國立交通大學

電子工程學系 電子研究所

碩士論文

高速低密度同位元檢查碼解碼器於 IEEE 802.3an 之設計與實作

Design and Implementation of High-Throughput LDPC Decoder for IEEE 802.3an Applications

研究生：鄭佳瑋
指導教授：張錫嘉 博士

中華民國九十六年十一月

高速低密度同位元檢查碼解碼器於 IEEE 802.3an 之設計與實作

Design and Implementation of High-Throughput LDPC Decoder for IEEE 802.3an Applications

研究生：鄭佳瑋　　Student：Chia-Wei Cheng

指導教授：張錫嘉 博士　　Advisor：Dr. Hsie-Chia Chang

國立交通大學

電子工程學系 電子研究所

碩士論文

A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering
November 2007
Hsinchu, Taiwan, Republic of China

中華民國九十六年十一月

高速低密度同位元檢查碼解碼器於 IEEE 802.3an 之設計與實作

研究生：鄭佳瑋　　　　　　　　　　指導教授：張錫嘉 博士

國立交通大學

電子研究所

摘要

這篇論文實作一個用於 IEEE 802.3an 標準的 (2048, 1723) 低密度同位元檢查碼解碼器。根據 IEEE 802.3an 標準中採用的低密度同位元檢查碼的結構，在選擇訊息儲存位址時，我們提出使用移位暫存器代替多功器的架構，可減少硬體的使用量與繞線的複雜度。同時採用部份平行化的架構不僅讓面積縮小，也減少繞線困難性。在表現出相同的位元錯誤率的條件下，使用階層解碼演算法可以有效降低解碼迴圈數。使用 UMC 90nm 製程實作後，在操作頻率 100MHz 時所提出的架構可以達到最高傳輸速率每秒 5.6G bits。實作後完整的低密度同位元檢查碼解碼器核心面積大小是 4.0 x 4.0 mm^2，在電壓供應 0.9 伏特的平均功率消耗為 993mW。

Design and Implementation of High-Throughput LDPC Decoder for IEEE 802.3an Applications

Student：Chia-Wei Cheng　　　Advisor：Dr. Hsie-Chia Chang

Department of Electronics Engineering & Institute of Electronics

National Chiao Tung University

ABSTRACT

In this thesis, a (2048, 1723) LDPC decoder applied to IEEE 802.3an standard is implemented. According to the structure of the LDPC code adopted in IEEE 802.3an, the architecture using shift registers instead of multiplexers for message storage is proposed to reduce the hardware cost and routing congestion. The partial parallel scheme is used for decreasing area as well as routing resource. The layered decoding algorithm is also applied for decreasing decoding iterations with similar BER performance. After implemented with UMC CMOS 90nm process, the proposed decoder can achieve 5.6 Gbps decoding throughput under clock frequency of 100MHz. The core size is 4.0 x 4.0 mm^2 and the average power consumption with a 0.9V supply is 993mW.

誌謝

在這段碩士的修業過程中，我擁有很好的學習機會與環境，也學到了很多做研究的態度與方式。首先，要感謝的是我的指導教授張錫嘉老師，在老師的全力支援下，不管是實驗設備、資訊交流、知識傳遞或是生活大小事，都能安心而無後顧之憂的往目標邁進。由於老師的親切與體貼，讓實驗室的氣氛一直維持著歡樂和諧的狀態，也讓我們的學習充滿樂趣。

接著感謝帶領我進入通道編碼大門的建青學長，在學長的努力下，Ocean 這個研究團隊平穩而順利的成長，提供了我們這些研究通道編碼的學生一個豐富而完整的交流平台。在這個團隊中，彥欽學姊和阿龍學長更是時常給予我許多研究上的建議以及指導，讓我能順利解決在研究上的瓶頸。還有，紹維學長、陸志豪學長和義凱在 LDPC 上的專精也給了我不少的啟發，高守學弟也提供了很大的幫助。整個 Ocean 團隊的成員互動帶給我的絕對不僅止於一本論文的份量而已，這個團隊的每個份子對我而言都是良好的學習對象與伙伴。

1896

雖然 Oasis 實驗室內的成員研究方向不盡相同，但彼此的相互激勵與關心仍然是不可或缺的動力。所以在此也要感謝大頭、昭逸、胖威、祐徵、耀仁學長們的照顧，以及修齊、國光、博元、俊閔的相互陪伴，還有可愛的學弟妹們帶給我的成長與歡笑。

最後，感謝我的父母與姊姊，你們的付出與關愛對我而言，絕對是最強大也最完美的支持。

Contents

1 Introduction 1

1.1 Motivation . . . 1

1.2 Thesis Organization . . . 1

2 Low-Density Parity-Check Codes 3

2.1 Concept of LDPC Codes . . . 3

2.1.1 Belief Propagation Algorithm . . . 5

2.1.2 Decoding Procedure . . . 9

2.2 LDPC Codes Based on Shortened RS Code . . . 14

2.3 LDPC Code for IEEE 802.3an . . . 18

3 Implementations of BP Algorithm 20

3.1 Min-Sum Approximation . . . 22

3.2 Look-up Table Approximation . . . 25

3.3 Layered Decoding Algorithm . . . 31

4 Proposed Partial Parallel Architecture 35

4.1 Overview . . . 35

4.2 Check Node Unit . . . 38

4.3 Variable Node Unit . . . 39

4.4 Comparison and Implementation Results . . . 40

5 Conclusion and Future Work 43

5.1 Conclusion . . . 43

5.2 Future Work . . . 44

List of Figures

2.1 The bipartite graph for the (6, 3) code 4
2.2 An example of normal graph 6
2.3 Graph representation of the extrinsic and the intrinsic probabilities 6
2.4 Graph representation of the message passing between two nodes 7
2.5 The check node with degree d 10
2.6 The variable node with degree d 11
2.7 The procedure of iterative decoding for LDPC codes 12
2.8 Parity-check matrix of RS-based (3, 3)-regular LDPC code 17
2.9 The roughly view of H of (6, 32)-regular LDPC code 18
2.10 Parity check matrix of (2048, 1723) RS-based LDPC code 18

3.1 Simulation result (1): comparison of different iterations 21
3.2 Simulation result (2): fixed-point simulation 22
3.3 The inputs and the outputs of a check node 22
3.4 The function $\Psi(x) = -\ln(\tanh(\frac{x}{2}))$ 24
3.5 Simulation result (3) : comparision of BP algorithm and min-sum algorithm 25
3.6 The function $g(x) = \ln(1 + e^{-|x|})$ 26
3.7 Parallel configuration for computing check-node updates 27
3.8 The function $h(x) = \ln|e^x - 1|$ 28
3.9 Simulation result (4) : the result of applying look-up table approximation . 29
3.10 The simlified structure of LUT 29
3.11 Simulation result (5) : the result of simplified structure of LUT 30
3.12 Illustration of the process of conventional BP algorithm and layered decoding algorithm 33

3.13 Simulation result (6) : layered decoding algorithm applied with min-sum approximation 34
3.14 Simulation result (7) : layered decoding algorithm applied with look-up table approximation 34

4.1 The brief overview of the proposed LDPC decoder architecture 36
4.2 The schedule of the decoder while running two codewords simulaneously . 36
4.3 The overview of the proposed LDPC decoder architecture 37
4.4 The architecture of check node unit. 38
4.5 The architecture of variable node unit at cycle=12k+1. 40
4.6 The layout of the (2048, 1723) LDPC decoder. 41

List of Tables

3.1 Quantization table for $g(x) = \ln(1 + e^{-|x|})$ 26

3.2 Quantization table for $h(x) = \ln|e^{x} - 1|$ 28

3.3 Comparison of different check node structure 31

4.1 The synthesis result of the proposed architecture 41

4.2 The comparison of different architecture 42

Chapter 1

Introduction

1.1 Motivation

Low-density parity-check (LDPC) codes was first invented by Gallager in 1962 [1] [2]. Due to the difficulty of circuit implementation, LDPC codes have been out of consideration for nearly forty years before being rediscovered by Mackay [3] [4]. The highly parallelizable algorithm of these codes with simple arithmetic computations facilitates high-data-rate applications. Recently, LDPC codes have been adopted by several communication standards, such as WiMAN (IEEE 802.16e standard) [5] and 10 Gigabit Ethernet (IEEE 802.3an standard) [6].

IEEE 802.3an is a new standard of high-thoughput Ethernet. After analyzing the parity check matrix of LDPC code for the standard, we divide the check nodes in to several groups as a partial parallel scheme, and find out that when the number of groups is equal to the degree of the variable node, each column of each group will have exactly one entry. So that layered decoding algorithm is suitable for applying on this parity check matrix. According to the property of this LDPC code, we propose an architecture of the decoder, and the detail discussion will be in the following chapters.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes the concept and decoding algorithm of LDPC codes, as well as the structure of LDPC codes based on shortened

RS codes, which construct the LDPC code in IEEE 802.3an. Chapter 3 introduces some methods to approximate the belief propagation (BP) algorithm when implementing the decoder and the corresponding simulation results. The proposed LDPC decoder architecture, including the details of the functional units and the implementation result, are presented in Chapter 4. Finally, conclusion and future work are made in Chapter 5.

Chapter 2

Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes was first invented by Gallager in 1962 and rediscovered by Mackay in the middle 1990's. It is a linear block code defined by a very sparse parity check matrix, and moreover, LDPC codes are proved to achieve the capacity when the block length is large [3]. The decoding algorithm is simple in arithmetic computations due to the inherent parallelism, leading to easier realization of high-throughput decoders.

2.1 Concept of LDPC Codes

A binary LDPC code is constructed by a sparse parity check matrix **H**, which contains mostly 0's and only a small fraction of 1's. The parity check matrix **H** of an (N, K) LDPC code has N columns and M rows ($M \leq N - K$), and defined as an LDPC code with the block length of N bits and M parity checks. The code rate is $R = K/N$. An LDPC code can be called a regular LDPC code if each row of H has the same number ($= k$) of 1's, and so does each column (j 1's). It can be named as a (j, k) regular LDPC code; otherwise, it is an irregular LDPC code.

The block length of an LDPC code is often designed as large as possible, because the minimum distance of a code grows with block length. The minimum distance is an important factor that influences the bit-error-rate (BER) performance.

An LDPC code is often represented by the bipartite graph, or Tanner graph [7]. As an example, if we have a parity check matrix

$$\mathbf{H} = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{cccccc|c} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & \\ 1 & 1 & 0 & 1 & 0 & 0 & \mathrm{C}_0 \\ 0 & 1 & 1 & 0 & 1 & 0 & \mathrm{C}_1 \\ 1 & 0 & 1 & 0 & 0 & 1 & \mathrm{C}_2 \end{array},$$

than

$$[x_0\ x_1\ x_2\ x_3\ x_4\ x_5] \times \mathbf{H}^T = [C_0\ C_1\ C_2]\,,$$

where $\mathbf{X} = (x_0, x_1, x_2, x_3, x_4, x_5)$ is a codeword. According to the property of a linear block code, $\mathbf{XH}^T$ should be zero, or $\mathbf{C} = (C_0, C_1, C_2) = \mathbf{0}$. We can represent it as three equations:

$$\begin{aligned} C_0 &: \ x_0 \oplus x_1 \oplus x_3 = 0 \\ C_1 &: \ x_1 \oplus x_2 \oplus x_4 = 0 \\ C_2 &: \ x_0 \oplus x_2 \oplus x_5 = 0 \end{aligned}$$

The bipartite graph of this parity check matrix has 6 variable nodes and 3 check nodes. The values in the parity check matrix show the connections between variable nodes and check nodes. In other words, if a 1 appears in the entry (i, j) of $\mathbf{H}$, an edge connects between ith check node and jth variable node. Fig. 2.1 represents the bipartite graph of this $\mathbf{H}$.

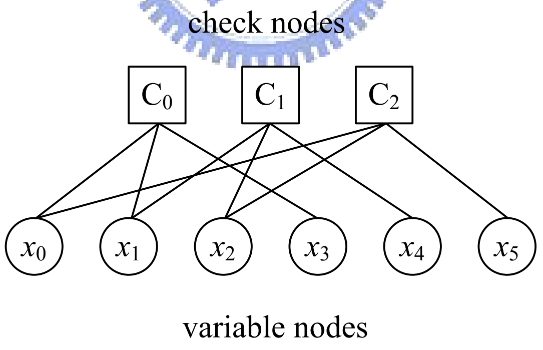


Figure 2.1: The bipartite graph for the (6, 3) code

The column weight of $\mathbf{H}$ determines the number of edges connected to each variable node, also called the degree of each variable node. And it is the same with the row weight of $\mathbf{H}$, which is called the degree of each check node.

2.1.1 Belief Propagation Algorithm

The decoding algorithm of LDPC codes is based on soft iterative decoding, which relies on the belief propagation (BP) algorithm, or message passing (MP) algorithm [8]. The basic principle is to break up a computationally difficult problem into a multitude of easier subproblems, and the key is to pass and exchange probabilistic messages in a graph iteratively.

First, consider the a posteriori probability. For a variable x, if the probability of event $x = a$ (expressed as $P(x = a)$) is dependent of the occurrence of event E, we denote the intrinsic probability for x with respect to E by

$$P_E^{int}(x = a) = P(x = a), \tag{2.1}$$

and the a posteriori probability is the conditional probability when E is true, it is denoted by

$$P_E^{post}(x = a) = P(x = a|E). \tag{2.2}$$

According to the Bayes' theorem, a posteriori probability can be expressed as

$$\overbrace{P(x = a|E)}^{a\ posteriori} = \frac{1}{P(E)} \overbrace{P(E|x = a)}^{prop.\ to\ extrinsic} \overbrace{P(x = a)}^{intrinsic}. \tag{2.3}$$

The term $P(x = a)$ is the a priori probability and is also referred to the intrinsic probability for x. The other term $P(E|x = a)$ is proportional to the extrinsic probability, which is a probability that describes the new information for x obtained from the event E. Define the extrinsic probability for x with respect to E by

$$P_E^{ext}(x = a) = (\sum_{a' \in A} P(E|x = a'))^{-1} P(E|x = a) = \rho_e P(E|x = a), \tag{2.4}$$

assuming a' takes values from an alphabet set $\mathbf{A}$. ρ_e is a normalization constant that makes the summation of $P_E^{ext}(x = a')$ equals to 1. Rewrite the equation as

$$P_E^{post}(x = a) = P(x = a|E) = \rho_c P_E^{int}(x = a) P_E^{ext}(x = a), \tag{2.5}$$

where $\rho_c = (\rho_e P(E))^{-1}$.

For binary variables, $\mathbf{A} = \{0, 1\}$. We define $L(x) = \ln \frac{P(x=0)}{P(x=1)}$ as log-likelihood ratio (LLR), than rewrite equation (2.5) as

$$L^{post}(x) = \ln \frac{P^{post}(x = 0)}{P^{post}(x = 1)} = \ln \frac{\rho_c P^{int}(x = 0) P^{ext}(x = 0)}{\rho_c P^{int}(x = 1) P^{ext}(x = 1)} = L_{int}(x) + L_{ext}(x). \tag{2.6}$$

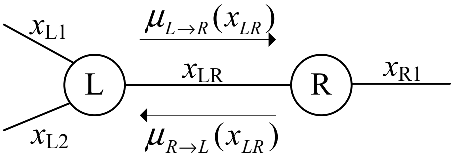


Figure 2.2: An example of normal graph

Note that when $P(x=0) > P(x=1)$, $L(x) > 0$, and vice versa.

