

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

低密度對偶檢查碼結構之改進以及其解碼器之
超大型積體電路實現



An Improved LDPC Code Structure and Its VLSI
Decoder Realization

研究生：朱元志

指導教授：陳紹基 博士

中華民國九十四年七月

低密度對偶檢查碼結構之改進以及其解碼器之超大型
積體電路實現

**An Improved LDPC Code Structure and Its VLSI
Decoder Realization**

研究生：朱元志

Student：Yuan-Jih Chu

指導教授：陳紹基 博士

Advisor：Sau-Gee Chen

國立交通大學

電子工程學系 電子研究所碩士班



Submitted to Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

July 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年七月

低密度對偶檢查碼結構之改進以及其解碼器之超大型積體電路實現

學生：朱元志

指導教授：陳紹基 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘 要

由於低密度對偶檢查碼 (LDPC) 的編碼增益接近向農 (Shannon) 極限以及解碼程序上擁有低複雜度的特性，所以在近年來受到廣泛的討論。本文中，我們利用差分集合 (difference family) 的概念來建構一種新的低密度對偶檢查碼結構，此結構在編碼上擁有低複雜度的特性，以及在解碼器的設計上易於超大型積體電路 (VLSI) 實現。此外，在解碼器的設計上，我們使用部分平行 (semi-parallel) 的架構並使其平行度為 10，設計一個碼率為 $3/4$ 、長度為 960 位元、最大循環解碼次數為 10 的非規則低密度對偶檢查碼解碼器，在 $0.18\ \mu\text{m}$ 製程下，此解碼器之資料流為每秒 370MHz、面積為 80 萬個邏輯閘、消耗功率為 550mW。

An Improved LDPC Code Structure and Its VLSI Decoder Realization

Student: Yuan-Jih Chu Advisor: Dr. Sau-Gee Chen

Department of Electronics Engineering &

Institute of Electronics

National Chiao Tung University



In recent years, low-density parity-check (LDPC) codes have attracted a lot of attention due to the near Shannon limit coding gain when iteratively decoded. In this thesis, we construct a new structure of irregular LDPC codes based on using the difference families. The resulting codes can be encoded with low complexity and are suitable for the VLSI implementation of their decoder. With the semi-parallel architecture and a parallel factor of 10, an irregular LDPC decoder has been implemented, of which the code rate is $3/4$, the code length is 960 bits, and the maximum number of decoding iterations is 10, respectively. The irregular LDPC decoder can achieve a data decoding throughput of up to 370Mbps, an area of 800k gate counts, and a power consumption of 550mW using the UMC 0.18 μm ASIC process technology.

誌 謝

本篇論文的完成承蒙指導教授 陳紹基博士兩年多來的悉心指導教誨，讓我能夠確立研究的方向，給予我多方面的協助，在此至上由衷的感激。

其次，感謝曲健全學長以及廖彥欽學姊無私地提供協助，使我受益良多。謝謝實驗室的同學世民、觀易、佳旻、偉廷以及承穎，謝謝你們在課業及生活上給予我許多的幫助。還有實驗室的學弟妹們，謝謝你們帶給我們許多美好的回憶。

最後，感謝我的家人在背後支持與鼓勵我，還有在天上的爸爸，因為有你的支持及栽培，我才能順利的完成學業，謹致上無限的敬意與感激。



Contents

中文摘要.....	I
ABSTRACT	II
ACKNOWLEDGEMENT	III
CONTENTS.....	IV
LIST OF TABLES	VI
LIST OF FIGURES	VII
Chapter 1 Introduction.....	1
Chapter 2 Low-Density Parity-Check Code.....	3
2.1 Fundamental Concept of LDPC Code	3
2.2 Code Construction	7
2.3 Encoding	10
2.4 Decoding	17
2.4.1 Decoding Procedure in One Iteration	18
2.4.2 Iterative Decoding Procedure	23
2.4.3 Efficient Check Node Computation.....	25
Chapter 3 A New Structure for Low-Density Parity-Check Code Using the Difference Family.....	33
3.1 The Difference Family	33
3.2 The Proposed Structure of LDPC Code.....	35
Chapter 4 Simulation Results	39
4.1 Floating-Point Simulations	42
4.2 Fixed-Point Simulations.....	46
4.2.1 Quantization of Initially Received Signal.....	46
4.2.2 Quantization of $r_{m,l}$ and $q_{m,l}$	50
4.2.3 Summary of Fixed-Point Simulation Results	54
Chapter 5 VLSI Implementation of LDPC Decoder	56
5.1 Semi-parallel Decoder Architecture for the Proposed LDPC Codes	56
5.2 Architectures of the Check Node Function Unit and the Variable Node Function Unit	59

Chapter 6 Conclusion76

References78

Autobiography81



List of Tables

Table 2.1	Efficient computation step of $p_1^T = -\gamma^{-1}(-ET^{-1}A + C)s^T$14
Table 2.2	Efficient computation step of $p_2^T = -T^{-1}(As^T + Bp_1^T)$14
Table 2.3	Summary of Richardson’s encoding procedure.14
Table 2.4	Summary of the sum-product algorithm29
Table 2.5	Summary of the min-sum based algorithm.....30
Table 2.6	Summary of the min-sum algorithm31
Table 4.1	Polynomials of each of the circulant matrices of the proposed irregular LDPC codes40
Table 4.2	Polynomials of each of the circulant matrices of the quasi-cyclic irregular LDPC codes41
Table 5.1	Area, speed, latency and power consumption of the reformulated CNFUs and VNFUs architectures for the sum-product algorithm70
Table 5.2	Area, speed, latency and power consumption of the re-mapped CNFUs and VNFUs architectures for the sum-product algorithm.....70
Table 5.3	Area, speed, latency and power consumption of the CNFUs and VNFUs architectures for the min-sum based algorithm71
Table 5.4	Summary of comparison the area, speed and power consumption of the different CNFUs and CNFUs architectures for the sum-product algorithm and the min-sum based algorithm71
Table 5.5	Summary of the proposed LDPC decoder74
Table 5.6	Comparison of LDPC decoders74
Table 5.7	Basic MCS set of TGnSync proposal75

List of Figures

Figure 2.1	Example of a (8, 4, 2)-regular LDPC code and its corresponding Tanner graph. There are 8 variable nodes (v_i) and 4 check nodes (c_i)..	4
Figure 2.2	Example of a low-density parity-check code matrix where $(n, j, k) = (20, 3, 4)$	7
Figure 2.3	Example of a rate-1/2 quasi-cyclic code from two circulant matrices, where $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$	10
Figure 2.4	The parity-check matrix in an approximate lower triangular form.....	12
Figure 2.5 (a)	Example of a rate-1/2 quasi-cyclic code. (a) Parity-check matrix with two circulants, where $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$	17
Figure 2.5 (b)	Example of a rate-1/2 quasi-cyclic code. (b) Corresponding generator matrix in systematic form	17
Figure 2.6	Notations for iterative decoding procedure.....	24
Figure 2.7	Serial configuration for computing check node update	26
Figure 2.8	Parallel configuration for computing check node update	28
Figure 4.1	Floating-point simulations of various parity-check matrix structures in AWGN channel, code length=720, code rate=2/3, maximum iteration=10, using the sum-product algorithm.....	43
Figure 4.2	Floating-point simulations of various parity-check matrix structures in AWGN channel, code length=960, code rate=3/4, maximum iteration=10, using the sum-product algorithm.....	43
Figure 4.3	Floating-point simulations of various parity-check matrix structures in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10, using the sum-product algorithm.....	44
Figure 4.4	Floating-point simulations of the proposed parity-check matrix	

	structure, under the three decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	44
Figure 4.5	Floating-point simulations of the proposed parity-check matrix structure, under the three decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	45
Figure 4.6	Floating-point simulations of the proposed parity-check matrix structure, under the three decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	45
Figure 4.7	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	47
Figure 4.8	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	48
Figure 4.9	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	48
Figure 4.10	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	49
Figure 4.11	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding	

	algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	49
Figure 4.12	Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	50
Figure 4.13	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	51
Figure 4.14	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	51
Figure 4.15	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	52
Figure 4.16	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	52
Figure 4.17	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	53

Figure 4.18	Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	53
Figure 4.19	Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10.....	54
Figure 4.20	Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10.....	55
Figure 4.21	Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10.....	55
Figure 5.1	Semi-parallel decoder for the proposed irregular LDPC code structure of rate 3/4, and code length 960.....	57
Figure 5.2	Illustration of overlapped decoding procedure	59
Figure 5.3	Function plot of $\phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$	60
Figure 5.4	Architecture of check node function unit for the sum-product algorithm	61
Figure 5.5	Architecture of a 4-input variable node function unit for the sum-product algorithm.....	61
Figure 5.6	Reformulated architecture of check node function unit for the	

	sum-product algorithm.....	63
Figure 5.7	Reformulated architecture of a 4-input variable node function unit for the sum-product algorithm.....	63
Figure 5.8	Re-mapped architecture performing both check node update and variable node update operations for the sum-product algorithm	65
Figure 5.9	Function plot of $g(x) = \ln(1 + e^{- x })$	66
Figure 5.10	Function plot of $h(x) = \ln(e^x - 1)$	67
Figure 5.11	Architecture of check node function unit for the min-sum based algorithm.....	68
Figure 5.12	Architecture of a 4-input variable node function unit for the min-sum based algorithm.....	68



Chapter 1

Introduction

With the continuous growth of wireless communication technology, people have eventually become addicted to wireless products such as mobile phones and wireless LAN due to the convenience and enjoyment it has brought to our lives. However, the resources of the wireless frequency spectra are limited and valuable. The improvement of transmission efficiency for wireless communication has therefore become the focus of research in communication systems. The use of error correction codes is one of the main solutions to raising the transmission efficiency. Among various error correction codes, one called low-density parity-check code (LDPC) should be especially taken into account. LDPC codes were first presented by Gallager [1] in 1962 and have received great attention recently due to, its near Shannon limit coding gain when iterative decoded [2]. LDPC codes are currently widely considered a serious competitor to the turbo codes. The main advantages of LDPC codes over turbo codes are that LDPC decoders are known to require an order of magnitude less arithmetic computations, and the decoding algorithm for LDPC codes is parallelizable and can potentially be accomplished at significantly greater speeds. The disadvantage of the LDPC codes is the high complexity required in encoding. Recently, several efficient encoding approaches have been proposed [3,4,5]. In [5], it introduced an approach that used difference families to construct irregular quasi-cyclic codes free of

4-cycles while reducing the encoding complexity to become linear to the code length. However, the performance was not as good as expected. The aim of this thesis is to construct a new structure of LDPC codes that improves the performance while using the concept of the difference families, and contact VLSI design of the corresponding decoder.

This thesis is organized as follows. In chapter 2, basic concept of the LDPC codes including the code construction, encoding and decoding will be introduced. Chapter 3 will propose a new structure of LDPC codes by using difference families. In chapter 4, the simulation results for the LDPC codec will be discussed in chapter 2 and chapter 3 will be shown. Chapter 5 will discuss the VLSI implementation of the LDPC decoder. In the end of this thesis, brief conclusions will be presented in chapter 6.

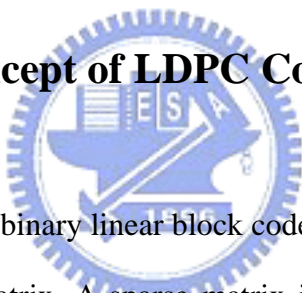


Chapter 2

Low-Density Parity-Check Code

In this chapter, an introduction to low-density parity-check code will be given, including the fundamental concepts of LDPC code, code construction, encoding and decoding mechanism.

2.1 Fundamental Concept of LDPC Code



A binary LDPC code is a binary linear block code that can be defined by a sparse binary $m \times n$ parity-check matrix. A sparse matrix is a matrix where only a small fraction of its entries are ones. Non-binary LDPC codes over $GF(q)$ are discussed in [6]. Hereafter, binary LDPC codes will be called LDPC codes for short.

For any $m \times n$ parity-check matrix H , it defines a (n, j, k) -regular LDPC code if every column vector of H has the same weight j and every row vector of H has the same weight k . Here the weight of a vector is the number of ones in the vector. By counting the ones in H , it follows that $n \times j = m \times k$. Hence if $m < n$, then $j < k$. Suppose the parity-check matrix has full rank, the code rate of H is $r = (n - m)/n = (k - j)/k = 1 - j/k$. If not all the columns or all the rows of the parity-check matrix H have the same number of ones, an LDPC code is said to be irregular.

As suggested by Tanner [7], an LDPC code can be represented as a bipartite graph. An LDPC code corresponds to a unique bipartite graph and a bipartite graph also corresponds to a unique LDPC code. In a bipartite graph, one type of nodes, called the variable nodes, correspond to the symbols in a codeword. The other type of nodes, called the check nodes, correspond to the set of parity check equations. If the parity-check matrix H were an $m \times n$ matrix, it would have m check nodes and n variable nodes. A variable node v_i is connected to a check node c_j by an edge, denoted as (v_i, c_j) , if and only if the entry $h_{i,j}$ of H is one. A cycle in a graph of nodes and edges is defined as a sequence of connected edges which starts from a node and ends at the same node, and satisfies the condition that no node (except the initial and final node) appears more than once. The number of edges on a cycle is called the length of the cycle. The length of the shortest cycle in a graph is called the girth of the graph.

Regular LDPC codes are those where all nodes of the same type have the same degree. The degree of a node is the number of edges connected to that node. For example, Figure 2.1 shows a (8, 4, 2)-regular LDPC code and its corresponding

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

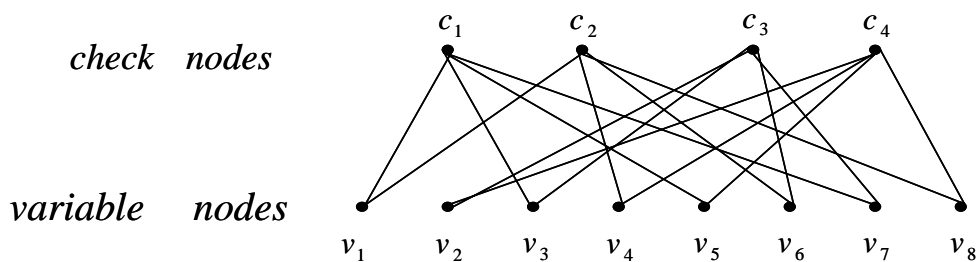


Figure 2.1 Example of a (8, 4, 2)-regular LDPC code and its corresponding Tanner graph. There are 8 variable nodes (v_i) and 4 check nodes (c_i).

Tanner graph. In this example, all the variable nodes have a degree of 2 and all the check nodes have a degree of 4. The edges (c_1, v_3) , (v_3, c_3) , (c_3, v_7) , and (v_7, c_1) depict a cycle in the Tanner graph. Since this turns out to be the shortest cycle, the girth of this graph is 4. Irregular LDPC codes were introduced in [8] and [9]. For such codes, the degrees of each set of nodes are chosen according to some distribution.

A polynomial $\gamma(x)$ of the form

$$\gamma(x) = \sum_{i \geq 2} \gamma_i x^{i-1} \quad (2.1)$$

is a degree distribution if $\gamma(x)$ has nonnegative coefficients and $\gamma(1) = 1$. Given a degree distribution pair (λ, ρ) to form a sequence of code ensembles $C^n(\lambda, \rho)$, where n is the length of the code and where

$$\begin{aligned} \lambda(x) &= \sum_{i=2}^{d_v} \lambda_i x^{i-1} \\ \rho(x) &= \sum_{i=2}^{d_c} \rho_i x^{i-1} \end{aligned} \quad (2.2)$$

specify the variable and check node degree distributions. More precisely, λ_i and ρ_i represent the fraction of edges emanating from variable and check nodes of degree i respectively; d_v and d_c are denoted as the maximum variable and check node degree. Assume that the code has n variable nodes. The number of variable nodes of degree i is then

$$n \frac{\lambda_i / i}{\sum_{j \geq 2} \lambda_j / j} = n \frac{\lambda_i / i}{\int_0^1 \lambda(x) dx} \quad (2.3)$$

$$\left(\int_0^1 \lambda(x) dx = \int_0^1 \sum_{i=2}^{d_v} \lambda_i x^{i-1} dx = \sum_{i=2}^{d_v} \lambda_i \frac{x^i}{i} \Big|_0^1 = \sum_{i=2}^{d_v} \frac{\lambda_i}{i} \right)$$

and so the total number of edges emanating from all variable nodes E is equal to

$$E = \sum_{i \geq 2} \left(n \frac{\lambda_i / i}{\int_0^1 \lambda(x) dx} \cdot i \right) = \frac{n}{\int_0^1 \lambda(x) dx} \quad (2.4)$$

Similarly, assuming that the code has m check nodes, E can also be expressed as

$$E = \frac{m}{\int_0^1 \rho(x) dx} \quad (2.5)$$

Since the number of edges emanating from all variable nodes is equal to that emanating from all check nodes, we have

$$E = \frac{n}{\int_0^1 \lambda(x) dx} = \frac{m}{\int_0^1 \rho(x) dx} \quad (2.6)$$

Hence

$$m = n \frac{\int_0^1 \rho(x) dx}{\int_0^1 \lambda(x) dx} \quad (2.7)$$

Assuming that H has full rank, the rate of LDPC codes in the ensemble is

$$r(\lambda, \rho) = \frac{n - m}{n} = 1 - \frac{\int_0^1 \lambda(x) dx}{\int_0^1 \rho(x) dx} \quad (2.8)$$

Further more, the average degree \bar{j} of a variable node and average degree \bar{k} of a check node are

$$\bar{j} = \frac{E}{n} = \frac{1}{\sum_{j=2}^{d_v} \lambda_j / j} = \frac{1}{\int_0^1 \lambda(x) dx} \quad (2.9)$$

$$\bar{k} = \frac{E}{m} = \frac{1}{\sum_{j=2}^{d_c} \rho_j / j} = \frac{1}{\int_0^1 \rho(x) dx}$$

2.2 Code Construction

Gallager's method [1]

Define an (n, j, k) parity-check matrix as a matrix of n columns that has j ones in each column, k ones in each row, and zeros elsewhere. It follows from this definition that an (n, j, k) parity-check matrix has nj/k rows and thus a rate $r \geq 1 - j/k$. In order to construct an ensemble of (n, j, k) matrices, consider first the special (n, j, k) matrix in Figure 2.2, for which n, j and k will be 20, 3 and 4, respectively.

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
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
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	1	0	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	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Figure 2.2 Example of a low-density parity-check code matrix where $(n, j, k) = (20, 3, 4)$

This matrix is divided into j sub-matrices, each containing a single 1 in each column. The first of these sub-matrices contains all its 1's in descending order which is, the i^{th} row contains 1's in columns $(i-1)k+1$ to ik . The other sub-matrices are

merely column permutations of the first. We define the ensemble of (n, j, k) codes as the ensemble resulting from random permutations of the columns of each of the bottom $(j-1)$ sub-matrices of a matrix such as in Figure 2.2 with equal probability assigned to each permutation. This definition is somewhat arbitrary and is made for mathematical convenience. In fact such an ensemble does not include all (n, j, k) codes as just defined. Also, at least $(j-1)$ rows in each matrix of the ensemble are linearly dependent. This simply means that the codes have a slightly higher information rate than the matrix indicates.

MacKay's method [10]

A computer-generated code was introduced by MacKay [10]. The parity-check matrix is randomly generated. First, the parameters n , m , j , and k are chosen to conform an (n, m, j, k) -regular LDPC code where n , j and k are the same as in Gallager's code and m is the number of the parity-check equations in H . Then, 1's are randomly generated into j different positions of the first column. The second column is generated in the same way, but checks are made to insure that no two columns have a 1 in the same position more than twice. This constraint is to avoid a 4-cycle to appear in the Tanner graph, which will cause the performance to drop by about 0.5dB. An avoidance of 4-cycles in a parity-check matrix is therefore required. The next few columns are generated sequentially and checks for 4-cycles must be performed on each generation. In this procedure, the number of 1's in each row must be recorded, and if any row already has k 1's, the next column generating will not select that row.

Construction by Quasi-Cyclic Code [5]

A code is quasi-cyclic if, for any cyclic shift of a codeword by l places, the resulting word is also a codeword. A cyclic code is a quasi-cyclic code with $l = 1$. Consider the binary quasi-cyclic codes described by a parity-check matrix

$$H = [A_1, A_2, \dots, A_l] \quad (2.10)$$

where A_1, A_2, \dots, A_l are binary $v \times v$ circulant matrices. The algebra of $(v \times v)$ binary circulant matrices is isomorphic to the algebra of polynomials modulo $x^v - 1$ over GF(2). A circulant matrix A is completely characterized by the polynomial

$$a(x) = a_0 + a_1x + a_2x^2 + \dots + a_{v-1}x^{v-1} \quad (2.11)$$

where the coefficients are from the first row of A , and a code C with parity-check matrix of the form (2.10) can be completely characterized by the polynomials $a_1(x), a_2(x), \dots, a_l(x)$. Figure 2.3(a) shows an example of a rate-1/2 quasi-cyclic code where $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$. Figure 2.3(b) shows the corresponding Tanner graph representation. For this example we can see the edges $(c_1, v_6), (v_6, c_4), (c_4, v_8), (v_8, c_1)$ depict a 4-cycle in this graph which is to be avoided for performance consideration. The method for avoiding 4-cycle condition will be discussed in the next chapter.

$$H = \left[\begin{array}{cccc|cccc} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

(a) A parity-check matrix with two circulant matrices

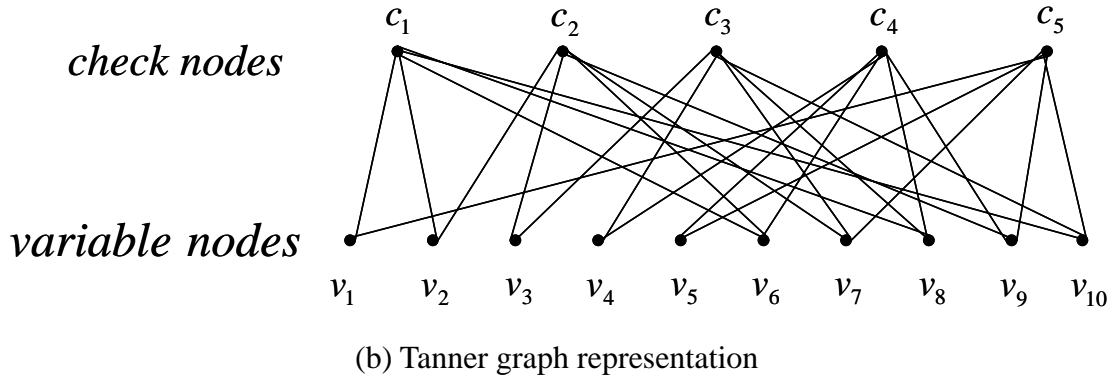


Figure 2.3 Example of a rate-1/2 quasi-cyclic code from two circulant matrices, where $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$

2.3 Encoding

Since LDPC code is a linear block code, it can be encoded by the conventional method. However, using conventional method will introduce an encoding complexity proportional to the quadratic of the code length. The high encoding cost of LDPC codes becomes a major drawback when compared to the turbo codes which has a linear encoding complexity with time. In this section, we will introduce some improved methods.

