國立交通大學

電子工程學系　電子研究所碩士班

碩 士 論 文

適用於快閃記憶體之(9216,8195)拉丁方陣低密度奇偶校驗碼解碼器

A (9216,8195) LDPC Decoder based on Latin Square for NAND Flash Memory

學生：曾士家

指導教授：張錫嘉 教授

中華民國一百年十二月

適用於快閃記憶體之(9216,8195)拉丁方陣

低密度奇偶校驗碼解碼器

A (9216,8195) LDPC Decoder based on Latin Square for NAND Flash Memory

研 究 生：曾士家　　Student：Shih-Jia Zeng

指導教授：張錫嘉 博士　Advisor：Hsie-Chia Chang

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 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

Dec 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年十二月

適用於快閃記憶體之(9216,8195)拉丁方陣低密度奇偶校驗碼解碼器

學生：曾士家　　指導教授：張錫嘉 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘 要

BCH碼因為硬體架構非常簡單而且只需要硬式輸入來解碼，目前是應用在快閃記憶體系統上錯誤更正碼的主流。雖然二位元軟式輸入被提出以加強錯誤更正能力。但二位元軟輸入對於BCH碼的錯誤更正能力並沒有很大的幫助。因此，本論文提出適用於快閃記憶體系統的低密度奇偶校驗碼（Low Density Parity Check, 簡稱LDPC Codes）及其解碼器架構，以二位元軟輸入之LDPC Codes提供在相同編碼率下比BCH碼更好的錯誤更正能力。

我們使用拉丁方陣演算法建構出編碼率為0.89的(9216,8195) LDPC Codes，並利用Area-Efficient Column Shuffle Decoding架構來降低硬體複雜度，解碼過程中從行的方向把奇偶校驗矩陣分割成36組，每一組再從列的方向分割為4個小組，這樣的架構能夠使檢查節點運算元被簡化為一個三對二的排序器。另外，我們利用加權平均數的概念來達到二位元軟輸入之最佳化，在信噪比(Signal to Noise Ratio) 5.0dB的情況下，我們所提出的LDPC Code位元錯誤率為10^{-9}，然而具有73個錯誤更正能力的BCH碼在此情況下的位元錯誤率為10^{-2}。使用UMC 90nm製程，所提出的解碼器邏輯閘數約為605.3k，在4次遞代解碼次數的情況下，可達到 1.58Gb/s 吞吐量。

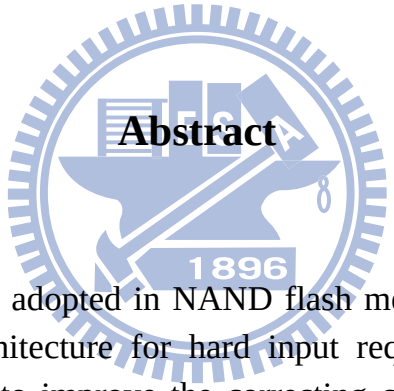
A (9216,8195) LDPC Decoder based on Latin Square for NAND Flash Memory

Student：Shih-Jia Zeng Advisor：Dr. Hsie-Chia Chang

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University



Abstract

BCH code is mainly adopted in NAND flash memory system because of its simple hardware architecture for hard input requirement. Although soft input can be considered to improve the correcting capability, BCH code has little improvement when soft input is provided. In this thesis, a 2-bit soft input LDPC decoder is presented to outperform BCH code under same code rate.

The (9216, 8195) LDPC code with code rate 0.89 is constructed from Latin square algorithm. An Area-Efficient Column Shuffled decoding architecture is proposed to reduce hardware complexity. Columns in parity-check matrix are divided into 36 groups, and all the rows of each column group are divided into 4 subgroups. Following this architecture, a check node update unit can be simplified as a 3-to-2 sorter. In addition, the concept of weighted mean is applied to optimize 2-bit soft input quantization. At signal to noise ratio (SNR) of 5.0dB, bit error rate (BER) of our proposed LDPC code is 10^{-9} whereas BCH code that can correct 73 errors is 10^{-2}. Using 90nm CMOS technology, our design with 605.3k equivalent gates can achieve the maximum throughput 1.58 Gb/s under 4 decoding iterations.

誌　謝

研究所生活這兩年，受到很多人的幫忙與照顧。首先我要感謝父母家人對我的付出與支持，讓我能夠無憂無慮的順利完成學業，我真的很愛你們。我也要感謝我的指導教授張錫嘉老師，感謝老師在研究上的指導，也感謝老師在生活上的關心，能當你的研究生真的是一件十分幸福的事。

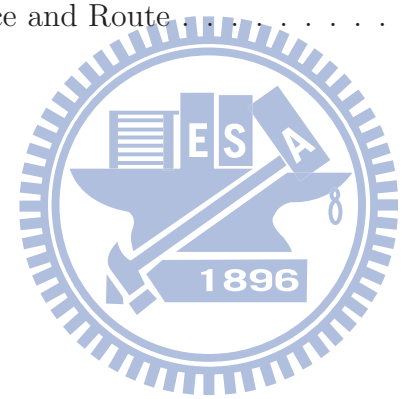
再來要感謝 Ocean 與 Oasis 的夥伴們，大家都很好相處而且熱心助人，這兩年真的從大家身上學習到很多東西。特別要感謝何堅柱學長及陳志龍學長，真的非常謝謝學長在研究上耐心並且不厭其煩的指導。

最後要感謝體貼的女友庭安，當我壓力大的時候、身體不舒服的時候總是在我身邊照顧我，真的很感謝有妳的陪伴。

衷心的祝福大家心想事成，未來都過著幸福快樂的生活

Contents

List of Figures



List of Tables

Chapter 1

Introduction

1.1 Motivation

Modern NAND Flash momory system adopts error correction codes to improve device reliability [1] [2]. BCH code [3] [4] is mainly used in single level cell (SLC) NAND flash memory system because of its simple hardware architecture and hard input requirement. The area occupation of Multi-level cell (MLC) is only half compare with SLC. However, MLC also leads to degradation of reliability. More powerful error corrections codes for next generation NAND flash memory system is needed.

Soft-input is provided to improve the correcting capability of error correction code. However, BCH code has only little improvement when soft input is provided [5] [6]. LDPC code is a good candidate for its powerful correcting capability. 2-bit soft LDPC code can outperform BCH code with same code rate.

Low Density Parity Check (LDPC) codes were first discovered by Gallager in 1962 [7] and were rediscovered and generalized by MacKay in 1999 [8]. Well designed LDPC codes decoded with iterative decoding using belief propagation (BP) algorithm, achieve performance close to the Shannon limit [9]. Consequently, LDPC codes were widely adopted for error control in many communication and digital storage systems.

High code rate is a necessary condition for error correction code applied on NAND flash memory system. A high code rate LDPC code introduces high row degree. This makes implementation difficult due to the large number of inputs to sorter and the increased routing complexity. Column shuffled decoding [10] is a good solution to this problem.

Variable nodes are divided into 36 groups. Only 1st, 2nd min are stored to reduce the storage cost. With row divided into 4 groups, VNU can be simplified to a 2-input adder and a 2-input subtractor. Shifting networks are applied between CNUs and memories. The maximum throughput can achieve 1.581 Gbps with 4 iterations, using 90nm CMOS technology. The proposed LDPC code decoder has a better performance than BCH code with the same code rate when 2-bit soft input is provided.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter II gives the introduction of NAND flash memory. Chapter III introduces the column shuffled decoding algorithm and the code construction. In Chapter IV, decoder architecture is detailed explained. The simulation result is given in Chapter V and conclusion in Chapter VI.

Chapter 2

NAND Flash Memory

2.1 Introduction of NAND Flash Memory

2.1.1 Flash Memory System

Flash memory was invented by Dr. Fujio Masuoka of Toshiba Corp. in 1984. NAND Flash is employed for data storage in a variety of portable and mobile applications. Since flash memory is non-volatile, no power is needed to maintain the information stored. Flash memory cell is based on the Floating Gate (FG) illustrated in Fig. 2.1. The isolated gate constitutes an excellent 'trap' for electrons. The operations performed to inject and remove electrons from the isolated gate are called program and erase. More details of these operations will be presented in next section.

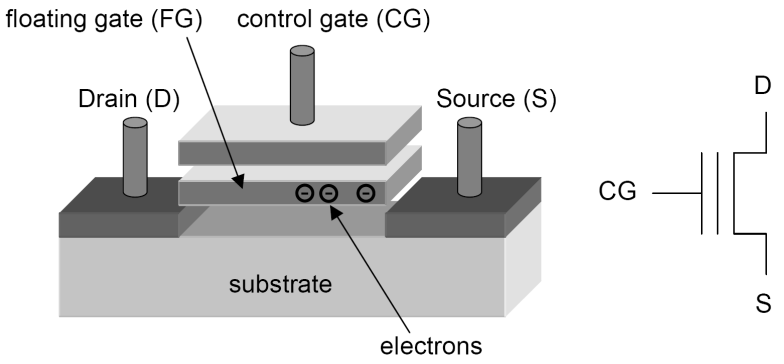


Figure 2.1: Floating gate memory cell its schematic symbol [1]

The memory cells are packed to form a matrix in order to optimize silicon area occupation. In the NAND string, the cells are connected in series, in groups of 32 or 64,

as shown in Fig. 2.2. Two selection transistors are placed at the edges of the string, to ensure the connections to the source line and to the bit line. Each NAND string shares the bit line contact with another string. Control gates are connected through word lines.

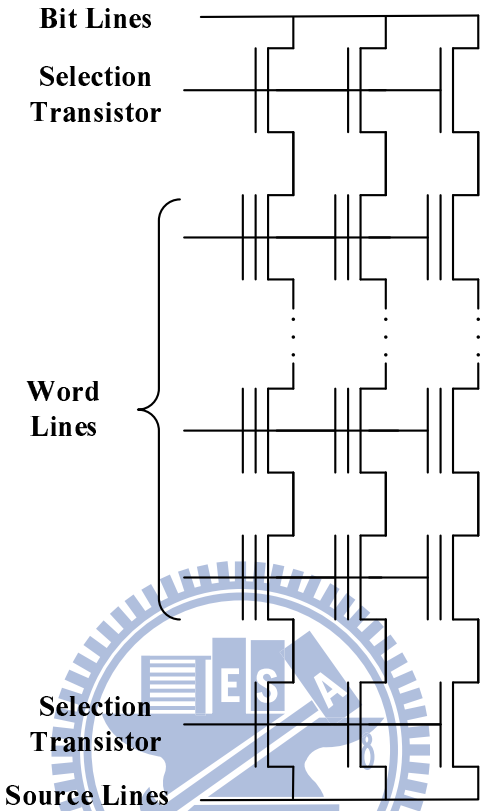


Figure 2.2: NAND string

A NAND memory is divided in pages and blocks. A block is the smallest erasable unit. Each block contains multiple pages. The number of pages within a block is typically a multiple of 16. A page is the smallest addressable unit for reading and writing. Each page is composed of main area and spare area. Main area can range from 4 to 8 kB or even 16 kB. Spare area can be used for ECC.

2.1.2 NAND Flash Cell Program

Programming of NAND memories exploits the quantum-effect of electron tunneling in the presence of a strong electric field. In order to trigger the injection of electrons into the floating gate, the following voltages are applied, as shown in Fig. 2.3. $V_{PGM}(20-25V)$ is applied on the selected gate to be programmed, and $V_{PASS,P}(8-10V)$ on the unselected gates. V_{DD} on the gate of the drain selector, and GND on the gate of the source selector. GND on the bit line to be programmed, and V_{DD} on other bit lines. When the bit lines are driven to V_{DD}, drain transistors are diode-connected and the corresponding bit lines are floating. $V_{PASS,P}$ is applied to the unselected word lines to inhibit the tunneling phenomena.

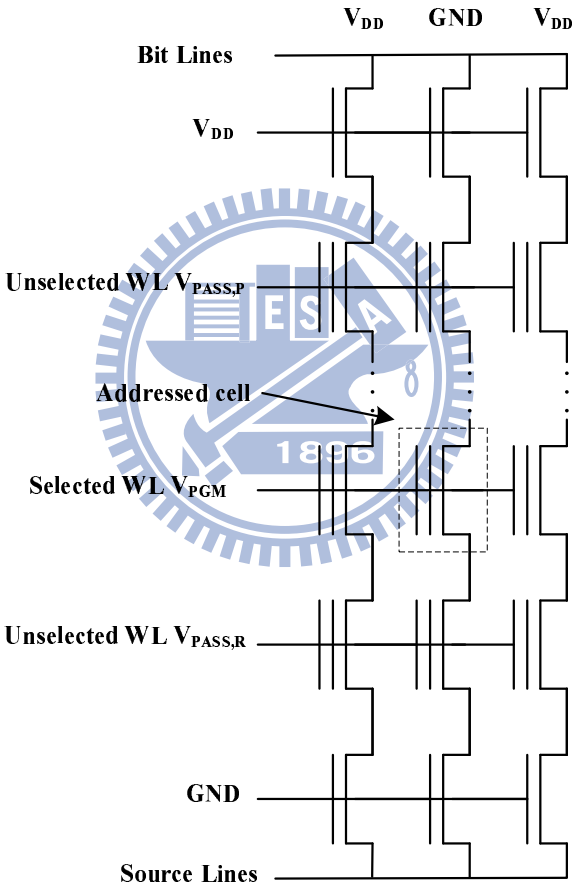


Figure 2.3: Program operation in a NAND string

2.1.3 NAND Flash Cell Erase

Erasing of NAND memories is the inverse process of programming. When NAND flash Cell is earsed, 0V is applied to the Source, Drain and Gate. And high voltage V is applied to the Substrate. Electrons in Floating Gate are attracted to the Substrate and no more electrons are left in Floating Gate. Fig. 2.4 is a simple illustration for this operation.

