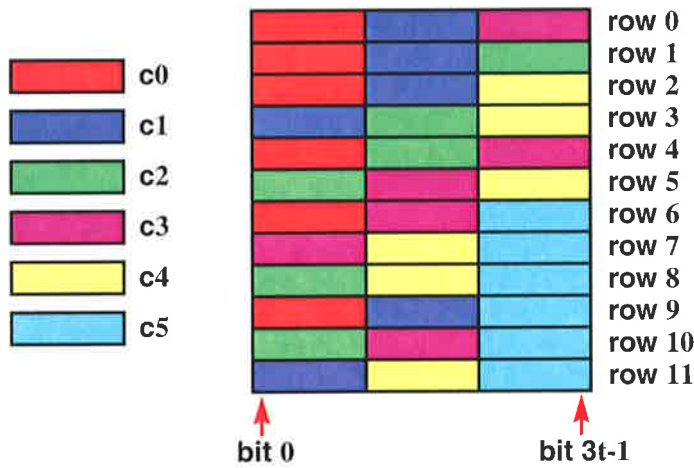
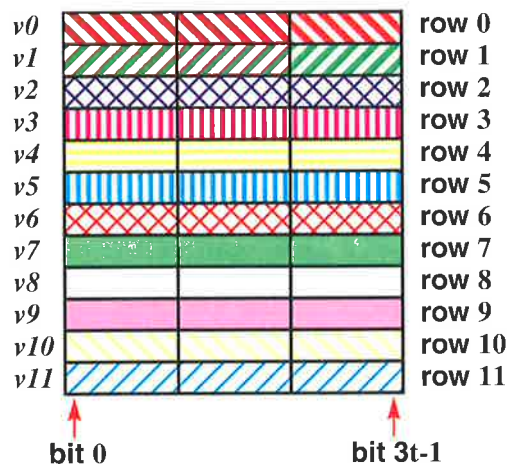
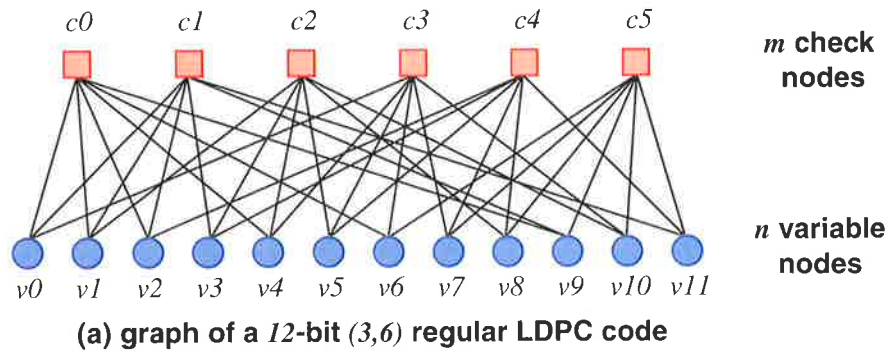


Figure 7.2: (a) Graph representation of a 12-bit (3,6) regular LDPC code, (b) $3t$ bit by 12 row memory storing the messages exchanged by a decoder for the code and (c) memory locations of the messages for the check nodes.

7.4 Parallel Decoders

To achieve high throughput very high bandwidth access to the messages being exchanged by the decoder is required. An efficient implementation of the high bandwidth memory would be to store the messages in flip-flops, latches or register banks. A high throughput decoder can be designed using a large number of parallel functional units with data routed to the appropriate destination and latched, with a small amount of select or multiplexing logic to control the destination and source flip-flops, latches or registers for each the operations of each cycle. A parallel decoder is therefore a direct instantiation of the bipartite graph representation of the code, as shown in Figure 7.2 part (a), with dedicated wires representing each edge of the graph. A processing functional unit representing each node in the graph can be instantiated, or a small degree of hardware reuse performed by sharing each of the nodes a small number of times and multiplexing the correct messages in and out of the node at each time step. Although the functional units representing the nodes are only used once during a single decoding iteration, they are reused at the next iteration to update the same node in the block they represent.

However, the processing units for the variable and check nodes are extremely simple, especially for hard decision decoders. The amount of control logic required to multiplex and select data for the input and output registers, flip-flops or latches can be greater than the processing logic being shared. As a result it is far simpler to implement the entire

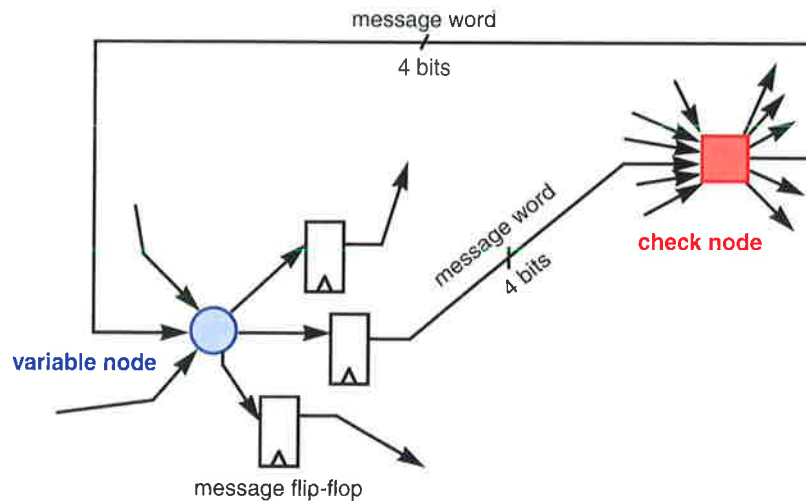


Figure 7.3: Pipeline flip-flops for 4-bit messages in a soft decision parallel decoder, only showing one variable and check node.

decoder in parallel, with a functional unit instantiated for every variable and check node of the graph. A decoder which performs a complete iteration per clock cycle can be constructed as shown in Figure 7.3, with pipeline latches for the messages at the output of the variable node. With the variable and check node update for the entire decoder occurring in parallel the clock frequency required for the decoder can be calculated to be:

$$f_{clk} = \text{throughput} \times \# \text{ of iterations} / \text{block length} \quad (7.4)$$

For the wireless application to achieve a 1Mbs^{-1} throughput using the 1024-bit rate-1/2 soft decision code from Section 6.4 the required clock frequency for a parallel decoder performing 64 decoder iterations is:

$$f_{clk} = (1024)^2 \times 64 / 1024 \approx 65.5 \text{ kHz} \quad (7.5)$$

For the optical application the exact throughput requirement of the OC-768 standard [74], often referred to as 40G, is:

$$f_{bit} = \frac{255}{236} \times 768 \times 51.84 \text{ MHz} \approx 43.02 \text{ GHz} \quad (7.6)$$

where the data rate of the code has been reduced from 239/255 to 236/255 through the use of three data bytes as network control information. Implementing a parallel decoder for the 32,640-bit rate 239/255 code from Section 5.5 which performs 51 decoder iterations with a throughput of 43Gbs^{-1} requires a clock frequency of:

$$f_{clk} = 43.02 \times 51 / (32640) \approx 67.27 \text{ MHz} \quad (7.7)$$

The low clock frequency required to implement a high throughput parallel decoder enables the implementation of the adders in the decoder with simple ripple carry adders, reducing the complexity and area of the processing nodes.

A new block of data samples is required every time the decoding of the previous block has been completed. If the number of iterations that the decoder performs for each

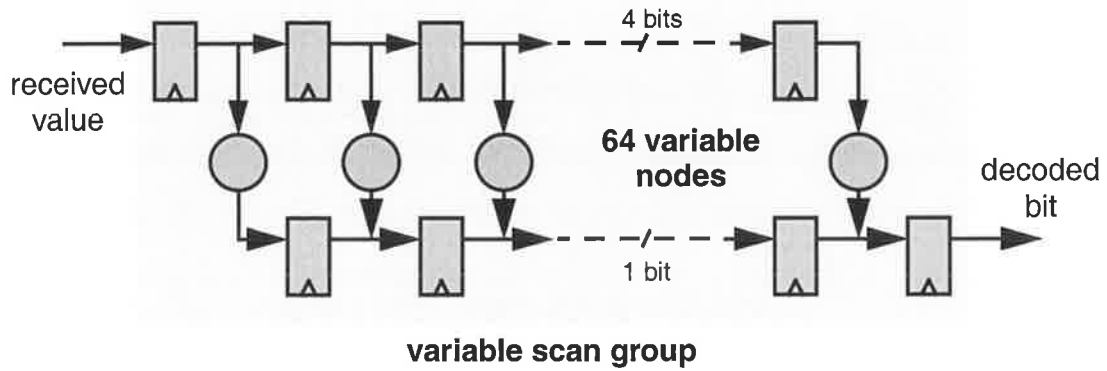


Figure 7.4: Scan chains used to load received channel samples and output decoded bits in a parallel decoder which performs 64 decoder iterations.

block of data is fixed, data can be loaded in and out of the decoder using shift registers or scan chains. The complexity of the decoder is greatly simplified if the depth of the scan chains can be made equal to the number of decoder iterations. Then an input bus of width, w , equal to the block length divided by the number of decoder iterations, d , can be used to load data into the decoder,

$$w = n / d \quad (7.8)$$

By using w shift registers in parallel, each with a depth of d elements, the shift registers will contain an entire data block every time the decoder finishes decoding the previous data block. One scan chain element can then be associated with each variable node in the decoder. While a block of data is decoded the next block is scanned into the decoder and the previous block can be scanned out. Using a global control signal to indicate the start of a new data block the variable nodes can copy the new block into local latches and also load the output scan chain with the decoded bits of the previous block.

Using this approach the input and output for the 1024-bit decoder was divided into 16 variable scan groups, each with 64 pipeline stages, with one variable scan group shown in Figure 7.4. Not shown in Figure 7.4 is the multiplexer at the input to each latch in the output decoded data scan-chain. The multiplexers are used to load the next block of decoded bits into the output scan chain and are selected using a block start signal. The 32,640 bit decoder was divided into 640 scan chains, each consisting of 51 pipeline stages, as will be shown later in Section 7.7 and Figure 7.19.

7.5 Message Switching Activity

A parallel architecture results in the data processed by a variable or check node only changing when the messages incident to the node change or when a new block of data is loaded into the decoder. Once a block of data has converged to a valid codeword the messages passed by the nodes of the decoder also converge and cease switching. The result of this convergence is a very low switching activity for the messages exchanged by the decoder. This is illustrated in Figure 7.5 for the 4-bit message passing soft decision decoding algorithm designed in Section 6.4 for the 1024-bit rate 1/2 irregular code. It can be seen that even when the decoder never converges to a valid codeword and has a packet error rate of 100% the messages exchanged in the decoder switch less than 9% of the time. The low switching activity reflects a property of the decoding algorithm, namely that only a small number of errors, and hence messages, are corrected in each iteration. The switching activity is a function of the signal-to-noise ratio of the data that the decoder is operating on, and reduces to 1% when the packet error rate is at the operating target error rate of 10^{-2} .

