國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

適用於IEEE 802.16e及802.11n標準之可配置低密度同位元檢查碼解碼器設計與實現

Design and Implementation of a Configurable LDPC Decoder for IEEE 802.16e and 802.11n

研究生：劉士賢

指導教授：劉志尉 博士

中華民國九十六年九月

適用於 IEEE802.16e 及 802.11n 標準之可配置低密度同位元檢查碼解碼器設計與實現發

Design and Implementation of a Configurable LDPC Decoder for IEEE 802.16e and 802.11n

研究生：劉士賢　　Student: Shih-Hsien Liu
指導教授：劉志尉 博士　　Advisor: Dr. Chih-Wei Liu

國立交通大學
電子工程學系 電子研究所碩士班
碩士論文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Electronics Engineering

September 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年九月

適用於 IEEE 802.16e 及 802.11n 標準之
可配置低密度同位元檢查碼解碼器設計與實現

研究生：劉士賢　　　　　　指導教授：劉志尉 博士

國立交通大學
電子工程學系 電子研究所

摘要

這篇論文提出一個支援全模式適用於 IEEE 802.16e 及 802.11n 標準之可配置低密度同位元檢查碼解碼器。一個採用列跟新訊息傳遞演算法的部份平行化架構被設計來達到高速傳輸速率及解碼能力。我們討論一些主題包括，解碼演算法的分析及最佳化，架構設計及實現，提早中斷機制(early termination)，多區塊解碼技巧(multi-codeword decoding)，排程以及後段佈局的模擬數據(post-layout simulation)

低密度同位元檢查碼(LDPC code)是最好的更正碼其中之一。最近因它良好的解碼能力及稀疏矩陣的特性，引起許多研究興趣。由於它的高度平行化特性，使其容易設計及實現高速需求的架構。一些高速的通訊系統如建立在 IEEE 802.16e 標準的 WiMAX 和 802.11n 標準的 WiFi 均採用低密度同位元檢查碼來提供通道更正的能力。我們設計一個可配置的架構適用於 802.16e 跟 802.11n 的所有的碼率及碼長。

一個核心面積(core size)為 2.14×2.14 mm^2解碼器被實現在台積電 0.13μm 1P8M CMOS的製程下。在 802.16e中，10 次迴圈下，它具有最高傳輸速率 590 Mb/s 且平均功率消耗是 451 mW。而在 802.11n中，最高速率是 506 Mb/s而功率消耗是 436 mW。藉著降低操作頻率到 66 MHz (333 MHz的五分之一)，以符合 802.16e 最低傳輸速率的要求，30 Mb/s，傳輸速率降為 42.6~118 Mb/s根據不同的碼長及碼率。平均功率消耗則被降至 91 mW針對 802.16e中，碼率 5/6，碼長 2304 位元測量下。

Design and Implementation of a Configurable LDPC Decoder for IEEE 802.16e and 802.11n

Student: Shih-Hsien Liu Advisor: Dr. Chih-Wei Liu

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

This thesis presents a fully compliant, configurable LDPC decoder for 802.16e and 802.11n. A partially parallel architecture with scheduled row-update message passing algorithm is designed to archive high throughput and decoding performances. We discuss some topics, decoding algorithm analysis and optimization, architecture design and implementation, early termination, multi-codeword decoding technique, scheduling, and post-layout simulation results.

LDPC code is one of best error correction codes. Recently, it engages much research interest because of its sparse matrix and well decoding performance. Due to its high parallelizable algorithm, the high speed architecture is easy to be designed and implemented. Some high speed communication systems, WiMAX based on IEEE 802.16e standard and WiFi based on 802.11n standard both take LDPC codes to provide channel correction ability. We design a configurable architecture for full code rates and codeword lengths in 802.11n and 802.16e.

The decoder with a core size 2.14×2.14 mm^2 is implemented in TSMC 0.13 μm 1P8M CMOS technology. It has a peak throughput of 590 Mb/s and power dissipation of 451 mW with 10 iterations for 802.16e and a throughput of 506 Mb/s with 436 mW power consumption for 802.11n. By slowing down operating frequency to 66 MHz (one-fifth of 333 MHz) to meet required minimum throughput, 30 Mb/s for 802.16e, its throughput is 42.6~118 Mb/s for different code rates and codeword lengths. Average power consumption is lower to 91 mW for code rate 5/6, codeword length 2304 bits in 802.16e.

誌 謝

研究生涯兩年來過的很快，轉眼即逝，苦澀酸甜隨著論文有著結束，但卻在記憶裡迴盪不去，兩年來受過不少人的幫助，點滴在心頭。

首先，感謝劉志尉老師。老師的指導使我在專業知識及研究態度上，更加成熟與進步。還有 DSP-LAB 的學長姐們，泰吉學長、彥欽學姐、歐、阿圳、小A、郭等博班學長們的指導、同屆的慶至、卓毅、志宏、heko、王炳、大家彼此相互的學習與鼓勵、當然還有碩一學弟們的平日的幫忙。

還要感謝 OCEAN group 的成員們，除了張錫嘉老師的指導外，還有建青學長、彥欽學姊、志豪學長、阿龍、大頭、紹維等博班學長姐的無私的給予協助和精神上的鼓勵。同屆的名威、義閔、大嘉、佳瑋、國光、修齊、俊閔、義凱以及其他的學弟妹們、兩年來的相處給的很多，這份成果和回憶我會謹記在心。

當然還有口試委員們的指導，周世傑老師，陳紹基老師，以及吳安宇老師，感謝老師們撥空指導，使得論文的豐富性更具完整。

最後，還有我親愛的家人們，爺爺、奶奶、爸、媽以及弟弟，謝謝你們一路上的支持跟鼓勵，常常在外一直沒好好的陪你們。每逢佳節倍思親，現在的我很能體會這感受，這些年來細心的栽培和鼓勵，我獻上最深的感謝及祝福。

謹將此篇論文獻給所有曾支持我、協助我的人，衷心的感謝並祝福你們。

士賢
謹誌於 新竹
2007 九月

CONTENTS

ABSTRAC (CHINESE) ..... **I**

ABSTRAC (ENGLISH) ..... **II**

ACKNOWLEDGEMENT ..... **III**

CONTENTS ..... **IV**

LIST OF TABLES ..... **VI**

LIST OF FIGURES ..... **VII**

CHAPTER 1 INTRODUCTION **1**

1.1 Overview of Wireless Communication System 1
1.2 Motivation 2
1.3 Thesis Organization 3

CHAPTER 2 LDPC IN 802.16E AND 802.11N **6**

2.1 Quasi-Cyclic Matrix in 802.11n and 802.16e 7
  2.1.1 Parameters for 802.11n 9
  2.1.2 Parameters for 802.16e 10
2.2 LDPC Encoder Method 12

CHAPTER 3 LOW DENSITY PARITY CHECK CODE **16**

3.1 Concept of Low-density Parity Check Codes 17
  3.1.1 Message Passing Algorithm 18
  3.1.2 LDPC Decoding Algorithm 25
3.2 Optimized Approach for Hardware Implementation 34
  3.2.1 Min-Sum Algorithm 35
  3.2.2 Row-Update Message Passing Algorithm 38
  3.2.3 Trade-off between Decoding Algorithms and Code Structures 42
3.3 Design Considerations 43
  3.3.1 Normalization Factor 43
  3.3.2 Bit Width for Hardware Cost 44
  3.3.3 Number of Iteration 46
3.4 Implementation Issue 47

CHAPTER 4 ARCHITECTURE DESIGN AND IMPLEMENTATION **49**

4.1 Architecture Design 50

4.1.1 Decoding Flow..50
4.1.2 Overall Architecture Design..52
4.2 IMPLEMENTATION 69
4.2.1 IMPLEMENTATION RESULTS 71

CHAPTER 5 SUMMARY 76

REFERENCE 78

List of Tables

Table 2-1：Parameters of LDPC code for 802.11n 10
Table 2-2：Parameters of LDPC code for WiMAN 802.16e 11
Table 2-3：Steps of encoding method 14
Table 3-1：Parameters summary 47
Table 4-1：Comparison with design [21] for area 58
Table 4-2：Throughput enhancement with scheduling for 802.16e 68
Table 4-3：Summary of RAMs and ROMs 69
Table 4-4：Synthesis results 71
Table 4-5：Summary of memory 72
Table 4-6：Summary specification of LDPC decoder 73
Table 4-7：Average power consumption in low power mode at 66 MHz, *1 74
Table 4-8：Comparison of LDPC code decoder 75

List of Figures

Figure 1-1：Example of QC parity check matrix structure for code rate 3/4 3
Figure 2-1：Example of parity check matrix for code rate 5/6, Z_F=54 in 802.11n 9
Figure 2-2：Elements of Parity Check Matrix 12
Figure 2-3：Example of 8-by-8 T^{-1} matrix 13
Figure 2-4：The block diagram of encoder method 14
Figure 3-1：Example of bipartite graph for equation (3.1) 18
Figure 3-2：Example of normal graph 20
Figure 3-3：Graph presentation of the intrinsic and extrinsic probabilities 20
Figure 3-4：Graph presentation of message passing between two vertices 22
Figure 3-5：Bipartite graph of matrix (3.23) 25
Figure 3-6：Transmission probability for BSC 27
Figure 3-7：Merged graph of check node and bit node with L+1 degree 28
Figure 3-8：Function graph of $\psi(x)$ 37
Figure 3-9：Process of BP algorithm 39
Figure 3-10：Process of row-update message passing algorithm 39
Figure 3-11：BP v.s RMP algorithm in 802.11n with codeword length 648 bits 40
Figure 3-12：BP v.s RMP algorithm in 802.11n with codeword length 1944 bits 41
Figure 3-13：BP v.s RMP algorithm in 802.16e codeword length 576 bits 41
Figure 3-14：BP v.s RMP algorithm in 802.16e, codeword length 2304 bits 42
Figure 3-15：Simulation for normalization factor simulation 44
Figure 3-16：Fixed-point simulation with fixed 6 bits integer 45
Figure 3-17：Fixed-point simulation for integer part 45
Figure 3-18：Simulation for iteration number, 5,8,10,15,20 46
Figure 4-1：Decoding Flow Chart 51
Figure 4-2：Decoding process diagram for $M \times N$ parity check matrix 51
Figure 4-3：Overall architecture 53
Figure 4-4：Data path from "in_buffer" to "MS" 54
Figure 4-5：3-by-6 parity matrix with Z_F=6 55
Figure 4-6：6-by-6 identity matrix with shift amounts 0, 2, respectively 55
Figure 4-7：Register arrangement for shift amounts 0, 1 with shift size 6 56
Figure 4-8：Example of shift size Z_F smaller than maximum register size L 56
Figure 4-9：Permutation design with head and tail pointers 57
Figure 4-10：Permutation for maximum size 8 and shift size 6 and shift amount 1 57
Figure 4-11：Three levels of logic barrel shifter for 96 data 58

Figure 4-12：Data path from MS to CHK......59
Figure 4-13：Architecture for processing units (CHK/VAR)......60
Figure 4-14：Block diagram of “Beta_ram” memory and controls......61
Figure 4-15：Block diagram for multi-codeword decoding technique......62
Figure 4-16：Point based permutation for multi-codeword......63
Figure 4-17：Throughput for 802.11n, full code rates at 333 MHz......63
Figure 4-18：Throughput for 802.16e, length 576, 1152, 2304 at 333 MHz......64
Figure 4-19：Hard decision based early termination v.s parity check constraints......65
Figure 4-20：Process of dynamic early termination......66
Figure 4-21：Example of scheduling......67
Figure 4-22：Overlapped v.s non-overlapped......68
Figure 4-23：Design flow of proposed LDPC decoder......70
Figure 4-24：Photo of LDPC decoder......73

Chapter 1 Introduction

1.1 Overview of Wireless Communication System

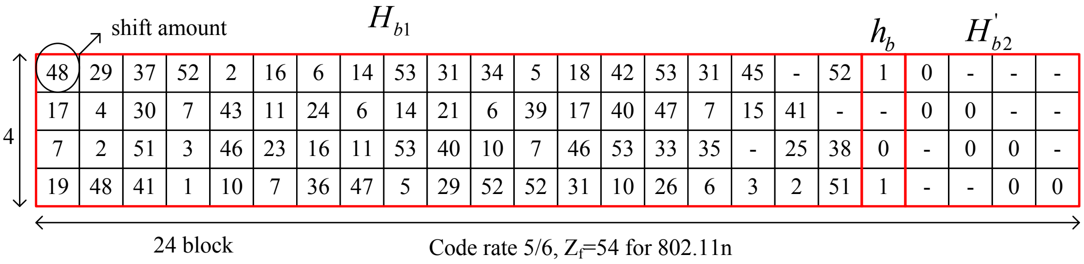
Recently, the Worldwide Interoperability for **M**icrowave **A**ccess **(WiMAX)** and **Wi**reless **Fi**delity **(WiFi)** have been received wide attention in wireless broadband standard. They are proposed to provide end-users to travel throughout a hot zone cell without losing connectivity. The WiMAX standard group is collectively called IEEE 802.16. The standard for fixed WiMAX, i.e. the stationary devices such as home or office PCs, is 802.16-2004, which offers data transfer rate of up to 75 Mbps (megabits per second) over distances of up to 30 miles (4~6 miles is typical). The advanced standard, 802.16e [1], established specifications for mobile WiMAX, i.e. laptops or cell phones, offers similar speeds over slightly shorter distances, typically 1~3 miles.

WiFi, on the other hand, adheres to the IEEE 802.11 standard, which provides close-range, wireless broadband access in fixed environment. This standard went through several waves of development before arriving at the current leader, 802.11g, which supports speeds of as much as 54 Mbps over distances of up to 300 feet. In 2003, the IEEE responded to growing demand for increased wireless performance by authorizing the creation of IEEE 802.11 Task Group. They developed and modified the 802.11 specification, called 802.11n [2], to support a minimum speed of 100Mbps with MIMO technology. Developers asset the final specification may support transfer speeds exceeding 200Mbps over longer distances than 802.11 currently supports. 802.11n is backward-compatible with earlier standards: 802.11a, 802.11b, and 802.11g.

As a fixed broadband access technology, WiFi has its weakness. The user can only use the technology within the confines of a 300 feet radius and, hence, the level of mobility is limited. For practical purposes, most observers have considered WiMAX to be an outdoor technology. A combined scenario of WiMAX (for the building) and Wi-Fi (for the interior) looks like a viable solution. Combining the ability to use both kinds of networks on a single device allows consumers to take advantage of the best each has to offer.

1.2 Motivation

LDPC code is first introduced by Gallager [6] in 1962. It can provide better error correction capacities than other channel codes. In recent years, many papers discuss the implementation architecture and structured parity check matrix. The defining LDPC codes in **IEEE 802.11n** and **802.16e** are classical Quasi-Cyclic (QC) structured parity check matrix. Figure1-1 illustrates an example of Quasi-Cyclic matrix for code rate 3/4. Each element in the parity check matrix denotes a shift amount which can be expanded to an identity circular right shift matrix. The properties of Quasi-Cyclic matrix will be presented in chapter 2.



48	29	37	52	2	16	6	14	53	31	34	5	18	42	53	31	45	-	52	1	0	-	-	-
17	4	30	7	43	11	24	6	14	21	6	39	17	40	47	7	15	41	-	-	0	0	-	-
7	2	51	3	46	23	16	11	53	40	10	7	46	53	33	35	-	25	38	0	-	0	0	-
19	48	41	1	10	7	36	47	5	29	52	52	31	10	26	6	3	2	51	1	-	-	0	0

Figure 1-1：Example of QC parity check matrix structure for code rate 5/6

Because 802.16e and 802.11n have similar technology modulation, OFDM system and their application properties of WiFi for wireless data transmission on local distance and WiMAN for mobility and portability, a combined scenario looks like a variable solution. It provides combinational properties to design a configurable data-path for the similar code structures of LDPC codes in 802.11n and 802.16e. In this thesis, a configurable LDPC decoder is proposed for multi-standard. The architecture adopts partially parallel decoding for QC LDPC codes and supports 19 modes in 802.16e and 3 modes in 802.11n. A high throughput of 590 Mb/s and power dissipation of 451-mW for 802.16e and 506 Mb/s, 436-mW for 802.11n with a core size 4.58 mm^2 are estimated in post-layout simulation/***PrimePower*** at 333 MHz. A multi-codeword decoding technique for preserving hardware utilization and early termination to save power are considered. Scheduling issue and **R**ow-update **M**essage **P**assing algorithm **(RMP)** [17] are applied to accelerate the speed of convergence. This work is the first published LDPC decoder for multi-standard 802.11n and 802.16e. The detail discussion and proposed architecture will be given in the following chapters.

1.3 Thesis Organization

The rest of this thesis is organized as follows. The definition of parity check matrix of LDPC in 802.16e and in 802.11n is presented in chapter 2. Several efficient LDPC decoding algorithms including min-sum approximation and other implementation issues are briefly described in Chapter 3. It also shows the simulation results of c-codes and discusses the related performance comparisons. Among them, scheduled row-update message passing algorithm is suggested. Chapter 4 introduces the proposed architecture including the detail functional implementation and memory arrangements. We summarize implementation results in chapter 4. Finally summary will be given in chapter 5.

1896

Chapter 2 LDPC in 802.16e and 802.11n

In this chapter, we introduce the specification of LDPC codes in 802.11n [2] and 802.16e [1]. LDPC code, a linear codeword code can be defined by a parity check matrix. The parity check matrix based on the methods of construction, can be generally classified into two categories: 1) random codes generally generated by computer search under certain design constraints, e.g. the girth and degree distributions [7, 9, 10]; 2) structured codes constructed by algebraic geometry and combinatorial method [31, 32]. One class of structured LDPC codes that allows low complexity encoding [33] is the quasi-cyclic (QC) LDPC codes. Well designed QC-LDPC codes have been shown to perform as well as regular or irregular computer-generated random LDPC codes [32]. QC-LDPC codes, moreover, have advantages in VLSI implementations of decoders since the cyclic symmetry results in simple regular wiring and modular structure.

2.1 Quasi-Cyclic Matrix in 802.11n and 802.16e

QC-LDPC codes, one class of linear block codes, are specified by sparse circular parity-check matrices. An $m_b \times n_b$ matrix is said to be in circular form if it is generated by an array of $\{\boldsymbol{P}_{i,j}\}_{0 \le i \le m_b - 1; 0 \le j \le n_b - 1}$ circulant of same size, where a circularnt is a square matrix in which each row is the *cyclic shift* (one place to the right) of the row above it, and the first row is the cyclic shift of the last row. The Z_f is the abbreviation of Z-factor. Without loss of generality, for sparse matrices, the circular $\boldsymbol{P}_{i,j}$ is assumed to be either a p(i,j,Z_f) cyclically shifted identity matrix or a zero matrix of size $Z_f \times Z_f$ $m \times n$ where p(i,j,Z_f) is non-negative integer and Z_f is positive integer related to parameter *f*. Then, $\boldsymbol{P}_{i,j}$ is the $Z_f \times Z_f$ identity matrix if p(i,j,Z_f)=0; and, for simplicity, we define p(i,j,Z_f)$\equiv$-1, if $\boldsymbol{P}_{i,j}$ is a zero matrix. Note that, either the zero matrix or the cyclically shifted idenrity matrix is a special circulant. Consequently, the QC-LDPC code with rate $\frac{n_b - m_b}{n_b}$ and length of $n_b \times Z_f$ can be defined by the following sparse parity check matrix,

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{P}_{0,0} & \boldsymbol{P}_{0,1} & \boldsymbol{P}_{0,2} & \cdots & \boldsymbol{P}_{0,n_b-2} & \boldsymbol{P}_{0,n_b-1} \\ \boldsymbol{P}_{1,0} & \boldsymbol{P}_{1,1} & \boldsymbol{P}_{1,2} & \cdots & \boldsymbol{P}_{1,n_b-2} & \boldsymbol{P}_{1,n_b-1} \\ \boldsymbol{P}_{2,0} & \boldsymbol{P}_{2,1} & \boldsymbol{P}_{2,2} & \cdots & \boldsymbol{P}_{2,n_b-2} & \boldsymbol{P}_{2,n_b-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{P}_{m_b-1,0} & \boldsymbol{P}_{m_b-1,1} & \boldsymbol{P}_{m_b-1,2} & \cdots & \boldsymbol{P}_{m_b-1,n_b-2} & \boldsymbol{P}_{m_b-1,n_b-1} \end{bmatrix} = \boldsymbol{P}^{H_b} \qquad (2.1)$$

where n_b=24 in 802.11n and 802.16e and m_b is a variable according to different code rates.

In 802.11n and 802.16e, the m×n parity check matrix $\boldsymbol{H}$ is expanded from a binary base matrix $\boldsymbol{H_b}$ of size of m_b-by-n_b, where $m = m_b \times Z_f$, $n = n_b \times Z_f$. Because each circular matrix is specified by a single circular right shift, the binary base matrix information and permutation replacement information can be combined into a single compact model matrix $\boldsymbol{H}_{bm}$. The matrix $\boldsymbol{H}_{bm}$ is the same size as the binary matrix $\boldsymbol{H}_b$, with each binary entry (i,j) of the base matrix $\boldsymbol{H}_b$ replaced to create the model matrix $\boldsymbol{H}_{bm}$. Each element $\boldsymbol{P}_{i,j}$ in $\boldsymbol{H}_b$ is replaced by a denote a circular right shift matrix described in last paragraph. The model matrix $\boldsymbol{H}_{bm}$ can the be directly expanded to $\boldsymbol{H}$.

