國立交通大學

電子工程學系　　電子研究所碩士班

碩 士 論 文

非二進位之低密度同位檢查碼的編碼器設計與實作

Design and Implementation of Non-binary Low-density Parity-check Codes (NB-LDPC) Decoders

學生：涂淑文

指導教授：李鎮宜教授

中華民國一〇一年八月

非二進位之低密度同位檢查碼的解碼器設計與實作

Design and Implementation for Non-binary Low-density Parity-check Codes (NB-LDPC) Decoders

研 究 生：凃淑文
指導教授：李鎮宜 博士

Student：Shu-Wen Tu
Advisor：Dr. Chen-Yi Lee

國 立 交 通 大 學
電子工程學系 電子研究所碩士班
碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering

& Institute of Electronics

College of Electrical and Computer Engineering

National Chiao-Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In

Electronics Engineering

Aug. 2012
Hsinchu, Taiwan, Republic of China

中 華 民 國 一 〇 一 年 八 月

非二進位之低密度同位檢查碼的解碼器設計與實作

學生：凃淑文　　　　　　　　　　　　指導教授：李鎮宜　教授

國立交通大學

電子工程學系　電子研究所碩士班

摘　　要

由低密度同位檢查碼衍伸而來的非二進位低密度同位檢查碼，不僅具有極佳的錯誤更正能力，並且在通訊品質較差的傳輸環境下更能克服通道的雜訊。非二進位低密度同位檢查碼具有極佳的解碼能力，但是複雜的運算以及大量的記憶體需求，是其硬體實現上急需克服的問題與挑戰。在本篇論文中，應用可以加速收斂速度的分層解碼架構，我們提出了具有高硬體效能的非二進位低密度同位檢查碼的解碼器設計。根據擴展最小和算法(EMS)的解碼演算法，我們在檢查點(CNU)的運算上使用雙倍吞吐量去提升整體解碼器的運算速度，並善加利用同位檢查矩陣(H)本身的結構特性，使所需的訊息儲存量縮減為原本的一半。最後，利用 UMC 90 奈米 CMOS 製程，我們實作了一個(112,56)，應用在 GF(64)下的非二進位低密度同位檢查碼之解碼器來展示我們的構想，與目前其他研究的成果相比，我們所提出的解碼器架構，在硬體效能上擁有至少 4 倍以上的優勢。

誌　謝

碩班其實真的很短，這兩年裡，除了跟研究題目日夜糾纏外，也充滿著跟實驗室同伴們相處的點點滴滴，正因為有大家的陪伴與幫助，讓我在研究上有可以討論的對象並時常得到寶貴的意見，在生活上有一起吃吃喝喝的同好，更在 Tape-out 與趕畢業之際有互相扶持的好夥伴們，讓我可以開心且順利的渡過這兩年的碩班生活。

首先要感謝指導教授李鎮宜老師，總是以溫和的言語給與我寶貴且實際的建議，指引我研究的方向與要點，還有活潑且親和力十足的錫嘉老師，以自己對研究的態度給我們做了最好的示範，謝謝老師在我研究上的指導與鼓勵。接下來就是要感謝 SI2 與 OCEAN 的學長姐們，特別要感謝指導我的佳龍，對於我有時粗心難以理解的思維還是給予耐心且好脾氣的教導與指正，還有阿龍在研究上時常給我一針見血的意見，並且擴大我的新竹美食地圖。還有駕駛技術一流的長宏，尤其感謝你在 Tape-out 期間對我們的協助與照顧，可愛的飯咖兼美食家小肥，好脾氣且盡責處理實驗室大小事的欣儒，開始分享爸爸經的義澤，總是給我最中肯誠實意見的義閔，用言語鞭策我要努力的智翔，在研究上給我很多幫助的 RTL 達人巴博，當然還有 SI2 與 OCEAN 的眾學長姐們：柏均、李曜、歐陽、宣婷、魚爺、建螢、宋仔、佳融、人偉、渠、其衡、小約、玉祥、士家，很感謝你們在研究上對我的協助與建議。

除了感謝老師及學長姐們外，我也要感謝過程中一起同甘共苦的好朋友們，很開心能與雞皮、柚子、小朱哥、奕勳在最後畢業時刻，一起互相扶持努力的撐過去，感謝佩好、美維一直以來的聆聽與鼓勵，以及在各種大小事上的協助幫忙，當然還有 SI2 與 OCEAN 的同學及學弟們：博堯、恕平、家麟、宇滔、大嘴、思齊、日和、方舟、泓源，感謝你們給我的加油打氣，跟大家相處的日子都是我很開心的回憶，當然還有很多沒有提及的朋友們，很感謝大家在我碩班生涯的陪伴。

最後，我要感謝我的家人，謝謝我的爸爸和媽媽，在我學習的路上總是給我自由發揮的空間，給予我最大的精神鼓勵而且是我最重要的後盾，謝謝我的姐姐，不管是學業與生活，總是在一旁協助鼓勵我，給我最真誠的意見，還有我可愛的弟弟，謝謝你總是對我信心滿滿，讓我也覺得自己可以做得更好，謝謝我親愛的家人讓我一路以來開心的學習並健全的成長。

中華民國　一〇一年　九月

涂淑文

Abstract

Non-binary LDPC codes which extended from binary LDPC codes have excellent decoding performance, and it is robust to various channel impairments. With the remarkable decoding ability, the high computational complexity and huge memory usage are the main challenges for non-binary LDPC codes to be implemented in practical. This thesis presents a high hardware efficient architecture for implementing non-binary LDPC decoder using improved Extended Min-Sum decoding algorithm with layered scheduling. Based on the enhancement in the check node processing and efficient memory storing, the proposed decoder can double the throughput and have half reduction in storing the edge messages. Using 90-nm CMOS process technology, a (2,4)-regular non-binary QC-LDPC decoder over GF(2^6) is implemented. In the post-layout simulation results, the decoder throughput can reach over 100 Mbps at 10 iterations. Compared with state-of-the-art designs, this implementation has at least 4.3 times improvement in hardware efficiency (throughput-to-gate-count-ratio), and the decoding performance still keep competitive.

Contents

1 Introduction **1**
1.1 Research Motivation . . . 1
1.2 Thesis Organization . . . 3

2 Principle of Non-binary Low Density Parity Check Codes **4**
2.1 Basic Introduction of Finite Field . . . 4
2.2 Non-binary LDPC Codes . . . 5
2.3 Encoding of Non-binary LDPC Codes . . . 6
2.4 Decoding of Non-binary LDPC Codes . . . 7
2.4.1 Sum of Product Algorithm (SPA) . . . 8
2.4.2 Extended Min-Sum Algorithm (EMS) . . . 13
2.4.3 Min-Max Algorithm . . . 16

3 Non-binary LDPC Decoder Architecture **19**
3.1 Non-binary Quasi-Cyclic LDPC Codes . . . 19
3.1.1 Code Structure . . . 19
3.2 Decoder Architecture . . . 20
3.3 Forward and Backward Algorithm . . . 21
3.4 Check Node Unit (CNU) . . . 22
3.4.1 Check Elementary Step (CES) . . . 23
3.4.2 Bubble Check Algorithm . . . 26
3.4.3 Proposed Check Elementary Step . . . 30
3.5 Variable Node Unit (VNU) . . . 35
3.5.1 Variable Elementary Step (VES) . . . 35
3.5.2 Proposed Variable Node Unit . . . 38
3.6 Scheduling . . . 43
3.6.1 Layered Scheduling . . . 43
3.6.2 Early Termination . . . 44
3.6.3 Memory Configuration . . . 44

4 Implementation Results **47**
4.1 Chip Plan . . . 47
4.2 Post-layout Results . . . 49
4.3 Comparisons . . . 49
4.4 Application . . . 52

5 Conclusion and Future Work **55**
5.1 Conclusion . . . 55
5.2 Future Work . . . 56

Bibliography **57**

List of Figures

1.1 Performance of LDPC and NB-LDPC in high modulations 2

2.1 Tanner graph of **H** described in (2.2) 7
2.2 Permutation 9
2.3 Decoding flow 10
2.4 Check node updating unit 12
2.5 Variable node updating unit 12
2.6 Extended min-sum decoding algorithm 13
2.7 Candidate set 16
2.8 Pperformance curve 18

3.1 Proposed decoder architecture 21
3.2 Forward/Backward recursive structure 22
3.3 Candidate map 24
3.4 Updating process in conventional CES 25
3.5 Example of bubble check algorithm 26
3.6 Procedure of bubble check algorithm 28
3.7 Candidate choosing step 30
3.8 Performance comparison of different processing cycles 30
3.9 Candidate choosing flow 31
3.10 Region definition in M 32
3.11 Candidate choosing in region a 32
3.12 Performance comparison of defining regions 34
3.13 Comparison with other conventional algorithms 35
3.14 Conventional VES 37
3.15 Channel value calculation 38
3.16 Proposed VNU 39
3.17 Posterior probability computation 41
3.18 Decision circuit 41
3.19 Internal buffer controlling of ECU 42
3.20 Function unit of CES and VNU 43
3.21 Memory configuration 45
3.22 Memory collision problem 45

4.1 Chip plan 48
4.2 Components in the non-binary LDPC decoder 48
4.3 Post-layout photo 50
4.4 Performance curve 52
4.5 Performance curve in SC mode 53
4.6 Performance curve in HSI mode 54

List of Tables

1.1 Efficiency comparison 2

2.1 Representation of the elements in $GF(2^4)$ 5
2.2 Candidate set of the example in Figure 2.7 16
2.3 Complexity comparison in CNU 18
2.4 Complexity comparison in VNU 18

3.1 The relation between n_s and n_m 27
3.2 Comparison of the memory usage in channel values 40

4.1 Post-layout results 50
4.2 Comparison Table 51

Chapter 1

Introduction

1.1 Research Motivation

Error control code (ECC) plays an important role in many practical application related to the design for reliable digital transmission and storage system. ECC is applied in the channel encoder and channel decoder in the digital transmission system. In channel encoder, adding the redundant message called parity bits to the source information by the mathematical calculation. Then based on the arithmetic relationship between information bits and parity bits to detect and correct the errors caused by transmitting channel, and recover the message. Thus, ECC can efficiently resist the channel effect and provide reliable communication system.