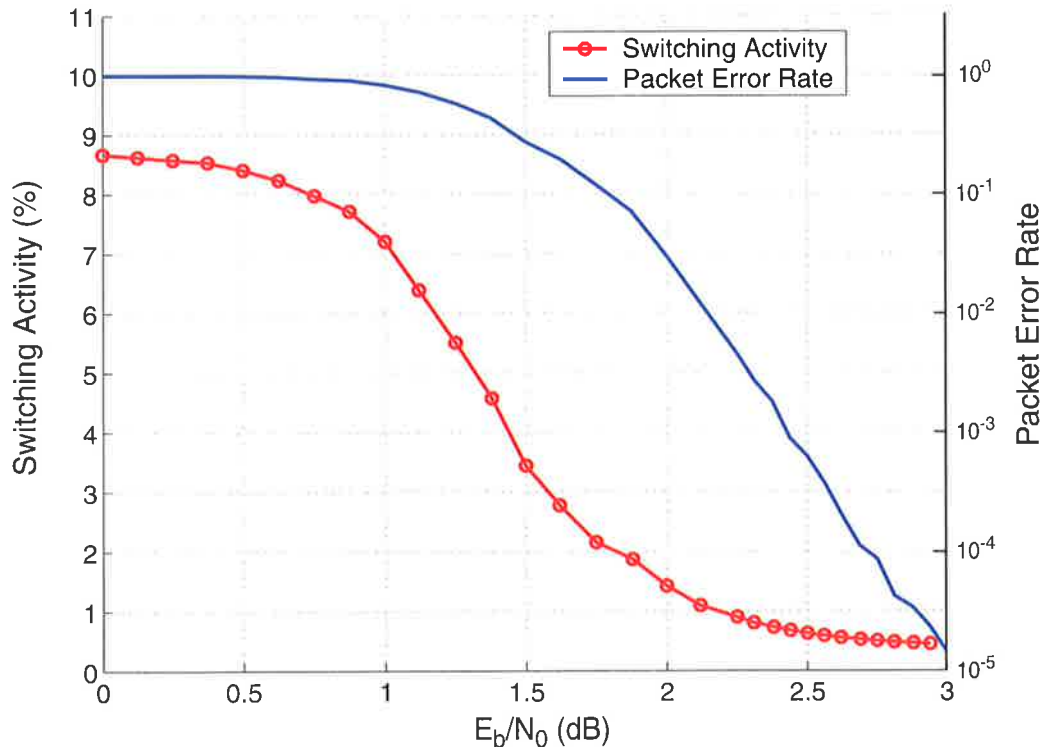


Figure 7.5: Message switching activity for a parallel 1024-bit rate 1/2 soft decision decoder using 4-bit messages and performing 64 decoder iterations per block [25].

7.6 A 1024-Bit Rate 1/2 1Gb/s Soft Decision Decoder

A soft decision decoder for the 1024-bit rate 1/2 irregular code designed in Section 3.7 using the fixed point soft decision algorithm derived in Section 6.4 was designed using the parallel architecture proposed in Section 7.4. The chip was implemented as a collaborative effort between the author and Andrew Blanksby who is a member of technical staff, Agere Systems. All place-and-route and back end implementation of the chip was performed by Andrew Blanksby. All algorithms and custom computer aided design (CAD) tools required to implement the decoder were designed and written by Andrew Blanksby. The VHDL description of the clock drivers, control logic and testing interface were written by Andrew Blanksby. The author wrote the VHDL description of the decoder, verified the VHDL against the C-model of the decoder, performed synthesis of the VHDL to a netlist of standard cells, designed the test vectors for the decoder and tested the fabricated chip.

A VHDL description of the decoder, describing the architecture of the variable nodes shown in Figure 7.6 and check nodes shown in Figure 7.7, was developed. Not shown in Figure 7.6 is the zero testing of the group sum and all differences. If a result is zero the sign of the received value is used as the sign of the next decoded bit or message, replacing the sign of the addition or subtraction result.

Messages exchanged in the decoder are represented using a sign-magnitude (SM) representation. Using a sign-magnitude representation allows the parity and magnitude update of the check nodes, given by equation (6.35), to be performed independently, as shown in Figure 7.7. Implementing the subtraction of numbers represented in sign-magnitude form is difficult. At the variable nodes the messages are therefore converted to two's complement (2's) before the addition and subtractions are performed. When the differences of the group sum and the message values are formed the results are scaled and converted back from two's complement to sign-magnitude representation. The logic synthesised from a VHDL description to convert a message from sign-magnitude representation to two's complement is shown in Figure 7.8. It can be seen that the conversion is simple and can be implemented using nine standard cells.

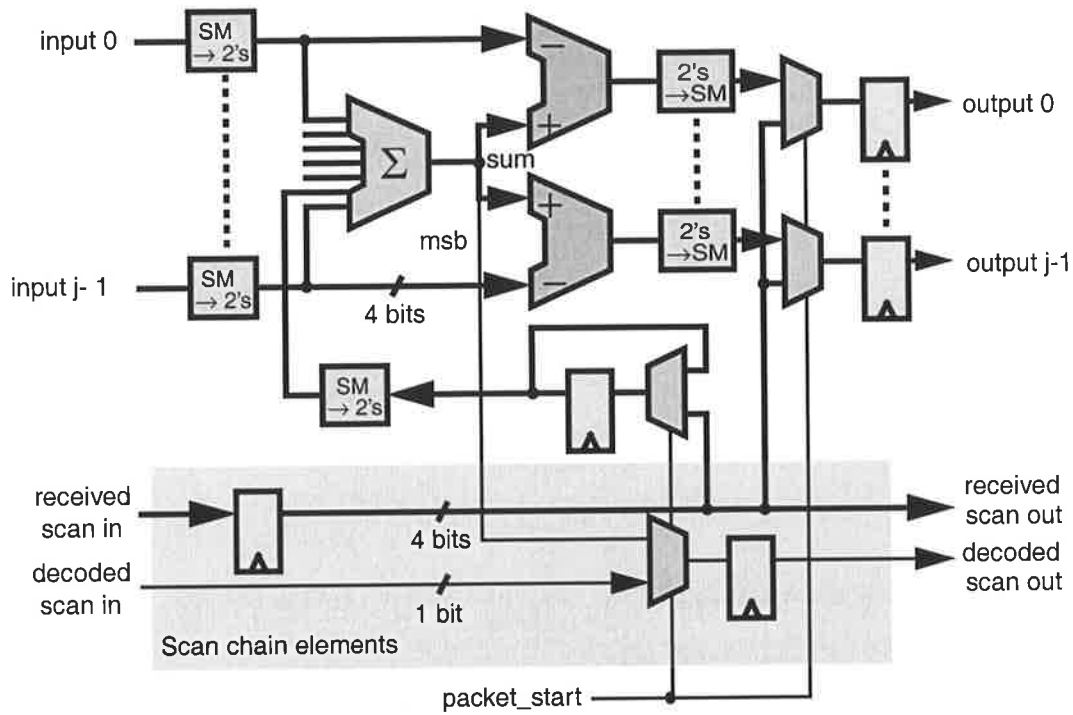


Figure 7.6: Variable node architecture for a 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages [6].

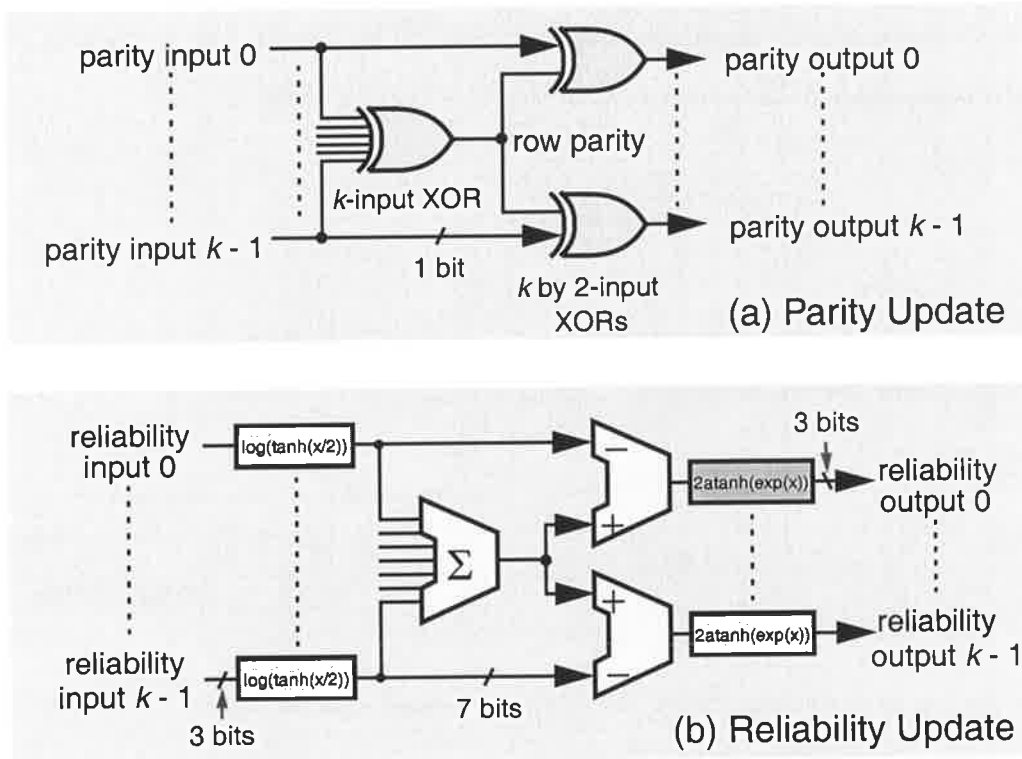


Figure 7.7: Check node architecture for a 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages [6].

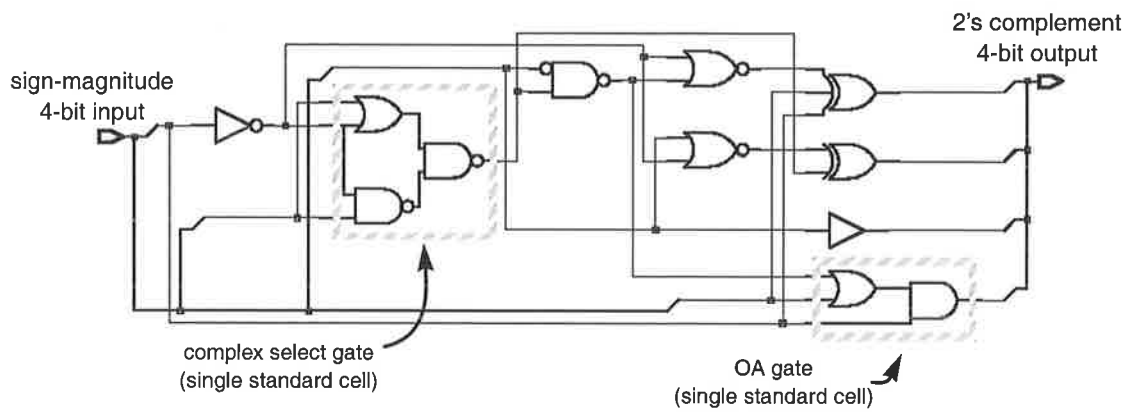


Figure 7.8: Check message conversion from sign-magnitude representation to 2's complement.