Conventional method

Let $u = [u_0, u_1, u_2, \dots, u_{k-1}]$ be a row vector of message bits with length k and $c = [c_0, c_1, c_2, \dots, c_{n-1}]$ be a codeword with length n . Let G with dimension $k \times n$ be the generating matrix of this code. It can be derived that

$$c = uG. \quad (2.12)$$

If H is the parity-check matrix of this code with dimension $r \times n$ where $r = n - k$.

Then

$$\begin{aligned}
Hc^T = 0^T &\Rightarrow cH^T = 0 \\
&\Rightarrow uGH^T = 0 \\
&\Rightarrow GH^T = 0
\end{aligned} \tag{2.13}$$

Suppose a sparse parity-check matrix H with full rank is constructed. Gaussian elimination and column reordering can be used to derive an equivalent parity-check matrix in the systematic form $H_{systematic} = [P|I_r]$. Thus equation (2.13) can be solved to get the generating matrix in systematic form as

$$G_{systematic} = [I_k|P^T]. \tag{2.14}$$

Finally, the generating matrix G can be obtained by doing the reverse column reordering to the $G_{systematic}$.

Forcing H to have lower triangular form [4]



In [4] it was suggested to force the parity-check matrix to be in the lower triangular form. Under this restriction, it guarantees a linear time encoding complexity, but, in general, it also results in some loss of performance.

Richardson's method [3]

Figure 2.4 shows how to bring the parity-check matrix into an approximate lower triangular form using row and column permutations. Note that since this

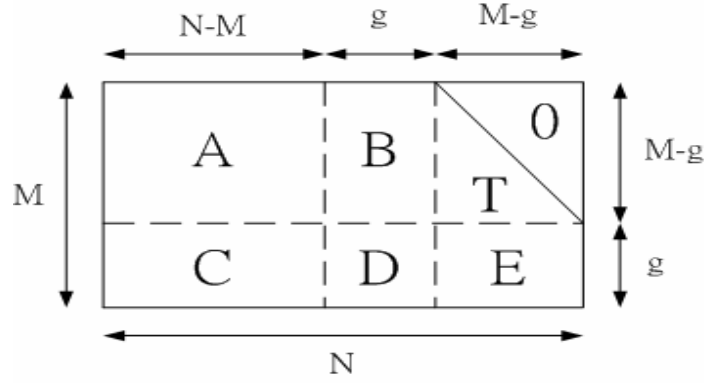


Figure 2.4 The parity-check matrix in an approximate lower triangular form transformation was accomplished solely by permutations, the matrix is still sparse.

More precisely, assume that the matrix is written in the form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} \quad (2.15)$$

where A is $(m-g) \times (n-m)$, B is $(m-g) \times g$, T is $(m-g) \times (m-g)$, C is $g \times (n-m)$, D is $g \times g$, and E is $g \times (m-g)$. Further, all these matrices are sparse and T is lower triangular with ones along the diagonal. Multiplying this matrix from the left by

$$\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix} \quad (2.16)$$

can result in

$$\begin{pmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{pmatrix}. \quad (2.17)$$

Let $x = (s, p_1, p_2)$ denote the codeword of this parity-check matrix where s is the message bits with length $(m-n)$, p_1 and p_2 combined are the parity bits, p_1 has length g , and p_2 has length $(m-g)$. The constrained equation $Hx^T = 0^T$ splits naturally into two equations, namely

$$As^T + Bp_1^T + Tp_2^T = 0 \quad (2.18)$$

and

$$(-ET^{-1}A + C)s^T + (-ET^{-1}B + D)p_1^T = 0. \quad (2.19)$$

Define $\gamma = -ET^{-1}B + D$ and assume for the moment that γ is nonsingular. Then

from equation (2.19) we conclude that

$$p_1^T = -\gamma^{-1}(-ET^{-1}A + C)s^T. \quad (2.20)$$

Hence, once the $g \times (n - m)$ matrix $-\gamma^{-1}(-ET^{-1}A + C)s^T$ has been pre-computed, the determination of p_1 can be accomplished with a time complexity of $O(g \times (n - m))$ simply by performing a multiplication with this (generically dense) matrix. This complexity can be further reduced as shown in Table 2.1. Rather than pre-computing $-\gamma^{-1}(-ET^{-1}A + C)s^T$ and then multiplying with s^T , p_1 can be determined by breaking the computation into several smaller steps, each of which is computationally efficient. To this end, we first determine As^T , which has complexity of $O(n)$, since A is sparse. Next, we multiply the result by T^{-1} . Since $T^{-1}[As^T] = y^T$ is equivalent to the system $[As^T] = Ty^T$, this can also be accomplished in $O(n)$ time with by back-substitution, because T is lower triangular and sparse. The remaining steps are fairly straightforward. It follows that the overall complexity of determining p_1 is $O(n + g^2)$. In a similar manner, noting from equation (2.18) that $p_2^T = -T^{-1}(As^T + Bp_1^T)$, we can accomplish the determination of p_2 in time complexity of $O(n)$ as shown step by step in Table 2.2.

A summary of this efficient encoding procedure is given in Table 2.3. It entails two steps, the preprocessing step and the actual encoding step. In the preprocessing step, we first perform row and column permutations to bring the parity-check matrix into the approximate lower triangular form with as small a gap g as possible. In actual encoding then entails the steps listed in Table 2.1 and 2.2. The overall encoding complexity is $O(n + g^2)$, where g is the gap of the approximate triangulation.

Table 2.1 Efficient computation step of $p_1^T = -\gamma^{-1}(-ET^{-1}A + C)s^T$

Operation	Comment	Complexity
As^T	Multiplication by sparse matrix	$O(n)$
$T^{-1}[As^T]$	$T^{-1}[As^T] = y^T \Leftrightarrow [As^T] = Ty^T$	$O(n)$
$-E[T^{-1}As^T]$	Multiplication by sparse matrix	$O(n)$
Cs^T	Multiplication by sparse matrix	$O(n)$
$[-ET^{-1}As^T] + [Cs^T]$	Addition	$O(n)$
$-\gamma^{-1}[-ET^{-1}As^T + Cs^T]$	Multiplication by dense $g \times g$ matrix	$O(g^2)$

Table 2.2 Efficient computation step of $p_2^T = -T^{-1}(As^T + Bp_1^T)$

Operation	Comment	Complexity
As^T	Multiplication by sparse matrix	$O(n)$
Bp_1^T	Multiplication by sparse matrix	$O(n)$
$[As^T] + [Bp_1^T]$	Addition	$O(n)$
$-T^{-1}[As^T + Bp_1^T]$	$-T^{-1}[As^T + Bp_1^T] = y^T \Leftrightarrow -[As^T + Bp_1^T] = Ty^T$	$O(n)$

Table 2.3 Summary of Richardson's encoding procedure It entails two steps: A

processing step and the actual encoding step

Preprocessing: Input: Non-singular parity-check matrix H . Output: An equivalent parity-check matrix of the form $\begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$ such that $-ET^{-1}B + D$ is non-singular.

1. [Triangulation] Perform row and column permutations to bring the parity-check matrix H into the approximate lower triangular form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$$

with as small a gap g as possible.

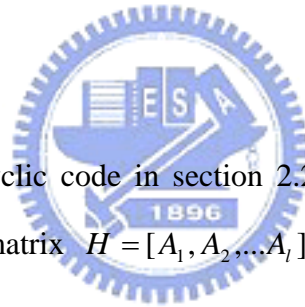
2. [Check] Check that $-ET^{-1}B + D$ is non-singular, performing further column permutations if necessary to ensure this property.

$$\begin{pmatrix} I & 0 \\ ET^{-1} & I \end{pmatrix} \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} = \begin{pmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{pmatrix}$$

Encoding: Input: Parity-check matrix of the form $\begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$ such that $-ET^{-1}B + D$ is non-singular and a vector s denote the message bits has length $(m - n)$. Output: The vector $x = (s, p_1, p_2)$ where p_1 has length g and p_2 has length $(m - g)$, such that $Hx^T = 0^T$.

1. Determine p_1 as shown in Table 2.1.
2. Determine p_2 as shown in Table 2.2.

Quasi-cyclic code [5]



As a review of quasi-cyclic code in section 2.2, the quasi-cyclic code can be described by a parity-check matrix $H = [A_1, A_2, \dots, A_l]$ and each of a circulant matrix A_j is completely characterized by the polynomial $a(x) = a_0 + a_1x + \dots + a_{v-1}x^{v-1}$ with coefficients from its first row. A code C with parity-check matrix H can be completely characterized by the polynomials $a_1(x), a_2(x), \dots, a_l(x)$. As for the encoding, if one of the circulant matrices is invertible (say A_l) the generator matrix for the code can be constructed in the following systematic form

$$G = \begin{bmatrix} & (A_l^{-1}A_1)^T \\ & (A_l^{-1}A_2)^T \\ & \dots \\ I_{v(l-1)} & (A_l^{-1}A_{l-1})^T \end{bmatrix} \quad (2.21)$$

resulting in a quasi-cyclic code of length vl and dimension $v(l-1)$. Encoding can be achieved with linear complexity using a $v(l-1)$ -stage shift register. Regarding the

algebraic computation, the polynomial transpose is defined as

$$a(x)^T = \sum_{i=0}^{n-1} a_i x^{n-i}, \quad x^n = 1. \quad (2.22)$$

For a binary $[n, k]$ code, length $n = vl$ and dimension $k = v(l-1)$, the k -bit message $[i_0, i_1, \dots, i_{k-1}]$ is described by the polynomial $i(x) = i_0 + i_1x + \dots + i_{k-1}x^{k-1}$ and the codeword for this message is $c(x) = [i(x), x^k p(x)]$, where $p(x)$ is given by

$$p(x) = \sum_{j=1}^{l-1} i_j(x) * (a_l^{-1}(x) * a_j(x))^T, \quad (2.23)$$

$i_j(x)$ is the polynomial representation of the information bits $i_{v(j-1)}$ to i_{vj-1} , where

$$i_j(x) = i_{v(j-1)} + i_{v(j-1)+1}x + \dots + i_{vj-1}x^{v-1} \quad (2.24)$$

and polynomial multiplication $(*)$ is mod $x^v - 1$.

As an example, consider a rate-1/2 quasi-cyclic code with $v=5$, $l=2$, first circulant is described by $a_1(x) = 1+x$ and the second circulant is described by $a_2(x) = 1+x^2+x^4$, which is invertible and

$$a_2^{-1}(x) = x + x^2 + x^4. \quad (2.25)$$

The generator matrix contains a 5×5 identity matrix and the 5×5 matrix described by the polynomial

$$(a_2^{-1}(x) * a_1(x))^T = (1+x^2)^T = 1+x^3. \quad (2.26)$$

Figure 2.5 shows the example parity-check matrix and the corresponding generator matrix.

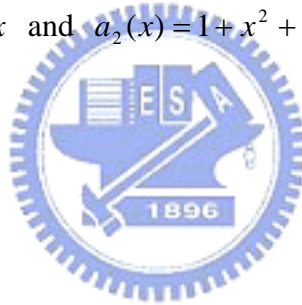
$$H = \left[\begin{array}{ccccc|ccccc} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

(a) A parity-check matrix with two circulants

$$G = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array} \right]$$

(b) The corresponding generator matrix in systematic form

Figure 2.5 Example of a rate-1/2 quasi-cyclic code. (a) Parity-check matrix with two circulants, where $a_1(x) = 1 + x$ and $a_2(x) = 1 + x^2 + x^4$. (b) Corresponding generator matrix in systematic form.



2.4 Decoding [11]

There are several decoding algorithm for LDPC codes. All of them are iterative decoding. Messages between variable nodes and check nodes are exchanged back and forth. The decoder expects that error will be corrected progressively by using this iterative message-passing algorithm. At present, there are three types of iterative decoding algorithms applied to LDPC codes in general.

- Sum-product algorithms, also known as message passing algorithm.
- Min-sum based algorithms.
- Min-sum algorithms.

2.4.1 Decoding Procedure in One Iteration

Now we make a description of the message passing algorithm in one iteration. Here is a simple example of irregular LDPC code. The parity check matrix is shown below.

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} S_1 \\ S_2 \end{matrix}$$

$x_1 \quad x_2 \quad x_3 \quad x_4$

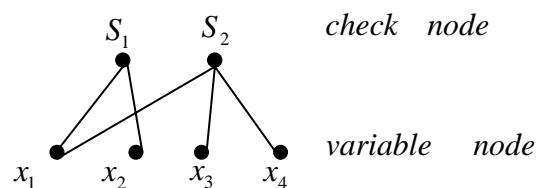
If the received codeword sequence is \bar{x} , then we can use $H\bar{x}^T = 0^T$ to try whether the received codeword sequence is a codeword, i.e.,

$$Hx^T = 0^T \Rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (2.27)$$

$$\Rightarrow \begin{matrix} \text{Equation } S_1 : x_1 \oplus x_2 = 0 \\ \text{Equation } S_2 : x_1 \oplus x_3 \oplus x_4 = 0 \end{matrix}$$

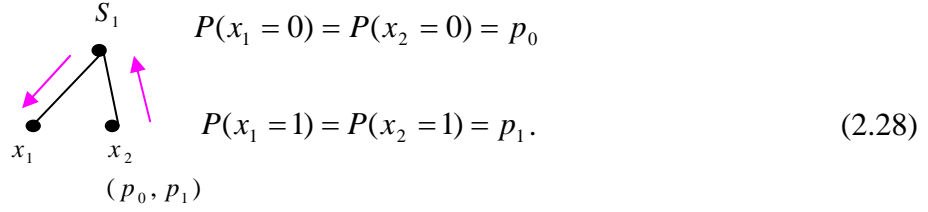
where “ \oplus ” denotes the modulo-2 addition.

The message passing algorithm uses Tanner graph for decoding procedure, which is shown below.



For x_1 estimation:

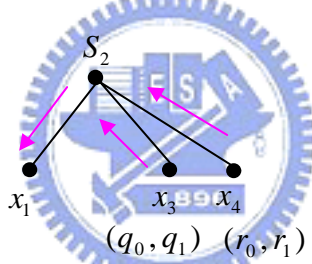
Step1: Suppose p_0 and p_1 are the priori-probability of x_2 , where $p_0 + p_1 = 1$, we can use Equation S_1 ($x_1 \oplus x_2 = 0$) to estimate the post-probability of x_1 as follows:



In the same way, suppose q_0 and q_1 are the priori-probability of x_3 , where $q_0 + q_1 = 1$ and r_0 and r_1 are the priori-probability of x_4 where $r_0 + r_1 = 1$, we can use Equation S_2 that $(x_1 \oplus x_3 \oplus x_4 = 0)$ to estimate the post-probability of x_1 , using the following equation:

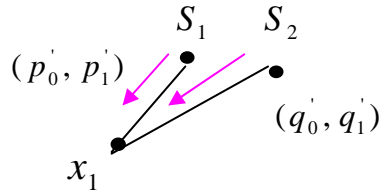
$$\begin{aligned}
 P(x_1 = 0) &= P(x_3 \oplus x_4 = 0) \\
 &= P(x_3 = 0)P(x_4 = 0) + P(x_3 = 1)P(x_4 = 1) = q_0r_0 + q_1r_1
 \end{aligned}
 \tag{2.29}$$

$$\begin{aligned}
 P(x_1 = 1) &= P(x_3 \oplus x_4 = 1) \\
 &= P(x_3 = 1)P(x_4 = 0) + P(x_3 = 0)P(x_4 = 1) = q_1r_0 + q_0r_1
 \end{aligned}$$



Step2: Based on Equation S_1 and Equation S_2 , we can estimate the final post-probability of x_1 , by using:

$$\begin{aligned}
 P(x_1 = 0) &\propto P(S_1 = 0 \text{ and } x_1 = 0)P(S_2 = 0 \text{ and } x_1 = 0) = p'_0q'_0 \\
 P(x_1 = 1) &\propto P(S_1 = 0 \text{ and } x_1 = 1)P(S_2 = 0 \text{ and } x_1 = 1) = p'_1q'_1
 \end{aligned}
 \tag{2.30}$$



where $p'_0 = p_0$, $p'_1 = p_1$, $q'_0 = q_0r_0 + q_1r_1$ and $q'_1 = q_1r_0 + q_0r_1$. It can be summed up that if a check node S_i is connected by three variable nodes x_i , x_j and x_k , and if the priori-probability of the variable nodes x_i and x_j are (q_0, q_1) and

(r_0, r_1) , respectively, we can use the check Equation S_i to estimate the post-probability of x_k in step1 which is

$$CHK(q_0, q_1, r_0, r_1) = (q_0 r_0 + q_1 r_1, q_1 r_0 + q_0 r_1). \quad (2.31)$$

Similarly, if a variable node x_i is connected by two check nodes that are S_i and S_j , and if the message of the S_i and S_j are collected from step1 are (p_0', p_1') and

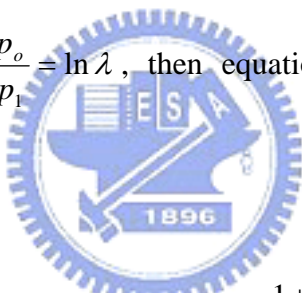
(q_0', q_1') , respectively, we can estimate the final post-probability of x_i as

$$VAR(p_0', p_1', q_0', q_1') = \left(\frac{p_0' q_0'}{p_0' q_0' + p_1' q_1'}, \frac{p_1' q_1'}{p_0' q_0' + p_1' q_1'} \right). \quad (2.32)$$

Since the summation of the priori-probability on any variable node x_k is one, in other words $p_0 + p_1 = 1$, we can transform the priori-probability to a single-value

function. Let $L(p_0, p_1) = \ln \frac{p_0}{p_1} = \ln \lambda$, then equations (2.31) and (2.32) can be

rewritten as



$$\begin{aligned} CHK(L_1, L_2) &= CHK(L_1 \oplus L_2) = \ln \frac{1 + \lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \\ &= \ln \frac{1 + e^{L_1} e^{L_2}}{e^{L_1} + e^{L_2}} = \ln \frac{e^{\frac{L_1+L_2}{2}} + e^{\frac{L_1+L_2}{2}}}{e^{\frac{L_1-L_2}{2}} + e^{\frac{L_1-L_2}{2}}} \\ &= \ln(\cosh(\frac{L_1 + L_2}{2})) - \ln(\cosh(\frac{L_1 - L_2}{2})) \\ &= 2 \tanh^{-1}(\tanh(\frac{L_1}{2}) \times \tanh(\frac{L_2}{2})) \end{aligned} \quad (2.33)$$

$$VAR(L_1, L_2) = \ln(\lambda_1 \lambda_2) = \ln \lambda_1 + \ln \lambda_2 = L_1 + L_2. \quad (2.34)$$

Equations (2.33) and (2.34) are computation in Log-Likelihood Ratio (LLR) form.

This transform can reduce the number of parameters, and equation (2.34)

$VAR(L_1, L_2)$ only needs an addition operation rather than multiplication.

Furthermore, equation (2.33) can be further reformulated to different manners.

There are

$$\begin{aligned} CHK(L_1 \oplus L_2) &= 2 \tanh^{-1}(\tanh(\frac{L_1}{2}) \times \tanh(\frac{L_2}{2})) \\ &= \text{sign}(L_1) \text{sign}(L_2) \phi(\phi(|L_1|) + \phi(|L_2|)) \end{aligned} \quad (2.35)$$

where

$$\phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right) = \ln\left(\frac{e^x + 1}{e^x - 1}\right) \quad \text{and} \quad \phi(\phi(x)) = x, \quad (2.36)$$

and

$$\begin{aligned} CHK(L_1 \oplus L_2) &= \ln(\cosh(\frac{L_1 + L_2}{2})) - \ln(\cosh(\frac{L_1 - L_2}{2})) \\ &= \left|\frac{L_1 + L_2}{2}\right| - \left|\frac{L_1 - L_2}{2}\right| + \ln \frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}} \\ &= \text{sign}(L_1) \times \text{sign}(L_2) \times \min(|L_1|, |L_2|) + \ln \frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}} \end{aligned} \quad (2.37)$$

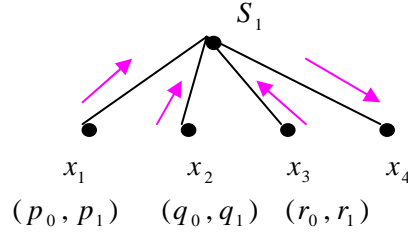
$$\approx \text{sign}(L_1) \times \text{sign}(L_2) \times \min(|L_1|, |L_2|). \quad (2.38)$$

When the check node computation is in the form of equation (2.35), we call it the sum-product algorithm. Similarly, when the check node computation is in the form of equation (2.37), we call it the min-sum based algorithm, and the fourth term $\ln \frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}}$ in equation (2.37) is called the correction factor. Last of all, when the check node computation is the form of equation (2.38), or in other words an approximate form, we call it the min-sum algorithm.

The above discussion of check node computation is only about a check node connected by two or three variable nodes. Now, we will discuss the case when the number of variable nodes are more than three, and then discuss the general form.