Next, we use undirected graphs, referred to the normal graph, in the graph representation. A normal graph consists of nodes and edges. An edge connected to two nodes is an internal edge, which denotes the state variable, while an external edge is connected to only one node and denotes the symbol variable. A node denotes a constraint. Fig. 2.2 shows an example of a normal graph with two nodes. In this example, x_{L1}, x_{L2} and x_{R1} are external edges, and x_{LR} is an internal edge. $\mu_{L\to R}(x_{LR})$ represents the message passing from node L to node R, and $\mu_{R\to L}(x_{LR})$ is passed from R to L. $\mu_{L\to R}(x_{LR})$ is defined as the extrinsic probability from node L, and to be used as the intrinsic probability in node R, so that

$$\begin{aligned}\vec{\mu}_{L\to R}(x_{LR}) &= \vec{P}_L^{ext}(x_{LR}) = \vec{P}_R^{int}(x_{LR})\\ \vec{\mu}_{R\to L}(x_{LR}) &= \vec{P}_R^{ext}(x_{LR}) = \vec{P}_L^{int}(x_{LR})\end{aligned} \tag{2.7}$$

show the message-passing flow between these two nodes.

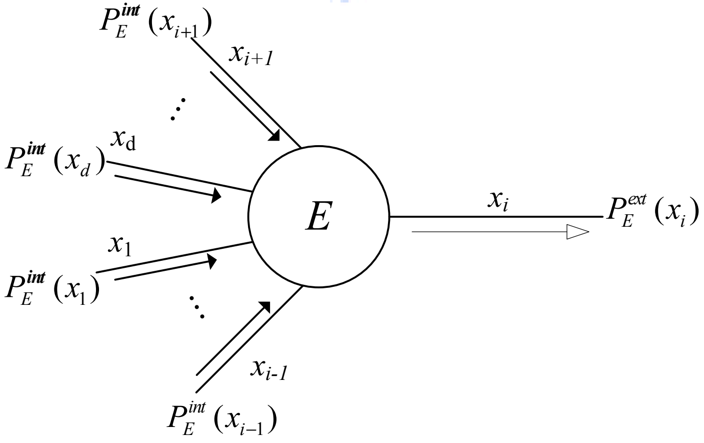


Figure 2.3: Graph representation of the extrinsic and the intrinsic probabilities

Consider another normal graph with a single node E and d external edges, as shown in Fig. 2.3. We define a set $\mathbf{S}_E$ which is a subspace of the d-dimensional vector space $\mathbf{A}^d$

($\mathbf{S}_E \subset \mathbf{A}^d$), and any d-tuple $\mathbf{x} = (x_1, x_2, ..., x_d) \in \mathbf{S}_E$ will satisfy the constraint E. For $j = 1 \sim d$, assume that each edge has the probability $P_E^{int}(x_j)$ associated with the symbol x_j, then the a posteriori probability of a symbol x_i with respect to E will be obtained from the combination of the intrinsic probabilities of x_j and the extrinsic probability of x_i. So we have to evaluate $P_E^{ext}(x_i)$ based on the constraint E and other intrinsic probabilities. As the definition of (2.4), $P_E^{ext}(x_i) = \rho_e P(E|x_i)$. Assume the symbol variables $x_1, x_2, ..., x_d$ are independent of each other, therefore, $P(E|x_i)$ is the conditional probability which can be presented as

$$\begin{aligned} P(E|x_i) &= \sum_{x_j, \forall j \neq i, \mathbf{x} \in \mathbf{S}_E} P(E, x_1, ..., x_{i-1}, x_{i+1}, ..., x_d | x_i) \\ &= \sum_{x_j, \forall j \neq i, \mathbf{x} \in \mathbf{S}_E} P(E|x_1, ..., x_{i-1}, x_{i+1}, ..., x_d) P(x_1, ..., x_{i-1}, x_{i+1}, ..., x_d | x_i) \\ &= \sum_{x_j, \forall j \neq i, \mathbf{x} \in \mathbf{S}_E} P(x_1, ..., x_{i-1}, x_{i+1}, ..., x_d | x_i) \\ &= \sum_{x_j, \forall j \neq i, \mathbf{x} \in \mathbf{S}_E} \prod_{j=1, j \neq i}^{d} P_E^{int}(x_j). \end{aligned} \tag{2.8}$$

So that $P_E^{ext}(x_i)$ can be evaluated by multiplying normalization constant ρ_e by the result of the conditional probability $P(E|x_i)$.

1896

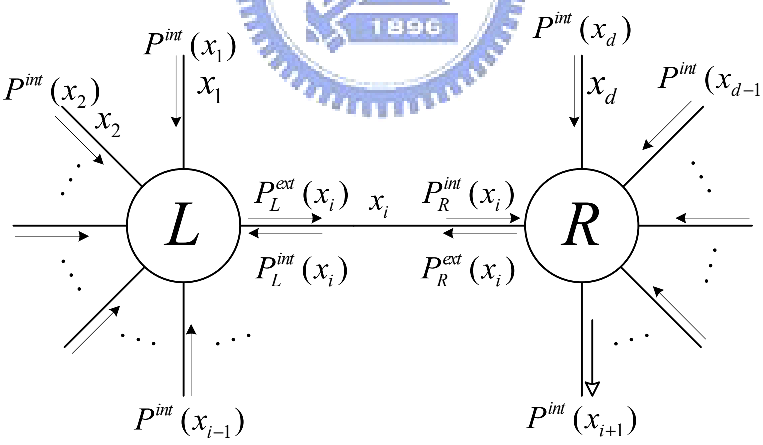


Figure 2.4: Graph representation of the message passing between two nodes

Additionally, we consider the graph with two nodes (constraints), L and R, as Fig. 2.4 shows. The constraint L has $i-1$ external edges and one internal edge, labeled by the symbol $x_1 \sim x_{i-1}$ and x_i. The constraint R has $d-i$ external edges corresponding to the symbol $x_{i+1} \sim x_d$ and one internal edge x_i. Define two constraint sets $\mathbf{S}_L$ and $\mathbf{S}_R$, such that $\mathbf{x}_L = (x_1, x_2, ..., x_i) \in \mathbf{S}_L$, $\mathbf{x}_R = (x_i, x_{i+1}, ..., x_d) \in \mathbf{S}_R$ will satisfy L and R respectively. To evaluate the extrinsic probability for the external edges based on both L and R, we start from x_{i-1} first. If only node R is considered, the extrinsic probability can be written based on (2.8) as

$$P^{ext}(x_{i+1}) = \rho_R P(R|x_{i+1}) = \rho_R \sum_{\mathbf{x}_R \backslash x_{i+1}, \mathbf{x}_R \in \mathbf{S}_R} P_R^{int}(x_i) \prod_{j=i+2}^{d} P^{int}(x_j). \tag{2.9}$$

But the intrinsic probability $P_R^{int}(x_i)$ of R is an internal edge so that we cannot get it from the inputs. So we also use the constraint L to evaluate

$$P^{ext}(x_{i+1}) = \rho_c P(L, R|x_{i+1}). \tag{2.10}$$

Rewrite the conditional probability as

$$\begin{aligned} P(L, R|x_{i+1}) &= \sum_{\mathbf{x}_R \backslash x_{i+1}, \mathbf{x}_R \in \mathbf{S}_R} P(L, R, x_i, x_{i+2}, ..., x_d|x_{i+1}) \\ &= \sum_{\mathbf{x}_R \backslash x_{i+1}, \mathbf{x}_R \in \mathbf{S}_R} P(R|L, \mathbf{x}_R) P(L, x_i, x_{i+2}, ..., x_d|x_{i+1}) \\ &= \sum_{\mathbf{x}_R \backslash x_{i+1}, \mathbf{x}_R \in \mathbf{S}_R} P(L, x_i, x_{i+2}, ..., x_d|x_{i+1}), \end{aligned} \tag{2.11}$$

where the second equality comes from a Markov chain, and

$$P(L, R|x_i) = P(L|R, x_i) P(R|x_i) = P(L|x_i) P(R|x_i). \tag{2.12}$$

So we can write

$$P(R|L, \mathbf{x}_R) = P(R|\mathbf{x}_R) = 1, \ \ for \ \mathbf{x}_R \in \mathbf{S}_R, \tag{2.13}$$

because $\mathbf{x}_R$ satisfies the constraint R. Factorization the term in (2.11),

$$\begin{aligned} P(L, x_i, x_{i+2}, ..., x_d|x_{i+1}) &= P(L|\mathbf{x}_R) P(x_i, x_{i+2}, ..., x_d|x_{i+1}) \\ &= P(L|x_i) P(x_i) P(x_{i+2}) ... P(x_d) \\ &= (\rho_L)^{-1} P_L^{ext}(x_i) P^{int}(x_i) \prod_{j=i+2}^{d} P^{int}(x_j). \end{aligned} \tag{2.14}$$

$P_L^{ext}(x_i) = \rho_L P(L|x_i)$ is the extrinsic probability of x_i with respect to L, and $P^{int}(x_i)$ represents the intrinsic probability for the internal edge variable x_i. Since the internal edge connect L and R without other external input, the intrinsic probability can be initialized to be a constant, $P^{int}(x_i) = \frac{1}{|A|}\ for\ x_i \in \mathbf{A}$. Then we rewrite extrinsic probability of x_{i+1} as

$$P^{ext}(x_{i+1}) = \rho'_L \sum_{\mathbf{x}_R \backslash x_{i+1}, \mathbf{x}_R \in \mathbf{S}_R} P_L^{ext}(x_i) \prod_{j=i+2}^{d} P^{int}(x_j), \tag{2.15}$$

where $\rho'_L = \rho_C / (\rho_L |\mathbf{A}|)$. From Fig. 2.4, we know that $P_R^{int}(x_i) = P_L^{ext}(x_i)$ if $P_L^{ext}(x_i)$ from L is available. As a result, only constraint R is necessary for evaluating $P^{ext}(x_{i+1})$. The extrinsic probabilities for edge $(i+2) \sim d$ can be estimated with the same method. Correspondingly, extrinsic probabilities for edge $1 \sim (i-1)$ can be calculated when the extrinsic probability $P_R^{ext}(x_i)$ is available, and $P_L^{int}(x_i) = P_R^{ext}(x_i)$. The two equalization relationships between intrinsic and extrinsic probability of L and R, are called message passing. With the message passing algorithm, we break up a two node graph into two single node graph, and the problem becomes much simpler. In conclusion, the message passed on the edge x_i can be represented by

$$\begin{aligned} \mu_{L \to R}(x_i) &= P_L^{ext}(x_i) = \rho_L \sum_{\mathbf{x}_L \backslash x_i, \mathbf{x}_L \in \mathbf{S}_L} \prod_{j=1}^{i-1} P^{int}(x_j) \\ \mu_{R \to L}(x_i) &= P_R^{ext}(x_i) = \rho_R \sum_{\mathbf{x}_R \backslash x_i, \mathbf{x}_R \in \mathbf{S}_R} \prod_{j=i+1}^{d} P^{int}(x_j). \end{aligned} \tag{2.16}$$

The operation in the message passing is the sum of product, thus the message passing algorithm is also called the **sum-product algorithm**.

Finally, we consider a graph that consists of $d+1$ nodes, $N_0, N_1, ..., N_d$, and node N_0 has d internal edges that connect to $N_1 \sim N_d$ respectively with symbol variables $x_1, x_2, ..., x_d$. Assume the messages $\mu_{N_j \to N_0}(x_j)$ with $j = 1 \sim d$ have been obtained from $N_0 \sim N_d$, we can evaluate $\mu_{N_0 \to N_i}$ by

$$\mu_{N_0 \to N_i}(x_i) = \sum_{\mathbf{x} \backslash x_i, \mathbf{x} \in \mathbf{S}_{N_0}} \prod_{j=1, j \neq i}^{d} \mu_{N_j \to N_0}(x_j), \tag{2.17}$$

where $\mathbf{S}_{N_0}$ is the constraint set for N_0. All messages can be found by this equation, and the outputs will become the intrinsic probability inputs for other nodes.

2.1.2 Decoding Procedure

As Fig. 2.1 shows, every parity check matrix of an LDPC code can be represented as a bipartite graph. A bipartite graph consists of two kinds of nodes, check nodes and variable nodes, and each kind of nodes has different constraints. We will introduce the two different operations and the overall decoding flow in this section.

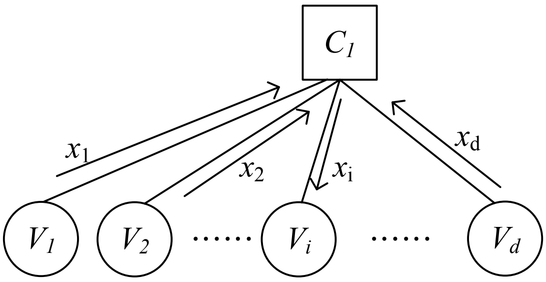


Figure 2.5: The check node with degree d

The basic principle of the decoding algorithm of LDPC codes is to update the a posteriori probabilities of the variable nodes iteratively, using message passing algorithm or belief propagation (BP) algorithm. Considering the message passing for a check node, as shown in Fig. 2.5, the constraint of C_1 is $V_1 \oplus V_2 \oplus ... \oplus V_d = 0$ according to the property of the linear block code. First, starting with the node of only two edges connected to V_1 and V_i, assume the probability of $x_1 = 0$ is q_1, denoted as $P(x_1 = 0) = q_1$, and $P(x_1 = 1) = p_1$. Note that $q_1 + p_1 = 1$ is always true. Since the two messages must satisfy $x_1 \oplus x_i = 0$, we can obtain $P(x_i = 0) = P(x_1 = 0) = q_1$ and $P(x_i = 1) = P(x_1 = 1) = p_1$. Extend the calculation to 3 edges, connected to V_1, V_2 and V_i. If $P(x_2 = 0) = q_2$ and $P(x_2 = 1) = p_2$, for satisfying $x_1 \oplus x_2 \oplus x_i = 0$, the probability of x_i may becomes

$$\begin{aligned} P(x_i = 0) &= P(x_1 \oplus x_2 = 0) = P(x_1 = 0)P(x_2 = 0) + P(x_1 = 1)P(x_2 = 1) \\ P(x_i = 1) &= P(x_1 \oplus x_2 = 1) = P(x_1 = 1)P(x_2 = 0) + P(x_1 = 0)P(x_2 = 1). \end{aligned} \tag{2.18}$$

It can be concluded as $P(x_i = 0) = q_1q_2 + p_1p_2$ and $P(x_i = 1) = p_1q_2 + q_1p_2$. Rewrite the equation, we will get

$$\begin{aligned} 2P(x_i = 0) - 1 &= (q_1 - p_1)(q_2 - p_2) = (1 - 2p_1)(1 - 2p_2) \\ 2P(x_i = 1) - 1 &= -(q_1 - p_1)(q_2 - p_2) = -(1 - 2p_1)(1 - 2p_2). \end{aligned} \tag{2.19}$$

By induction, the results for one check node with d edges can be generalized as

$$\begin{aligned} 2P(x_i = 0) - 1 &= \prod_{j=1, j \neq i}^{d} (1 - 2p_j) \\ 2P(x_i = 1) - 1 &= -\prod_{j=1, j \neq i}^{d} (1 - 2p_j). \end{aligned} \tag{2.20}$$

As a result, the output is

$$\begin{aligned} P(x_i = 0) &= \frac{1}{2}\left[1 + \prod_{j=1, j\neq i}^{d} (1 - 2p_j)\right] \\ P(x_i = 1) &= \frac{1}{2}\left[1 - \prod_{j=1, j\neq i}^{d} (1 - 2p_j)\right]. \end{aligned} \tag{2.21}$$

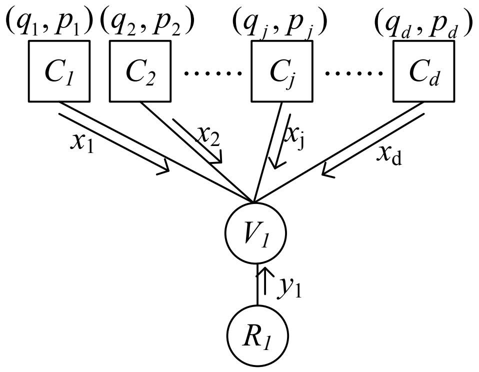


Figure 2.6: The variable node with degree d

Fig. 2.6 shows the variable node with degree d, and the probability of x_i is represented as $(P(x_i = 0), P(x_i = 1)) = (q_i, p_i)$. The constraint of V_1 is $x_1 = ... = x_j = ... = x_d$. For the condition of two edges to variable node, we have (q_1, p_1) and (q_j, p_j), so that the variable node probability can be calculated as

$$\begin{aligned} P(V_1 = 0) &\propto P(C_1 = 0 \ and \ x_1 = 0)P(C_j = 0 \ and \ x_j = 0) = q_1 q_j \\ P(V_1 = 1) &\propto P(C_1 = 0 \ and \ x_1 = 1)P(C_j = 0 \ and \ x_j = 1) = p_1 p_j. \end{aligned} \tag{2.22}$$

To normalize the probability to $P(V_1 = 0) + P(V_1 = 1) = 1$, we obtain that

$$\begin{aligned} P(V_1 = 0) &= \frac{q_1 q_j}{q_1 q_j + p_1 p_j} \\ P(V_1 = 1) &= \frac{p_1 p_j}{q_1 q_j + p_1 p_j}. \end{aligned} \tag{2.23}$$

So if the edges is extended to d, output message V_1 will be

$$\begin{aligned} P(V_1 = 0) &= \rho_b \prod_{k=1}^{d} (x_k = 0) \\ P(V_1 = 1) &= \rho_b \prod_{k=1}^{d} (x_k = 1). \end{aligned} \tag{2.24}$$

and the normalization factor is

$$\rho_b = \frac{1}{\sum_{y_1}\prod_{k=1,k\neq j}^{d}(x_k)}. \tag{2.25}$$

Based on all the information mentioned above, the decoding flow can be shown as below. Fig. 2.7 illustrates the iterative decoding flow of LDPC codes. For a binary codeword, the decoding operations can be performed in terms of LLR, $L(x) = \ln\frac{P(x=0)}{P(x=1)} = \ln\lambda$, in order to decrease the complexity of computations. We will describe each block of Fig. 2.7 in LLR terms as follows.