The matrix $\boldsymbol{H}_b$ is partitioned into two sections, where $\boldsymbol{H}_{b1}$ corresponds to the systematic bits and $\boldsymbol{H}_{b2}$ corresponds to the parity check bits, such that $\boldsymbol{H}_b = [(\boldsymbol{H}_{b1})_{n_b \times k_b} \mid (\boldsymbol{H}_{b2})_{n_b \times k_b}]$. Section $\boldsymbol{H}_{b2}$ is further partitioned into two sections, where vector $\boldsymbol{h}_b$ has odd weight, and $\boldsymbol{H}'_{b2}$ has a dual-diagonal structure with matrix elements at row i, column j equal to 1 for $i = j$, 1 $for\ i = j+1$ and 0 elsewhere. The base matrix has $h_b(0) = 1$, $h_b(m_{b-1}) = 1$, and a third value $h_b(j) = 1,\ 0 < j < (m_{b-1})$ equal to 1. Equation (2.2) shows the definition of $\boldsymbol{H}_{b2}$.

1896

$$\boldsymbol{H}_{b2} = [\boldsymbol{h}_b \mid \boldsymbol{H}'_{b2}] = \left[\begin{array}{c|ccccc} h_b(0) & 0 & & & & -1 \\ h_b(1) & 0 & \cdots & & & \\ h_b(2) & & \cdots & O & & \\ \cdots & & & O & \cdots & \\ h_b(m_b - 1) & -1 & & & \cdots & 0 \end{array}\right] \qquad (2.2)$$

In particular, the non-zero sub-matrices are circularly right shifted by a particular circular shift value. Each 1 in $\boldsymbol{H}'_{b2}$ is assigned a shift amount of 0, and is replaced by a $Z_f \times Z_f$ identity matrix when expanding to $\boldsymbol{H}$. The two located at the top and the bottom of h_b are assigned equal shift amounts, and the third 1 in the middle of h_b is given an unpaired shift amount.

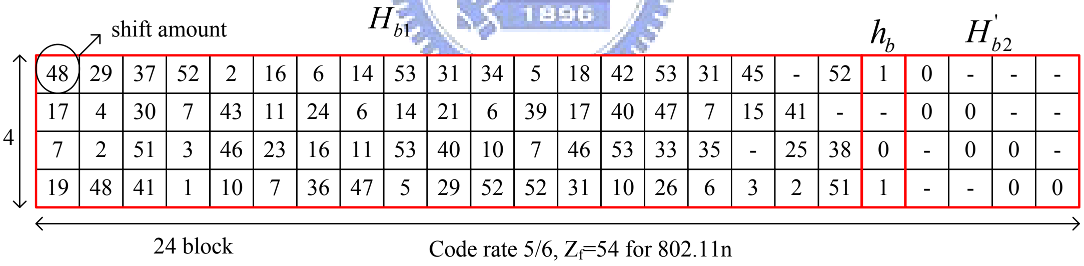
Following we describe the shift amount p(i,j,Z_f) for the circulant $\boldsymbol{P}_{i,j}$ in (2.1) according to different code rates in 802.11n and 802.16e, respectively: 1) For rate 1/2A, 2/3B, 3/4(A,B), and 5/6 in 802.16e, we have

$$p(i,j,Z_f)=\begin{cases} p(i,j) & ;\text{if } p(i,j)\le 0 \\ \left\lfloor \dfrac{p(i,j)Z_f}{96} \right\rfloor & ;\text{otherwise} \end{cases} \tag{2.3}$$

2) For rate 2/3A in 802.16e and rate 1/2, 2/3, 3/4 and 5/6 in 802.11n, we have

$$p(i,j,Z_f)=\begin{cases} p(i,j) & ;\text{if } p(i,j)\le 0 \\ p(i,j) \bmod Z_f & ;\text{otherwise} \end{cases} \tag{2.4}$$

where $\lfloor x \rfloor$ in (2.3) is the floor function that returns the largest integer less than or equal to x, and *p(i,j)* is either −1 or non-negative integer, given by the specification, which is used to determine the shift amounts for all other lengths of the same rate. And, in 802.11n, $Z_f = 81-27f$, $f = 0$, 1, and 2; while in 802.16e, $Z_f=81-4f$, $f = 0$, 1,., 18. To be summarized, there are 19 modes for 802.16e and 3 modes for 802.11n, respectively (totally 22 modes). Figure 2-1 shows an example of parity check matrix for code rate 5/6 and Z_f 54, defined in 802.11n.



48	29	37	52	2	16	6	14	53	31	34	5	18	42	53	31	45	-	52	1	0	-	-	-
17	4	30	7	43	11	24	6	14	21	6	39	17	40	47	7	15	41	-	-	0	0	-	-
7	2	51	3	46	23	16	11	53	40	10	7	46	53	33	35	-	25	38	0	-	0	0	-
19	48	41	1	10	7	36	47	5	29	52	52	31	10	26	6	3	2	51	1	-	-	0	0

Figure 2-1：Example of parity check matrix for code rate 5/6, Z_f=54 in 802.11n

2.1.1 Parameters for 802.11n

In 802.11n, there are 3 types of codeword length, 648, 1296 and 1944 bits. The lengths are multiple of 24. The required minimal throughput is 300 Mb/s. The $\boldsymbol{Z_f}$ is defined as $\dfrac{\text{codeword length}}{24}$, corresponding to different codeword lengths. So there are 3 types of Z_f, 27, 54, 81 corresponding to codeword lengths, 648, 1296 and 1944 bits. There are 4 types of code rate, including 1/2, 2/3, 3/4, 5/6. Table 2-1 shows the 3 types of codeword length and their corresponding parameters. The “k” presents the information bit length. The “n” presents the total transmission codeword length.

Table 2-1：Parameters of LDPC code for 802.11n

n(bits)	n(bytes)	Z_f	k (bytes) (information bits)			
			Rate 1/2	Rate 2/3	Rate 3/4	Rate 5/6
648	81	27	40.5	54	60.75	67.5
1296	162	54	81	108	121.5	135
1944	243	81	121.5	162	182.25	202.5

2.1.2 Parameters for 802.16e

In WiMAN 802.16e, there are 19 types of codeword length from 576 to 2304 bits. The required minimal throughput is 30 Mb/s. The Z_f varies form 24 to 96 with increment of 4. Totally there are 19 Z-factor from 24 to 96 corresponding to different codeword lengths. There are 4 types of code rate, including 1/2, 2/3, 3/4, 5/6 and define 6 types of parity check matrix, 1/2, 2/3A, 2/3B, 3/4A, 3/4B, and 5/6. Table 2-2 shows the summary of 19 types of codeword length and their corresponding parameters. The “n” presents the total transmission codeword length. The data throughput requires a minimal 30 Mb/s.

Table 2-2：Parameters of LDPC code for WiMAN 802.16e

n(bits)	n(bytes)	Z_f	k(bytes) (information bits)			
			Rate 1/2	Rate 2/3 (A,B)	Rate 3/4 (A,B)	Rate 5/6
576	72	24	36	48	54	60
672	84	28	42	56	63	70
768	96	32	48	64	72	80
864	108	36	54	72	81	90
960	120	40	60	80	90	100
1056	132	44	66	88	99	110
1152	144	48	72	96	108	120
1248	156	52	78	104	117	130
1344	168	56	84	112	126	140
1440	180	60	90	120	135	150
1536	192	64	96	128	144	160
1632	204	68	102	136	153	170
1728	216	72	108	144	162	180
1824	228	76	114	152	171	190
1920	240	80	120	160	180	200
2016	252	84	126	168	189	210
2112	264	88	132	176	198	220

2208	276	92	138	184	207	230
2304	288	96	144	192	216	240

2.2 LDPC Encoder Method

The general method of encoding is quite complex by determining a generator matrix $\boldsymbol{G}$ form $\boldsymbol{H}$ such that $\boldsymbol{GH}^T$=0. LDPC encoder in 802.11n and 802.16e provides a memory efficient method to encode codeword instead of generator matrix $\boldsymbol{G}$, $\boldsymbol{G}\times\boldsymbol{x}=\boldsymbol{v}$ for the properties of defined parity check matrices. Because the parity check matrix $\boldsymbol{H}$ is an approximate lower triangular form, so the m×n matrix can be written in the form,

$$\boldsymbol{H}=\begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \tag{2.5}$$

where A is $(m-Z_f)\times k$, B is $(m-Z_f)\times Z_f$, T is $(m-Z_f)\times(m-Z_f)$, C is $Z_f\times k$, D is $Z_f\times Z_f$, and E is $Z_f\times(m-Z_f)$. And k is the length of information x. The $\begin{pmatrix} B \\ D \end{pmatrix}$ and D correspond to the expanded $\boldsymbol{h}_b$ and $\boldsymbol{h}_b(m_b-1)$, respectively. Figure 2-2 shows the summary of notations above.

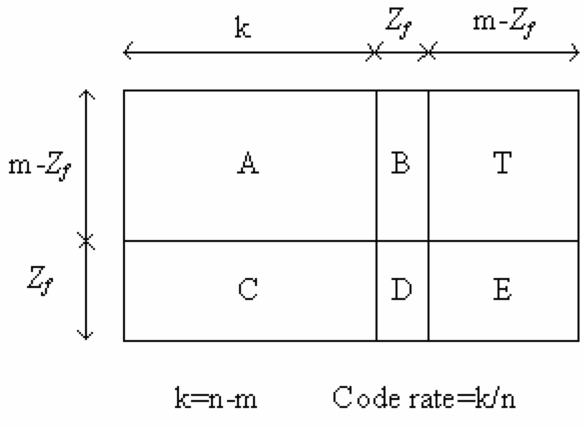


Figure 2-2：Decomposition of Parity Check Matrix

Let codeword $\boldsymbol{v} = (\boldsymbol{x}, \boldsymbol{p}_1, \boldsymbol{p}_2)$, $\boldsymbol{x}$ is the systematic information, $\boldsymbol{p}_1, \boldsymbol{p}_2$ are the parity parts from encoder. $\boldsymbol{p}_1$ has length Z_f and $\boldsymbol{p}_2$ has length $m - Z_f$. With the definition, $\boldsymbol{v}$ must satisfy the condition $\boldsymbol{H} \times \boldsymbol{v}^T = 0$. We replace $\boldsymbol{v} = (\boldsymbol{x}, \boldsymbol{p}_1, \boldsymbol{p}_2)$ into the equation $\boldsymbol{H} \times \boldsymbol{v}^T = 0$. Then we can derive the equalities.

$$\begin{cases} A\boldsymbol{x}^T + B\boldsymbol{p}_1{}^T + T\boldsymbol{p}_2{}^T = 0 & \text{--- (1)} \\ C\boldsymbol{x}^T + D\boldsymbol{p}_1{}^T + E\boldsymbol{p}_2{}^T = 0 & \text{---(2)} \end{cases} \tag{2.6}$$

From equation (1) of (2.6), we can rewrite $\boldsymbol{p}_2{}^T$ as $\boldsymbol{p}_2{}^T = T^{-1}(A\boldsymbol{x}^T + B\boldsymbol{p}_1{}^T)$ and replace $\boldsymbol{p}_2{}^T$ into (2) of (2.6). It can derive as

$$(ET^{-1}A + C)\boldsymbol{x}^T + (ET^{-1}B + D)\boldsymbol{p}_1{}^T = 0 \tag{2.7}$$

Define $\phi = (ET^{-1}A + C)$ and with the parity check matrix as indicated $\phi = I$, I denotes identity matrix. $\phi = (ET^{-1}A + C) = I$ is the property of parity check matrix with the definition of $\boldsymbol{h}_b$.

Continuing with the derivation, $\boldsymbol{p}_1{}^T$ can be rewritten as

$$\boldsymbol{p}_1{}^T = (ET^{-1}A + C)\boldsymbol{x}^T \tag{2.8}$$

Because the matrix T is a dual diagonal matrix, lower triangular matrix is the characteristic of T^{-1} shown as the example in Figure 2-3.

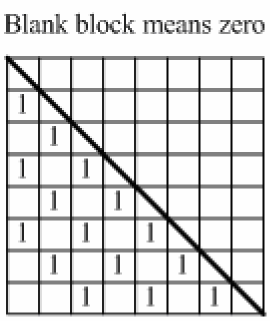


Figure 2-3：Example of 8-by-8 T^1 matrix

By the equation $\boldsymbol{p}_2^T = T^{-1}(A\boldsymbol{x}^T + B\boldsymbol{p}_1^T)$ and (2.7), we can derive the values, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. Figure 2-4 shows the block diagram of encoder and Table 2-5 summarize all the encoding steps.

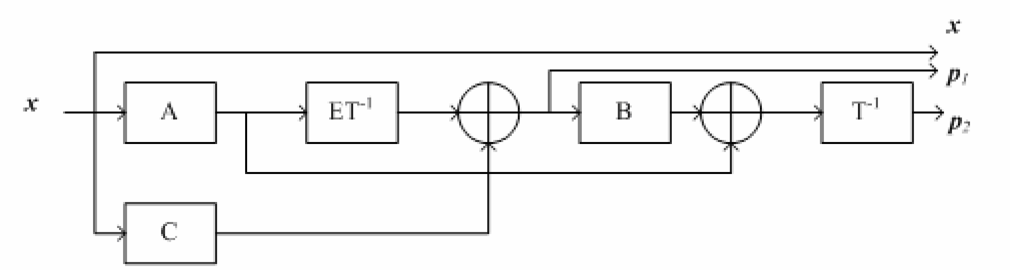


Figure 2-4：The block diagram of encoder method

Table 2-3：Steps of encoding method

Step1	Compute $A\boldsymbol{x}^T$ and $C\boldsymbol{x}^T$
Step2	Compute $ET^{-1}(A\boldsymbol{x}^T)$
Step3	Compute $\boldsymbol{p}_1^T = ET^{-1}(A\boldsymbol{x}^T) + C\boldsymbol{x}^T$
Step4	Compute $T\boldsymbol{p}_2^T = A\boldsymbol{x}^T + B\boldsymbol{p}_1^T$

CHAPTER 3 LOW DENSITY PARITY CHECK CODE

Low-**d**ensity **P**arity **C**heck **(LDPC)** code was first introduced by Gallager in 1962 [6], but was almost forgotten until its rediscovery it in the late 1990s. The graphical representation for the LDPC code was presented by R. N. Tanner [7, 30] in 1981. Mackay and Neal rediscovered the LDPC code and investigated its graph based iterative decoding algorithm [8, 9]. It has been shown in [10] that long LDPC codes based on the belief propagation [11] can achieve an error performance very closing to the Shannon limit. Many high speed communication systems such as *IEEE* 802.11n, 802.16e and DVB-S2 have considered employing LDPC code to enhance performance for its benefits, including good error performance and high parallelism. Besides, the decoding algorithm provides very simple arithmetic computations to decrease the complexity of hardware design and parallelism to increase the data rate. In this chapter, we discuss the decoding algorithms including belief propagation, row-update message passing algorithm and some implementation issues, Min-Sum approach, fixed-point simulation and etc. A trade-off between decoding algorithms is analyzed and scheduled row-update message passing with Min-Sum is suggested.

3.1 Concept of Low-density Parity Check Codes

Low-density Parity Check code, a linear block code defined by a very sparse parity check matrix ***H*** which means there are only a small number of ones in the entries. It was first introduced by Gallager [6] and rediscovered by MacKay [8,9]. For the properties of a sparse matrix, it makes the decoding algorithm simple and practical at good communication rates [9]. The sparse matrix also reduces the complexity of computation in decoding and encoding. However, LDPC decoders, which are highly parallelizable, have a much higher decoding speed than other decoder. The decoding algorithm based on **s**um-**p**roduct **a**lgorithm **(SPA)** is capable of parallel implementation, leading to a much higher decoding speed than other channel code decoder. We often divide LDPC codes into two types according to its degree distribution. One is regular LDPC code, the other is irregular. The regular LDPC codes mean that each row has the same number of ones, and each column does so. The irregular mean that numbers of ones in the rows or in the columns are different.

In the following example, it shows a parity check matrix of (10,5) regular LDPC code and its constraint equation,

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.1}$$

And

$$[x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9] \times H^T = [c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4] \tag{3.2}$$

$$\begin{aligned} c_0 &: x_3 + x_4 + x_7 + x_9 = 0 \\ c_1 &: x_0 + x_1 + x_5 + x_9 = 0 \\ c_2 &: x_1 + x_3 + x_6 + x_8 = 0 \\ c_4 &: x_2 + x_4 + x_5 + x_6 = 0 \\ c_5 &: x_0 + x_2 + x_7 + x_8 = 0 \end{aligned} \tag{3.3}$$

Figure 4-1 represents the five parity check equations (3.3) in the bipartite graph with 10 bit nodes (or variable node) and 5 check nodes. The column weight of H determines the number of edges (or degree) for each bit node. We usually illustrate the relationship between bit nodes and check nodes or treat direction of message passing from its bipartite graph.

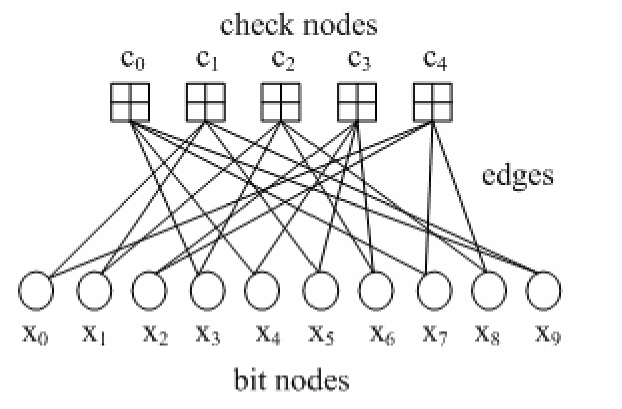


Figure 3-1：Example of bipartite graph for equation (3.1)

3.1.1 Message Passing Algorithm

LDPC decoding algorithm is based on soft iterative decoding which relies on the *message passing algorithm* or *belief propagation* [11,12]. We consider the following conditional probability,

$$P(x = a \mid C) \tag{3.4}$$

which is a *posterior* probability x to be a value a based on the condition event C. According to the Bayes's theorem, we extend the posterior equation as

$$P(x=a\,|\,C)=\frac{P(C\,|\,x=a)P(x=a)}{P(C)} \tag{3.5}$$

We want to know the value $P(x=a\,|\,C)$ when knowing other terms. The term $P(x=a)$ is a *prior* probability or referred to the *intrinsic* probability, denoted by $P_{int}(x=a)$. The other term $P(C\,|\,x=a)$ is proportional to the *extrinsic* probability which describes the probability that new information for x is obtained from the event C when assuming a is a value from alphabet set $\boldsymbol{A}$. We can express extrinsic probability as

$$P_{ext}(x=a)=(\sum_{a'\in \mathrm{A}} P(C\,|\,x=a'))^{-1}\,p(C\,|\,x=a)=\rho_e P(C\,|\,x=a) \tag{3.6}$$

The ρ_e represents the normalization constant to satisfy the condition $\sum_{a'\in \mathbf{A}} P_{ext}(x=a')=1$.

The posteriori probability in (3.5) can be described as

$$P_{ext}(x=a)=(\sum_{a'\in \mathrm{A}} P(C\,|\,x=a'))^{-1}\,p(C\,|\,x=a)=\rho_e P(C\,|\,x=a) \tag{3.7}$$

where $\rho_p=(\rho_e P(C))^{-1}$ is also the normalization factor ($\rho_p=\left(\sum_{a'\in A} P_{ext}(x=a')P_{\mathrm{int}}(x=a')\right)^{-1}$). If $\boldsymbol{A}=GF(2)$, the log-likelihood ratio representation for (3.7) will be

$$LLR_{post}(x)=\ln\frac{P_{post}(x=1)}{P_{post}(x=0)}=\ln\frac{P_{ext}(x=1)}{P_{ext}(x=0)}+\ln\frac{P_{\mathrm{int}}(x=1)}{P_{\mathrm{int}}(x=0)}=LLR_{ext}(x)+LLR_{\mathrm{int}}(x) \tag{3.8}$$

In the graph representation, we use an undirected graph, referred to the ***normal graph*** [13,14]. The vertices (nodes) denote the constraints. The ordinary edges denote the state variables for message passing. Symbol variables are denoted by left edges (half edges). Figure 3-2 shows the example with three vertices; the edges connecting only one node are left edges, the edges connecting two vertices are ordinary.

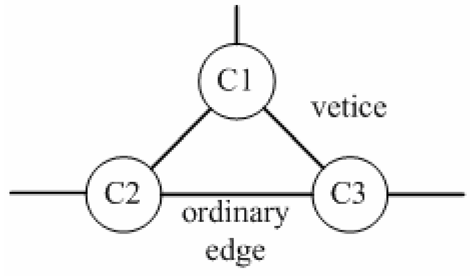


Figure 3-2：Example of normal graph

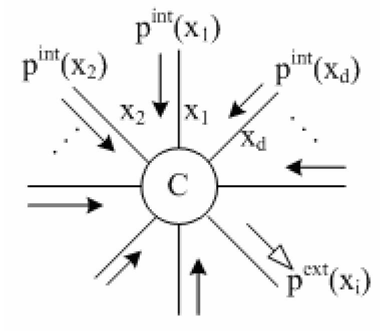


Figure 3-3：Graph presentation of the intrinsic and extrinsic probabilities

Figure 3-3 illustrates the graph of a single node (vertex) and ***d*** edges with the intrinsic and extrinsic probabilities. There are *d* symbols, x_1, x_2,..., x_d, respect to the constraint *C*. We define a set $\boldsymbol{S_c}$ which is a subspace of the ***d***-dimensional vector space $A^d(S_c \subset A^d)$, and any *d*-tuple $\boldsymbol{x} = (x_1, x_2, ..., x_d) \in \boldsymbol{S}_c$ will satisfy the constraint *C*. Each edge has the intrinsic probability $P_{\text{int}}(x_i)$ associated with the symbol x_i for $i = 1 \sim d$. Therefore a posteriori probability of a symbol x_i will be the combination of the intrinsic probability $P_{\text{int}}(x_i)$ and extrinsic $P_{ext}(x_i)$ (3.8).