Non-binary Low-Density-Parity-Check (NB-LDPC) codes are an extension of binary LDPC codes, were investigated by Davey and Mackay in 1998 [1]. It was shown that, non-binary LDPC codes can outperform than binary LDPC codes when the code length is small or applying to the higher-order modulation as shown in Figure 1.1. In Figure 1.1, it displays that non-binary LDPC codes can outperform binary LDPC codes especially in the higher modulations. Furthermore, non-binary LDPC can combat burst errors and further approach Shannon limit with good error floors [2] [3].

Although non-binary LDPC codes have so many advantages in decoding performance, the computational complexity and huge storage requirements are the considerable challenge to im-

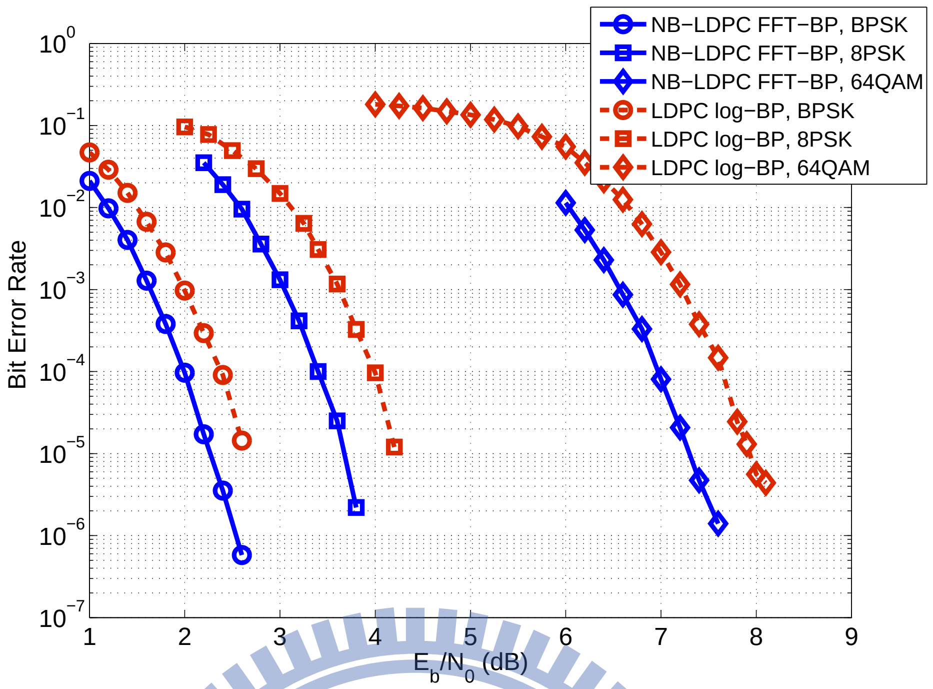


Figure 1.1: Bit error performance of (672,336) binary LDPC code with floating point log-BP and (112,56) non-binary LDPC codes over GF(2^6) with floating point FFT-BP. The simulation is performed under AWGN channel, and modulations are BPSK, 8PSK, and 64QAM. The maximum number of iterations is 50.

plement non-binary LDPC codes practically. In the state-of-the-art works, there are few designs can reach over 100 M bps in throughout and the gate counts all exceed 1 M as listed in Table 1.1. In order to enhance the hardware efficiency of non-binary LDPC codes decoder, we improve the throughput in CNU and VNU which are the main computation components in the decoder. In addition, we reduce the storage element of the edge messages and the channel values, and use layered scheduling to increase convergence speed.

Table 1.1: Hardware efficiency (throughput-to-gate-count-ratio) of existing designs

	[4] T-CASI-2012 Synthesis	[5] T-CASI-2011 Post-layout	[6] VLSI-2011 Synthesis	[7] T-CASI-2010 Synthesis
Throughput	31.2 Mb/s	47.69 Mb/s	10 Mb/s	60 Mb/s
Gate count	1.24 M	1.92 M	1.6 M	14.9 M
Hardware Efficiency (M bps/M gates)	50.4	24.84	6.25	8.06

1.2 Thesis Organization

This thesis is organized as follows. In Chapter 2, we introduce the concept of non-binary low-density parity-check block codes and discuss several conventional decoding algorithms for non-binary LDPC codes. Chapter 3 will first introduce the conventional function units of non-binary LDPC decoder in hardware implementation. Then, the details of proposed function units and decoder architecture are provided. Chapter 4 gives the implementation results and the comparison with other related designs. Finally, conclusions and future works are given in Chapter 5.

Chapter 2

Principle of Non-binary Low Density Parity Check Codes

Because of being the Shannon limit approaching codes, LDPC codes is an attractive solution in many communication and digital storage systems. Non-binary LDPC codes is an extension of binary LDPC codes, and it has excellent decoding ability and resist burst error. The more details about non-binary LDPC are prepared to discuss in section 2.2. In this chapter, we will first give the concept of the finite field. In addition, non-binary LDPC performance and its impact to system designs will be addressed. Finally, survey of available non-binary LDPC solutions will be briefly discussed.

2.1 Basic Introduction of Finite Field

A field contains only finitely many elements is called a Finite Field, or Galois Field. Like general field, finite field is also a framework for doing arithmetic, and it is closed under the operations defined like addition, subtraction, multiplication and division. Basic finite field can be constructed by a prime number p, and the field elements are $0, 1, ..., p-1$. In $GF(p)$, a non-zero element a is called a primitive element if $a^{p-1} = 1$, and the powers of a generate the overall element over $GF(p)$. Extending $GF(p)$ to $GF(p^m)$, the field with p^m elements is constructed as 0,1,α,...,α^{p^m-1} based on the modulo p arithmetic, and the coefficient in polynomial form is $0, 1, ..., p-1$. The elements defined over GF(2^p) are represented as binary sequence and have the advantage of easier arithmetic especially in addition. Therefore, binary finite field is widely used in several area like error control code (ECC) and cryptography.

Table 2.1 shows an example of the three kinds of representation for $GF(2^4)$ and the primitive polynomial is $f(x) = x^4 + x + 1$.

Table 2.1: Representation of the elements in $GF(2^4)$

Power	Polynomial	4-tuple
0	0	0 0 0 0
1	1	0 0 0 1
α	α	0 0 1 0
α^2	α^2	0 1 0 0
α^3	α^3	1 0 0 0
α^4	$\alpha + 1$	0 0 1 1
α^5	$\alpha^2 + \alpha$	0 1 1 0
α^6	$\alpha^3 + \alpha^2$	1 1 0 0
α^7	$\alpha^3 + \alpha + 1$	1 0 1 1
α^8	$\alpha^2 + 1$	0 1 0 1
α^9	$\alpha^3 + \alpha$	1 0 1 0
α^{10}	$\alpha^2 + \alpha + 1$	0 1 1 1
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	1 1 1 0
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	1 1 1 1
α^{13}	$\alpha^3 + \alpha^2 + 1$	1 1 0 1
α^{14}	$\alpha^3 + 1$	1 0 0 1

2.2 Non-binary LDPC Codes

In 1998 [1], Davey and MacKay extended the binary LDPC to the finite field $GF(q)$, and the non-zero entries in **H** are replaced by the elements in finite field. It was shown that, non-binary LDPC codes can perform better than binary LDPC codes when the code length is small or applying to the higher-order modulation. Furthermore, non-binary LDPC can combat burst errors and further approach Shannon limit with good error floors. In [8], they confirm that non-binary LDPC codes outperform both convolutional turbo codes and quasi-cyclic LDPC codes for all code rate, modulation and codeword lengths. In addition, non-binary LDPC codes can have good performance especially in more complex system, like MIMO and M-QAM channels [9]. In application, because of the good decoding ability and low throughput, non-binary LDPC codes are appealing solution for space/satellite communication systems [10] [11].

Even if non-binary LDPC codes have so excellent error correcting ability, but the relatively high complexity is always the main problem of the hardware implementation in practical. In order to figure out the efficient hardware implementation, several simplified versions from belief-propagation (BP) decoding algorithms are put forward in recent years. We will introduce some of them more detail in section 2.4.

2.3 Encoding of Non-binary LDPC Codes

The non-binary LDPC code is defined of the sparse parity check matrix **H**, and the construction of **H** are as follows. The matrix size of **H** is M rows by N columns, and (M,N) stand for the numbers of check equations (check nodes) and variable nodes respectively. In (2.1), the codeword c are defined as the null space of **H**, and the generator matrix **G** can be obtained by performing Gauss-Jordan elimination on **H**. The number of rows in **G** is represented as K, the length of information symbols. The entries of **H** are the elements which are defined in $GF(q)$, and (d_v,d_c) stand for the column and row degree (or variable node and check node degree) respectively. Note that the code is said regular if it has the same number of non-zero elements in all the columns and rows, otherwise it is called irregular code.

$$c = uG \quad , \quad GH^T = 0 \tag{2.1}$$

In (2.2) shows an example of (1,2) regular non-binary LDPC code over $GF(8)$. Note that the code with d_v=1 is not general, and it is for convenient to illustrate the example. The parity check matrix **H** can be represented as a graphic form, called Tanner graph to illustrate the connection between variable nodes and check nodes. Based on the **H** described in (2.2), the corresponding Tanner graph is depicted in 2.1. In Tanner graph, the connections between variable and check nodes represent the non-zero elements in **H**, and the permutation node stands for the corresponding symbol.

$$H = \begin{pmatrix} \alpha^3 & 0 & \alpha^5 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha^7 & 0 & 0 & 0 & \alpha^6 & 0 & 0 \\ 0 & 0 & 0 & \alpha^2 & 0 & 0 & \alpha^4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \alpha^5 \end{pmatrix} \tag{2.2}$$