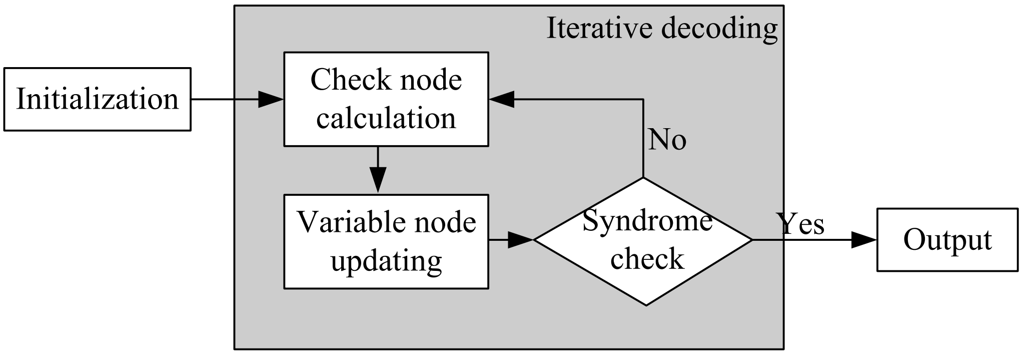


Figure 2.7: The procedure of iterative decoding for LDPC codes

1. Initialization:

   Assume the channel is AWGN channel with distribution $(0,\sigma)$, and BPSK mapping $(0 \to +1,\ 1 \to -1)$ is used. Denote u_i as the inputs of channel, y_i as the outputs of channel. Then

   $$\begin{aligned} P(u_i=+1|y_i) &= \frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(y_i+1)^2}{2\sigma^2}} \\ P(u_i=-1|y_i) &= \frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(y_i-1)^2}{2\sigma^2}}. \end{aligned} \tag{2.26}$$

   Rewrite it in LLR term, the initial message of V_i will be

   $$L_{V_i}(u_i) = \ln\frac{P(u_i=+1|y_i)}{P(u_i=-1|y_i)} = \frac{(y_i+1)^2}{2\sigma^2} - \frac{(y_i-1)^2}{2\sigma^2} = \frac{2}{\sigma^2}y_i, \tag{2.27}$$

   which is called ”channel value” or ”channel information”.

2. Check node updating:

   We labeled the message passing from ith edge of C_j to C_j in LLR term as $q_{j,i}$, that means $q_{j,i} = \frac{P(x_i=0)}{P(x_i=1)} = \ln \lambda_i$. If a C_j has only two edges, $q_{j,1}$ and $q_{j,2}$, the check node operation can be represented as

$$\begin{aligned} CHK(q_{j,1}, q_{j,2}) &= CHK(q_{j,1} \oplus q_{j,2}) = \ln \frac{1+\lambda_1\lambda_2}{\lambda_1+\lambda_2} \\ &= \ln \frac{1+e^{q_{j,1}}e^{q_{j,2}}}{e^{q_{j,1}}+e^{q_{j,2}}} = \ln \frac{e^{-\frac{q_{j,1}+q_{j,2}}{2}}+e^{\frac{q_{j,1}+q_{j,2}}{2}}}{e^{\frac{q_{j,1}-q_{j,2}}{2}}+e^{-\frac{q_{j,1}-q_{j,2}}{2}}} \\ &= 2\tanh^{-1}(\tanh(\frac{q_{j,1}}{2}) \times \tanh(\frac{q_{j,2}}{2})). \end{aligned} \tag{2.28}$$

And the general form of check node operation with d edges is

$$\begin{aligned} CHK(q_{j,1}, q_{j,2}, ..., q_{j,d}) &= CHK(q_{j,1} \oplus q_{j,2} \oplus ... \oplus q_{j,d}) \\ &= CHK(CHK(...CHK(q_{j,1} \oplus q_{j,2})...) \oplus q_{j,d}). \end{aligned} \tag{2.29}$$

Labeling the message passing from C_j to the ith edge of C_j as $r_{j,i}$, than the check node operation notation can be simplify as

$$r_{j,i} = CHK^{d}_{l=1,l\neq i}(\sum \oplus q_{j,l}). \tag{2.30}$$

3. Variable node updating:

   Similarly, we denote the message passing from jth edge of V_j to V_j as $r_{j,i} = \ln \lambda_j$, the variable node operation of two edges $r_{1,i}$ and $r_{2,i}$ is represented as

$$VAR(r_{1,i}, r_{2,i}) = \ln(\lambda_1\lambda_2) = \ln \lambda_1 + \ln \lambda_2 = r_{1,i} + r_{2,i}. \tag{2.31}$$

   In general form with k edges, the message passing from V_j to the jth edge of V_j, notated as $q_{j,i}$, is

$$q_{j,i} = VAR(VAR^{k}_{l=1,l\neq i}(r_{l,i}), L_{V_i}(u_i)) = L_{V_i}(u_i) + \sum_{l=1,l\neq i}^{k} r_{l,i}, \tag{2.32}$$

   where $L_{V_i}(u_i)$ is the channel value which has been calculated at the initialization step.

4. Syndrome check:

   Compute the a posteriori probabilities for each codeword variable by the equation $L^{post}_{V_i}(u_i) = L_{V_i}(u_i) + \sum_{l=1}^{k} r_{l,i}$. De-map the probability by the sign. If the value is negative, the output $\hat{u}_i$ is set to 1, otherwise it is set to 0. Then check if it is a legal codeword by confirming if $\hat{\mathbf{u}}\mathbf{H}^T = 0$.

1896

2.2 LDPC Codes Based on Shortened RS Code

In this section, we will introduce the method of constructing the LDPC codes based on shortened RS codes with two information symbols [9], which has been adopted to be used in the standard IEEE 802.3an (10GBASE-T). For convenience, we use the simplified name, the RS-based LDPC code, in this thesis.

The construction of this code is built by an algebraic method, which is based on shortened Reed-Solomon code with two symbols, being indicated by the name. First, like the RS code, we have a primitive element α of the Galois field $GF(p^s)$, where p is a prime. With a positive integer ρ, where $2 \leq \rho < p^s$, we can construct a generator polynomial of the RS code over $GF(p^s)$,

$$\begin{aligned} \mathbf{g}(X) &= (X-\alpha)(X-\alpha^2)...(X-\alpha^{\rho-2}) \\ &= g_0 + g_1 X + g_2 X^2 + ... + X^{\rho-2}, \end{aligned} \tag{2.33}$$

where $g_i \in p^s$. Note that it is a cyclic RS code of length $p^s - 1$, dimension $p^s - \rho + 1$, and minimum distance $\rho - 1$, represented as $(p^s - 1,\ p^s - \rho + 1,\ \rho - 1)$. The generator polynomial $\mathbf{g}(X)$ is a minimum-weight code polynomial, so that all g_i are nonzero factors. The original generator matrix should be a $(p^s - \rho + 1) \times (p^s - 1)$ matrix. We shorten it by deleting the first $p^s - \rho - 1$ information symbols, and then we obtained this shortened RS code with only two information symbols. The generator matrix becomes

$$\mathbf{G}_b = \begin{bmatrix} g_0 & g_1 & g_2 & \dots & 1 & 0 \\ 0 & g_0 & g_1 & g_2 & \dots & 1 \end{bmatrix}, \tag{2.34}$$

and it is a $(\rho,\ 2,\ \rho - 1)$ shortened RS code. The linear combinations of these two rows of $\mathbf{G}_b$ over $GF(p^s)$ generates $(p^s)^2$ codewords. We define the codeword set to be C_b. The nonzero codewords of C_b have only two different weights, $\rho - 1$ and ρ, hence the minimum distance of C_b is $\rho - 1$. Therefore, two codewords in C_b have at most one location with the same code symbol.

Then we separate the codeword set C_b into p^s subset $C_b^{(1)} \sim C_b^{(p^s)}$. To complete the partition, we define R_1 as the first information symbol and R_2 to be the second one, so that we have $R_1 = (g_0\ g_1\ ...\ 1\ 0)$ and $R_2 = (0\ g_0\ g_1\ ...\ 1)$. Adding R_1 and R_2, we will have a codeword in C_b with weight ρ. Then we construct

$$C_b^{(1)} = \{c(R_1 + R_2)\ :\ c \in GF(p^s)\}. \tag{2.35}$$

Since $(R_1 + R_2)$ is a codeword with weight ρ, each codeword in the set $C_b^{(1)}$ differs from others at every locations, hence the minimum distance is ρ. Continuing the process, we construct the ith subset as

$$C_b^{(i)} = \{\alpha^{i-2} R_1 + c(R_1 + R_2) \ : \ c \in GF(p^s)\} \ for \ 2 \leq i \leq p^s. \tag{2.36}$$

Note that each subset has p^s codeword, and each code word has ρ codeword symbols. We denote a codeword as $\mathbf{c} = (c_1, c_2, ..., c_\rho)$. Observing the subsets, the two codewords in the same subset must differ in all ρ locations, and two codewords in different subsets differ in $\rho - 1$ locations.

Another p^s-tuple vector $\mathbf{z}$ is defined as location vector over $GF(2)$, where it can be represented as $\mathbf{z} = (z_\infty, z_0, z_1, ..., z_{p^s-2})$. each z_i corresponds to α^i, which is an element of $GF(p^s)$, and $\alpha^\infty = 0$. We represent this location factor $\mathbf{z}(\alpha^i)$ as

$$\begin{array}{lccccccccccc} & & z_\infty & z_0 & ... & z_{i-1} & z_i & z_i + 1 & ... & z_{p^s-2} & \\ \mathbf{z}(\alpha^i) & =(& 0 & 0 & ... & 0 & 1 & 0 & ... & 0 &). \\ position & & 1 & 2 & ... & i+1 & i+2 & ... & ... & p^s & \end{array} \tag{2.37}$$

That is, the component of z_i is equal to 1 and others to be 0. Then we expend the location factor $\mathbf{z}$ to a ρp^s-tuple symbol location vector $\mathbf{Z}$ over $GF(2)$,

$$\mathbf{Z}(\mathbf{c}) = (\mathbf{z}(c_1), \mathbf{z}(c_2), ..., \mathbf{z}(c_\rho)). \tag{2.38}$$

Because the weight of each $\mathbf{z}(c_j)$ is 1, the weight of $\mathbf{Z}(\mathbf{c})$ is ρ.

We construct a $p^s \times \rho p^s$ matrix $\mathbf{A}_i$ as following,

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{Z}(\mathbf{c}_{i1}) \\ \mathbf{Z}(\mathbf{c}_{i2}) \\ \vdots \\ \mathbf{Z}(\mathbf{c}_{ip^s}) \end{bmatrix} \ for \ \mathbf{c}_{ij} \in C_b^{(i)}, \tag{2.39}$$

and note that each $\mathbf{c}_{ij}$ is non-identical to others. Since two $\mathbf{c}_{ij}$ in the same set $C_b^{(i)}$ differs from every position, two symbol location vectors in $\mathbf{A}_i$ must have no 1-component in common. So that each row of $\mathbf{A}_i$ will have ρ 1-components and each column has only one 1-component. $\mathbf{Z}(\mathbf{c}_{ij})$ from two different $\mathbf{A}_i$ will have only one 1-component in common since the distance between two $\mathbf{c}_{ij}$ is $\rho - 1$. And the matrix $\mathbf{A}_i$ could be considered as a structured matrix with ρ permutation matrices of $p^s \times p^s$.

Choose the factor γ, which is a positive integer and $1 \leq \gamma < p^s$. We form the following $\gamma p^s \times \rho p^s$ matrix as

$$\mathbf{H}(\gamma) = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_\gamma \end{bmatrix}. \tag{2.40}$$

This matrix will have ρ 1-components in each row and γ 1-components in each column, as a (γ, ρ)-regular matrix. The property that only one 1-component in common between rows also tells us that the girth of this matrix is absolutely larger than 4. This matrix $\mathbf{H}(\gamma)$ has been complete as the parity check matrix of a RS-based LDPC code by above steps.