From equation (3.8), we have to evaluate $P_{ext}(x_i)$ based on the constraint *C* and the other intrinsic probabilities $P_{\text{int}}(x_j)$ with $j \neq i$. The $P_{ext}(x_i)$ will be

$$\begin{aligned} P_{ext}(x_i) &= \rho_e P(C \mid x_i) \\ &= \rho_e \sum_{\substack{x_j, \forall j \neq i \\ x \in \mathbf{S}_c}} P(C, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d \mid x_i) \\ &= \rho_e \sum_{\substack{x_j, \forall j \neq i \\ x \in \mathbf{S}_c}} P(C \mid x_{1,} x_2, \ldots, x_d) P(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d \mid x_i) \\ &= \rho_e \sum_{\substack{x_j, \forall j \neq i \\ x \in \mathbf{S}_c}} \prod_{\substack{j=1 \\ j \neq i}}^{d} P_{\text{int}}(x_j) \end{aligned} \tag{3.9}$$

The $P(C \mid x_{1,} x_2, \ldots, x_d)$ is always equal to 1 because the constraint is true given $(x_{1,} x_2, \ldots, x_d)$ where x_j for j=1~d belong to the constraint set $\boldsymbol{S_C}$. Then a posteriori probability can be written as

$$\begin{aligned} P^{post}(x_i) &= \rho_p P^{\text{int}}(x_i) P^{ext}(x_i) \\ &= \rho_p \sum_{\substack{x_j, \forall j \neq i \\ x \in \mathbf{S}_c}} \prod_{j=1}^{d} P^{\text{int}}(x_j) \end{aligned} \tag{3.10}$$

where we assume the symbol variables $x_1, x_2, \ldots, x_d$ are independent, and ρ_e ρ_p is also a normalization constant.

- **Two Vertices**

Now we consider message passing between two vertices. Figure 3-4 shows the graph presentation of two constraints (two vertices, C_1 and C_2). The C_1 constraint has i edges where i-1 edges are left edges and only x_i is ordinary edge. On the other hand, $x_1 \sim x_d$ are constrained by C_2 where only x_i on the ordinary edge. Besides the two constraints $\boldsymbol{S}_{c_1}, \boldsymbol{S}_{c_2}$ sets are defined such that $\boldsymbol{x}_1 = (x_1, x_2, \ldots, x_i) \in \boldsymbol{S}_{c_1}$ and $\boldsymbol{x}_2 = (x_1, x_2, \ldots, x_d) \in \boldsymbol{S}_{c_2}$.

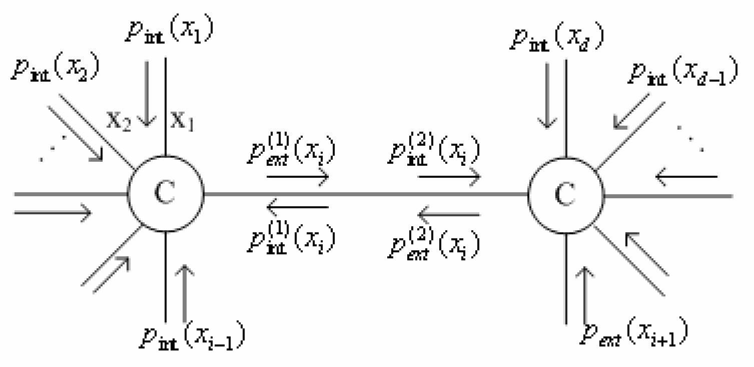


Figure 3-4：Graph presentation of message passing between two vertices

As shown in Figure 3-4, we have to evaluate the extrinsic probabilities for the left edges constrained by C_1 and C_2. First we only consider the constraint C_2. According to (3.9), the extrinsic probability can be written as：

$$\begin{aligned} P_{ext}(x_{i+1}) &= \rho_2 P(C_1 \mid x_{i+1}) \\ &= \rho_2 \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in S_{c_2}}} P_{\text{int}}^{(2)}(x_i) \end{aligned} \tag{3.11}$$

But we have to consider the value $P_{\text{int}}^{(2)}(x_i)$ constrained with C_1. Therefore we evaluate the extrinsic probability based on both constrains C_1 and C_2.

$$\begin{aligned} P_{ext}(x_{i+1}) &= \rho_e P(C_1, C_2 \mid x_{i+1}) \\ &= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in S_{c_2}}} P(C_1, C_2, x_i, x_{i+2}, \ldots, x_d \mid x_{i+1}) \\ &= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in S_{c_2}}} P(C_1 \mid C_1, \boldsymbol{x}_2) P(C_1, x_i, x_{i+2}, \ldots, x_d \mid x_{i+1}) \\ &= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in S_{c_2}}} P(C_1, x_i, x_{i+2}, \ldots, x_d \mid x_{i+1}) \end{aligned} \tag{3.12}$$

where the third equality comes from a Markov chain

$$P(C_1, C_2 \mid x_i) = P(C_1 \mid x_i) P(C_2 \mid x_i) \tag{3.13}$$

such that the term

$$P(C_2 \mid C_1, \boldsymbol{x}_2) = P(C_2 \mid \boldsymbol{x}_2) = 1, \ \ for\ \boldsymbol{x}_2 \in \boldsymbol{S}_{c_2} \tag{3.14}$$

Continuing from (3.12), we derive the equality as

$$\begin{aligned} P(C_1, x_i, x_{i+2}, \ldots, x_d \mid x_{i+1}) &= P(C_1 \mid \boldsymbol{x}_2) P(x_i, x_{i+2}, \ldots, x_d \mid x_{i+1}) \\ &= P(C_1 \mid x_i) P(x_i) P(x_{i+2}) \ldots P(x_d) \\ &= (\rho_1)^{-1} P_{ext}^{(1)}(x_i) P_{\text{int}}(x_i) \prod_{j=i+2}^{d} P(x_j) \end{aligned} \tag{3.15}$$

From Figure 3-4,

$$P_{ext}^{(1)}(x_i) = \rho_1 P(C_1 \mid x_i) \tag{3.16}$$

it is the extrinsic probability of x_i with respect to *C₁*. $P_{\text{int}}(x_j)$ is the intrinsic probability for the left edge connecting *C₂*, and $P_{\text{int}}(x_i)$ is the intrinsic probability for the ordinary edge x_i. Because the ordinary connecting *C₁* and *C₂* without external input, we can initialize the $P_{\text{int}}(x_i)$ to be a constant. We set $P_{\text{int}}(x_i) = \frac{1}{|\boldsymbol{A}|}$ for $x_i \in \boldsymbol{A}$. Then the extrinsic probability in (3.12) will be written as

$$P_{ext}(x_i) = \rho_e{}' \sum_{\substack{x_2 \backslash x_{j+1} \\ x_2 \in S_{c2}}} P_{ext}^{(1)}(x_i) \prod_{j=i+2}^{d} P(x_j) \tag{3.17}$$

where $\rho_e{}' = \rho_p / (\rho_1 |\boldsymbol{A}|)$.

Referring to Figure 3-4, we know if the extrinsic probability $P_{ext}^{(1)}(x_i)$ from C_1 is available and

$$P_{\text{int}}^{(2)}(x_i) = P_{ext}^{(1)}(x_i) \tag{3.18}$$

only the constraint C_2 is necessary to estimate $P_{ext}(x_{i+1})$. Therefore $P_{ext}(x_j)$ for $j = (i+2) \sim d$ can also be calculated by the same method. For $P_{ext}(x_l)$ with $l = 1 \sim (i-1)$, the extrinsic probability $P_{ext}^{(2)}(x_i)$ with respect to C_2 should be first computed and the intrinsic probability for C_1 is set to be

$$P_{\text{int}}^{(1)}(x_i) = P_{ext}^{(2)}(x_i) \tag{3.19}$$

The process of (3.18) or (3.19) is the message passing between vertices C_1 and C_2. With the message passing algorithm, we can simplify the problem of solving both C_1 and C_2 into the problem of solving the single vertex graph, which is much simpler than the two vertices case. The message passed on the edge x_i can be represented by

$$\mu_{C_1 \to C_2}(x_i) = P_{ext}^{(1)}(x_i) = \rho_1 \sum_{\substack{x_1 \setminus x_i \\ x_1 \in S_{c_1}}} \prod_{j=1}^{i-1} P_{\text{int}}(x_j) \tag{3.20}$$

$$\mu_{C_2 \to C_1}(x_i) = P_{ext}^{(2)}(x_i) = \rho_2 \sum_{\substack{x_2 \setminus x_i \\ x_2 \in S_{c_2}}} \prod_{j=i+1}^{d} P_{\text{int}}(x_j) \tag{3.21}$$

The operation of (3.20) and (3.21) are the **sum of products**, thus the message passing algorithm is also called the **sum-product algorithm** [15]. Generally in the graph with vertices, $C_0, G_1, \ldots, G_d$, the vertex C_0 has d ordinary edges that respectively connect to $G_1, G_2, \ldots, G_d$ with symbol variables $x_1, x_2, \ldots, x_d$. Assuming the messages $\mu_{C_j \to C_0}(x_j)$ with j = 1~ d have been obtained from G_1~G_d, we can get the value $\mu_{C_0 \to C_i}$ by

$$\mu_{C_0 \to C_i}(x_i) = \sum_{\substack{\mathbf{x} \setminus x_i \\ \mathbf{x} \in \mathbf{S}_{c_0}}} \prod_{\substack{j=1 \\ j \neq i}}^{d} \mu_{C_j \to C_0}(x_j) \tag{3.22}$$

where S_{C_0} is the constrain set for C_0, and $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$. The message $\mu_{C_0 \to C_i}$ for i=1~d can be obtained because they are the intrinsic probability inputs for vertices C_1~C_d.

Based on the concept of message passing algorithm above, LDPC decoding algorithm will be introduced on the next section.

3.1.2 LDPC Decoding Algorithm

Same as the linear block codes, a *m*-by-*n* LDPC code have a codeword $\boldsymbol{x} = [x_1, x_2, ..., x_N]$ needed to satisfy the equality $\boldsymbol{Hx}^T = 0$. The bipartite graph, Figure 3-5 is used to describe the relation between parity check matrix and codeword. It is the graph representation of equation (3.23). The check nodes and bit nodes are denoted as column constraint of parity check matrix and row index of codeword. The message passing algorithm is applied to passing the message between two nodes.

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.23}$$

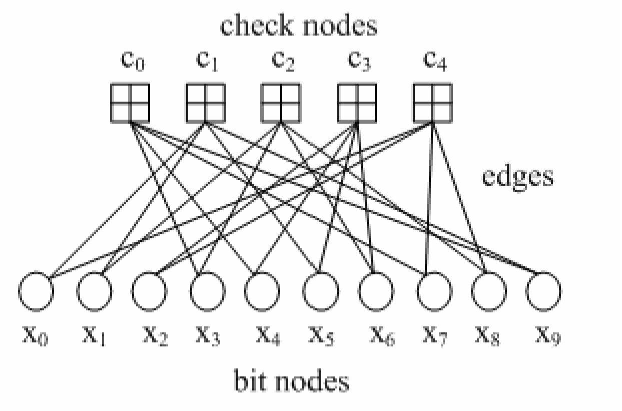


Figure 3-5：Bipartite graph of matrix (3.23)

$$\begin{aligned} c_0 &: x_3 + x_4 + x_7 + x_9 = 0 \\ c_1 &: x_0 + x_1 + x_5 + x_9 = 0 \\ c_2 &: x_1 + x_3 + x_6 + x_8 = 0 \\ c_4 &: x_2 + x_4 + x_5 + x_6 = 0 \\ c_5 &: x_0 + x_2 + x_7 + x_8 = 0 \end{aligned} \tag{3.24}$$

Equation (3.24) presents the parity check constraints that are similar to the role of vertex nodes in the massage passing graph. The operation in (3.24) all are exclusive-or. LDPC decoding algorithm is based on the belief propagation algorithm also called message passing algorithm. The bit node transfer information to other bits under the check node constraints. By iteratively exchanging more reliable information from other bits, the error bits are corrected.

The message passing algorithm is an APP (a posterior probability) only if the code graph has no cycles. The cycle-free implies that all code bits $x_{0,}x_1,....,x_{n-1}$ are independent. However the algorithm performs remarkably well even if dependent. Now we will derive the message passing algorithm for LDPC from first principles. Prior to introduce the decoding algorithm, we have to know some notations, n code bits, the number of information bits is $k = n - m$, and the code rate is $\frac{k}{n}$, n channel received output $\boldsymbol{r} = (r_0, r_1, ..., r_{n-1})$, $\boldsymbol{r} = \boldsymbol{x} + \text{noise}$, M(j) be the set of parity nodes connected to the code bits x_j, C_j is the event that all parity check constraints associated with x_j are satisfied and N(m) be the set of bit nodes connected to the m'th parity check, and a parity check matrix is *m*-by-*n* dimension. The derivation below referred to the description of Gallager [6].

Using the assumption of code bit independence and Baye's rule, a posterior probability $P(x_j = b \mid C_j, r)$ can be written as

$$P(x_j = b \mid C_j, \boldsymbol{r}) = K \times P(r_j \mid x_j) \times P(C_j \mid x_j = b, \boldsymbol{r}) \tag{3.25}$$

where K is a constant for both b=1 or 0,

$$K = \frac{p(b_i)p(b_i, \boldsymbol{r})}{p(S_i, \boldsymbol{r})p(r_i, b_i)} = \frac{p(b_i)p(b_i)p(\boldsymbol{r} \mid b_i)}{p(S_i, \boldsymbol{r})p(b_i)p(r_i \mid b_i)} = \frac{p(b_i)\prod_{j=1, j \neq i}^{n} p(r_j)}{p(S_i, \boldsymbol{r})}$$

The equation (3.25) is similar to equation (3.5).

For Gaussian noise：

$$P(r_j \mid x_j) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp^{\frac{(rj+(-1)^b)^2}{2\sigma^2}} \tag{3.26}$$

For BSC (Binary Synchronous Communication) ：

$$P(r_j \mid x_j = 0) = p^{r_j}(1-p)^{1-r_j} \tag{3.27}$$

$$P(r_j \mid x_j = 0) = p^{1-r_j}(1-p)^{r_j} \tag{3.28}$$

where assuming false probability is equal to p, so true probability is $1-$ p as shown in Figure 3-6.

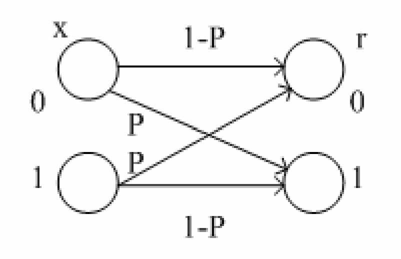


Figure 3-6：Transmission probability for BSC

The second term in (3.25) is the probability that all parity check constraints connected to x_j are satisfied given $\boldsymbol{r}$ and $x_j = b$. Notice that $C_j = \{C_{0j},..,C_{kj}\}$ is a collection set of events, where C_{mj} is the m'th parity check node connected to x_j is satisfied. Because of the assumption of code bits independence, the term can be written as

$$P(C_j \mid x_j = b, \boldsymbol{r}) = P(C_{0j}, C_{1j}, \ldots, C_{kj} \mid x_j = b, \boldsymbol{r}) = \prod_{m \in M(j)} P(C_{mj} \mid x_j = b, \boldsymbol{r}) \tag{3.28}$$

If b=0, it implies that code bits other than x_j connected to the m'th parity check have an even number of 1's. If b=1, the other bits must have odd parity. Using the fact, we will derivate $P(C_j \mid x_j = b, \boldsymbol{r})$ as a relatively simple form as follows.

As a preliminary calculation, suppose two bits satisfy a parity check constraint

$x_1 \oplus x_2 = 0$, and we know that $p_1 = P(x_1 = 1)$ and $p_2 = P(x_2 = 1)$. Let $q_1 = 1 - p_1$ and $q_2 = 1 - p_2$. Then when the constraint is satisfied, the probability will be

$$\begin{aligned} P(x_1 \oplus x_2 = 0) &= (1-p_1)(1-p_2) + p_1 p_2 \\ &= 2p_1p_2 - p_1 - p_2 + 1 \end{aligned} \tag{3.29}$$

which can be written as

$$2P(x_1 \oplus x_2 = 0) - 1 = (1-2p_1)(1-2p_2) = (q_2 - p_2)(q_1 - p_1) \tag{3.30}$$

We suppose L+1 bits connected to one parity check node as shown in Figure 3-7. First we compute the term $P(x_1 \oplus x_2 = 0)$ as a new node x' then compute the probability $P(x' \oplus x_2 = 0)$, iteratively to find out the probability to satisfy the constraint C with L+1 nodes ($(x_1, x_2, ..., x_L)$). It is a mathematically technique to reduce the complex problem to be simple two nodes problem. Following description will derive the detail LDPC decoding algorithm.

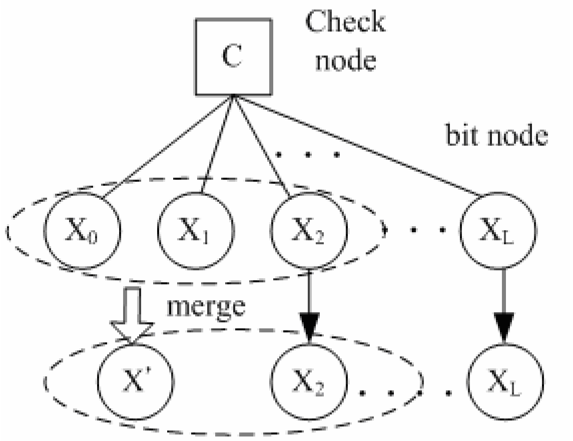


Figure 3-7：Merged graph of check node and bit node with *L*+1 degree

For known probabilities $\{p_1, p_2, ..., p_L\}$ corresponding to the bits $\{x_1, x_2, ..., x_L\}$. We want to generalize (3.30) to find the probability distribution for $z_L = x_1 + x_2 + ... + x_L$, where $z_L = z_{L-1} \oplus x_L$.

$$\begin{aligned} 2P(z_L =) - 1 &= (1 - 2P(z_{L-1} = 1))(1 - 2p_L) \\ &= (2P(z_{L-1} = 0)(1 - 2p_L)) \end{aligned} \tag{3.31}$$

where $p_L = P(x_L = 1)$. Applying the recursively yields

$$2P(z_L = 0) - 1 = \prod_{i=1}^{L}(1 - 2p_i) \tag{3.32}$$

or
$$P(z_L = 0) = \frac{1}{2}(1 + \prod_{i=1}^{L}(1 - 2p_i)) = \frac{1}{2}(1 + \prod_{i=1}^{L} q_i - p_i) \tag{3.33}$$

Similarly it can show

$$P(z_L = 1) = \frac{1}{2}(1 - \prod_{i=1}^{L}(q_i - p_i)) \tag{3.34}$$

Returning to our calculation of $P(S_{mj} \mid x_j = b, \boldsymbol{r})$, we derivate the equalities according to $x_j = 1$ or $x_j = 0$ to choose $P(z_L = 1)$ or $P(z_L = 0)$.

$$P(C_{mj} \mid x_j = 0, \boldsymbol{r}) = \frac{1}{2}(1 + \prod_{n'=N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})) \tag{3.35}$$

$$P(C_{mj} \mid x_j = 1, \boldsymbol{r}) = \frac{1}{2}(1 - \prod_{n'=N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})) \tag{3.36}$$

where $q^0_{mn'}$ is the probability that code bit $x_{n'} = 0$, given $\boldsymbol{r}$ and excluding any information about $x_{n'}$ from parity check m. It must need the exclusion operation because we desire extrinsic knowledge about $x_{n'}$ form parity check constraints to get the extrinsic information about x_j. By combination of (3.25) (3.27) (3.35), we get the final expressions for a posterior probability.

$$\begin{aligned} P(x_j = 0 \mid C_j, \boldsymbol{r}) &= K \times P(r_j \mid x_j = 0) P(C_j \mid x_j = 0, \boldsymbol{r}) \\ &= K \times P(r_j \mid x_j = 0) \prod_{m \in M(j)} \frac{1}{2}(1 + \prod_{n' \in N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})) \end{aligned} \tag{3.37}$$

$$\begin{aligned} P(x_j = 1 \mid C_j, \boldsymbol{r}) &= K \times P(r_j \mid x_j = 1) P(C_j \mid x_j = 1, \boldsymbol{r}) \\ &= K \times P(r_j \mid x_j = 1) \prod_{m \in M(j)} \frac{1}{2}(1 - \prod_{n' \in N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})) \end{aligned} \tag{3.38}$$