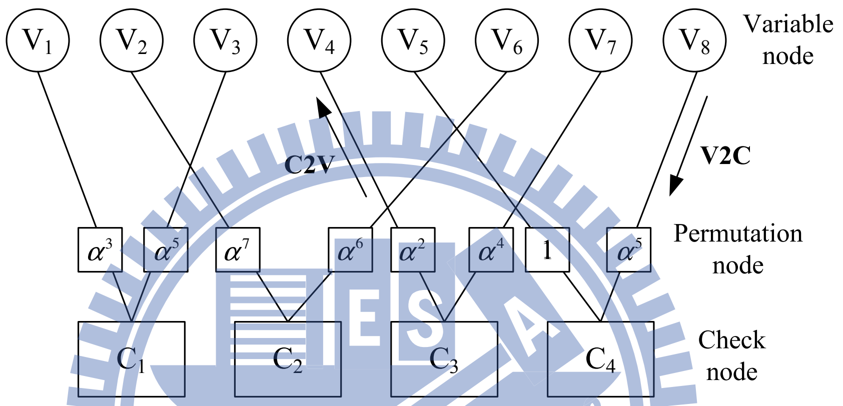


Figure 2.1: Tanner graph of **H** described in (2.2)

2.4 Decoding of Non-binary LDPC Codes

Like binary LDPC, the decoding of non-binary LDPC codes is based on belief-propagation (BP) algorithm, or sum-product-algorithm (SPA) which iteratively updates the posterior probabilities of each variable node. In non-binary case, because the non-zero entries in **H** are not equal to 1, the arithmetic in finite field is required. In [12], the author proposed FFT-SPA to perform in the frequency domain, which transfers the complicated convolution operation into simpler multiplications. In order to further reduce the computational complexity, the decoding algorithm can be transferred to the logarithm domain [13]. In log-domain algorithm, it requires fewer quantization bits for storing message, and it is more robust to the quantization effect [14].

SPA (or BP) and FFT-SPA are the decoding algorithms without performance loss, but the complicated computation and huge memory usage are very hard to be implemented in hardware. Therefore, the simplified versions with acceptable performance loss, like Extend Min-Sum (EMS) and Min-Max decoding algorithms are invented. In the following section, we will introduce SPA, EMS, and Min-Max decoding algorithms respectively. Except SPA, EMS and Min-Max are introduced in Log-Likelihood-Ratio (LLR) form. There are some comparison in complexity between several main decoding algorithms in non-binary LDPC codes, and it will be presented finally.

2.4.1 Sum of Product Algorithm (SPA)

In non-binary LDPC, the entries in **H** are defined in finite field, and the non-zero elements are not always equal to one. That is, the finite field multiplication is applied in check equations, and the single row check sum with degree d_c over GF(2^p) is shown in (2.3). Note that $v_i(x)$ represents the variable symbol in polynomial form, and $p(x)$ is the primitive polynomial.

$$\sum_{i=1}^{d_c} h_i(x)v_i(x) = 0 \mod p(x) \tag{2.3}$$

In non-binary LDPC codes, the operation of multiplying the non-zero element in **H** is called permutation, and it is not required in binary LDPC decoding. Since the arithmetic operation in finite field is closed, the multiplication is a one-to-one mapping process. If the elements represented in power form, multiplication is actually a cyclic shift of whole elements in the field as shown in Figure 2.2. This is the reason that multiplication of the non-zero element in **H** in non-binary LDPC decoding procedure is called permutation.

Algorithm 1 shows the overall SPA decoding procedure in non-binary LDPC codes, and it is similar with the binary one except permutation and inverse permutation.

In Algorithm 1, one decoding iteration requires permutation, check node update, inverse permutation, and variable node update these steps to accomplish. In the following, we will discuss these processing steps in more details.

First, we introduce the notations used in the decoding process. In non-binary LDPC codes,

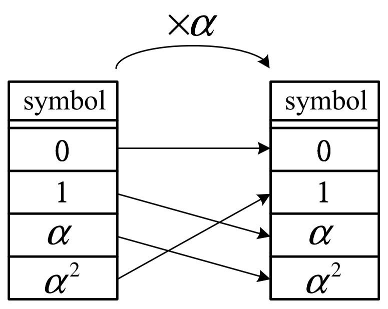


Figure 2.2: Finite field multiplication over GF(2^2) with power form

Algorithm 1: Decoding Procedure

```
   Input: The messages after adding the channel noises
   Output: Decoded symbols
1  Initialization:
2  Initialize the channel value as the V2C message in the first time
3  while syndrome is zero or the maximum iteration number is reached do
4      Permutation:
5      Multiply C2V messages by corresponding non-zero element in H
6      Check Node Update:
7      Compute C2V messages based on the check equations
8      Inverse Permutation:
9      Multiply the corresponding non-zero inverse element in H
10     Variable Node Update:
11     Compute V2C messages and the posterior probability of each variable to decide
       decoded symbols
12     Syndrome Computation:
13     Compute syndrome
14 end
```

the channel value and the message transmitted between nodes are all represented as a vector. The vector contains the possibility of each element defined in finite field, and the vector length depends on the size of the finite field. Note that in Figure 2.3, the two directions of message vector are represented as U(up) and D(down), and the direction from check node to variable node is U, and the other is D. The suffix of U and D stands for the incoming node and the outgoing node of this message vector. For example, D_{VC} means the message vector going from variable node to the check node. And $D_{VC}[i]$ stands for the possibility which the symbol of the element is equal to i. Furthermore, L_{V_j} represents the channel value vector of variable node V_j, and $L_{V_j}[i]$ stands for the possibility of channel value is equal to i. These notations are all depicted in Figure 2.3.

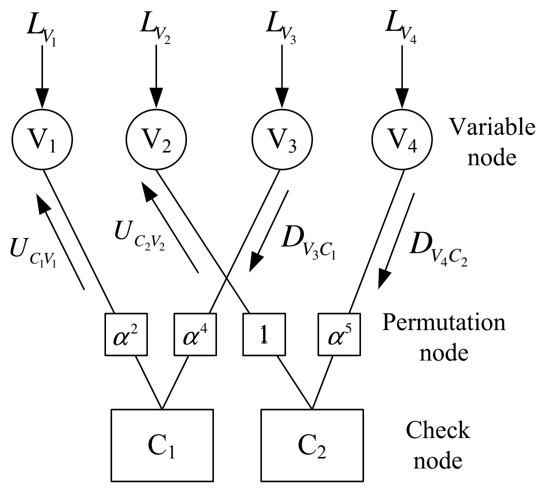


Figure 2.3: Notations used in non-binary LDPC decoding procedure

In the following, we will introduce these five decoding processes in the details.

1. Initialization

Using the information with the Gaussian additive noise from channel to calculate the probability of each element over GF(2^p) in each variable node. Note that the symbol likelihood value is denoted as $L[i_1, i_2, ..., i_p]$, and its polynomial form is

$$i(x) = (i_1, i_2, ..., i_p) = \sum_{k=1}^{p} i_k x^{k-1} \tag{2.4}$$

For example, $L[1, 0, 1]$ represents the possibility which the polynomial form of the element is $x^2 + 1$ over GF(2^3).

Define b_k is the k_{th} bit of the symbol over GF(2^p), and y_k is the result after adding the Gaussian noise n_k. After transmitting in the channel, the corresponding possibility of each symbol in variable node is calculated as

$$L[i_1, i_2, ..., i_p] = \prod_{k=1}^{p} l(i_k) \tag{2.5}$$

In (2.5), $l(i_k) = probability(y_k|b_k = i_k)$ and where $y_k = b_k + n_k$. Using the channel value vector L as the V2C message vector in the first iteration.

2. Permutation and inverse permutation

In non-binary LDPC codes, the non-zero elements in the parity check matrix **H** are not only equal to 1, so the multiplication in the finite field is required when operating the check equations. Furthermore, the multiplication with inverse element after doing the check node update is also needed. In particular, the multiplication become much easier if the symbol of elements are stored as power form. Then, the multiplication is just to do the addition of the power items, and match the concept of naming permutation.

3. **Check node update (CNU)**

   Using the incoming message D_{VC} to update the check node of degree d_c. According to the combination of elements in the input vectors, to find out the check-sum set that satisfies the check node equation such as

$$h_1 D_{V_1C} + h_2 D_{V_2C} + h_3 D_{V_3C} = 0 \tag{2.6}$$

   and calculate the summation of all the elements in this check-sum set. Assume the updating symbol in polynomial form is $i^{(a)}(x)$ of the m_{th} edge in Figure 2.4, and $U_{CV}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}]$ represents the possibility which the symbol in U_{CV} vector is equal to a. The updating equation is

$$\begin{aligned} & U_{CV_m}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] \\ & = \sum_{\{..,i^{(a)}(x),..| \sum_{l=1,l\neq m}^{d_c} i_l(x) = i^{(a)}(x)\}} \prod_{k=1,k\neq m}^{d_c} D_{V_kC}[i_1, i_2, ..., i_p] \end{aligned} \tag{2.7}$$

4. **Variable node update (VNU)**

   In Figure 2.5, using the incoming message $U_{CV}s$ and the channel values to compute the V2C messages ($D_{VC}s$) of degree d_v. The (2.8) describes the computing function for the case that updating symbol is a in the m_{th} edge of variable node.

$$\begin{aligned} & D_{VC_m}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] \\ & = L_V[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] \prod_{k=1,k\neq m}^{d_v} U_{C_kV}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] \end{aligned} \tag{2.8}$$

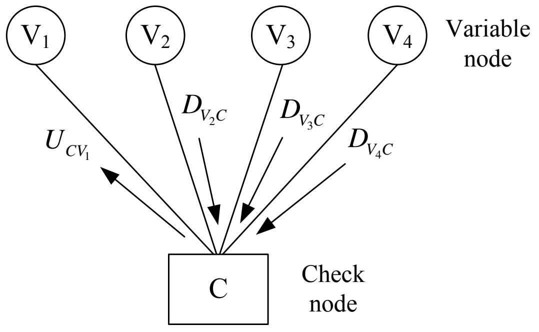


Figure 2.4: A CNU when d_c is 4 and compute the first edge($m = 1$)

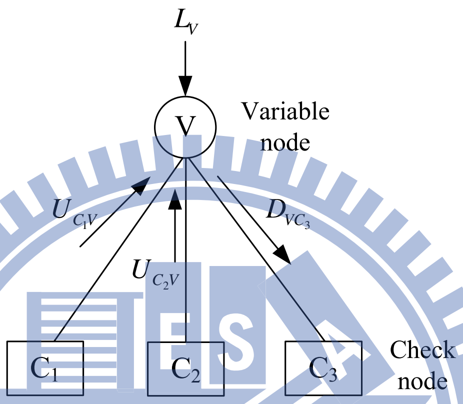


Figure 2.5: A VNU when d_v is 3 and compute the third edge($m = 3$)

Note that we need to normalize the possibilities in the message vector after doing VNU. Because the possibilities stored in SPA are the pdf forms, we need to ensure the summation of all possibilities is equal to 1. By means of normalization process, can prevent some possibilities to become very close to zero after several iteratively calculations.