The check node shown in Figure 7.7 does not require control or clock signals, since it is solely composed of combinatorial logic. The variable nodes are clocked and require two control signals. One is a packet or block start signal required to load a new block from the received value scan chain, change the messages propagated in the next decoding iteration and load the decoded bits into the decoded output scan chain. The second control signal determines the weight of the received value in the group summation, selecting between one of the two values. A reset signal was not distributed to the flip-flops in the design since no state information is kept between two data blocks by the variable

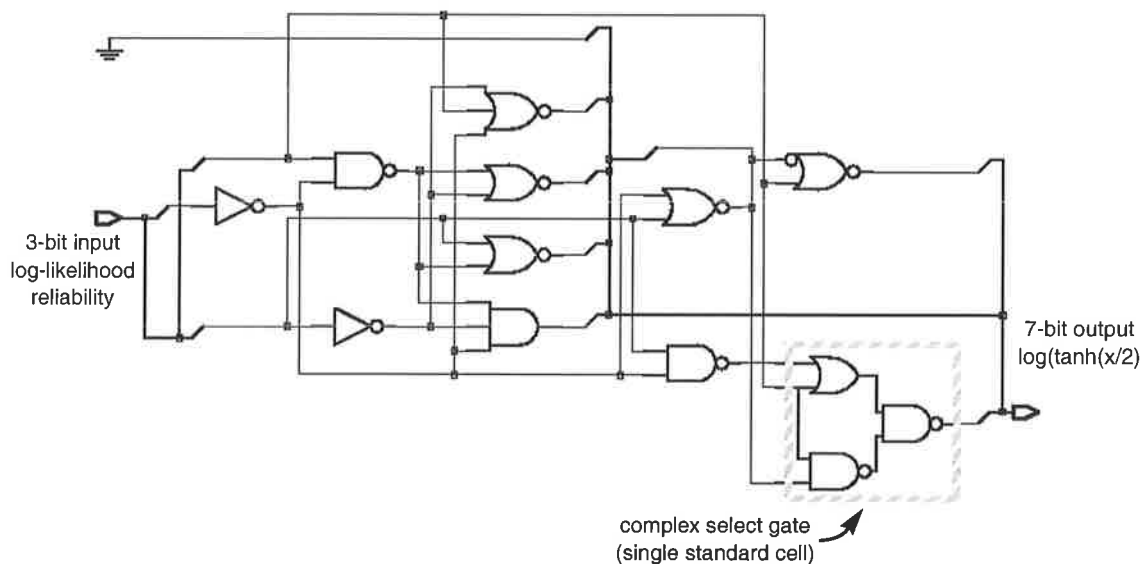


Figure 7.9: Standard cell implementation of the check node scaled input function, $\log(\tanh(x/2))$.

nodes, and the packet start signal therefore effectively resets the state by loading the next block of data into the decoder.

At the variable nodes the received values are stored locally in flip-flops for the 64 cycles of decoding the data block. The received values are scaled and converted from a sign-magnitude representation to a two's complement representation before being added to the two's complement representation of the check node messages. The scaling is controlled by a select signal, selecting the appropriate received weighting of the received value for the current iteration.

The logic required to implement the scaled merged logarithm and complex hyperbolic tangent from equation (6.38) at the input to the check nodes is shown in Figure 7.9. It can be seen that the function is implemented using only 11 standard cells. The logic required to implement the leading zeros approximation of the check node output is shown in Figure 7.10. The output function is also implemented using only 11 standard cells.

Apart from the variable and check nodes only two other blocks were implemented. One performed a 512-input OR operation on the parity result of all rows of the code. The result of the OR operation was sampled at the end of decoding each block and used as a packet error detection signal, indicating whether the decoder did or did not converge to a valid codeword.

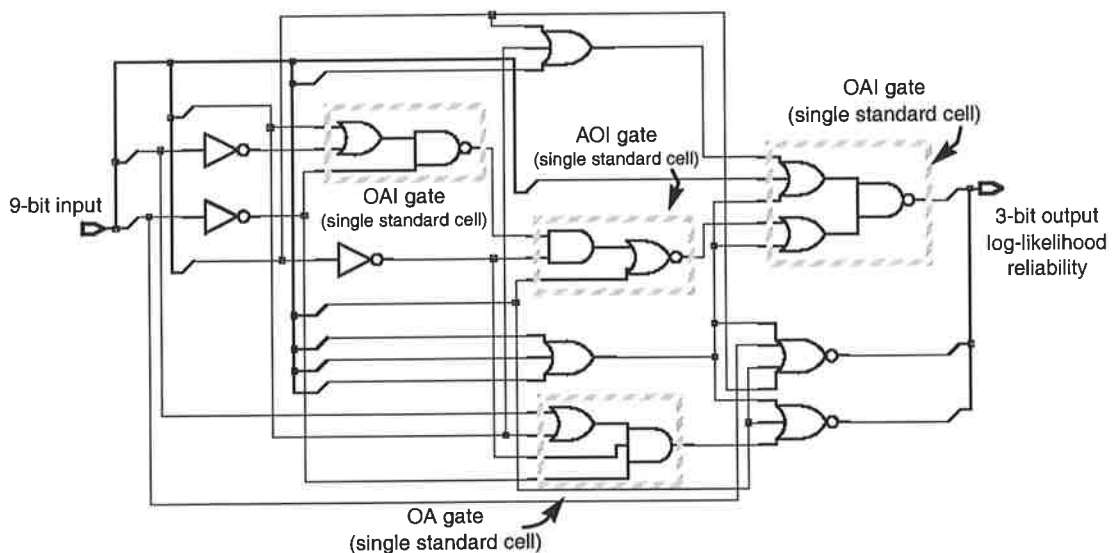


Figure 7.10: Standard cell implementation of the check node output function, $2 \tanh(\exp(x))$, approximated as a leading zeros count.

The final logic block is a small control circuit consisting of a state counter to issue the packet start and received weight signals. The controller had a number of inputs including

- a reset signal,
- a run start signal
- a data bus,
- an address bus,
- read and write enable signals.

The data bus was used to program the decoder iteration at which the weighting of the received value was to be changed, and shared the first few bits of the received value data bus. The controller could be programmed to change the received weight at a particular iteration, 1 to 63, to never activate it or always activate it. The address bus was used during the testing of the decoder. If the clock was stopped and the read enable signal was activated the address bus was used to select a group of sixteen row parities to send to the data outputs of the chip, from the 32 possible parity groups. Included with the small control circuits were buffers to drive the clock and control signal drivers which were distributed around the chip.

The operating frequency of 65kHz required to achieve a throughput of 1Mbs^{-1} with the parallel decoder architecture is extremely low given the process technology and the very small logic depth between pipeline stages of the decoder. It was therefore decided that the high throughput capabilities of the architecture would also be investigated by designing the decoder to achieve a throughput of 1Gbs^{-1} . The clock frequency required to achieve the desired throughput can be calculated using equation (7.4),

$$f_{clk} = 1\text{Gb/s} \times 64 \text{ iterations} / 1024 \approx 67.1 \text{ MHz} \quad (7.9)$$

This clock frequency is feasible given the process technology in which the decoder was to be implemented.

The messages exchanged by the nodes in the decoder represent 26,624 nets in the decoder which are randomly connected across a large circuit. A *typical* data path, e.g. FIR filter or Viterbi decoder, has many localised nets and very few long nets. The decoder was implemented using standard cells, synthesis and a place-and-route tool. The atypical wire length distribution results in very poor performance of a place-and-route tool when the design is placed-and-routed without hierarchy. To reduce the problems of the place-and-route tools the circuit was implemented hierarchically, with macro cells placed and routed for the check nodes, variable scan groups and the control logic. All of the macros were generated using only metals one, two and three, leaving the upper 2 metalisation layers for distributing power, clock, control signals and the messages exchanged between the variable and check nodes.

The floor plan of the decoder is shown in Figure 7.11, where the sixteen variable scan groups of 64 variable nodes surround the 512 check nodes. Since the check nodes do not require clock or control signals, the clock, packet start and received weight signals are distributed as a ring around the outside of the chip, avoiding the use of routing resources in the centre of the chip.

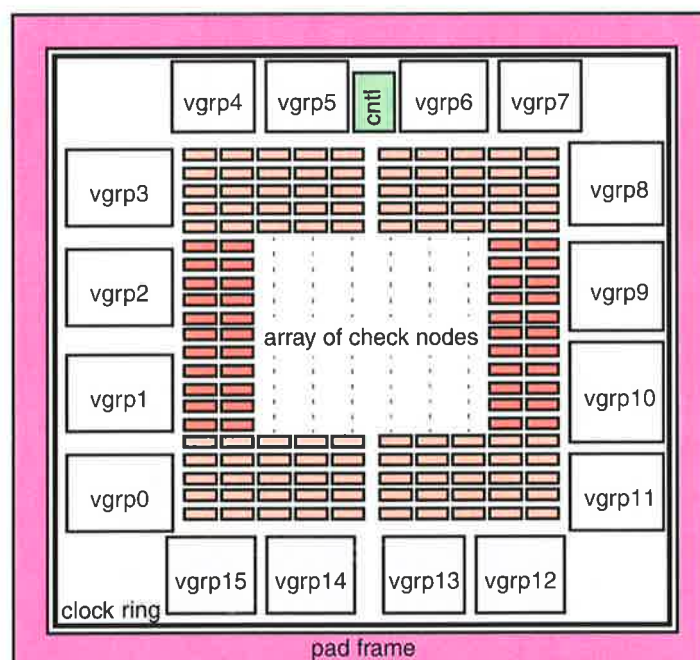


Figure 7.11: Floorplan for a parallel 1024-bit rate 1/2 soft decision decoder with 512 check nodes in the center of sixteen scan groups each containing 64 variable nodes and a small control block.

7.6.1 Parallel Decoder Routing Congestion

The selection of the placement of the 512 check nodes when implementing the floor plan shown in Figure 7.11 influences the total length of the wires required to connect the variable and check nodes. Blanksby designed an algorithm to optimise the placement of the nodes that minimises the total length of wire required to connect the nodes [6]. After placing all of the check node and variable scan group macro cells routing of the messages is non trivial. A histogram of the number of wires as a function of length is shown in Figure 7.12. The histogram shows a very large number of the top level networks are extremely long wires, with the average wire almost 3mm long. All of the longer wires require repeater buffers to meet the timing requirements of the circuit and maintain high signal slew rates.

Using a standard place-and-route tool to complete the top level buffer placement and routing results in enormous congestion in the centre of the chip. This is because all wires are routed by attempting to take the shortest path between its source and destination. The congestion results in a chip which is unroutable and has many timing violations. By attempting to make diagonal wires the place-and-route tool introduces hundreds of small jogs and metal level changes on the longer wires. Each metal level change introduces another via, with a high resistance.

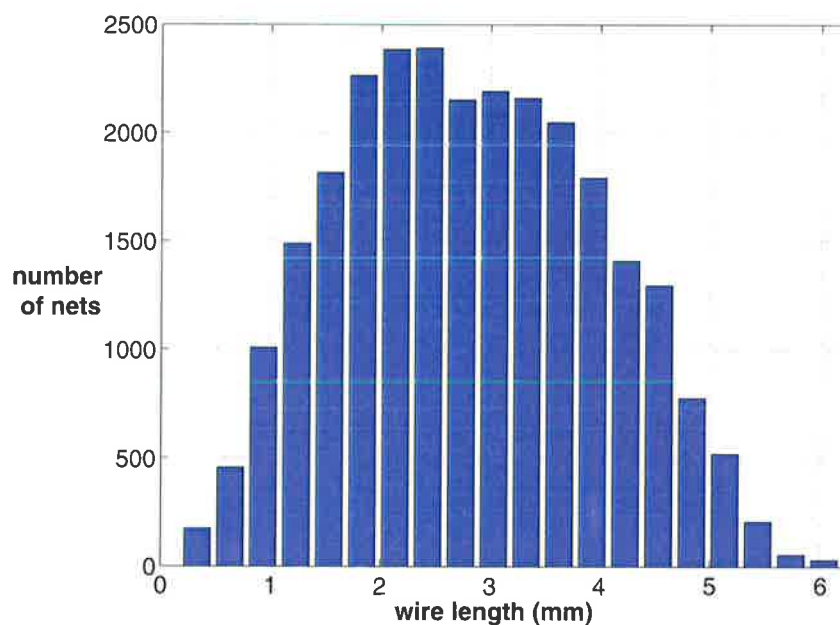


Figure 7.12: Histogram of message wire length for a 1024-bit rate 1/2 soft decision decoder with 4-bit messages, [6].