Inspection of the APP in (3.38), we can denote some operations as “**check node”,** some operations as : **”bit node”**. For example,

$$P(x_j=1\,|\,C_j,\mathbf{r})=K\times\underbrace{\overbrace{P(r_j\,|\,x_j=1)}^{\text{channel value}}\overbrace{\prod_{m\in M(j)}\frac{1}{2}(1-\prod_{n'\in N(m)\backslash j}(q^0_{mn'}-q^1_{mn'}))}^{\text{check node update}}}_{\text{bit node update (variable)}}\qquad(3.39)$$

The notation can be simplified by letting $\delta q_{mj}=q^0_{mn'}-q^1_{mn'}$ and then define the check node equation as

$$r^0_{mj}=\frac{1}{2}(1+\prod_{n'\in N(m)\backslash j}\delta q_{mn'})\qquad(3.40)$$

$$r^1_{mj}=\frac{1}{2}(1-\prod_{n'\in N(m)\backslash j}\delta q_{mn'})\qquad(3.41)$$

For the BSC, the right terms of expressions in (3.39) can be simplified. When first iteration, the probabilities r^0_{mj},r^1_{mj} can be rewritten as

$$\begin{aligned}q^0_{mj}&=P(x_j=0\,|\,r_j)=p^{r_j}(1-p)^{1-r_j}\\q^1_{mj}&=P(x_j=1\,|\,r_j)=p^{1-r_j}(1-p)^{r_j}\end{aligned}\qquad(3.42)$$

We rewrite (3.40) (3.41) with (3.42),

$$\begin{aligned}r^0_{mj}&=\frac{1}{2}(1+\prod_{n'\in N(m)\backslash j}(1-2p)(-1)^{r_{n'}})\\&=\frac{1}{2}(1+(1-2p)^{|N(m)|-1}\prod_{n'=N(m)\backslash j}(-1)^{r_{n'}})\\r^1_{mj}&=\frac{1}{2}(1-(1-2p)^{|N(m)|-1}\prod_{n'=N(m)\backslash j}(-1)^{r_{n'}})\end{aligned}\qquad(3.43)$$

According to numbers of N(m), we summarize the check node functions as (3.44)

$$\begin{aligned}r^0_{mj}&=\frac{1}{2}(1-(1-2p)^{|N(m)|-1}),\quad\text{if the bits of N(m)\j is odd}\\&=\frac{1}{2}(1+(1-2p)^{|N(m)|-1}),\quad\text{if the bits of N(m)\j is even}\\r^1_{mj}&=\frac{1}{2}(1-(1-2p)^{|N(m)|-1}),\quad\text{if the bits of N(m)\j is even}\\&=\frac{1}{2}(1+(1-2p)^{|N(m)|-1}),\quad\text{if the bits of N(m)\j is odd}\end{aligned}\qquad(3.44)$$

The APP can be further simplified as

$$P(x_j=0\,|\,C_j,\mathbf{r})=P(r_j\,|\,x_j=0)\prod_{m\in M^{odd}(j)}\frac{1}{2}(1-(1-2p)^{|N(m)|-1})\prod_{m\in M^{even}(j)}\frac{1}{2}(1+(1-2p)^{|N(m)|-1})$$
$$P(x_j=1\,|\,C_j,\mathbf{r})=P(r_j\,|\,x_j=1)\prod_{m\in M^{odd}(j)}\frac{1}{2}(1+(1-2p)^{|N(m)|-1})\prod_{m\in M^{even}(j)}\frac{1}{2}(1-(1-2p)^{|N(m)|-1})$$
(3.45)

where $M^{odd}(j)$ are the sets of nodes connected to x_j with odd parity, and $M^{even}(j)$ are sets with even parity.

➢ Decision Step

With the analysis of APP operation, we can decode the value $x=0$ or $x=1$. We derivate the equation as,

$$x_j=\begin{cases}0\ ;\ \text{if}\,P(x_j=0\,|\,C_j,\boldsymbol{r})\geq P(x_j=0\,|\,C_j,\boldsymbol{r})\\ 1\ ;\text{if}\,P(x_j=0\,|\,C_j,\boldsymbol{r})<P(x_j=0\,|\,C_j,\boldsymbol{r})\end{cases}\qquad(3.46)$$

We have to analyze the magnitude of the APPs to decode the value. We can simplify the equation (3.46) to compare with sign value by dividing the APPs as **log** domain.

The check node and bit node operations are the main procedures to iteratively passing message to correct error bits. In next sub-section, we will introduce the **B**elief **P**ropagation **(BP)** algorithm in **log** domain. The multiplication in log domain will be simplified to be summation form. Obviously, the advantages are to reduce the hardware complexity and to decode the soft results more easily by verifying the sign of APPs.

● Message Passing algorithm in log domain

We introduce the APP equations in **log** domain. The operations in **log** domain make computations more clear and easy. Equality (3.46) combines (3.25) with $x_j=0$ and $x_j=1$.

$$LLR^{posterior}(x_j) = \log \frac{P(x_j = 0 \mid \boldsymbol{r}, C_j)}{P(x_j = 1 \mid \boldsymbol{r}, C_j)} \tag{3.46}$$

x_j=1 if $LLR^{posterior}(x_j) > 0$, otherwise $x_j = 1$. We only compute that if APP is positive or negative value. The decision for APPs will become easy in **log** domain.

First,

$$\begin{aligned} LLR^{posterior}(x_j) &= \log \frac{P(x_j = 0 \mid \boldsymbol{r}, C_j)}{P(x_j = 1 \mid \boldsymbol{r}, C_j)} \\ &= \log \frac{P(r_j \mid x_j = 0)}{P(r_j \mid x_j = 1)} + \log \prod_{m \in M(j)} \frac{1 + \prod_{n'=N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})}{1 - \prod_{n'=N(m)\backslash j} (q^0_{mn'} - q^1_{mn'})} \end{aligned} \tag{3.47}$$

For Gaussian noise,

$$\log \frac{P(x_j = 0 \mid \boldsymbol{r}, C_j)}{P(x_j = 1 \mid \boldsymbol{r}, C_j)} = \frac{2r_j}{\sigma^2} \tag{3.48}$$

Letting $\delta q_{mj} = q^0_{mn'} - q^1_{mn'}$ and $LLR(q^0_{mn'}) = \log \frac{q^0_{mn'}}{q^1_{mn'}}$, simple substitution gives $\delta q_{mn'} = \tanh(LLR(\frac{q^0_{mn'}}{2}))$. We rewrite (3.47) as (3.49).

$$\begin{aligned} \frac{2r_j}{\sigma^2} + \sum_{m \in M(j)} \log \frac{1 + \prod_{n' \in N(m)\backslash j} \delta q_{mn'}}{1 - \prod_{n' \in N(m)\backslash j} \delta q_{mn'}} &= \frac{2r_j}{\sigma^2} + \sum_{m \in M(j)} \log \frac{1 + s_{mj} e^{A_{mj}}}{1 - s_{mj} e^{A_{mj}}} = \frac{2r_j}{\sigma^2} - \sum_{m \in M(j)} \log \frac{1 - s_{mj} e^{A_{mj}}}{1 + s_{mj} e^{A_{mj}}} \\ &= \frac{2r_j}{\sigma^2} - \sum_{m \in M(j)} \log - \left(\frac{s_{mj} e^{A_{mj}} - 1}{s_{mj} e^{A_{mj}} + 1}\right) \end{aligned} \tag{3.49}$$

where

$$\begin{aligned} \operatorname{sgn}(x) &= \begin{cases} 1 & ; x \geq 0 \\ -1 & ; x < 0 \end{cases} \\ s_{mj} &= \prod_{n'=N(m)\backslash j} \operatorname{sgn}(\delta q_{mn'}) = \prod_{n'=N(m)\backslash j} \operatorname{sgn}(LLR(q^0_{mn'})) \\ A_{mj} &= \prod_{n'=N(m)\backslash j} \log(|\tanh(\frac{LLR(q^0_{mn'})}{2})|) \end{aligned} \tag{3.50}$$

Because the argument of **log()** in (2.50) is always positive. The equation (3.49) can be simplified further to (3.51)

$$\begin{aligned}&\frac{2r_j}{\sigma^2}-\sum_{m\in M(j)} s_{mj}\log(-\tanh(\frac{A_{mj}}{2}))\\&=\frac{2r_j}{\sigma^2}-\sum_{m\in M(j)} s_{mj}\log(|\tanh(\frac{A_{mj}}{2})|)\end{aligned} \tag{3.51}$$

Letting $\psi(x)=\log(|\tanh(\frac{x}{2})|)$ called persi function**,** (3.47) will be substituted as

$$LLR^{posteriror}(x_j)=\frac{2r_j}{\sigma^2}-\sum_{m\in M(j)} s_{mj}\psi(A_{mj}) \tag{3.52}$$

where $$A_{mj}=\prod_{n'\in N(m)\backslash j}\log(|\tanh(\frac{LLR(q^0_{mn'})}{2})|)=\sum_{n'=N(m)\backslash j}\psi(LLR(q^0_{mn'})) \tag{3.53}$$

We summarize the **BP (Belief Propagation)** algorithm in **log** domain for LDPC codes as follows.

- Notation：

$$\psi(x)=\log(\tanh(\frac{x}{2}))=\log\frac{e^x+1}{e^x-1}=\psi^{-1}(x) \tag{3.54}$$

- Initialization：

$$\mathrm{LLR}(\mathrm{q}^0_{\mathrm{jm}})=\log(\frac{\mathrm{q}^0_{\mathrm{jm}}}{\mathrm{q}^1_{\mathrm{jm}}})=\frac{2\mathrm{r_j}}{\sigma^2};\ \mathit{for}\ \mathrm{j}=0,\ldots,\mathrm{N}-1 \tag{3.55}$$

- Check node (CHK/row operation) ：

$$r_{ji}=s_{ji}\psi(\sum_{i'\in N(j)\backslash i}\psi(|q_{i'j}|));\qquad s_{ji}=\prod_{i'\in N(j)\backslash i}\mathrm{sgn}(q_{i'j}) \tag{3.56}$$

- Bit node (VAR/column operation)：

$$q_{ij}=p_i+\sum_{j'\in M(i)\backslash j} r_{j'i}=\sum_{j'\in M(i)\backslash j} r_{j'i}+p_i-r_{ji} \tag{3.57}$$

- Decision： $$q_j=p_i+\sum_{j'\in M(i)} r_{j'i}\qquad x_j=\begin{Bmatrix}1;\ \text{if}\ q_j\ge 0\\0;\ \text{if}\ q_j<0\end{Bmatrix} \tag{3.58}$$

3.2 Optimized Approach for Hardware Implementation

In chapter 3.1, we will derive the LDPC decoding algorithm, message passing or *belief propagation* (BP) for LDPC codes. But it is hard to implement the function $\psi(x) = \log(\tanh(\frac{x}{2})) = \log\frac{e^x+1}{e^x-1}$ in hardware and the complexity of exponential and log is very huge. Some methods are suggested to approximate the nonlinear function, like **L**ook **U**p **T**able **(LUT)** or *Min-Sum* approach [16]. LUT is limited to the trade-off between size of LUTs and data accuracy. The other method, min-sum approach can reduce decoding complexity and all nonlinear calculations can be averted. However there would be approximation inaccuracy between BP algorithm and Min-Sum approach. To compensate the performances, a constant normalization factor or an offset value is often applied [24, 25, 26]. However, min-sum algorithm does not consider the area problem like LUTs. It has well decoding performance with an appropriate compensation value and calculations avoid arithmetic log. The compensation factor should be explored by simulations which provide error correction performance closed to BP algorithm. In chapter 3.2.1, we will introduce min-sum algorithm.

BP algorithm iteratively exchanges message to correct error bits, but its over numbers of iteration is the most disadvantage. Chapter 3.2.2 will introduce the **r**ow-update **m**essage **p**assing algorithm [17] **(RMP)** to reduce the iteration problem. The concept of **RMP** is to use newer information to instead of older information as soon as possible. It can be taken as a scheduling layered decoding [18]. The obvious effect is that information converges faster than BP algorithm. Row-update message passing algorithm is adopted with min-sum approach in the proposed architecture. The main rationale in detail will be introduced in Chapter 3.2.2.

3.2.1 Min-Sum Algorithm

The ratio of defined in (3.35) and (3.36) can be simplified to two node and be rewritten as

$$LLR(x_i) = \log \frac{P(C_{m\backslash M(j)} \mid x_j = 0, \boldsymbol{r})}{P(C_{m\backslash M(j)} \mid x_j = 1, \boldsymbol{r})} \tag{3.60}$$

We redefine the notation to simplify equations.

$$LLR(x_i) = \log \frac{P(x_i = 0)}{P(x_i = 1)} = \log \frac{1 - P(x_i = 1)}{P(x_i = 1)} \tag{3.61}$$

$P(x_i = 0)$ is a extrinsic probability $x_i = 0$, given computing node $x_j = 0$ and received ***r***.

If dealing with two connecting nodes, notated as x and y, rewriting (3.58) as (3.59) and continue the derivation.

$$P(x=1) = \frac{1}{e^{LLR(x)}+1}; \quad P(x=0) = \frac{e^{LLR(x)}}{e^{LLR(x)}+1} \tag{3.62}$$

With the definition, $\tanh(\frac{x}{2}) = \frac{e^x - 1}{e^x + 1}$,

$$1 - 2P(x=1) = \frac{e^{LLR(x)} - 1}{e^{LLR(x)} + 1} = \tanh(\frac{LLR(x)}{2}) \tag{3.63}$$

Then calculate the probabilities of constraints satisfied with two nodes,

$$P(x \oplus y = 0) = \frac{1 + e^{LLR(x)} e^{LLR(y)}}{(1 + e^{LLR(x)})(1 + e^{LLR(y)})}$$

$$P(x \oplus y = 1) = \frac{e^{LLR(x)} + e^{LLR(y)}}{(1 + e^{LLR(x)})(1 + e^{LLR(y)})}$$

$$\Rightarrow LLR(x \oplus y) = \log \frac{1 + e^{LLR(x)} e^{LLR(y)}}{e^{LLR(x)} + e^{LLR(y)}} \tag{3.63}$$

We want to derive the formulas based on tanh rules, so express the notations as tanh form. And then take use of tanh algebraic property and then express (3.63) as (3.64).

$$LLR(x\oplus y)=\log\frac{(e^{LLR(x)}+1)(e^{LLR(y)}+1)+(e^{LLR(x)}-1)(e^{LLR(y)}-1)}{(e^{LLR(x)}+1)(e^{LLR(y)}+1)-(e^{LLR(x)}-1)(e^{LLR(y)}-1)}$$
$$=\log\frac{1+(e^{LLR(x)}-1)(e^{LLR(y)}-1)/(e^{LLR(x)}+1)(e^{LLR(y)}+1)}{1-(e^{LLR(x)}-1)(e^{LLR(y)}-1)/(e^{LLR(x)}+1)(e^{LLR(y)}+1)} \tag{3.64}$$

Note：

$$\tanh(LLR(x)/2)=\frac{e^{LLR(x)}-1}{e^{LLR(x)}+1};\qquad \tanh^{-1}(x)=\frac{1}{2}\log\frac{1+x}{1-x};$$

Rewrite (3.61) as

$$LLR(x\oplus y)=2\tanh^{-1}(\tanh(\frac{LLR(x)}{2})\tanh(\frac{LLR(y)}{2})) \tag{3.65}$$

With decomposition of operations, we can simplify the operations to sign-operation and absolute value operation. The following will use the cosh(x) to substitute tanh(x),

Note：

$$\log(\cosh(x))=\log(\frac{e^{x}+e^{-x}}{2})=\log\frac{e^{|x|}(1+e^{-2|x|})}{2}$$
$$=|x|-\ln 2+\ln(1+e^{-2|x|}) \tag{3.66}$$

(3.65) can be rewrite as

$$LLR(x\oplus y)=\log\frac{1+e^{LLR(x)}e^{LLR(y)}}{e^{LLR(x)}+e^{LLR(y)}}=\log\frac{(e^{-\frac{LLR(x)+LLR(y)}{2}}+e^{\frac{LLR(x)+LLR(y)}{2}})/2}{(e^{\frac{LLR(x)-LLR(y)}{2}}+e^{-\frac{LLR(x)-LLR(y)}{2}})/2}$$
$$=\log(\cosh(\frac{LLR(x)+LLR(y)}{2}))-\log(\cosh(\frac{LLR(x)-LLR(y)}{2}))$$
$$=\left|\frac{LLR(x)+LLR(y)}{2}\right|-\left|\frac{LLR(x)-LLR(y)}{2}\right|+\Delta(LLR(x),LLR(y))$$
$$=\operatorname{sgn}(LLR(x))\operatorname{sgn}(LLR(y))\min(|LLR(x)|,|LLR(y)|)+\Delta(LLR(x),LLR(y))$$
$$=CHK(x\oplus y)$$

(3.67)

Note： $$\Delta=\log\frac{1+e^{-|LLR(x)+LLR(y)|}}{1+e^{-|LLR(x)-LLR(y)|}} \tag{3.68}$$

When extending two nodes to d degrees, we can derive (3.67). The “min” in equation (3.67) is to find out the minimal value. As the same idea, it derives equation (3.69) when applying *d* nodes.

$$
\begin{aligned}
LLR(C_n \mid \mathbf{r}, x_i) &= LLR(x_1 \oplus x_2 \oplus \ldots \oplus x_{j-1} \oplus x_{j+1} \oplus \ldots \oplus x_d) \\
&= \prod_{n' \in N(m)\backslash n} \mathrm{s}ign(LLR(x_{n'})) \times \psi\Big(\sum_{n' \in N(m)\backslash n} \psi(|LLR(x_{n'})|)\Big) \\
&\approx \prod_{n' \in N(m)\backslash n} \mathrm{s}ign(LLR(x_{n'})) \times \min_{n' \in N(m)\backslash n} (|LLR(x_{n'})|) \times \beta; \;\; 0 \le \beta \le 1 \qquad (3.69) \\
&\approx \prod_{n' \in N(m)\backslash n} \mathrm{s}ign(LLR(x_{n'})) \times \min_{n' \in N(m)\backslash n} (|LLR(x_{n'})|) - \alpha; \;\; \alpha \ge 0
\end{aligned}
$$

Note： $\psi(x) = \psi^{-1}(x)$

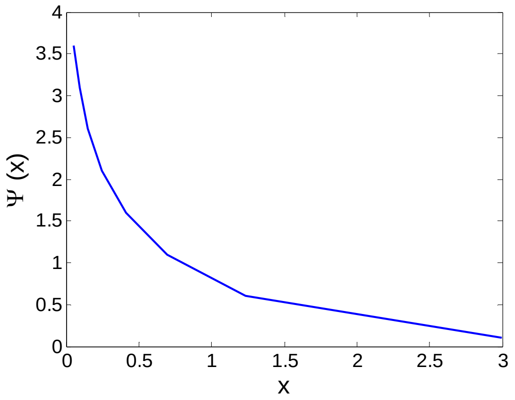


Figure 3-8：Function graph of $\psi(x)$

Figure 3-8 shows the functional curve of $\psi(x)$. Its output is as smaller as x increasing and $\psi(x) = \psi^{-1}(x)$ so that we can simplify $\psi(x)$ by sorting absolute *LLR(x)* to find out the corresponding minimal value. The α, β are compensation factors to compensate performance degradations due to inaccuracy approximation. The β is called normalization factor [24,25,26] that often own better compensation capacity than α [27,28,29] called offset factor. It is difficult to find an adequate constant factor for different LDPC codes with various degree distributions. To improve the approximation accuracy, a self-compensation technique is proposed by using dynamic normalization in [19]. However, considering of hardware implementation, a fixed factor is adopted compensate the deviation according to the simulations with different factors, 0.625, 0.75, 0,875 and 1. The related simulation results are shown in chapter 3.3.1.

The min-sum algorithm reduces the hardware complexity, but the iteration problem still exists. We adopt **r**ow-update **m**essage **p**assing algorithm **(RMP)** with min-sum to solve the problem. The main rationale is described in the next section.

3.2.2 Row-Update Message Passing Algorithm

The main concept of *row-update message passing algorithm* **(RMP)** [17] is a scheduling version from layered decoding algorithm that update information as soon as possible, not same as BP algorithm, that has to update check node first and then bit node, called one iteration. On layered decoding algorithm, each row of ***H*** can be considered as a component code (or a layer). We call the iterations within a layer as the sub-iterations and the overall process for layer to layer as super-iterations (or just iteration). From layer to layer, the component code is just an interleaved version of each other. As each next layer starts decoding, like Turbo decoding, its inputs are combined from the last layer or other prior layers, if necessary. Simulation results show that the layered decoding algorithm requires only 20~50% number of iterations of the conventional BP algorithm to achieve the same error-correction performance [17, 18]. For high parallelizable implementation, some rows are merged into one layer and decoded simultaneously. For efficient message update, row/column schedule for the parity check matrix is suggested to group a collision-free layer in which the column weight is at most one. The algorithm decode them row-by-row in sequence so that called *row-update message passing algorithm*, which is compatible for QC-LDPC because of itself Quasi-Cyclic structure. Figure 3-9 and Figure 3-10 show the differences between BP algorithm and *row-update message passing algorithm* (RMP), respectively. Step1~2 describe the detail equation steps for BP algorithm and RMP algorithm, respectively.