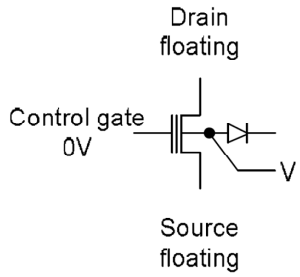
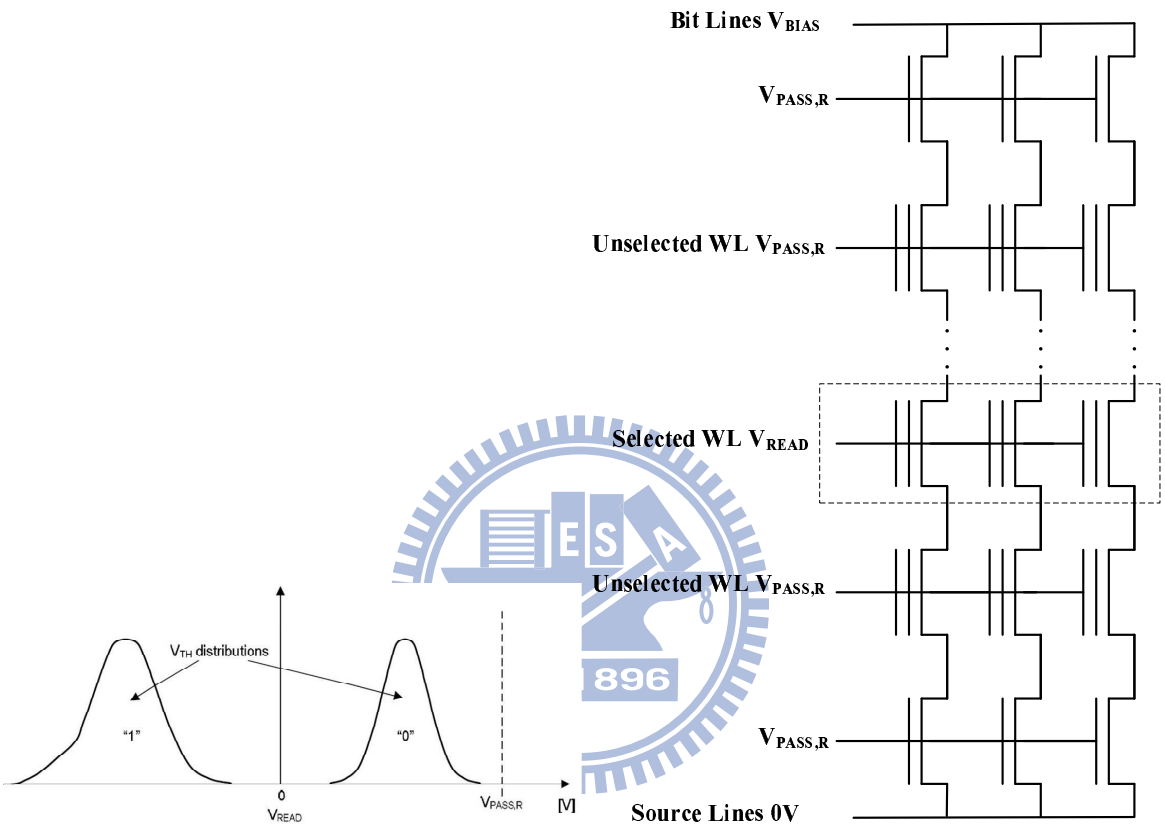


Figure 2.4: Erase operation in a NAND cell

2.1.4 NAND Flash Cell Read

A Single Level Cell (SLC) means that only 1 bit data is stored per cell. Therefore, the threshold voltage region of a SLC is divided into two levels. Fig. 2.5(a) shows the threshold voltage distribution of SLC and we will use Fig. 2.5(a) to explain read operation.

When we read a cell in Fig. 2.5(a), its gate is driven at $V_{READ}(0V)$, while the other cells are biased at $V_{PASS,R}(4-5V)$, so that they can act as pass-transistors. In fact, an erased SLC has a V_{TH} smaller than 0 V; vice versa, a written SLC has a positive V_{TH} smaller than 4 V. In this example, biasing the gate of the selected cell with a voltage equal to 0 V, the series of all the cells will conduct current if the addressed cell is erased.



(a) Threshold voltage distribution of a Signle Level Cell

(b) NAND string biasing during read

Figure 2.5: Read operation in a NAND string

2.2 Reliability of NAND Flash Memory

2.2.1 Program Disturb

Program operation in a NAND string described in 2.1.2 will cause disturb in other unselected cells. We use Fig. 2.6 to explain program disturb and pass disturb.

Cell A is the cell to be programmed. Cell B will suffer from the program disturb. The effective programming voltage for cells B is $V_{PGM} - V_{ch}$. V_{ch} is the equivalent potential in the channel. To lower the effective programming voltage, a high $V_{PASS,P}$ is applied in other cells. Pass disturb occurs in the cell C. It's effective programming voltage is $V_{PASS,P}$. Therefore, the program disturb can be reduced by increasing $V_{PASS,P}$ at the expense of an increased pass disturb.

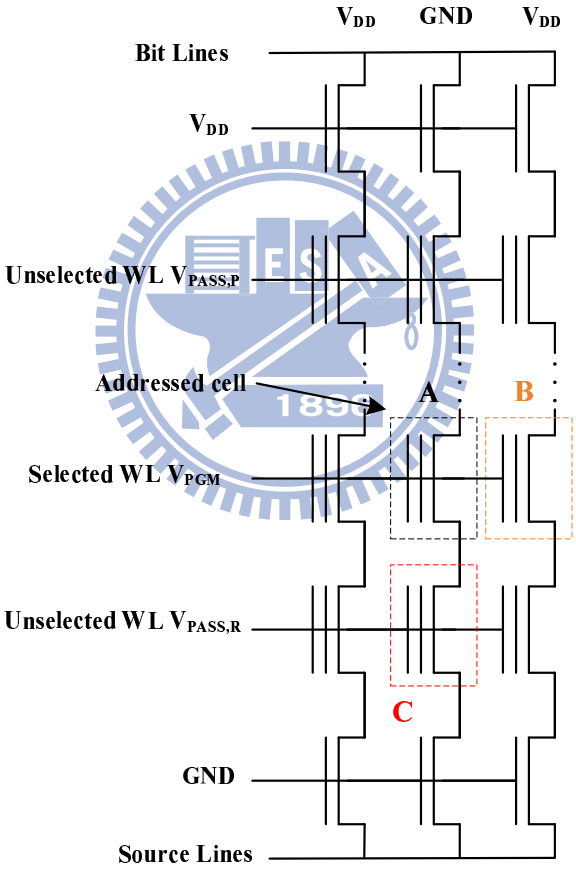


Figure 2.6: Program disturb in a NAND string

2.2.2 Read Disturb

Read disturbs are the most frequent source of disturbs in NAND architectures. This kind of disturb may occur when reading many times the same cell without any erase operation. Unselected cells in Fig. 2.7 will suffer from read disturb due to the $V_G = 4.5V$ applied in unselected cells.

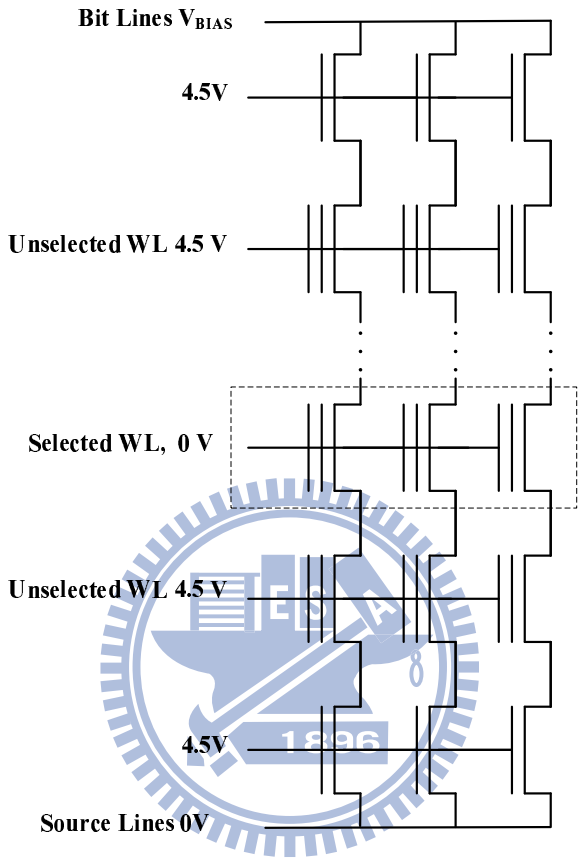


Figure 2.7: Read disturb in a NAND string

2.2.3 NAND Flash Multi-level Cell

Fig. 2.8 shows a 2bits/cell NAND flash cell. The obvious advantage of a 2 bit/cell implementation (MLC) with respect to a 1 bit/cell device (SLC) is that the area occupation of the matrix is half as much. On the other hand, the area of the periphery circuits increases. Threshold voltage region is divided into 4 levels and region for each level is narrower. Therefore, the probability of threshold voltage shifting to other level is increased and led to degradation of reliability.

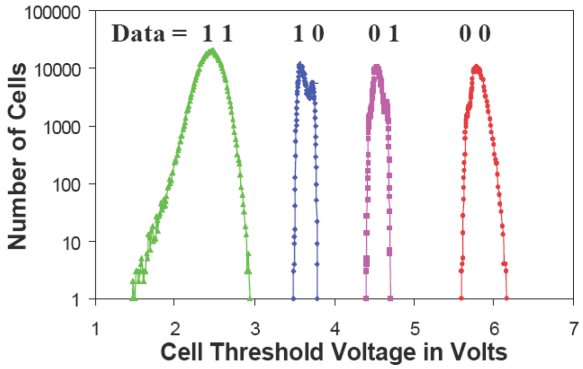


Figure 2.8: Threshold voltage distribution of a 2bits/cell NAND flash cell

Advanced technology scale down and more bits of data stored per NAND flash cell will cause the degradation of reliability. More parity bits are required to improve the correcting capability of BCH code. The increase of spare area (area for parity bits storage) greatly degrades the data storage capacity and is infeasible to commerical product. To overcome this problem, NAND flash memory system will provide more information (soft input) in the next generation standard and much powerful error correcting code can be adopted. BCH code is feasible for its simple hardware architecture and only hard input requirement. However, BCH code has only little improvement when soft input is provided. LDPC code is probability-based and soft information can be well-used. Therefore, LDPC code is a good candidate for the next generation NAND flash memory system. Providing soft input will inrease reading latency in flash memory system. This is a trade-off between correcting capability and system latency. This thesis shows that only 2-bits soft input LDPC code can outperform BCH code under same code rate.

Chapter 3

Construction of Low Density Parity Check Codes

Low Density Parity Check (LDPC) codes were first discovered by Gallager in 1962 [7] and were rediscovered and generalized by MacKay in 1999 [8]. Based on the methods of construction, LDPC codes can be classified into random-like codes and structured codes [11]. Well designed LDPC codes decoded with iterative decoding using belief propagation (BP) algorithm, achieve performance close to the Shannon limit. Consequently, LDPC codes were widely adopted for error control in many communication and digital storage systems.

In this chapter, structured code construction methods will be introduced. Code parameters related to performance and implementation complexity will be discussed.

3.1 Code Construction

3.1.1 General Construction of QC-LDPC Codes

We start code construction from a base matrix W with size $d_v \times d_c$. d_v represents column degree and d_c represents row degree. $w_{i,j}$ means the element located in i-th row and j-th column in W. $w_{i,j}$ could be a numeral value or an element in finite field. The algebra to determine $w_{i,j}$ is diverse and make constructed QC-LDPC codes have different performance and characteristic.

$$W = \begin{pmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,d_c-1} \\ w_{1,0} & w_{2,1} & \cdots & w_{1,d_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d_v-1,0} & w_{d_v-1,1} & \cdots & w_{d_v-1,d_c-1} \end{pmatrix}$$

Let P be a circulant permutation matrix(CPM) with size p. It's top row is given by the p-tuple $(0\ 1\ 0\ 0\ \cdots\ 0)$. P consists of p-tuple first row and its $p-1$ right cyclic shifts as other rows. P^i, the product of P with itself i times, is also a CPM whose top row has a single 1-component at the position i. Fig. 3.1 is a demonstration for CPMs with size 4.

$$\underset{P^0 = P^4}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \quad \underset{P^1}{\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}} \quad \underset{P^2}{\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}} \quad \underset{P^3}{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}$$

Figure 3.1: Circulant Permutation Matrices with size 4

Replacing elements in the base matrix W with CPMs will derive the parity-check matrix H. The correspondence between elements in the base matrix W and CPMs also diverse. We will introduce two kinds of algorithm to construct QC-LDPC codes.

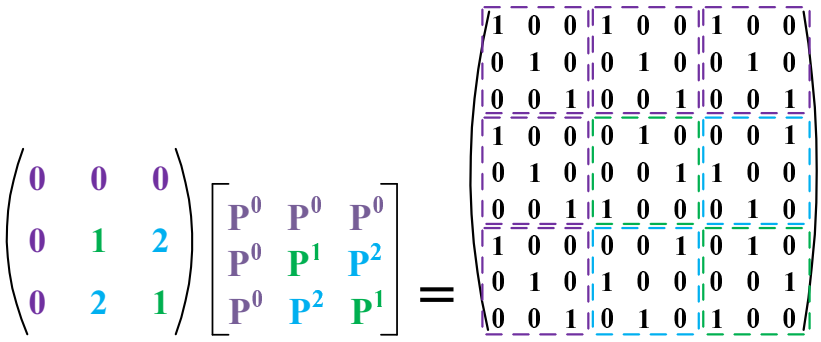
3.1.2 Product QC-LDPC codes

The base matrix W of product QC-LDPC codes [11] is constructed in a prime field. Assume a prime number p is chosen, $w_{i,j}$ will be $(i \times j \mod p)$ for $0 \leq i, j < p$. Maximum size of the base matrix W will be $p \times p$.

$$W = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & j-1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & i-1 & \cdots & (i-1) \times (j-1) \end{pmatrix}$$

After column degree d_v and row degree d_c is determined, we can select a sub-matrix with size $d_v \times d_c$ from the base matrix W. Denoting the sub-matrix as W_{sub}, $w_{sub\ i,j}$

represents the element located in i-th row and j-th column in W_{sub}. The CPM size of product QC-LDPC codes is p. Let P be a CPM with size p, elements in selected sub-matrix will be replaced by $P^{w_{sub\ i,j}}$ for $0 \leq i < d_v, 0 \leq j < d_c$. Fig. 3.2 illustrates a base matrix and its correspondent parity-check matrix.