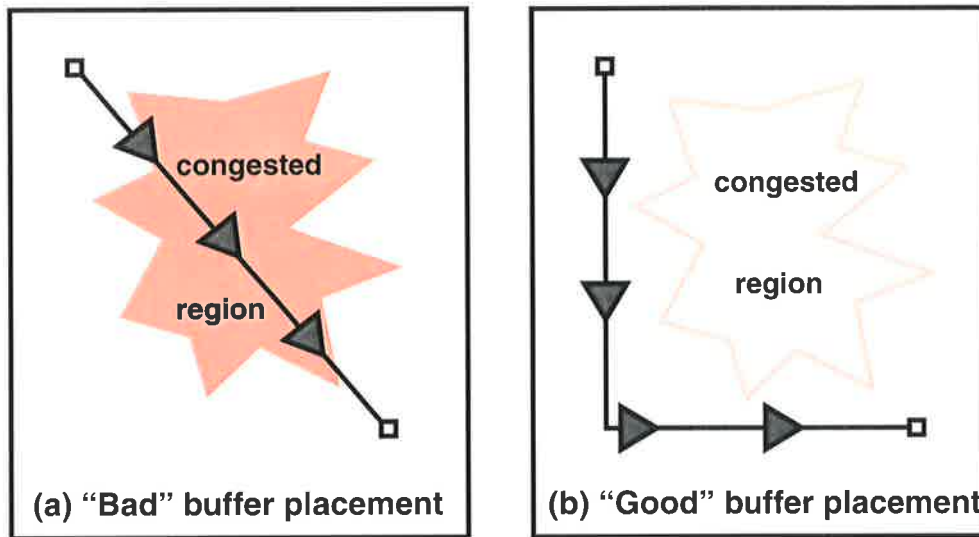


Figure 7.13: Manhattan geometry buffer placement to reduce routing congestion and metal level changes [6].

Long wires with hundreds of vias have a very high resistance. The high resistance results in large RC delay times and a chip which cannot meet timing for the design goal of a 1Gbs^{-1} throughput.

To alleviate the congestion, enable routing and reduce the RC delay of the long wires propagating the messages between the nodes, Blanksby wrote a CAD tool to place repeater buffers using Manhattan geometry [6], as shown in Figure 7.13. The placement of the buffers results in relatively straight wires between the repeaters, less congestion since many wires avoid the centre of the chip, and lower RC delays. The use of custom macro cell placement and buffer insertion were critical to successful implementation of the parallel decoder.

7.6.2 Fabricated Chip

The decoder was fabricated in Agere Systems' 1.5V $0.16\mu\text{m}$ CMOS process with 5 levels of metal. A microphotograph of the chip is shown in Figure 7.14. The $7.5\text{mm} \times 7.0\text{mm}$ chip with 1,750,000 gates achieved a utilisation of 50%. The standard cell density was limited by routing congestion and could therefore be improved by using a fabrication technology with more levels of metalisation.

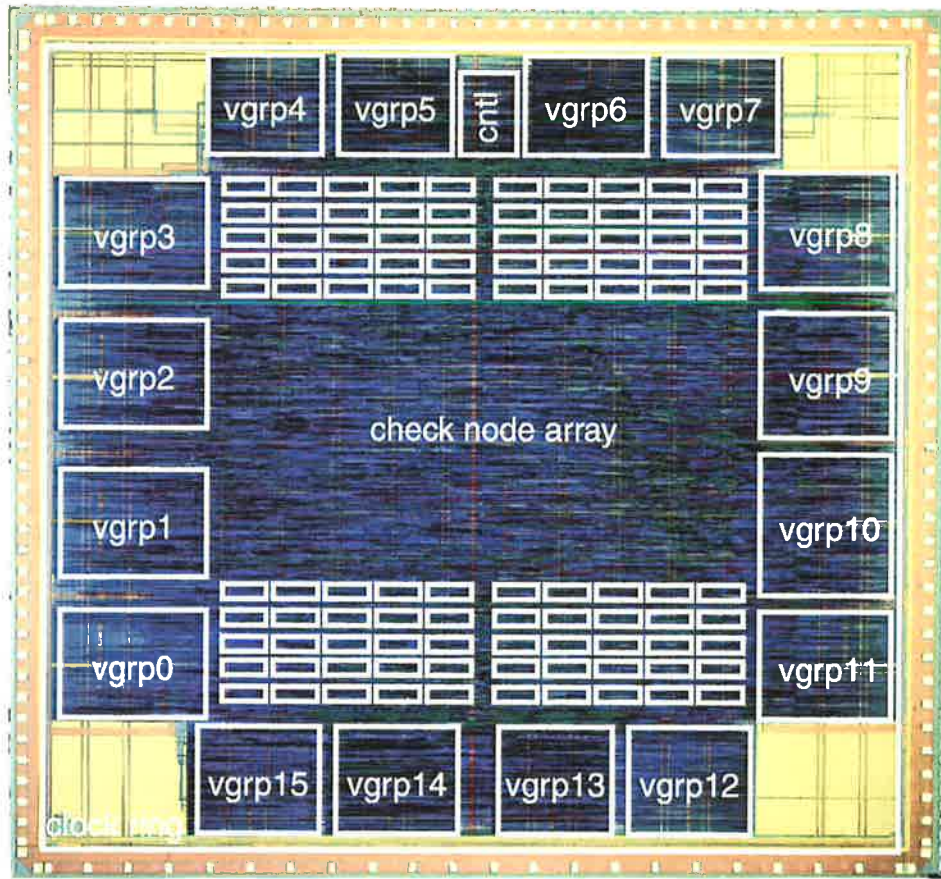


Figure 7.14: Microphotograph of a 1Gb/s^{-1} 1024-bit rate $1/2$ soft decision decoder.

The decoder was tested and functional at 67MHz when supplied with 1.5V, achieving the design target throughput of 1Gb/s^{-1} . By increasing the power supply voltage to 2.25 V the decoder was functional at 100MHz, corresponding to a throughput of 1.5Gb/s^{-1} . Operating with a supply voltage of 0.75V the decoder was functional up to 10MHz, corresponding to a throughput of 15Mbs^{-1} . When the decoder is used for wireless applications, where low power dissipation is extremely important, correct functionality from a very low supply voltage achieves the target of low power dissipation.

7.6.3 Measured Power Dissipation

The measured power dissipation of the decoder when performing 64 decoder iterations is shown in Figure 7.15. It can be seen that the power dissipation is a function of the signal-to-noise ratio as well as the clock frequency, as expected due to the switching

activity of the messages in the decoder. The power dissipation for coded input with a signal-to-noise ratio of zero dB is only slightly lower than that for random data. At a supply voltage of 1.5V there is a lower limit of power dissipation of 4.5mW due to transistor leakage. When the clock frequency is lowered it is possible to reduce the supply voltage and still meet the reduced timing requirements, further reducing the power dissipation of the decoder, as shown for a 3dB input signal to noise ratio. The estimated power dissipation using the switching activity and parasitic capacitances of the message wires and parasitic capacitance of the clock distribution is shown. The estimated power dissipation is a factor of three lower than the actual measured dissipation. The estimation of the switching activity in Figure 7.5 did not include the effect of glitches. There is considerable skew in arrival of the variable messages at the check nodes due to the large variance in the path lengths and associated delays. The check node update is then performed using ripple

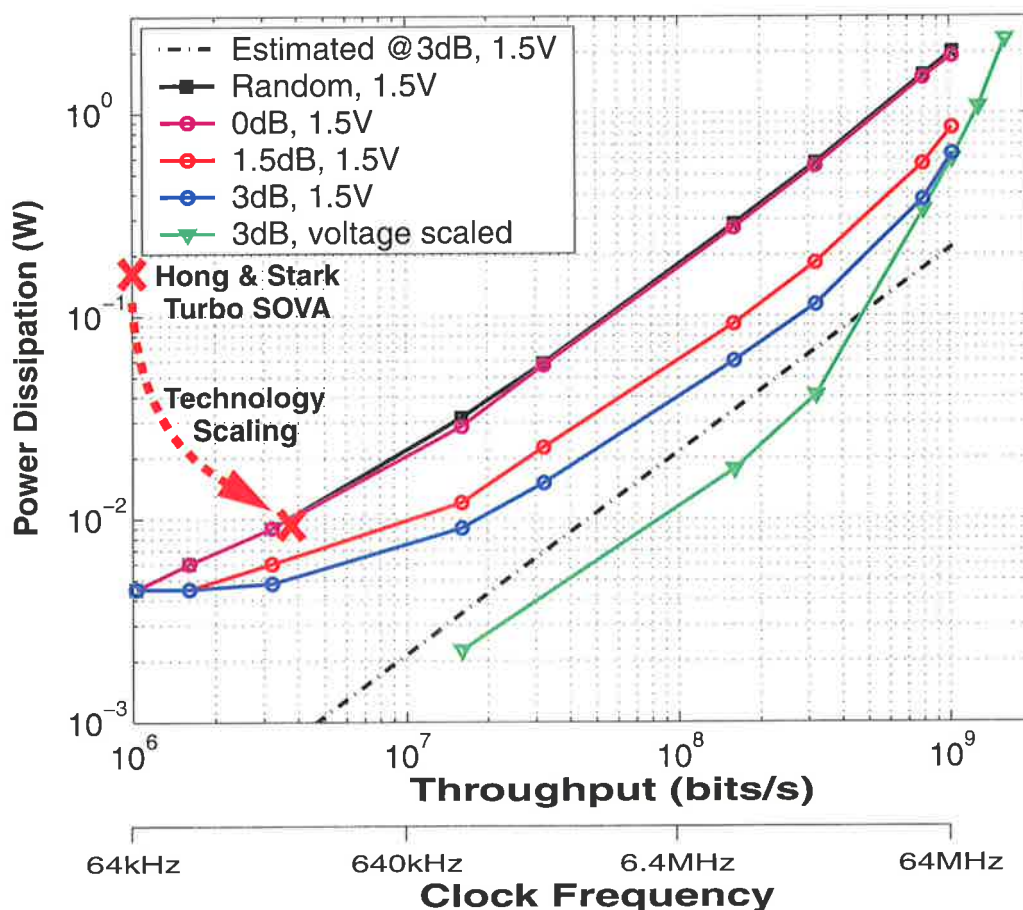


Figure 7.15: Measured power versus throughput for 64 iterations of a 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages compared with Hong and Stark's 256-bit turbo decoder performing 3 iterations.

carry adders, introducing further glitches. The parity check results are also propagated back along long wires to the variable nodes. The long message wires also have very large parasitic capacitances associated with them. At the variable nodes the difference in arrival times of the check node messages can be even larger than that at the input to the check nodes. The update of the variable node is again performed using ripple carry adders and further glitching is introduced as the calculation is performed. A significant portion of the dynamic power in the decoder is due to the glitches of the messages sent from the check nodes to the variable nodes. The dynamic power could therefore be reduced by almost a factor of three by latching the outputs of the check nodes before propagating the values back to the variable nodes. Therefore all future implementations of parallel decoders will include latches at all outputs of the check nodes.