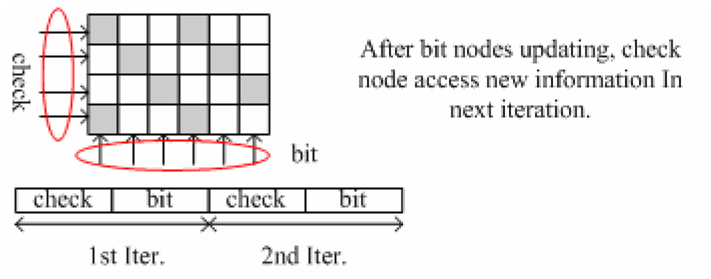

check
bit
After bit nodes updating, check node access new information In next iteration.
check
bit
check
bit
1st Iter.
2nd Iter.

Figure 3-9：Process of BP algorithm

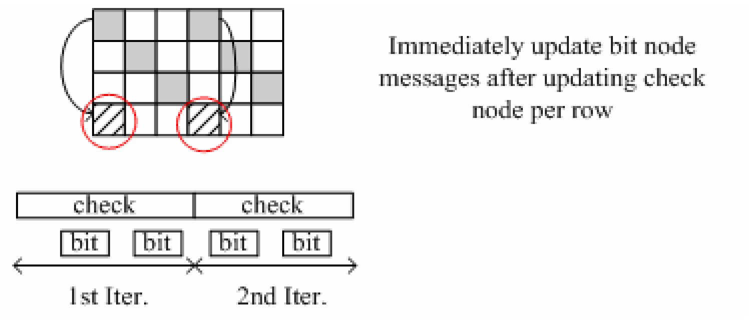

Immediately update bit node messages after updating check node per row
check
check
bit
bit
bit
bit
1st Iter.
2nd Iter.

Figure 3-10：Process of row-update message passing algorithm

LDPC Decoding Algorithm Steps：

- Initialization：

$$\mathrm{LLR}(q_i)=\log\left(\frac{q_i^0}{q_i^1}\right)=\frac{2r_i}{\sigma^2};\ \textit{for}\ i=0,\ldots,N-1$$

- Check node (CHK/row operation)：

$$r_{ji}=s_{ji}\psi\Big(\sum_{i'\in N(j)\backslash i}\psi(|q_{i'j}|)\Big);\qquad s_{ji}=\prod_{i'\in N(j)\backslash i}\mathrm{sgn}(q_{i'j}) \qquad \text{<1>}$$

- Bit node (VAR/column operation)：

$$q_{ij}=p_i+\sum_{j'\in M(i)\backslash j}r_{j'i}=\sum_{j'\in M(i)\backslash j}r_{j'i}+p_i-r_{ji} \qquad \text{<2>}$$

$;\ p_i=LLR(q_i);$ initial channel value

✧ **Belief Propagation Algorithm**

Step1-1: $\{\forall j\rightarrow\{\forall i':\text{<1>}\}\}\ \rightarrow\{\forall j\rightarrow\{\forall i':\text{<2>}\}\}$

Step1-2: check early termination: finish decoding or return step1-1.

✧ **Optimized Row-Update Message Passing Algorithm**

Step2-1: $\forall j \rightarrow \{\forall i' : <2>\} \rightarrow \{\forall i' : <1>\}$

Step2-2: check early termination: finish decoding or return step 2-1.

The following Figures will show the floating simulation results and comparison between BP and row-update message passing algorithm. Because channel coding is belong to outer receiver and a good receiver can recover multi-channel, fading channel and other channel model into simple AWGN channel, so that we only consider AWGN channel model which is enough to stand for error-correction performances. For simulation time, we only consider BPSK modulation excluding other modulation types. The simulation environment is setting as **BPSK** modulation and **AWGN** channel model and **10^8** data is simulated to show **10^{-6} BER** (**B**it **E**rror **R**ate) by C-language. In Figure 3-11~14, they show floating simulation results of maximum codeword lengths and minimal lengths for 802.11n and 802.16e, respectively.

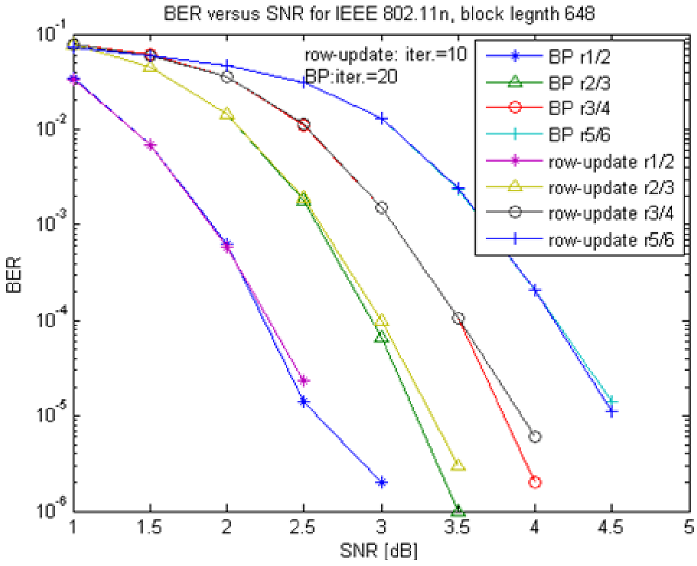


Figure 3-11：BP v.s RMP algorithm in 802.11n with codeword length 648 bits

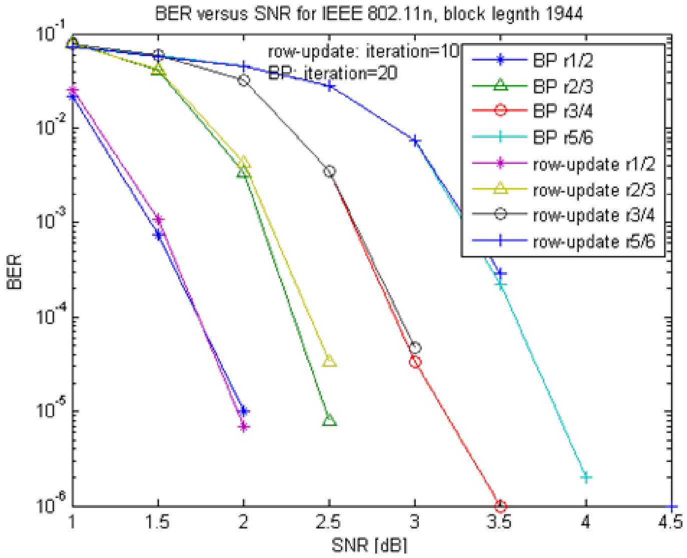


Figure 3-12：BP v.s RMP algorithm in 802.11n with codeword length 1944 bits

Figure 3-11 and 3-12 show the simulations for full code rates, 1/2, 2/3, 3/4, and 5/6 in IEEE 802.11n with the maximum codeword length 1944 bits and minimal codeword length 648 bits, respectively.

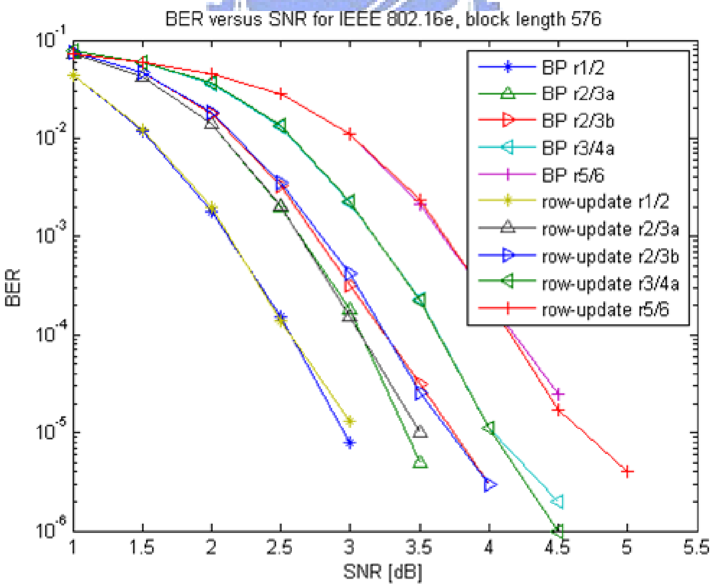


Figure 3-13：BP v.s RMP algorithm in 802.16e codeword length 576 bits

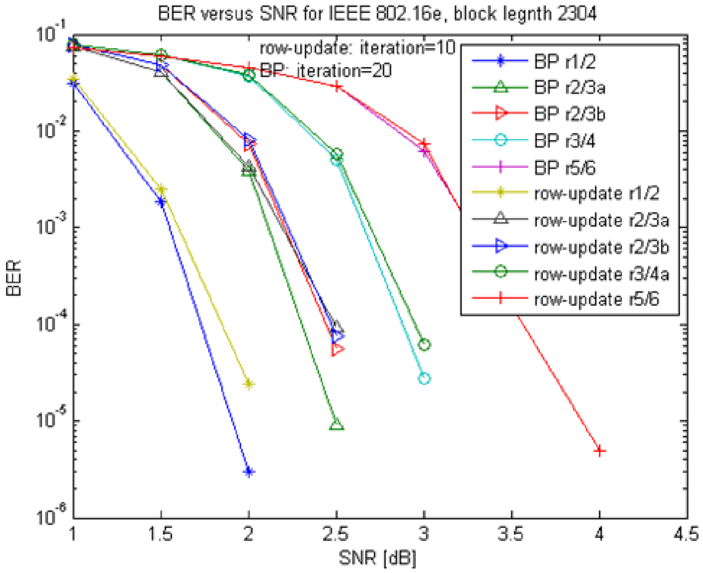


Figure 3-14：BP v.s RMP algorithm in 802.16e, codeword length 2304 bits

Figure 3-13 and 2-14 show the simulations for with code rate, 1/2, 2/3A, 2/3B, 3/4A, and 5/6 in IEEE 802.16e with the maximum codeword length 2304 bits and minimal codeword length 576 bits, respectively. Maximum number of iteration is limited to 20-time for Belief Propagation and 10-time for row-update message passing algorithm. Only half iterations (10 iterations) are needed to achieve the same performance as BP algorithm (20 iterations). Some hardware design issues are considered deeply in section 3.3.

3.2.3 Trade-off between Decoding Algorithms and Code Structures

Row-update algorithm is an optimum version of layered decoding [18] for QC-LDPC because the column weight is at most one in one layer. It is the most adaptable decoding algorithm for QC-LDPC, having the fastest speed of convergence up to now. For 802.11n and 802.16e, *Belief Propagation* algorithm often needs 20 iterations to achieve 10^{-6} BER (**B**it **E**rror **R**ate). However, *row-update message passing algorithm* only needs 10 iterations to achieve 10^{-6} BER. Besides, its grouped layer property provides highly parallelism on VLSI implementations. It not only provides throughput enhancement but also flexibility for adaptive code rates and lengths for future wireless communication systems. However, a configurable data-path for variable code rates and lengths is the main design bottleneck, hence a high flexible permutation design is proposed to overcome different size of Z_f (totally 22 modes). The detail design with row-update is discussed in next chapter.

3.3 Design Considerations

Figures 2-11~2-14 present the advantages to adopt RMP algorithm. This sub-section will simulate some parameters for design considerations, including min-sum algorithm, fixed-point simulations and number of iteration, etc. According to the trade-off between error-correction performance and hardware complexity, the appropriate parameters are chosen for VLSI implementations.

3.3.1 Normalization Factor

The CHK operation is listed as (3.70). Min-sum algorithm is adopted to approximate the nonlinear function $\psi(x) = -\log(\tanh(|x|/2))$, so that a normalization factor is needed to compensates the performance degradation due to inaccuracy approximation. However, it is difficult to find an adequate factor depending on coding type, code rate and etc. A self-compensation technique [19] is proposed to improve the performances. To ease implementation design, a fixed normalization constant is chosen by exploring some factors, 0.625, 0.75, 0.875, 1 which performance is close to RMP algorithm without min-sum approach (the theoretical result).

$$\begin{aligned} r_{ji} &= s_{ji}\left(\min_{i'\in N(j)\backslash i}(|q_{i'j}|)\right)\times\beta;\quad \beta=0.75 \\ &= s_{ji}\left(\min_{i'\in N(j)\backslash i}\left(\left|\underbrace{\sum_{j'\in M(i)\backslash j} r_{j'i} + \underbrace{p_i}_{\text{channel value}} - \underbrace{r_{ji}}_{\text{last old message}}}_{\text{bit node update}}\right|\right)\right)\times\beta \end{aligned} \tag{3.70}$$

Figure 3-15 shows the simulation with factors, 0.625, 0.75, 0.875 and 1 (without normalization factor).

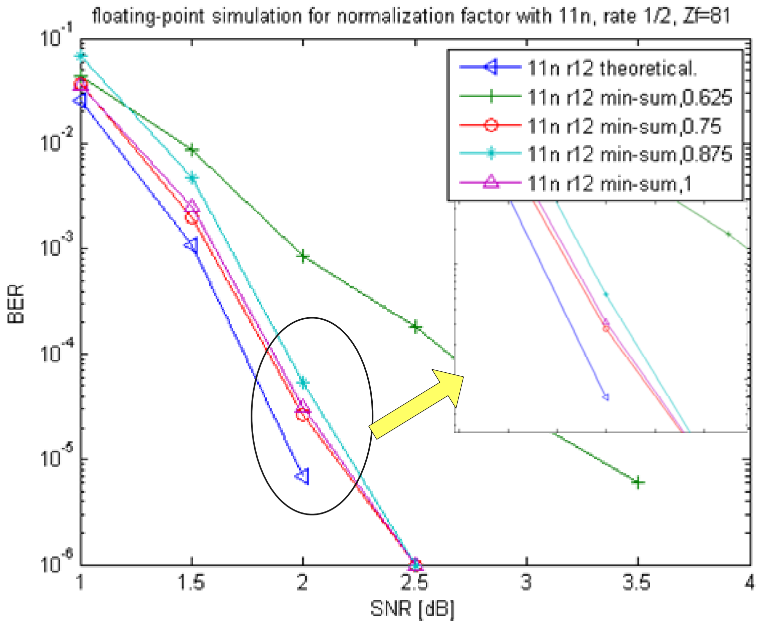


Figure 3-15：Simulation for normalization factor simulation (802.11n, floating point, rate 1/2, block length 1944 bits, BPSK, AWGN)

0.75 is the optimum factor most close to the theoretical result. A formula $x \times 0.75 = x \times 0.5 + x \times 0.25 = x >> 1 + x >> 2$ is implemented for HDL (hardware description language).

3.3.2 Bit Width for Hardware Cost

In hardware implementation, we have to decide how many bits to present one data. (Integer, Fraction) denotes bit widths of integer part and fractional part, respectively. Figure 3-16 shows fixed-point simulation with (6,0), (6,1) and (6,2). Figure 3-17 shows fixed-point simulation for different integer parts, (6,0), (7,0) and (8,0). The theoretical result means floating point simulation with RMP algorithm with min-sum. The object code simulates code rate 1/2, codeword length 1944 bits in 802.11n.

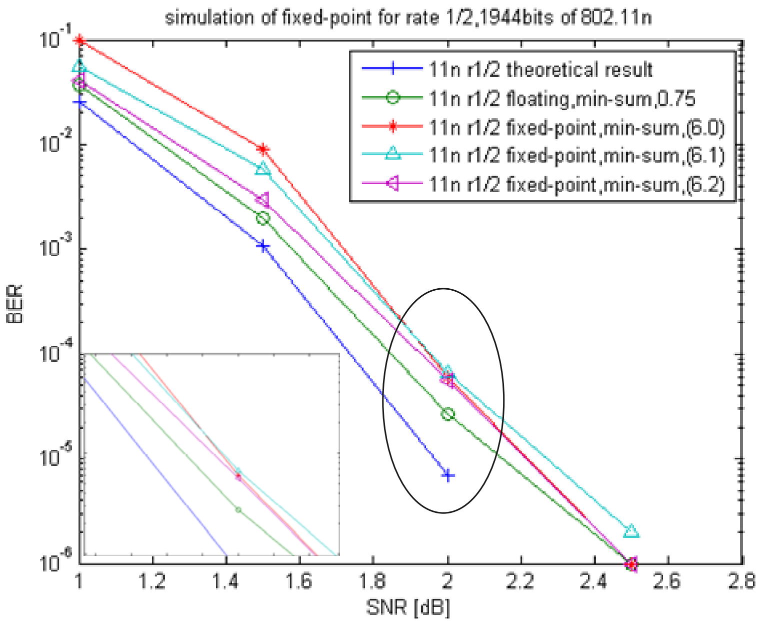


Figure 3-16：Fixed-point simulation with fixed 6 bits integer (802.11n, 10 iterations, rate 1/2, block length 1944 bits, BPSK, AWGN)

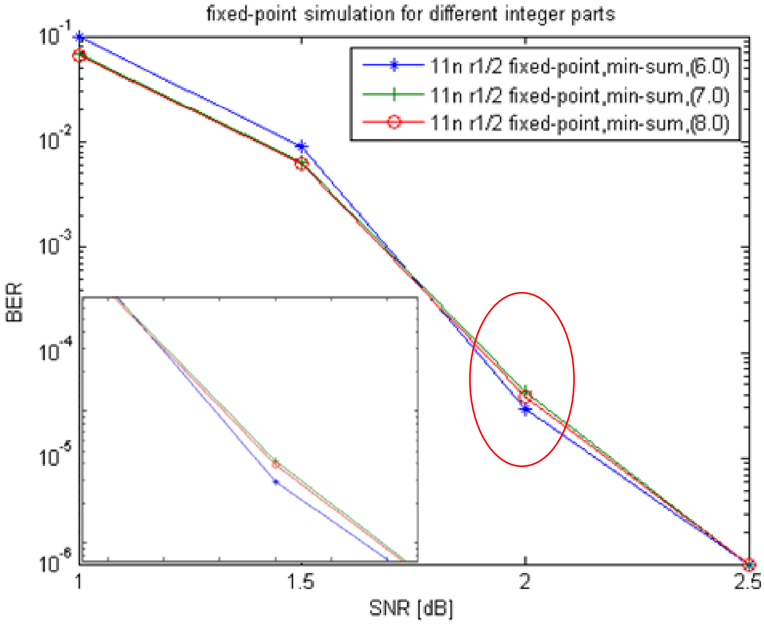


Figure 3-17：Fixed-point simulation for integer part

Of course, larger bit widths can approach floating simulation curve more. However, a trade off between bit width and performance should be considered. Finally, we choose 6 bits for integer parts and 0 bit for fraction part, (6,0) because (6,0) is good enough.

3.3.3 Number of Iteration

In the decoding process, maybe an error bit string causes that codeword can't be corrected in finite number of iteration, so that we must set the maximum number of iteration to stop infinite iterations. How many number of iteration is the adequate number to have an acceptable data rate and error-correction performance? We simulate the fixed-point simulation with different number of iterations, 5, 8, 10, 15, and 20 for 802.11n, rate 1/2, length 1944 bits as shown in Figure 3-18.

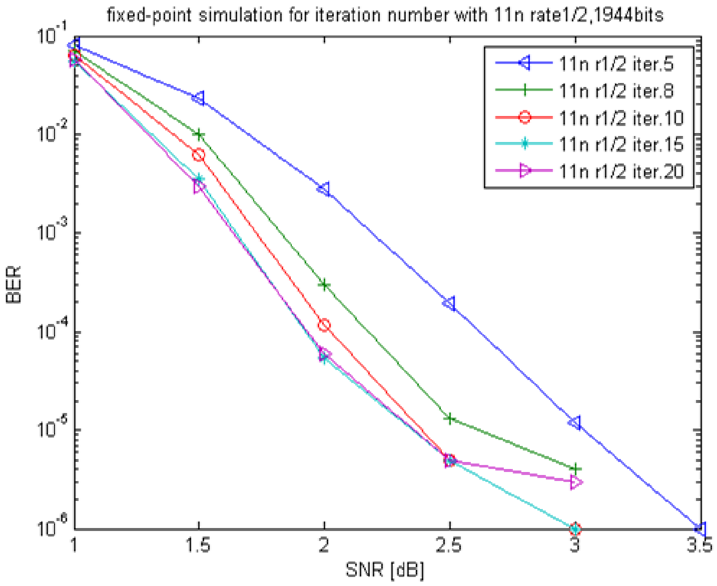


Figure 3-18：Simulation for iteration number, 5,8,10,15,20

(802.11n, floating point, rate 5/6, block length 1944 bits, BPSK, AWGN)

Of course, the simulation curve with more number of iterations can own well error correction performance. However, the data throughput has spoiling effect. Except for data throughput, 10-time and 20-time iteration performances are almost close, because of its performance saturation. Consequently, limited 10-iteration is good enough for high decoding speed and acceptable error correcting performance.

We summarize the parameters discussed in the prior sub-sections as Table 3-1.

Table 3-1： Parameters summary

Decoding algorithm		Min-Sum RMP algorithm
Max. Iteration		10
Normalization factor		0.75
Bit width	Integer	6 bits
	Fraction	0 bit

3.4 Implementation Issue

We present the decoding algorithm above. Some implementation issues to design a configurable architecture should be considered as follows,

1. A flexible permutation design is a very important bottleneck to merge 22 modes in 802.11n and 802.16e corresponding to different codeword lengths. How to design a configurable data-path merging 22 types of hardware units.

2. We group Z_f rows into one layer and decode them in parallel. 96 PEs are implemented to process data in parallel and merge other Z_f's (24~96). When decoding a short codeword length, low hardware utilization is necessary to be enhanced. Multi-codeword decoding technique is proposed to solve the problem.

3. In order to avoid memory access confliction, schedule is applied on *row-update message passing algorithm* in VLSI implementation. There are 31%~44% enhanced throughput with scheduling.