We use a RS-based $(3, 3)$-regular LDPC code as an example. Considering $GF(2^2)$ with primitive polynomial $p(X) = 1 + X + X^2$, the four elements of this field are $0 = \alpha^\infty$, $1 = \alpha^0$, α and $\alpha^2 = 1 + \alpha$. Choosing $\rho = 3$, the generator polynomial will be

$$\mathbf{g}(X) = \alpha + X, \tag{2.41}$$

and the shortened generator matrix is

$$\mathbf{G}_b = \begin{bmatrix} \alpha & 1 & 0 \\ 0 & \alpha & 1 \end{bmatrix}. \tag{2.42}$$

Construct the subsets $C_b^{(1)} \sim C_b^{(4)}$ of the codeword set according to the rule described in (2.35) and (2.36),

$$\begin{array}{lcllllll}
C_b^{(1)} & = & \{ & \mathbf{0} & 1 \cdot (R_1 + R_2) & \alpha(R_1 + R_2) & \alpha^2(R_1 + R_2) & \} \\
C_b^{(2)} & = & \{ & 1 \cdot R_1 & R_1 + (R_1 + R_2) & R_1 + \alpha(R_1 + R_2) & R_1 + \alpha^2(R_1 + R_2) & \} \\
C_b^{(3)} & = & \{ & \alpha \cdot R_1 & \alpha R_1 + (R_1 + R_2) & \alpha R_1 + \alpha(R_1 + R_2) & \alpha R_1 + \alpha^2(R_1 + R_2) & \} \\
C_b^{(4)} & = & \{ & \alpha^2 \cdot R_1 & \alpha^2 R_1 + (R_1 + R_2) & \alpha^2 R_1 + \alpha(R_1 + R_2) & \alpha^2 R_1 + \alpha^2(R_1 + R_2) & \}.
\end{array}$$

With $R_1 = (\alpha\ 1\ 0)$, $R_2 = (0\ \alpha\ 1)$ and $R_1 + R_2 = (\alpha\ \alpha^2\ 1)$, we represent the equation above as

$$\begin{array}{lcllllll}
C_b^{(1)} & = & \{ & (0\ 0\ 0) & (\alpha\ \alpha^2\ 1) & (\alpha^2\ 1\ \alpha) & (1\ \alpha\ \alpha^2) & \} \\
C_b^{(2)} & = & \{ & (\alpha\ 1\ 0) & (0\ \alpha\ 1) & (1\ 0\ \alpha) & (\alpha^2\ \alpha^2\ \alpha^2) & \} \\
C_b^{(3)} & = & \{ & (\alpha^2\ \alpha\ 0) & (1\ 1\ 1) & (0\ \alpha^2\ \alpha) & (\alpha\ 0\ \alpha^2) & \} \\
C_b^{(4)} & = & \{ & (1\ \alpha^2\ 0) & (\alpha^2\ 0\ 1) & (\alpha\ \alpha\ \alpha) & (0\ 1\ \alpha^2) & \}.
\end{array} \tag{2.43}$$

The location vectors are

$$\begin{aligned} \mathbf{z}(0) &= \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \\ \mathbf{z}(1) &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \\ \mathbf{z}(\alpha) &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \\ \mathbf{z}(\alpha^2) &= \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned} \tag{2.44}$$

Combine (2.43) and (2.44), the 4×12 matrix constructed from $C_b^{(1)}$ is

$$\mathbf{A}_1 = \begin{bmatrix} \mathbf{Z}(\mathbf{c}_{11}) \\ \mathbf{Z}(\mathbf{c}_{12}) \\ \mathbf{Z}(\mathbf{c}_{14}) \\ \mathbf{Z}(\mathbf{c}_{14}) \end{bmatrix} = \left[\begin{array}{cccc:cccc:cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{2.45}$$

Construct $\mathbf{A}_2$, $\mathbf{A}_3$ and $\mathbf{A}_4$ based on $C_b^{(2)}$, $C_b^{(3)}$ and $C_b^{(4)}$ respectively with the same form. Then, choose $\gamma = 3$, the parity matrix $\mathbf{H}(3)$ is formed as Fig. 2.8. The null space gives a $(12,\ 4)$ LDPC code of length $= 12$, rate $= 1/3$. The rate is related with the rank of $\mathbf{H}$. That is, since the dimension of this $\mathbf{H}$ is 8, the number of parity bits of a codeword is 8, and the number of information bit is $12 - 8 = 4$.

$$\mathbf{H}(3) = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \end{bmatrix} = \left[\begin{array}{cccc:cccc:cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Figure 2.8: Parity-check matrix of RS-based (3, 3)-regular LDPC code

Moreover, apply the method to construct a (6, 32)-regular LDPC code which is relative to the adopted code in IEEE 802.3an. It is constructed in $GF(2^6)$, so the primitive polynomial is $p(X) = 1 + X + X^6$. ρ and γ is chosen as 32 and 6 respectively. As a result, we will have a parity check matrix $\mathbf{H}_{384\times2048}$. The construct of $\mathbf{H}$ can be roughly shown as Fig. 2.9.

It is reminded that each row of $\mathbf{H}$ is constructed by one codeword of the shortened RS code, which has 32 elements, and every element can be expanded to 64 bits by location

$$\mathbf{H}=\left.\overbrace{\begin{bmatrix} \mathbf{H}^{0,0}_{64\times64} & \cdots & \cdots & \cdots & \mathbf{H}^{0,31}_{64\times64}\\ \vdots & & \ddots & & \vdots \\ \mathbf{H}^{5,0}_{64\times64} & \cdots & \cdots & \cdots & \mathbf{H}^{5,31}_{64\times64}\end{bmatrix}}^{\rho=32}\right\}\gamma=6$$

Figure 2.9: The roughly view of H of (6, 32)-regular LDPC code

vector **z**. So the code length is 2048. Besides, every 64 rows are based on one subset of shortened RS code, in which two codewords differ from other at every location. Then every submatrix $\mathbf{H}^{i,j}$ is a permutation matrix of size 64 × 64, that means, each row and each column will have only one 1-component in $\mathbf{H}^{i,j}$. Although there are 384 rows in **H**, the number of parity bits is less than 384, because **H** is not full rank. In other words, not every row is independent, and the dimension of **H** is only 325. Therefore, this parity matrix correspond to a (2048, 1723) LDPC code, and the rate is about 0.84. Fig. 2.10 shows the parity check matrix of (6, 32)-regular LDPC code, each spot represents a 1-component in **H**.

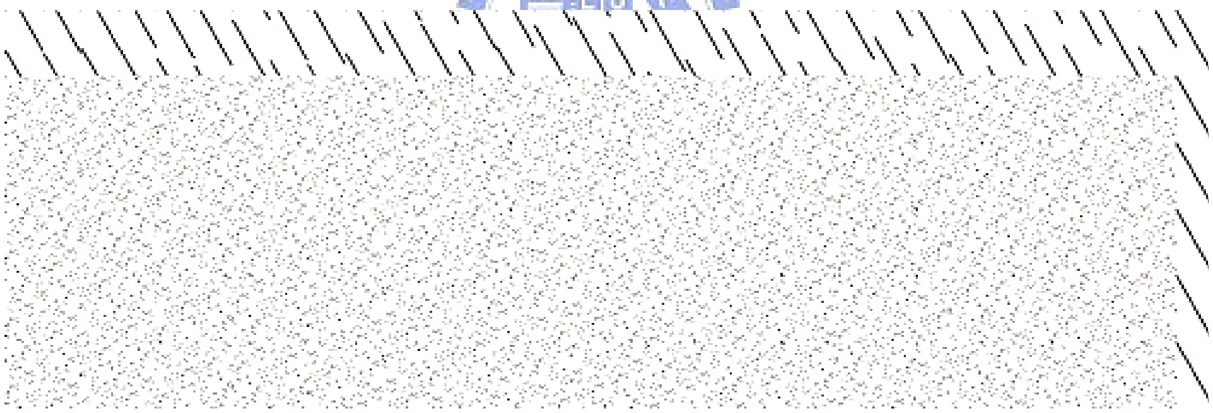


Figure 2.10: Parity check matrix of (2048, 1723) RS-based LDPC code

2.3 LDPC Code for IEEE 802.3an

IEEE 802.3an is an amendment to IEEE 802.3. It specifies a new physical coding sublayer interface and a new physical medium attachment sublayer interface for 10Gb/s Ethernet over copper (10GBASE-T) [6].

The error control code (ECC) used in IEEE 802.3an is a (2048, 1723) LDPC code, which is a (6, 32)-regular LDPC code constructed based on RS-based LDPC codes. This code meets the target bit error rate (BER) of the 10GBASE-T, that means, this code can reach BER=10^{-12} with other modulation and coding schemes. The generator matrix and parity check matrix used in IEEE 802.3an is available at:

http://standards.ieee.org/downloads/802/802.3an-2006/matrices.zip.

Each number in the files represents the column index of 1-component.

However, the parity check matrix is not exactly the same with the **H** constructed by the RS-based LDPC code. The columns have been swapped hence the null space of the parity check matrix can form a systematic generator matrix. And it also increases the minimum distance and improves the error floor.

Chapter 3

Implementations of BP Algorithm

We have introduced the LDPC decoding algorithm (also called BP algorithm) in chapter 2. It is shown that complicated calculation in this process, such as hyperbolic tangent in trigonometry, is not proper for implementation. So we have to find some simpler method to approximate it. In this chapter, two approximate methodologies are introduced, and another decoding algorithm that can decrease the number of iterations for reaching same BER performance is also presented. Besides, the simulation results will be also displayed after each section.

Since we try to implement the LDPC code for IEEE 802.3an, the following simulations are all based on the parity check matrix defined in the standard. The simulation environment is set by the C language, and all the results are signal-to-noise ratio (SNR) versus bit-error-rate (BER) regardless of the changing of the parameters. And we use AWGN noise channel, BPSK modulation.

Before starting the approximation, we have to set two parameters, iteration number and the quantization bits needed. First, we provide the performance for different maximum iterations with conventional BP algorithm in Fig. 3.1. All the functions we need are computed with the libraries of C.

The step of symdrome check in BP algorithm may not always converge to zero. So we need to set the maximum number of iterations to stop the decoding process, and to ouput the values computed so far. The maximum iteration number affects the performance of an LDPC code. But the gain of performance decreases when the maximum iteration number becomes larger. It is because that we assume all inputs of a node are independent

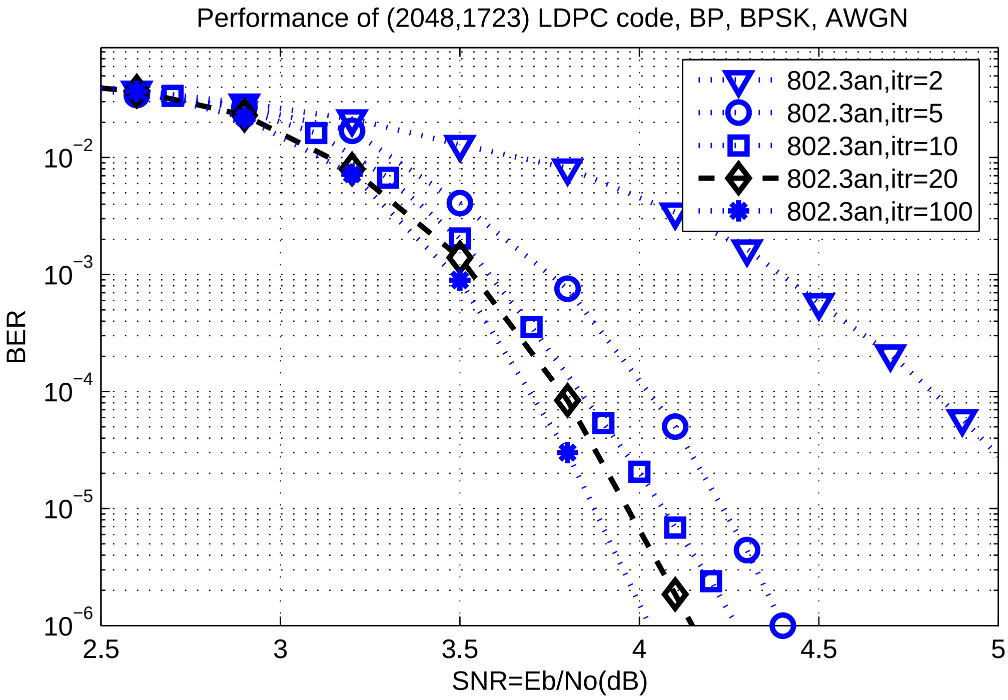


Figure 3.1: Simulation result (1): comparison of different iterations

when applying with the BP algorithm, but actually the dependency of messages grows up with the number of iterations while using iterative decoding scheme. So the conditions are not always true. As the simulation result of Fig. 3.1 shows, at the BER of 10^{-6}, the performance for 20 iterations has about 0.1 dB loss than for 100 iterations, and 10 iterations has about 0.2 dB loss. We consider the loss of iteration number of 20 is acceptable.

Then we try to derive the quantization bits needed to present the message. We have to process simulations to determine the proper parameters. The maximum decoding iteration number is set to 20 with applying BP algorithm while running simulations. If we fix the fraction part at 2 bits, the integer part at 3, 4 and 5 bits is simulated as Fig. 3.2 shows. We can find that the integer part bits equal to 4 is the best choice because that the performance of 4 bits is similar to performance of 5 bits, and much better than performance of 3 bits. With the same method, we fixed the integer part at 4 bits, and find that the fraction part bits equal to 2 performs best.

According to the simulation results above, we set the maximum iteration number of 20, integer part is 4 bits and fraction part is 2 bits.

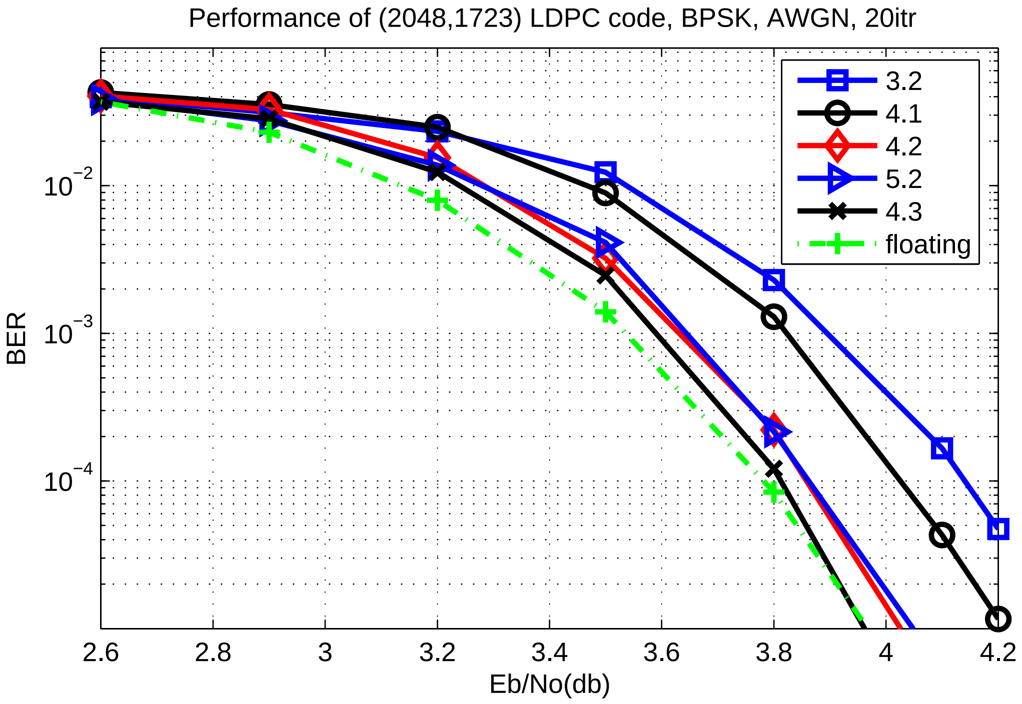


Figure 3.2: Simulation result (2): fixed-point simulation

3.1 Min-Sum Approximation

Since the most complicated calculation is in the check node update process, we analyze the check node update equation and try to simplified it. The first approximation is called **min-sum algorithm** [10].

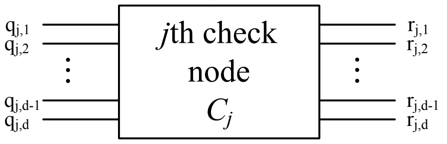


Figure 3.3: The inputs and the outputs of a check node

According to equation (2.28) and (2.30), the LLR message $r_{j,i}$ passing from check node C_j with d degrees to the ith edge of C_j can be rewritten as

$$r_{j,i} = 2\tanh^{-1}\left(\prod_{l=1,l\neq i}^{d} \tanh\left(\frac{q_{j,l}}{2}\right)\right), \tag{3.1}$$

where $q_{j,i}$ is the LLR message passing from the ith edge of C_j to the check node C_j. Fig. 3.3 represents the notation of the inputs and the outputs for node C_j. And the hyperbolic tangent is defined as

$$\tanh\left(\frac{x}{2}\right) = \frac{e^x - 1}{e^x + 1}. \tag{3.2}$$

Define a function Ψ for x> 0 as

$$\Psi(x) = \Psi^{-1}(x) = \ln\frac{1+e^{-x}}{1-e^{-x}} = -\ln(\tanh(\frac{x}{2})). \tag{3.3}$$

Then rewrite the term in equation (3.1),

$$\begin{aligned}\prod_{l=1,l\neq i}^{d}\tanh\left(\frac{q_{j,l}}{2}\right) &= \left(\prod_{l=1,l\neq i}^{d}\operatorname{sign}\left(\tanh\left(\frac{q_{j,l}}{2}\right)\right)\right)\exp\left(\sum_{l=1,l\neq i}^{d}\ln\left|\tanh\left(\frac{q_{j,l}}{2}\right)\right|\right)\\ &= \left(\prod_{l=1,l\neq i}^{d}\operatorname{sign}\left(q_{j,l}\right)\right)\exp\left(\sum_{l=1,l\neq i}^{d}\ln\left(\tanh\left(\frac{|q_{j,l}|}{2}\right)\right)\right),\end{aligned} \tag{3.4}$$

note that the sign of $\tanh\left(\frac{q_{j,i}}{2}\right)$ is the same with $q_{j,i}$. Thus we have

$$r_{j,i} = \left(\prod_{l=1,l\neq i}^{d}\operatorname{sign}\left(q_{j,l}\right)\right)\Psi^{-1}\left(-\sum_{l=1,l\neq i}^{d}\ln\left(\tanh\left(\frac{|q_{j,l}|}{2}\right)\right)\right). \tag{3.5}$$

For any integer t, derive Ψ^{-1} with sign as

$$(-1)^t\Psi^{-1}(x) = \ln\frac{1+(-1)^t e^{-x}}{1-(-1)^t e^{-x}}. \tag{3.6}$$

Then, we can rewrite equation (3.5) as

$$r_{j,i} = \left(\prod_{l=1,l\neq i}^{d}\operatorname{sign}\left(q_{j,l}\right)\right)\Psi^{-1}\left(\sum_{l=1,l\neq i}^{d}\Psi\left(|q_{j,l}|\right)\right). \tag{3.7}$$

This equation is much easier for implementation compared with equation (3.1) since it uses additions instead of multiplications.