5. **Decision Unit**

After an iteration every time, taking all the updated incoming messages of the variable node with the channel values to compute the posterior probabilities. Based on the q-ary probabilities, choose the largest one as the decoded symbol. According to the decoded symbols, the syndrome of each check equation is calculated. If the syndromes are not all zeros, the decoding process should be continued. But if there is no non-zero syndrome, the decoding process can be stopped even if the maximum iteration number is not reached called early termination.

2.4.2 Extended Min-Sum Algorithm (EMS)

The high computational complexity and the huge memory requirements are the main problems for non-binary LDPC to implement in practical. In [15] [16], Extended Min-Sum (EMS) algorithm is to simplify the CNU computation, and truncate the message vector from the original field size q to a limited number denoted as n_m. The n_m elements are selected according to the order of possibilities, so the message vectors need to store the first n_m largest possibilities and the corresponding symbols. Because of storing the incomplete messages, the compensated value γ is required to represent the possibility of the $(q - n_m)$ truncated elements. For simplicity, γ usually sets a constant value decided by performance simulation.

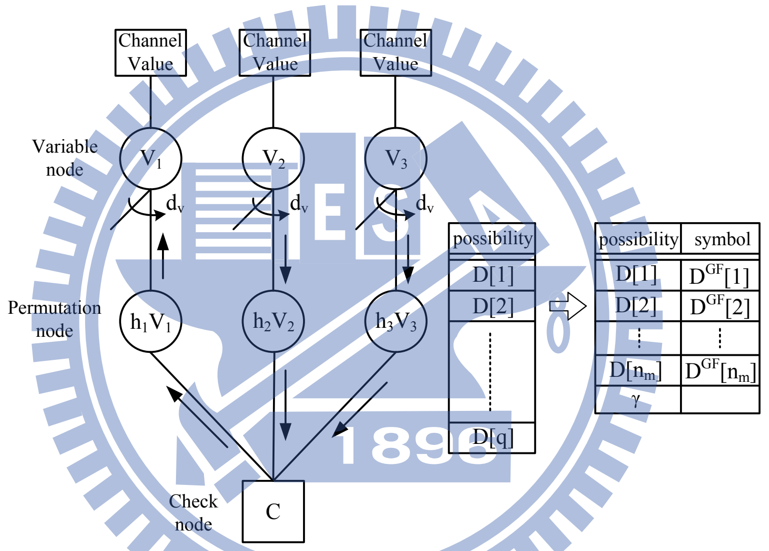


Figure 2.6: One check equation and edge message reduction in EMS decoding algorithm

Like SPA algorithm, we use U and D to represent the C2V and V2C message vectors respectively. Moreover, the notations for possibility and symbol of each element should be distinguished. For example, U is still represented as the vector which stores the possibilities, but $U[i]$ changes to stand for the i_{th} largest possibility (in SPA, $U[i]$ stands for the possibility of symbol i). And $U^{GF}[i]$ represents the corresponding symbol of $U[i]$. Note that the form of storing the possibilities can transfer to log-domain with Log-Likelihood-Ratio (LLR). It transform the complicated multiplication into simpler summations. In log-domain, the real-value addition transform to complicated operation related to logarithm, and EMS is to simplify it by

max function which is to find out the maximum among all inputs. Furthermore, it was shown that arithmetic operation in log-domain are more robust to quantization effect, and the required number of quantization bits are also smaller. Assume $P(a)$ is the possibility of symbol a, and the LLR form is calculated

$$U[k] = \log \frac{P(U^{GF}[k])}{P(U^{GF}[1])} \quad k \in \{1, 2, ..., n_m\} \tag{2.9}$$

The decoding process of EMS algorithm is similar to the SPA decoding algorithm introduced in section 2.4.1. The most different thing is the simplified version of CNU, and the elements in edge message vector truncate to a limited size n_m. In the following, the decoding function based on the truncated messages with LLR form will be introduced.

1. **Initialization**

   In general, the channel values are stored in complete field size in order to avoid performance loss. In the first iteration, sort out n_m elements with the first n_m largest possibilities be the V2C message vectors.

2. **Variable Node Unit**

   Assume that the variable node degree is d_v, and the updating function is

$$D_{VC_m}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] = L_V[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] + \sum_{k=1, k\neq m}^{d_v} U_{C_m V}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] \tag{2.10}$$

   In VNU, we need to combine the possibilities of identical symbols in $(d_v - 1)$ message vectors. Because symbols in the message vector are stored by the order of possibility, the storing of symbols is non-regular and incomplete. Therefore, the operation of searching the same symbol from each input vector is required. If the identical symbol does not exist, it will be replaced by the compensated value γ.

3. **Permutation and Inverse Permutation**

   In fact, the permutation and inverse permutation steps are just to do the multiplication in finite field, and the multiplicator is the non-zero element in parity check matrix **H**.

Because the set of symbols stored in message vector is incomplete, the set of symbols is changed after multiplying a symbol which is not equal to one. But for consistency, we still use ”permutation” to represent the multiplication with the non-zero elements in **H**.

4. **Check Node Unit**

   In SPA, we need multiplications and summations to accomplish a CNU. Using the LLR form, the real-value multiplication can transform to simpler summation, but the real-value summation will become more complicated. For example, the addition like $x_1 + x_2$ changes to $ln(e^{x_1} + e^{x_2})$. EMS is to simplify the summation related with logarithm to max operation according to (2.11).

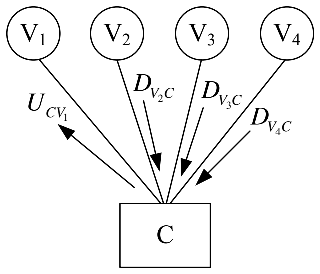
$$\ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}) \tag{2.11}$$

   Note that max operation is to find out the maximum from all inputs. Assume that the check node degree is d_c, the updating function is.

$$U_{CV_m}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] = \max_{a=\sum_{k=1,k\neq t}^{d_c} a_k} \left(\sum_{k=1,k\neq m}^{d_c} D_{V_mC}[i_1^{(a_k)}, i_2^{(a_k)}, ..., i_p^{(a_k)}] \right) \tag{2.12}$$

   Note that the output vector updated is still sorted in decreasing order. The function target of CNU is to sort out the first n_m largest possibilities from the candidate set. The example of constructing a candidate set is illustrated in Figure 2.7 and Table 2.2. If the check node degree is d_c, the size of the set is equal to $n_m^{(d_c-1)}$. Sorting from this huge set is too complex to implement in practical. For this reason, in non-binary LDPC codes, we usually apply the approach like divide and conquer to simplify an updating function step by step [13].

In EMS algorithm, n_m is a significant factor, and its size directly affects the performance on hardware efficiency and decoding ability. It is a trade-off between hardware cost and decoding performance. Deciding the value of n_m and figuring out a cost-performance balance is the most important issue when implementing non-binary LDPC decoder in practical. In general, the



	$D_{V_2C}^{GF}$
$D_{V_2C}[1]$	$D_2(010)$
$D_{V_2C}[2]$	$D_2(100)$
$D_{V_2C}[3]$	$D_2(011)$
$D_{V_2C}[4]$	$D_2(000)$
	γ_2

	$D_{V_3C}^{GF}$
$D_{V_3C}[1]$	$D_3(110)$
$D_{V_3C}[2]$	$D_3(111)$
$D_{V_3C}[3]$	$D_3(010)$
$D_{V_3C}[4]$	$D_3(101)$
	γ_3

	$D_{V_4C}^{GF}$
$D_{V_4C}[1]$	$D_4(000)$
$D_{V_4C}[2]$	$D_4(001)$
$D_{V_4C}[3]$	$D_4(100)$
$D_{V_4C}[4]$	$D_4(111)$
	γ_4

Figure 2.7: Update the first edge message U_{CV_1} when $d_c = 4$ and $n_m = 4$

Table 2.2: Candidate set of the example in Figure 2.7

$U_1(000)$	$U_1(001)$	.....
$D_2(010) + D_3(110) + D_4(100)$	$D_2(010) + D_3(111) + D_4(100)$	.....
$D_2(010) + D_3(010) + D_4(000)$	$D_2(010) + D_3(010) + D_4(001)$	.....
$D_2(010) + D_3(101) + D_4(111)$	$D_2(100) + D_3(010) + D_4(111)$	.....
$D_2(100) + D_3(101) + D_4(001)$	$D_2(100) + D_3(101) + D_4(000)$	.....
$D_2(011) + D_3(111) + D_4(100)$	$D_2(011) + D_3(110) + D_4(100)$	.....
$D_2(011) + D_3(010) + D_4(001)$	$D_2(011) + D_3(010) + D_4(000)$	.....
$D_2(000) + D_3(111) + D_4(111)$	$D_2(011) + D_3(101) + D_4(111)$	.....
	$D_2(000) + D_3(110) + D_4(111)$	.....
	$D_2(000) + D_3(101) + D_4(100)$	.....

performance loss in EMS is the least ($<$ 0.1 dB) in all simplified non-binary LDPC decoding algorithms.