For comparison the power and throughput of a 256-bit rate 1/3 turbo decoder based on the SOVA algorithm [24] is shown in Figure 7.15. The decoder was implemented in 100mm^2 using a 3.3V $0.6\mu\text{m}$ CMOS process and dissipated an estimated 170mW with a throughput of 1Mbs^{-1} . Scaling linearly with process technology feature size, and quadratically with supply voltage gives an estimated power dissipation of 9 mW in a 1.5V $0.16\mu\text{m}$ CMOS process. A linear scaling of throughput with feature size gives a throughput of 3.75Mbits^{-1} . Therefore the parallel LDPC decoder has a power dissipation performance advantage over the turbo code decoder at a signal-to-noise ratio of 3 dB of approximately a factor of two. However, at a throughput of 1 Mb/s the power dissipation of the parallel LDPC decoder is limited by leakage. It can be expected that for higher throughputs, the power dissipation advantage of the parallel LDPC decoder over a turbo code decoder would be substantially higher. A 256-bit SOVA based turbo code decoder also results in a significantly lower coding gain than the 1024-bit LDPC decoder performing 64 decoding iterations.

The turbo code decoder of Hong and Stark uses 100mm^2 of silicon in a $0.6\mu\text{m}$ CMOS process and is dominated by the area required to integrate the memory for the decoder. Scaling quadratically with feature size yields an estimated area of 7mm^2 in a $0.16\mu\text{m}$ CMOS process. By contrast the area of the parallel LDPC decoder is 52.5mm^2 . However, the area of the parallel LDPC code decoder is largely determined by routing

congestion of a data path. Using a CMOS fabrication technology with 6 or 7 levels of metal would result in a significant reduction in the required die size for the LDPC decoder, but not for the turbo code decoder.

Increasing the throughput of the turbo code decoder to match the throughput of the LDPC decoder would require significantly more area, using techniques such as the sliding window Viterbi decoder of Black and Meng [5]. This would result in the area difference between the parallel LDPC decoder and the turbo decoder becoming much smaller.

The power dissipation versus input signal-to-noise ratio for the 1024-bit soft decision decoder operating with a throughput of 1 Gbs^{-1} is shown in Figure 7.16. The dependence of the power dissipation of the decoder on input signal-to-noise ratio matches qualitatively that of the switching activity of the messages shown in Figure 7.5.

A summary of the characteristics of the fabricated decoder are given in Table 7.1

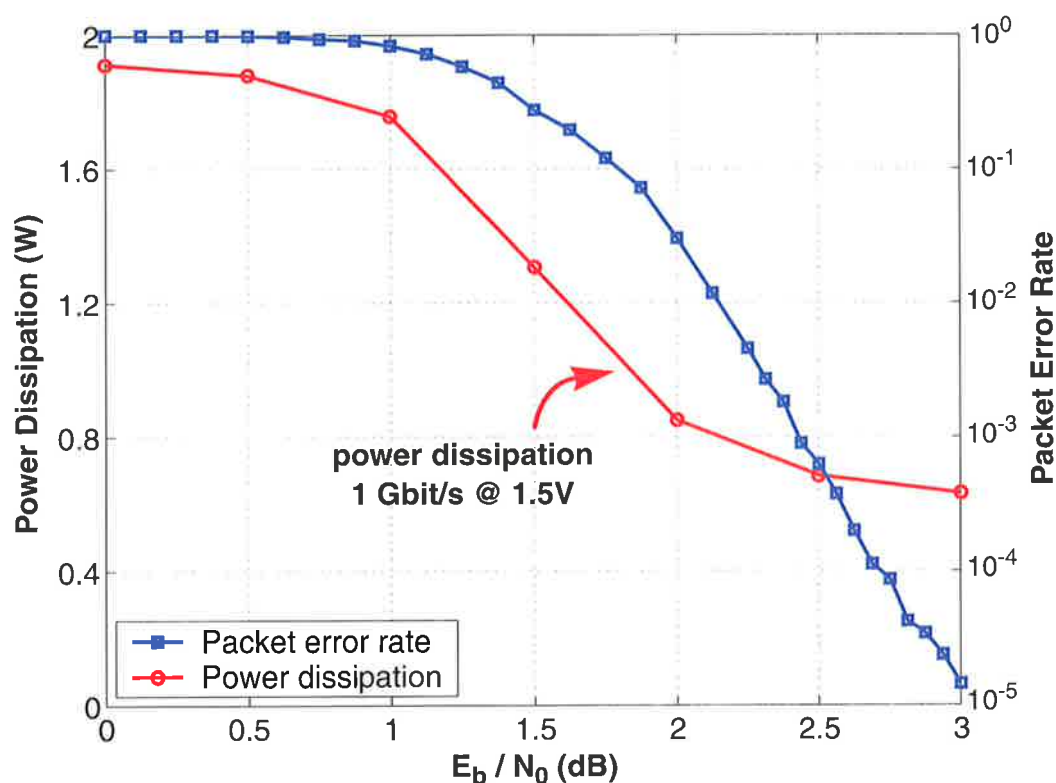


Figure 7.16: Measured power versus signal to noise ratio for a 1024-bit rate 1/2 soft decision decoder operating with a throughput of 1 Gbs^{-1} from a 1.5V supply.

Maximum Clock Frequency		64 MHz	
Maximum Throughput		1 Gbit/s	
Total Power Dissipation		690mW (@ 1.5V)	
	Digital Core	630mW	(91%)
	Pad Frame	60mW	(9%)
Number of Gates		1750 K	
	Variable Node Logic	986K	(56%)
	Check Node Logic	686K	(39%)
	Buffers	75K	(4.7%)
	Control	5K	(0.3%)
Die Size		7.5mm × 7.0 mm	52 mm ²
Utilisation		50%	
Technology		1.5V 0.16µm CMOS 5-LM	

Table 7.1: Summary of device characteristics [6].

7.7 A 32640-Bit Rate 239/255 43Gb/s Hard Decision Decoder

A parallel decoder for the 32,640-bit rate 239/255 code designed in Section 3.6 using the relative reliability weighted hard decision decoding algorithm derived in Section 5.5.2 with a throughput of 43Gbs⁻¹ was implemented in Agere Systems' 1.5V 0.16µm CMOS process with 7 layers of metalisation.

The implementation was the result of collaboration between Andrew Blanksby, Douglas Brinthaup and the author. Douglas Brinthaup is a consulting member of technical staff, Agere Systems. Andrew Blanksby wrote all custom CAD tools, performed all place-and-route, and together with Douglas Brinthaup performed all formal verification, design rule checks and timing analysis. Douglas Brinthaup wrote all of the documentation and VHDL implementation of the interface to the decoder, all synthesis scripts for the interface components, all formal verification scripts for the entire decoder and all timing models and analysis tools. The author wrote the VHDL description of the decoder, matched

the VHDL to a bit accurate C-model of the decoder, designed an optimised netlist for the variable nodes, designed wafer level functional tests for the decoder and optimised the code through column permutations such that the routing congestion due to the graphs structure was reduced.

To achieve the target throughput the parallel decoder is required to operate at approximately 67MHz. A decoding iteration must therefore be completed in approximately 14.9ns. In that period the decoder must

- propagate messages from the latched outputs of the variable nodes,
- perform a 79 or 80 input exclusive-or to form the row parity result,
- buffer and propagated the parity result of each row to all 79 or 80 variable nodes participating in the parity check of each row,
- perform a weighted group summation at the variable nodes of the check messages formed as a function of the outgoing variable messages, the parity results, the received bit and the weight of the received bit.
- determine the value of the variable messages to use in the next decoder iteration by forming the difference of the group sum and each check message, and
- latch the variable messages for the next iteration.

The area required to implement the decoder was approximately 185 mm². It was not possible to propagate messages across a block of this size, through a substantial amount of logic, back across a large distance and again through a significant amount of logic in a single clock period. The decoder was therefore pipelined further so that two data blocks are decoded at the same time. No change to the input and output scan chains is required to implement the pipelining. The change is such that while one data block is propagating variable messages to the parity checks and forming the parity check results, the other data block is propagating the parity check results back to the variable nodes and updating the decoded bit and the variable messages for the next iteration. The decoder latency becomes three data blocks and there are always four data blocks in the decoder at any instant, one being read in, two being decoded and one being read out. When the decoding of a block is

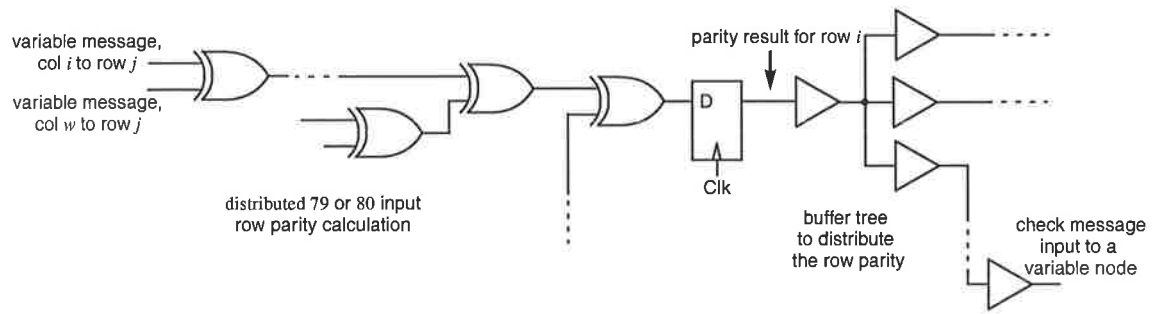


Figure 7.17: Distributed parity check calculation and propagation of the parity check result back to the variable nodes using a buffer tree.

completed another is loaded into the decoder while the decoder has performed exactly 25 and 1/2 decoding iterations on the second block in the decoder. Using an odd number of decoder iterations greatly simplifies the variable node control logic for a two-stage pipelined decoder.

The extra pipelining required is the addition of latches at the output of all parity check exclusive-or trees, as shown in Figure 7.17, and two stage shift register loops for the variable node message latches and the received bit latch, as shown in Figure 7.18. Another difference in the implementation of this decoder is the multiplexing of the received bit back into the received bit scan chain as a new block is loaded and the decoded bits are latched. The decoded bits and the received bits can then be compared at the output of the decoder and a count of the number of bits corrected in a block performed. The number of ones corrected to a zero and vice versa are also counted and used as a feedback signal to adjust the threshold of the off-chip comparator performing the slicing of the analog signal into hard bits.

The variable nodes, including local clock and output message buffers, were implemented using only 128 standard cells. The decoder contains far too many gates to be placed-and-routed as a flat design. The routing congestion problems encountered in the 1024-bit soft decision chip are significantly worse for this decoder which has 163,200 messages from variable nodes from which to form the 2048 parity check results. Therefore a placed-and-routed macro representing a variable node was again constructed for use in a hierarchical decoder implementation.