For the operation of $\Psi(x)$, we plot this function as Fig. 3.4. As the figure shows, the result of Ψ becomes larger when the x gets smaller. So, considering the equation (3.7), the term $\sum_{l=1,l\neq i}^{d}\Psi\left(|q_{j,l}|\right)$ will be dominated by the smallest $|q_{j,l}|$. Thus we rewrite the equation (3.7) in approximate term as

$$r_{j,i} \approx \left(\prod_{l=1,l\neq i}^{d}\operatorname{sign}\left(q_{j,l}\right)\right)\min_{l=\{1\sim d\}\backslash i}|q_{j,l}|. \tag{3.8}$$

This is called min-sum algorithm. However, according to the simulation result with 20 decoding iterations shown in Fig. 3.5, the BER performance loss is large, about 0.5dB at BER=10^{-6}.

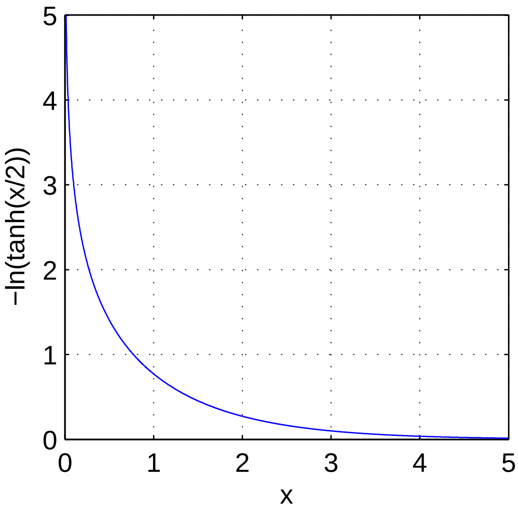


Figure 3.4: The function $\Psi(x) = -\ln(\tanh(\frac{x}{2}))$

Therefore, some compensation methods have been brought up to reduce the loss of performance with BP algorithm. Comparing the equations (3.7) with (3.8), we find that the message $r_{j,i}$ of min-sum algorithm will be larger than the BP algorithm. Then the first method called offset compensation is expressed as

$$r_{j,i} \approx \left(\prod_{l=1, l \neq i}^{d} \text{sign}\left(q_{j,l}\right) \right) \min_{l=\{1 \sim d\} \setminus i} |q_{j,l}| - \alpha, \tag{3.9}$$

where the compensation factor α satisfies $\alpha > 0$. Another method named normalization compensation is represent as

$$r_{j,i} \approx \beta \times \left(\prod_{l=1, l \neq i}^{d} \text{sign}\left(q_{j,l}\right) \right) \min_{l=\{1 \sim d\} \setminus i} |q_{j,l}|, \tag{3.10}$$

where β is also a compensation factor with $0 < \beta \leq 1$. Equation (3.10) often performs better than (3.9), because α is only a constant and would not change its value for different input information. That is, if we do not know the range of values of $r_{j,i}$, it is hard to choose a proper α. Hence we use the second compensation method for approximation. The simulation result with choosing $\beta = 0.75$ is also shown in Fig. 3.5, and it shows that the BER performance is quite similar to BP algorithm.

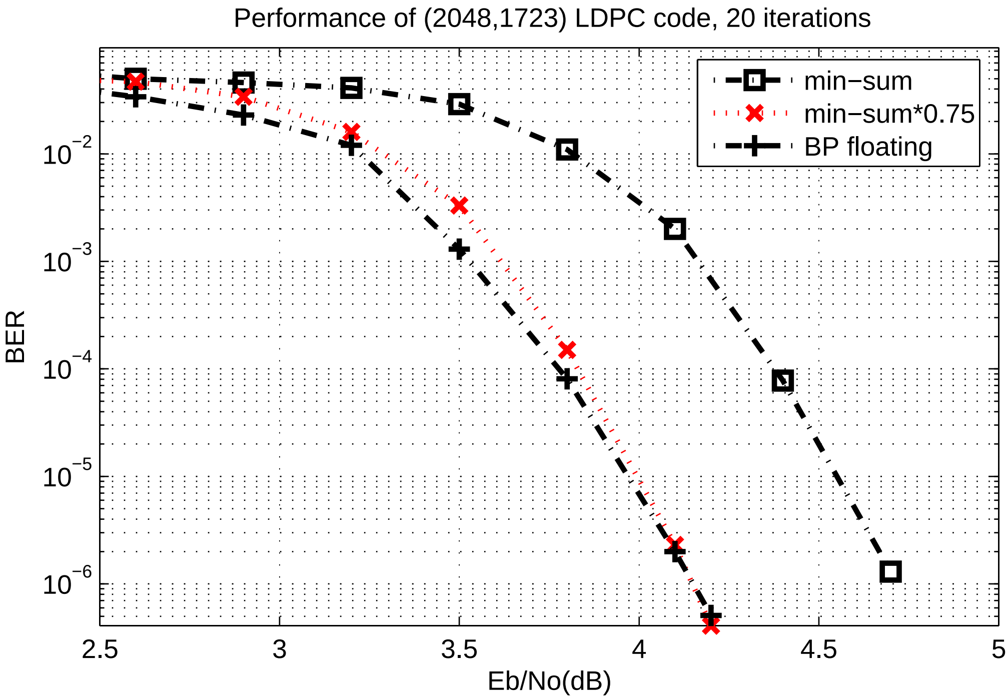


Figure 3.5: Simulation result (3) : comparision of BP algorithm and min-sum algorithm

3.2 Look-up Table Approximation

Another approximation method we have tried is look-up table (LUT) approximation instead of the mathematical functions [11]. In chapter 2, we have already derived the check node updating operation as

$$r_{j,i} = CHK^{d}_{l=1,l\neq i}(\sum \oplus q_{j,l}) = CHK(q_{j,l} \oplus ... \oplus q_{j,i-1} \oplus q_{j,i+1} \oplus ... \oplus q_{j,d}), \qquad (3.11)$$

where the notations are the same as Fig. 3.3.

According to the equation (2.28), the check node operation with two edges $q_{j,m}$ and qj,n can be rewritten as

$$\begin{aligned} CHK(q_{j,m}, q_{j,n}) &= CHK(q_{j,m} \oplus q_{j,n}) = \ln \frac{1 + e^{q_{j,m}} e^{q_{j,n}}}{e^{q_{j,m}} + e^{q_{j,n}}} \\ &= \ln(1 + e^{q_{j,m}+q_{j,n}}) - \ln(e^{q_{j,m}} + e^{q_{j,n}}) \\ &= \max[0, (q_{j,m} + q_{j,n})] + \ln(1 + e^{-|q_{j,m}+q_{j,n}|}) \\ &\quad - \max[q_{j,m}, q_{j,n}] - \ln(1 + e^{-|q_{j,m}-q_{j,n}|}) \\ &= \text{sign}(q_{j,m}) \times \text{sign}(q_{j,n}) \times \min(|q_{j,m}|, |q_{j,n}|) \\ &\quad + \ln(1 + e^{-|q_{j,m}+q_{j,n}|}) - \ln(1 + e^{-|q_{j,m}-q_{j,n}|}). \qquad (3.12) \end{aligned}$$

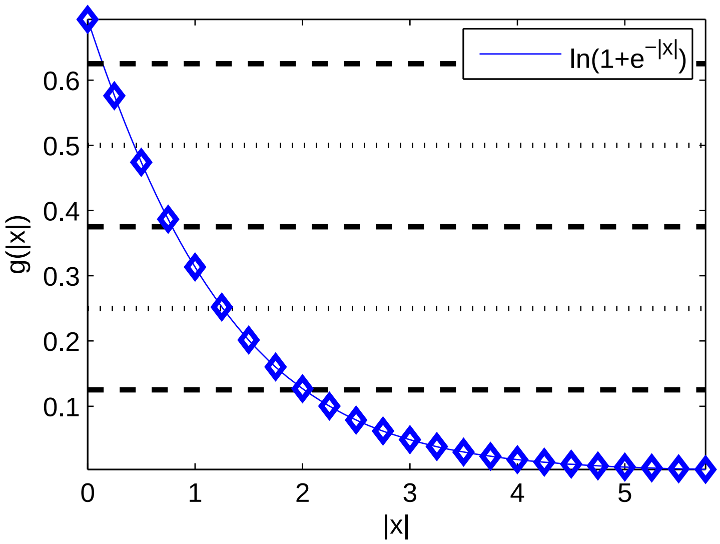


Figure 3.6: The function $g(x) = \ln(1 + e^{-|x|})$

Table 3.1: Quantization table for $g(x) = \ln(1 + e^{-|x|})$

$\|x\|$	$\ln(1 + e^{-\|x\|})$
0	0.75
[0.25, 0.75]	0.50
[1.00, 2.00]	0.25
≥ 2.25	0.00

The term $\text{sign}(q_{j,m}) \times \text{sign}(q_{j,n}) \times \min(|q_{j,m}|, |q_{j,n}|)$ can be easily implemented, so the term $\ln(1+e^{-|q_{j,m}+q_{j,n}|}) - \ln(1+e^{-|q_{j,m}-q_{j,n}|})$ is what we want to approximate with look-up table.

First, plot the function $g(x) = \ln(1 + e^{-|x|})$, as Fig. 3.6 shows. Since the fraction part is set to 2 bits, we mark the points with the interval 0.25 on x-axis. The dash lines lei on y = 0.125, y = 0.375 and y = 0.625, which indicate that the points falling between two lines are quantized as the middle value of these two lines. For example, the points lei between y = 0.125 and y = 0.375 will be quantized to the value 0.25. Base on the figure, we list the quantization table as Table 3.1, and the output of this table need only 2 bits. The maximum approximation error is less than 0.125.

Considering the general case as equation (3.11), the equation can be present as

$$\begin{aligned} CHK(q_{j,1}, q_{j,2}, ..., q_{j,d}) &= CHK(q_{j,l} \oplus q_{j,2} \oplus ... \oplus q_{j,d}) \\ &= CHK(CHK(...CHK(q_{j,l} \oplus q_{j,2})...) \oplus q_{j,d}). \qquad (3.13) \end{aligned}$$

This equation can be implemented in a serial configuration. So it needs $(d-2)$ time units in computing one output message because of $(d-1)$ inputs.

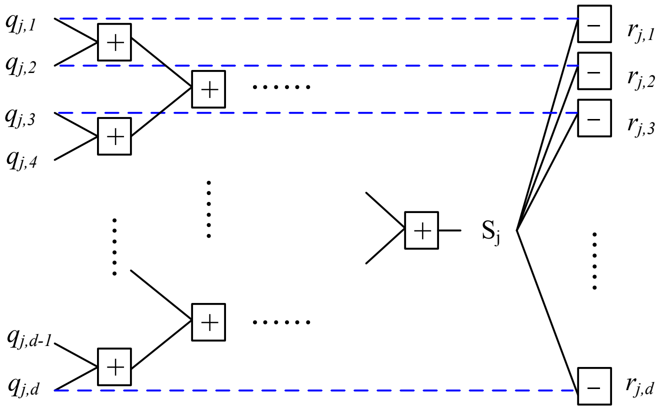


Figure 3.7: Parallel configuration for computing check-node updates

Since the latency is too long for high throughput requirements, there is another efficient implementation for computing check node update, called tree topology. It begins with defining an auxiliary binary random variable $S_j = CHK^d_{l=1}(\sum \oplus q_{j,l})$, where S_j of the check node j can be computed using the tree topology as Fig. 3.7. The operation at each node in the tree is also $CHK(q_{j,m} \oplus q_{j,n})$ as equation (3.12). Therefore, the latency of computing S_j will become $\log_2 d$, and it is much better than serial trellis topology, especially when the degree of check node is large. Then, we have to compute the outgoing message $r_{j,i}$. Now, describe S_j with respect to the input $q_{j,i}$, so

$$\begin{aligned} S_j &= CHK^d_{l=1}\left(\sum \oplus q_{j,l}\right) = CHK\left(q_{j,i} \oplus CHK^d_{l=1,l\neq i}\left(\sum \oplus q_{j,l}\right)\right) \\ &= \ln \frac{1+e^{CHK^k_{l=1,l\neq i}(\sum \oplus q_{j,l})+q_{j,i}}}{e^{CHK^k_{l=1,l\neq i}(\sum \oplus q_{j,l})}+e^{q_{j,i}}}. \end{aligned} \tag{3.14}$$

And as equation (3.11), $r_{j,i} = CHK^d_{l=1,l\neq i}(\sum \oplus q_{j,l})$ has been known, so it can be reorganized after algebra as

$$r_{j,i} = CHK^k_{l=1,l\neq i}\left(\sum \oplus q_{j,l}\right) = \ln \frac{e^{q_{j,i}+S_j}-1}{e^{q_{j,i}-S_j}-1} - S_j. \tag{3.15}$$

Finally, define the operation $CHK(q_{j,i} \ominus S_j)$, extrinsic information $r_{j,i}$ can be compute simultaneously by parallel implementation of this new operation as shown in Fig. 3.7. Similarly, $CHK(q_{j,i} \ominus S_j)$ can be approximated with look-up table, where the first term of the right hand side of the equation (3.15) can be expressed as $\ln|e^{q_{j,i}+S_j}-1| - \ln|e^{q_{j,i}-S_j}-1|$.

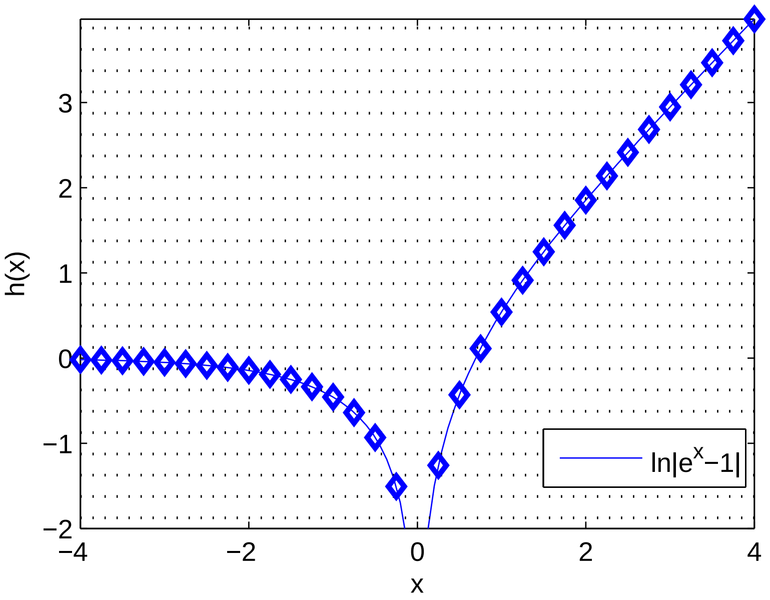


Figure 3.8: The function $h(x) = \ln|e^x - 1|$

Table 3.2: Quantization table for $h(x) = \ln|e^x - 1|$

x	$\ln\|e^x - 1\|$	x	$\ln\|e^x - 1\|$
≤ -2.25	0.00	0.50	-0.50
[-2.00, -1.25]	-0.25	[0.75, 1.25]	2x-1.50
[-1.00, -0.50]	-x-1.50	[1.50, 2.00]	x-0.25
[-0.25, 0.00]	-1.50	≥ 2.25	x
0.25	-1.25		

We calculate $h(x) = \ln|e^x - 1|$ and plot it as Fig. 3.8. Similarly, Fig. 3.8 is ploted with the interval of 0.25. We also construct a quantiation table as Table 3.2.