2.4.3 Min-Max Algorithm

In [17], Min-Max decoding algorithm is further simplifier than EMS decoding algorithm in check node processing. Based on the concept that the largest value dominates the result in addition, Min-Max replace the summation to max operation to reduce the computational complexity. The main differences in decoding process from EMS are initialization and CNU, so we describe these two steps in more detail as follows. Note that we use the complete field size without considering n_m for convenient in presenting the notations.

1. **Initialization**

   The priori information is calculated by

$$D_i(a) = \ln(P(x_i = s_i|channel)/P(x_i = a|channel))$$
$$, where\ s_i\ is\ the\ most\ likely\ symbol\ for\ x_i \tag{2.13}$$

Note that $P(x = a|channel)$ stands for the probability that the symbol representation of x is equal to a after adding the channel effect. The possibilities in message vector are initialized by (2.13), and the smaller value represents the higher possibility on the contrary.

2. **Check Node Update (CNU)**

   Compared with the CNU computation in EMS, Min-Max replaces the summation to max operation in order to further simplify the updating function. Because of the overestimation in CNU, Min-Max is a little sub-optimal than EMS algorithm. The updating function in CNU is described by

$$U_{CV_m}[i_1^{(a)}, i_2^{(a)}, ..., i_p^{(a)}] = \min_{a=\sum_{k=1,k\neq m}^{d_c} a_k} (\max(D_{V_mC}[i_1^{(a_k)}, i_2^{(a_k)}, ..., i_p^{(a_k)}])) \tag{2.14}$$

   In (2.14), the min and max functions are to find out the minimum and maximum among all the inputs respectively. Because the smaller value stored in Min-Max decoding algorithm stands for the higher possibility, the CNU use min function to choose the most likely result.

For the reason of simplicity, almost non-binary LDPC decoder are implemented by Min-Max algorithm, and there are many improved Min-Max decoding algorithms are presented [7] [6] . In the following, the complexity in CNU and VNU with different decoding algorithms discussed in previous sections are presented [18] [19] [8], and the decoding performance is depicted in Figure 2.8. From the performance curve in Figure 2.8, the EMS is very close to FFT-SPA which is no performance loss, and outperforms than Min-Max about 0.2 dB. Supposing that it is defined over GF(q), and n_t stands for the required clock cycles to accomplish a CNU with degree 2. Note that except SPA, EMS and Min-Max are applied in LLR form and use the truncated number n_m.

Table 2.3: Number of operations of check node updating function ($d_c = 2$) for different decoding algorithms

	Multiplications	Max/Min	Additions (real)	Additions (GF(q))
SPA	q^2	0	$q(q-1)$	0
EMS	0	$n_t n_m$	$n_t + n_m$	$n_t + n_m$
Min-Max	0	$n_t n_m + n_t + n_m$	0	$n_t + n_m$

Table 2.4: Number of operations of variable node updating function ($d_v = 2$) for different decoding algorithms

	Multiplications	Divisions	Max	Additions (real)
SPA	q	q	0	$q-1$
EMS	0	0	$n_m(n_m+2)$	n_m
Min-Max	0	0	$n_m(n_m+2)$	n_m

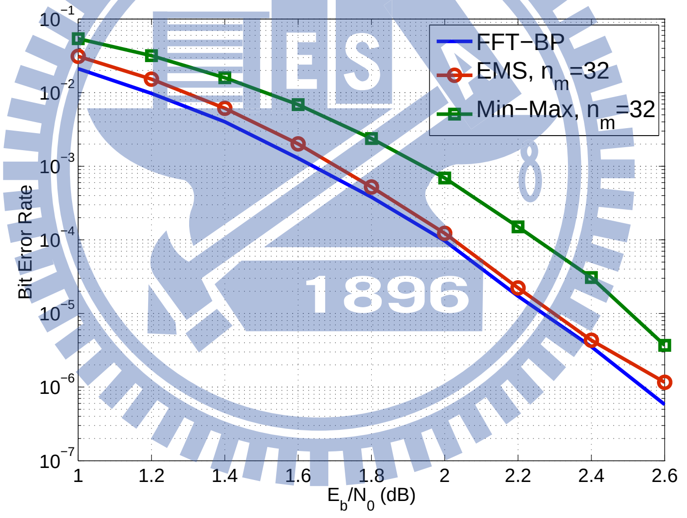


Figure 2.8: Performance simulation for a (112,56) non-binary LDPC code over GF(2^6) with FFT-BP, EMS, and Min-Max decoding algorithms. The simulation is performed under BPSK modulation and AWGN channel. The maximum number of iterations is 50.

Chapter 3

Non-binary LDPC Decoder Architecture

In this chapter, we will first show the overall proposed decoder architecture. Then, the conventional approach for implementing each main function component in non-binary LDPC decoder with EMS algorithm [20] will be introduced. Following this, the proposed function units of the decoder are presented.

3.1 Non-binary Quasi-Cyclic LDPC Codes

Because of the code regularity and special code structure, QC-LDPC codes are an appealing solution for VLSI implementation. Furthermore, QC-LDPC codes have good decoding performance and relatively low error floor [21]. In our design, we also choose (2,4)-regular 64-ary non-binary QC-LDPC code to implement. Note that (2,4) stands for the column and row degrees respectively.

3.1.1 Code Structure

Quasi-cyclic code is composed of r by r sub-matrix with cyclic shift to identity matrix I_0, and the subscript of each I denotes the times of shift. With the feature of regularity in quasi-cyclic code, the complexity of decoder implementation can be simplified. Note that the block row stands for the r rows grouped by sub-matrices, and the same as block columns.

The non-binary QC-LDPC code used in our design is described in (3.1)

$$H = \begin{pmatrix} 0 & 0 & 0 & I_5 & 0 & 0 & I_0 & 0 & 0 & 0 & I_0 & I_1 & 0 & 0 \\ 0 & 0 & I_4 & 0 & 0 & 0 & I_4 & 0 & 0 & I_4 & 0 & 0 & I_5 & 0 \\ 0 & I_7 & 0 & 0 & 0 & I_7 & 0 & 0 & 0 & I_7 & 0 & 0 & 0 & I_0 \\ 0 & 0 & I_0 & 0 & 0 & I_1 & 0 & 0 & I_6 & 0 & 0 & 0 & 0 & I_6 \\ I_2 & 0 & 0 & 0 & I_1 & 0 & 0 & 0 & I_1 & 0 & 0 & 0 & I_2 & 0 \\ I_4 & 0 & 0 & I_7 & 0 & 0 & 0 & I_7 & 0 & 0 & 0 & I_7 & 0 & 0 \\ 0 & I_1 & 0 & 0 & I_6 & 0 & 0 & I_3 & 0 & 0 & I_3 & 0 & 0 & 0 \end{pmatrix} \tag{3.1}$$

Our code is transformed by a binary LDPC code, and replace the non-zero elements from 1 to the symbol over GF(q) randomly. This binary LDPC code is built from the CP-PEG algorithm presented in [22]. The code is a regular QC code, and it is suitable for layered decoding algorithm. In addition, it was shown in [23], the "ultra-sparse" codes ($d_v = 2$) have excellent error-correcting performance especially in high order finite field ($q \geq 64$). Because of the minimum connectivity in variable nodes, a class of regular (2,d_c) codes can simplify the decoder complexity in hardware implementation. Furthermore, we figure out several improved approaches in edge memory reduction and efficient VNU processing based on this property. And this code can be separated into two sub-blocks from the middle in matrix, because these two sub-blocks contain the same number of non-zero sub-matrices in each block row. For this reason, if we want to increase the memory banks for improving bandwidth, it is a convenient property for configuring the memories.

3.2 Decoder Architecture

Figure 3.1 shows the overall decoder architecture, and it is mainly composed of several Processing Elements (PEs), storage elements , and control circuits. A PE is composed of one

CNU and its associated VNUs, and it is the main computation unit in our proposed design. Furthermore, N_p stands for the number of parallelism when implementing the decoder, and the number of N_p is directly affect the throughput and hardware cost. Note that the dashed line is the extra bypass path to solve the memory collision problem which will be discussed in 3.6.3.1.

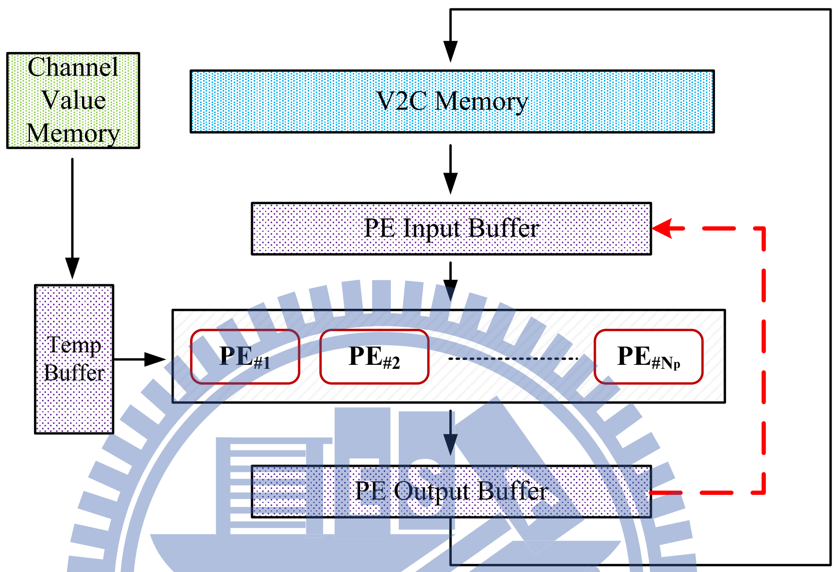


Figure 3.1: Proposed decoder architecture for non-binary QC-LDPC codes

3.3 Forward and Backward Algorithm

In non-binary LDPC codes, the complexity of updating function grows in exponential according to the number of node degrees. For this reason, using the efficient approach called forward and backward algorithm [24], which is a recursive structure like the concept of divide and conquer to accomplish the updating function step by step. First, the updating function is decomposed by several basic functions called elementary steps, and then executing the elementary steps recursively to accomplish an updating function. An elementary step is defined as an updating function composed of only 2 input vectors and 1 output vector. In general, if the node degree is equal to d, it requires $3(d-2)$ elementary steps to finish a node updating function. Figure 3.2 illustrates the required elementary steps and the corresponding forward/backward recursive structure when the check node degree d_c is equal to 4.