Consider a check node S_1 connected by four variable nodes x_1, x_2, x_3 and x_4 . The priori-probability of variable nodes x_1, x_2 and x_3 are (p_0, p_1) , (q_0, q_1) and (r_0, r_1) . We can use the check Equation S_1 , that is, $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ to estimate the post-probability of x_4 , namely,

$$\begin{aligned}
P(x_4 = 0) &= P(x_1 \oplus x_2 \oplus x_3 = 0) = P(x_1 = 0)P(x_2 = 0)P(x_3 = 0) + \\
&\quad P(x_1 = 1)P(x_2 = 1)P(x_3 = 0) + P(x_1 = 0)P(x_2 = 1)P(x_3 = 1) + \\
&\quad P(x_1 = 1)P(x_2 = 0)P(x_3 = 1) = p_0q_0r_0 + p_1q_1r_0 + p_0q_1r_1 + p_1q_0r_1 \\
P(x_4 = 1) &= P(x_1 \oplus x_2 \oplus x_3 = 1) = P(x_1 = 1)P(x_2 = 1)P(x_3 = 1) + \\
&\quad P(x_1 = 1)P(x_2 = 0)P(x_3 = 0) + P(x_1 = 0)P(x_2 = 1)P(x_3 = 0) + \\
&\quad P(x_1 = 0)P(x_2 = 0)P(x_3 = 1) = p_1q_1r_1 + p_1q_0r_0 + p_0q_1r_0 + p_0q_0r_1
\end{aligned} \tag{2.39}$$



Then, one can transform equation (2.39) to a LLR form, and obtain

$$\begin{aligned}
CHK(L_1 \oplus L_2 \oplus L_3) &= \ln \frac{p_0q_0r_0 + p_1q_1r_0 + p_0q_1r_1 + p_1q_0r_1}{p_1q_1r_1 + p_1q_0r_0 + p_0q_1r_0 + p_0q_0r_1} = \ln \frac{\lambda_1\lambda_2\lambda_3 + \lambda_1 + \lambda_2 + \lambda_3}{1 + \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1} \\
&= \ln \frac{e^{L_1}e^{L_2}e^{L_3} + e^{L_1} + e^{L_2} + e^{L_3}}{1 + e^{L_1}e^{L_2} + e^{L_2}e^{L_3} + e^{L_3}e^{L_1}} = \ln \frac{1 + \left[\frac{1 + e^{L_1}e^{L_2}}{e^{L_1} + e^{L_2}} \right] + e^{L_3}}{1 + e^{L_1}e^{L_2} + e^{L_3}} \\
&= \ln \frac{1 + e^x e^{L_3}}{e^x + e^{L_3}} = \ln \frac{e^{\frac{x+L_3}{2}} + e^{\frac{x+L_3}{2}}}{e^{\frac{x-L_3}{2}} + e^{\frac{x-L_3}{2}}} \\
&= \ln \left(\cosh \left(\frac{x+L_3}{2} \right) \right) - \ln \left(\cosh \left(\frac{x-L_3}{2} \right) \right) \\
&= 2 \tanh^{-1} \left(\tanh \left(\frac{x}{2} \right) \tanh \left(\frac{L_3}{2} \right) \right) \tag{2.40}
\end{aligned}$$

where $x = \ln \frac{1 + e^{L_1}e^{L_2}}{e^{L_1} + e^{L_2}} \Rightarrow e^x = \frac{1 + e^{L_1}e^{L_2}}{e^{L_1} + e^{L_2}}$. From equation (2.33), it can be seen that

$x = CHK(L_1 \oplus L_2)$. Equation (2.40) can be computed in a recursive manner such that

$CHK(L_1 \oplus L_2 \oplus L_3) = CHK(CHK(L_1 \oplus L_2) \oplus L_3)$. The general form for check node

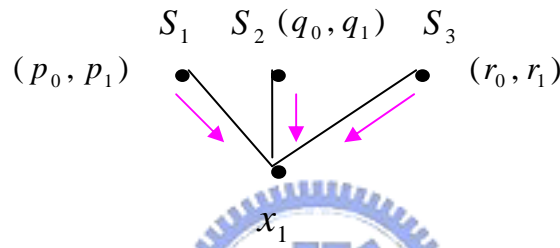
computation can be derived as

$$CHK(L_1 \oplus L_2 \oplus \dots \oplus L_l) = CHK(CHK(\dots CHK(CHK(L_1 \oplus L_2) \oplus L_3) \dots) \oplus L_l). \quad (2.41)$$

Similarly, consider that a variable node x_1 connected by three check nodes S_1, S_2 and S_3 , and the message collected by S_1, S_2 and S_3 are $(p_0, p_1), (q_0, q_1)$ and (r_0, r_1) , respectively. The final post-probabilities of the variable node x_1 are

$$P(x_1 = 0) = P(S_1 = 0 \text{ and } x_1 = 0)P(S_2 = 0 \text{ and } x_1 = 0)P(S_3 = 0 \text{ and } x_1 = 0) = p_0 q_0 r_0$$

$$P(x_1 = 1) = P(S_1 = 1 \text{ and } x_1 = 1)P(S_2 = 1 \text{ and } x_1 = 1)P(S_3 = 1 \text{ and } x_1 = 1) = p_1 q_1 r_1. \quad (2.42)$$



Then, one can transform equation (2.38) into a LLR form, and obtain

$$VAR(L_1, L_2, L_3) = \ln(\lambda_1 \lambda_2 \lambda_3) = \ln \lambda_1 + \ln \lambda_2 + \ln \lambda_3 = L_1 + L_2 + L_3. \quad (2.43)$$

So equation (2.43) can also be computed in a recursive manner such that $VAR(L_1, L_2, L_3) = VAR(VAR(L_1, L_2), L_3)$, and the general form to the variable node

computation can be derived as

$$VAR(L_1, L_2, \dots, L_l) = VAR(VAR(\dots(VAR(VAR(L_1, L_2), L_3) \dots), L_l)). \quad (2.44)$$

2.4.2 Iterative Decoding Procedure [12]

The discussion in section 2.4.1 is about the decoding procedure in one iteration. Now, we consider the actual decoding procedure. It means that there will involve many iterations for a decoding process. First, let us describe some notations for the iterative decoding procedure in Figure 2.6. $M(l)$ denotes the set of check nodes that are connected to the variable node l , i.e., positions of “1”s in the l^{th} column of the

parity-check matrix. $L(m)$ denotes the set of variable nodes that participate in the m^{th} parity-check equation, i.e., the positions of “1”s in the m^{th} row of the parity-check matrix. $L(m) \setminus l$ represents the set $L(m)$ excluding the l^{th} variable node and $M(l) \setminus m$ represents the set $M(l)$ excluding the m^{th} check node. $q_{l,m}$ denotes the

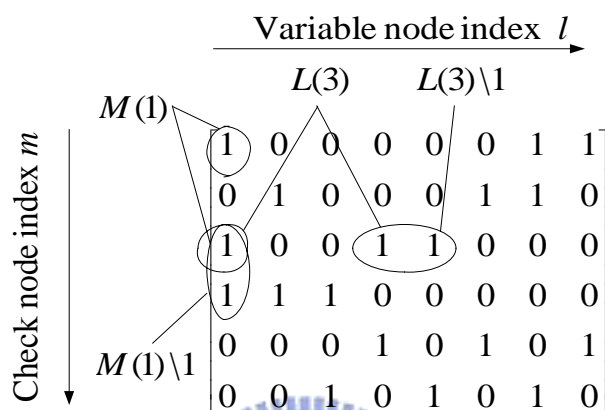


Figure 2.6 Notations for iterative decoding procedure

probability message that variable node l sends to check node m . $r_{m,l}$ denotes the probability message that the m^{th} check node gathers for the l^{th} variable node. The probability message of $q_{l,m}$ and $r_{m,l}$ are computation in LLR domain. The iterative decoding procedure is shown below.

1. Initialization

Let

$$L_l = \ln \frac{P(y_l | x_l = 0)}{P(y_l | x_l = 1)} = \frac{2}{\sigma^2} y_l \quad (2.45)$$

be the log likelihood ratio of a variable node, where $P(a|b)$ specifies that given b is transmitted, the probability that the receiver received a , where σ^2 is the noise variance. For every position (m,l) such that $H_{m,l} = 1$, $q_{m,l}$ is initialized as

$$q_{m,l} = L_l. \quad (2.46)$$

2. Message passing

Step1 (message passing from check nodes to variable nodes): Each check node m gathers all the incoming message $q_{m,l}$'s, and update the message on the variable node l based on the messages from all other variable nodes connected to the check node m .

$$r_{m,l} = \text{CHK} \left(\sum_{l' \in L(m) \setminus l} \oplus q_{m,l'} \right). \quad (2.47)$$

Step2 (message passing from variable nodes to check nodes): Each variable node l passes its probability message to all the check nodes that are connected to it.

$$q_{m,l} = \text{VAR} \left(\text{VAR}_{m' \in M(l) \setminus m} (r_{m',l}), L_l \right) = L_l + \sum_{m' \in M(l) \setminus m} r_{m',l}. \quad (2.48)$$

Step3 (decoding): For each variable node l , messages from all the check nodes that are connected to the variable node l are summed up.

$$q_l = \text{VAR} \left(\text{VAR}_{m \in M(l)} (r_{m,l}), L_l \right) = L_l + \sum_{m \in M(l)} r_{m,l}. \quad (2.49)$$

Hard decision is made on q_l , and the resulting decoded input vector \hat{x} is checked against the parity-check matrix H . If $H\hat{x}^T = 0$, the decoder stops and output \hat{x} . Otherwise it repeats steps 1-3 until it reaches the specified maximum iteration loops.

2.4.3 Efficient Check Node Computation

According to equation (2.41), the check node update computation can be implemented in a serial configuration. Consider a particular check node m with l connections from variable nodes. The incoming messages are then $q_{m,1}, q_{m,2}, \dots, q_{m,l}$.

The goal is to compute the outgoing messages $r_{m,1}, r_{m,2}, \dots, r_{m,l}$. Let us define two sets

of auxiliary binary random variables $f_1 = q_{m,1}$, $f_2 = f_1 \oplus q_{m,2}$,

$f_3 = f_2 \oplus q_{m,3}, \dots, f_l = f_{l-1} \oplus q_{m,l},$ and $b_l = q_{m,l}, b_{l-1} = b_l \oplus q_{m,l-1}, \dots,$
 $b_1 = b_2 \oplus q_{m,1}.$ We can obtain $CHK(f_1), CHK(f_2), \dots, CHK(f_l)$ and $CHK(b_1),$
 $CHK(b_2), \dots, CHK(b_l)$ in a recursive manner based on the knowledge of
 $q_{m,1}, q_{m,2}, \dots, q_{m,l}.$ Using the parity-check node constraint $(q_{m,1} \oplus q_{m,2} \oplus \dots \oplus q_{m,l}) = 0,$
the outgoing message from check node m can be simply expressed as

$$\begin{aligned}
r_{m,i} &= CHK(f_{i-1} \oplus b_{i+1}), \quad i = 2, 3, \dots, l-1, \\
r_{m,1} &= CHK(b_2), \\
r_{m,l} &= CHK(f_{l-1}).
\end{aligned} \tag{2.50}$$

The total computational load consists of the forward recursive computation of $CHK(f_i),$ the backward recursive computation of $CHK(b_i),$ and the final pair-wise part in equation (2.50), which amounts to $3(l-1)$ core operation of the type $CHK(a \oplus b)$ per check node. Clearly, the above procedure is exactly the forward-backward algorithm, as shown in Figure 2.7. The serial nature of computations makes a latency of $O(l)$ units of time in computing a check node update.

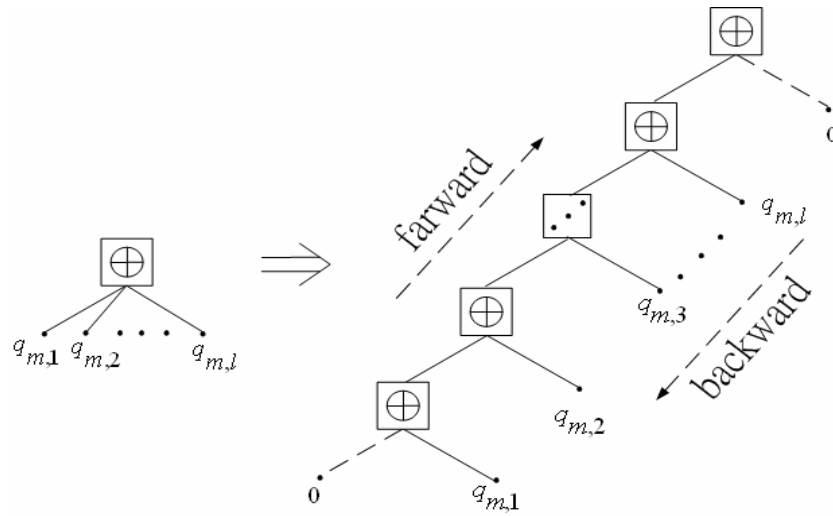


Figure 2.7 Serial configuration for computing check node update

An efficient implementation for computing check node update is introduced by

[13]. A simple parallel configuration that enables fast check node update is described here. First, an auxiliary binary random variable $S_m = \sum_{i=1}^l \oplus q_{m,i}$ is defined. Then, S_m can be computed using the parallel configuration shown in Figure 2.8. The computation at each check node in the parallel configuration is $CHK(a \oplus b)$. The latency in computing the S_m is of order $O(\log l)$, resulting in a speed-up factor of $O[d_c/\log(l)]$ compared to the serial configuration. Having obtained S_m , the outgoing message $r_{m,i}$, $i = 1, 2, \dots, l$, can be computed in an efficient way. Consider

$$\begin{aligned} CHK(S_m) &= CHK\left(\sum_{i=1}^l \oplus q_{m,i}\right) = CHK\left(q_{m,i} \oplus \sum_{j=1, j \neq i}^l \oplus q_{m,j}\right) \\ &= \ln \frac{1 + e^{CHK\left(\sum_{j=1, j \neq i}^l \oplus q_{m,j}\right) + q_{m,i}}}{e^{CHK\left(\sum_{j=1, j \neq i}^l \oplus q_{m,j}\right)} + e^{q_{m,i}}}. \end{aligned} \quad (2.51)$$

Since the term $CHK\left(\sum_{j=1, j \neq i}^l \oplus q_{m,j}\right)$ in equation (2.51) is exactly equivalent to the outgoing message $r_{m,i}$ from check node m to all the variable nodes i , where $i \in (1, 2, \dots, l)$, equation (2.51) becomes

$$CHK(S_m) = \ln \frac{1 + e^{r_{m,i} + q_{m,i}}}{e^{r_{m,i}} + e^{q_{m,i}}}. \quad (2.52)$$

Then, $r_{m,i}$ can be obtained by reformulating equation (2.52) as

$$\begin{aligned} e^{CHK(S_m)} &= \frac{1 + e^{r_{m,i} + q_{m,i}}}{e^{r_{m,i}} + e^{q_{m,i}}} \Rightarrow e^{r_{m,i} + CHK(S_m)} + e^{q_{m,i} + CHK(S_m)} = 1 + e^{r_{m,i} + q_{m,i}} \\ &\Rightarrow e^{r_{m,i}} (e^{q_{m,i}} - e^{CHK(S_m)}) = e^{q_{m,i} + CHK(S_m)} - 1 \\ &\Rightarrow e^{r_{m,i}} = \frac{e^{q_{m,i} + CHK(S_m)} - 1}{e^{q_{m,i}} - e^{CHK(S_m)}} = \frac{e^{q_{m,i} + CHK(S_m)} - 1}{(e^{q_{m,i} - CHK(S_m)} - 1) \times e^{CHK(S_m)}} \\ &\Rightarrow e^{r_{m,i}} = \frac{e^{q_{m,i} + CHK(S_m)} - 1}{e^{q_{m,i} - CHK(S_m)} - 1} \times e^{-CHK(S_m)} \\ &\Rightarrow r_{m,i} = \ln \frac{e^{q_{m,i} + CHK(S_m)} - 1}{e^{q_{m,i} - CHK(S_m)} - 1} - CHK(S_m). \end{aligned} \quad (2.53)$$

Lastly, let's define

$$r_{m,i} = \text{CHK}(S_m \ominus q_{m,i}), \text{ where } i = 1, 2, \dots, l. \quad (2.54)$$

It can be seen that for each $i \in \{1, 2, \dots, l\}$, the message $r_{m,i}$ can be computed simultaneously by a parallel implementation of the new core computation $\text{CHK}(S_m \ominus q_{m,i})$ as shown in Figure 2.8. Clearly, only $l-1$ core computation of type $\text{CHK}(a \oplus b)$ and l core computation of type $\text{CHK}(a \ominus b)$ are necessary for a particular check node update in this parallel configuration.

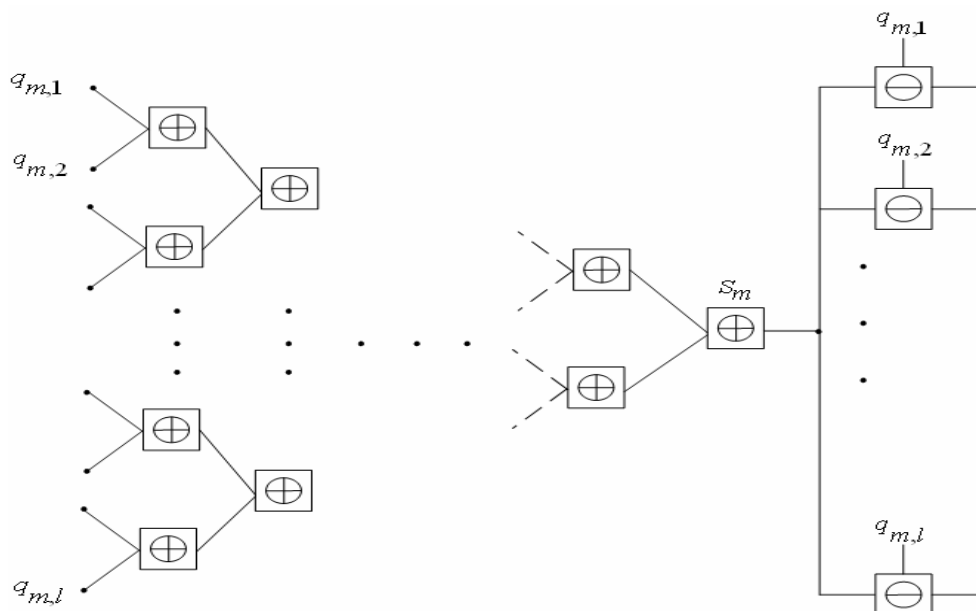


Figure 2.8 Parallel configuration for computing check node update

In the end of this section, we synthesize the contents discussed in sections 2.4.1, 2.4.2 and 2.4.3, and give a summary to the sum-product algorithm, min-sum based algorithm and min-sum algorithm in Table 2.4, Table 2.5 and Table 2.6, respectively.

Table 2.4 Summary of the sum-product algorithm

1. Initialization:

For $1 \leq l \leq n$

$$L_l = \ln \frac{P(y_l | x_l = 0)}{P(y_l | x_l = 1)} = \frac{2}{\sigma^2} y_l, \text{ where } \sigma^2 \text{ is the noise variance}$$

For every l, m such that $H_{m,l} = 1$

$$q_{m,l} = L_l$$

2. Message passing:

Step1: Message passing from check nodes to variable nodes. For each l, m ,

compute

$$\begin{aligned} r_{m,l} &= \underset{l' \in L(m) \setminus l}{\text{CHK}}(q_{m,l'}) = \underset{l' \in L(m) \setminus l}{\text{CHK}}(\sum \oplus q_{m,l'}) \\ &= \text{sign}(q_{m,l}) \prod_{l' \in L(m)} \text{sign}(q_{m,l'}) \times \phi(\phi(\sum_{l' \in L(m)} |q_{m,l'}|) - \phi(|q_{m,l}|)) \end{aligned}$$

where $\phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right) = \ln\left(\frac{e^x + 1}{e^x - 1}\right)$ and $\phi(\phi(x)) = x$.

Step2: Message passing from variable nodes to check nodes. For each l, m ,

compute

$$q_{m,l} = \text{VAR}(\underset{m' \in M(l) \setminus m}{\text{VAR}}(r_{m',l}), L_l) = L_l + \sum_{m' \in M(l) \setminus m} r_{m',l}$$

Step3: Decoding

For each l ,

$$q_l = \text{VAR}(\underset{m \in M(l)}{\text{VAR}}(r_{m,l}), L_l) = L_l + \sum_{m \in M(l)} r_{m,l}$$

For $1 \leq l \leq n$,

$$\hat{x}_l = 0 \text{ if } q_l > 0, \hat{x}_l = 1 \text{ if } q_l < 0$$

If $(H\hat{x}^T = 0)$, then \hat{x} is the estimated codeword ,
or the number of iteration exceeds a predetermined threshold
 \Rightarrow the algorithm stops
else
 \Rightarrow return to step 1

Table 2.5 Summary of the min-sum based algorithm

1. Initialization:

For $1 \leq l \leq n$

$$L_l = \ln \frac{P(y_l | x_l = 0)}{P(y_l | x_l = 1)} = \frac{2}{\sigma^2} y_l, \text{ where } \sigma^2 \text{ is the noise variance}$$

For every l, m such that $H_{m,l} = 1$

$$q_{m,l} = L_l$$

2. Message passing:

Step1: Message passing from check nodes to variable nodes. First, compute

$$\begin{aligned} \text{CHK}(S_m) &= \text{CHK}(\oplus_{l \in L(m)} q_{m,l}) \\ &= \text{CHK}(\dots \text{CHK}(\text{CHK}(q_{m,1} \oplus q_{m,2}) \oplus (\text{CHK}(q_{m,3} \oplus q_{m,4}) \dots)) \\ &= \text{CHK}(\text{CHK}(\dots \text{CHK}(\text{CHK}(q_{m,1} \oplus q_{m,2}) \oplus q_{m,3}) \dots) \oplus q_{m,l'}) \end{aligned}$$

$$\text{where } \text{CHK}(a \oplus b) = \text{sign}(a)\text{sign}(b) \times \text{Min}(|a|, |b|) + \ln \frac{1 + e^{|a+b|}}{1 + e^{|a-b|}}$$

Then, for each l, m , compute

$$r_{m,l} = \ln \frac{e^{q_{m,l} + \text{CHK}(S_m)} - 1}{e^{q_{m,l} - \text{CHK}(S_m)} - 1} - \text{CHK}(S_m)$$

Step2: Message passing from variable nodes to check nodes. For each l, m ,

compute

$$q_{m,l} = \text{VAR}(\text{VAR}_{m' \in M(l) \setminus m}(r_{m',l}), L_l) = L_l + \sum_{m' \in M(l) \setminus m} r_{m',l}$$

Step3: Decoding

For each l ,

$$q_l = \text{VAR}(\text{VAR}(r_{m,l}), L_l) = L_l + \sum_{m \in M(l)} r_{m,l}$$

For $1 \leq l \leq n$,

$$\hat{x}_l = 0 \text{ if } q_l > 0, \hat{x}_l = 1 \text{ if } q_l < 0$$

If $(H\hat{x}^T = 0, \text{ then } \hat{x} \text{ is the estimated codeword,}$

or the number of iteration exceeds a predetermined threshold)

\Rightarrow the algorithm stops

else

\Rightarrow return to step1

Table 2.6 Summary of the min-sum algorithm

1. Initialization:

For $1 \leq l \leq n$

$$L_l = \ln \frac{P(y_l | x_l = 0)}{P(y_l | x_l = 1)} = \frac{2}{\sigma^2} y_l, \text{ where } \sigma^2 \text{ is the noise variance}$$

For every l, m such that $H_{m,l} = 1$

$$q_{m,l} = L_l$$

2. Message passing:

Step1: Message passing from check nodes to variable nodes. First, compute

$$\text{CHK}(S_m) = \text{CHK}(\oplus_{l' \in L(m)} q_{m,l'}) = \prod_{l' \in L(m)} \text{sign}(q_{m,l'}) \times \text{MIN}\{|q_{m,l'}|\}$$

Then, for each l, m , compute

$$r_{m,l} = \ln \frac{e^{q_{m,l} + \text{CHK}(S_m)} - 1}{e^{q_{m,l} - \text{CHK}(S_m)} - 1} - \text{CHK}(S_m)$$

Step2: Message passing from variable nodes to check nodes. For each l, m ,

compute

$$q_{m,l} = \text{VAR}(\text{VAR}(r_{m',l}), L_l) = L_l + \sum_{m' \in M(l) \setminus m} r_{m',l}$$

Step3: Decoding

For each l ,

$$q_l = \text{VAR}(\text{VAR}(r_{m,l}), L_l) = L_l + \sum_{m \in M(l)} r_{m,l}$$

For $1 \leq l \leq n$,

$$\hat{x}_l = 0 \text{ if } q_l > 0, \hat{x}_l = 1 \text{ if } q_l < 0$$

If $(H\hat{x}^T = 0, \text{ then } \hat{x} \text{ is the estimated codeword,}$

or the number of iteration exceeds a predetermined threshold)

\Rightarrow the algorithm stops

else

\Rightarrow return to step1



Chapter 3

A New Structure for Low-Density

Parity-Check Code Using the Difference

Family

In this chapter, we will partition the discussion into two sections. In section 3.1, an introduction to the difference family and the construction of an irregular quasi-cyclic code based on this concept will be discussed. In section 3.2, we will propose a new structure of the low-density parity-check code, and expecting the new structure to bring performance improvement.