The result of applying the look-up table approximation described above with 20 decoding iterations is shown as Fig. 3.9. It shows that the loss of BER performance is quite small, about 0.05dB at BER=10^{-6}.

But the hardware costs of implementing LUT approximation is huge. According to Fig. 3.7 and equation(3.12), the operation of each node in the tree in Fig. 3.7 is $CHK(q_{j,m} \oplus q_{j,n})$, which includes two LUTs of $g(x) = \ln(1 + e^{-|x|})$ refered to Fig. 3.6. Similarly, the operation $CHK(q_{j,m} \ominus q_{j,n})$ costs two LUTs of $h(x) = \ln|e^x - 1|$ refered to Fig. 3.8. As a result, one check node operation of 32 degrees needs $2 \times 31 = 62$ tables of

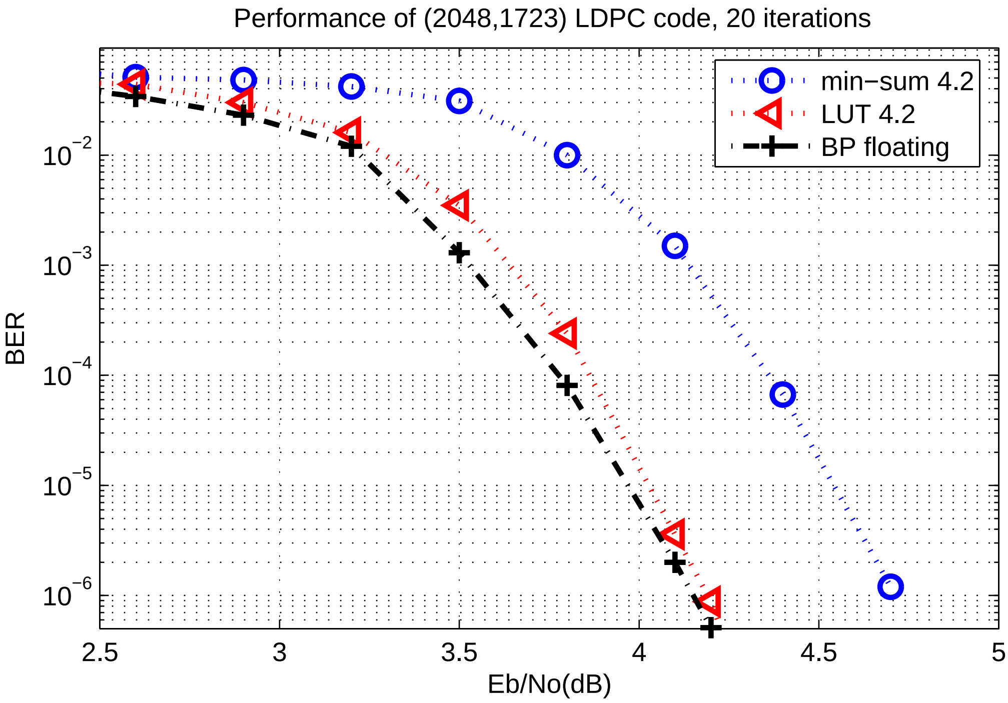
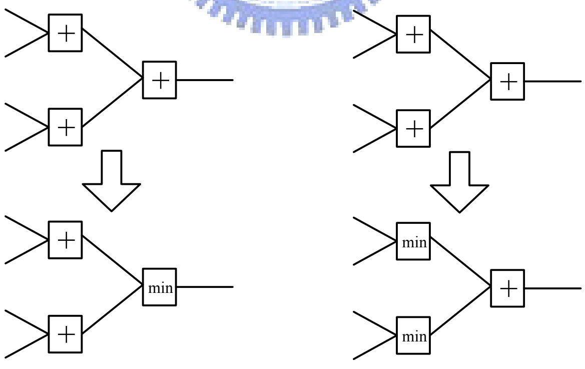


Figure 3.9: Simulation result (4) : the result of applying look-up table approximation $g(x) = \ln(1 + e^{-|x|})$ and $2 \times 32 = 64$ tables of $h(x) = \ln|e^x - 1|$. Synthesized with design compiler using UMC 0.13-μm cell library, The LUTs occupy about half of the total gate counts of the check node operation, and the gate counts increase rapidly with stricter timing constraints. So, it will also be the critical barrier to reduce the critical path.



(a) (b)

Figure 3.10: The simlified structure of LUT

For reducing the complexity, we try to replace part of $CHK(q_{j,m} \oplus q_{j,n})$ operations with $\text{sign}(q_{j,m}) \times \text{sign}(q_{j,n}) \times \min(|q_{j,m}|, |q_{j,n}|)$, similar to min-sum algorithm which has been described in section 3.1. By simulated with inputs of uniform distribution, the average difference between two structures of Fig. 3.10(a) is 0.114, which is smaller than the maximum approximation error. And the average difference of Fig. 3.10(b) is 0.177. Although it is larger than the approximation error, it is still smaller then the precision. The performance of the simplified structures is shown in Fig 3.11. Since there are 5 stages of $CHK(q_{j,m} \oplus q_{j,n})$ of check node operation with 32 degrees, we name the structure as 5 letters. 'S' represent the original operation $CHK(q_{j,m} \oplus q_{j,n})$, and 'm' means that we change the operation to $\text{sign}(q_{j,m}) \times \text{sign}(q_{j,n}) \times \min(|q_{j,m}|, |q_{j,n}|)$.

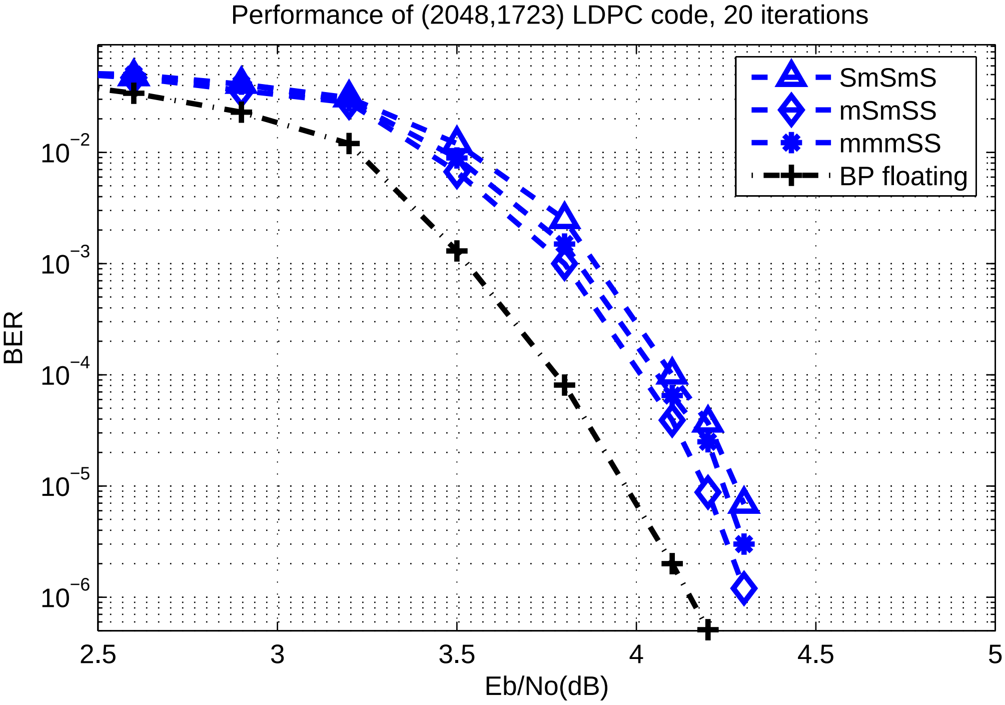


Figure 3.11: Simulation result (5) : the result of simplified structure of LUT

The BER performance loss of simplified structures 'mSmSS' and 'SmSmS' are about 0.2 ∼ 0.3 dB at BER=10^{-5}. We find that when the 'S' appears at later stages, the performance is better. So we also try the structure of 'mmmSS'. As a result, the BER performance of it is even better than 'SmSmS', though it has only two stages of calculating $CHK(q_{j,m} \oplus q_{j,n})$. In conclusion, if the later stages have more precise calculations, the BER performance will be better. Table 3.3 compares three check node structures synthesized with design compiler using UMC 0.13-μm cell library, and the synthesis results are based on the whole stages before S_j in Fig. 3.7 (the tree). Since 'SmSmS' has worse BER performance and costs more hardware, it is not listed in the table. Among these three

Table 3.3: Comparison of different check node structure

	original	mSmSS	mmmSS
no. of LUT of g(x)	62	22	6
critical path (ns)	15	12	11
total gate count (gates)	16.6K	12.8K	9.4K
BER at SNR=10^{-5} (dB)	4.0	4.2	4.25

structures, better performance follows more cost and worse timing constraints. So trade-off between hardware and perfomance is needed.

3.3 Layered Decoding Algorithm

The layered decoding algorithm [12] [13] allows updated information to be utilized more quickly thus speeding up the decoding, which is different from conventional BP algorithm. In conventional BP algorithm, we update all check nodes, and then update all variable nodes, and it is called one iteration. Layered decoding, as the name indicates, decodes in layers. It views a parity check matrix **H** as horizontal layers, and each layer represent a component code. The check node update and variable node update within a layer is called a sub-iteration, and one iteration indicates the process of completing every layers of **H** one by one. The simulation results shows that layered decoding algorithm can reduce nearly 50% of iterations for same BER performance [12].

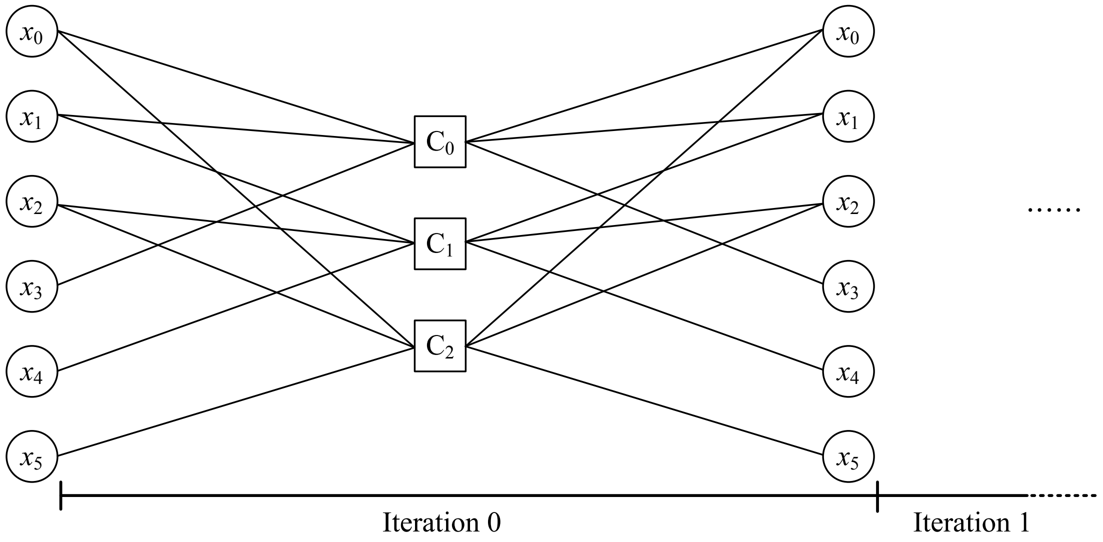
Divide the rows of **H** into G non-overlapping groups. The word non-overlapping indicates that each column of each group has weight of one at most. Denote the LLR of messages passing from mth check node to nth variable node at the ith iteration as $r_{m,n}^{(i)}$, and the opposite messages as $q_{m,n}^{(i)}$. $q_n^{(i\times G+g)}$ is denoted as the a posteriori LLR of nth variable at the gth sub-iteration of ith iteration. $N(m)$ is the set of variable nodes connecting with mth check node, $M(n)$ is the set of check nodes connecting with nth variable node, and $C(g)$ is the set of check nodes of group g. The decoding process of one group is described as below. At the ith iteration, the inputs of check node operation are the outputs of last layer or other prior layers, which is the most recently updated extrinsic message $q_{m,n}^{(i)}$ of variable nodes. After check node operation of this group, variable node update is calculated immediately. Since there is only one entry of each column at most, only one or none degree of each variable node has new input $r_{m,n}^{(i)}$. Then the $r_{m,n}^{(i-1)}$ of the last iteration will become a subtrahend and this new input will become an addend of the a posteriori value of the variable nodes. It can be organized as following:

Algorithm 1 Layered decoding

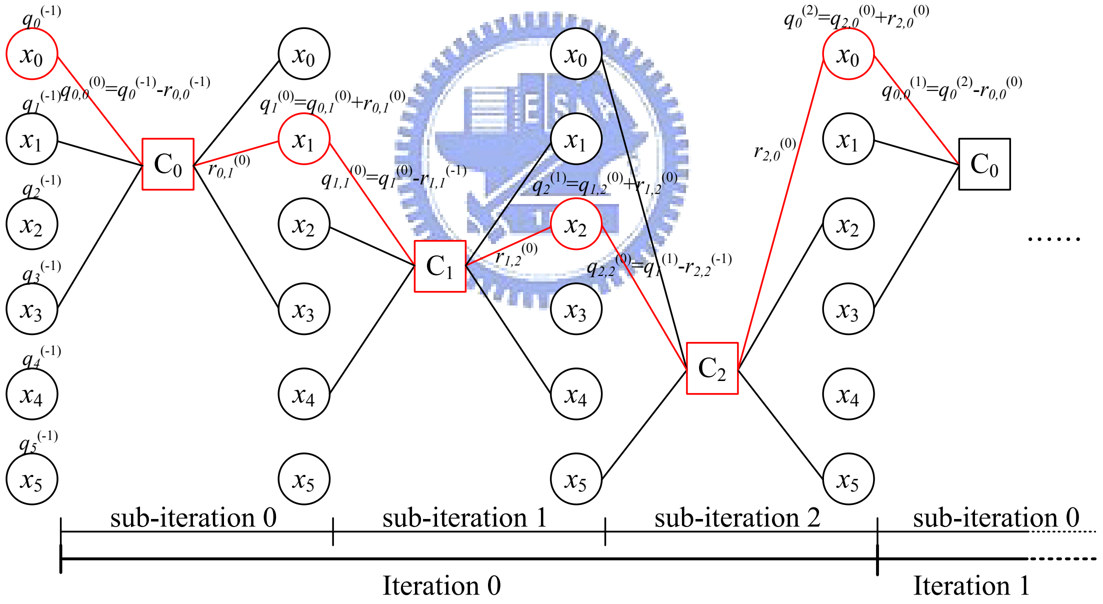
```
$r_{m,n}^{(-1)} = 0;\ q_n^{(-1)}$ =intrinsic information; //Initialization
for $i = 0$ to $(I_{max} - 1)$ do
    for $g = 0$ to $(G - 1)$ do
        $\forall m \in C(g),\ q_{m,n}^{(i)} = q_n^{(i\times G+g-1)} - r_{m,n}^{(i-1)};$  (1)
        $\forall m \in C(g),\ r_{m,n}^{(i)} = 2\tanh^{-1}\left\{\prod_{l\in N(m)\backslash n}\tanh\left(\frac{q_{m,l}^{(i)}}{2}\right)\right\};$  (2)
        $\forall m \in C(g),\ q_n^{(i\times G+g)} = q_{m,n}^{(i)} + r_{m,n}^{(i)};$  (3)
    end for
end for
```

We take the parity check matrix **H** of bipartite graph in Fig. 2.1 as an example. To satisfy the group condition of non-overlapping, this **H** is divided into 3 groups, and each group has only one check node. Fig. 3.12 illustrate the decoding procedure of conventional BP algorithm in (a) and layered decoding algorithm in (b). The tags in Fig. 3.12(b) are matched up with the pseudo code above, to make the procedure more clearly.