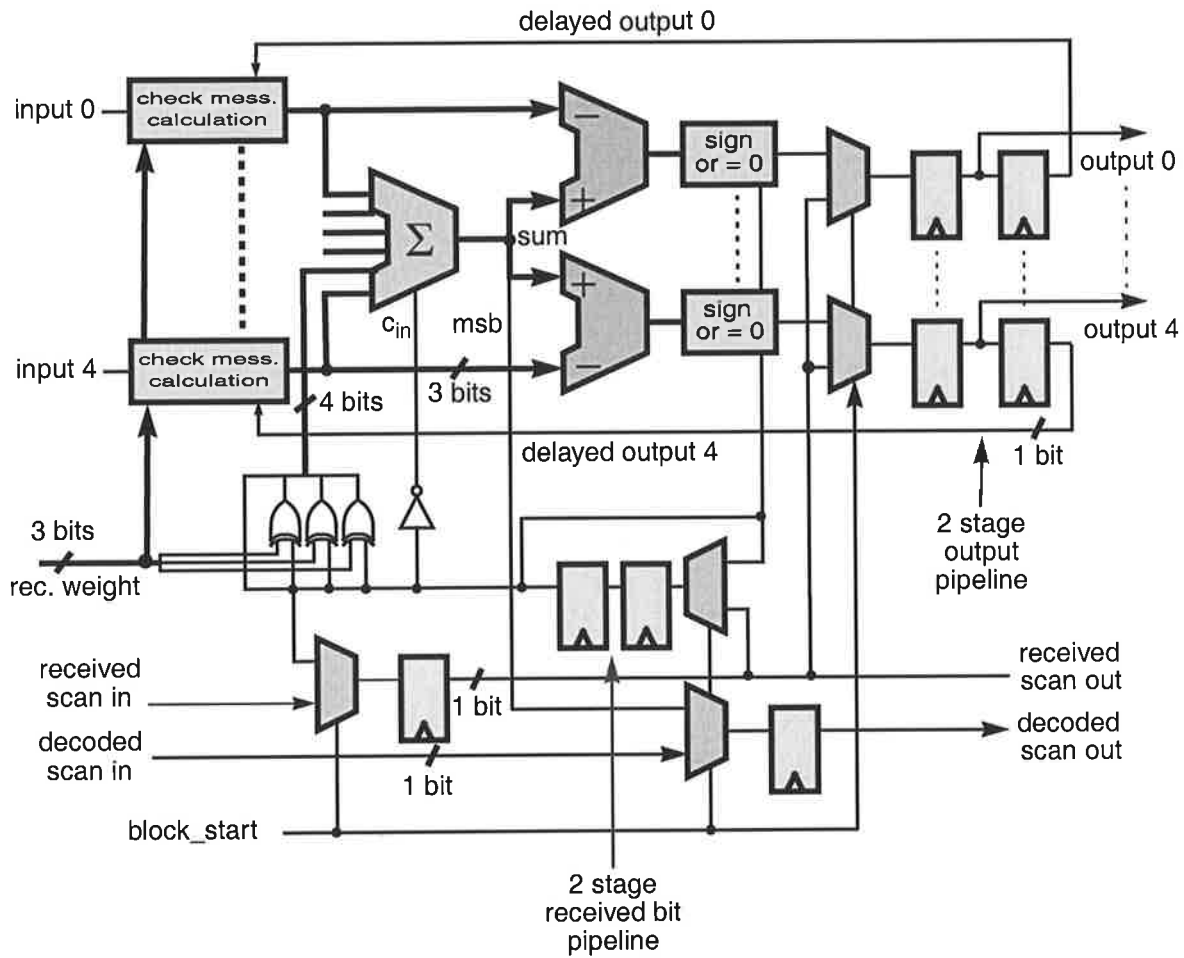


Figure 7.18: Variable node architecture for the two stage pipelined variable node implementing the relative reliability weighted decoding algorithm.

The variable nodes consist of more than 100 mm^2 of standard cells, while the check nodes only require 4 mm^2 of exclusive-or gates. In comparison, the variable node logic was 56% of the 1024-bit rate 1/2 soft decision decoder compared to 39% of the logic for the check nodes. The difference in the ratio of variable node logic to check node logic between the 1024-bit and 32,640-bit decoders is due to both the code rate, one being 1/2 and the other 239/255, and one decoder being a soft decision decoder while the other is a hard decision decoder. The floorplan used to implement the soft decision 1024-bit decoder, wherein the variable and check nodes were both hard macros placed such that the variable nodes surround the check nodes, is therefore not feasible. Such a floorplan would require the routing of the 163,200 variable message wired into an area containing only 4 mm^2 of gates. It was therefore decided to floorplan the decoder as a grid of variable node macros

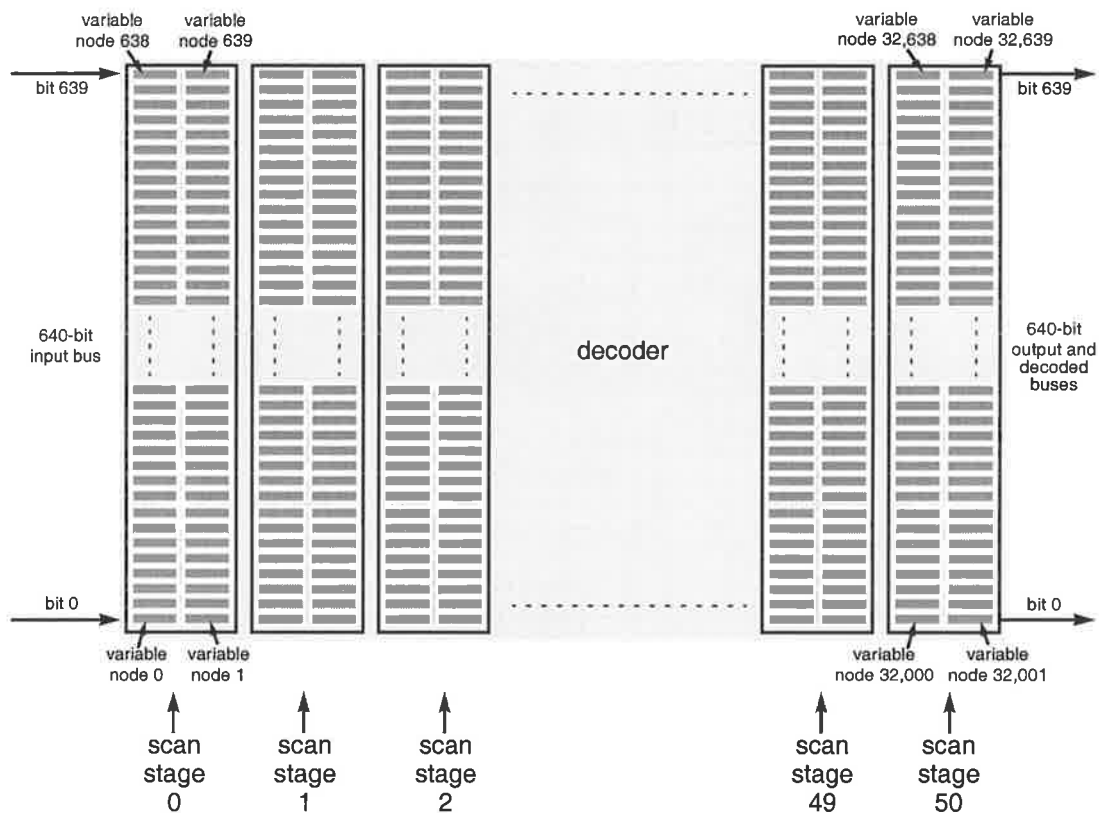


Figure 7.19: Floorplan for the 32,640-bit rate 239/255 hard decision decoder.

and place the parity checks as distributed logic amongst the variable node macros, as shown in Figure 7.19. The decoder was organised such that the incoming data arrives on a 640-bit bus from the left of the decoder, with bit zero in the lower left corner and bit 639 in the upper left corner. The scan chains were constructed as a horizontal connection of variable nodes, scanning data from the input on the left of the decoder to the output on the right of the decoder.

Routing congestion in the centre of the decoder was again a problem. The exclusive-or and buffer trees forming the parity checks were therefore designed and placed using custom software and algorithms, written by Blanksby, such that Manhattan geometry of the connections was enforced. The buffer trees distributing the parity check results were also designed using custom algorithms to place the buffers in positions enforcing Manhattan geometry. The decoder was successfully placed and routed using 7 levels of metal with a standard cell utilisation of over 80 percent. The high density is a result of using highly optimised macros for the variable nodes, with a utilisation of over 95 percent and custom algorithms to place the check node logic and buffer trees.

7.8 Event Driven Decoders

The shortcoming with decoders using memory based message exchange is the required memory bandwidth and irregular memory addressing. While parallel decoders support very high throughputs and have a relatively low power dissipation, the area required to implement them is not suitable for many applications. Therefore a different approach to implementing LDPC decoders may be required for some applications. The average number of iterations required to decode a block of data is significantly lower than the number of iterations a parallel decoder is required to perform to ensure convergence. Early termination of decoding once a decoder has converged and all of the parity checks are satisfied would significantly increase the throughput of a decoder. The parallel decoder architecture is not easily modified to enable early termination of the decoding to be translated into higher throughput. The scan-chain based input-output for a parallel decoder would require modification to act as a first-in-first-out (FIFO) queue to enable variable rate input and output.

Early termination of decoding can more easily be used with a memory based decoder which implements the sum-product algorithm or relative reliability weighted decoding algorithm. The reduction of the required memory bandwidth for high throughput applications is not significant enough to justify resolution of the implementation difficulties. The 32,640-bit rate 239/255 code requires an average of 8.6 decoder iterations to achieve an output bit error rate of 6×10^{-9} . Further reduction of the average number of iterations is small as the bit error rate is reduced further. Soft decision decoding requires significantly more iterations. The average number of decoder iterations of a sum-product decoder for the 32,640-bit rate 239/255 code to obtain a bit error rate of 10^{-8} is 22 iterations. Gallager showed that the average number of decoder iterations required was $O(\log(\log(n)))$ [22]. Di, Urbanke and Richardson have also shown that codes with a large minimum distance require many iterations for the decoder to converge [17]. Therefore increasing the block length of the code will also increase the average number of decoder iterations required. The implementation of very long block length soft decision decoders would therefore seem extremely difficult. Early termination alone is not a solution to the memory bandwidth and address contention problems of a memory based decoder.

When the switching activity of the 1024-bit rate 1/2 soft decision decoder is examined in Figure 7.5 it can be seen that a potential solution to the memory bandwidth problem is to perform the iterative update for a node only if the inputs to the node have changed. A decoder driven by the event of the inputs to a node changing could translate the low message switching activity into a very large reduction in the number of iterative updates performed. The potential reduction of decoder operations and throughput increase can be seen when simulating the parallel decoder using a switch level, event driven VHDL simulator. If the signal to noise ratio results in a low output error rate the simulator runs very quickly and maintains a high simulation throughput as there are few events triggering activity in the decoder. At lower signal to noise ratios the simulation throughput of the decoder is very low due to the much higher activity of the circuit.

Further reduction of the number of operations to decode a block of data could be obtained through the investigation of a serial decoding algorithm rather than the parallel algorithm. If the iterative update is only performed for nodes with the lowest reliabilities then the number of calculations required to decode a block of data could potentially be reduced further.

7.9 Summary

Architectures for implementing decoders for low-density-parity check codes have been examined. Two architectures proposed in the literature have been reviewed, both of which are in the class of memory based message exchange decoders. Neither of these has been demonstrated to be practical through the implementation of the architecture.

A parallel decoder architecture has been proposed to enable the implementation of very high throughput iterative decoders. The switching activity of a parallel decoder was shown to converge when the decoder converges to a valid codeword. The reduced switching activity has been shown to reduce the power dissipation of a parallel decoder. The high throughput and low power capabilities of parallel decoders were demonstrated with the implementation of a 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages. The decoder was implemented in Agere Systems' 1.5V 0.16 μ m CMOS process

with 5 levels of metal and dissipated 690 mW when decoding 1Gbs^{-1} of coded data. The measured power as a function of the input signal-to-noise ratio demonstrated the same dependence as the switching activity, therefore verifying that the reduced switching activity can be used to reduce the power dissipation of the decoder.

A second parallel decoder was implemented to decode the 32,640-bit rate 239/255 code using the relative reliability weighted decoding algorithm with a throughput of 43Gbs^{-1} of coded data, 40Gbs^{-1} of uncoded data. The decoder was implemented in Agere Systems' 1.5V 0.16 μm CMOS process with 7 levels of metal and is currently awaiting fabrication.