3.1 The Difference Family

In [5], a concept using the difference family to construct an irregular quasi-cyclic code with a Tanner graph free of 4-cycle was introduced. A difference family is an arrangement of a group of v elements, such as Z_v , into not necessarily disjoint subsets of equal size which meet certain difference requirements. More precisely:

Definition 1: The t γ -element subsets of the group Z_v , D_1, D_2, \dots, D_t with $D_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,\gamma}\}$ form a (v, γ, λ) difference family if the difference

$(d_{i,x} - d_{i,y}) \bmod v$, $(i = 1, 2, \dots, t; x, y = 1, 2, \dots, \gamma, x \neq y)$ give each nonzero element of Z_v exactly λ times.

For example, the subsets $D_1 = \{1, 2, 5\}$, $D_2 = \{1, 3, 9\}$ of Z_{13} form a $(13, 3, 1)$ difference family with differences

$$\begin{aligned} \text{From } D_1: & 2-1=1, 1-2=12, 5-1=4, \\ & 1-5=9, 5-2=3, 2-5=10 \\ \text{From } D_2: & 3-1=2, 1-3=11, 9-1=8, \\ & 1-9=5, 9-3=6, 3-9=7. \end{aligned}$$

In this work where the difference families with $\lambda=1$ allows the design of codes free of 4-cycles. For an irregular quasi-cyclic code, define the column weight distribution of a length vl rate $l-(1/l)$ code as the vector $W = [w_1, w_2, \dots, w_l]$, where w_j is the column weight of the columns in the j^{th} circulant. Denote that w_{\max} is the maximum column weight of the parity-check matrix H

$$w_{\max} = \max\{w_1, w_2, \dots, w_l\}. \quad (3.1)$$

To construct an irregular quasi-cyclic code with length vl and rate $l-(1/l)$, so that its parity-check matrix $H = [a_1(x), a_2(x), \dots, a_l(x)]$ has a weight distribution $W = [w_1, w_2, \dots, w_l]$, l sets D_1, D_2, \dots, D_l of a $(v, \gamma, 1)$ difference family with $\gamma \geq w_{\max}$, and $a_j(x)$ can be defined using w_j of the elements of D_j as

$$a_j(x) = x^{d_{j,1}} + x^{d_{j,2}} + \dots + x^{d_{j,w_j}}. \quad (3.2)$$

To ensure that the code can be encoded, $x^v - 1$ must be divisible by at least one of the $a_j(x)$.

For a regular code, all of the elements in each set are included in each circulant, while for an irregular code the choice of which elements in the set to use is arbitrary. The row weight, ρ , of the parity-check matrix is constant, and given by

$$\rho = \sum_{i=1}^l w_i . \quad (3.3)$$

To demonstrate that the quasi-cyclic codes are free of 4-cycles we need a well known result of the difference families.

Lemma 3.1 [5]: A pair of elements from Z_v occur together exactly λ times in the set of translates of every set in a (v, γ, λ) difference family.

Lemma 3.2: The codes of construction by using difference families have Tanner graphs free of 4-cycles.

Proof: Follows from the choice of $\lambda = 1$. First consider the regular case. Each column of $H = [a_1(x), a_2(x), \dots, a_l(x)]$ is a translate of one of the sets D_j in the difference family. To show that there can be no 4-cycles in H , we need to show that no two columns of H can have a nonzero entry in the same two rows, which is equivalent to requiring that two elements of Z_v can occur together in at most one of all the translates of the sets in the difference family. Since two elements occur together in exactly λ translates, we need only choose $\lambda = 1$ to avoid 4-cycles. The argument follows naturally in the irregular construction. Since only w_j of the elements in a given set of the difference family will be taken, removing elements from the set of translates will keep it free of 4-cycles.

3.2 The Proposed Structure of LDPC Code

According to section 3.1, we can use difference family to construct an irregular quasi-cyclic code free of 4-cycles. In the following section we will describe the construction we wish to propose for LDPC codes using these difference families. Below is our proposed structure of the parity-check matrix H,

$$H = \begin{bmatrix} A_1 & A_2 & \dots & A_{l-1} & 0 \\ B_1 & B_2 & \dots & B_{l-1} & B_l \end{bmatrix}. \quad (3.4)$$

where $A_1, A_2, \dots, A_{l-1}, B_1, B_2, \dots,$ and B_l are all $v \times v$ circulant matrices. The code length is vl and the code rate is $(1 - \frac{2}{l})$. We can use the difference families to determine the polynomials of each of the circulant matrix $a_i(x)$ and $b_j(x)$, where $i \in \{1, 2, \dots, l-1\}$ and $j \in \{1, 2, \dots, l\}$, just as the quasi-cyclic code. In order to avoid any 4-cycles in the new structure of the parity-check matrix, we provide a new difference family to solve this problem. First, construct two $(v, \gamma, 1)$ difference families Family A and Family B and combine the two families to form a new difference Family C which are needed to add the following two constraints.

Constraint 1: The differences $[(a_{i,x} - a_{i,y}) \bmod v]$ and $[(b_{i,x} - b_{i,y}) \bmod v]$, where $i = 1, 2, \dots, l-1; x, y = 1, 2, \dots, \gamma, x \neq y$, give each element, can not be the same.

Constraint 2: The differences $[(a_{i,x} - a_{j,y}) \bmod v]$ and $[(b_{i,x} - b_{j,y}) \bmod v]$, where $i, j = 1, 2, \dots, l-1, i \neq j; x, y = 1, 2, \dots, \gamma$, give each element, can not be the same.

More precisely, if a parity-check matrix is 4-cycles free, it represents that no two columns of H can have a nonzero entry in the same two rows. Suppose the new circulant matrix is $C_i = [A_i, B_i]^T$ where $i \in \{1, 2, \dots, l\}$. Constraint 1 is added to avoid the case where any two columns of C_i have a nonzero entry in the same two rows.

Constraint 2 is added to avoid the case where a column of $C_i, i \in \{1, 2, \dots, l\}$ and another column of $C_j, j \in \{1, 2, \dots, l\}, i \neq j$ have a nonzero entry in the same rows.

For example, the subsets from the difference Family A are $A_1 = \{3, 7\}$ and $A_2 = \{1, 6\}$, and the subsets from the difference Family B are $B_1 = \{1, 7\}, B_2 = \{2, 3\}$ and $B_3 = \{4, 6\}$ of Z_{13} , which form a new $(13, 2, 1)$ difference family C. The differences from Constraint 1:

From A_1 : $3-7=9$, $7-3=4$ From B_1 : $1-7=7$, $7-1=6$

From A_2 : $1-6=8$, $6-1=5$ From B_2 : $2-3=12$, $3-2=1$.

The differences from Constraint 2:

From A_1 and A_2 : $3-1=2$, $3-6=10$, $7-1=6$, $7-6=1$

From B_1 and B_2 : $1-2=12$, $1-3=11$, $7-2=5$, $7-3=4$.

Regarding the encoding for the new structure, suppose that two of the circulant matrices A_{l-1} and B_l are invertible, we can derive two generator matrices in the following systematic forms

$$G_{1\text{systematic}} = \begin{bmatrix} & (A_{l-1}^{-1}A_1)^T & & \\ I_{v(l-2)} & (A_{l-1}^{-1}A_2)^T & & \\ & \dots & & \\ & & (A_{l-1}^{-1}A_{l-2})^T & \end{bmatrix} = [I_{v(l-2)}G_1] \quad (3.5)$$

and

$$G_{2\text{systematic}} = \begin{bmatrix} & & & & (B_l^{-1}B_1)^T & \\ & & & & (B_l^{-1}B_2)^T & \\ & & & & \dots & \\ & & & & & (B_l^{-1}B_{l-1})^T & \end{bmatrix} = [I_{v(l-1)}G_2]. \quad (3.6)$$

Let $c = [d, p_1, p_2]$ denote the codeword of the proposed parity-check matrix where d is the message bits with length $v(l-2)$, and p_1 and p_2 combined are the parity bits, each having the same length v . The encoding procedure is partitioned into two steps.

Encoding Step1: We can use the generator matrix G_1 to get the parity bits p_1 . That is

$$p_1 = d \times G_1. \quad (3.7)$$

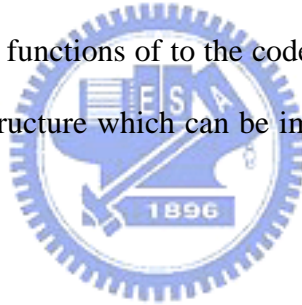
Then, combine the parity bits p_1 with the message bits d to form an intermediate codeword c' where $c' = [d, p_1]$.

Encoding Step2: The last parity bits p_2 can be derived from the generator matrix

G_2 and the intermediate codeword c' . That is

$$p_2 = c' \times G_2. \quad (3.8)$$

In fact, the encoding procedure for the proposed structure is very similar to the quasi-cyclic code discussed in section 2.3. The parity bits p_1 can be generated with linear complexity by using a shift register of size $v(l-2)$ while encoding of the random codes is via matrix multiplication. For example, encoding of the Encoding Step1 requires $v\alpha_1$ binary operations, α_1 is one less than the column weight of G_1 , while matrix multiplication requires $v[2v(l-2)-1]$ binary operations. Similarly, the parity bits p_2 can also be obtained by using a shift register of size $v(l-1)$ that needs $v\alpha_2$ binary operations to complete the computation, where α_2 is one less than the column weight of G_2 . Since the encoding complexities of Encoding Step1 and Encoding Step2 are linear functions of to the code length, so is the total encoding complexity of the proposed structure which can be implemented by shift register and some combinatory logic.



Chapter 4

Simulation Results

In the beginning of this chapter, we will make a comparison of error correction performances by using some different structures of parity-check matrices such as irregular quasi-cyclic code, randomly constructed code and the proposed structure irregular code. Then, we will make a comparison of error correction performances by using some different decoding algorithms such as sum-product algorithm, min-sum based algorithm and min-sum algorithm. In the end, we will furthermore analyze the finite-precision effects on the decoding performance, and decide proper finite word lengths of variables considering tradeoffs between the performance and the hardware cost.

Before proceed to the following simulation, some parameters should be described here:

1: The polynomials of each of the circulant matrices of the proposed LDPC code structure are shown in Table 3.1. Three proposed structures of irregular LDPC codes have been constructed. When the rate is $2/3$ and code length is 720 with degree distribution $W=[4, 4, 4, 4, 5, 3]$, the parity-check matrix is of the form

$$H = \begin{bmatrix} A_5 & A_6 & A_7 & A_8 & A_9 & 0 \\ B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} \end{bmatrix} \quad (4.1)$$

where $A_5, A_6, \dots, A_9, B_5, B_6, \dots, B_9$ and B_{10} are 120×120 circulant matrices. When the rate is $3/4$ and code length is 960 with degree distribution $W=[4, 4, 4, 4, 4, 4, 5, 3]$,

the parity-check matrix is of the form

$$H = \begin{bmatrix} A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & 0 \\ B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} \end{bmatrix} \quad (4.2)$$

where $A_3, A_4, \dots, A_9, B_3, B_4, \dots, B_9$ and B_{10} are 120×120 circulant matrices. When the rate is $4/5$ and code length is 1200 with degree distribution $W=[4, 4, 4, 4, 4, 4, 4, 4, 5, 3]$, the parity-check matrix is of the form

$$H = \begin{bmatrix} A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & 0 \\ B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} \end{bmatrix} \quad (4.3)$$

where $A_1, A_2, \dots, A_9, B_1, B_2, \dots, B_9$ and B_{10} are 120×120 circulant matrices.

Table 4.1 Polynomials of each of the circulant matrices of the proposed LDPC code

structure

$a_1(x)$	$x^{20} + x^{37}$	$b_1(x)$	$x^2 + x^{56}$
$a_2(x)$	$x^{11} + x^{16}$	$b_2(x)$	$x^{33} + x^{46}$
$a_3(x)$	$x^6 + x^{21}$	$b_3(x)$	$x^{35} + x^{53}$
$a_4(x)$	$x^7 + x^{20}$	$b_4(x)$	$x^6 + x^{31}$
$a_5(x)$	$x^3 + x^{14}$	$b_5(x)$	$x^7 + x^{24}$
$a_6(x)$	$x^{11} + x^{13}$	$b_6(x)$	$x^{20} + x^{31}$
$a_7(x)$	$x^1 + x^7$	$b_7(x)$	$x^4 + x^{13}$
$a_8(x)$	$x^2 + x^5 + x^{34}$	$b_8(x)$	$x^3 + x^7$
$a_9(x)$	$1 + x^{10} + x^{30}$	$b_9(x)$	x^{43}
		$b_{10}(x)$	$1 + x^{10} + x^{30}$

2: The polynomials of each of the circulant matrices of the irregular quasi-cyclic codes are shown in Table 3.2. Three quasi-cyclic irregular LDPC codes have been

constructed. When the rate is 2/3 and code length is 720 with degree distribution $W=[4, 5, 3]$, the parity-check matrix is of the form

$$H = [A_3 \quad A_4 \quad A_5] \quad (4.4)$$

where A_3, A_4 and A_5 are 240×240 circulant matrices. When the rate is 3/4 and code length is 960 with degree distribution $W=[4, 4, 5, 3]$, the parity-check matrix is of the form

$$H = [A_2 \quad A_3 \quad A_4 \quad A_5] \quad (4.5)$$

where A_2, A_3, A_4 and A_5 are 240×240 circulant matrices. When a rate 4/5, code length is 1200 with degree distribution $W=[4, 4, 4, 5, 3]$, the parity-check matrix is of the form

$$H = [A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5] \quad (4.6)$$

where A_1, A_2, A_3, A_4 and A_5 are 240×240 circulant matrices.

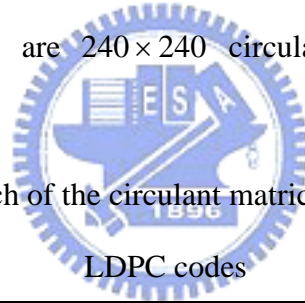


Table 4.2 Polynomials of each of the circulant matrices of the quasi-cyclic irregular LDPC codes

$a_1(x)$	$1 + x^3 + x^{21} + x^{45}$
$a_2(x)$	$x^3 + x^{43} + x^{84} + x^{101}$
$a_3(x)$	$x + x^{51} + x^{57} + x^{65}$
$a_4(x)$	$x^2 + x^6 + x^{11} + x^{18} + x^{33}$
$a_5(x)$	$1 + x^{10} + x^{30}$

3: The randomly constructed codes are derived from [14] and [15], and they have a regular column weight of four with similar parameters. This means that for a rate of 2/3 and code length of 720 with a random structure, the column weight is four and the averaged row weight is twelve. Similarly, for a rate of 3/4 and code length of 960 with

a random structure, the column weight is four and the average row weight is sixteen. Finally, for a rate of 4/5 and code length of 1200 with a random structure, the column weight is four and the average row weight is twenty.

4: For the decoding algorithm, we adopt the sum-product algorithm, min-sum based algorithm and min-sum algorithm. The maximum iteration loops = 10 .

5: We use the AWGN channel and BPSK modulation method as our test environment.

4.1 Floating-Point Simulations

Figures 4.1-4.3 show the error correction performance for different structures of the parity-check matrix that use the sum-product algorithm for iterative decoding. We can see that in Figures 4.1-4.3, using the proposed structures of the parity-check matrix, the decoding performance is the best, compared to the irregular quasi-cyclic codes and randomly constructed codes. Figures 4.4-4.6 show the error correction performance for different decoding algorithms such as the sum-product algorithm, the min-sum based algorithm and the min-sum algorithm. In the simulations and figures the proposed parity-check matrix structures assume some different code lengths and code rates. We can see that in Figures 4.4-4.6, the decoding performances are almost the same for the sum-product and the min-sum based algorithms combined with iterative decoding. As shown, the min-sum algorithm has the worst performance of all the compared algorithms. This is due to the fact that the min-sum algorithm in the check node update is an approximate form and using the approximation will cause a performance penalty of about 0.5dB.

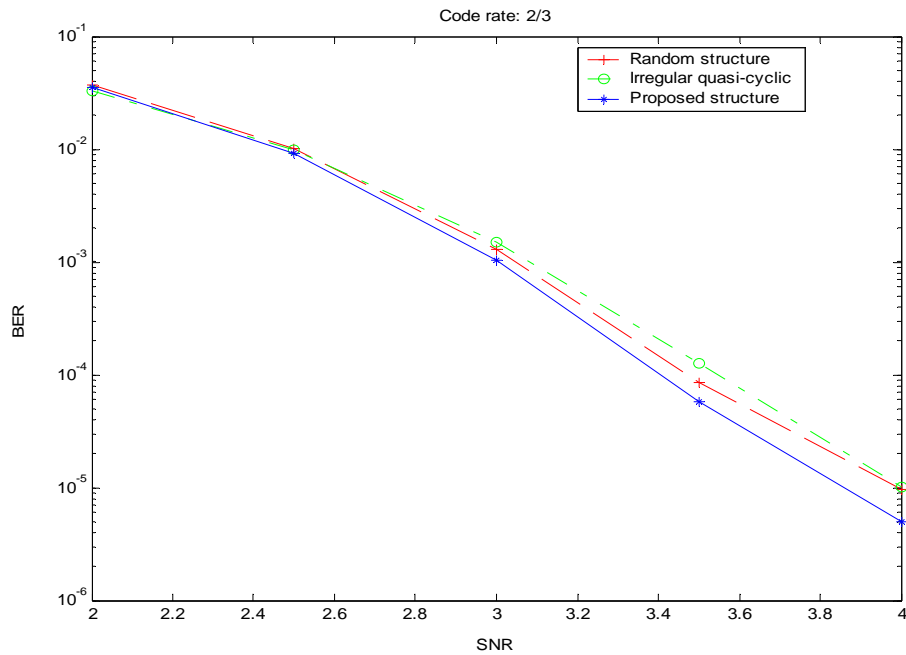


Figure 4.1 Floating-point simulations of various parity-check matrix structures in AWGN channel, code length=720, code rate=2/3, maximum iteration=10, using the sum-product algorithm

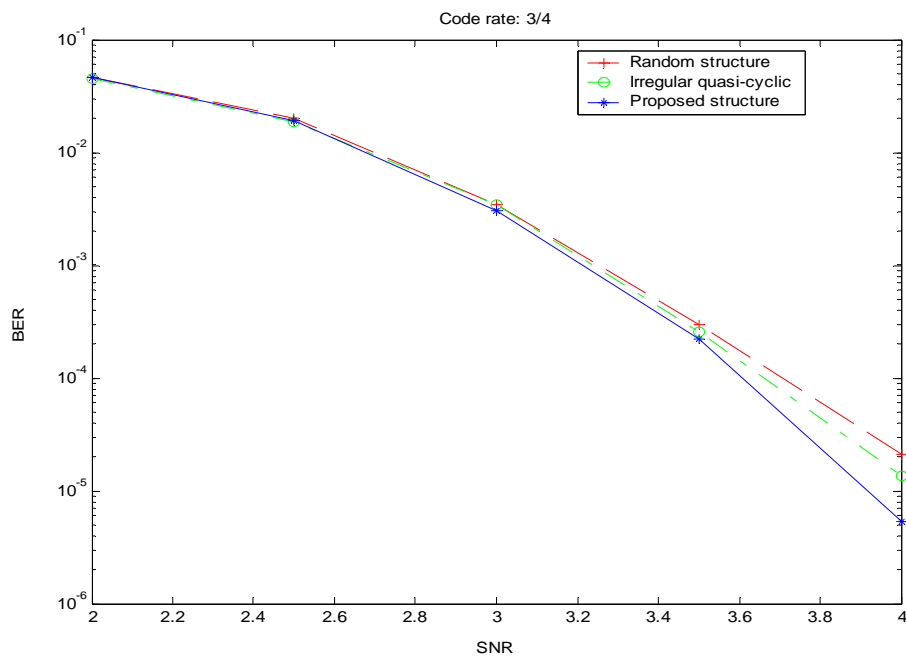


Figure 4.2 Floating-point simulations of various parity-check matrix structures in AWGN channel, code length=960, code rate=3/4, maximum iteration=10, using the sum-product algorithm