4. Number of iteration plays an important role in error-correction performance and date rate. Hard decision based early termination is implemented to reduce redundant iterations.

In chapter 4, we will introduce our proposed configurable decoder architecture and related solutions in detail.

Chapter 4 Architecture Design and Implementation

For a trade-off between hardware complexity and performance, the partially parallel architecture is designed with *row-update message passing algorithm* with min-sum. We will introduce the design analysis, overall architecture, related functional block and techniques to solve some design bottlenecks and enhance decoding performance. For example, flexible permutation is proposed to merge all types of Z_f's and multi-codeword decoding technique is adopted for preserving hardware utilization. Scheduling, hard decision based early termination and other techniques will be introduced and discussed as following content. The LDPC decoder with a core size of $2.14\times2.14\ \text{mm}^2$ is implemented in TSMC 0.13 μm CMOS technology. The detail post-layout simulation results including throughput, area, and power and some comparisons with state-of-arts will be presented in the final section.

4.1 Architecture Design

Fully parallel architecture can achieve a maximum throughput but the lack of flexibility and its large area is the major problem in current and future wireless systems that require support for adaptive code rates and codeword lengths. A partially parallel not only provides a trade-off between hardware complexity and throughput but also high flexibility for different code rate and lengths. Thus, we adopt a partially parallel architecture to match the QC-structure with variable Z_f. However, how many parallel levels for functional units should be considered with adopted decoding algorithm? Because the maximum Z_f for 802.16e and 802.11n is 96, 96 processing units in parallel is instinctively the best choice to merge all cases. Hence, we group Z_f rows into one layer which can be view as a component code and decode layers in sequence. There are 12 layers for code rate 1/2, 8 layers for code rate 2/3, 6 layers for code rate 3/4 and 4 layers for code rate 5/6. On the other hand, there are $24\times(1-R)$ layers for code rate R. The parallel 96 processing units are the best numbers of PEs to achieve maximum data rate without losing error-correction performance, although it is a over design when decoding a short codeword ($Z_f<96$). However, multi-codeword decoding technique discussed later is proposed to preserve hardware utilization.

4.1.1 Decoding Flow

Figure 4-1 shows the decoding flow chart with row-update message passing algorithm. The rows of parity check matrix are grouped into a layer and updating in sequence as shown in Figure 4-2. Initialization includes the input receiver and setting for some parameters. “shift message” means the operation to shift data according to a shift amount, because data has be shifted to appropriate permutation for next layer.

Check node update and bit node update denote the corresponding CHK (4.1) and VAR (4.2), respectively. When decoding one layer, PEs first read *d* degree data according to its row degree distribution from memory, then update check node/bit node. And then shift the updated message, store data and temperate parameters back into corresponding memory, respectively. The time of next layer update can be overlapped with storing operation by appropriate pipelining. After finish 24×(1-R) layers update, the decoder will stop when a valid codeword is found, otherwise, it moves toward the next iteration. However, if the number of iterations exceeds a predefined value, the decoder claims decoding failure and terminate the decoding process. We will discuss detail the corresponding architecture design in next sub-section.

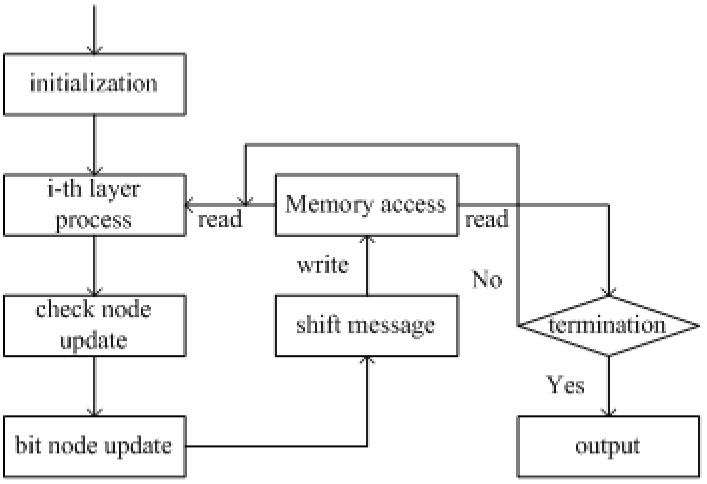


Figure 4-1：Decoding Flow Chart

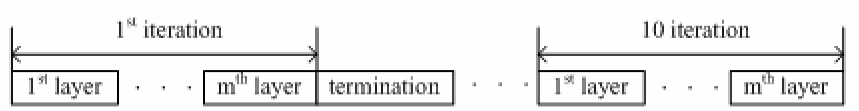


Figure 4-2：Decoding process diagram for *m*×*n* parity check matrix

$$r_{ji} = s_{ji}\beta \times \min_{i'\in N(j)\backslash i} |q_{i'j}|, \qquad \beta = 0.75, \;\; s_{ji} = \prod_{i'\in N(j)\backslash i} \operatorname{sgn}(q_{i'j}) \tag{4.1}$$

$$q_{ij} = p_i + \sum_{j'\in M(i)\backslash j} r_{j'i} = \sum_{j'\in M(i)} r_{j'i} + p_i - r_{ji} \tag{4.2}$$

4.1.2 Overall Architecture Design

Figure 4-3 shows the overall decoding architecture. The “input buffer” block, having data bandwidth 96×6 bits, is the input buffer serially receiving input data. The “permutation” denotes the interconnection design to permute data between successive layers.

ROM tables are constructed by prior analysis for defined parity check matrices in 802.11n and 802.16e to store shift amounts and addresses defined in parity check matrices, respectively. The “Proc.#1~96” denote 96 processing units for CHK/VAR update. It serially accesses *d* data according to address ROMs (*d* is the row degrees). After CHK/VAR update, messages are fist shifted with related shift amounts for next successive layers by “permutation”, then stored back into original memory. It immediately updates each layer’s information for next layer’s update.

“Beta_ram” is the temperate memory that stores relative parameters from last iteration, Beta1, Beta2, index, and sign value. “Beta1” means the normalized first minimum, $\beta\times\min1$ (min1: fist minimum). “Beta2” means the normalized second minimum, $\beta\times\min2$ (min2: second minimum). “Index” denotes the index for the first minimum value. Sign value indicates sign part of updated messages. When processing one layer, we need to subtract older messages and then produce new messages and related parameters, index, Beta1, Beta2, sign, store back into the memory with same addresses. “Termination” execute decoding stop if a valid codeword is found or the number of iterations exceed a predefined value, 10, otherwise move toward next iteration.

A roughly description presents basic operations and decoding flow for the proposed architecture. Then we discuss the detail functional block respectively in continuous sub-sections.

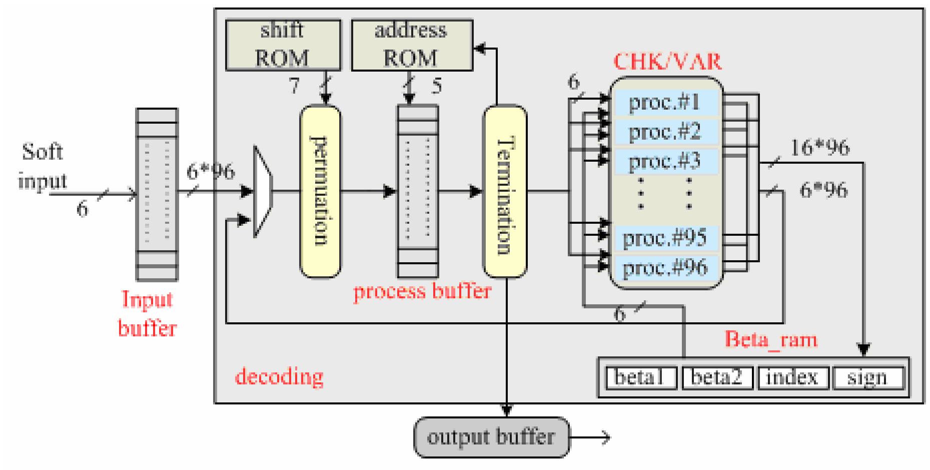


Figure 4-3：Overall architecture

- Input/Process Buffer

Suppose that N denote codeword length and N_1 represent the total number of edges in the Tanner graph of the LDPC code. From (4.1) and (4.2), the sum-product algorithm, within an iteration, requires to store the message traveling on each edge. And, apart from these, the N channel extrinsic data have to be stored. Thus, the memory required from iteration to iteration is $(N+N_1)\times b$ for conventional BP algorithm, where b is the bit width of the fixed-point number. Since N_1 is generally larger than N greatly, the message passing on each edge can be generated by updated VAR indeed, and, hence, when in VLSI implementation, the designer usually uses $2N\times b$, in stead of $(N+N_1)\times b$, memory for message of node not edge. For efficient memory saving, a value-reuse architecture is proposed with RMP decoding algorithm. We first store channel value, $\frac{2r_j}{\sigma^2}$ from "input buffer"into memory, "MS" in Figure 4-3 and "Input buffer" is capable of receiving next block codeword. Thus, there are only $N\times b$ sizes needed for process buffer so that it save almost half size of memory compared to the conventional BP algorithm. "In_buffer" is a single port memory and "MS" is a dual port memory in order to read and write simultaneously for scheduling issue. They are both 24 entries with 576 bits (96×6) per entry. Figure 4-4 shows the

"In_buffer" and "MS" memory block diagram.

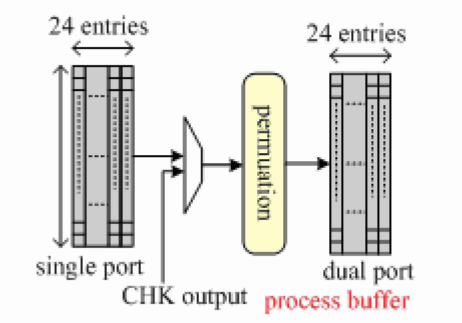


Figure 4-4：Data path from "in_buffer" to "MS"

Flexible Permutation Design

A configurable data-path is a design bottleneck for adaptive code rates and codeword lengths. In 802.11n and 802.16e, LDPC codes define identity right shift block matrix according to different code lengths and shift amounts. Total 22 modes need to be considered in the design, including 3 modes for 802.11n and 19 modes for 802.16e. Figure 4-5 illustrates a simple 3×6 parity check matrix with 6×6 block matrix, Z_f=6. The numbers in parity matrix denote shift amounts and all the shift amounts are smaller than 6 because Z_f is equal to 6.

For the example, shift amounts shown in Figure 4-5, we expand block matrices with related shift amounts and a size of 6×6 matrix in Figure 4-6. When processing first layer, we have to read data from column addresses 1, 3, 5. After CHK/VAR operations, we have to permute the output results and restore back the same addresses, 1, 3, 5. Then, process addresses 2, 4, 6 and continue decoding until a valid codeword is found or termination stop. In hardware, we store 6 data in registers in sequences according to its shift amounts. However, permutation is needed to cyclically shift data for required arrangement of next processing layer. We store differences of successive shift amounts in column direction instead of absolute values defined in parity check matrices in ROM tables. For example, the sets of difference, (1, 1, 2) are shown as

Figure 4-5. The trick can provide us to only permute data one time when decoding layer-by-layer. On the other hand, it does not need inverse permutation except for output operation.

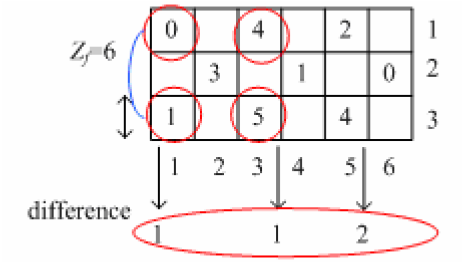


Figure 4-5：3-by-6 pararity matrix with Z_f=6

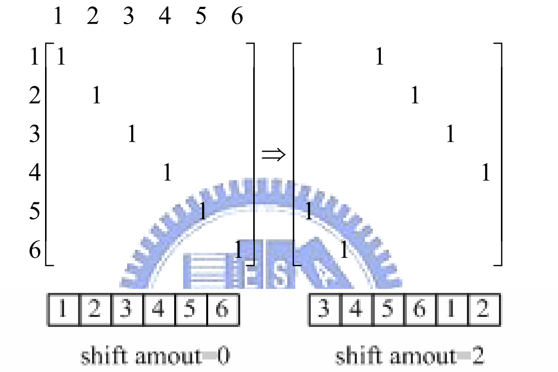


Figure 4-6：6-by-6 identity matrix with shift amounts 0, 2, respectively

Figure 4-6 shows shift amounts and their data sequences in registers. We define that L means L 6-bit data registers are constructed in hardware and Z_f denotes a block matrix size $Z_f \times Z_f$ related to its codeword length. When L is equal to Z_f, it is a simple barrel shift problem that can be solved easily by design ware or logic multiplexers. An L 6-bit data shifter can support **0~L-1** shift amounts of cyclic shifter. However, the hardware cost is too large to implement 22 types of permutation units. We want to construct 96 6-bit data shifter to merge all variable defined shift sizes Z_f, from 24~96. It produces another implementation issue with a dynamic shift size when L is not equal to Z_f. Figure 4-8 shows a simple example of the problem, $L \neq Z_f$ with shift size, Z_f=6 and L=8,. Totally there are 22 types of Z_f from 24 to 96 in 802.11n and 802.16e according to different code rates and lengths.

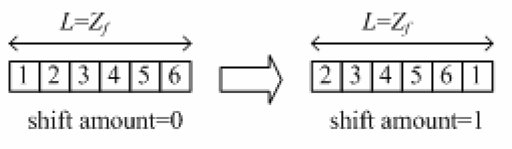


Figure 4-7：Register arrangement for shift amounts 0, 1 with shift size 6

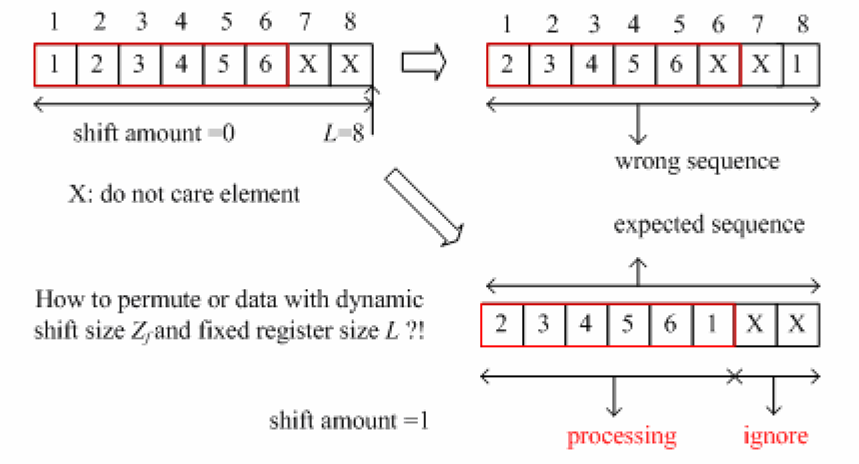


Figure 4-8：Example of shift size Z_f smaller than maximum register size L

Configurable, point based permutation is proposed to solve the bottleneck by enhancing enable bits of patent [20] and the main rationale is illustrated as Figure 4-9. The head and tail points the available data length, Z_f. The pointers rotate left, like a cyclic buffer with a shift amount S=shift amount. The distance between updated pointer head (head’) and old pointer head indicates desired parts. Likewise, tail and tail’ do so. An expected data sequences can be available by combining the left part and right part from L-data, respectively. Figure 4-10 shows an example of permutation for this idea with L=8, shift amount=1, Z_f=6. Because the design supports a multi-codeword decoding mode, a multi-codeword shifter is needed to be supported. The design also supports multi-codeword permutation.

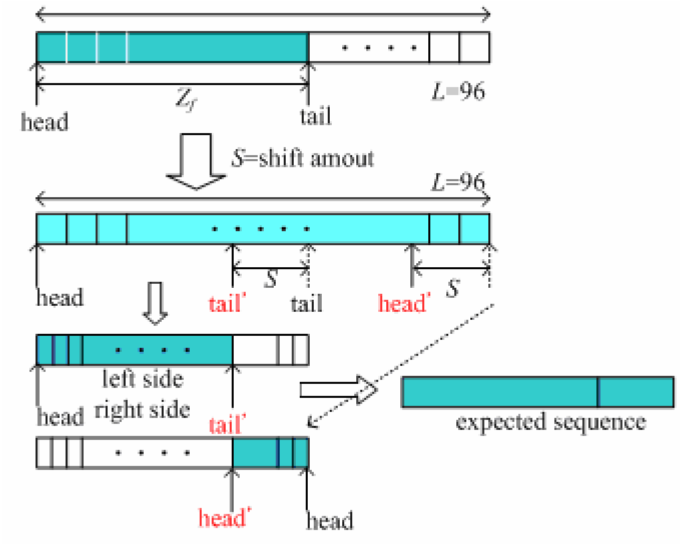


Figure 4-9：Permutation design with head and tail pointers

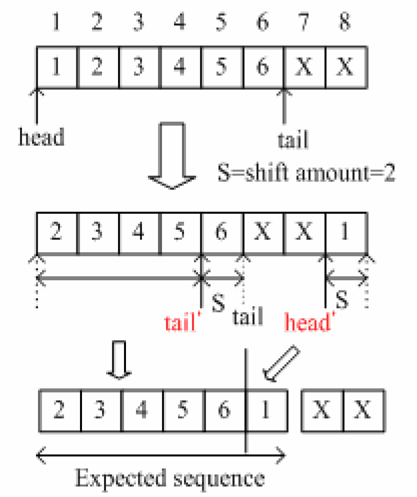


Figure 4-10：Permutation for maximum size 8 and shift size 6 and shift amount 1

Number 96 can be presented in binary 7 bits, so that shift amount is 7-bit in Figure 4-11. The hierarchical architecture of the logic barrel shifter to permute 96 6-bit data is shown in Figure 4-11. Although the latency increases with more logic levels, the area is scalar down. Three level multiplexers are the optimum choice for a trade off between area and timing.

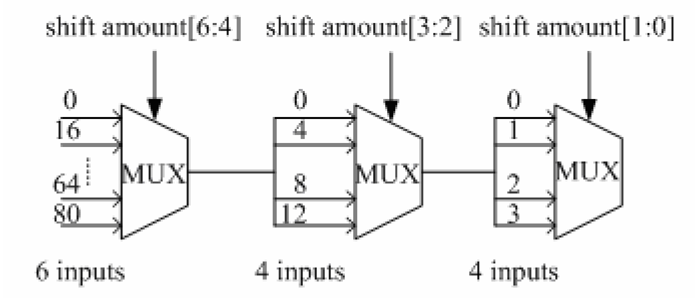


Figure 4-11：Three levels of logic barrel shifter for 96 data

Without pipelining, we optimize the design. The term "Bit" denotes number of bit for one data. We list the area comparison shown in Table 4-1 for interconnection design. We support total 22 modes and still have well performance on area and timing. The timing constraint is 2.5 ns in synthesis level.