Chapter 8

Conclusion

Although Richardson and Urbanke proved that given a long enough block length the performance of random low-density parity-check codes converges to the average code performance [52], for relatively short block lengths random codes can perform significantly worse than carefully constructed codes. Due to the number of permutations possible it is not feasible to generate a large number of random codes and then select the one with the best performance. Therefore an algorithm or heuristic for constructing good codes is required. A novel code construction technique has been proposed based on the minimisation of a cost metric. A very good cost metric which measures short cycles in the graph of the code has been found for use with the proposed algorithm. The method was used to design two codes, a 1024-bit rate $1/2$ irregular code and a 32,640-bit rate $239/255$ regular code.

Encoding low-density parity-check codes using generator matrices is impractical due to the large block length of the codes. Two types of solution to this have been examined. The first type involves modifying the structure of the code to enable single pass encoding using the parity check matrix. Modifying the parity check matrix to simplify encoding results in loss of coding gain and is therefore undesirable. The second method is to encode LDPC codes using an algorithm which solves the simultaneous equations that the parity check matrix represents. Considering the parity check matrix as a set of linear simultaneous equations and solving for the unknown parity bits results in practical and efficient encoders for random LDPC codes. Encoders based on this method can take advantage of the sparse structure of the parity check matrix and reduce the amount of logical operations or hardware required to encode the code. Richardson and Urbanke have proposed such an

encoder for LDPC codes [53]. Although this algorithm is very good for irregular codes it is not efficient for regular codes. A low complexity encoding algorithm for regular codes was therefore developed. Using the proposed encoding algorithm an encoder architecture was also proposed and demonstrated through the implementation of an encoder for the 32,640-bit rate 239/255 code. The use of the new encoding algorithm reduced the number of exclusive-or operations required to implement the dense matrix multiply by a factor of more than 125 compared to Richardson and Urbanke's algorithm. The encoder was implemented to support a throughput of 40Gbs^{-1} of uncoded systematic data, 43Gbs^{-1} of coded data using Agere Systems' 1.5V 0.16 μm CMOS process with 7 layers of metal in an area of 27mm^2 .

Gallager's Algorithm's A and B, expander graph decoding and combinations of them have been examined as potential decoding algorithms for the 32,640-bit rate 239/255 LDPC code to be used in a fiber optic transceiver. The decoding algorithms did not perform adequately for this application, especially when the complexity of implementing the codes is considered. By taking into consideration the relative probability of a parity check being satisfied or unsatisfied and the relative reliability of the messages in the decoder compared to the reliability of the received data, a new decoding algorithm called relative reliability weighted decoding has been developed with significantly better performance than existing hard decision decoding algorithms. Relative reliability weighted decoding requires no additional information exchange between the variable and check nodes of the bipartite graph representation of a decoder and does not require the distribution of control information above that required to implement Gallager's Algorithm B. The only change required to modify the update rules used at the variable nodes in the decoder. When decoding the 32,640-bit rate 239/255 code the relative reliability weighted algorithm provides a coding gain of 8dB at an output bit error rate of 10^{-15} , compared to the 4.4dB of Gallager's Algorithm B, an increase of 3.6dB.

Soft decision decoding algorithms for low-density parity-check codes have been reviewed. The fixed point implementation of the algorithms has been investigated and a decoding algorithm which exchanges 4-bit messages and performs 64 decoding iterations for a 1024-bit rate 1/2 irregular LDPC code has been developed which results in a loss of

less than 0.2 dB of coding gain when compared to a double precision floating point decoder performing 1000 decoding iterations. The extremely small loss of coding gain shows that the sum-product algorithm is very well suited to implementation using very few quantisation levels. The efficient mapping of the algorithm to the fixed point representation is also evident through this small loss.

Architectures for implementing decoders for low-density parity-check codes have been examined. Two architectures proposed in the literature have been reviewed, neither of which has been demonstrated to be practical through the implementation of the architecture.

A parallel decoder architecture was proposed in Section 7.4. A parallel decoder architecture enables the implementation of very high throughput iterative decoders. The switching activity of a parallel decoder was shown to converge when the decoder converges to a valid codeword. The reduced switching activity can be utilised as a method of reducing the power dissipation of a parallel decoder. The high throughput and low power capabilities of parallel decoders was demonstrated with the implementation of a 1024-bit rate 1/2 soft decision decoder exchanging 4-bit messages. The decoder was implemented in Agere Systems' 1.5V 0.16 μ m CMOS process with 5 levels of metal and dissipated 690 mW when decoding 1Gbs⁻¹ of coded data. A second parallel decoder was implemented to decode the 32,640-bit rate 239/255 code designed in Section 3.6 using the relative reliability weighted decoding algorithm derived in Section 5.5.2 with a throughput of 43Gbs⁻¹ of coded data, 40Gbs⁻¹ of uncoded data. The decoder has also been implemented in Agere Systems' 1.5V 0.16 μ m CMOS process with 7 levels of metal.

8.1 Thesis Contributions

This thesis addresses many important issues regarding the implementation low-density parity-check codecs. The contributions presented in this thesis by the author are:

- The development of a minimum cycle cost code construction algorithm for finding LDPC codes that perform significantly better than random codes or codes constructed by only preventing very short length cycles in the graph of the code.
- The development of a practical encoding algorithm optimised for very long block length regular codes.
- The development of an architecture for implementing the encoder of random LDPC code which is capable of achieving very high throughput. The architecture was demonstrated through the implementation of an encoder with a coded data throughput of 43Gbs^{-1} .
- The development of a new hard decision decoding algorithm, relative reliability weighted decoding, which uses the relative probability of a parity check being satisfied and unsatisfied to infer greater information transfer between the nodes of a decoder while not increasing the number of bits sent.
- The development of a fixed point implementation of a soft decision sum-product decoding algorithm which results in only 0.14dB degradation of coding gain compared to decoding using double precision floating point accuracy.
- The implementation of the first published low-density parity-check code decoder as an application specific integrated circuit. The device was successfully tested and demonstrated the very high throughput and low power potential of a parallel decoder architecture. The decoder supports a coded data throughput of 1Gbs^{-1} while dissipating 690mW from a 1.5V supply.

- The implementation of a 43Gbs^{-1} decoder for fiber optic communications systems. Since commercial parts implementing a Reed Solomon decoder are still under design and not yet available the author believes this decoder to be the highest throughput iterative decoder to have been implemented to date.

8.2 Further Work

There are a number of areas the author believes important contributions can be made in the future. Methods of constructing low-density parity-check in a regular and systematic way, through algebraic or projective geometric constructions, structured matrices or convolutional codes, which result in codes with a large girth should be investigated. Codes of this type would enable the simplification of the encoding process and offer the potential to simplify a decoder using memory to exchange messages.

A large potential to improve the performance of hard decision decoding has been demonstrated through the heuristic development of the relative reliability weighted decoding algorithm. A formal mathematical derivation should be undertaken to further optimise the algorithm. Derivation of optimal values for the relative weighting of messages in the variable node should be undertaken. The oscillation of the weighting of the received bits should also be further investigated.

The development of a fixed point soft decision decoding algorithm which exchanges 4-bit messages, and its subsequent implementation, has demonstrated the suitability of soft decision decoding of LDPC codes to implementation. Soft decision decoding can be further optimised to reduce the gate count required to implement the decoders, thus making the decoder more suitable for consumer products. One potential idea comes from neural networks where much research has indicated that the particular non-linear function used in the update rule is not particularly important. The use of non-linear functions other than the hyperbolic tangent function should be examined as a method of simplifying the implementation of soft decision decoders. Another area to investigate is the form of the messages exchanged by the decoder. The information exchange between the functional nodes may better utilise the available dynamic range if a representation other than log-likelihood

messages is used. Richardson and Urbanke have proposed a 3-bit soft decision decoder for regular $(3,6)$ codes which passes messages in the non-linear domain of the check nodes [52].

The demonstration of the extremely low switching activity of the messages exchanged by a parallel decoder for LDPC codes could be exploited further. The use of a serial decoding algorithm may further reduce the number of operations required to decode LDPC codes. The implementation of event driven decoders offers the potential to reduce the amount of hardware required to implement LDPC decoders. Using a programmable decoder with memory based message exchange also offers the potential to decode different LDPC codes using the same hardware. Therefore programmable event driven decoders could potentially reduce the hardware required to implement a decoder compared to a parallel implementation, while not reducing the throughput of the decoder.

Patents

European patent number: 01304531.5-2206

“Methods and apparatus for decoding of general codes on probability dependency graphs”, A.J. Blanksby and C.J. Howland.

United States of America patent application:

“Methods and apparatus for decoding of general codes on probability dependency graphs”, A.J. Blanksby and C.J. Howland, filed May 2000.

Publications

C.J. Howland and A.J. Blanksby, “Parallel Decoding Architectures for Low Density Parity Check Codes”, In *Proceedings of IEEE International Symposium of Circuits and Systems ‘ISCAS 2001’*, vol. 4, pp 742-745 May 2001.

C.J. Howland and A.J. Blanksby, “A 220mW, 1024-Bit, Rate-1/2, 1Gbs⁻¹ Low Density Parity Check Code Decoder”, In *Proceedings of IEEE Custom Integrated Circuits Conference ‘CICC2001’*, May 2001.

A.J. Blanksby and C.J. Howland, “A 630 mW, 1Gbs⁻¹ 1024-Bit, Rate-1/2 Low-Density Parity-Check Code Decoder”, submitted to and accepted for publication in the *IEEE Journal of Solid State Circuits*, 2001, invited paper for the special edition on ‘CICC2001’.

Bibliography

- [1] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transactions on Information Theory*, pp 363-77, March 1974.
- [2] J. Baylis, "Error-Correcting Codes", Chapman & Hall, 1998.
- [3] L. Bazzi, T. Richardson, R.L. Urbanke, "Exact Thresholds and Optimal Codes for The Binary Symmetric Channel and Gallager's Decoding Algorithm A", submitted to *IEEE Transactions on Information Theory*.
- [4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting codes and decoding", *Proc. Int. Conf. Comm. '93*, pp 1064-1070, May 1993.
- [5] P.J. Black and T.H.Y. Meng, "A 1-Gb/s, Four-State, Sliding Block Viterbi Decoder", *IEEE Journal of Solid State Circuits*, Vol. 32 No. 6, pp 797-805, June 1997.
- [6] A.J. Blanksby and C.J. Howland, "A 630 mW, 1Gbs⁻¹ 1024-Bit, Rate-1/2 Low Density Parity Check Code Decoder", submitted to *IEEE Journal of Solid State Circuits*, 2001, accepted for publication.
- [7] M. Bossert, *Channel Coding for Telecommunications*, John Wiley & Sons, 1999.
- [8] J. Boutros, O. Pothier and G. Zemor, "Generalized Low Density (Tanner) Codes", *ICC'99*, Vancouver Canada, June 1999.
- [9] D. Burshtein and G. Miller, "Expander Graph Arguments for Message-Passing Algorithms", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 782-790, Feb. 2001.
- [10] Y. Cai, N. Ramanujam, J.M. Morris, T. Adali, G. Lenner, A. B. Puc and A. Pilipetskii, "Performance Limit of Forward Error Correction Codes in Optical Fiber Communications", proceedings of the *Optical Society of America's Optical Fiber Communications Conference 2001 'OFC2001'*.
- [11] J.F. Cheng, "Iterative Decoding", Ph.D. dissertation, California Institute of Technology, Pasadena, California, 1997.