Consider the $(2048, 1723)$ LDPC code for IEEE 802.3an. It is a $(6, 32)$-regular LDPC code. So, based on the degree number of variable node, we partition the 384 check nodes into 6 groups equally. Hence there are 64 check nodes in each group. By observing the submatrix $\mathbf{H}_{64\times 2048}$, we find out that each column of $\mathbf{H}_{64\times 2048}$ will have exactly one entry due to the construction of this code, which is related to the construction of RS-based LDPC code. So it is appropriate applying layered decoding algorithm, because each group will renew the value of one degree of each variable node. We apply the layered decoding algorithm on both min-sum approximation and LUT approximation described in section 3.1 and 3.2. The results are shown in Fig. 3.13 and Fig.3.14. The BER performance loss is about only 0.1dB at BER=10^{-6} when applying with min-sum, and about $0.1 \sim 0.2$dB when applying with LUT, both are $6 \sim 8$ iterations of layered decoding algorithm, and compared with 20 iterations of the conventional BP approach.



(a) conventional BP algorithm


$q_0^{(-1)}$
x_0
$q_{0,0}^{(0)}=q_0^{(-1)}-r_{0,0}^{(-1)}$
$q_1^{(-1)}$
x_1
$q_2^{(-1)}$
x_2
$q_3^{(-1)}$
x_3
$q_4^{(-1)}$
x_4
$q_5^{(-1)}$
x_5
C_0
$r_{0,1}^{(0)}$
$q_1^{(0)}=q_{0,1}^{(0)}+r_{0,1}^{(0)}$
$q_{1,1}^{(0)}=q_1^{(0)}-r_{1,1}^{(-1)}$
C_1
$r_{1,2}^{(0)}$
$q_2^{(1)}=q_{1,2}^{(0)}+r_{1,2}^{(0)}$
$q_{2,2}^{(0)}=q_1^{(1)}-r_{2,2}^{(-1)}$
C_2
$r_{2,0}^{(0)}$
$q_0^{(2)}=q_{2,0}^{(0)}+r_{2,0}^{(0)}$
$q_{0,0}^{(1)}=q_0^{(2)}-r_{0,0}^{(0)}$
C_0
……
sub-iteration 0
sub-iteration 1
sub-iteration 2
sub-iteration 0
Iteration 0
Iteration 1

(b) layered decoding algorithm

Figure 3.12: Illustration of the process of conventional BP algorithm and layered decoding algorithm

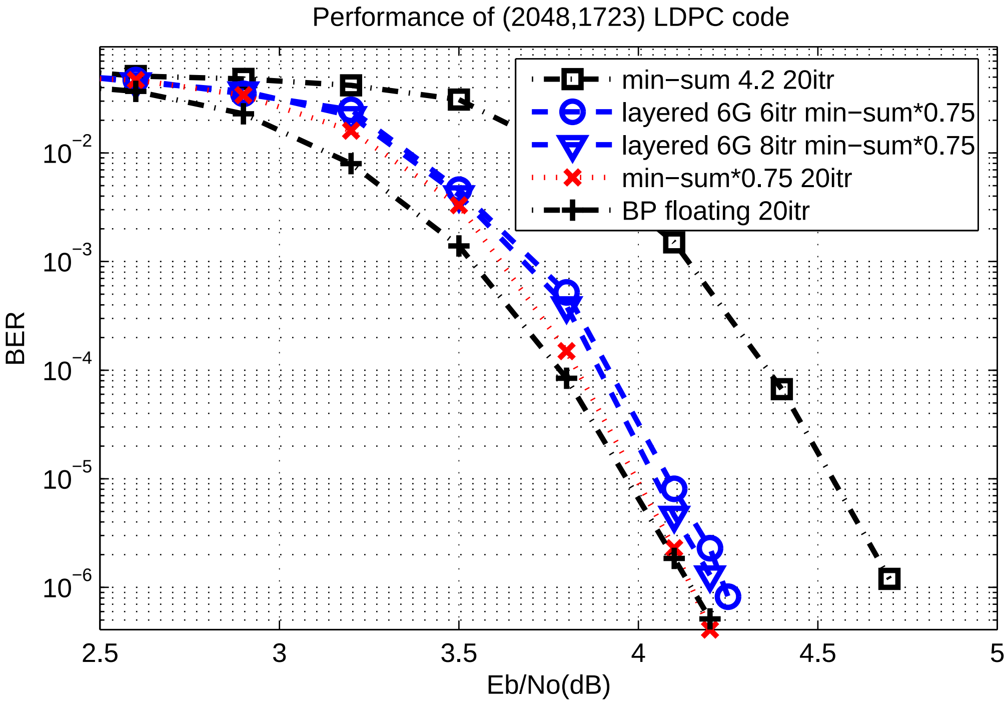


Figure 3.13: Simulation result (6) : layered decoding algorithm applied with min-sum approximation

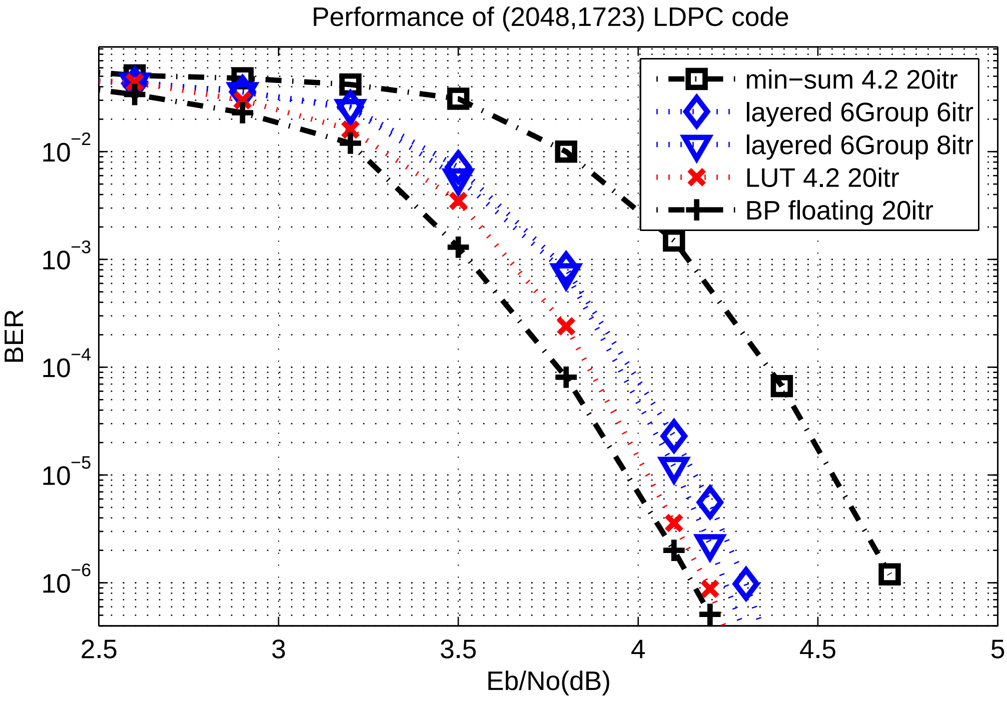


Figure 3.14: Simulation result (7) : layered decoding algorithm applied with look-up table approximation

Chapter 4

Proposed Partial Parallel Architecture

The design of the LDPC decoder applied to IEEE 802.3an standard will be proposed and described in details in this chapter. According to the simulation results of the approximation methods described in chapter 3, considering both hardware cost and BER performance, min-sum approach with normalization factor is adopted. Besides, the partial parallel scheme is used since it is a structured LDPC code. The layered decoding algorithm is also applied for reducing the iterations needed while reaching the same BER perfomance. The last section will present the implementation result of the proposed architecture.

4.1 Overview

As described in section 3.3, the LDPC code for IEEE 802.3an is a $(6, 32)$-regular code. Hence we partition the 384 check nodes of parity check matrix $\mathbf{H}_{384\times2048}$ into 6 groups equally, and find out that each submatrix $\mathbf{H}_{64\times2048}$ will have exactly on entry at each column. Whlie applying the layered decoding algorithm, each variable node will have exactly one renew value in each sub-iteration, so the architecture of variable nodes is regular.

The brief overview of the proposed decoder is presented as Fig. 4.1. VNU and CNU represent the variable node unit and the check node unit, respectively. As the figure

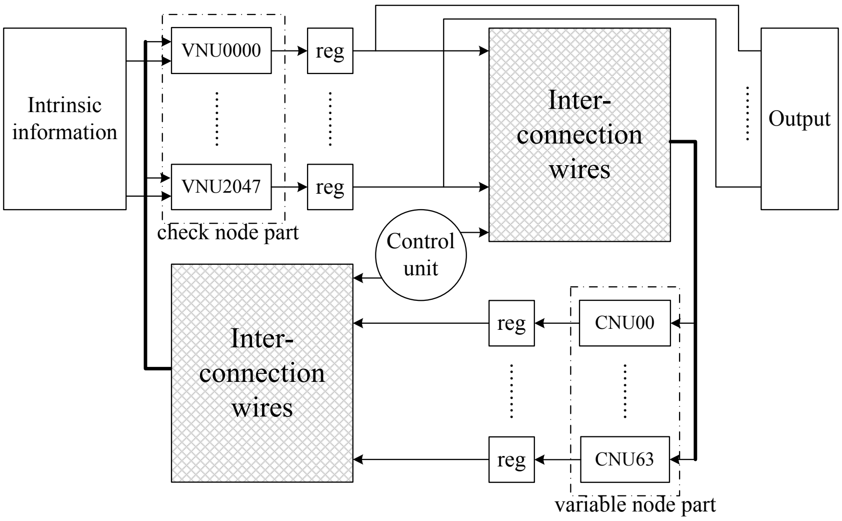


Figure 4.1: The brief overview of the proposed LDPC decoder architecture

cycle	**0**	**1**	**2**	**3**	**4**	....	**10**	**11**	**12**	**13**	....
CNU	$\mathbf{H}_{L0}$	$\mathbf{H}_{R0}$	$\mathbf{H}_{L1}$	$\mathbf{H}_{R1}$	$\mathbf{H}_{L2}$	....	$\mathbf{H}_{L5}$	$\mathbf{H}_{R5}$	$\mathbf{H}_{L0}$	$\mathbf{H}_{R0}$	....
VNU	Ini.	$\mathbf{H}_{L0}$	$\mathbf{H}_{R0}$	$\mathbf{H}_{L1}$	$\mathbf{H}_{R1}$		$\mathbf{H}_{R4}$	$\mathbf{H}_{L5}$	$\mathbf{H}_{R5}$	$\mathbf{H}_{L0}$	

Figure 4.2: The schedule of the decoder while running two codewords simulaneously

shows, check node update and variable node update need one cycle respectively. Since we partition the parity check matrix into 6 parts, we need 2×6 cycles to complete a iteration. Half of the cycles update check nodes and variable node update is done by another half. The hardware is uneconomical since half of it idle by turns. Therefore, we run two codewords at the same time. The two codewords share the hardware, that is, one codeword is processing check node update while another is processing variable node update, and being exchanged at the next cycle. The scheduled timing diagram is presented in Fig. 4.2. $\mathbf{H}_L$ and $\mathbf{H}_R$ represent the parity check matrices of two different codewords, then $\mathbf{H}_{Li}$ and $\mathbf{H}_{Ri}$ indicate the check nodes of i-th group of the first codeword and of the second codeword, respectively, for $i = 0 \sim 5$. The initial step of VNU at $cycle = 0$ sets the registers to the corresponding values. For example, a posteriori message q_n of nth variable node is set to the intrinsic message and the message $r_{m,n}$ from mth check node to nth variable node is set to zero. Therefore, the cycle number C_N needed for I iteration can be calculated as

$$C_N = 12 \times I + 1. \tag{4.1}$$

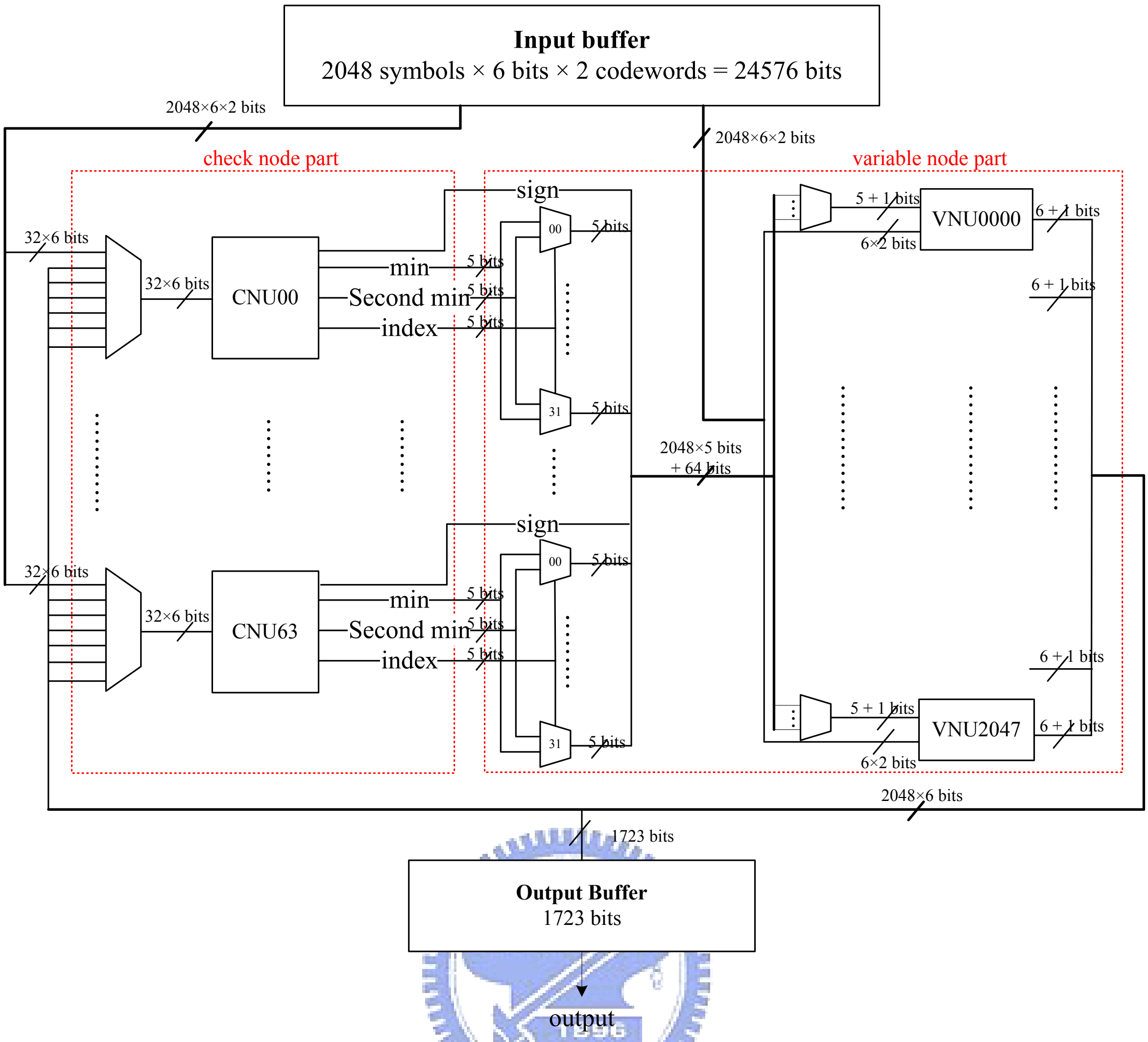


Figure 4.3: The overview of the proposed LDPC decoder architecture

The overhead of processing two codewords simultaneously is the increasing of the usage of storage. All messages passing from check node to variable node ($r_{m,n}$) has to be saved in the registers because we need $r_{m,n}$ of previous cycle to process each sub-iteration when applying layered decoding. Hence usage of the storage for saving $r_{m,n}$ is twice larger.