Table 4-1：Comparison with design [21] for area

	Applications	Area(um^2)	Bit	Technology
[21]	11n, 3 modes	20471	7	0.13 μm
proposed	11n&16e, 22modes	21852	6	0.13 μm

- Processing Units (CHK/VAR)

In processing units, we execute *SPA* (**S**um-**P**roduct **A**lgorithm) for CHK and VAR updates for row-update message passing algorithm with min-sum. Equations 4.3 and 4.4 present CHK operation and VAR equations. We combine them then derive it as equation (4.5) and overlap the successive CHK/VAR operations to enhance throughput by pipelining the stages. We simultaneously process Z_f rows and access *d* degree messages according to its degree distribution. We store information on nodes of bipartite graph not that on edges. If there are *d* degrees for one layer, it must store d data when considering edge information, hence, it needs a large size of memory. By storing the information on nodes, memory size is reduced to almost 1/2 compared to information on edges. An efficient memory saving method is described in prior section, architecture design.

$$r_{ji} = s_{ji}\beta \times \min_{i'\in N(j)\backslash i} |q_{i'j}|, \qquad \beta = 0.75,\ \ s_{ji} = \prod_{i'\in N(j)\backslash i} \operatorname{sgn}(q_{i'j}) \tag{4.3}$$

$$VAR:\ q_{ij} = p_i + \sum_{j'\in M(i)\backslash j} r_{j'i} = \sum_{j'\in M(i)} r_{j'i} + p_i - r_{ji} \tag{4.4}$$

$$\Rightarrow\ r_{ji} = s_{ji}\beta \times \min_{i'\in N(j)\backslash i} \Big|\ \underbrace{\Big(\sum_{j'\in M(i)} r_{j'i} + p_i\Big)}_{\text{MS value into MS memory}} - r_{ji}\Big|, \qquad \beta = 0.75 \tag{4.5}$$

We rewrite equation (4.3) as (4.6) by index, Beta1, Beta2, and sign value. “min1” denotes the first minimum value of the set. “min2” denotes the second minimum value of the set. Beta1 and Beta2 are the scaling min1, min2 by β 0.75. Index means the index for the first minimal value of the set. By min-sum, we only store Beta1, Beta2, index, sign, and MS values instead of information on each edge. It reduces complexity of computation and achieves efficient memory saving. Figure 4-12 shows the partially parallel data-path for from “MS” to “processing units”.

$$CHK:\ r_{ji} = \beta \times \prod_{i'\in N(j)\backslash i} \operatorname{sgn}(q_{i'j}) \times \begin{cases} \min 1\ ;i' = \text{index} \\ \min 2\ ;i' \neq \text{index} \end{cases}$$

$$;\ \ \min 1:\ \min_{i'\in N(j)} |q_{i'j}|\ ,\ \ \min 2:\ \text{second}\ \min_{i'\in N(j)} |q_{i'j}| \tag{4.6}$$

$$;\ \ \text{index: order of minimal value}\ q_{i'j}$$

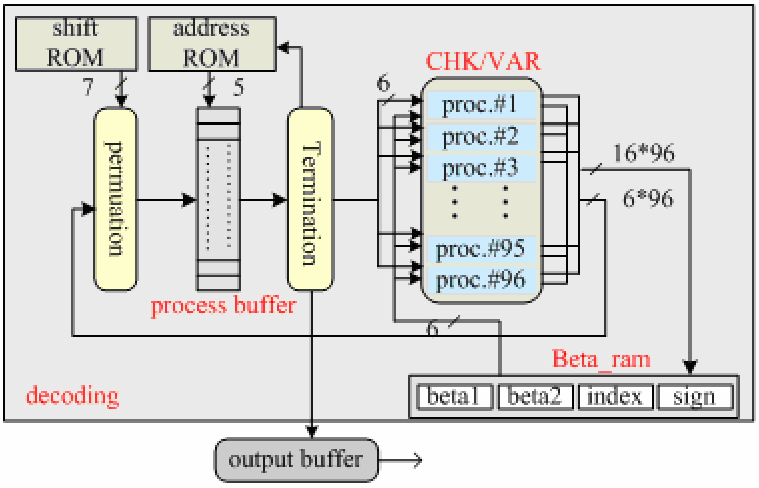


Figure 4-12：Data path from MS to CHK

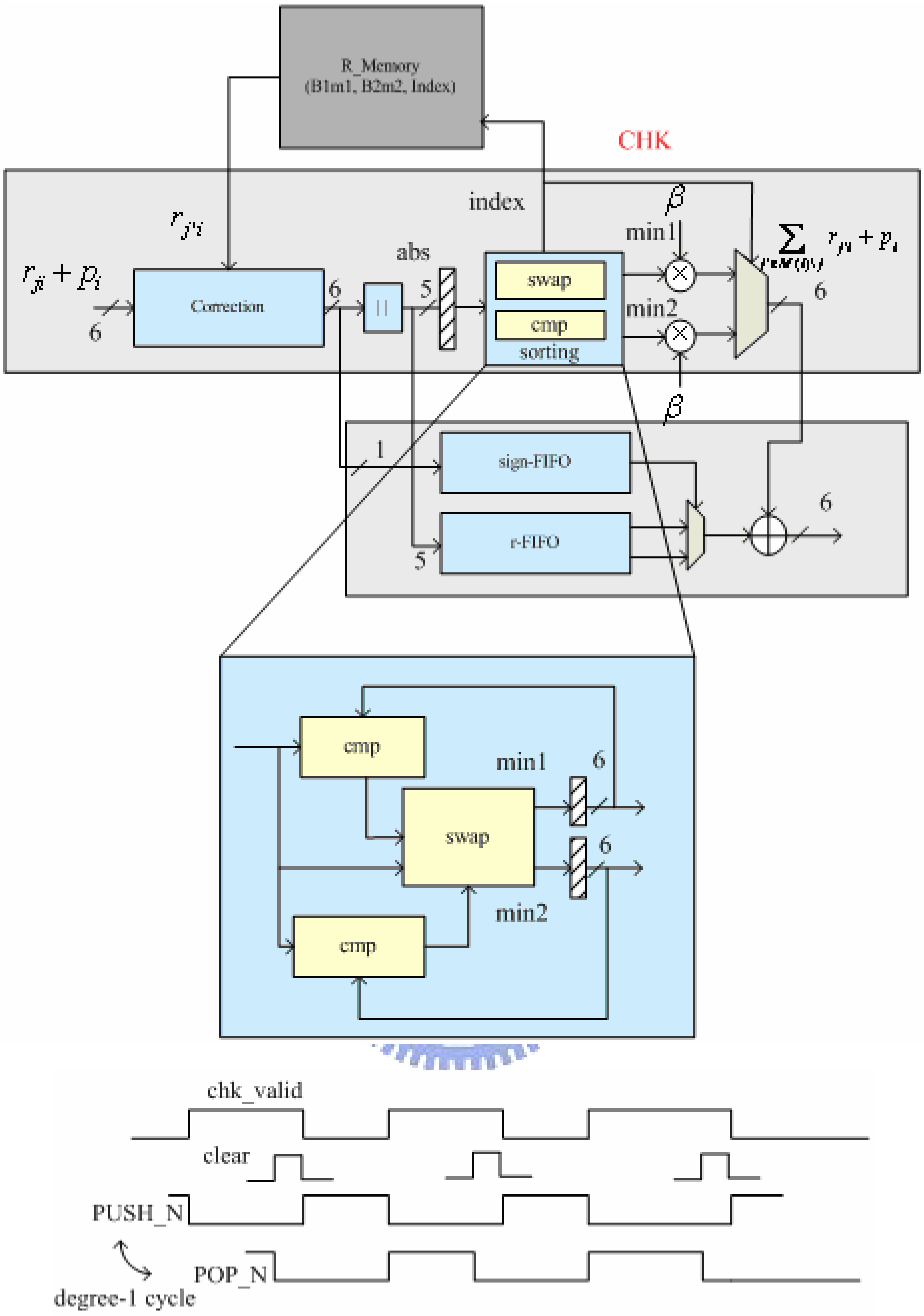


Figure 4-13：Architecture for processing units (CHK/VAR)

We map equations (4.4) ~ (4.6) to CHK/VAR in Figure 4-13. The correction block means that input message must subtract old stored information with index, Beta1, Beta2, sign value from "R_Memory" and it is mapped to subtraction notation of equation (4.5). For CHK, we first have the absolute operation (abs), $|x|$ in order to sort the magnitude value to find out the first minimum value, min1 and second minimum value, min2, and then scaling the minimum value with β=0.75 to produce

CHK output. The sorting block needs two comparators and a swap operation for minimum value. There are two FIFOs (**F**irst **I**n **F**irst **O**ut) to store sign part of corrected value and corrected value, “crt_out”, respectively. The VAR function receives PUSH_N and POP_N control signals to control FIFOs and cumulate newer updated values with outputs of CHK function. The updated VAR outputs will be permuted for next layer and then stored back processing buffer. The related parameters, index, Beta1, Beta2, and sign back are also stored into “Beta_ram”. Figure 4-14 shows the architecture of “Beta_ram”.

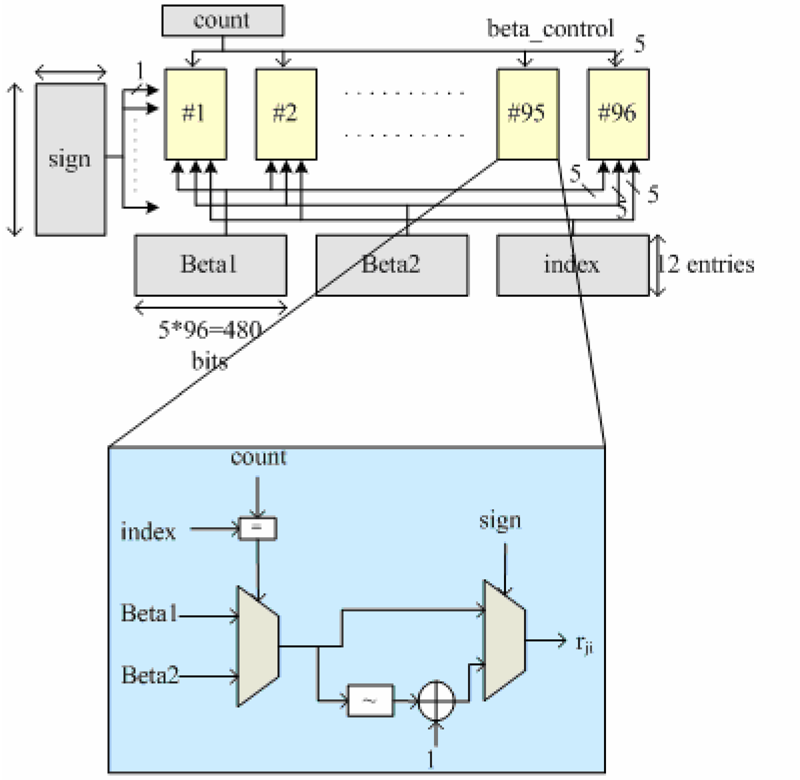


Figure 4-14：Block diagram of “Beta_ram” memory and controls

Some control signals are produced from “termination” block, and it is an important problem for high fan-out loading when running the post-layout simulation. We must duplicate control signals to reduce wire loadings for the loading is too heavy, hard to drive 96 function units.

● Multi-codeword Decoding Technique

For different code rates and codeword lengths, it has the related Z_f because codeword length n= $Z_f \times 24$. 96 function units are implemented for partially parallel decoding because of maximum Z_f, 96 to merge other Z_f's. The data-path process 96 6-bit data simultaneously, even if decoding a short codeword length, $Z_f<96$. Thus, it causes spoiling hardware utilization for idle redundant function units. We proposed the multi-codeword decoding technique to preserving hardware utilization. We divide 96 function units into 2 parts, upper and lower parts in Figure 4-15. The upper part receives first block and lower part process next block. Obviously, it has a constraint that codeword length must be less 1152 bits ($Z_f<48$) in multi-codeword mode. Of cause, the permutation design should be also modified to support multi-codeword codeword permutation, and we present the rationale in Figure 4-16. Because we process the same sequences of non-zero block matrices, just different block codeword, and thus the shift amounts are same. So the proposed method has no more hardware cost and only need some control circuit.

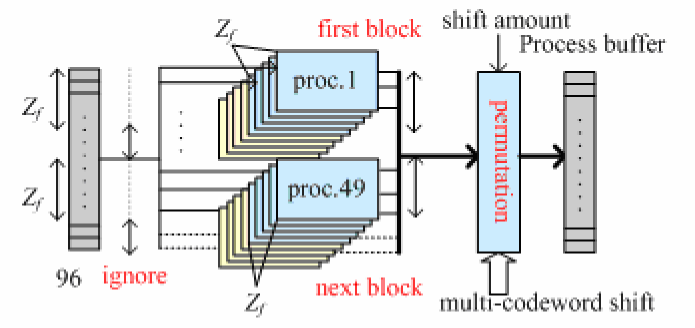


Figure 4-15：Block diagram for multi-codeword decoding technique

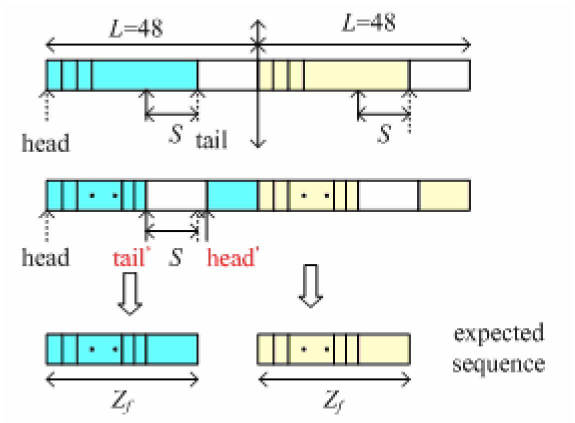


Figure 4-16：Point based permutation for multi-codeword

The advantage of multi-codeword decoding is not only to increase hardware utilization but also data rate. We summarize the throughputs for 802.11n and 802.16e with post-layout simulation results at 333 MHz as Figure 4-17 and 4-18, respectively. We can observe that throughput of a short codeword length (Z_f<48) is double of original single decoding, e.g. the throughput of length 648 (Z_f= 27) is 240Mb/s, same as that of length 1296 (Z_f=54) for any code rates in 802.11n, There are 8 types (24, 27, 28, 32, 36, 40, 44, 48) of Z_f's enhanced among totally 22 modes for 802.11n and 802.16e. We have the decoding throughput of 240~506 Mb/s for 802.11n and 213~590 Mb/s for 802.16e. The multi-codeword decoding technique doubles the throughputs and provides more design spaces for low power consideration. Later we summarize the post-layout simulation results and discuss performance comparison.

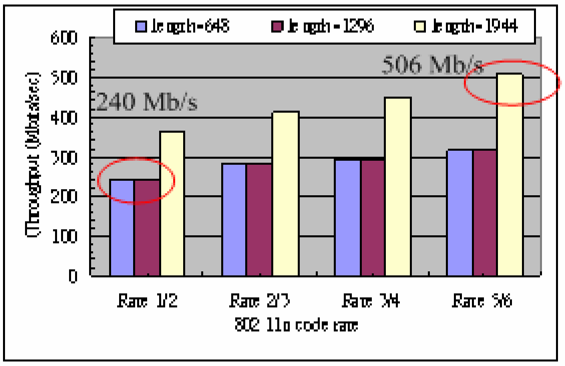


Figure 4-17：Throughput for 802.11n, full code rates at 333 MHz

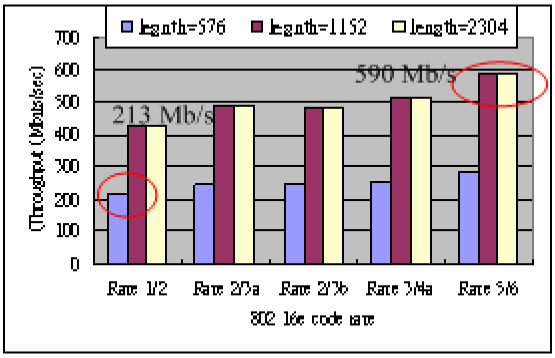


Figure 4-18：Throughput for 802.16e, length 576, 1152, 2304 at 333 MHz

- Dynamic Early Termination

An analysis for number of iteration has been discussed in chapter 3. We set maximum number of iteration to be 10 for a trade-off between error-correction performance and throughput. Early termination mechanism is proposed to reduce the redundant number of iteration and achieve low power consumption. It is generally to check the traditional parity check constraint, $\boldsymbol{Hx}^T$=0, if satisfying the equality, decoder terminates the process or continues the iteration until 10-iteration is achieved. However, it is impractical to implement the matrix multiplication in hardware. For example as matrix (4.7), the parity check constraints (4.8) have a large overhead to store the random constraints and the time to check all constraints is also hard to be handled. However, hard decision based early termination is implemented to provides an easy method to verify the valid codeword. Moreover, its hardware cost is less than the parity checker, $\boldsymbol{Hx}^T$=0. We store the sign part of *LLR* and compare successive decoded outputs, if same stop decoding otherwise, continues the iteration until 10-iteration is achieved.

Of course, there is performance degradation compared to the checker based on parity constraints. It will produce errors when soft message changes but its hard decision doesn’t change. Figure 4-18 shows the simulation with early termination based on parity check constraints, $\boldsymbol{Hx}^T=0$ and hard decision checker. The "theoretical." curve denotes the decision based on parity check constraints. The simulation environment is BPSK, AWGN, fixed-point and object is code rate 1/2, length, 1944 and 2304 bits in 802.11n and 802.16e, respectively. There are a little acceptable performance degradations when considering of implementation and enhanced throughput.

$$H=\begin{bmatrix} 0&0&0&1&1&0&0&1&0&1\\ 1&1&0&0&0&1&0&0&0&1\\ 0&1&0&1&0&0&1&0&1&0\\ 0&0&1&0&1&1&1&0&0&0\\ 1&0&1&0&0&0&0&1&1&0 \end{bmatrix} \qquad (4.7)$$

$$\boldsymbol{Hx}^T=0 \Rightarrow \begin{cases} c_0:\ x_3+x_4+x_7+x_9=0\\ c_1:\ x_0+x_1+x_5+x_9=0\\ c_2:\ x_1+x_3+x_6+x_8=0\\ c_4:\ x_2+x_4+x_5+x_6=0\\ c_5:\ x_0+x_2+x_7+x_8=0 \end{cases} \qquad (4.8)$$

$$with\ \boldsymbol{x}=[x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9]$$

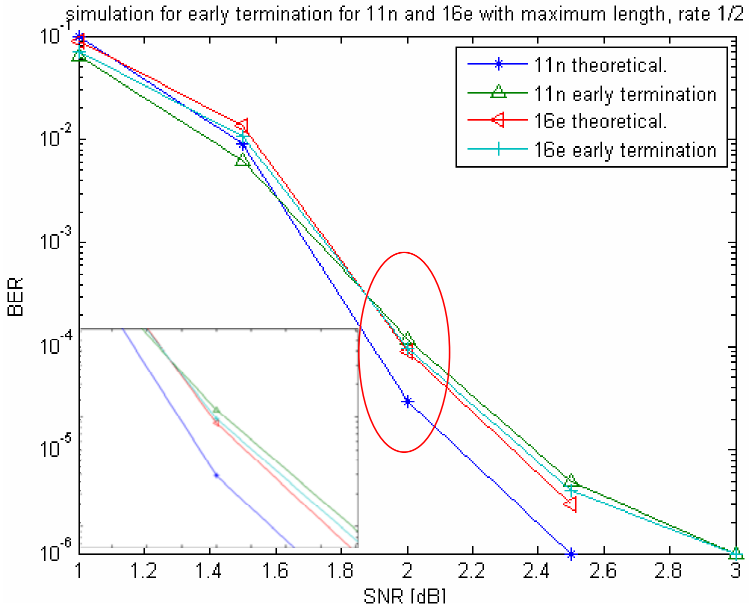


Figure 4-19：Hard decision based early termination v.s parity check constraints

Except for purely early termination, we provide users dynamic termination mechanism to control whether turn on early termination or not. Because the early termination cost latency in decoding process, 10-iteration is a better choice in bad transmission channel or high code rates for error-correction performance. However, it is suggested to turn on early termination in low code rates or well transmission channel for reducing decoding latency. Figure 4-20 shows the difference of fixed-iteration and early termination. The configurable control lets the design more flexible for adaptive code rates and different transmission environment.

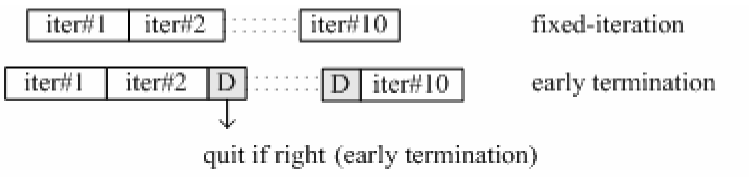


Figure 4-20：Process of dynamic early termination

● Scheduled RMP Algorithm

The rationale of row-update message passing algorithm is to update variable nodes as soon as possible for that the check node can access newer information. However, the later layer has to wait for the memory updated by the last layer. On the other hand, it exist intra- and inter-iteration precedence constraints (or data dependence) and the layers work one after another. The fact makes it difficult to design a high throughput as well as a high hardware utilization LDPC decoder. Figure 4-20 shows that non-scheduling v.s scheduling. Suppose that there are layer1 with column blocks, 1, 2, 8, 9, 12 and 13 and layer2 with column blocks, 0, 2, 8, 9, 15 and 16 needed to be processed. If layer1 updates check nodes in sequences, 1,2,8,9,12,13, and sequences 1,2,8,9,12,13 for layer2. However, there is a large of latency to start layer2 after finishing layer1. To overcome this problem, the overlapped operations with a systematic scheduled RMP algorithm is proposed for QC-LDPC. We re-schedule the column addresses for memory access to avoid memory confliction. If a dual port memory is available, instead of sequential operations from layer to layer, the next layer can start before the previous layer is finished.

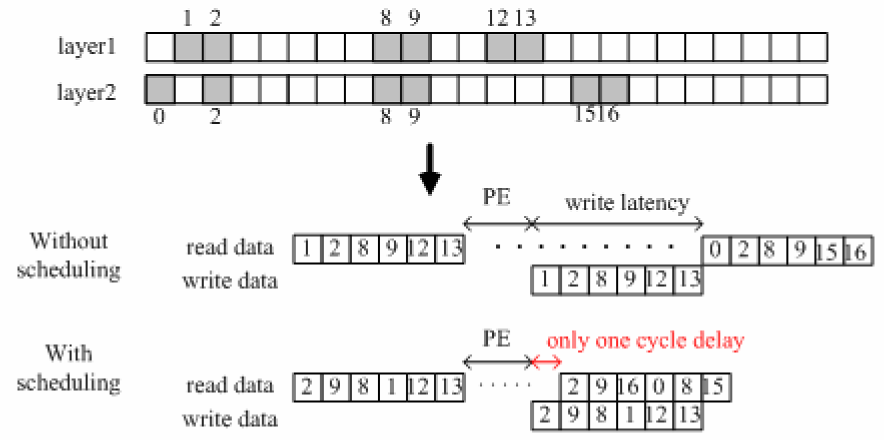


Figure 4-21：Example of scheduling

Suppose that both CHK update (including read data from memory) and VAR update (including write data back to memory) need C_{CHK} and C_{VAR} cycles to finish computing one non-zero block matrix of size $Z_f \times Z_f$. For an $m \times n$ QC-LDPC with m_b non-zero matrices for each row, it needs $(C_{\mathrm{CHK}}+C_{\mathrm{VAR}}) \times m_b \times m$ clock cycles to finish one iteration by original RMP algorithm without scheduling. Suppose further that the next layer decoding can be initiated only after w cycles' computations of VAR update of the previous sub-iteration, where $1 \leq w \leq C_{\mathrm{CHK}}$, thus comparing with the conventional RMP algorithm, as shown in Fig. 4-22, the throughput gain with scheduling is

$$\frac{(C_{CHK}+C_{VAR}) \times m_b \times m}{(C_{CHK} \times m_b + w) \times (m-1) + (C_{CHK}+C_{VAR}) \times m_b} \tag{4.9}$$

In our design for both 802.16e and 802.11n systems, which will be described in the next section, the throughput gain of the scheduled RMP algorithm is about 32~44% as shown in Table 4-2 for 802.16e.