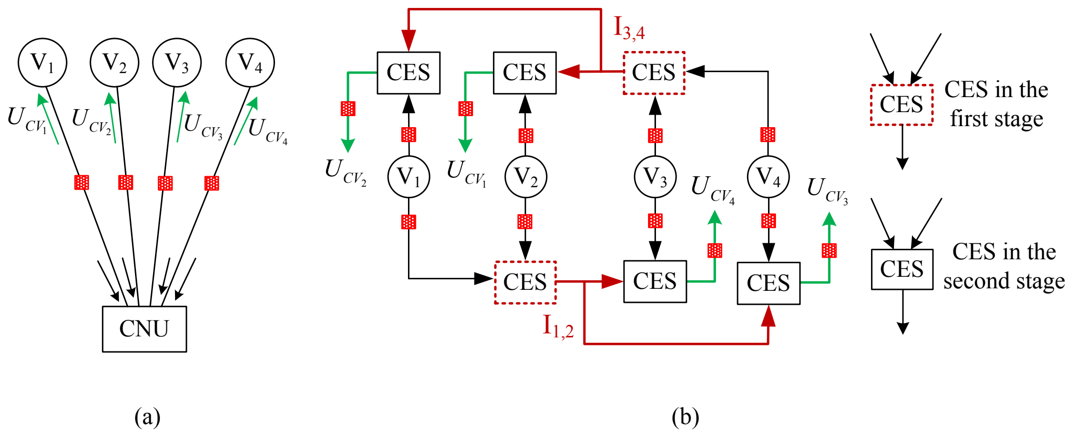


Figure 3.2: (a) A CNU with $d_c = 4$ (b) Forward/Backward recursive structure with $d_c = 4$

The following describe a check node ($d_c = 4$) updating procedure with the forward/backward algorithm. At the first stage, using 2 Check Elementary Steps (CES) to compute 2 internal vectors, $I_{1,2}$ and $I_{3,4}$. Note that $I_{1,2}$ stands for the internal message vector containing the information from input vectors V_1 and V_2. At the second stage, the 4 output vectors (U_{CV_1},U_{CV_2},U_{CV_3},U_{CV_4}) are computed from the combination of each input vector and the 2 internal results calculated in the first stage. Therefore, in the case that d_c is equal to 4, we need $3(4-2) = 6$ elementary steps and 2 stages to accomplish a check node updating function.

3.4 Check Node Unit (CNU)

Check node unit (CNU) is the most complicated component in non-binary LDPC decoding procedure, and it is usually the bottleneck when implementing the decoder in practical. Based on the forward and backward recursive structure, a CNU is decomposed of several check elementary steps (CESs) to implement. In this section, we first introduce the conventional approach in processing a CES, and then an efficient algorithm [25] especially target on simplifying the CES in EMS decoding algorithm is addressed. Following this, we will present our proposed CES architecture for improving the throughput without extra decoding performance loss.

3.4.1 Check Elementary Step (CES)

Check elementary step (CES) is an updating function of degree 2, and it is assumed that the input vectors already finish the permutation step. Suppose the vector size is n_m, and the notations of input and output vectors are I_1, I_2, and O respectively. Note that the possibilities in each message vector are represented as log-likelihood-ratio (LLR) and sorted in decreasing order. Define a candidate set S_c, and the elements in S_c are the combinations from the input vectors which satisfy the equation $I_1^{GF}[i] \oplus I_2^{GF}[j] = O^{GF}[k]$ and the corresponding possibility is $I_1[i] + I_2[j] = O[k]$. Then, the goal of CES is to explore the non-repeating symbols with the first n_m largest possibilities from S_c. The equation of CES to compute the k_{th} largest possibility is

$$O[k] = \max_{\substack{I_1^{GF}[i] \oplus I_2^{GF}[j] = O^{GF}[k] \\ i,j \in [1,n_m]^2}} \{I_1[i] + I_2[j]\} \tag{3.2}$$

There are n_m^2 elements in candidate set S_c, and exploring in whole S_c is very complicated. Using the sorted property in each message vector, the more efficient way to realize a CES is to search elements from the graphic form [24]. Define a candidate map called M as shown in Figure 3.3, and M displays the possibilities distribution of the n_m^2 elements in S_c. In order to reduce the size of S_c, the author in [24] proposed a "sorter" which stores n_m candidates for exploring at a time, and then the size of S_c changes from n_m^2 to n_m. Note that candidate set S_c is redefined as the elements in the sorter, and it updates every cycle.

Based on the candidate map M and a sorter of size n_m, the processing operations of CES using graphic approach are as follows [20]. Note that the meaning of sorting used in EMS algorithm is to insert one element into a sorted sequence denoted as sorter.

1. **Initialization**

   Insert n_m elements in the first column of M into the sorter.

2. **Output**

   Output an element with the largest possibility in the sorter.

3. **Check**

S / P	24 / 20.3	8 / 17.1	3 / 15	15 / 13.4	7 / 11.1
1 / 13.2	25 / 33.5	9 / 30.3	2 / 28.2	14 / 26.6	6 / 24.3
16 / 11.7	8 / 32	24 / 28.8	19 / 26.7	31 / 25.1	23 / 22.8
4 / 10.1	28 / 30.4	12 / 27.2	7 / 25.1	11 / 23.5	3 / 21.2
2 / 7.9	26 / 28.2	10 / 25	1 / 22.9	13 / 21.3	5 / 19
8 / 6	16 / 26.3	0 / 23.1	11 / 21	7 / 19.4	15 / 17.1

Figure 3.3: Candidate map M, (S,P) stands for symbol and possibility respectively

Check whether the symbol is redundant. If this symbol is already in the output vector, discard this symbol. This operation is implemented by p bits comparator circuit when the symbol is defined over $GF(2^p)$.

4. **Candidate Choose**

   Choose the right side symbol of the output symbol as the candidate inserting to the sorter.

5. **Sort**

   According to the possibility, insert the candidate symbol into the sorted sequence in the sorter.

Repeat step 2 to step 5 till output vector is full or the predetermined processing cycles is reached. The operations are described in Algorithm 2, and Figure 3.4 illustrates a CES procedure.

If message vector contains the repeated symbols, it means that the valid number of n_m becomes smaller. After iteratively decoding process, the valid symbols will be fewer and fewer and result in the significant decoding performance loss. Therefore, the operation of symbol checking is necessary, and the repeated symbols should be discarded.

```
Algorithm 2: Check Elementary Step
  Input: $I_1$ and $I_2$ (input vectors)
  Output: $O$ (output vector)
  Data: $M$ (candidate map, [row,column]), $S$ (sorter size $n_m$), $t_{pre}$ (predetermined cycles)
  Initialization:
  forall the $i$ such that $n_m \leq i \leq 1$ do
      $S[i] \leftarrow M[i,1]$
  end
  $i = 1, n = 1$
  while $n < n_m$ or $i < t_{pre}$ do
      Output:
      $S_{max} \leftarrow S[1]$, and $S[1] = M[r,c]$
      Check:
      if $S_{max}^{GF} \notin O^{GF}$ then
          $O[n] = S_{max}$
          $O^{GF}[n] = S_{max}^{GF}$
          $n = n + 1$
      end
      Candidate Choose:
      $Candidate \leftarrow M[r, c+1]$
      Sort:
      forall the $j$ such that $n_m \leq j \leq 1$ do
          $Diff[j] = Candidate - S[j]$
      end
      if $Diff[k] > 0$ and $Diff[k-1] \leq 0$ then
          forall the $j$ such that $n_m \leq j \leq k+1$ do
              $S[j] \leftarrow S[j-1]$
              $S^{GF}[j] \leftarrow S^{GF}[j-1]$
          end
          $S[k] \leftarrow Candidate$
          $S^{GF}[k] \leftarrow Candidate^{GF}$
      end
      $i = i + 1$
  end
```

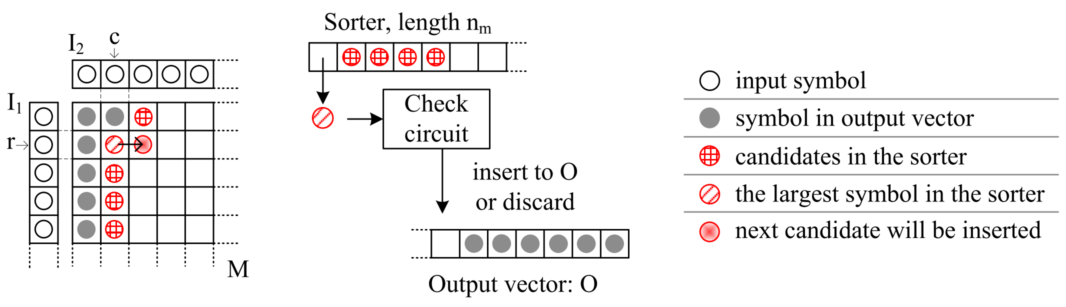


Figure 3.4: Updating process of conventional CES.

3.4.2 Bubble Check Algorithm

In [24], the author proposed a method to reduce the number of elements when searching the largest possibility, and the size of candidate set S_c is smaller than n_m. Every clock cycle, the operation of a CES is to find out an element with the largest possibility from the sorter. For this reason, it just needs to ensure that the element with the largest possibility is in the sorter in every clock cycle. Bubble check algorithm [25] uses this property to efficiently reduce the number of elements in S_c at one time. It means that only the element which probably has the largest possibility at that time will be considered, and this is based on the regular distributing property of possibilities in M.