(a) Base matrix W for $p = 3$ (b) Correspondent parity-check matrix H

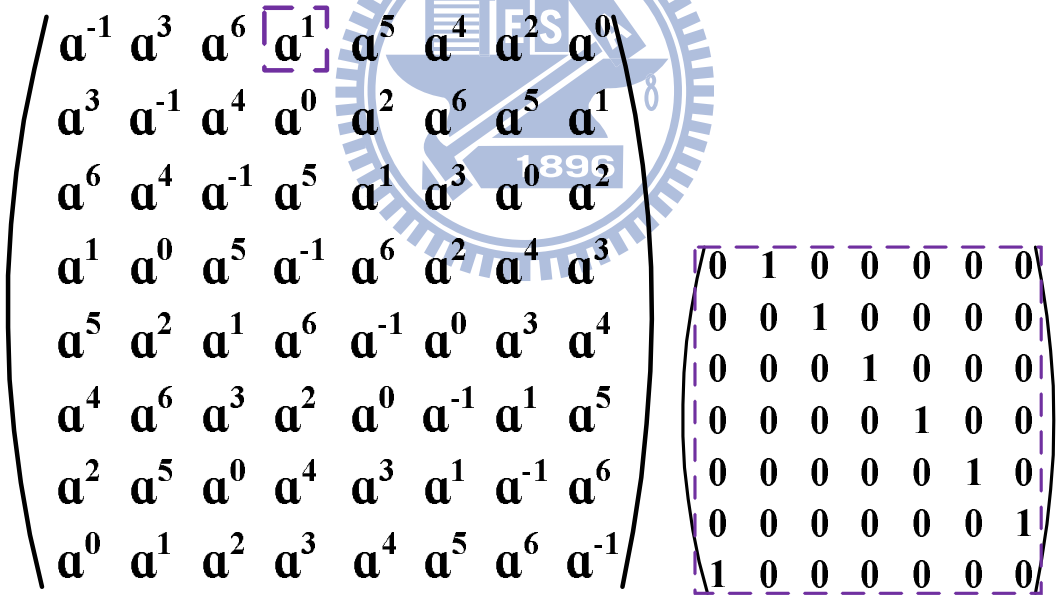
Figure 3.2: Illustration of Product QC-LDPC codes

3.1.3 Latin Square QC-LDPC codes

The base matrix W of Latin square QC-LDPC codes [12] is constructed in a galois field $GF(2^m)$. Maximum size of the base matrix W is $2^m \times 2^m$ and size of the CPM is $(2^m - 1) \times (2^m - 1)$. $w_{i,j}$ is $(\alpha^i \eta - \alpha^j)$ for $0 \leq i, j < 2^m$,$\alpha^0 = 1$, $\alpha^{-\infty} = 1$.

$$W = \begin{pmatrix} \alpha^0\eta - \alpha^0 & \alpha^0\eta - \alpha^1 & \cdots & \alpha^0\eta - \alpha^{-\infty} \\ \alpha^1\eta - \alpha^0 & \alpha^1\eta - \alpha^1 & \cdots & \alpha^1\eta - \alpha^{-\infty} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{-\infty}\eta - \alpha^0 & \alpha^{-\infty}\eta - \alpha^1 & \cdots & \alpha^{-\infty}\eta - \alpha^{-\infty} \end{pmatrix}$$

η is an element in $GF(2^m)$. Choosing different η only permutes the rows in W. We can also select a sub-matrix with size $d_v \times d_c$ from the base matrix W. However, the sub-matrix W_{sub} should be chosen carefully without element α^{-1}. $w_{sub\ i,j}$ represents the element located in i-th row and j-th column in W_{sub}. Assume $w_{sub\ i,j}$ is α^k, elements in W_{sub} will be replaced by P^k. α^{-1} may exist in W_{sub}, but the CPM P^{-1} is not defined. Fig. 3.3 illustrates a base matrix and a CPM of Latin Square QC-LDPC codes.



(a) Base matrix W for $m = 3, \eta = \alpha^0$

(b) CPM with size $(2^m - 1) \times (2^m - 1)$

Figure 3.3: Illustration of Latin Square QC-LDPC codes

3.1.4 Comparison between Product and Latin Square QC-LDPC codes

In 3.1.2 and 3.1.3, the algebra of product and Latin square QC-LDPC codes were introduced. Comparison between product and Latin square QC-LDPC codes were showed in table 3.1.

Table 3.1: Comparison between Product and Latin Square QC-LDPC codes

	Product	Latin Square
$w_{i,j}$	$(i \times j \mod p)$	$(\alpha^i \eta - \alpha^j)$
size of the CPM	prime number p	$(2^m - 1)$
dependent rows in H	less	more
performance	good	excellent

The algebra of product QC-LDPC codes generates the base matrix W with the same column offsets in each row. Fig. 3.4 shows that the offsets between i-th column and $i-1$-th column are the same in each row. Regular offsets in the base matrix can reduce the complexity of the shifter in the decoder. Besides, product QC-LDPC codes is constructed in a prime field, but Latin square QC-LDPC codes should be constructed in a galois field $GF(2^m)$. Product QC-LDPC codes is more flexible than Latin square QC-LDPC codes.

					offset
0	0	0	0	0	0
0	1	2	3	4	1
0	2	4	6	8	2
0	3	6	9	12	3
0	4	8	12	16	4

Figure 3.4: Base matrix of Product QC-LDPC codes without mod operation

Dependent rows in parity-check matrix H will affect the code rate. With the same d_v, d_c, and CPM size, code rate of Latin square QC-LDPC codes is higher than product QC-

LDPC codes. Let $d_v = 4$, $d_c = 36$, CPM size = 127, a $(4572, 4067)$ product QC-LDPC code with code rate 0.8895 is constructed, and a $(4572, 4081)$ Latin square QC-LDPC code with code rate 0.8926 is constructed. Code rate is also a important requirement for NAND flash memory.

Performance comparison between product and Latin square QC-LDPC codes is showed in Fig. 3.5. They have the same (p,d_v, d_c) and approximately the same (N,K). At SNR 4.3, the BER of Latin square LDPC codes is 2.2×10^{-7} whereas the BER of product LDPC code is about 1.5×10^{-6}.

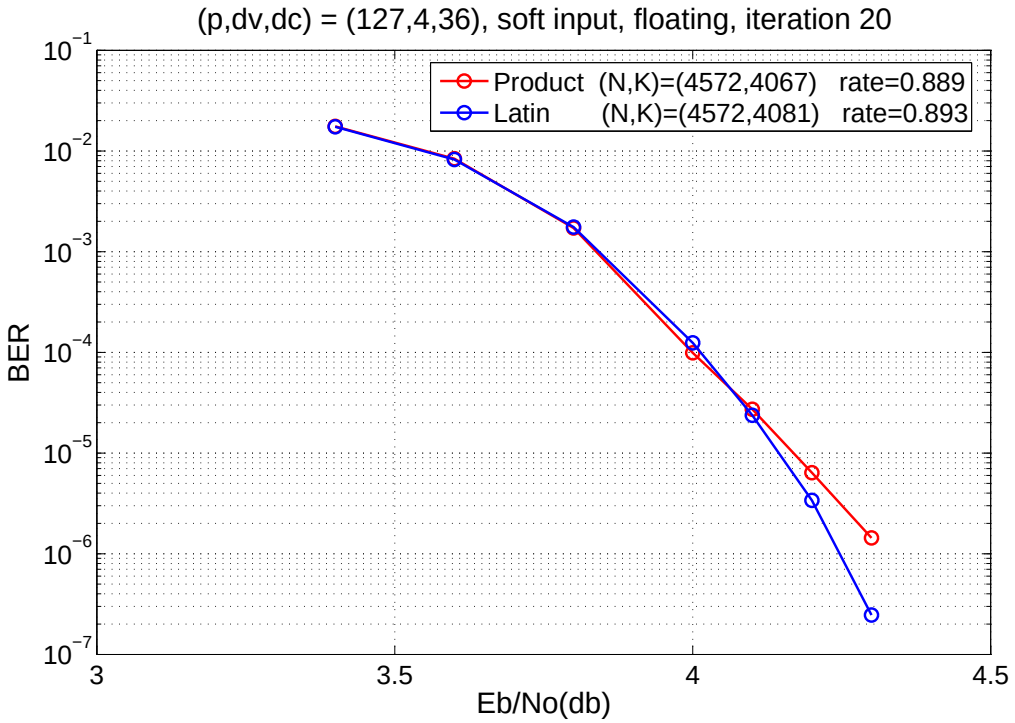


Figure 3.5: Performance comparison between Product and Latin Square QC-LDPC codes

3.1.5 Parameters in Code Construction

For a (N, K) QC-LDPC code, N is the codeword length and K is the information length. Denote M as the numbers of check equations in H. Consider a base matrix W with size $d_v \times d_c$ and CPMs with size $p \times p$, equation (3.1) shows the relationship between (N, K, M) and (p, d_v, d_c). Code rate is mainly decided by d_v and d_c (3.2).

$$\begin{aligned} N &= d_c \times p \\ M &= d_v \times p \\ K &= N - (M - numbers\ of\ dependent\ rows\ in\ H) \end{aligned} \tag{3.1}$$

$$\frac{K}{N} = \frac{(d_c - d_v)}{d_c} + \frac{(numbers\ of\ dependent\ rows\ in\ H)}{(d_c \times p)} \tag{3.2}$$

Given N around 9200 and code rate around 0.9, we take product QC-LDPC codes as an example. First, decide column degree d_v and use equation (3.2) to calculate the d_c that meets the code rate requirement. Once d_c is determined, use equation (3.1) to find possible p. Table 3.2 lists some possible codes that meet the requirements.

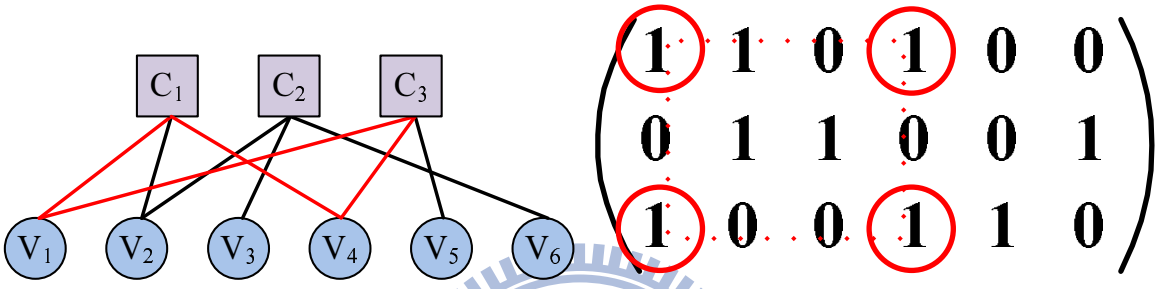
Table 3.2: Codes from Product QC-LDPC codes

d_v	d_c	p	N	K
3	30	307	9210	8291
4	40	229	9160	8247
6	60	151	9060	8159
8	81	113	9153	8256

3.2 Performance-Related Parameters

3.2.1 Cycles in Tanner Graph

A cycle in a graph of vertices and edges is defined as a sequence of connected edges which starts from a vertex and ends at the same vertex, and satisfies the condition that no vertex (except the initial and the final vertex) appears more than once. The number of edges on a cycle is called the length of the cycle. Fig. 3.6 illustrates a Tanner Graph with cycle-4 cycles and its corresponding parity check matrix. The length of the shortest cycle in a graph is called the girth of the graph.



(a) A tanner graph with cycle-4 (b) Correspondent parity-check matrix H

Figure 3.6: An example of a tanner graph with cycle-4

While decoding a LDPC code with BP algorithm, these short cycles, especially cycles of length 4, make some variable nodes highly correlated and hence severely limit the decoding performance. Therefore, it is important to design codes without short cycles in their Tanner graphs, especially cycles of length 4. Because the parity-check matrix H is constructed from the base matrix W with CPMs, we can use base matrix W instead of parity-check matrix H to compute cycles in LDPC codes.

Fig. 3.7 illustrates cycle-4 produced from base matrix W. Note that the $1st$ row in W and the $2nd$ row in W produce the same check equations labeled with the same color in H. The value of the $2nd$ row in W is just the value of the $1st$ row in W added by 1. Due to the characteristic of CPM, adding a fixed value in a row in W will not change the check equations produced by the row.

We use Fig. 3.8 to prove that if the difference between the cyclic shift amount in one sub-matrix row is equal to the difference between the cyclic shift amount in other sub-matrix row, cycle-4 is formed. $w_{m,s}$ represents the shift value in m-th row, s-th column.

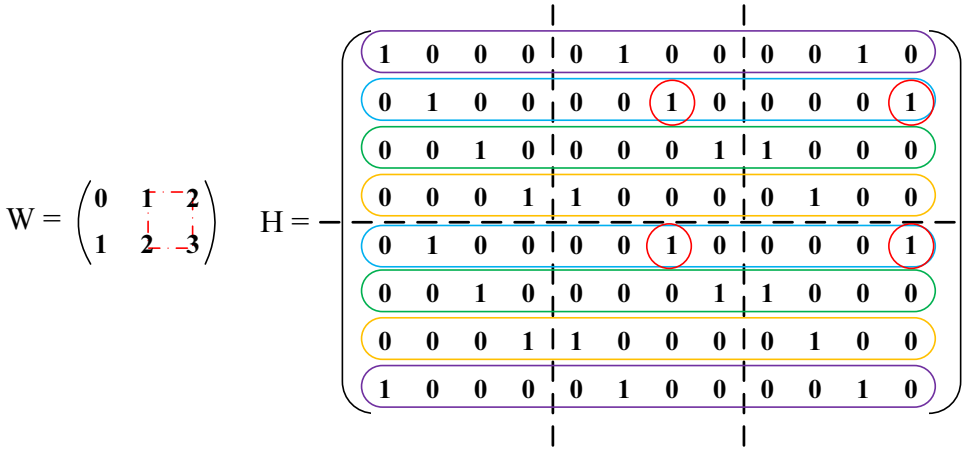


Figure 3.7: Demonstration of cycle-4 in base matrix W and parity-check matrix H.

Subtract the value in m-th row by A and subtract the value in n-th row by C. Now, $w_{m,s} = w_{n,s} = 0$, $w_{m,t} = (B - A)$, $w_{n,t} = (D - C)$. If the equation (3.3) is satisfied, cycle-4 exist in correspondent sub-blocks produced by these 4 shift value. Equation (3.3) can be rewrite as equation (3.4).

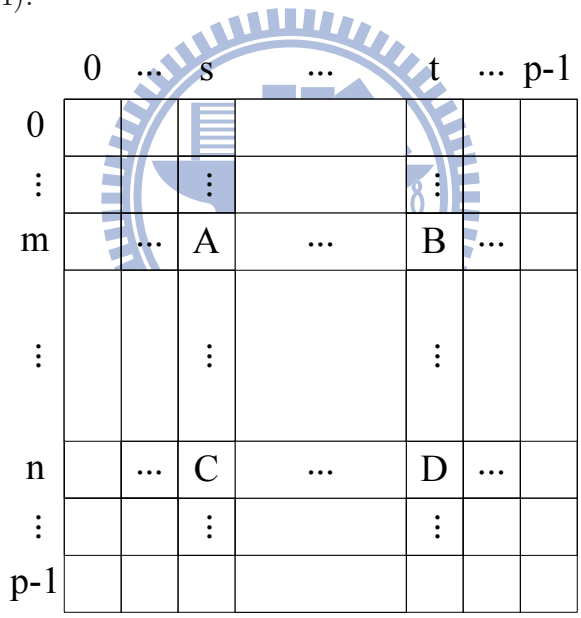


Figure 3.8: A base matrix W with $p \times p$

$$(B - A) = (D - C) \mod p \tag{3.3}$$

$$(B - A + C - D) \mod p = 0 \tag{3.4}$$

For product QC-LDPC codes, the shift value of $w_{i,j}$ is $i \times j \mod p$. Substitute this equation $w_{i,j}$ into equation (3.4) we can derive equation (3.6).

$$\begin{aligned}(B-A) &= m \times (t-s) \mod p \\ (C-D) &= n \times (s-t) \mod p\end{aligned} \tag{3.5}$$

$$(B-A+C-D) = (m-n) \times (t-s) \mod p = 0 \tag{3.6}$$

Since p is a prime number and $0 < m < n < (p-1),\ 0 < s < t < (p-1)$, equation (3.6) will not be satisfied for product QC-LDPC codes.