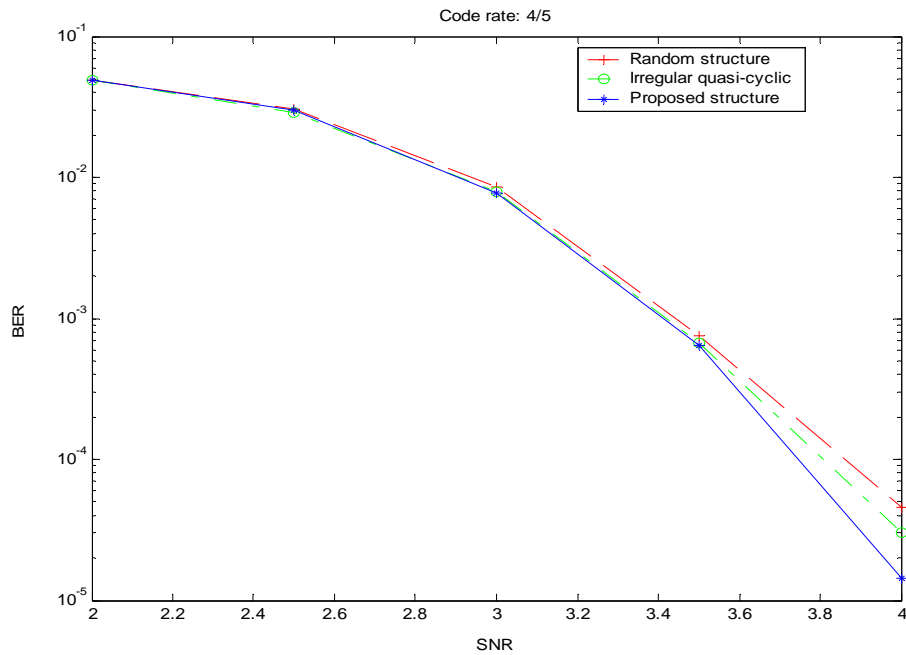


Figure 4.3 Floating-point simulations of various structure parity-check matrix structures in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10, using the sum-product algorithm

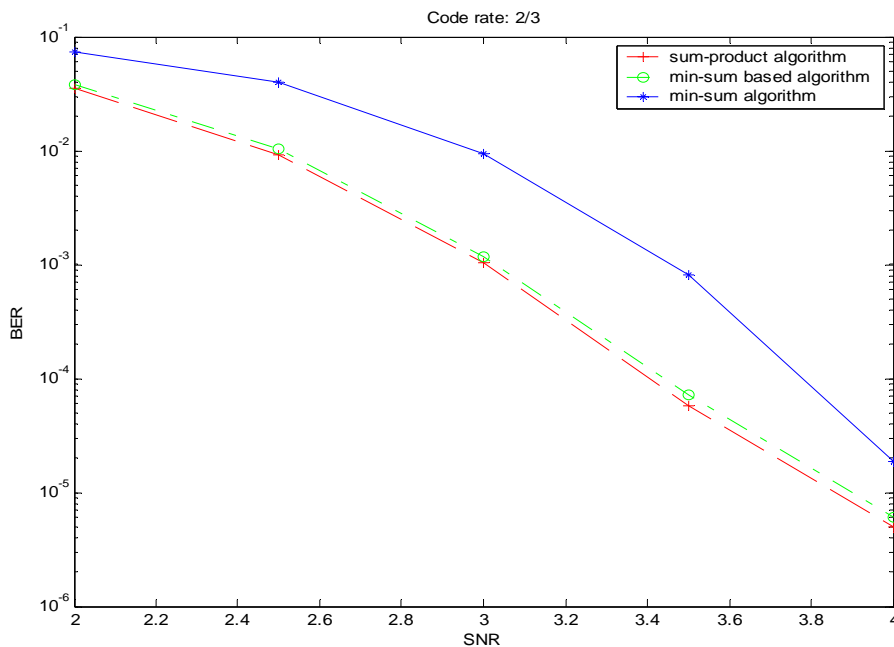


Figure 4.4 Floating-point simulations of the proposed parity-check matrix structure, under the three decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

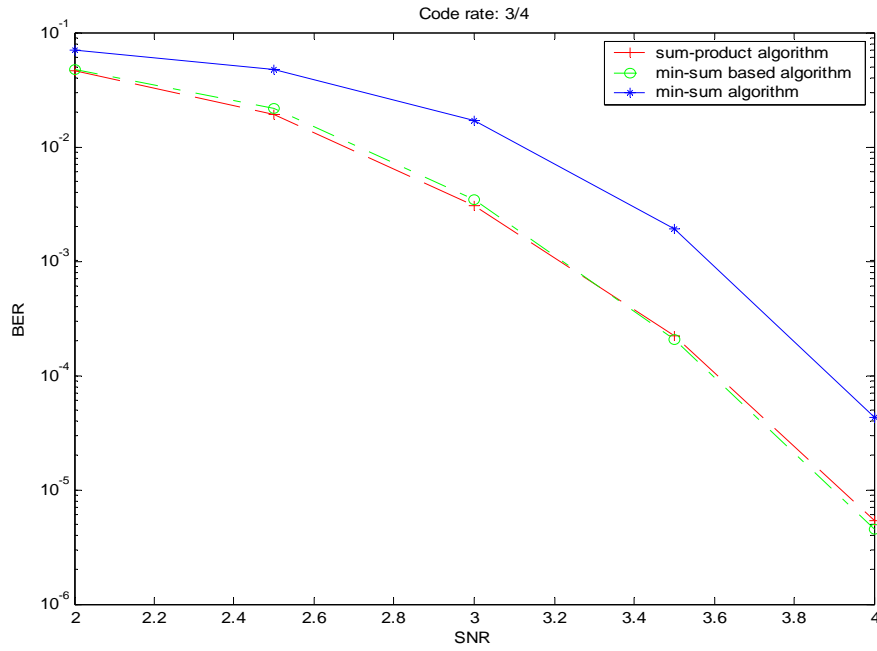


Figure 4.5 Floating-point simulations of the proposed parity-check matrix structure, under the three decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

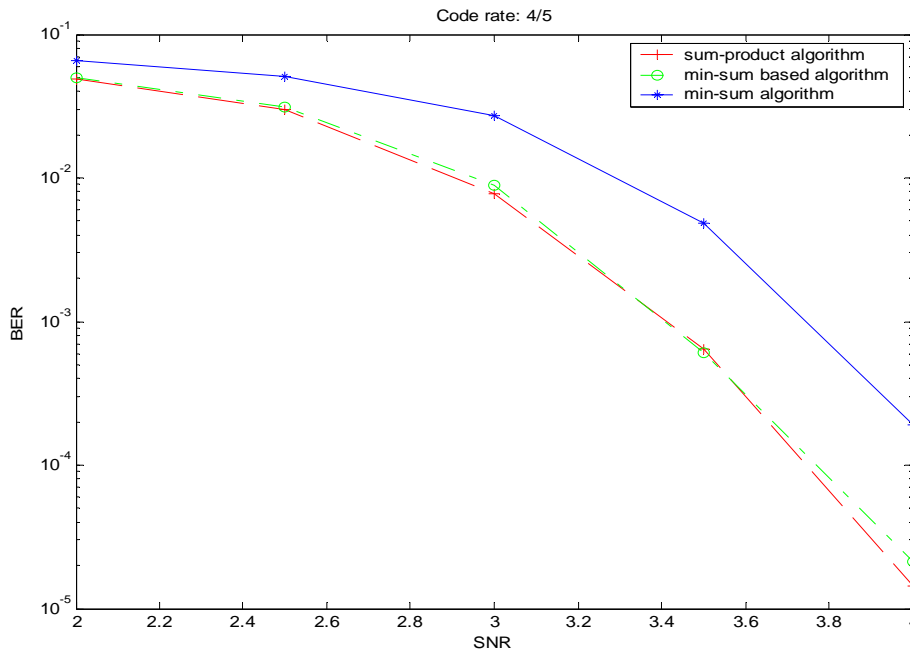


Figure 4.6 Floating-point simulations of the proposed parity-check matrix structure, under the three decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

4.2 Fixed-Point Simulations

In this section, we furthermore analyze the finite-word-length performance of the proposed LDPC codes. Possible tradeoff between hardware complexity and decoding performance will be discussed. It is shown that the performance degradation from the infinite precision is negligible if 6 bits are used for the initially received signal and 6 bits for the extrinsic messages $r_{m,l}$ and $q_{m,l}$.

4.2.1 Quantization of Initially Received Signal

We first consider the quantization of the initially received signal. Since a receiving buffer is needed for storing the received signal, quantization of the initially received signal significantly affects the total decoder complexity. A long word length not only increases the hardware overhead for the buffers, but also causes a large amount of hardware for the iterative decoding computation, while a short word length may result in very poor performance. Let $[t:f]$ denote the quantization scheme in which a total of t bits are used, of which f bits are used for the fractional part of the value. Various quantization schemes for the initially received signal such as [5:2], [6:2] and [7:3] are investigated here. It should be noted that if we use the min-sum based algorithm for iterative decoding, the quantized initially received signal can not be 0, because when the quantized signal is 0, the results of the check node update operation will also be 0 and will thus lose the ability of error correction. So if we adopt the min-sum based algorithm as the iterative decoding algorithm, we will restrict the quantized signal to a specified minimum value when the initially received

signal is close to 0. That means when we use the quantization schemes such as [5:2] and [6:2], the minimum quantized values will be ± 0.25 , and when the quantization scheme is [7:3], the minimum quantized values will be ± 0.125 . Figures 4.7-4.12 show the decoding performances of using these three different quantization schemes and various code lengths. It can be seen that the difference between [6:2] and [7:3] quantization schemes is quite small and the [5:2] is far away (by more than 0.2dB) from [6:2] and [7:3] schemes. Thus [6:2] scheme is the best choice.

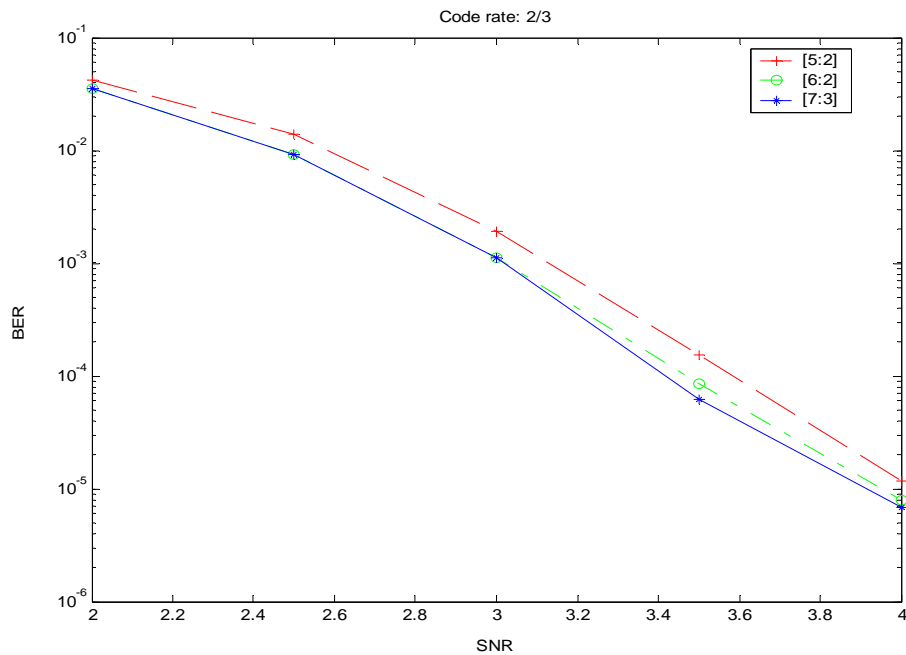


Figure 4.7 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

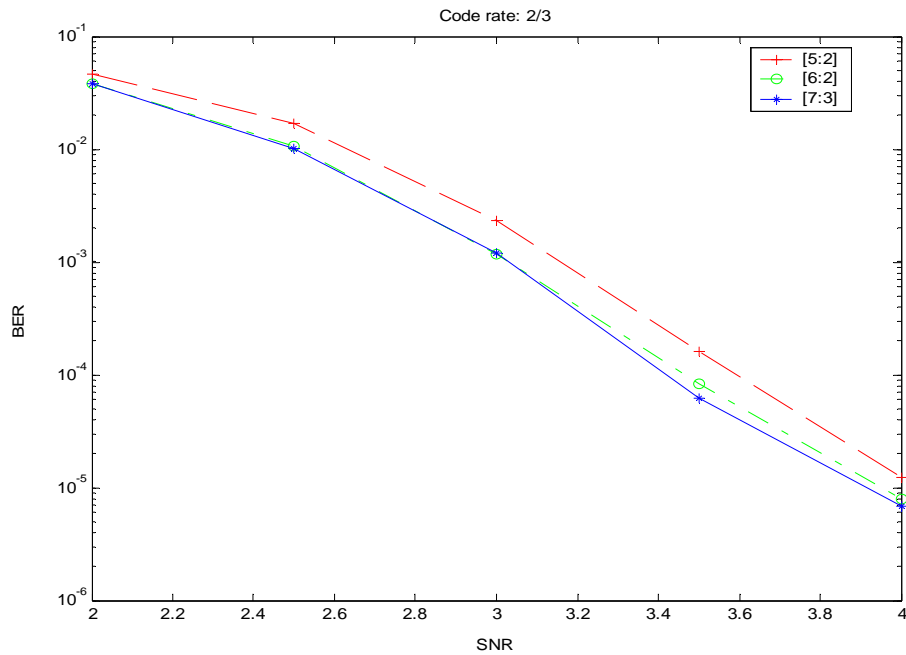


Figure 4.8 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

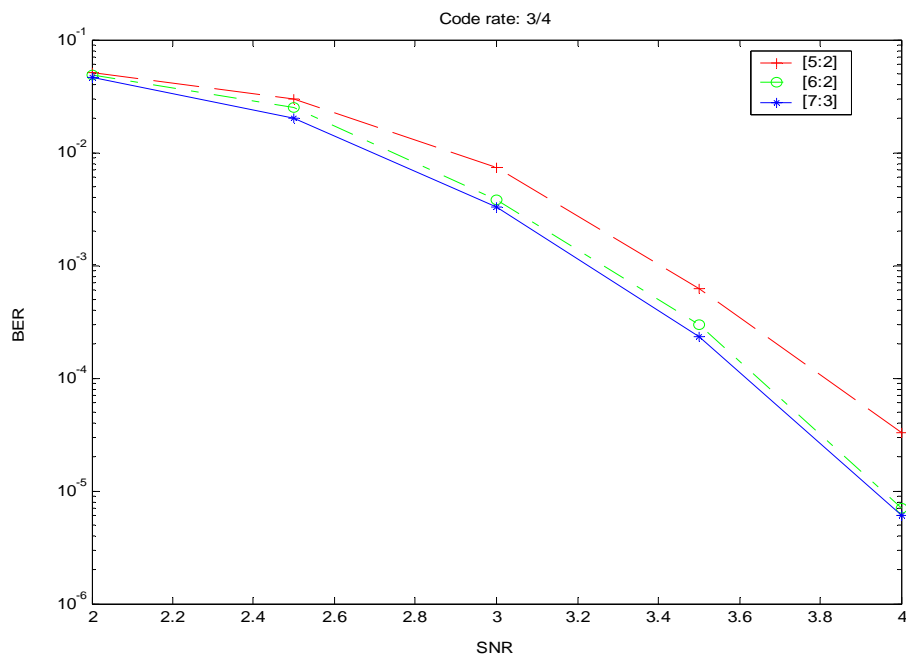


Figure 4.9 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

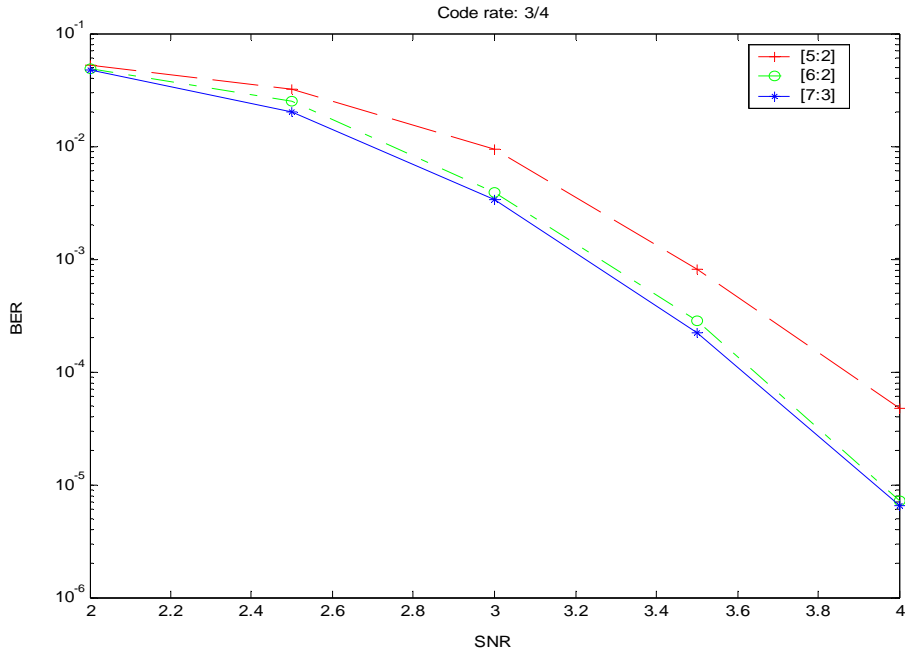


Figure 4.10 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

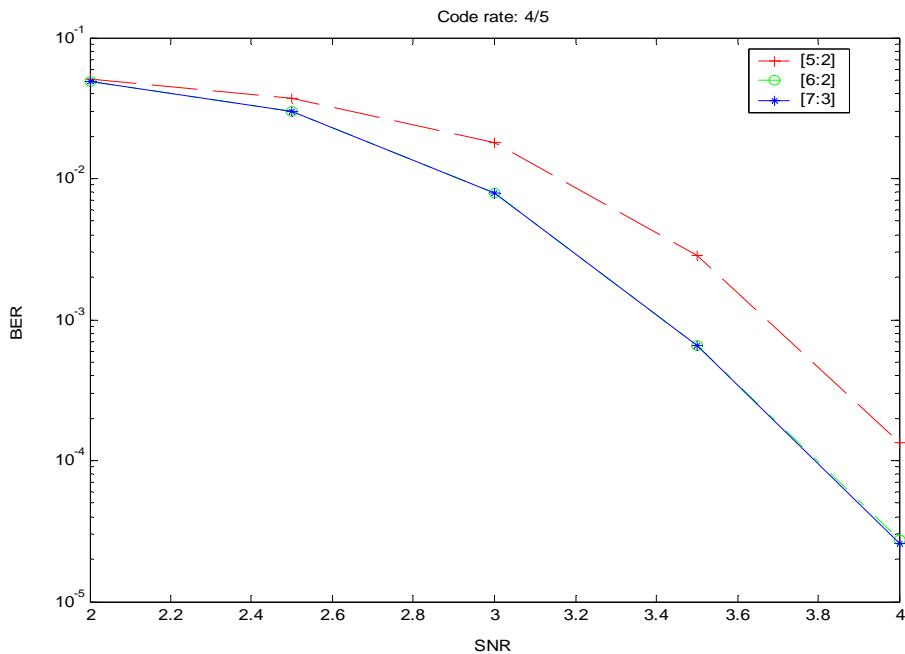


Figure 4.11 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

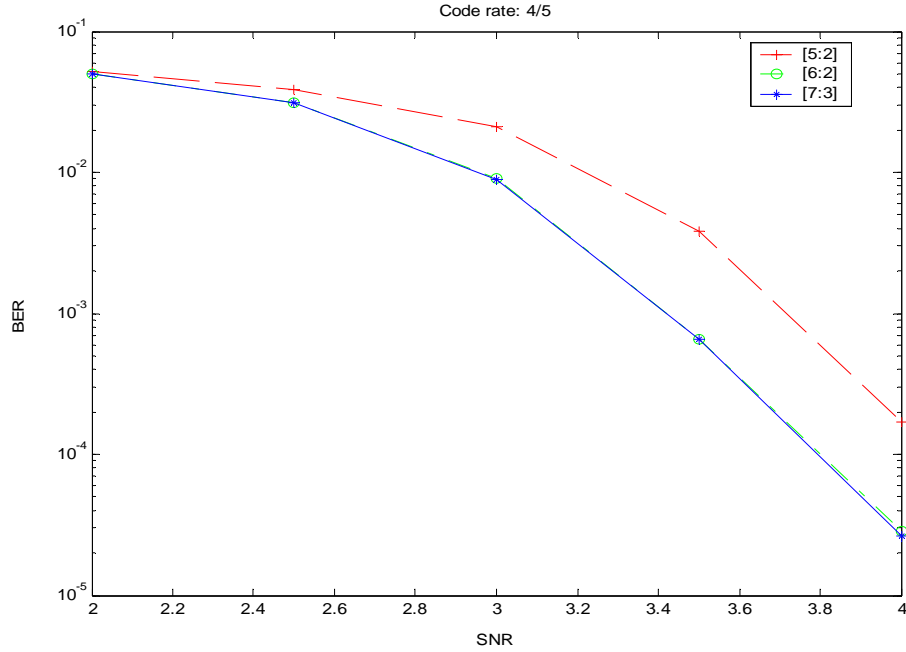


Figure 4.12 Three different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

4.2.2 Quantization of $r_{m,l}$ and $q_{m,l}$

We know that the whole decoding process mainly consists of iteratively exchanging and updating the extrinsic messages $r_{m,l}$ and $q_{m,l}$, performed by the check node update operations and the variable node update operations, respectively. Therefore, quantization of $r_{m,l}$ and $q_{m,l}$ is also critical for hardware implementation. Various quantization schemes for the extrinsic messages $r_{m,l}$ and $q_{m,l}$ such as [6:2] and [7:3] have been examined in this work. It turns out that there is almost no difference in the decoding performance for the [6:2] and [7:3] quantization schemes. Simulation results for these schemes with various code lengths are shown in Figures 4.13-4.18. Thus we suggest that the [6:2] scheme to be the best choice.