We use an example to describe the basic concept of bubble check algorithm. Assuming that there are already 4 elements in the output vector, and then we want to find next element with the fifth largest possibility. In Figure 3.5, it illustrates all possible distributions of the output symbols and their corresponding candidates which should be considered. Note that white circle represents the element in the output vector, and the black circle stands for the candidate which will be inserted to the sorter. The number in the white circle stands for the order in possibilities of output symbols.

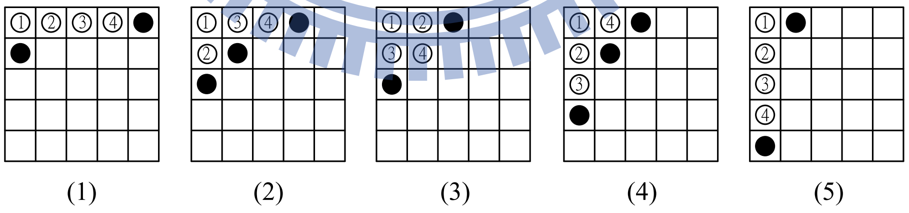


Figure 3.5: 5 possible conditions when finding the 5_{th} largest possibility.

According to the possibilities stored in input vectors are in decreasing order, the larger possibilities are centralized in the upper left of M. Based on the regularity in M, candidates choosing are decided from the right or down side of the output symbols. Because bubble check algorithm only considers the possible candidates with the largest possibility, we do not need to take account of more than one elements in the same row or the same column. Therefore, the minimum required number of candidates depends on the distributing shape of the output symbols. In the second and forth graphs, when the distributing shape is close to triangular, the number of possible candidates is more. Looking into other graphs, the number of candidates is the least when the distributing shape of the output symbols is rectangular. After considering these five situations, we can infer that the minimum required sorter size with n_m 5 should be 3.

Therefore, the minimum required sorter size depends on the distributing shape of elements in the output vector, and the worst case is when the distributing shape is triangular. Without considering the symbol repetition problem, the sorter size n_s is calculated by supposing there are already $(n_m - 1)$ symbols in the output vector and the distributing shape of these symbols is triangular. The relation between n_m and n_s is described by

$$\begin{aligned} \tfrac{(1+n_s)n_s}{2} &= (n_m - 1) + n_s \\ \Rightarrow n_s &= \left\lceil \tfrac{1+\sqrt{1+8(n_m-1)}}{2} \right\rceil \end{aligned} \tag{3.3}$$

Table 3.1: The relation between n_s and n_m

n_m	4	8	16	32	64
n_s	3	5	6	9	12

Based on (3.3), the number of n_s with different n_m are listed in Table 3.1. The process of the CES applying the bubble check algorithm is similar with conventional CES mentioned above, and the main difference is the smaller sorter size n_s and the way of choosing the candidate. The procedures are illustrated in Figure 3.6 and the operations are described as follows:

1. **Initialization**

   Insert n_s symbols in the first column of M to the sorter.

2. **Output**

   Output the symbol with the largest possibility in the sorter.

3. **Check**

   Check whether the symbol is redundant. If this symbol is already in the output vector, discard this symbol.

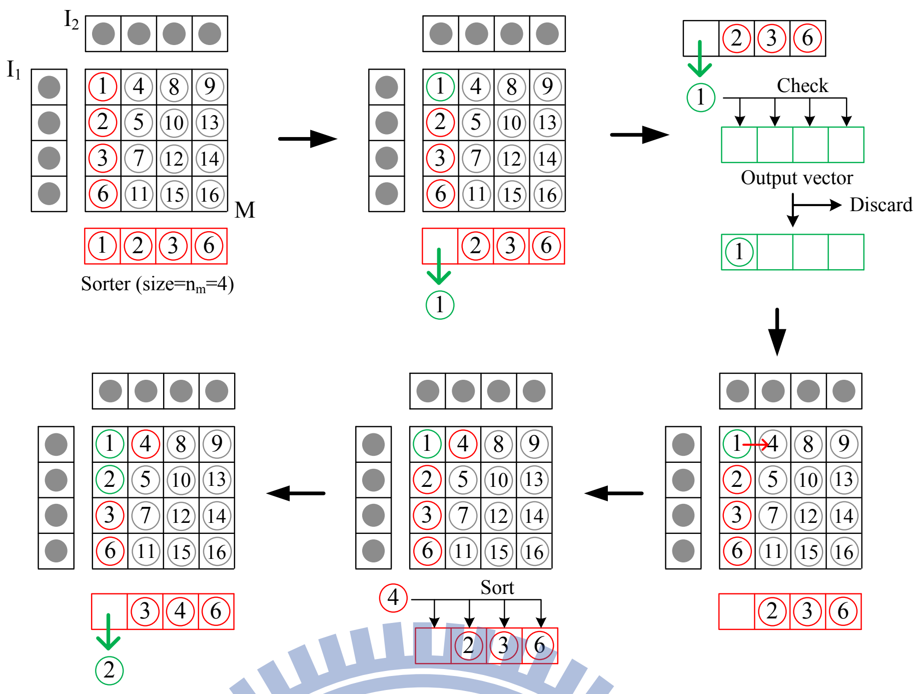


Figure 3.6: Procedure of bubble check algorithm in the beginning.

4. **Candidate choose**

   (a) If the output symbol is in the first column/row, choose its down/right side symbol as the candidate. (b) If not, maintain the same direction with last clock cycle.

5. **Sort**

   According to the possibility, insert the candidate symbol into the sorted sequence which is reduced to n_s.

Repeat (2) to (5) till output vector is full or the predetermined processing cycles is reached. The overall processing steps are described in Algorithm 3.

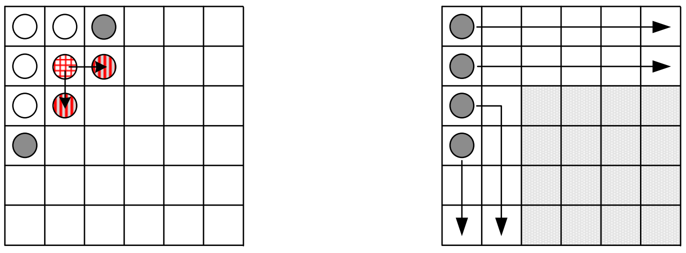
In bubble check algorithm, it needs to decide the candidate from right or down side of the element with the largest possibility as shown in Figure 3.7(a). This increases some complexity in the controlling circuit when choosing candidate. In order to simplify the controlling circuit, L-bubble check [26] is to determine the paths of choosing next candidate in advance. There is an example when n_s is equal to 4 in Figure 3.7(b), and the elements in dark zone will not be considered.

Using the regularity of candidate map M, bubble check algorithm can efficiently reduce the

Algorithm 3: Check Elementary Step with Bubble Check Algorithm

```
Input: $I_1$ and $I_2$ (input vectors)
Output: $O$ (output vector)
Data: $M$ (candidate map), $S$ (sorter size $n_s$), flag (the direction of candidate choosing)
Initialization:
forall the $i$ such that $n_s \leq i \leq 1$ do
    $S[i] \leftarrow M[i,1]$
end
$i = 1, n = 1$
while $n < n_m$ do
    Output:
    $S_{max} \leftarrow S[1]$, and $S[1] = M[r,c]$
    Check:
    if $S_{max}^{GF} \notin O^{GF}$ then
        $O[n] = S_{max}$
        $O^{GF}[n] = S_{max}^{GF}$
        $n = n + 1$
    end
    Candidate Choose:
    if $r = 1$ then
        $flag \leftarrow 0$
    else if $c = 1$ then
        $flag \leftarrow 1$
    else if $M[r + flag, c + \bar{flag}] \in S$ then
        $flag \leftarrow \bar{flag}$
    else
        $flag \leftarrow flag$
    end
    $Candidate \leftarrow M[r + flag, c + \bar{flag}]$
    Sort:
    forall the $j$ such that $n_s \leq j \leq 1$ do
        $Diff[j] = Candidate - S[j]$
    end
    if $Diff[k] > 0$ and $Diff[k-1] \leq 0$ then
        forall the $j$ such that $n_m \leq j \leq k + 1$ do
            $S[j] \leftarrow S[j-1]$
            $S^{GF}[j] \leftarrow S^{GF}[j-1]$
        end
        $S[k] \leftarrow Candidate$
        $S^{GF}[k] \leftarrow Candidate^{GF}$
    end
    $i = i + 1$
end
```

number of candidates and have $\sqrt{n_m}$ complexity reduction [25]. But like traditional CES, it still has symbol repetition problem and requires more than n_m processing cycles to fill up the



(a) (b)

Figure 3.7: (a) The illustration of choosing the candidate from right or down direction (b)Example of predetermined path for $n_s = 4$ in L-bubble check algorithm

output vector. In average, the processing time of a CES is equal to $2n_m$ cycles for avoiding the performance loss as shown in Figure 3.8. For this reason, if we want to improve the throughput, the processing cycles in a CES should be reduced.

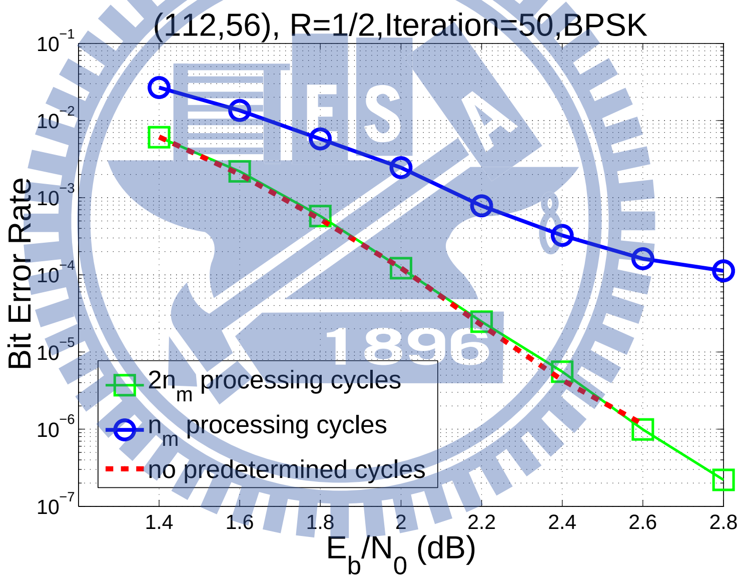


Figure 3.8: Performance curve when the processing cycles is decided as n_m and $2n_m$. No predetermined cycles stands for the case that the CES stop computing until output vector is full.