For Latin square QC-LDPC codes, $w_{i,j}$ is defined as $(\alpha^i \eta - \alpha^j)$. Assume $w_{i,j} = \alpha^k$, the shift value is k. We can not directly substitute the equation $w_{i,j}$ into equation (3.4), because the numeral value of k is decided by i, j and η. It's trivial that if $A = C$ and $B = D$, cycle-4 exists in the QC-LDPC code. Add the shift value in m-th row by c and add the value in n-th row by l. Equation (3.7) is the new value of $w_{m,s}, w_{m,t}, w_{n,s}$ and $w_{n,t}$. Equation (3.8) is the condition for $w_{m,s} = w_{n,s},\ w_{m,t} = w_{n,t}$.

$$\begin{aligned}w_{m,s} &= \alpha^c(\alpha^m \eta - \alpha^s),\ w_{m,t} = \alpha^c(\alpha^m \eta - \alpha^t) \\ w_{n,s} &= \alpha^l(\alpha^n \eta - \alpha^s),\ w_{n,t} = \alpha^l(\alpha^n \eta - \alpha^t)\end{aligned} \tag{3.7}$$

$$(\alpha^m - \alpha^n) \times (\alpha^t - \alpha^s) = 0 \tag{3.8}$$

Since $0 < m < n < (p-1),\ 0 < s < t < (p-1)$, equation (3.8) will not be satisfied for Latin square QC-LDPC codes.

3.2.2 Column Degree

Column degree d_v is defined as the numbers of check nodes connected to a variable node. From equation (4.3), a variable node with higher d_v receives more message from different check nodes. For LDPC code, we call the performance degradation in water fall region, the error floor. A LDPC code with higher column degree has better performance in water fall region. It means that it can suppress the error floor in lower bit error rate region. Fig. 3.9 shows the performance of LDPC codes with different column degree. s represents scaling factor in this thesis.

A QC-LDPC codes with high column degree has good performance in water fall region. In low SNR region, messages passed between check nodes and variable nodes will suffer

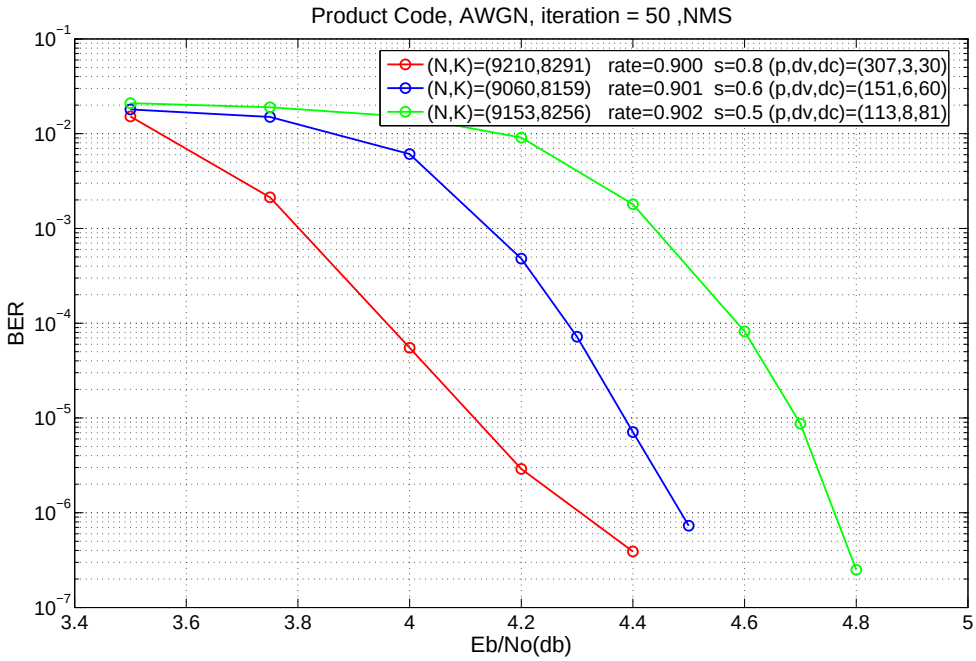


Figure 3.9: Performance of LDPC code with different column degree.

from more disturbance. In Fig. 3.9, (9210,8291) is a product QC-LDPC code, with column degree 3. It has poor performance at waterfall region due to its low column degree. LDPC code with column degree 6 and 8 has better performance at waterfall region. But, we can still find that the bit error rate (BER) difference between SNR 4.8 and SNR 4.6 in LDPC code with column 8 is large than the BER difference between SNR 4.5 and SNR 4.3 in LDPC code with column 6.

3.3 Proposed (9216,8195) QC-LDPC code

The requirement for codes applied in NAND flash memory includes information length $K >= 8192$, code rate > 0.9, and no performance degradation down to bit error rate near 10^{-12}. For good performance, we use Latin square algebra to construct the LDPC code. Although a LDPC code with higher column degree has better performance in water fall region, it also implies more hardware cost in variable node update units (VNUs). The selection of the column degree is a trade off between code performance and hardware cost. At first, a (9180,8179) Latin square QC-LDPC code with $(d_v, d_c, p) = (4, 36, 255)$ is proposed. However, it's information length is less than 8192. In order to increase the

information length, we enlarge the size of CPM to 256 according to the equation (3.2). Hence, the constructed (9216,8195) code is not a traditional Latin square QC-LDPC code. The equation (3.6) helps us to find the code without cycle-4.

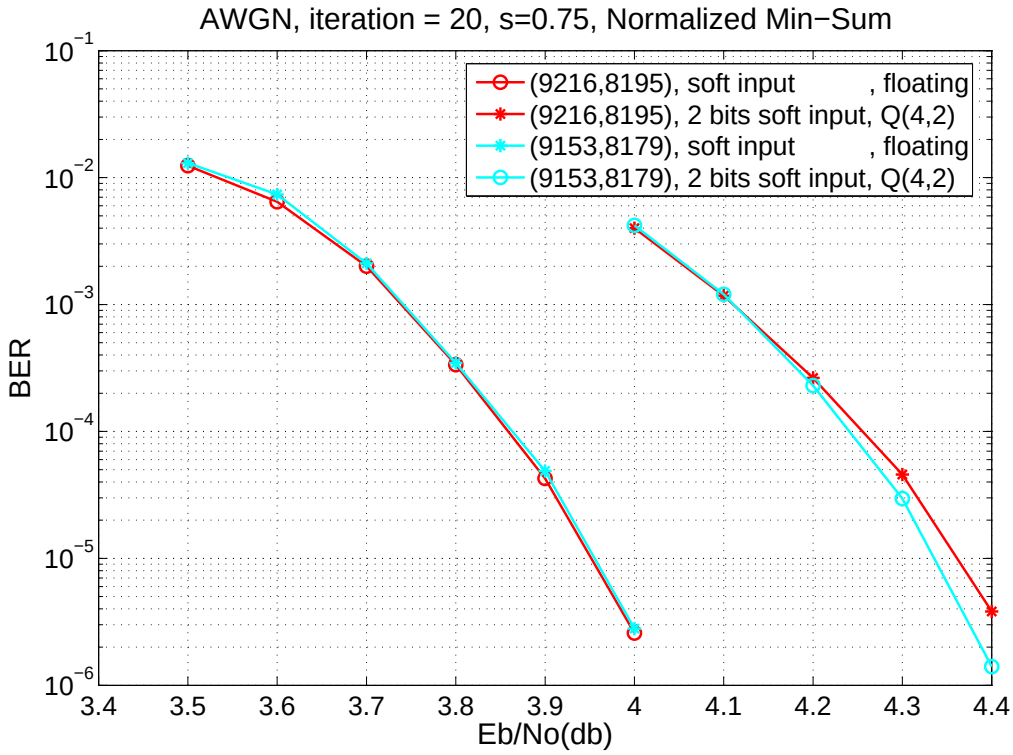


Figure 3.10: Performance of Proposed (9216,8195) QC-LDPC codes

Fig. 3.10 shows the performance of (9216,8195) and (9180,8179) LDPC codes. The lines with o (circle) are the performance with soft-input and no quantization in the decoder. The lines with $*$ (star) are the performance with 2 bits soft-input and 4 bits quantization in the decoder. Their performance are very close. Hence, the decoder architecture is designed for the (9216,8195) Latin square QC-LDPC code.

Chapter 4

LDPC Decoder Architecture

4.1 Decoding Algorithm

4.1.1 Standard BP Algorithm

The log-likelihood ratio (LLR) of intrinsic information of n-th variable node is denoted by P_n. The message from n-th variable node to m-th check node is denoted by z_{mn}. The message from m-th check node to n-th variable node is denoted by ε_{mn}. The a posteriori LLR of n-th bit is denoted by z_n. The standard BP is carried out as followed.

1. **Initialization:** Set $i = 1$,maximum number of iterations to I_{Max}. For each m, n, set $z_{mn}^{(0)} = P_n$,

2. **Iterative Decoding:**

   (a) Check node to variable node update step, for $1 \leq n \leq N$ and each $m \in M(n)$, process

   $$\epsilon_{mn}^{i} = 2\tanh^{-1}\Big(\prod_{n' \in N(m)\backslash n}^{d} \tanh\big(\frac{z_{mn'}^{i-1}}{2}\big)\Big) \tag{4.1}$$

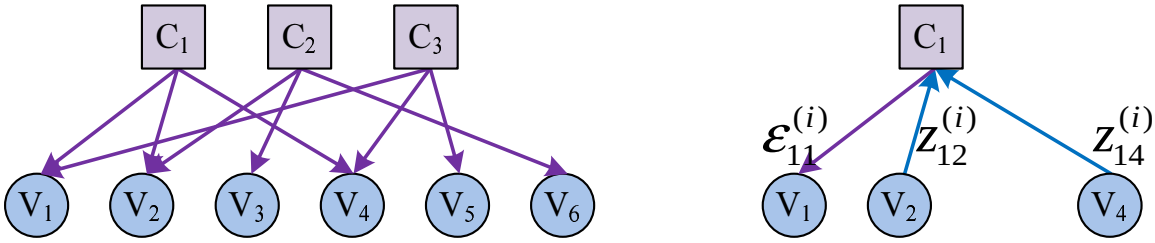
   (b) variable node to check node update step, for $1 \leq n \leq N$ and each $m \in M(n)$, process

   $$z_{mn}^{(i)} = P_n + \sum_{m' \in M(n)\backslash m} \varepsilon_{m'n}^{(i-1)} \tag{4.2}$$

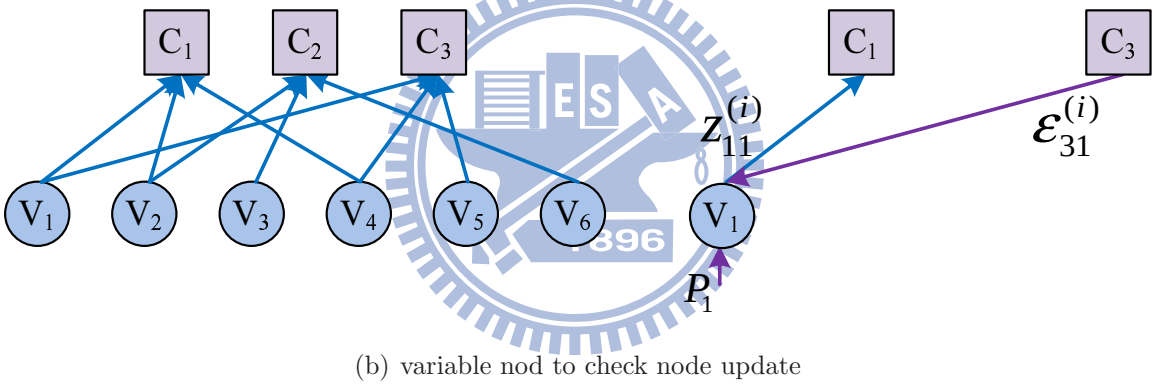
   $$z_{n}^{(i)} = P_n + \sum_{m \in M(n)} \varepsilon_{mn}^{(i-1)} \tag{4.3}$$

3. **Hard Decision:** Let X_n be the n-th bit of decoded codeword. If $z_n^{(i)} \geq 0$, $X_n = 0$, else if $z_n^{(i)} < 0$, $X_n = 1$. If $H(x^{(i)})^t = 0$ or I_{Max} is reached, stop and output the code word. Otherwise,set $i = i + 1$ and go to **Iterative Decoding**.

The iterative decoding processes for one iteration of standard BP is illustrated below. The messages are updated in parallel way between check nodes and variable nodes. The process are shown in Fig. 4.1(a) and 4.1(b). The arrows with purple color represent check node to variable node update message. The arrows with blue color represent variable node to check node update message.



(a) check node to variable node update



(b) variable nod to check node update

Figure 4.1: Illustration of standard BP.

Because of the numeral characteristic of *tanh* function, the absolute value of equation (4.1) will be dominated by $\min(\left|z_{mn'}^{(i-1)}\right|)$. We can approximate (4.1) as following equation. This is so called min-sum algorithm [13].

$$\varepsilon_{mn}^{(i)} \approx \Big(\prod_{n' \in N(m) \backslash n} sign(z_{mn'}^{(i-1)})\Big) \times \min_{n' \in N(m) \backslash n} \Big(\left|z_{mn'}^{(i-1)}\right|\Big) \tag{4.4}$$

The normalized min-sum (NMS) algorithm [14] applies a scaling factor β to compensate for the approximation error. Our LDPC decoder adopt the NMS algorithm, because

it reduces the computational complexity in check node to variable node update step.

$$\varepsilon_{mn}^{(i)} \approx \Big(\prod_{n' \in N(m) \backslash n} sign(z_{mn'}^{(i-1)}) \Big) \times \min_{n' \in N(m) \backslash n} \Big(\Big| z_{mn'}^{(i-1)} \Big| \Big) \times \beta \tag{4.5}$$

4.1.2 Column Shuffled Decoding Algorithm

From the equation (4.5), check node to variable node update step can be implemented by sorters and the number of inputs to sorters is determined by row degree. However, high code rate Quasi-Cyclic (QC) LDPC code constructed by Circulant Permutation Matrices introduce high row degree. The hardware cost and critical path of Check Node Unit (CNU) is greatly increased. Column shuffled decoding algorithm [10] divides received codeword into G groups and processes check node update step in G cycles. Thus, the number of inputs will be reduced.