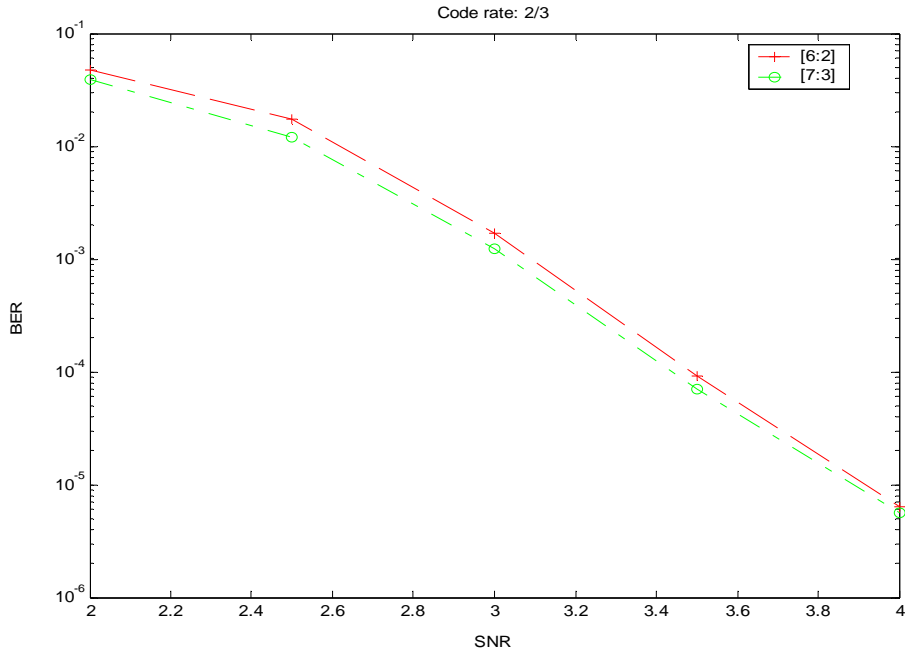


Figure 4.13 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

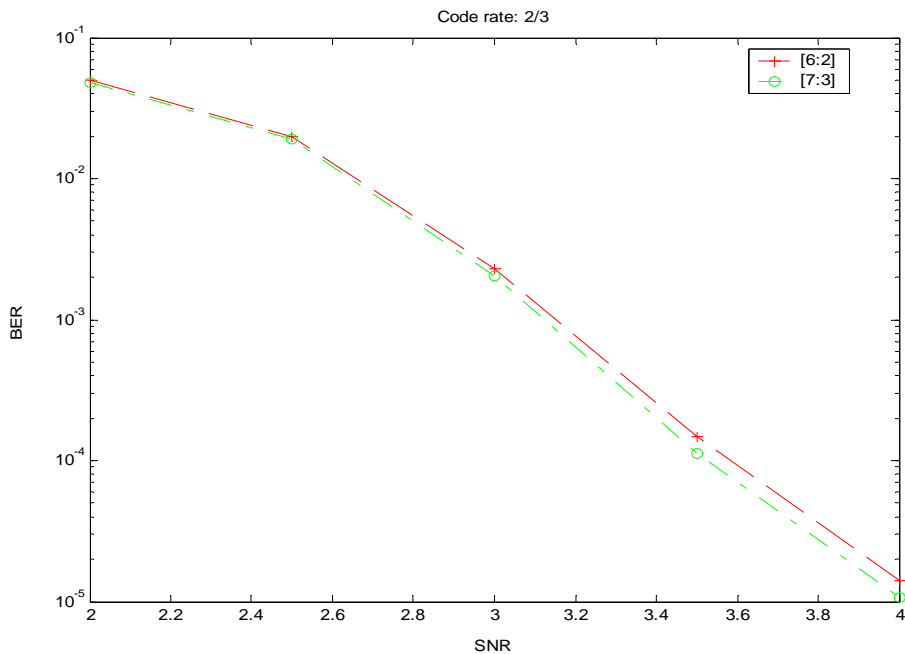


Figure 4.14 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

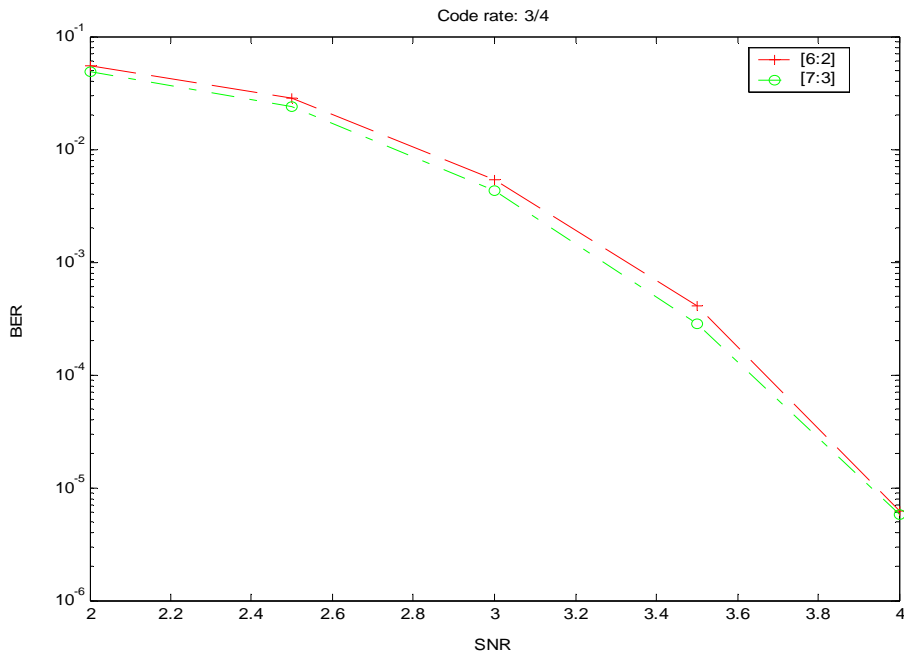


Figure 4.15 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

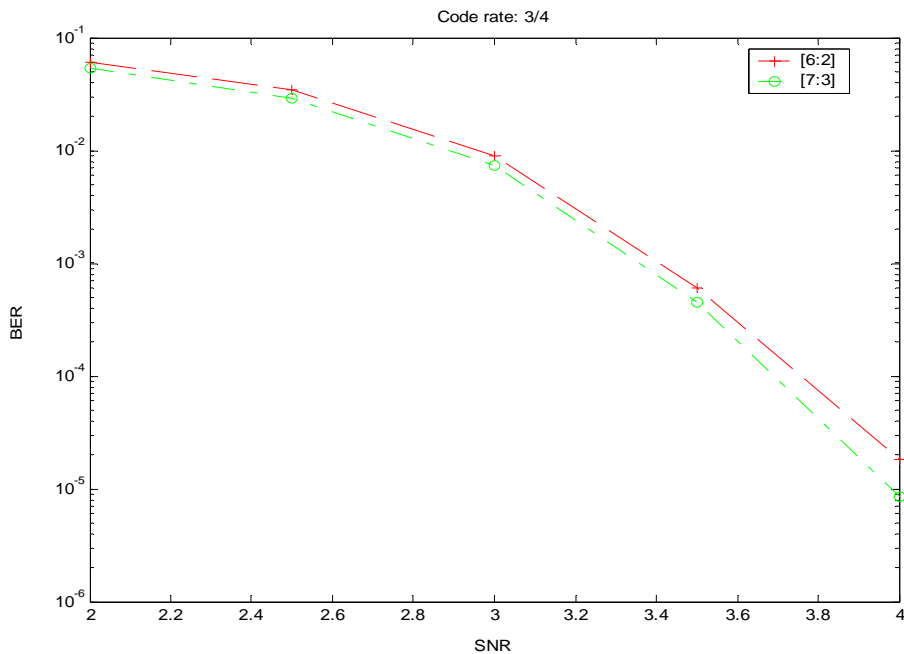


Figure 4.16 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

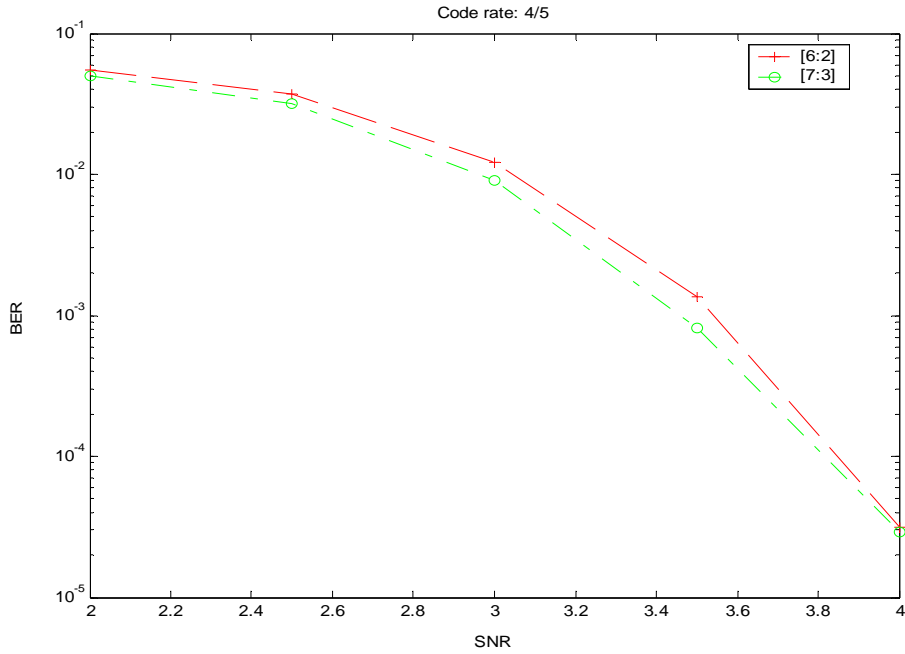


Figure 4.17 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the sum-product decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

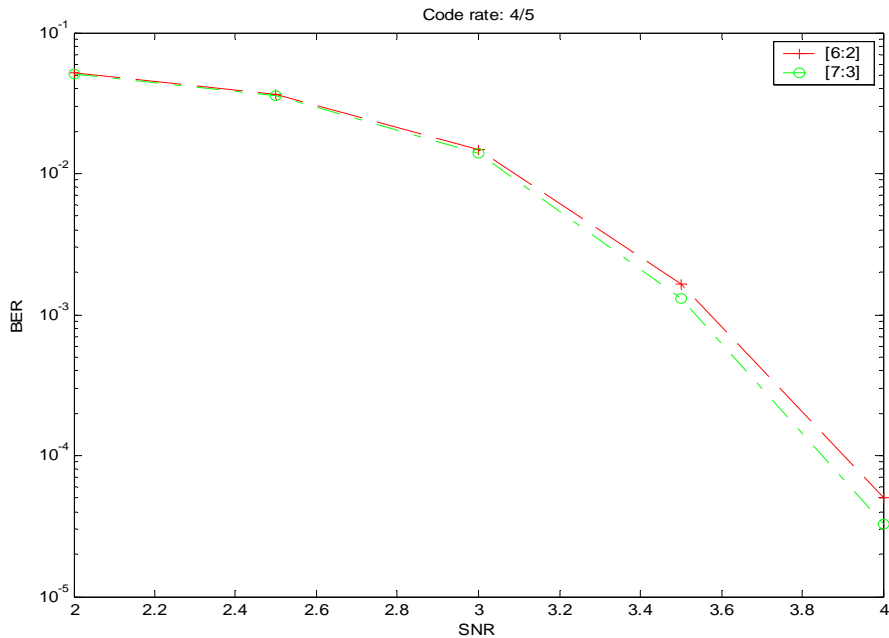


Figure 4.18 Two different fixed-point simulation results of the proposed parity-check matrix structure, based on the min-sum based decoding algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

4.2.3 Summary of Fixed-Point Simulation Results

Floating-point and fixed-point simulation results are shown in Figures 4.19-4.21, including the bit-error-rate (BER) and signal-to-noise ratio (SNR). The quantization scheme [6:2] are for both the initially received signal and the extrinsic messages $r_{m,l}$ and $q_{m,l}$. It can be seen that, for cases with code lengths 720, 960 and 1200, the total quantization loss compared with the floating-point case is about 0.1dB when using the sum-product algorithm as the decoding algorithm, and the loss compared with the fixed-point case is about 0.2dB when using the min-sum based algorithm.

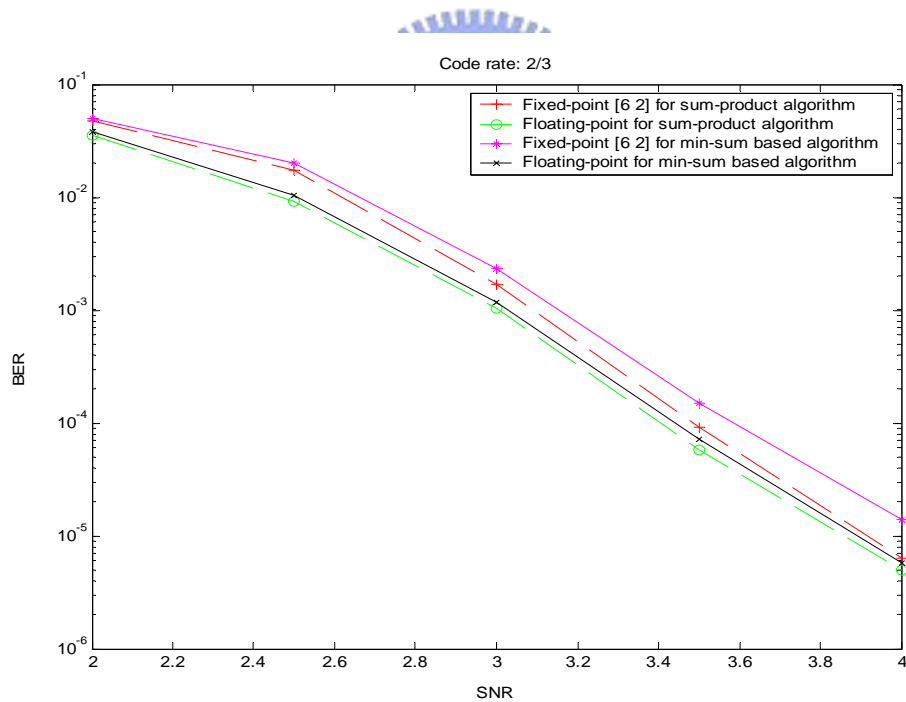


Figure 4.19 Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=720, code rate=2/3, maximum iteration=10

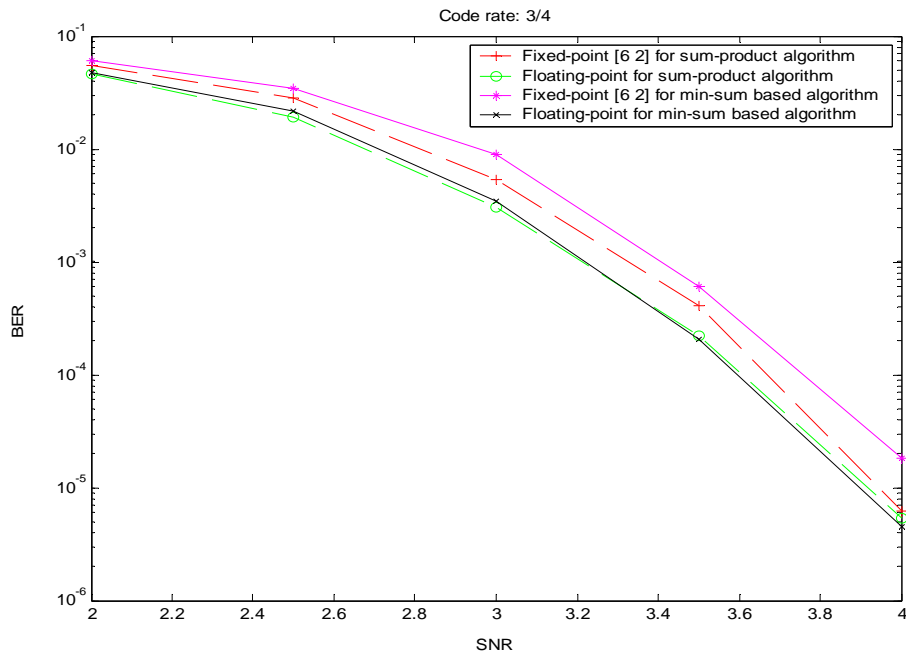


Figure 4.20 Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=960, code rate=3/4, maximum iteration=10

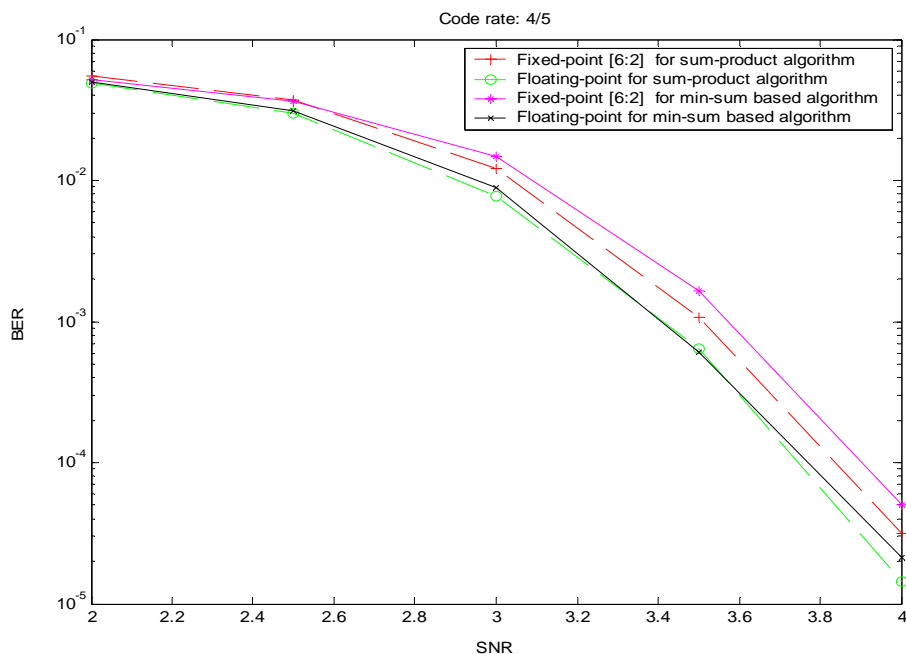


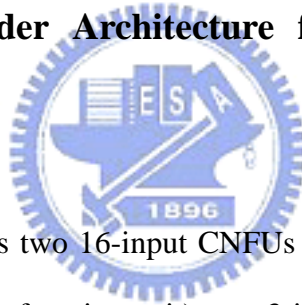
Figure 4.21 Floating-point vs. fixed-point simulation results of the proposed parity-check matrix structure for the sum-product and min-sum based algorithm in AWGN channel, code length=1200, code rate=4/5, maximum iteration=10

Chapter 5

VLSI Implementation of LDPC Decoder

In this chapter, we will implement an irregular LDPC decoder of rate 3/4, and code length 960. The parity-check matrix of this code was discussed in the last chapter and is adopted as our proposed structure.

5.1 Semi-parallel Decoder Architecture for the Proposed LDPC Codes



This architecture includes two 16-input CNFUs (Check node function unit), six 4-input VNFUs (Variable node function unit), one 3-input VNFU, one 5-input VNFU and 32 extrinsic message register-sets $R_{A,1}, R_{A,2}, \dots, R_{A,16}, R_{B,1}, \dots, R_{B,16}$ with each register-set R containing 120 symbols, where each symbol is represented by 6 bits. Figure 5.1 shows the block diagram of this decoder. The input signals are retrieved from L registers and x registers store the hard decisions of the soft outputs from VNFU. Consequently, the decoding process could be carried out as follows:

1. Initialization

Flush the received initial signals to both the L registers and the corresponding extrinsic message register-sets R . The data is stored serially in the L registers and the extrinsic message register-sets R .

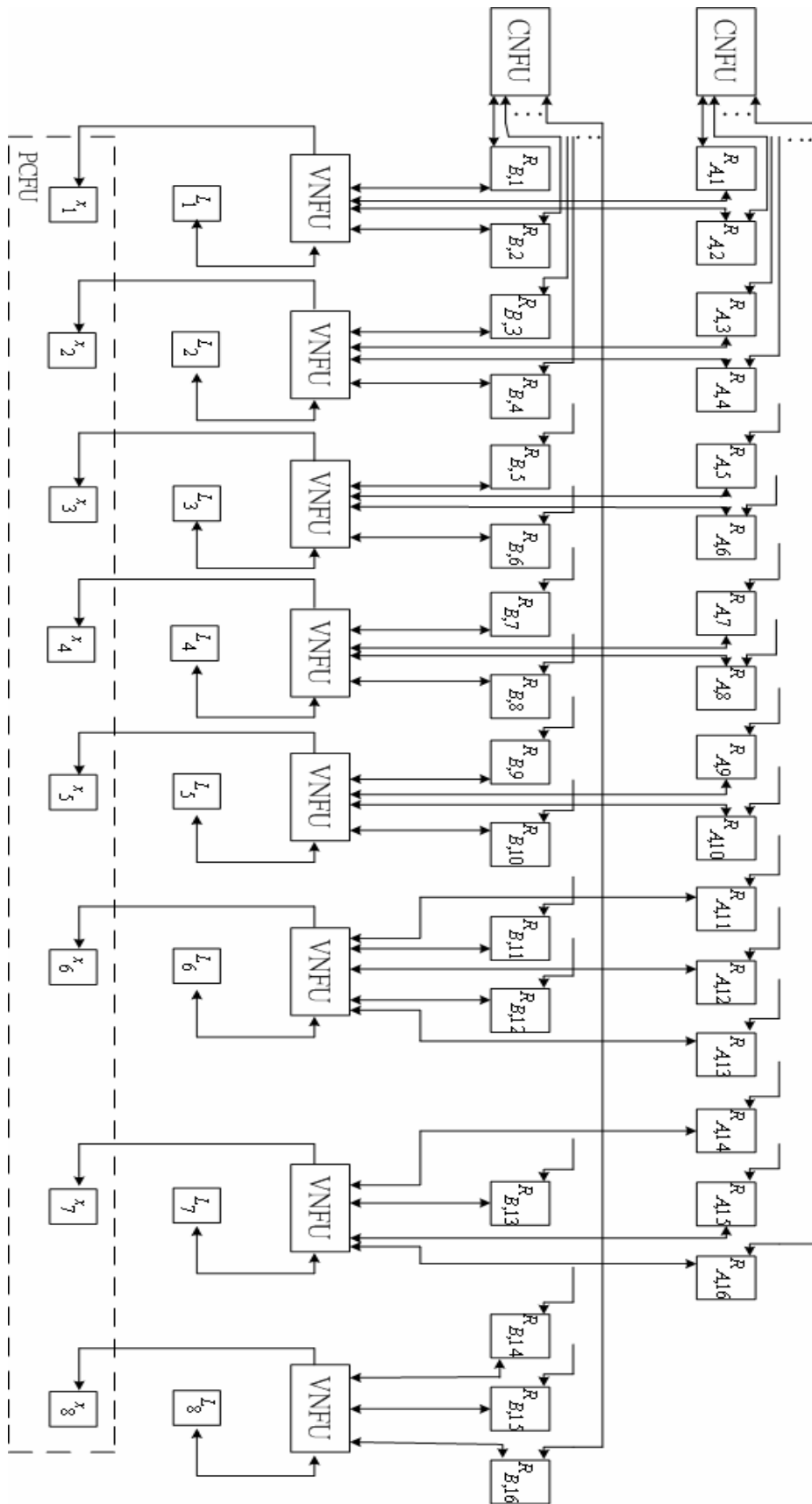


Figure 5.1 Semi-parallel decoder for the proposed irregular LDPC code structure of rate 3/4, and code length 960

2 Message passing

Step1 (message passing from check nodes to variable nodes): In each subsequent iteration, the update variable-to-check messages are simultaneously read from all the extrinsic message register-sets R by all the CNFUs, the positions of each of the update variable-to-check message can be selected by a multiplexer, and the control signal can be controlled by a simple counter. After the CNFU computation, the updated check-to-variable messages are stored back to the same positions, and this stored back operation can be controlled by a de-multiplexer, while the control signal to the de-multiplexer can also be controlled by a simple counter.