The calculation of check node units is much more complicated than the variable nodes, since the degree is much larger. The original processes of check node includes three parts. First, to find out the minimum and second minimum of the absolute values of inputs; second, to choose each output $|r_{m,n}|$ by the index of minimim value; third, to mutiply the outputs by the sign factor. In order to balance the critical paths in check node part and in variable node part, only the first step is done in the check node part, and other calculations is shifted to the variable node part. Fig. 4.3 presents the details of the connections between CNU and VNU, and shows the architectures of each part. Besides, as the process of choosing output shifted to variable node part, only sign, minimum value, second minimum value and index of minimum value have to be kept in the registers instead of all 32 $r_{m,n}$. Note that the $r_{m,n}$ of two codewords should be saved in variable node part according to this structure, and only q_n and $q_{m,n}$ for immediate cycle are needed to be kept instead of 6 q_n. Furthermore, the interconnections between these two parts also become much simpler because the decrease of connecting lines hence reduce the routing congestion. $|r_{m,n}|$ is transmitted instead of $r_{m,n}$, thus it can also be one bit less for transmissions from CNU to VNU.

4.2 Check Node Unit

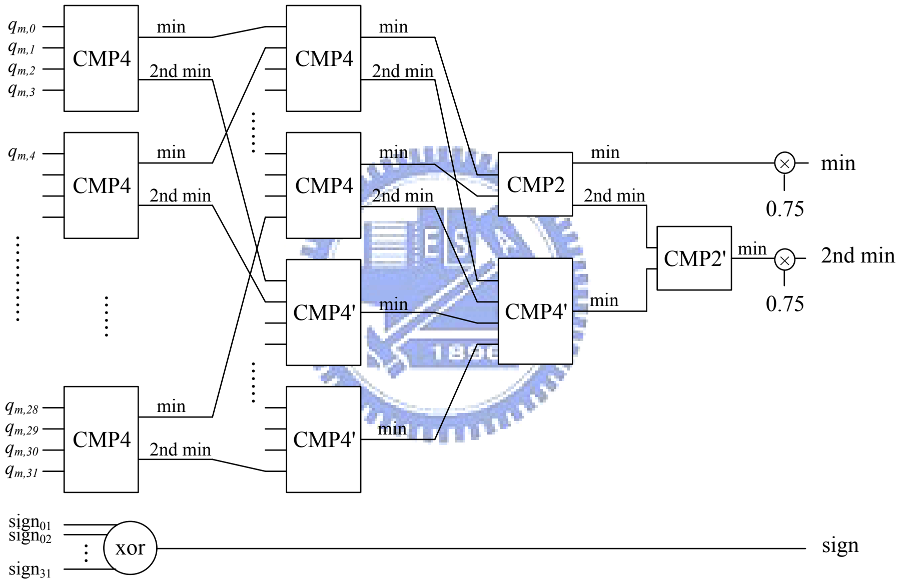


Figure 4.4: The architecture of check node unit.

The check node part of the decoder is discussed as following. According to the partial parallel scheme described above, there are 64 check nodes. Since it is a regular LDPC code, every check node unit (CNU) has the same number of degree of 32. Fig. 4.4 shows the architecture of one CNU. According to the min-sum algorithm and the process of balancing the critical paths which we have discussed at section 4.1, we only need to find out the minimum and second minimum of the absolute values of inputs $q_{m,n}$, the sign of the inputs, and the index of the minimum value for recognizing the output position of the second minimum value. Note that the calculation of index is the same with the minimum value thus it is not presneted in the figure. 4-input comparators is used instead of 2-input ones to decrease the stages and reduce the critical path.

4.3 Variable Node Unit

According to the description of algorithm 1 in section 3.3, for the layered decoding algorithm, only one degree will be renew in each variable node update operation. So, at the ith iteration, we need only an adder to add the new input and a subtractor to subtract the $r^{(i-1)}_{m,n}$ to complete the calculation. The value relative to new $r^{(i)}_{m,n}$ will be the input of the variable node unit (VNU) one by one, and this new $r^{(i)}_{m,n}$ will be used as a subtrahend after one iteration. By this property, we store new $r^{(i)}_{m,n}$ into a shift registers with the size of 12 in turns. The architecture of VNU is shown in Fig. 4.5. Besides, the equation (3.10) can be modified with the notation in section 3.3 as

$$r^{(i)}_{m,n} \approx \underbrace{\texttt{sign}(q^{(i)}_{m,n})}_{1st\ term} \underbrace{\beta \left(\prod_{l\in N(m)} \texttt{sign}(q^{(i)}_{m,l}) \right)}_{2nd\ term} \underbrace{\min_{l\in N(m)\backslash n} \left| q^{(i)}_{m,l} \right|}_{3rd\ term} . \tag{4.2}$$

The first term of the right hand side of this equation is kept in the register since $q^{(i)}_{m,n}$ is the output message of VNU of last two cycles. The second and third terms are available from the de-mutiplexer of choosing CNU output $\left| r^{(i)}_{m,n} \right|$ by the value of index as shown in Fig. 4.3, which is done before the VNU process. The details of the VNU process is shown as below, and the number of *cycle* used in the discription is refering to Fig. 4.2:

1. Initialization:

   At $cycle = 0$, set the shift registers to be zero with respect to the initial value of $r_{m,n}$, and output registers refer to the intrinsic information of $\mathbf{H}_{R0}$ since it will be the input of CNU at $cycle = 1$.

2. Calculate the output:

   At $cycle = 1$ to $cycle = 6 \times 12$, using equation (3) and (1) in algorithm 1 in squence to calculate the value of the output $q^{(i+1)}_{m,n}$, where new $r^{(i)}_{m,n}$ has been ready from equation (2) calculated by CNU and the $r^{(i-1)}_{m,n}$ of equation (1) has been kept at the end of shift registers. $q_n^{(i \times G + g)}$ of both $\mathbf{H}$ are needed for calculation in turns thus we also keep them in other registers.

3. Shift registers:

   At $cycle = 1$ to $cycle = 6 \times 12$, the end of the shift registers is $r_{m,n}^{(i-1)}$ which is the input of the last iteration and is used in the step 2. Push the new $r_{m,n}^{(i)}$ into the shift registers of size 12, and it will move each value by one register. The $r_{m,n}^{(i-1)}$ of next group needed for next cycle will be ready at the end of these registers after shifting.

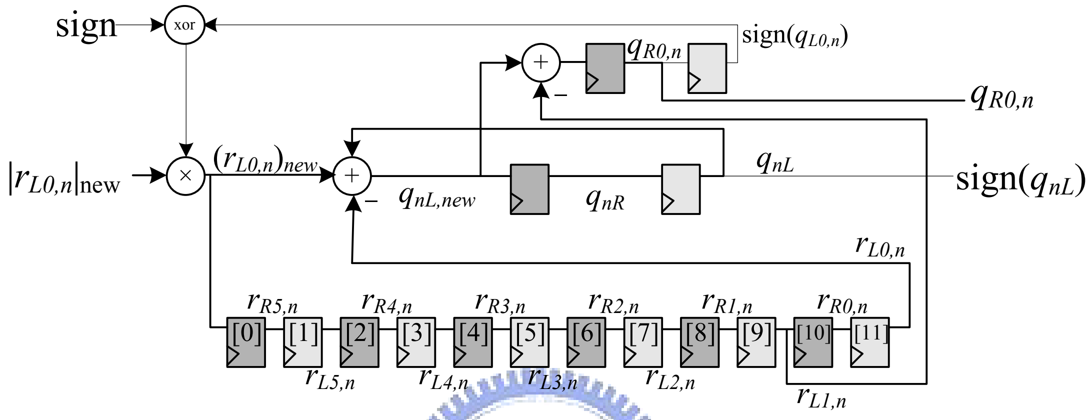


Figure 4.5: The architecture of variable node unit at cycle=12k+1.

The labels of the lines in Fig. 4.5 illustrate the value at $cycle = \{1, 13, ..., 12k + 1, ...\}$ as an example.

4.4 Comparison and Implementation Results

A (6,32)-regular (2048,1723) LDPC decoder for IEEE 802.3an is implemented. With input quantization of 6 bits (4-bit integer part and 2-bit fraction part), synthesized with RTL compiler using UMC CMOS 90nm cell library, the total gate number of the proposed architecture is 2.5M. 0.5M gates are for check node part and 2.0M are for variable node part. The synthesis result is listed in Table 4.1. Using the UMC CMOS 90nm technology with 9 metal layers, the layout plot is presented as Fig. 4.6 by SoC encounter for floorplanning, placement and routing. The core size is 4.0×4.0 mm^2 with 48% core utilization, and the critical path delay from the post-layout simulation is 10.0ns. With the clock frequency 100MHz and 6 iteration of layered decoding algorithm, the decoder results in

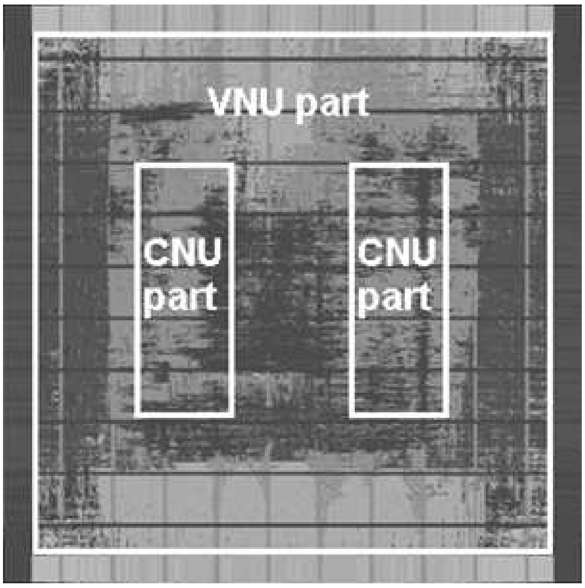


Figure 4.6: The layout of the (2048, 1723) LDPC decoder.

Table 4.1: The synthesis result of the proposed architecture

		check node part	variable node part
1 unit	critical path (ns)	4.7	4.2
	gate count (gates)	5.36K	904
Overall	critical path (ns)	6.2	6.2
	gate count (gates)	493K	2.02M
number of unit		64	2048
number of registers		$16 \times 64 = 1024$	$97 \times 2048 = 198.7\text{K}$

a total decoded data throughput of

$$2048\ \texttt{bits} \times 2 \times 100\ \texttt{MHz}/(12 \times 6 + 1) = 5.6\ \texttt{Gbps}.$$

The average energy per useful bit with a 0.9V supply is 0.211 nJ/bit at SNR=2.5dB. After implementation, the performance loss is about 0.1 dB at BER=10^{-6} compared with 20 iterations of standard BP algorithm as shown in Fig. 3.13. Table 4.2 lists the results of the layout, and the comparation of different architecture. Note that the average power is calculated at SNR equal to 2.5dB since the average power is similar at different SNR, as the early termination is not applied in this decoder.

Table 4.2: The comparison of different architecture

	[14]*	[14]**	[15]*	Proposed*
Process	0.13-μm CMOS	90-nm CMOS	0.18-μm CMOS	90-nm CMOS
Code length	(660,480)	(2048,1723)	(2048,1723)	(2048,1723)
Input quantization	4 bits	4 bits	5 bits	6 bits
Gate count (gates)	690K	2.23M	N/A	2.53M
Core area (mm^2)	7.3	N/A	43.9	16.0
Core utilization	72%	N/A	77%	48%
Clock freq. (Hz)	300M	250M	52M	100M
Performance*** (dB)	4.5(fixed)	N/A	4.7(floating)	4.2(fixed)
Power supply (V)	1.2	N/A	N/A	0.9
Average power (nJ/bit)	0.577 @SNR=4dB	N/A	N/A	0.211 @SNR=2.5dB
Throughput (bps)	3.3G	16.0G	7.1G	5.6G
Scheme	bit-serial fully parallel	bit-serial fully parallel	split-row (4) fully parallel	layered decoding partial parallel(6)

*Post-layout simulation result

**Synthesized result

***Performance indicates the SNR value at BER=10^{-6}

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this paper, we propose a decoding architecture of $(6, 32)$-regular $(2048, 1723)$ LDPC code for IEEE 802.3an. We use partial parallel scheme to reduce the area and congestion. Min-sum algorithm with normalization is applied to simplify the calculation. After analyzing the parity check matrix of this code, we find out that layered decoding algorithm is appropriate to apply with, and the architecture of variable node will be simple and regular according to the structure of this LDPC code. Thus we use shift registers instead of multiplexers for message storage in variable node process, which can reduce the hardware cost as well as routing congestion. Finally, the proposed decoding architecture is implemented with UMC CMOS 90nm process, and the proposed decoder can achieve the throughput of 5.6 Gbps according to the post-layout simulation. The core size is 4.0×4.0 $\mathtt{mm}^2$, clock frequency is 100MHz and average energy per useful bit with a 0.9V supply is 0.211nJ/bit.

Besides, we propose a method to simplify the conventional look-up table approximation. In this method, part of table look-up operations within one check node are replaced by taking mininum values, similar with min-sum algorithm. Although it can reduce the hardware cost and improve the critical timing path, the loss of error-correcting performance introduced by approximation cannot be ignored.

5.2 Future Work

Limited by the precision of look-up table, the proposed method cannot compete in error-correcting capability against the min-sum algorithm with scaling factor 0.75. If some other ways (e.g. look-up table composed of analog circuits) could break the limitation of precision, the performance of proposed method may be elevated to a superior level.

The total power is dominated by the shift registers, although the shift registers reduce the complexity. The power consumption may be too large for an integrated system, so it is suggested to find a better way to improve both issues. Furthermore, even if we find some way to reduce the congestion, the critical point of throughput is still the routing problem.

Bibliography

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[2] ——, *Low-density parity-check codes.* Cambridge, MA:MIT Press, 1963.

[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[4] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[5] *Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, IEEE Std. 802.16e, 2005.

[6] *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Std. 802.3an, 2006.

[7] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sep. 1981.

[8] J. L. Fan, *Constrained Coding and Soft Iterative Decoding.* Kluwer Academic Publisher, 2001.

[9] S. Lin and J. D. J. Costello, *Error Control Coding.* Prentice-Hall, 2004.

[10] N. Wiberg, *Codes and decoding on general graphs.* Ph.D. dissertation, Univ. Linkoping, 1996.

[11] X. Y. Hu, E. Eleftheriou, D. M. Arnold, and A. Dholakia, “Efficient implementation of the sum-product algorithm for decoding ldpc codes,” *IEEE GLOBECOM’01*, vol. 02, pp. 1036–1036E, Nov. 2001.

[12] D. E. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” *IEEE SiPS*, pp. 107–112, Oct. 2004.

[13] J. Zhang and M. P. C. Fossorier, “Shuffled iterative decoding,” *IEEE Trans. Commun.*, vol. 53, pp. 209–213, Feb. 2005.

[14] A. Darabiha, A. C. Carusone, and F. R. Kschischang, “A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS,” *IEEE CICC*, pp. 16–6–1–16–6–4, 2007.

[15] T. Mohsenin and B. M. Baas, “High-throughput LDPC decoders using a multiple split-row method,” *IEEE ICASSP*, vol. 2, pp. II–13–II–16, Apr. 2007.

作者簡歷

姓　　名：鄭佳瑋 Chia-Wei Cheng

出 生 地：台灣省新竹市

出生日期：1982. 09. 24

學　　歷：1989. 9~1995. 6 新竹市立北門國民小學

1995. 9~1998. 6 新竹市立光華國民中學

1998. 9~2001. 6 國立新竹女子高級中學

2001. 9~2005. 6 國立交通大學 電子工程學系 學士

2005. 9~2007.11 國立交通大學 電子研究所 系統組 碩士