In column shuffled decoding algorithm, the initialization, stopping criterion test, and output steps remain the same as the standard BP algorithm. The only difference between two algorithms lies in the updating procedure. Assume the N bits of a codeword are divided into G groups, so each group contains $N/G = N_G$ bits. The messages are only exchanged between one group of variable nodes and check nodes which are connected the group of variable nodes at a time. In addition, each group of messages is updated in order. Furthermore, it count one iteration when all groups have been updated. For $G = 1$, the column shuffled decoding becomes standard BP.

1. **Initialization:** $z_{mn}^{(0)} = P_n$

2. **Iterative Decoding:** For $0 \leq g \leq G-1$, perform the following two steps.

   (a) Check node to variable node update step, for $g \cdot N_G \leq n \leq (g+1) \cdot N_G - 1$ and each $m \in M(n)$, process

$$\varepsilon_{mn}^{(i)} \approx \prod_{\substack{n' \in N(m) \backslash n \\ n' \leq g \cdot N_G - 1}} sign(z_{mn'}^{(i)}) \times \prod_{\substack{n' \in N(m) \backslash n \\ n' \geq g \cdot N_G}} sign(z_{mn'}^{(i-1)})$$

$$\times \min \left\{ \min_{\substack{n' \in N(m) \backslash n \\ n' \leq g \cdot N_G - 1}} \left\{ \left| z_{mn'}^{(i)} \right| \right\}, \min_{\substack{n' \in N(m) \backslash n \\ n' \geq g \cdot N_G}} \left\{ \left| z_{mn'}^{(i-1)} \right| \right\} \right\} \times \beta \tag{4.6}$$

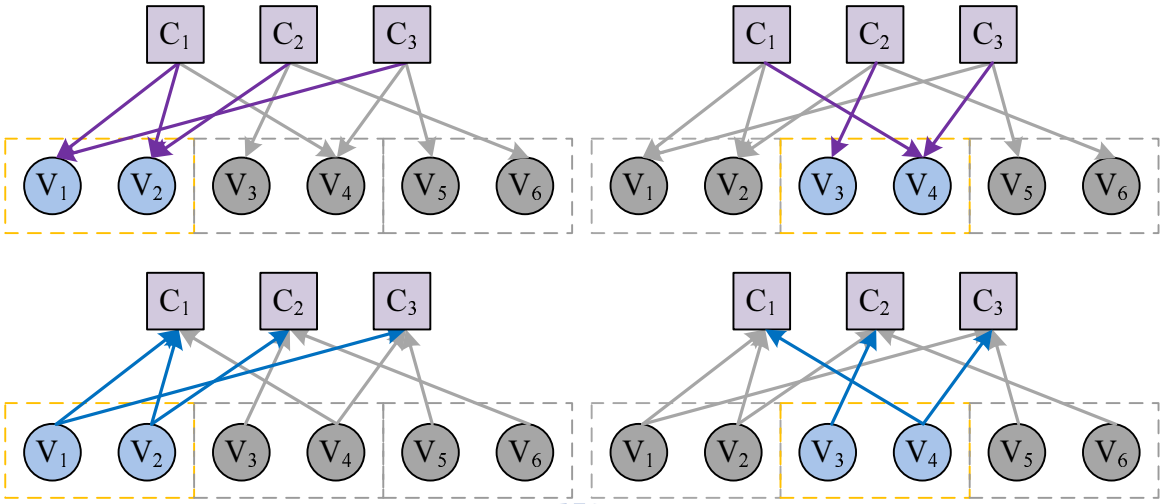
(b) variable node to check node update step, for $g \cdot N_G \leq n \leq (g+1) \cdot N_G - 1$, process

$$z_{mn}^{(i)} = P_n + \sum_{m' \in M(n) \backslash m} \varepsilon_{m'n}^{(i-1)} \tag{4.7}$$

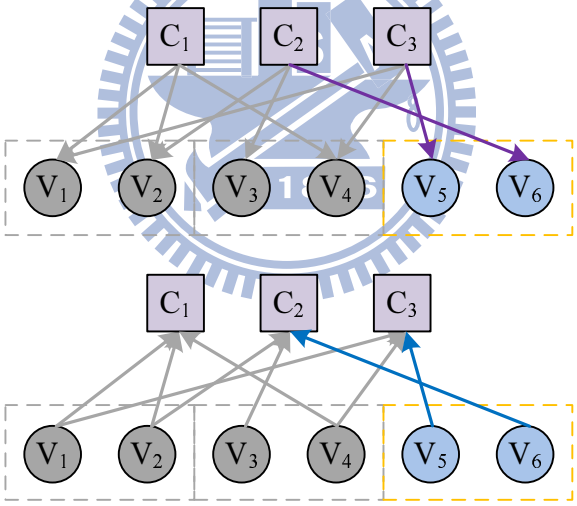
$$z_{n}^{(i)} = P_n + \sum_{m \in M(n)} \varepsilon_{mn}^{(i-1)} \tag{4.8}$$

3. **Hard Decision:** Let X_n be the n-th bit of decoded codeword. If $z_n^{(i)} \geq 0$, $X_n = 0$, else if $z_n^{(i)} < 0$, $X_n = 1$.

The decoding processes for one iteration of column shuffled decoding is illustrated in Fig. 4.2 with $G = 3$ as example. The arrows with purple color represent check node to variable node messages to be updated. The arrows with blue color represent variable node to check node messages to be updated. On the other hand, gray arrows represent that messages are not updated.



(a) Update first group (b) Update second group



(c) Update third group

Figure 4.2: Illustration of VSS

4.2 Area-Efficient Column Shuffled Decoding Architecture

Details of Column Shuffled decoding algorithm is introduced in previous chapter. Hardware architecture for the proposed (9216,8195) LDPC code will be fully explained in this section. Our design is focused on the hardware cost. Therefore, the decoder depicted in Fig. 4.4(a) is composed of partial-parallel CNUs and partial-parallel VNUs. Fig. 4.3 is proposed base matrix with $d_v = 4$, $d_c = 36$. Variable nodes are divided into 36 groups (G = 36). There are 256 Check Node Units (CNUs) and 256 Variable Node Units (VNUs). Let $\alpha_g^{(i)}$ denotes the sorted messages sent from variable nodes in the g-th group to one specific check node at i-th iteration, which is:

$$\alpha_g^{(i)} = \min_{\substack{n' \in N(m) \backslash n \\ g \cdot N_G \leq n' \leq (g+1) \cdot N_G - 1}} \left\{ \left| z_{mn'}^{(i)} \right| \right\} \tag{4.9}$$

Then the magnitude part of check node to variable node message in (4.6) could be computed by the following equation:

$$\left| \varepsilon_{mn}^{(i)} \right| = \min \left\{ \left\{ \alpha_j^{(i)} \right\}_{j<g}, \alpha_g^{(i)}, \left\{ \alpha_k^{(i-1)} \right\}_{k>g} \right\} \tag{4.10}$$

Fig. 4.4(b) demonstrates the timing diagram of proposed decoder. There are G initialization cycles required to calculate α_g^0 for $0 \leq g \leq G-1$. Since only one subgroup of the message $z_{mn}^{(i)}$ is updated in g-th cycle of one iteration, the main operation of CNU could be simplified. Calculate $\alpha_g^{(i)}$ (local sorting) in each cycle and then perform global sorting like equation (4.10).

To reduce the hardware cost, we choose $G = d_c = 36$, so the process of local sorting in equation (4.9) can be omitted. Furthermore, traditional column shuffled decoding completes a full variable node computation in 1 cycle. We divide the computation into d_c, 4 cycles. Fig. 4.3 illustrate that how we divide the check nodes and variable nodes.

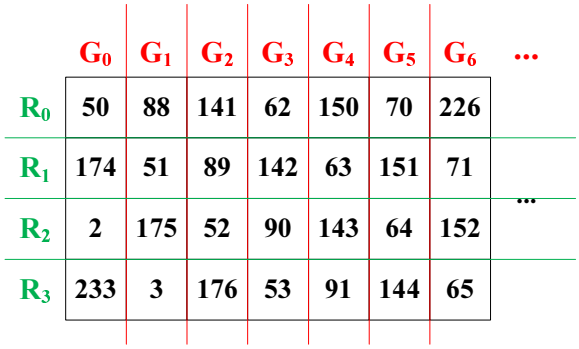
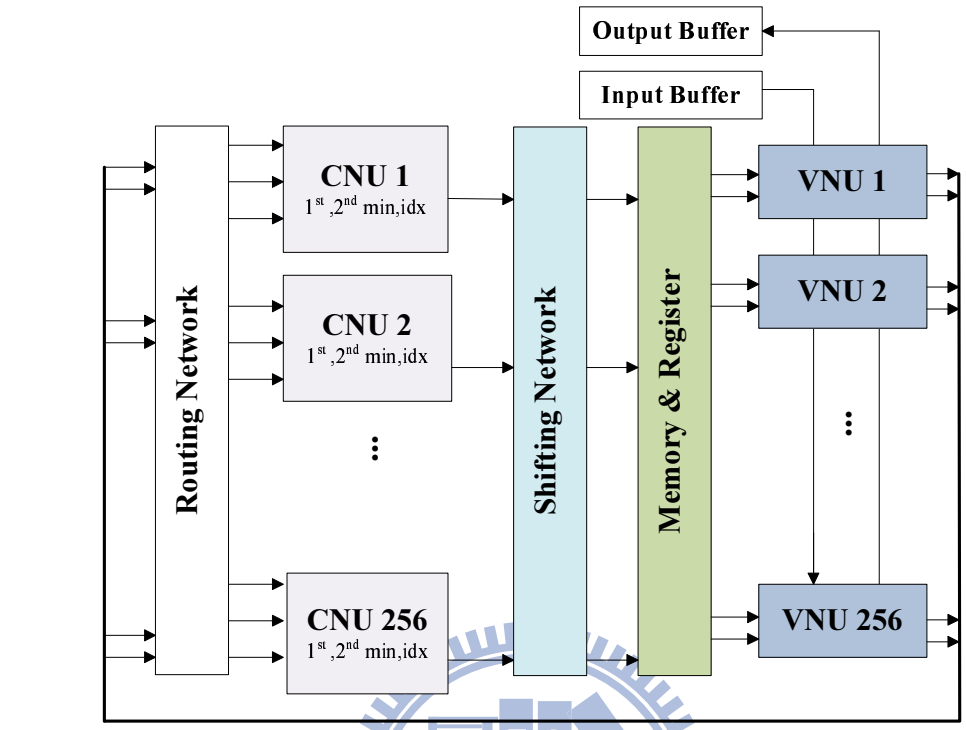
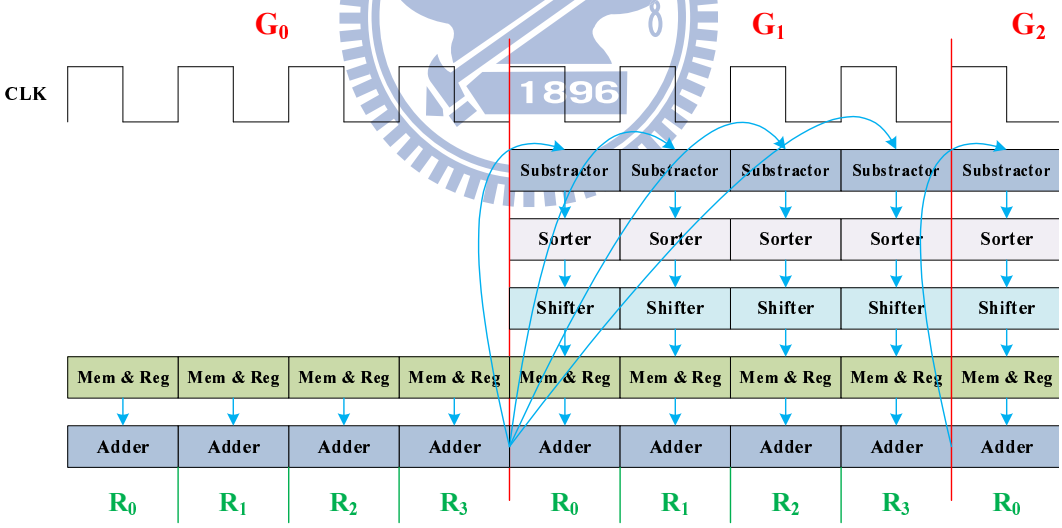


Figure 4.3: Division on the nodes

In the propose architecture, only messages $\alpha_g^{(i)}$ and $\varepsilon_{mn}^{(i)}$ are sorted. The sorted results could be represented by $1st$ min value, $2nd$ min value, and the index of $1st$ and $2nd$ value in NMS algorithm. Therefore, the proposed decoder only latches 2 values, 2 index, and sign part of messages in each subgroup, while the variable node to check node message $z_{mn}^{(i)}$ is on-the-fly calculated. The area-efficient column shuffled decoding architecture is feasible because the CNU could be updated immediately after VNU's operations.



(a) LDPC decoder architecture



(b) Area-efficient Column Shuffled decoding scheduling

Figure 4.4: Proposed architecture and scheduling

4.3 Check Node Unit

This section presents detail CNU architecture based on column shuffled decoding. The CNU architecture is further optimized to reduce storage requirement and the numbers of inputs to sorters. Different CNU architectures will affect the convergence speed and performance which will be discussed in the next chapter. The messages sent from VNU are converted from two's complement format to sign-magnitude format for efficient computation of CNU. Therefore, the operation of check node to variable node update could be divided into magnitude part and sign part.

4.3.1 Accumulative Sorter

For our proposed QC-LDPC codes with $d_c = 36$, The column shuffled with $G = 36$ could divide 36 inputs of the CNU into 36 parts. Thus, a CNU receives only 1 input in g-th group update according to equation (4.9). In NMS algorithm, to implement operation in the equation (4.10) perfectly needs to store $d_c - 1$ $z_{mn}^{(i)}$, these $d_c - 1$ $z_{mn}^{(i)}$ will be sorted with $\alpha_g^{(i)}$. The sorted $1st, 2nd$ min value will be sent as $\varepsilon_{mn}^{(i)}$ in equation (4.2).

However, due to the large storage cost, to store $d_c - 1$ $z_{mn}^{(i)}$ for the sorted $1st, 2nd$ min value is impractical. Only 2 $z_{mn}^{(i)}$ are stored in our CNU architecture. The inputs of the sorter are 2 $z_{mn}^{(i)}$ and 1 $\alpha_g^{(i)}$. It's a simple 3 to 2 accumulative sorter. Proposed CNU architecture reduces large storage cost and hardware cost. But, it suffer from performance loss, because it may lead to wrong results in sorted $1st, 2nd$ min value while reducing the numbers of stored $z_{mn}^{(i)}$.