By-passing technique is another method to solve the problem, but impractical because of irregular addresses. Thus, scheduling technique is still the better suggestion to overlap the operations.

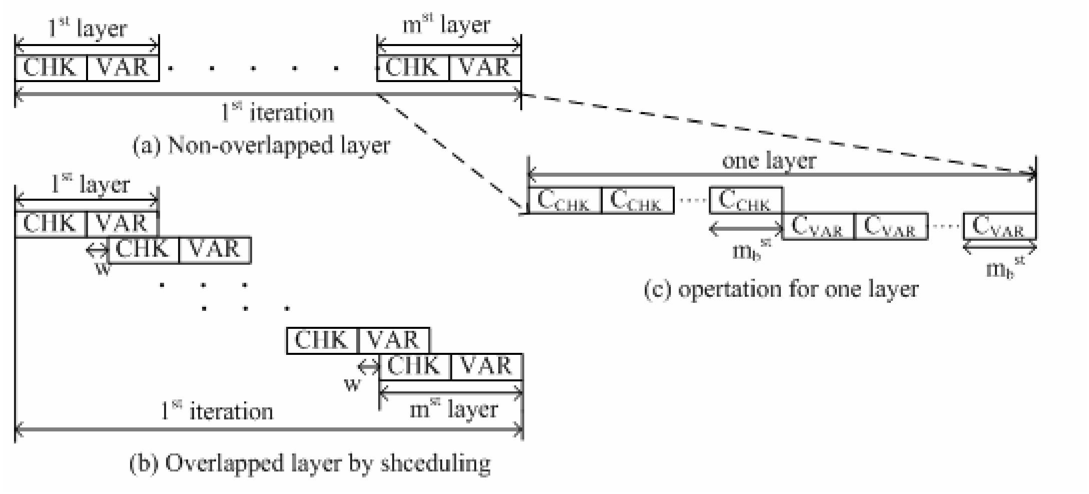


Figure 4-22：Overlapped v.s non-overlapped

Table 4-2：Throughput enhancement with scheduling for 802.16e

Code rates	1/2	2/3A	3/4A	5/6
Enhancement	44%	32%	44%	41%

- **Memory Arrangement**

In this sub-section, we will summarize memory arrangement, including RAMs and ROMs. “M1_24X72_inbuff” denotes a single port memory with 24 entries, 72 bits per entry, receiving codeword. “M2_24X72_MS” is the process buffer, a dual-port memory with 24 entries, 72 bits per entry. “M1_12X120_beta” pr a single port memory storing Beta1, Beta2, index and it has 12 entries, 120 bits per entry. “M2_88X48_sign” denotes a dual-port memory with 88 entries, 48 bits per entry storing sign values. We store differences of successive adjacent shift amounts defined in parity check matrices in “ROM_1152X84_shf”. “ROM_256X60_addr” stores column addresses of non-zero block matrices of each row in parity check matrix. We have to schedule addresses in advance to avoid memory confliction to overlap CHK/VAR operations. “ROM_256X84_final” stores the inverse shift amounts for output. All the sizes of memory are estimated in advance by analyzing parity check matrices defined in 802.11n and 16e.

Table 4-3：Summary of RAMs and ROMs

Memory block	Type	Gate Counts
M1_24X72_inbuff	single port	27200
M1_12X120_beta	single port	48000
M2_24X72_MS	dual port	36800
M2_88X48_sign	dual port	17200
ROM_1152X84_shf	ROM	24000
ROM_256X60_addr	ROM	12000
ROM_256X84_final	ROM	16000

The total area of ROMs and RAMs has 33% of total design area at 2.5ns in TSMC 0.13 µm 1P8M COMS technology. The related decoding performances and comparisons will be discussed in next section.

4.2 Implementation

Figure 4-23 illustrates the general ASIC design flow excised by the proposed LDPC decoder. The design flow can be classified into three categories: "**Algorithm Design**", "**Architecture Design**" and "**Gate-level Design**". The whole design phases are presented in sequence as follows.

The proposed LDPC decoder is implemented in TSMC 0.13 µm 1P8M CMOS technology. The chip operates at 333 MHz and, with 10 iterations for different code rates and code lengths. It has the decoding throughput of 213~590 Mb/s with power dissipation of 451 mW for 802.16e and throughput of 240~506 Mb/s with average power consumption 436 mW for 802.11n. In low power mode, we slow down the operation frequency to 66 MHZ (one-fifth of 333 MHz) to meet the required minimum throughput, 30 Mb/s in 802.16e. The detail results will be discussed and showed as follows.

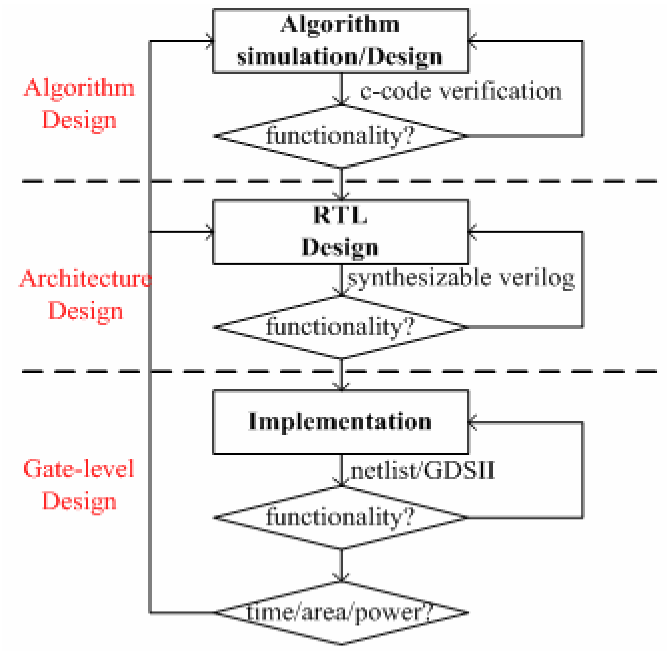


Figure 4-23：Design flow of proposed LDPC decoder

● Algorithm Design

We first adopt LDPC decoding algorithm, optimize decoding flow and run the simulation by C-language. According to simulation results, we decide related parameters, normalization factor and maximum number of iteration, and estimate roughly decoding performance. If decoding performances don't meet specification, we must modify algorithms or think of other solutions. We start architecture design until decoding algorithm verification meets the requirement.

● Architecture Design

A synthesizable RTL is conducted right by systematic architecture design. We have to design the architecture with appropriate pipelines to meet timing requirement. ROMs and RAMs have a trade off between operating frequency and available size. The fully synthesizable RTL codes are verified by C-code model with HDL simulator ***Verilog-XL***. We synthesize RTL codes and estimate timing and area without wire loading model. If timing is not meet, we have to modify RTL codes by pipelining or other coding types until meet the requirements.

- **Gate-level Design**

A synthesizable RTL codes are first transformed to the gate-level netlist by ***Synopsys Design Compiler***, and then the static timing analysis, logic equivalence checking are carried out to ensure timing closure and correct functionality. We implement gate-level design by encounter. The ***Synopsys physical compiler*** is furthermore applied after the trivial physical design which utilizes the ***SoC Encounter***. This is because as the technology advances rapidly, the placement has large impact on the circuit performance. Again, the gate-level simulation and verification are used to exercise the synthesized netlist through physical compiler. Finally, the physical design, i.e. floorplan, place & route etc. is carried out by ***SoC Encounter***. Finally, the ***PrimePower*** is used to estimate the power consumption. The functionality of netlist of post-layout is verified by C-language with ***Verilog-XL***.

4.2.1 Implementation Results

The proposed LDPC decoder is implemented with TSMC 0.13 μm 1P8M CMOS technology. Synthesis results are shows in Table 4-4 and it can be taken references to compared with total area of memory. Summary of memory is listed as Table 4-5. The area of memory has 33% of total design area.

Table 4-4：Synthesis results

Summary	Gate Counts
Total Synthesis Area	643469
Combination logic	188702
Noncombination logic	454937

Table 4-5：Summary of memory

Memory block	Type	Gate Counts
M1_24X72_inbuff	single port	27200
M1_12X120_beta	single port	48000
M2_24X72_MS	dual port	36800
M2_88X48_sign	Dual port	17200
ROM_1152X84_shf	ROM	24000
ROM_256X60_addr	ROM	12000
ROM_256X84_final	ROM	16000

We analyze the overhead of a dual-decoder from a single application, 802.11n to merge 802.16e. Except for modifying some parameters for ROM tables, number of processing units should increase from 81 (for 802.11n) to 96 (for 802.16e) because of Z_f. Synthesis area for a processing unit (CHK/VAR) is almost 3042 gate counts. Moreover, ROM tables have 24076 gate counts for 802.16e and the area of a signal decoder for 80211n is almost 573793 gate counts. Consequently, the design area is almost 15% overhead.

The specification of proposed LDPC decoder is summarized in Table 4-6. Figure 4-24 shows the die photo of proposed LDPC decoder. The core size is $2.14 \times 2.14 mm^2$ and die size is $2.69 \times 2.69 mm^2$. The decoder operates at 333 MHz with 10 iterations for different code rates and code lengths. It has a peak throughput of 590 Mb/s and power dissipation of 451 mW for code rate 5/6, code length 2304 bits in 802.16e. In 802.11n, its peak throughput is 506 Mb/s with power dissipation of 436-mW for code rate 5/6, code length 1944 bits. The decoding throughput is estimated by the equation (4.9) without input/output latency.

Table 4-6：Summary specification of LDPC decoder

	Proposed
Technology	TSMC 0.13 µm 1P8M CMOS
Supply voltage	1.2 V
Max. Clock freq.	333 MHz
Die size	2.69×2.69 mm^2
Core size	2.14×2.14 mm^2
Max. Throughput	11n, 10-iter., 506 Mb/s
	16e, 10-iter., 590 Mb/s
Power dissipation	11n, 10-iter., 436 mW
	16e, 10-iter., 451 mW

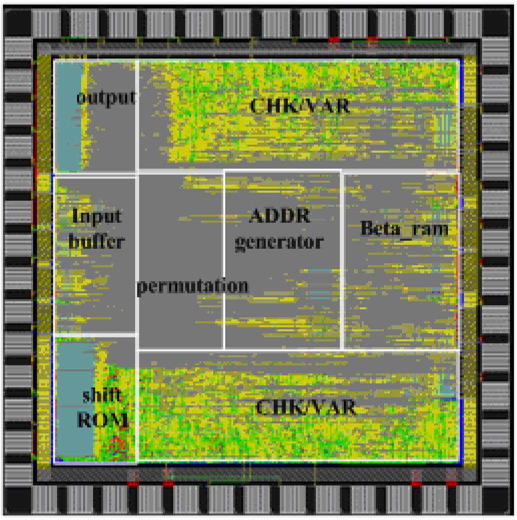


Figure 4-24：Photo of LDPC decoder

$$\text{Throughput}=\frac{\text{Frequency}\times\text{codeword length}}{\text{decoding cycle counts}} \tag{4.9}$$

Because the required minimum throughput of 802.16e is only 30 Mb/s, we can lower performance to support the low power mode. Some techniques are often applied in low power design, like slowing down VDD voltage or operating frequency. For ease to design, we divide operating frequency by 5 to 66 MHz. It has throughput 42.6~118 Mb/s for different code rates and codeword lengths in 802.16e at 66 MHz. The power dissipation as shown in Table 4-7 is lower to 86~101mW for different code rates, codeword length 2304 bits. The required minimal throughput of 802.11n is 300

Mb/s. The minimum throughput of our proposed design for 802.11n is 240 Mb/s for code rate 1/2, codeword length 648 bits. It almost meets the requirement.

Table 4-7：Average power consumption in low power mode at 66 MHz, *1

Code Rate	Power Dissipation
Rate 1/2	86 mW
Rate 2/3A	99 mW
Rate 2/3B	100 mW
Rate 3/4A	101 mW
Rate 5/6	91 mW

***1: different code rates with maximum codeword length 2304 bits**

● Comparison

In synthesis status, timing constraint is loose because it doesn't consider of physical problems. The proposed can meet an operating frequency 400 MHz in synthesis status and has a maximum throughput of 709 Mb/s for 802.16e and 607 Mb/s for 802.11n. When considering of physical design, the operating downs to 333 MHz in post-layout simulation. The comparison of our proposed LDPC code decoder with status-of-arts is presented in Table 4-8.

Table 4-8：Comparison of LDPC code decoder

	[21]	[22]	[23]	**Proposed**
Application	11n	16e	16e	**11n and 16e**
Technology	0.13 μm	0.13 μm	0.13 μm	**0.13 μm**
status	synthesis	synthesis	CHIP	**post-layout**
clock freq.	412 MHz	333 MHz	83.3MHz	**333 MHz**
Iteration	15	10 or 15	2~8	**2~10**
Termination	Yes	No	Yes	**Yes**
Throughput	736 Mb/s	610Mb/s	111Mb/s	**506Mb/s(11n), 590Mb/s(16e)**
Power	N/A	N/A	N/A	**436 mW(11n) 451 mW(16e)**
Low Power Mode	N/A	N/A	52 mW^{*1}	**91 mW^{*2}**
Core size	N/A	N/A	2.11×2.11mm^2	**2.14×2.14mm^2**

***1: 30Mb/s**

***2: 42Mb/s, for 802.16e, rate 5/6, codeword length 2304 bits, 10-iter.**

CHAPTER 5 SUMMARY

In this thesis, we proposed a configurable LDPC decoder for *IEEE* 802.11n and 802.16e. First, we analyze LDPC decoding algorithms for 802.11n and 802.16e and improvement spaces for row-update message passing, Belief Propagation, and Min-Sum algorithm, etc. According to simulation results by C-language, we decide normalization factor, number of iteration, bit width and other parameters for hardware implementation. A trade-off between hardware complexity and decoding performance is analyzed to decide the parameters. A configurable, partially parallel architecture with Z_f parallelization is proposed to apply row-update message passing algorithm with min-sum. Some design considerations are discussed and solved by the proposed methods.

For adaptive code rates and code lengths, a point based permutation is designed to merge 22 types of Z_f's for 802.11n and 802.16e. Besides, parallel multi-codeword decoding technique for preserving high hardware utilization and early termination mechanism to save power are considered. Multi-codeword decoding technique not only increases hardware utilization but also throughput when decoded codeword is less than 1152 bits ($Z_f < 48$). Moreover, a flexible control provides users to decide whether turning off early termination or not for adaptive code rates and transmission channel.

The design is implemented in TSMC 0.13 μm 1P8M CMOS technology. The core size is 2.14×2.14 mm^2 and die size is 2.69×2.69 mm^2. The decoder operates at 333 MHz with 10 iterations for different code rates and code lengths. It has a peak throughput of 590 Mb/s and power dissipation of 451 mW for code rate 5/6, code length 2304 bits in 802.16e. In 802.11n, its peak throughput is 506 Mb/s with power dissipation of 436-mW for code rate 5/6, code length 1944 bits.

In low power mode, we divide operating frequency by 5 to 66 MHz to meet the required minimum throughput, 30 Mb/s for 802.16e. It has throughput 42.6~118 Mb/s for different code rates and code lengths. Power dissipation is lower to 86~101 mW for 802.16e in low power mode.

Reference

[1] *IEEE Std* 802.16e-2005, Air Intreface for Fixed and Mobile Broadband Wireless Access Systems

[2] *IEEE* 802.11n Wireless LANsWWiSE Proposal: High Throughput extension to the 802.11 Standard. *IEEE* 11-04-0886-00-000n.

[3] P. Elias, "Coding for noisy channels," *IRE. Conv. Rec.*, pt.4, pp.37-47,1955.

[4] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Fields," J. Soc. Ind. Appl. Math., 8: 300-304, June 1960.

[5] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol 44, no. 10, pp. 1261-1271, Oct. 1996.

[6] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory,* vol. IT-8, pp.21-28, Jan. 1962.

[7] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory,* vol. IT-27, no. 5, pp. 533-547, Sept. 1981.

[8] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.,* vol. 33, no. 6, pp. 457-458, Mar. 1997.

[9] D. J. C. MacKay, "Good error-correction codes based on very sparse matrices," *IEEE Trans. Inform. Theory,* vol. 45, no. 2, pp. 399-431, Mar. 1999.

[10] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Commun. Lett.,* vol. 5, no. 2, pp. 58-60, Feb. 2001.

[11] J. Perl, *Probabilistic Reasoning in in intelligent systems: networks of plausible inference.* San Mateo: Morgan Kaufmann, 1988.

[12] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo decoding as a instance of Pearl's belief propagation algorithm," *IEEE J. Select. Areas Commun.,* vol. 16, no. 2, pp. 140-152, Feb. 1998.

[13] J. K. Fan, *Constrained coding and soft iterative decoding.* Netherlands: Kluwer Academic, 2001.

[14] G. D. Forney, Jr., "Codes on graphs: Normal realizations," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 520-548, Feb. 2001.

[15] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp.

498-519, Feb. 2001

[16] N.Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation, Univ. Linkoping, Sweden, 1996.

[17] P. Radosavljevic, A. de Baynast, and J. R. Cavallaro, “Optimized Message Passing Schedules for LDPC Decoding,” *In IEEE 39th Asilomar Conference on Signals, Systems and Computers,* pages 591-595, Nov. 2005

[18] D. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *Signal Processing Systems SIPS 2004, IEEE Workshop* on, pp. 107-112, Oct. 2004.

[19] Y. C. Liao, C. C. Lin, C. W. Liu, and H. C. Chang, “A Dynamic Normalization Technique for Decoding LDPC Codes,” IEEE Workshop on Signal Processing Systems (SIPS), Athens, Greece, pp.768~772, Nov. 2005

[20] C. H. Liu , C. C. Lin , H. C. Chang , C. Y. Lee and Y. S. Hsu, “Method and apparatus for switching data in communication systems,” Taiwan and US patent pending.

[21] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, “Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Implementation,” In *Proc. of 17th International Conference on Application -specific Systems and Processors*, pp. 360~367, 2006.

[22] T. Brack, M. Alles, F. Kienle, and N. When, “A Synthesizable IP Core for WiMAX 802.16e LDPC code Decoding,” *IEEE International Symposium on Personal, Indoor and Mobilie Radio Communication,* pp. 1~5, Sep. 2006

[23] X. Shih, C. Zhan, C. Lin, and A. Wu, “A 19-mode 8.29mm^2 52mW LDPC Decoder Chip for IEEE 802.16e System,” *IEEE Symp. VLSI Circuits and VLSI Technology* (SOVC-2007), Kyoto, JAPAN, pp 16-17, June 2007.

[24] A. Anastasopoulos, “A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution,” in *IEEE GlOBECOM’01*, vol. 2, pp. 1021-1025, Nov. 2001.

[25] X. Y. Hu, Eleftheriou, D. M. Arnold, and A. Dholakia, “Efficient implementation of sum-product algorithm for decoding ldpc codes,” in *IEEE GLOBECOM’01*, vol. 2, pp. 1036-1036E, Nov. 2001.

[26] H. S. Song and P. Zhang, “Very-low-complexity decoding algorithm for low-density parity-check codes,” in *IEEE PRIMRC’03,* vol. 1, pp. 161-165, Sep. 2003.

[27] J. Chen and M. P. C. Fossorier, “Near optimum universal belief propagation based decoding of low-density parity check codes,” *IEEE Trans. Commun.*, vol. 50, pp. 406-414, Mar. 2002.

[28] H. Jun and K. M. Chugg, “Optimization of scaling soft information in iterative

decoding via density evolution methods," in IEEE Trans. Commun., vol. 6, pp. 957-961, Jun. 2005.

[29] J. Chen and M. P. C. Fossorier, "Density evolution for two improved bp-based decoding algorithms of ldpc codes," *IEEE Communications Letters,* vol. 6, pp. 208-210, May 2002.

[30] D. B. West, introduction to graph theory, 2nd ed. NJ: Prentice-Hall, 2001.

[31] J. Xu, L. Chen, L. Q. Zeng, L. Lan, and S. Lin, "Construction of low-density parity-check codes by superposition," *IEEE Trans. Commun.,* vol. 54, no. 1, pp. 71-81, Jan. 2006.

[32] H. Tang, J. Xu, S. Lin, and K. Abdel-Ghaffar, "Codes on finite geometries," *IEEE Trans. Inf. Theory,* vol. 51, no. 2, pp. 572-596, Feb. 2005.

[33] Z. W. Li, L. Chen, L. Q. Zeng, S. Lin, and W. H. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," IEEE Trans. Commun., vol. 54. no. 1, pp. 71-81, Jan. 2006.

作者簡歷

姓名：劉士賢

出生地：台南縣

出生日期：1983 年 3 月 12 日

學歷：1989.9～1995.6 台南縣立龍潭國民小學

1995.9～1998.6 台南市私立長榮高級中學 國中部

1998.9～2001.6 台南市私立長榮高級中學 高中部

2001.9～2005.6 國立交通大學 電子工程學系 學士

2005.9～2007.8 國立交通大學 電子研究所 系統組 碩士