- [12] S.Y. Chung, "On the Construction of Some Capacity-Approaching Coding Schemes", Ph.D. dissertation, Mass. Institute of Technology, Cambridge, MA, 2000. Available at <http://truth.mit.edu/~sychung>
- [13] S.Y. Chung, T.J. Richardson and R.L. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 657-670, Feb. 2001.
- [14] S.Y. Chung, G.D. Forney, T.J. Richardson and R. Urbanke, "On the Design of Low-Density Parity-Check Codes Within 0.0045 dB of the Shannon Limit", To appear in *IEEE Communication Letters*.
- [15] T.M. Cover and J.A. Thomas, "Elements of Information Theory", John Wiley & Sons, 1991.
- [16] M.C. Davey and D. MacKay, "Low Density Parity Check Codes Over $GF(q)$ ", *IEEE Communications Letters*, Vol. 2, No. 6, June 1998.
- [17] C. Di, R.L. Urbanke and T.J. Richardson, "Weight Distributions: How Deviant Can You Be?", In *Proceedings of International Symposium on Information Theory 'ISIT2001'*, pp 50, June 2001.
- [18] A. Felstrom and K. Zigangirov, "Time-Varying Periodic Convolutional Codes With Low-Density Parity-Check Matrix", *IEEE Transactions on Information Theory*, Vol. 45, No. 6, pp 2181-2191, September 1999.
- [19] G.D. Forney, "Codes on Graphs: Normal Realizations", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 520-548, Feb. 2001.
- [20] M. Fossorier, M. Mihaljevic and H. Imai, "Reduced Complexity Iterative Decoding of Low Density Parity Check Codes Based on Belief Propagation", *IEEE Transactions on Communications*, Vol. 47, no. 5, pp 673-680, May 1999.
- [21] R. G. Gallager, "Low density parity check codes", *IRE Transactions on Information Theory*, Vol. IT-8, pp 21-28, Jan. 1962.
- [22] R. G. Gallager, "Low-Density Parity-Check Codes", MIT Press, Cambridge, MA, 1963.
- [23] D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger, J.R. Wall, "Coding Theory: The Essentials", Marcel Dekker, 1991.

- [24] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications", *Journal of VLSI Signal Processing*, Vol. 24, pp 43-57, 2000.
- [25] C.J. Howland and A.J. Blanksby, "Parallel Decoding Architectures for Low Density Parity Check Codes", In *Proceedings of IEEE International Symposium of Circuits and Systems 'ISCAS 2001'*, vol. 4, pp 742-745, May 2001.
- [26] C.J. Howland and A.J. Blanksby, "A 220 mW, 1024-Bit, Rate-1/2, 1 Gbs⁻¹ Low Density Parity Check Code Decoder", In *Proceedings of IEEE Custom Integrated Circuits Conference 'CICC2001'*, May 2001.
- [27] F.R. Kschischang and B.J. Frey, "Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models", *IEEE Journal on Selected Areas in Communications*, Vol. 16. No. 2, pp 219-30, Feb. 2000.
- [28] F. R. Kschischang, B. J. Frey and H.A. Loeliger, "Factor Graphs and the Sum-Product Algorithm", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 498-519, Feb. 2001.
- [29] M. Lentmanier and K.S. Zigangirov, "On Generalized Low-Density Parity-Check Codes Based on Hamming Component Codes", *IEEE Communications Letters*, Vol. 3, No. 8. August 1999.
- [30] S. Lin, and D.J. Costello (jr.), "Error Control Coding: Fundamentals and Applications", Prentice Hall, 1983.
- [31] S. Lin, H. Tang and Y. Kou, "On a Class of Finite Geometry Low Density Parity Check Codes", In *Proceedings of International Symposium on Information Theory 'ISIT2001'*, pp 2, June 2001.
- [32] M.G. Luby, M. Mitzenmacher, A. Shokrollahi, D.A. Spielman and V. Stemann, "Practical Loss-Resilient Codes", *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pp 150-159, 1997.
- [33] M.G. Luby, M. Mitzenmacher, A. Shokrollahi and D.A. Spielman, "Analysis of low density codes and improved designs using irregular graphs", *Proceedings 30th Annual ACM Symposium on the Theory of Computing*, pp 249-258, 1998.

- [34] M.G. Luby, M. Mitzenmacher, A. Shokrollahi and D.A. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation", *Proceedings 1998 IEEE International Symposium on Information Theory*, pp 117, 1998.
- [35] M.G. Luby, M. Mitzenmacher, A. Shokrollahi and D.A. Spielman, "Efficient Erasure Correcting Codes", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 569-584, Feb. 2001.
- [36] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes", *Electronic Letters*, Vol. 32(18), pp 1645-1646, Aug. 1996.
- [37] D. MacKay and M.C. Davey, "Evaluation of Gallager Codes for Short Block Length and High Rate Applications", in *Codes, Systems and Graphical Models*, B. Marcus and J. Rosenthal (Editors), Heidelberg Springer, vol. 123, pp 113-130, 2000.
- [38] D. MacKay, S.T. Wilson and M.C. Davey, "Comparison of Constructions of Irregular Gallager Codes", *Proceedings of 1998 Allerton Conference on Communication, Control and Computing*, 1998
- [39] D. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices", *IEEE Transactions on Information Theory*, Vol. 45, No. 2, pp 399-431, March 1999.
- [40] F.J. MacWilliams and N.J.A. Sloane, "The theory of Error-Correcting Codes", *North-Holland publishing company*, 1977.
- [41] G.A. Margulis, "Explicit Constructions of Graphs Without Short Cycles and Low Density Codes", *Combinatorica*, vol. 2, no. 1, pp 71-78, 1982.
- [42] T. Mittelholzer, A. Dholakia and E. Eleftheriou, "Reduced-Complexity Decoding of Low Density Parity Check Codes for Generalized Partial Response Channels", *IEEE Transactions on Magnetics*, Vol. 37, No. 2, pp 721-728, March 2001.
- [43] T. R. Oenning and J. Moon, "A Low-Density Generator Matrix Interpretation of Parallel Concatenated Single Bit Parity Codes", *IEEE Transactions on Magnetics*, Vol. 37, No. 2, pp 737-741, March 2001.
- [44] J. Pearl, "Probabilistic Reasoning in Intelligent Systems: Networks and Plausible Inference", Morgan Kaufmann, San Mateo 1988.

- [45] S.S. Pietrobon, "Implementation and Performance of a Serial MAP Decoder For Use in an Iterative Turbo Decoder", in *Proceedings of the IEEE International Symposium on Information Theory*, Whistler, B.C., Canada, 1995.
- [46] S.S. Pietrobon, "Implementation and Performance of a Turbo./MAP Decoder", submitted to *International Journal of Satellite Communications*, 21 February 1997, revised 2 April 1998.
- [47] L. Ping W.K. Leung and N. Phamdo, "Low Density Parity Check Codes with Semi-Random Parity Check Matrices", *Electronics Letters*, Vol. 35, No. 1, pp 38-39, Jan. 1999.
- [48] L. Ping and W.K. Leung, "Decoding Low Density Parity Check Codes With Finite Quantization Bits", *IEEE Communications Letters*, Vol. 4, No. 2, pp 62-64, Feb. 2000.
- [49] L. Ping and K.Y. Wu, "Concatenated Tree Codes: A Low-Complexity, High-Performance Approach", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 791-799, Feb. 2001.
- [50] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes", *Proceedings IEEE GLOBECOM '94*, pp 339-343, 1994.
- [51] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke, "Design of Provably Good Low-Density Parity Check Codes", submitted to *IEEE Trans. Inf. Theory*, March 1999.
- [52] T.J. Richardson and R.L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 599-618, Feb. 2001.
- [53] T.J. Richardson and R.L. Urbanke, "Efficient Encoding of Low-Density Parity-Check Codes", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 638-656, Feb. 2001.
- [54] 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*, Vol. 47, no. 2, pp 619-637, Feb. 2001.

- [55] J. Rosenthal and P.O. Vontobel, "Construction of Regular and Irregular LDPC Codes Using Ramanujan Graphs and Ideas from Margulis", In *Proceedings of International Symposium on Information Theory 'ISIT2001'*, pp 4, June 2001.
- [56] C. E. Shannon, "A Mathematical Theory of Communication", *Bell Systems Technical Journal*, no. 27 pp 379-423 and 623-656, July & Oct. 1948.
- [57] M. Sipser and D.A. Spielman, "Expander Codes", *IEEE Transactions on Information Theory*, Vol 42. pp 1710-1722, Nov. 1996..
- [58] L.L. Song and M.L. Yu, "Design and Implementation of Forward Error Correction Devices for Optical Communication Systems", *Bell Laboratories Technical Memorandum*, 2001.
- [59] V. Sorokine, F.R. Kschischang and S. Pasupathy, "Gallager Codes for CDMA Applications-Part I: Generalizations, Constructions and Performance Bounds", *IEEE Transactions on Communications*, vol. 48, no.10, pp 1660-1668, Oct. 2000.
- [60] V. Sorokine, F.R. Kschischang and S. Pasupathy, "Gallager Codes for CDMA Applications-Part II: Implementations, Complexity and System Capacity", *IEEE Transactions on Communications*, vol. 48, no.10, pp 1818-1628, Nov. 2000.
- [61] D.A. Spielman, "Linear Time Encodable and Decodable Error-Correcting Codes", *IEEE Transactions on Information Theory*, Vol 42. pp 1723-1731, Nov. 1996.
- [62] D. Sridhara and T.E. Fuja, "Bandwidth Efficient Modulation Based on Algebraic Low Density Parity Check Codes", In *Proceedings of International Symposium on Information Theory 'ISIT2001'*, pp 165, June 2001.
- [63] R.M. Tanner, "A Recursive Approach to Low Complexity Codes", *IEEE Transactions on Information Theory*, Vol. 42 no. 6, pp 533-547, 1981.
- [64] A.J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes", *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 2, Feb. 1998.
- [65] P.O. Vontobel and R.M. Tanner, "Construction of Codes Based on Finite Generalized Quadrangles for Iterative Decoding", In *Proceedings of International Symposium on Information Theory 'ISIT2001'*, pp 223, June 2001.

- [66] Q. Wang, L. Wei, "Graph-Based Iterative Decoding Algorithms for Parity-Concatenated Trellis Codes", *IEEE Transactions On Information Theory*, Vol. 47, No. 3, March 2001.
- [67] N. Wieberg, "Codes and Decoding on General Graphs", PhD thesis, University of Linköping, Sweden, 1996.
- [68] N. Wieberg, H. A. Loeliger and R. Kotter, "Codes and Iterative Decoding on General Graphs", *European Transactions on Telecomm.*, Vol 6, no. 5, pp 513-526, Sept. 1995.
- [69] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels", *IEEE Transactions on Magnetics*, Vol. 37, No. 2, pp 748-755, March 2001.
- [70] G. Zemor, "On Expander Codes", *IEEE Transactions on Information Theory*, Vol. 47, no. 2, pp 835-837, Feb. 2001.
- [71] T. Zhang, Z. Wang and K.K. Parhi, "On Finite Precision Implementation of Low Density Parity Check Codes Decoder", In *Proceedings of IEEE International Symposium of Circuits and Systems 'ISCAS 2001'*, vol.4 pp 202-205, May 2001.
- [72] V. Zybalov and M. Pinsker, "Estimation of the Error-Correction Complexity of Gallager Low-Density Codes", (in Russian):*Probl. Pered. Inform.*, vol. 11, pp 23-26, Jan. 1975. (English Translation): *Probl. Inform. Trans.*, vol. 11, no. 1, pp 18-28, 1976.
- [73] "3rd Generation Partnership Project (3GPP); Technical Specification Group Radio Access Network Multiplexing and channel coding (TDD)", available at <http://www.3gpp.org>
- [74] Telecommunications Standardization Sector, "Forward Error Correction for Submarine Systems", *Technical report, International Telecommunication Union*, 1996.