Fig. 4.5 is an example for the operation of accumulative sorters. In this example, we assume row degree = 5 and $G = 5$. Follow the operation in the equation (4.10), the sorted $1st, 2nd$ min result in 1-th group in iteration 2 should be 0.75 and 0.75. Fig. 4.5(a) shows the sorted results with 2 $z_{mn}^{(i)}$ stored. We get wrong sorted results in $1st, 2nd$ min value. Fig. 4.5(b) shows the sorted results with 3 $z_{mn}^{(i)}$ stored. The $1st$ min value is correct, but $2nd$ min value is still wrong. The problem resulted from the conflict between index of input value and the index of stored $1st, 2nd$ min value.

	initialize					iteation 1					iteation 2		
Group	**0**	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**4**	**0**	**1**	**2**
input	0.25	0.5	0.75	1.0	0.75	0.25	0.5	0.75	1.0	0.75	1.25	1.5	1.75
1st min	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.5	1.25	1.25
2nd min	∞	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.25	1.5	1.5

...

(a) Reserve 2 $z_{mn}^{(i)}$

	initialize					iteation 1					iteation 2		
Group	**0**	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**4**	**0**	**1**	**2**
input	0.25	0.5	0.75	1.0	0.75	0.25	0.5	0.75	1.0	0.75	1.25	1.5	1.75
1st min	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.5	0.75	0.75
2nd min	∞	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.75	1.25	1.25
3rd min	∞	∞	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	1.25	1.5	1.5

...

(b) Reserve 3 $z_{mn}^{(i)}$

Figure 4.5: Accumulative sorters with different numbers of stored $z_{mn}^{(i)}$

4.3.2 Optimization Strategy

Increase the number of stored $z_{mn}^{(i)}$ can reduce the index conflict problem at the cost of more storage and gate count. In proposed CNU architecture, the sorter is a very simple 3 to 2 accumulative sorter. The rules in replacing $1st, 2nd$ min should be considered carefully in order to reduce the conflict problem. The main idea is to reserve the latest index if the sorted value are the same. Equation (4.11) and (4.12) will lead to different sorted results which are demonstrated in Fig. 4.6.

$$input \ < \ 1st \ min \ , \ input \ < \ 2nd \ min \tag{4.11}$$

$$input \ \leq \ 1st \ min \ , \ input \ \leq \ 2nd \ min \tag{4.12}$$

The correct sorted $1st, 2nd$ min in 1-th group in iteration 2 should be 0.25 and 0.75. The sorted $1st, 2nd$ min following the equation (4.11) is 1.25 and 1.5. The sorted $1st, 2nd$ min following the equation (4.12) is 0.25 and 1.5. It's obvious that equation (4.12) is close to the correct $1st, 2nd$ min.

	initialize					iteation 1					iteation 2			
Group	0	1	2	3	4	0	1	2	3	4	0	1	2	
input	0.25	0.25	0.75	0.25	0.75	0.25	0.25	0.75	0.25	0.75	1.25	1.5	1.75	...
1st min	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	1.25	1.25	
2nd min	0.75	0.25	0.25	0.25	0.25	0.75	0.25	0.25	0.25	0.25	1.25	1.5	1.5	

(a) Equation (4.11)

	initialize					iteation 1					iteation 2			
Group	0	1	2	3	4	0	1	2	3	4	0	1	2	
input	0.25	0.25	0.75	0.25	0.75	0.25	0.25	0.75	0.25	0.75	1.25	1.5	1.75	...
1st min	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	
2nd min	0.75	0.25	0.25	0.25	0.25	0.75	0.25	0.25	0.25	0.25	0.25	1.5	1.5	

(b) Equation (4.12)

Figure 4.6: Accumulative sorters with different replacing rules

The two different replacing rules result in different performance. In Fig. 4.7, red line shows the BER using equation (4.12) and green line shows the BER using equation (4.11). Using equation (4.12) can achieve better performance.

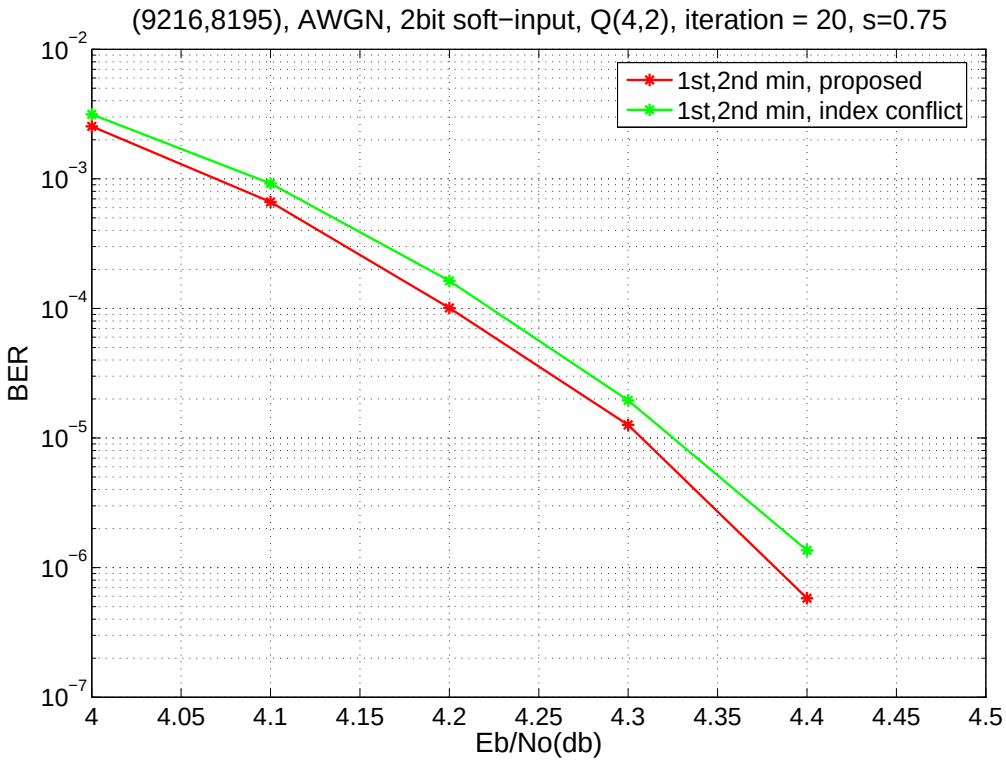


Figure 4.7: Performance for accumulative sorter with different replacing rules

4.4 Variable Node Unit

Fig. 4.8 shows the VNU architecture, where SM to TC represents sign-magnitude to two's-complement conversion, and TC to SM represent two's-complement to sign-magnitude conversion. Since column degree is 4, the adder takes 4 cycles to compute the posteriori LLR $z_n^{(i)}$ in g-th group. $\varepsilon_{mn}^{(i)}$ used in the adder should be stored to calculate the $z_{mn}^{(i)}$ sent to $(g+1)$-th group.

The bit width of messages passing between CNU and VNU is 4. Scaling factor 0.75 in NMS algorithm (4.5) is applied in our architecture. Small value 0.25 will not be multiplied by scaling factor in order to reserve its information. 2 bits channel value is mapped to 4 bits value by non-linear quantization. More details of non-linear quantization will be discussed in next chapter.

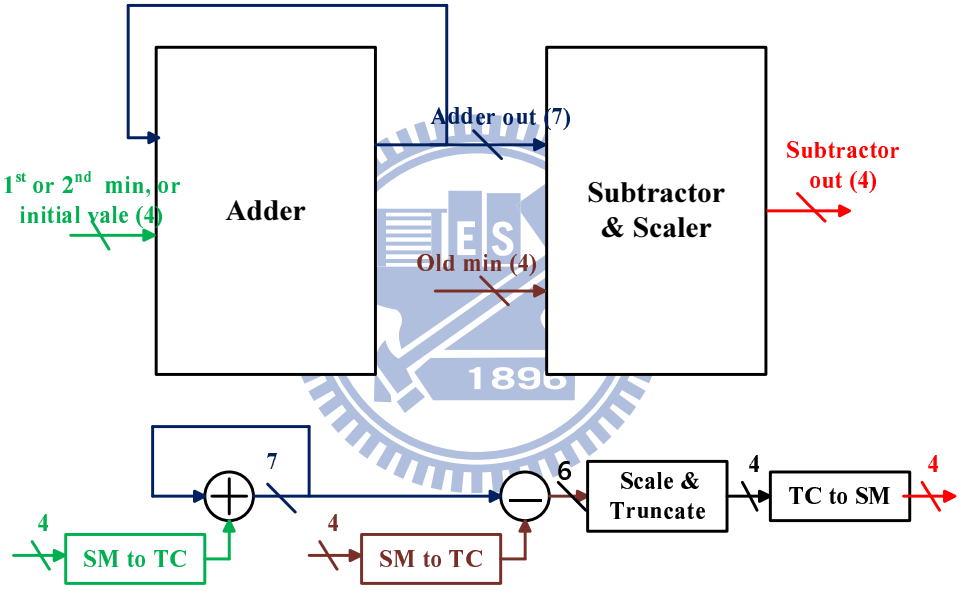


Figure 4.8: VNU architecture

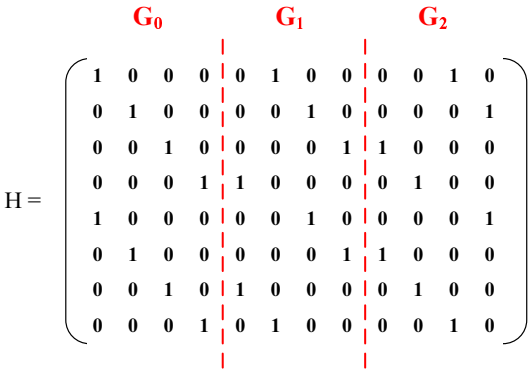
4.5 Shifting Network

High compexity of routing network between Check Node Units (CNU) and Varible Node Units (VNU), is the main difficulty for hardware implementation of LDPC code. Shifting Network [15] [16] has been proposed to reduce the routing complexity. There are two routing networks between CNUs and VNUs. One is the direction from CNUs to VNUs, while another one is the direction form VNUs to CNUs.

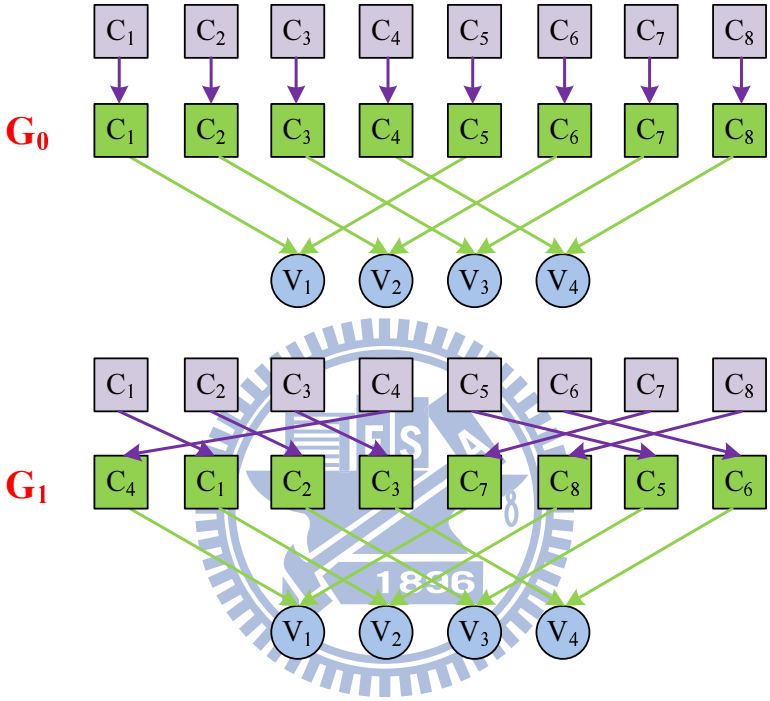
Due to the quasi-cyclic character in Latin square QC-LDPC code, the shifting network can be simplified. The value computed by CNUs will be stored to memories. Operations of VNUs start with fetching the value from memories. The routing networks from memories to VNUs are fixed. Therefore, shifting networks between CNUs and memories are needed. The idea is illustrated in Fig. 4.9. Green lines represent the routing networks from memories to VNUs. Purple lines represent shifting networks between CNUs and memories.

The shifting network in Fig. 4.9(b) in G_0 can be ignored, because the sub-matrix in G_0 are two identity matrices. From previous discussion 3.2.1, adding a fixed value in a row in base matrix W will not change the check equations produced by the row. W_1 in Fig. 4.10 is the original base matrix proposed. W_2 is the equivalent base matrix to W_1 with four identity matrices in G_0. Thus, the shifting network on the initial cycle in G_0 can be ignored.

However, the difference between cyclic shift amount of each group is a not constant. A table is constructed to record the difference between cyclic shift amount of each group. The shifting network in our design is a traditional Barrel shifter.



(a) A QC-LDPC code divided into 3 groups



(b) Shifting network for correspondent code

Figure 4.9: Illustration of networks between CNUs and VNUs

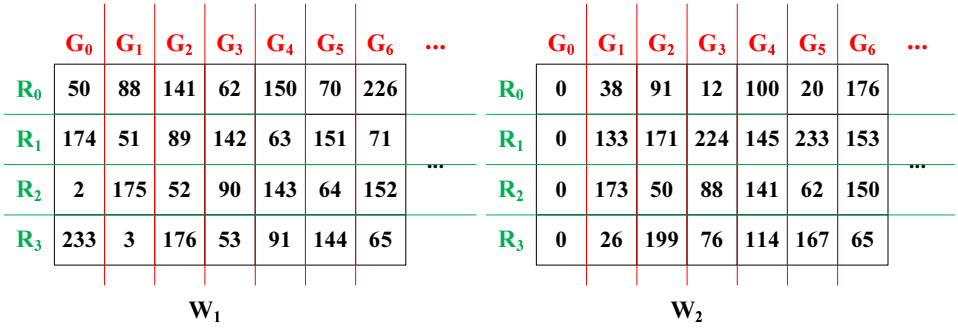


Figure 4.10: Equivalent base matrices W_1 and W_2

Chapter 5

Simulation and Implementation Results

5.1 Optimized Quantization

Belief Propagation (BP) is a probability-based message passing algorithm. When soft input is available, LDPC code can provide powerful correcting ability. LDPC code with 2-bits soft input can outperform BCH code under same code rate. Additive White Gaussian Noise (AWGN) channel with Binary Phase Shift Keying Modulation (BPSK) are used for demonstration and simulation. We assume that bit '0' is mapped to '1' and bit '1' is mapped to '−1'. 2-bits quantization can represents 4 levels. We select a threshold f to divide received channel value into 4 levels as shown in Fig. 5.1. A bit with channel value near 0 has a high probability to be an error bit. Therefore, a non-linear quantization is preferred.