Step2 (messages passing from variable nodes to check nodes): Similarly, in the same iteration, the updated check-to-variable messages are simultaneously read from all the extrinsic message register-sets R by all the VNFUs. After the VNFU computation, the updated variable-to-check messages are stored back to the extrinsic message register-sets R and the hard decisions of the soft output made from each of the VNFU are at the same time stored in registers x .

Step3 (decoding): At the end of each of the decoding iterations, the PCFU (Parity-check function unit) starts to check all the parity-check equations. The iterative process will be terminated when either one codeword \hat{x} satisfying $H\hat{x} = 0^T$ is found, or the pre-assigned maximum number of iterations is reached.

Step1 and Step3 of each decoding iteration are executed in overlap. In other words, when we are executing step3 of the i^{th} iteration, step1 of the $i+1$ -iteration is being computed simultaneously. Figure 5.2 shows the snap shot of the overlapped operations. This procedure can reduce the cycles of each decoding iteration and can increase the data throughput of the whole decoding procedure.

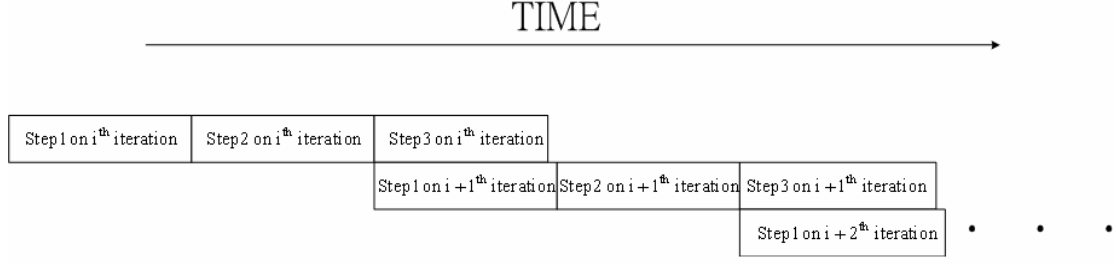


Figure 5.2 Illustration of overlapped decoding procedure

5.2 Architectures of the Check Node Function Unit and the Variable Node Function Unit

In this section, we will discuss the architecture of the check node function unit and the variable node function unit for the sum-product algorithm and the min-sum based algorithm respectively. After this discussion, we will conclude that using the sum-product algorithm is better than the min-sum based algorithm in terms of VLSI implementation. In the end of this section, we will select the architecture of the sum-product algorithm as our check node function unit and variable node function unit and give a summary to the whole LDPC decoder.

For now, let's review the sum-product algorithm. The check-to-variable message $r_{m,l}$ for the check node m and variable node l using the incoming variable-to-check messages $q_{m,l}$ is computed by CNFU as follows.

$$r_{m,l} = \text{sign}(q_{m,l}) \prod_{l' \in L(m)} \text{sign}(q_{m,l'}) \times \phi(\phi(\sum_{l' \in L(m)} |q_{m,l'}|) - \phi(|q_{m,l}|)). \quad (5.1)$$

where $L(m)$ denotes the set of variable nodes connected to the check node m . The function $\phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$ can be implemented with look-up-table (LUT) operations. Figure 5.3 shows the curve of function $\phi(x)$. On the other hand, the variable-to-check message $q_{m,l}$ for the check node m and variable node l using

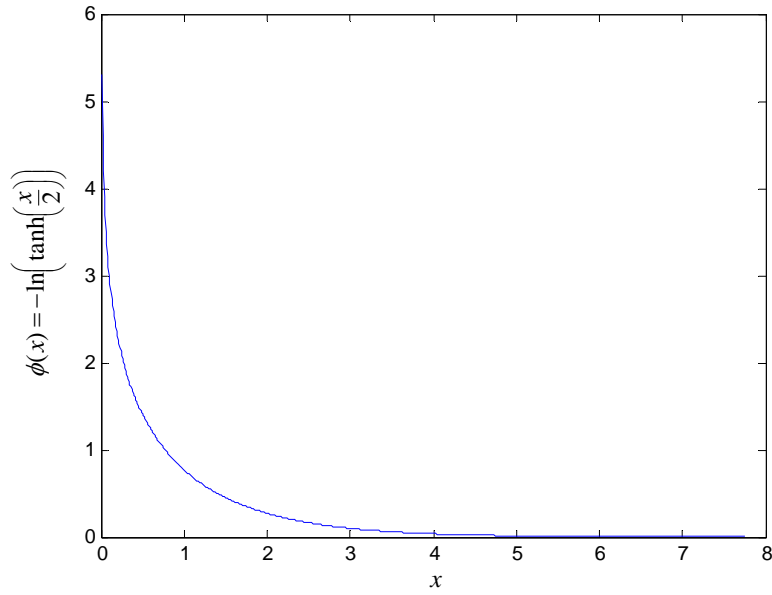


Figure 5.3 Function plot of $\phi(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$

the incoming check-to-variable messages $r_{m,l}$ and received initialized signal L_l is

computed by VNFU,

$$q_l = \text{VAR}\left(\text{VAR}(r_{m,l}, L_l)_{m \in M(l)}\right) = L_l + \sum_{m \in M(l)} r_{m,l} \quad (5.2)$$

$$q_{m,l} = q_l - r_{m,l} \quad (5.3)$$

where $M(l)$ is the set of check nodes connected to variable node l and $L_l = \frac{2}{\sigma^2} y_l$.

According to the above algorithm, the CNFU and VNFU can be implemented as illustrated in Figure 5.4 and Figure 5.5 respectively.

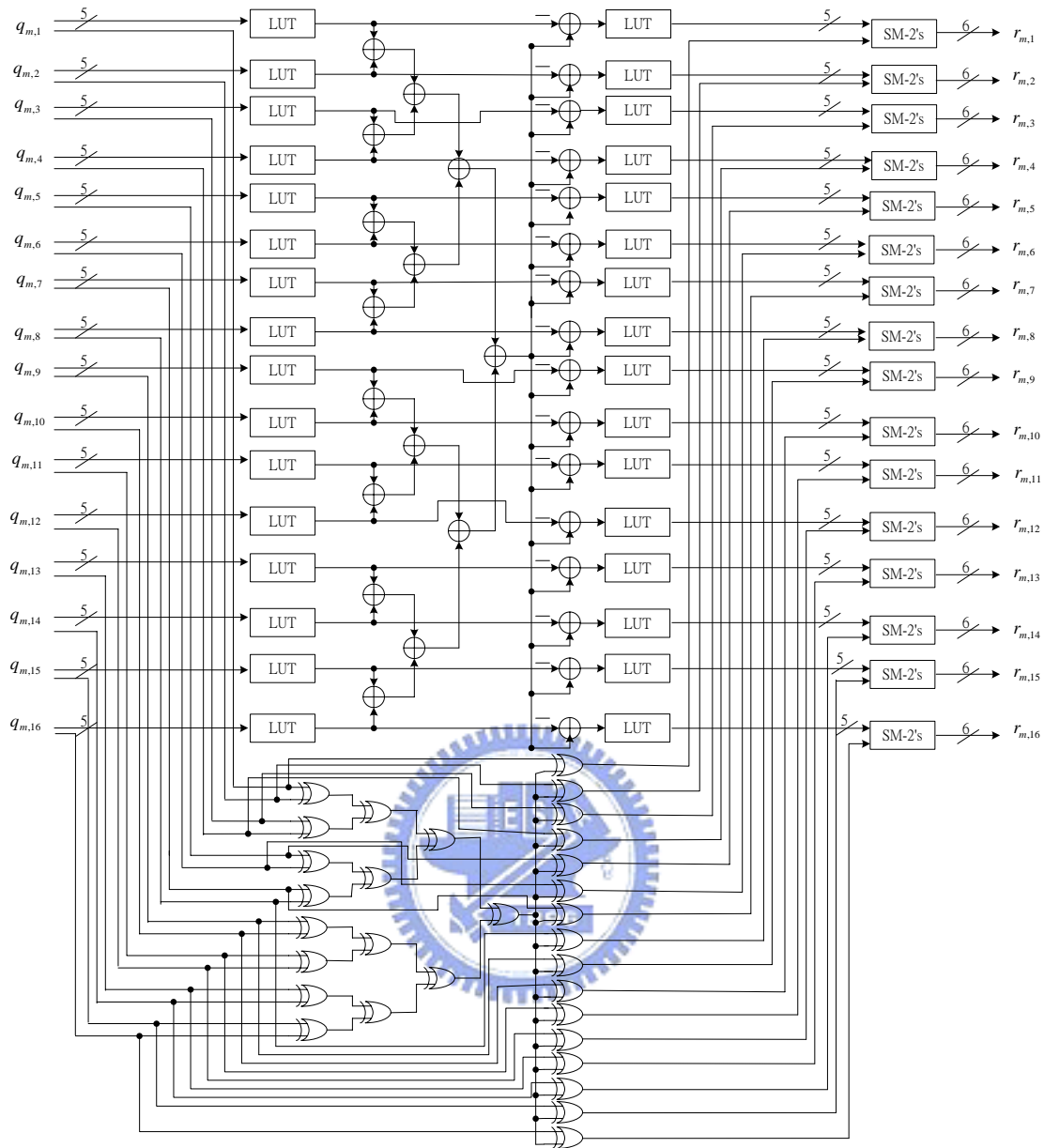


Figure 5.4 Architecture of check node function unit for the sum-product algorithm

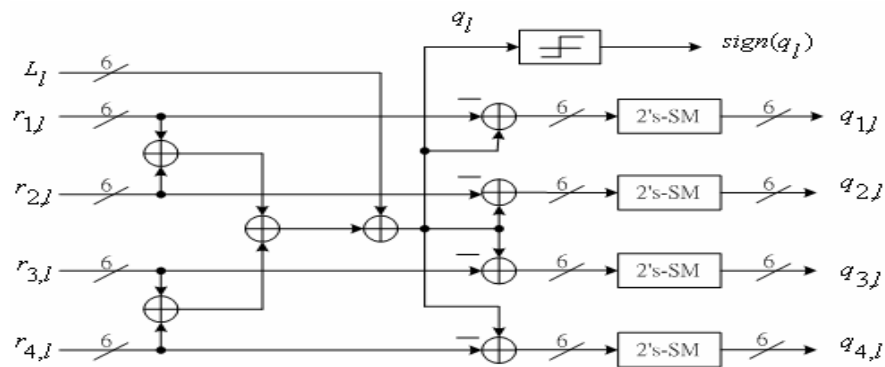


Figure 5.5 Architecture of a 4-input variable node function unit for the sum-product algorithm

It is worth noting that the data format transformation block, either from sign-magnitude (SM) to two's complement (2's) format or vice versa, exists in both types of functional units. The major advantage of using the sign-magnitude format for LUT operations is that each LUT size can be reduced by half by making use of the symmetry properties of the $\phi(x)$ function. As we can see in Figure 5.4 and Figure 5.5, CNFUs are more complicated than VNFUs. Two LUT operations are involved in the critical path of each CNFU. [16] introduced a method to re-distribute the computation load between CNFUs and VNFUs. In this method, it moves one LUT operation to the critical path of every VNFU. The sum-product algorithm could be equivalently reformulated as below for CNFUs,

$$r_{m,l} = \text{sign}(q_{m,l}) \prod_{l' \in L(m)} \text{sign}(q_{m,l'}) \times (\phi(\sum_{l' \in L(m)} |q_{m,l'}|) - \phi(|q_{m,l}|)) \quad (5.4)$$

and for VNFUs,

$$q_l = \text{VAR}(\text{VAR}_{m \in M(l)}(r_{m,l}), L_l) = L_l + \text{sign}(r_{m,l}) \times \phi(\sum_{m \in M(l)} r_{m,l}) \quad (5.5)$$

$$q_{m,l} = q_l - \text{sign}(r_{m,l}) \times \phi(r_{m,l}). \quad (5.6)$$

where all the notations remain the same as before. As a result, their corresponding architectures are depicted in Figure 5.6 and Figure 5.7 respectively.

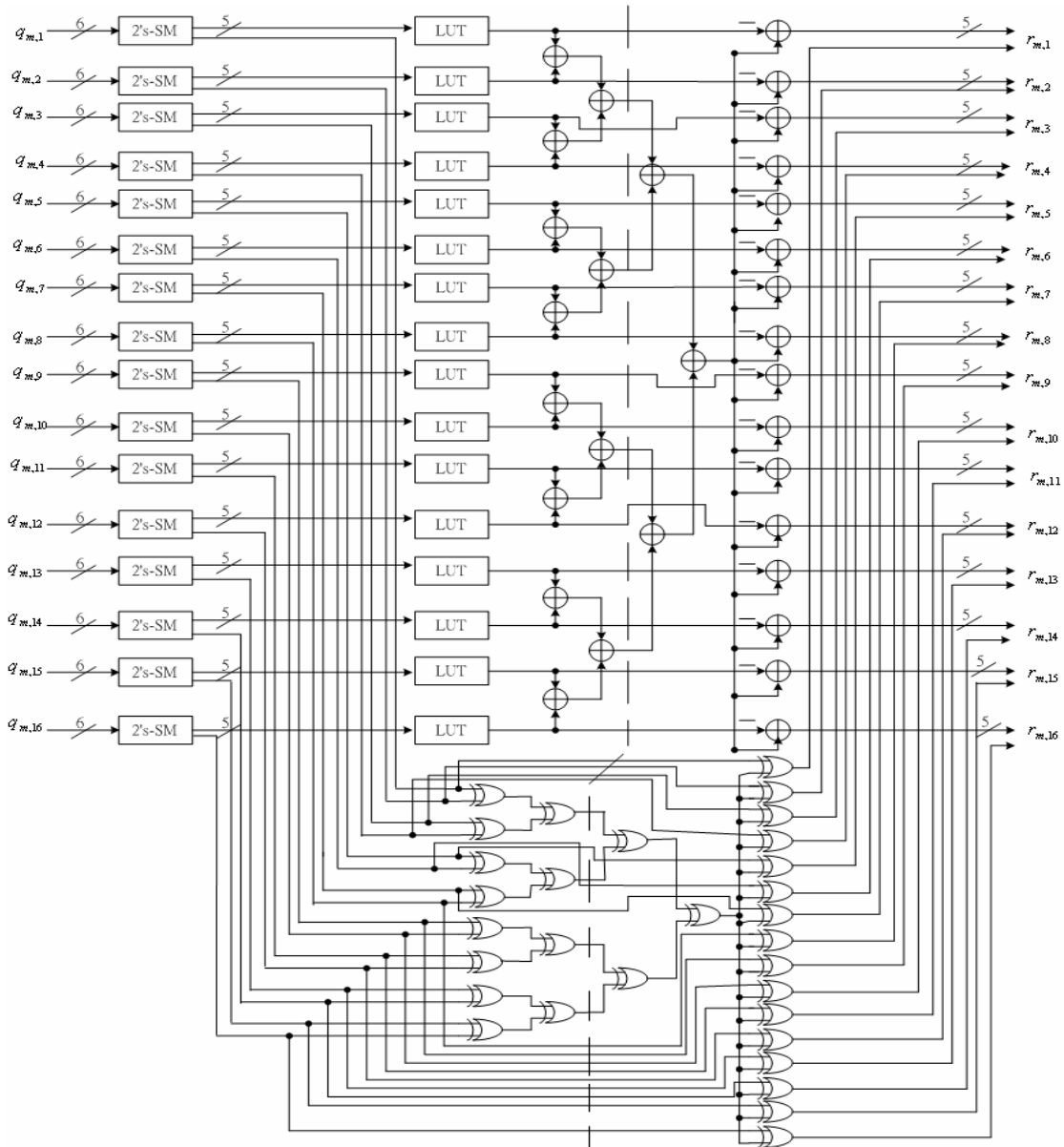


Figure 5.6 Reformulated architecture of check node function unit for the sum-product algorithm

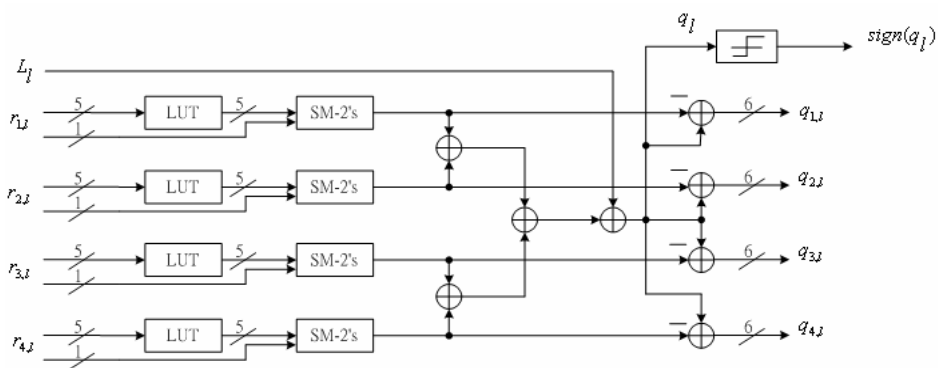


Figure 5.7 Reformulated architecture of a 4-input variable node function unit for the sum-product algorithm

The major benefit of the reformulated architecture is that the computation complexity is more equally shared amongst the CNFUs and VNFUs, resulting to a more balanced computation delay between the two. Besides, the dashed lines indicate possible positions for inserting pipeline stages and can further reduce the critical paths on both the CNFUs and VNFUs. Since we know that in each decoding iteration both check node update operation and variable node update operation have to be performed one after another. This leads to merely 50% hardware utilization efficiency (HUE) of the CNFUs and VNFUs, because all the VNFUs are idle when CNFUs are busy during the check node update and vice versa during the variable node update. To improve the HUE of CNFUs and VNFUs, [16] also introduced a re-mapped architecture that combines the CNFUs and VNFUs into the same hardware by making use of similarity between the CNFUs and VNFUs and get a smaller area design. The re-mapped architecture is shown in Figure 5.8. In Figure 5.8, there are a total of 32 inputs and 32 outputs denoted as In_s and Out_s respectively where $1 \leq s \leq 32$. The re-mapped architecture performs CNFU operations when the control signal is '0', thus the inputs are variable-to-check messages and the outputs are check-to-variable messages. On the other hand, when the control signal switches to '1', VNFU operations are performed, where the inputs are check-to-variable messages and the outputs are variable-to-check messages. Therefore, this architecture can perform both the check node update operations and the variable node update operations on the same piece of hardware, which will always be busy during every iteration and thus increasing the HUE to 100%. Last of all, the dashed lines in Figure 5.8 represent the possible positions for inserting the pipeline stages.

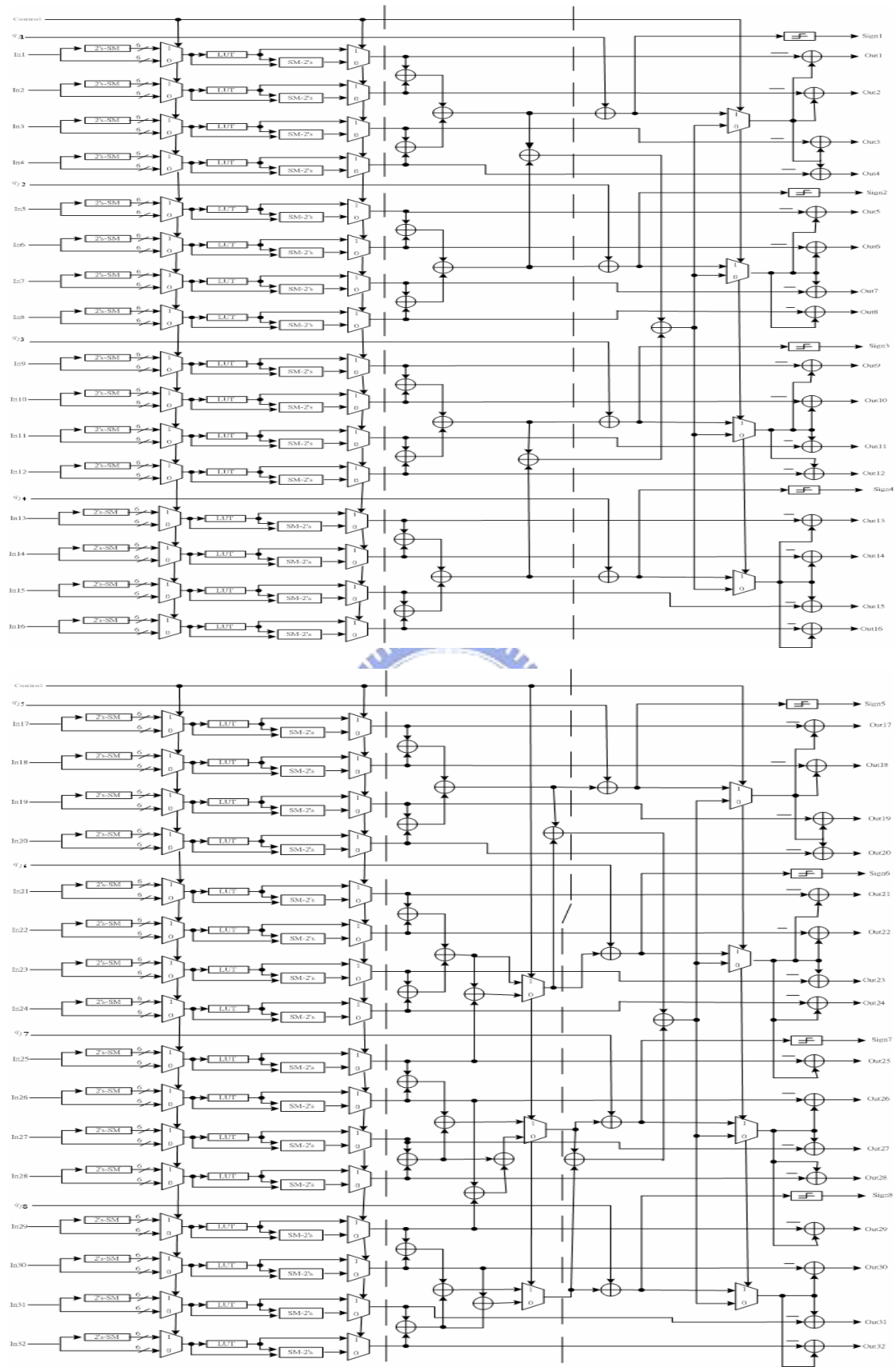


Figure 5.8 Re-mapped architecture performing both check node update and variable node update operations for the sum-product algorithm

Regarding architectures of CNFUs and VNFUs for the min-sum based algorithm, as already reviewed in the beginning, for the check node update operation, the $CHK(S_m)$ is first computed where

$$\begin{aligned} CHK(S_m) &= \underset{l' \in L(m)}{CHK}(\oplus q_{m,l'}) \\ &= CHK(\dots CHK(CHK(q_{m,1} \oplus q_{m,2}) \oplus (CHK(q_{m,3} \oplus q_{m,4}))) \dots) \end{aligned} \quad (5.7)$$

and

$$\begin{aligned} CHK(a \oplus b) &= \text{sign}(a)\text{sign}(b) \times \text{Min}(|a|, |b|) + \ln \frac{1 + e^{|a+b|}}{1 + e^{|a-b|}} \\ &= \text{sign}(a)\text{sign}(b) \times \text{Min}(|a|, |b|) + g(a+b) - g(a-b) \end{aligned} \quad (5.8)$$

Then, for each of the check-to-variable message $r_{m,l}$ can be computed as

$$\begin{aligned} r_{m,l} &= CHK(S_m \ominus q_{m,l}) \\ &= \ln \frac{e^{q_{m,l} + CHK(S_m)} - 1}{e^{q_{m,l} - CHK(S_m)} - 1} - CHK(S_m) \\ &= h(q_{m,l} + CHK(S_m)) - h(q_{m,l} - CHK(S_m)) - CHK(S_m) \end{aligned} \quad (5.9)$$

The functions $g(x) = \ln(1 + e^{-|x|})$ and $h(x) = \ln|e^x - 1|$ can be implemented with look-up-table (LUT) operations. Figure 5.9 and Figure 5.10 show the curves of the g-function and h-function, respectively.