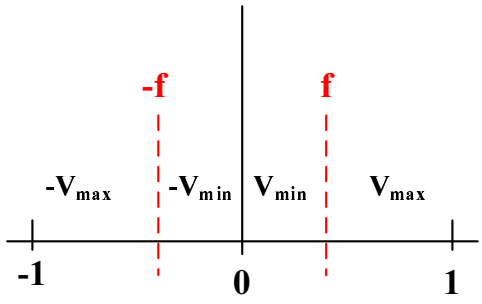


Figure 5.1: 2 bits (4 levels) non-linear quantization.

The value of f, V_{min} and V_{max} will affect the code performance severely. We use Fig. 5.2 to explain how to derive appropriate parameters for 2-bits quantization. Once the f is determined, received channel value is divided into 4 regions. The main idea is to find the value that can mostly represent all the value in the region. Therefore, the concept of weighted mean is applied.

$$x_w = \frac{\sum w_i x_i}{\sum w_i} \tag{5.1}$$

In Fig. 5.2, given $f = 0.35$, SNR= 4.0, $(V_{min}, V_{max}) = (0.2390, 1.0813)$ can be derived from equation (5.1).

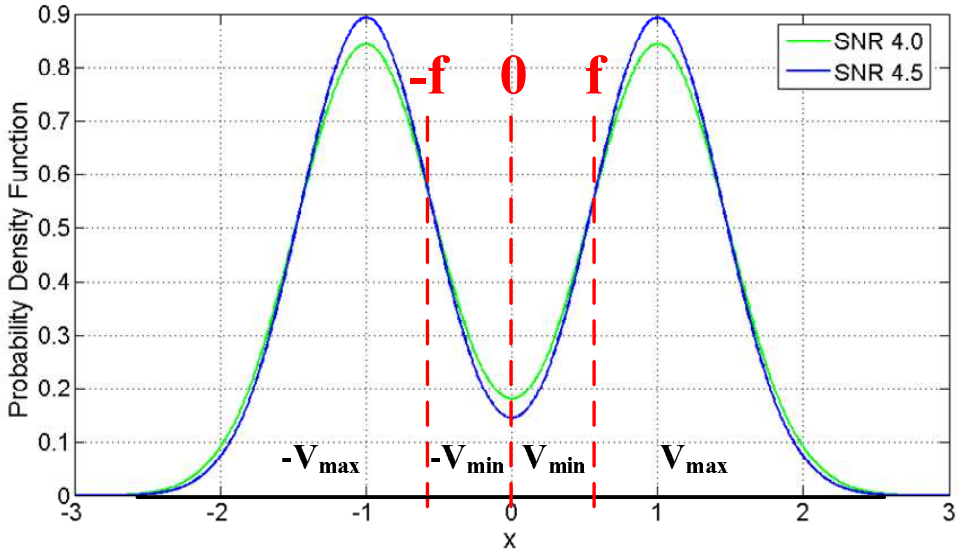


Figure 5.2: Received channel value distribution for (9216,8179) LDPC code

Fig. 5.3 shows the performance with different (f, V_{min}, V_{max}). The bit width of input LLR after non-linear quantization and messages passing between CNUs and VNUs in decoder is floating. Decoding algorithm is Normalized Min-Sum algorithm. $(f, V_{min}, V_{max}) = (0.35, 0.25, 0.75)$ and $(f, V_{min}, V_{max}) = (0.35, 0.5, 1.5)$ have the same performance, because they have the same (V_{max}/V_{min}) ratio. However, the parameters from equation (5.1) can appropriately represent the value in the divided region, it is not ensured that the parameters provide the best decoding performance. The bit width and the algorithm used in the decoder will affect the final result, but the (V_{max}/V_{min}) ratio is still a good reference for us. In Fig. 5.3, (V_{max}, V_{min}) with (V_{max}/V_{min}) ratio near the derived (V_{max}, V_{min}) have good performance.

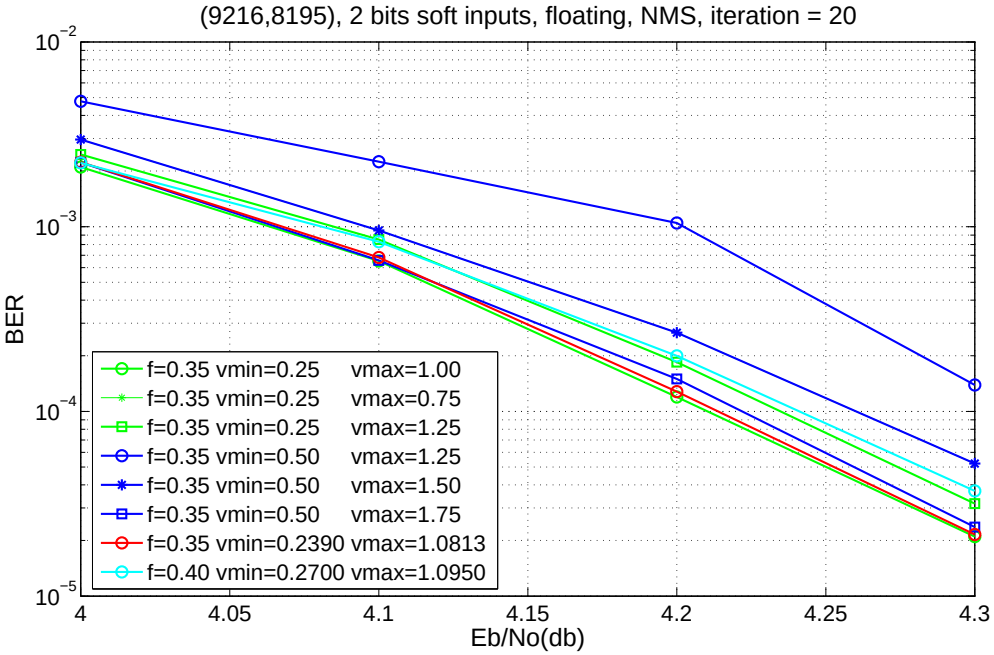


Figure 5.3: Code performance with different (f, V_{min}, V_{max}), floating

Fig. 5.4 also shows the performance with different (f, V_{min}, V_{max}), but the bit width of input LLR after non-linear quantization and messages passing between CNUs and VNUs in decoder is 4. $(f, V_{min}, V_{max}) = (0.35, 0.25, 0.75)$ and $(f, V_{min}, V_{max}) = (0.35, 0.5, 1.5)$, which have the same performance in Fig. 5.3, now have $0.15dB$ performance difference at BER 10^{-3}. Assume two sets of (V_{max}, V_{min}) with the same (V_{max}/V_{min}) ratio, larger (V_{max}, V_{min}) provides better performance. Hence, $(f, V_{min}, V_{max}) = (0.35, 0.5, 1.75)$ is chosen.

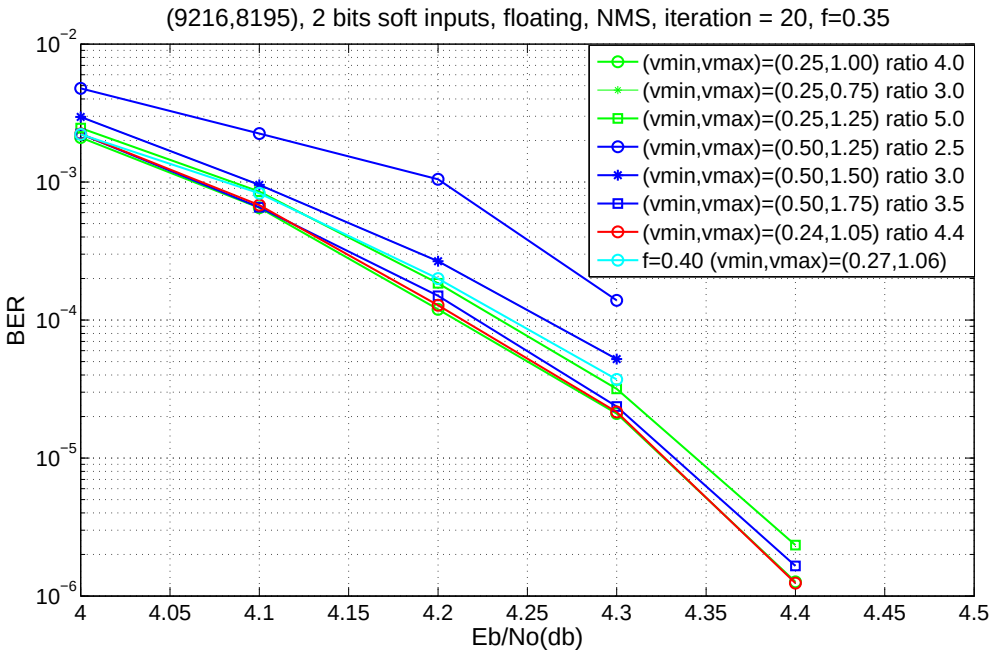


Figure 5.4: Code performance with different (f, V_{min}, V_{max}), Q(4,2)

5.2 Performance Evaluation

Fig. 5.5 shows that the BER performance of proposed Area-Efficient Column Shuffle decoding algorithm converges faster than and NMS algorithm and the CNU with index conflict cases.

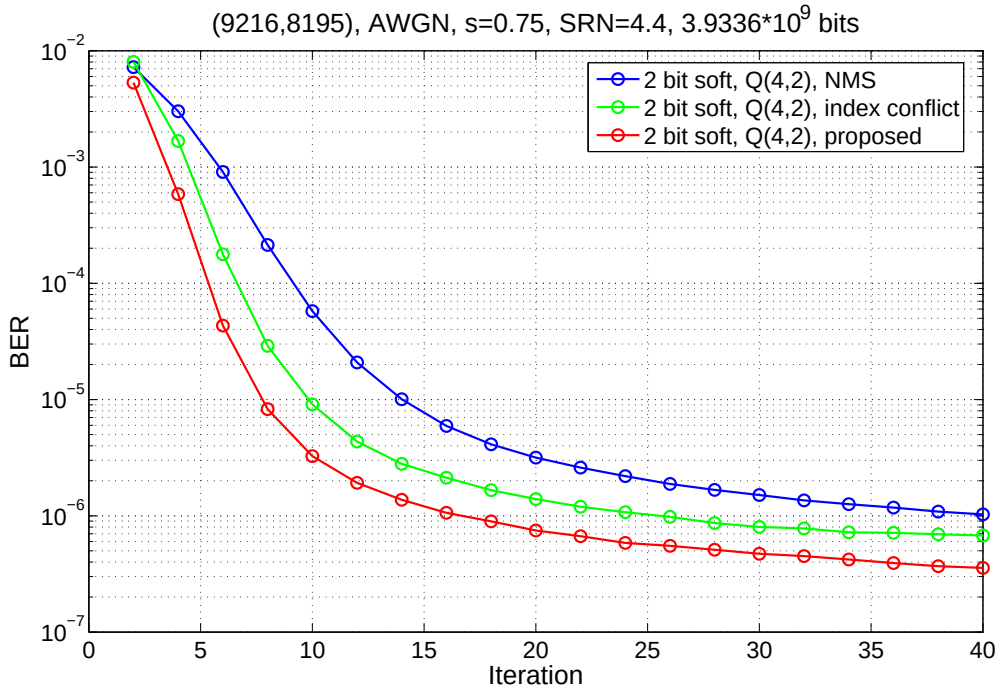


Figure 5.5: Converge Speed Comparison at SNR 4.4

In Fig. 5.6, there is $1.3dB$ performance gain of 2-bit non-linear soft input LDPC code over BCH code at BER 10^{-4}. 2-bit non-linear soft input LDPC code has a great potential to replace BCH code for NAND flash memory system. The simulation parameters of LDPC code are 4-bit quantization (2-bit integer and 2-bit decimal fraction), with scaling factor 0.75. The bit width of messages passing between CNU and VNU is 4. Area-Efficient Column Shuffle decoding architecture with 36 group partition, 4 row partition reduce the amount of CNUs and VNUs, inputs to CNUs, and inputs to VNUs. Since the converge speed of proposed algorithm is faster than the converge speed of NMS algorithm. With 20 iterations, the performance of proposed algorithm is better than NMS algorithm.

Unfortunately, Fig. 5.7 shows that error floor appears at BER 10^{-9}. The performance degradation may result from low column degree or information loss from soft input to 2-bit input.

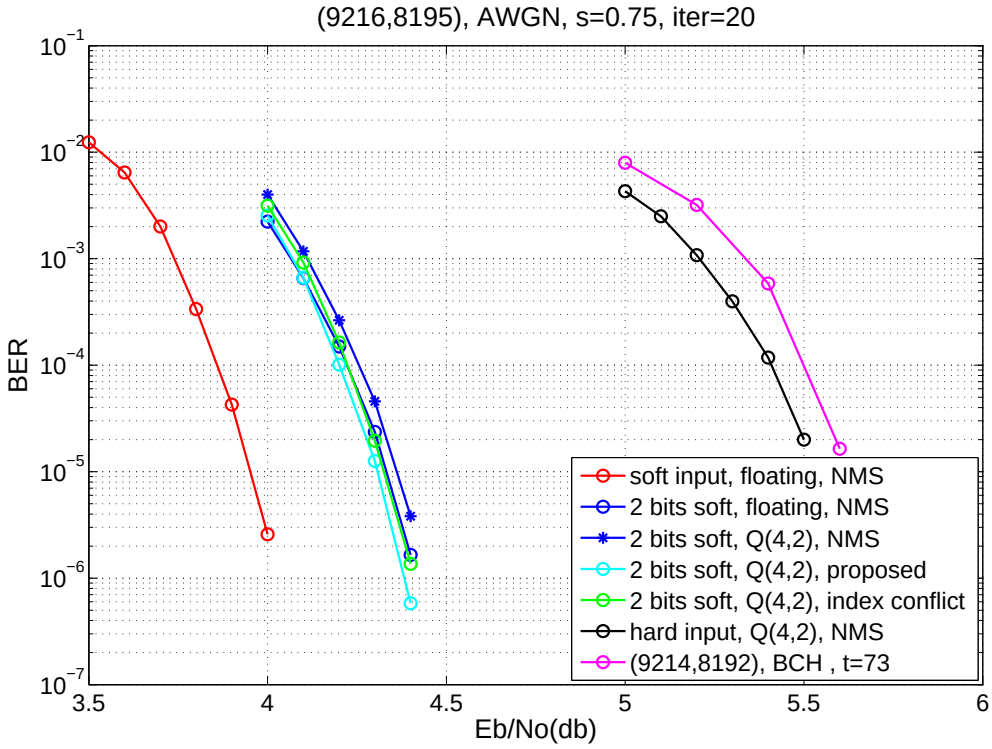


Figure 5.6: Code performance

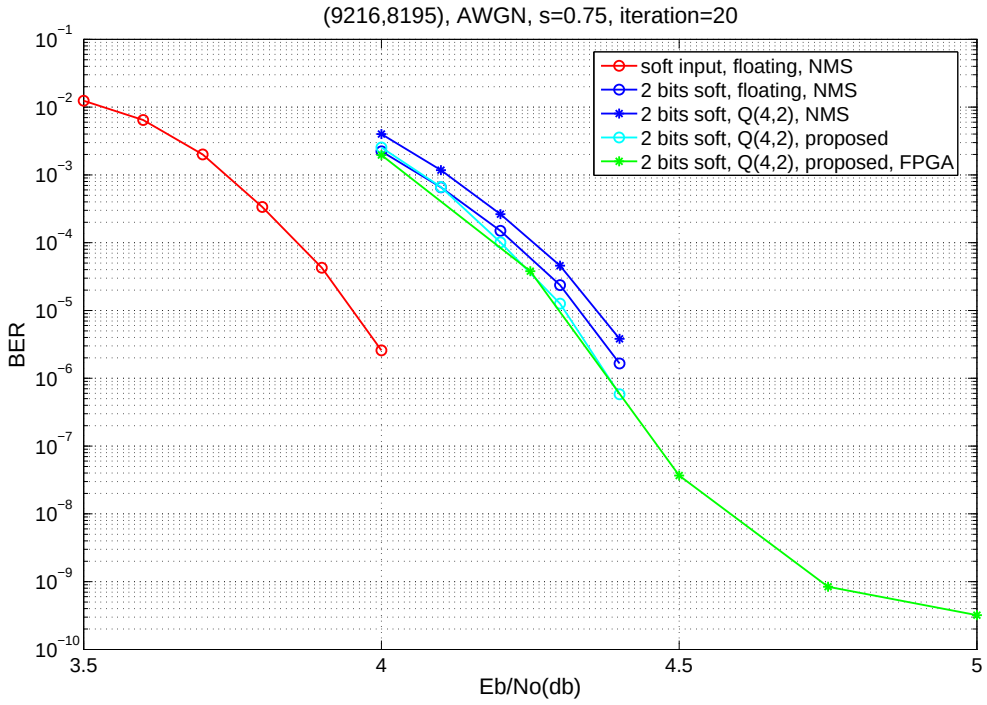


Figure 5.7: Code performance simulated by FPGA

5.3 Synthesis Results

The critical path of CNUs + Shifters + VNUs is 5ns. We assume that the critical path of control circuit is 1ns. Therefore the clock cycle after synthesis is 6ns. Clock period in Place and Route is 9ns.

According to the simulation result from Table 5.1, 4 decoding iteration is sufficient to decode most codewords in high SNR region.

Table 5.1: Early Termination Simulation at different SNR, 10^5 codewords

SNR	4.5	4.75	5.0	5.25
Average decoding iterations	4.137	3.323	2.853	2.426

$$\begin{aligned} Throughput &= \frac{Informationlength}{Cycles\ per\ iteration \cdot Numbers\ of\ iteration \cdot Cycle\ period} \\ &= \frac{8195}{36 \cdot 4 \cdot 4 \cdot 9} = 1.581 Gbps. \end{aligned}$$

Synthesis results is listed in Table. 5.2. Total gate count is 605.35k whereas the shifter accounts for 105.2k, 17.38% of total design.

Table 5.2: Synthesis Results with technology UMC90.

		Gate count
Combinational circuits	VNU (Adder,Substractor)	90.49k
	CNU (Sorter)	69.12k
	Shifter	105.20k
Memory	Channel value	80.80k
	Hard decision	40.40k
	Sign Bits	66.60k
Register	$1st, 2nd$ min, idx	147.40k
	Old sign, min	32.77k
Estimated result		632.78k
Final result		605.35k

5.4 Implementation Results

Table. 5.3 shows the postlayout results. Gate count after synthesis is 605.35k and Core area is $3.74mm^2$ without IO pad. Using 90nm CMOS technology, the maximum throughput is 1.581 Gbps under clock period 9ns with 4 iterations.

Table 5.3: Summary of implementation results (Place and Route).

	Proposed LDPC Decoder
Technology	UMC 90nm 1P9M
Code Spec	(9216,8195)
Code Rate	0.889
Column Degree	36
Row Degree	4
Algorithm	Area-efficient Column Shuffle Decoding
Area	$3.74mm^2$ (Without IO Pad)
Gate Count	605.35k
Iteration	20
Input Quantization	2 bits
Clock Period	9ns
Maximum Throughput	1.581 Gbps

The core density in this design is 69.83 %, but its density distribution is quite unbalanced. The 256 bits Barrel Shifter results in serious congestion problems. Clock period must be increased to solve the congestion problems. The clock period after synthesis is 6ns. Clock period in Place and Route is 9ns.

In Table. 5.4, the gate count of our proposed design is approximate 3 times of the (9214,8192) BCH code design.

Table 5.4: Comparison with BCH codes

	Proposed LDPC Code	BCH Code
Code Spec	(9216,8195)	(9214,8192)
Code Rate	0.89	0.89
Column Degree	4	t=73
Throughput	1.581 Gb/s	2.41 Gb/s
Gate Count(No I/O Buffer)	484.2k	166.4k

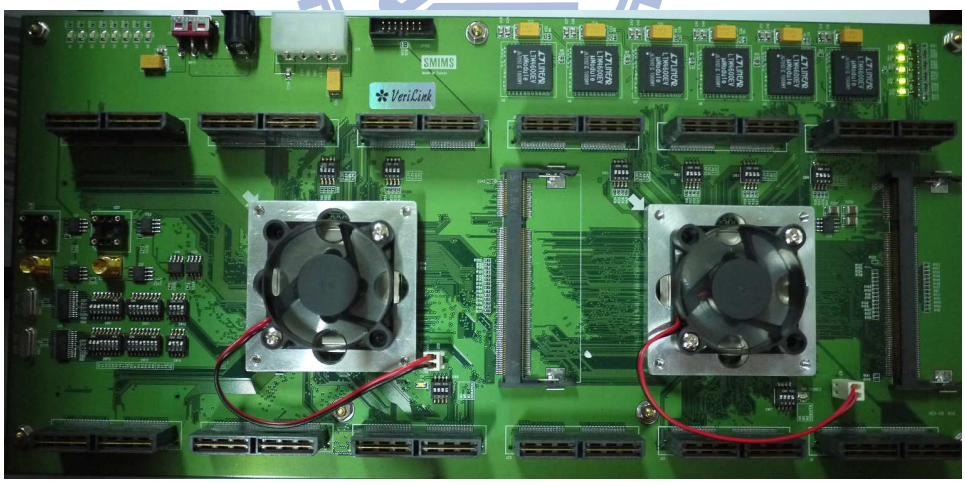


Figure 5.8: BPSK Emulation using FPGA: Xilinx Virtex-5 LX330 with FF1760 package

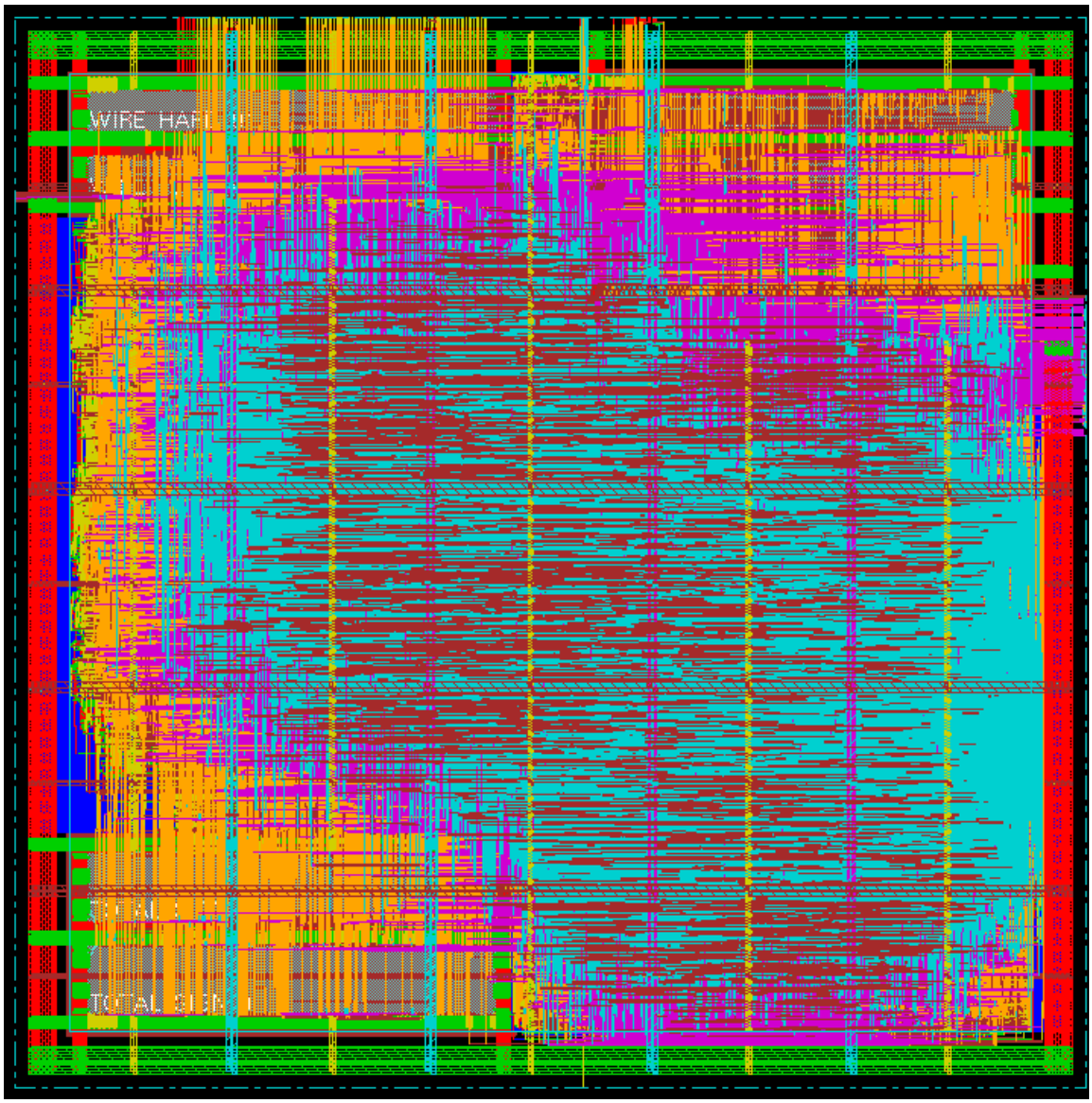


Figure 5.9: Chip Layout in Place and Route

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis proposes a (9216, 8195) LDPC code with code rate 0.89 for NAND flash memory system. (9216, 8195) LDPC code is constructed from the base matrix produced by Latin square algebra, with column degree 4, row degree 36. The size of CPM different from original Latin square is applied in order to make information length > 8192. Parameters for 2 bits quantization is calculated based on the concept of weighted mean. Simulations show that LDPC code with 2-bit soft input can outperform BCH code under same code rate. Therefore, LDPC code is a good candidate to replace BCH code in the next generation standard. High code rate LDPC code introduces high row degree. This makes implementation difficult due to the large number of inputs to sorter and the increased routing complexity. Area-efficient Column Shuffled decoding algorithm is a good solution to this problem. Variable nodes are divided into 36 groups. Check node update procedures are processed in 36 cycles, reducing the number of inputs to sorter. Only $1st, 2nd$ min are reserved to reduce the storage cost. Replacing rules in the accumulative sorter is further optimized for performance. With row divided into 4 subgroups, VNU can be simplified to a 2-inputs adder and a 2-input subtractor. Shifting networks are applied between CNUs and memories. The gate count of our design is 605.35k. The maximum throughput can achieve 1.581 Gbps with 4 iterations, using 90nm CMOS technology.

6.2 Future Work

The gate count of shifters account for 17.38% of total design. If we can further simplified the shifters, critical path and gate count of our design can be lowered, and the throughput can also be promoted. The study in the regulation of the base matrix may be a solution to this problem.

FPGA simulation shows that error floor appears at BER 10^{-9}. Error correcting code applied on NAND flash memory system requires no performance degradation down to BER near 10^{-12}. The most probable reason resulting in performance degradation is that only $1st, 2nd$ min are stored. Wrong sorted results will propagate in iterative decoding process. The replacing rules in accumulative sorter should be modified to make the sorted result more accurately.

There is no standard flash memory channel for any simulation. Therefore, a standard flash memory channel is desired if we want to compare performances of different error correcting code on flash memory. We may use the unsymmetrical AWGN channel for more accurate simulation.

Bibliography

[1] A. M. R. Micheloni, L. Crippa, *Inside NAND Flash Memories*. Springer, 2010.

[2] A. F. D. M. Greg Atwood and B. Reaves, "Intel strataflashtm memory technology overview," *Intel Technology Journal*, pp. 1–8, 4th Quarter 1997.

[3] R.C.Bose and D.K.Ray-Chaudhuri, "On a class of error-correcting binary group codes," *Inform. and Contr*, no. 3, pp. 68–79, March 1986.

[4] A. Hocquenghem, "Codes correcterus derreurs," *Chiffres*, no. 2, pp. 117–156, September 1959.

[5] I. Reid, W.J., L. Joiner, and J. Komo, "Soft decision decoding of bch codes using error magnitudes," *IEEE Int. Symp. on Info. Theory*, p. 303, June 1997.

[6] Y. M. Lin, C. L. Chen, H. C. Chang, and C. Y. Lee, "A 26.9 k 314.5 mb/s soft (32400,32208) bch decoder chip for dvb-s2 system," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 11, pp. 2330–2340, Nov. 2010.

[7] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

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

[9] 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.

[10] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[11] M. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, aug. 2004.

[12] L. Zhang, Q. Huang, S. Lin, K. Abdel-Ghaffar, and I. Blake, "Quasi-cyclic ldpc codes: An algebraic construction, rank analysis, and codes on latin squares," *IEEE Transactions on Communications*, vol. 58, no. 11, pp. 3126–3139, Nov. 2010.

[13] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, may 1999.

[14] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, mar 2002.

[15] H. C. C. C. H. Liu, C. C. Lin and C. Y. Lee, "Multi-mode message passing switch networks applied for qc-ldpc decoder," *IEEE Internatinal Symposium on Circuits and Systems*, vol. 18, no. 1, pp. 85–94, Jan. 2010.

[16] D. Oh and K. Parhi, "Area efficient controller design of barrel shifters for reconfigurable ldpc decoders," *IEEE Internatinal Symposium on Circuits and Systems*, pp. 240–243, May 2008.