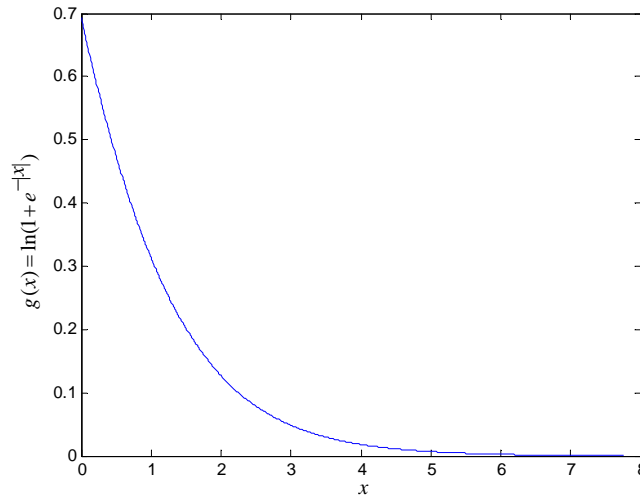


Figure 5.9 Function plot of $g(x) = \ln(1 + e^{-|x|})$

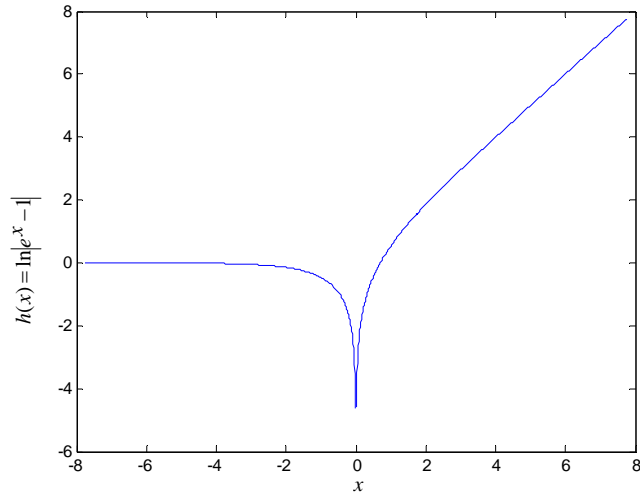


Figure 5.10 Function plot of $h(x) = \ln(e^x - 1)$

For the variable node update operation, each of the variable-to-check message $q_{m,l}$ can be computed as

$$q_l = \text{VAR}(\text{VAR}(r_{m,l}), L_l) = L_l + \sum_{m \in M(l)} r_{m,l} \quad (5.10)$$

and

$$q_{m,l} = q_l - r_{m,l}. \quad (5.11)$$

The corresponding architectures of the CNFUs and VNFUs for the min-sum based algorithm are shown in Figure 5.11 and Figure 5.12 respectively.

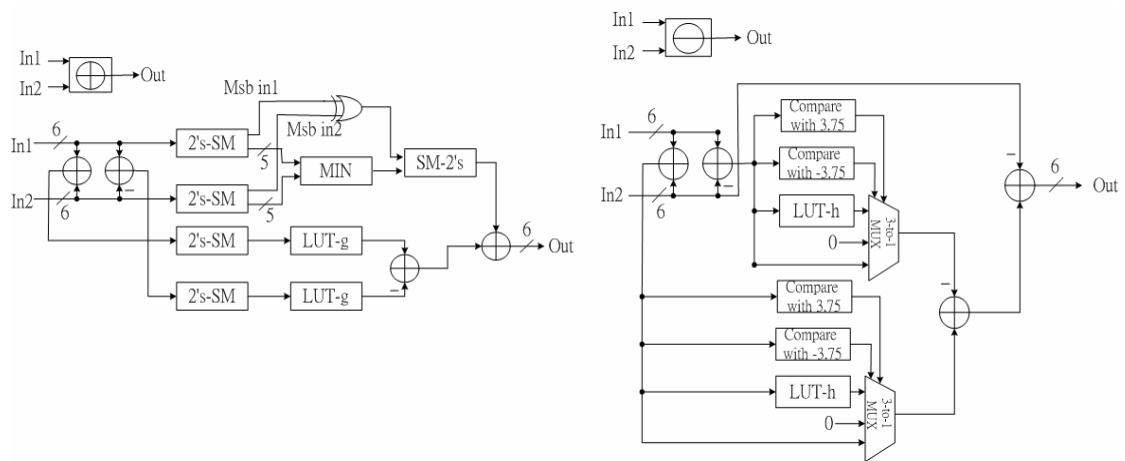
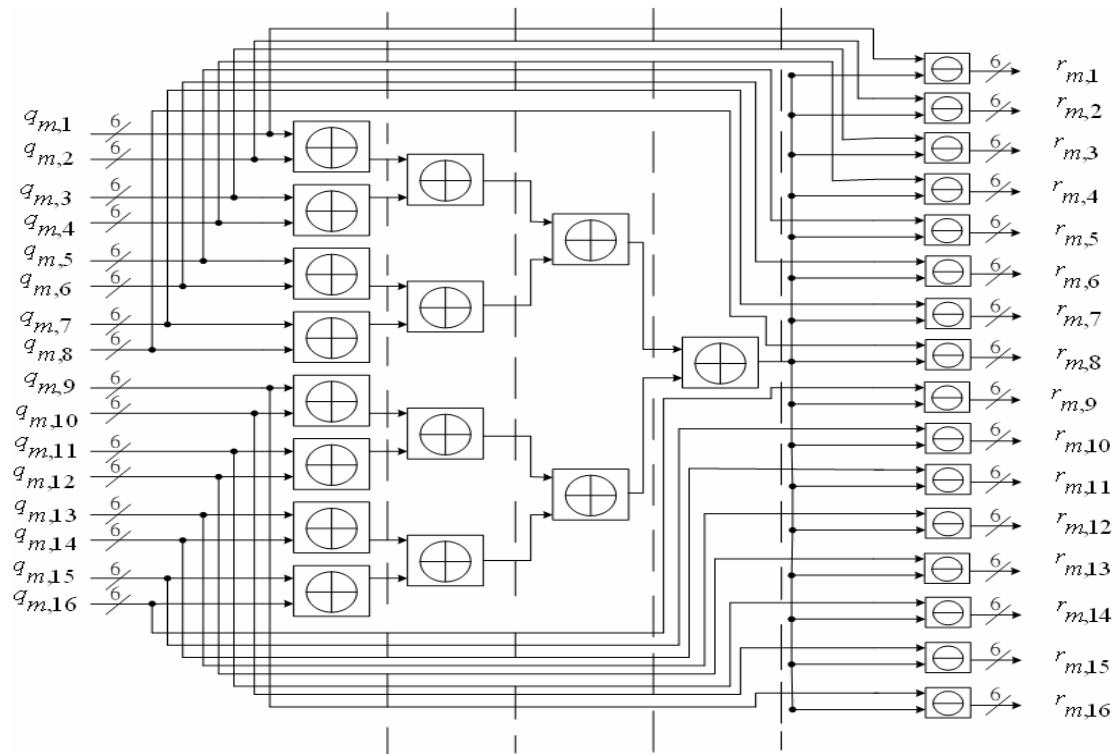


Figure 5.11 Architecture of check node function unit for the min-sum based algorithm

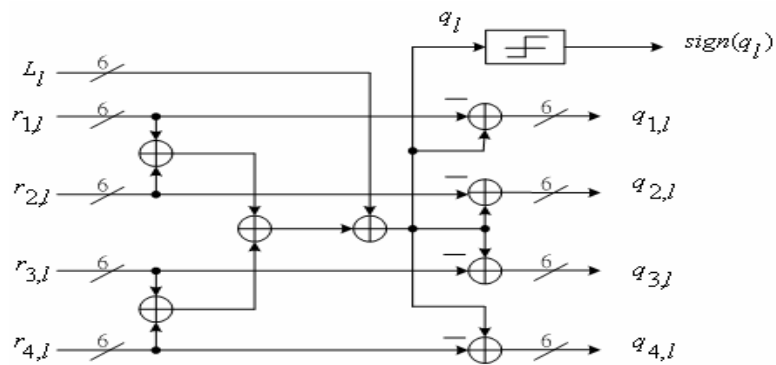


Figure 5.12 Architecture of a 4-input variable node function unit for the min-sum based algorithm

It should be noted that in Figure 5.11, the input range of the look-up-table for the h-function (LUT-h) is merely from -3.75 to 3.75 . The reason is that for the h-function where $h(x) = |e^x - 1|$, when x is larger than 3.75 , $h(x)$ equals to x or $h(x) = x$; and when x is smaller than -3.75 , the $h(x)$ will equal to 0 . This characteristic of the h-function can be clearly seen in Figure 5.10. So we can first compare the values $(in1 + in2)$ and $(in1 - in2)$, where $in1$ and $in2$ are the inputs of the core operation $CHK(in2 \ominus in1)$, to the values 3.75 and -3.75 respectively. Then, the compared results are used to form the control signals of a 3-to-1 multiplexer as shown in Figure 5.11 and determine the output signal of the 3-to-1 multiplexer. This shortening operation can reduce the number of entries of LUT-h to 32. Similarly, the dashed lines represent the possible positions for inserting the pipeline stages in Figure 5.11.

To compare the area, speed, latency, and power consumption of the architectures discussed in this section, we describe both architectures in VHDL, and afterwards simulate and synthesize it using the tools *SynopsisTM* and *PrimePower*. The process technology is UMC $0.18 \mu m$ process. Table 5.1 and Table 5.2 list the results of the reformulated and remapped architectures of CNFUs and VNFUs for the sum-product algorithm respectively. Table 5.3 lists the results for the architectures of CNFUs and VNFUs, based on the min-sum based algorithm. Furthermore, we give a summary on the comparisons of the different architectures in Table 5.4. It can be seen in Table 5.4 that the area and average power consumption of the architectures of CNFUs and VNFUs for the min-sum based algorithm are the worst when compared to the reformulated and remapped architecture of CNFUs and VNFUs on the sum-product algorithm. The reformulated and the remapped architecture of CNFUs and VNFUs for the sum-product algorithm are both tradeoffs between area and power

consumption. However, when comparing the reformulated architecture to the remapped architecture of CNFUs and VNFUs for the sum-product algorithm, the area of the remapped architecture is reduced by 20%; but the average power consumption of the remapped architecture is exceeded by 48%. We therefore select the reformulated architecture of CNFUs and VNFUs for the sum-product algorithm as the CNFUs and VNFUs in our decoder design.

Table 5.1 Area, speed, latency and power consumption of the reformulated CNFUs and VNFUs architectures for the sum-product algorithm

	16 input CNFU	3 input VNFU	4 input VNFU	5 input VNFU
Area (gate count)	5k	1.14k	1.6k	1.77k
Speed (MHz)	200	200	200	200
Latency(Cycles)	2	2	2	2
Power consumption (mW)	9.87	3.10	3.82	4.48

Table 5.2 Area, speed, latency and power consumption of the re-mapped CNFUs and VNFUs architectures for the sum-product algorithm

	32 input remapped hardware performing both CNFUs and VNFUs operations
Area (gate count)	18k

	Check node update operation	Variable node update operation
Speed (MHz)	200	200
Latency(Cycles)	3	3
Power consumption (mW)	30.82	43.74

Table 5.3 Area, speed, latency and power consumption of the CNFUs and VNFUs architecture for the min-sum base algorithm

	16 input CNFU	3 input VNFU	4 input VNFU	5 input VNFU
Area (gate count)	22k	0.74k	1.07k	1.61k
Speed (MHz)	200	200	200	200
Latency(Cycles)	5	1	1	1
Power consumption (mW)	23.2	2.23	3.53	4.33

Table 5.4 Summary of comparison the area, speed and power consumption of the different CNFUs and CNFUs architectures for the sum-product algorithm and the min-sum based algorithm

	Sum-product algorithm		Min-sum based algorithm
	Reformulated set	Remapped set	Conventional set

Area (gate count)	$5k+1.14k+1.6k*6+1.77k$ =22.51k	18k	$22k+0.74k+1.07k*6+1.61k$ =52.77k
Relative area	100%	80%	234%
Speed (MHz)	200	200	200
Total power consumption for check node update operation (mW)	$9.87*2$ =19.74	30.82	$23.2*2$ =46.4
Total power consumption for variable node update operation (mW)	$3.10+3.82*6+4.48$ =30.5	43.74	$2.33+3.53*6+4.33$ =27.74
Average Power consumption (mW)	$(19.74+30.5)/2$ =25.12	$(30.82+43.74)/2$ =37.28	$(46.4+27.74)/2$ =37.07
Relative average power	100%	148%	147%

consumption			
-------------	--	--	--

Having discussed the architecture of the CNFUs and VNFUs, we can further use the characteristics of the LDPC decoding, that is, it is inherently parallelizable and chooses a parallel factor of 10. That means that we can compute 20 rows simultaneously during the check node update operation, and 80 columns simultaneously during the variable node update operation as well. Now we set the number of the input bits of the whole decoder to 240 bits, which means that we can input 40 symbols in one clock cycle. Regarding a 960-symbol frame, it will take 24 cycles to complete the input operation. The number of the output bits is 10, so it will take 72 cycles to output the estimated data bits \hat{x} . Besides, we compute 20 parity-check equations in one cycle and it will take 12 cycles to finish all of the parity-check equations. Since the maximum iteration of the decoding procedure is 10 and the parallel factor is also 10, the total amount of cycles needed to complete the decoding procedure is $24 + 10 * (12 + 2) * 2 + 12 + 72 = 388$ cycles. According to our initial synthesis results, the clock frequency is 200MHz, thus the data decoding throughput is $200 * [960 * (3/4)] / 388 \approx 370$ Mbps. Regarding the power consumption of the whole decoder, by using the technique of gated clock, the VNFUs can be turned off when the CNFUs are busy during check node updates and vice versa during the variable node updates, one can reduce the total power consumption. Last of all, we give a summary of the whole decoder in Table 5.5. It is obvious that using gated clock can reduce 28% of the power consumption when compared to the case of without gated clock.

Table 5.5 Summary of the proposed LDPC decoder

	Proposed LDPC decoder without gated clock	Proposed LDPC decoder with gated clock
Area (gate count)	800k	800k
Speed (MHz)	200	200
Data throughput (Mbps)	370	370
Power consumption (mW)	770	550
Relative power consumption	100%	72%

According to the proposed irregular LDPC decoder, these results can be compared to other designs which are list in Table 5.6. As we can see in Table 5.6, using the semi-parallel architecture can make a more flexible design in hardware implementation.

Table 5.6 Comparison of LDPC decoders

	Proposed LDPC decoder	[18]	[17]
Code length	960	8088	1024
Code rate	3/4	1/2	1/2
Quantization bits	6	6	4

Architecture	Semi-parallel with parallel factor 10	Semi-parallel with parallel factor 24	Fully parallel
Process Technology (μm)	0.18	0.11	0.16
Clock rate (MHz)	200	212	64
Power (mW)	550		690
Area (gate count)	800k	742k	1750k
Throughput (Mbps)	370	188	500

Regarding the application of the proposed irregular LDPC decoder, it can be applied to the WLAN IEEE 802.11n standard. Table 5.6 shows the basic modulation coding scheme (MCS) set of TGnSync [19] proposal. It can be seen obviously that the decoder can support the data throughput requirement with all of the modulation method, when the transmission bandwidth is 20MHz and the code rate is 3/4.

Table 5.8 Basic MCS set of TGnSync proposal

Modulation	Code Rate	Data Rates* 20 MHz (Mbps) (1,2,3,4 spatial streams)	Data Rates* 40 MHz (Mbps) (1,2,3,4 spatial streams)
BPSK	1/2	6, 12, 18, 24	6 \ddagger , 13.5, 27, 45.5, 54
QPSK	1/2	12, 24, 36, 48	27, 54, 81, 108
QPSK	3/4	18, 35, 54, 72	40.5, 81, 121.5, 162
16 QAM	1/2	24, 48, 72, 96	54, 108, 162, 216
16 QAM	3/4	36, 72, 108, 144	81, 162, 243, 324
64 QAM	2/3	48, 96, 144, 192	108, 216, 324, 432
64 QAM	3/4	54, 108, 162, 216	121.5, 243, 364.5, 486
64 QAM	7/8	63, 126, 189, 252	141.7, 283.5, 425.2, 567
64 QAM	7/8 with 1/2 GI*	70, 140, 210, 280	157.5, 315, 472.5, 630

Chapter 6

Conclusion

From this work, we summarize that using the proposed structure of the LDPC codes can further improve the error correction performance when compared to the irregular quasi-cyclic codes. However, it is not expected that the proposed structure of the LDPC codes will outperform randomly constructed optimized irregular codes. The proposed structure of the LDPC codes has the advantage of a reduced encoding complexity and is suited for the VLSI implementation of the decoder.

Various quantization schemes for the received data and extrinsic message for the sum-product algorithm and the min-sum based algorithm of the irregular LDPC decoder were investigated and the optimal choice considering the tradeoff between the hardware complexity and the performance were discussed in this thesis. The overall fixed-point simulations show that the quantization scheme we have developed for the sum-product algorithm and min-sum based algorithm of the irregular LDPC decoder are effective in approximating the floating-point implementation and that using the sum-product algorithm is better than the min-sum based algorithm of the irregular LDPC decoder by about 0.1dB.

With the semi-parallel architecture and a parallel factor of 10, an irregular LDPC decoder has been implemented, of which the code rate is $3/4$, the code length is 960 bits, and the maximum number of decoding iterations is 10, respectively. The irregular

LDPC decoder can achieve the data decoding throughput of up to 370Mbps and the area is 800k gate counts using the UMC 0.18 μm ASIC process technology. Regarding the power consumption of the irregular LDPC decoder, by using the technique of gated clock one can reduce 28% of the total power consumption down to 550 mW.

The irregular LDPC decoder can support the data throughput requirement of the WLAN IEEE 802.11n standard when the transmission bandwidth is 20MHz and the code rate is 3/4. We believe that if we extend the parity-check matrix structure to code length 1920 and code rate 7/8, by increasing the parallel factor, we can implement a good LDPC decoder which can support the data throughput requirement with all of the different data rates.



References

- [1] R. G. Gallager, “Low-density parity-check codes,” Cambridge, MA: MIT Press, 1963.
- [2] D. J. C. Mackay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electron. Lett.*, Vol. 32, pp. 1645-1646, Aug. 1996.
- [3] T. J. Richardson and R. L. Urbanke, “Efficient encoding of low-density parity-check codes,” *IEEE Trans. Inform. Theory*, Vol. 47, pp. 638-656, Feb. 2001.
- [4] D. J. C. Mackay, S. T. Wilson, and M. C. Davey, “Comparison of constructions of irregular gallager codes,” *IEEE Trans. Comm.*, Vol. 47, pp. 1449-1454, Oct. 1999.
- [5] S. J. Johnson and S. R. Weller, “A family of irregular LDPC codes with low encoding complexity,” *IEEE Comm. Lett.*, Vol. 7, pp. 79-81, Feb. 2003.
- [6] M. C. Davey and D. J. C. Mackay, “Low-density parity-check codes over $GF(q)$,” *IEEE Comm. Lett.*, Vol. 2, pp. 165-167, Jun. 1998.
- [7] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inform. Theory*, Vol. 27, pp. 533-547, Sep. 1981.
- [8] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, “Practical loss-resilient codes,” *IEEE Trans. Inform. Theory*, Vol. 47, pp. 569-584, Feb. 2001.
- [9] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans.*

- Inform. Theory, Vol. 47, pp. 619-637, Feb. 2001.
- [10] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inform. Theory, Vol. 45, pp. 399-431, Mar. 1999.
- [11] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," IEEE Trans. Inform. Theory, Vol. 47, pp. 498-519, Feb. 2001.
- [12] H. Futaki and T. Ohtuski, "Low-density parity-check (LDPC) coded OFDM systems," IEEE VTS, Vol. 1, pp. 82-86, Fall. 2001.
- [13] X. Y. Hu, E. Eleftheriou, D. M. Arnold, and A. Dholakia, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," IEEE GLOBECOM'01, Vol. 02, pp. 1036-1036E, Nov. 2001.
- [14] I. V. Kozintsev. Software for low-density parity-check codes. [Online] Available at: <http://www.kozintsev.net/soft.html>.
- [15] A. Nayagam. Software for low-density parity-check codes. [Online] Available at: <http://arun-10.tripod.com/ldpc/ldpc.html>.
- [16] Z. Wang, Y. Chen, and K. K. Parhi, "Area efficient decoding of quasi-cyclic low density parity check codes," IEEE ICASSP'04, Vol. 5, pp. 49-52, May. 2004.
- [17] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," IEEE J. Solid-State Circuits, Vol. 37, pp. 404-412, Mar. 2002.
- [18] Y. Chen and D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," IEEE GLOBECOM'03, Vol. 3, pp. 113-117, Dec. 2003.

[19]TGnSync, “TGnSync Proposal,” [Online] Available at:
<http://www.tgnsync.org/home>.



自 傳

朱元志，1981年8月1日出生，高雄市人。2003年自國立東華大學電機工程學系畢業，隨即進入國立交通大學電子研究所攻讀碩士學位。研究興趣為通訊系統與數位信號處理，碩士論文題目為低密度對偶檢查碼結構之改進以及其解碼器之超大型積體電路實現。