3.4.3 Proposed Check Elementary Step

As mentioned in last section, $2n_m$ clock cycles are required in processing the CES because of the repeated symbols. There is still no efficient way to filter out the redundant symbols in candidate choosing step, so 2 times n_m processing cycles in CES is unavoidable. For this reason, the proposed method of improving the throughput in CES is directly to double the output symbol at a time.

3.4.3.1 Proposed Double Throughput Bubble Check Algorithm

According to the target of double throughput, we modify the approach in candidate choosing step and output 2 symbols at a time. Supposing that (n_m, n_s) is (7,5), we introduce the candidates choosing procedure with Figure 3.9, and discuss in the followings.

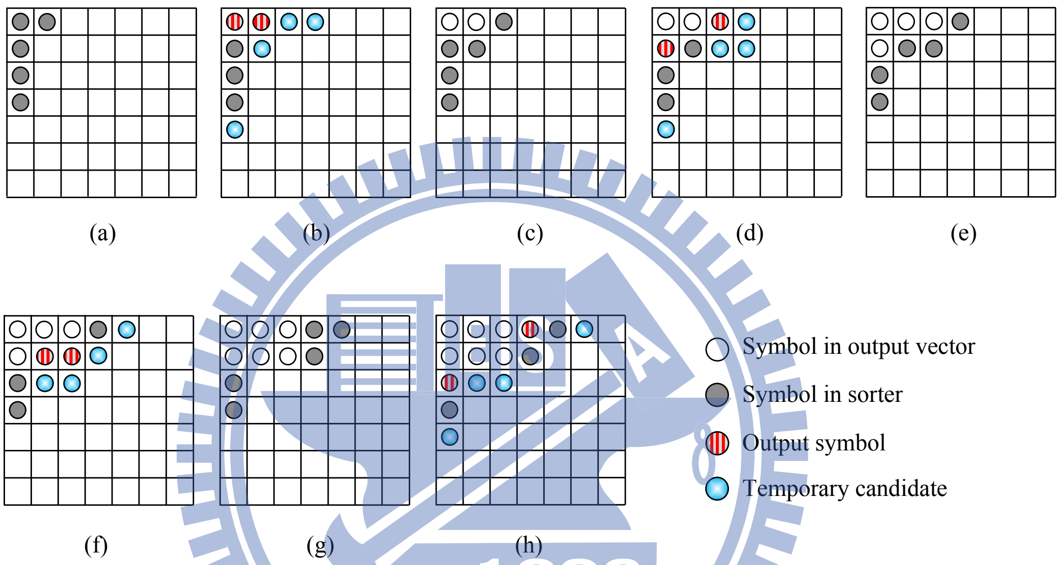


Figure 3.9: Candidate choosing procedure of double throughput bubble check, (n_m, n_s) = (7,5)

In original bubble check algorithm, it guarantees that the sorter contains the element with the largest possibility. But in our proposed method, the sorter output 2 elements at a time, so the first 2 largest possibilities should be considered. For this reason, the distribution for initialization is a little different from conventional one, and it is depicted in (a). The next 2 candidates are chosen among 4 temporary candidates, so we need to decide 4 temporary candidates first. These 4 temporary candidates are the right and down side symbols of the 2 output symbols. It is needed to check that these 4 temporary candidates should be the element with the largest possibility in its row or column as illustrated in (f) and (h). From this example, it is too complicated in controlling the direction of deciding the next 2 candidates. Therefore, we combine the L-bubble check algorithm to simplify the controlling circuit.

3.4.3.2 Proposed Double Throughput L-bubble Check Algorithm

In general, the paths for choosing candidates are determined after performance simulation, and we first define 2 regions in M for describing the procedure easily. In Figure 3.10, the darked zone in the first row and first column is denoted as region a, and the rest predetermined paths is region b. In our proposed method, if the output symbol is in region a, we need extra computation to decide the candidate.

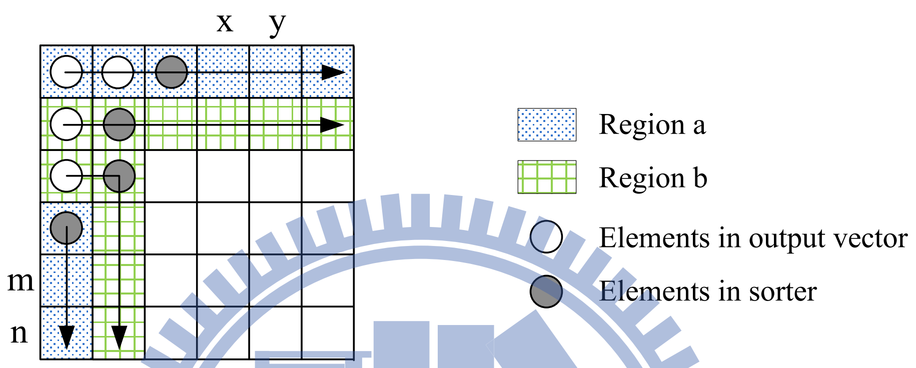


Figure 3.10: Candidate map M used in proposed CES

We use an example illustrated in Figure 3.11 to describe the procedure which the output symbol is in the region a. Every time, using 4 possibilities (x, y, m, n) and 3 comparators to determine the first 2 largest possibilities in region a, and we denote 1 and 2 in the circles to represent them. Then, depends on the number of output symbols in the region a to decide the next candidate. And Figure 3.12 shows the performance loss when directly applying the L-bubble check algorithm without separating the regions.

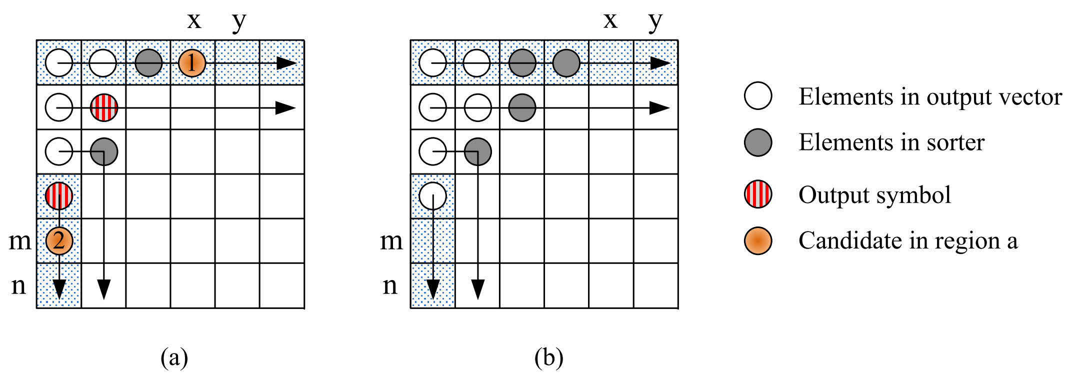


Figure 3.11: When output symbol is in region a, choose the larger one be the next candidate.

Algorithm 4: Candidates Choose in CES with Proposed Double Throughout L-Bubble Check Algorithm

```
Input: $S_1$ and $S_2$ (output elements with the first 2 largest possibilities), $M[r_1, c_1]$ and
        $M[r_2, c_2]$ (the corresponding position in $M$)
Output: $C_1$ and $C_2$ (the candidates prepared to insert to the sorter)
Data: Region a: $path_1$ records the position for current symbol in the sorter of the first
      row, $path_{n_s}$ records the position for current symbol in the sorter of the first
      column. Region b: ($L_r$,$L_c$) represents the predetermined path
Extra 2 Candidates in Region a, $Ca_1$ and $Ca_2$ ($x, y, m, n$ are defined as Figure 3.11):
$p_1 = x - m, p_2 = x - n, p_3 = m - y$
if $[sign(p_1), sign(p_2), sign(p_3)] = [0, 0, 1]$ then
    $Ca_1 \leftarrow M[1, c(path_1) + 1], Ca_2 \leftarrow M[1, c(path_1) + 2]$
else if $[sign(p_1), sign(p_2), sign(p_3)] = [0, 0, 0]$ then
    $Ca_1 \leftarrow M[1, c(path_1) + 1], Ca_2 \leftarrow M[r(path_{n_s}) + 1, 1]$
else if $[sign(p_1), sign(p_2), sign(p_3)] = [1, 1, 0]$ then
    $Ca_1 \leftarrow M[r(path_{n_s}) + 1, 1], Ca_2 \leftarrow M[r(path_{n_s}) + 2, 1]$
else
    $Ca_1 \leftarrow M[r(path_{n_s}) + 1, 1], Ca_2 \leftarrow M[1, c(path_1) + 1]$
end
Decide $C_1$ and $C_2$ :
if Both $M[r_1, c_1]$ and $M[r_2, c_2]$ are in regin a then
    $C_1 \leftarrow Ca_1, C_2 \leftarrow Ca_2$
else if $M[r_1, c_1]$ is in regin a, $M[r_2, c_2]$ is in regin b then
    $Cb_1 \leftarrow M[r_2 + L_r, c_2 + L_c]$
    $C_1 \leftarrow max(Ca_1, Cb_1), C_2 \leftarrow min(Ca_1, Cb_1)$
else if $M[r_1, c_1]$ is in regin b, $M[r_2, c_2]$ is in regin a then
    $Cb_1 \leftarrow M[r_1 + L_r, c_1 + L_c]$
    $C_1 \leftarrow max(Ca_1, Cb_1), C_2 \leftarrow min(Ca_1, Cb_1)$
else
    $Cb_1 \leftarrow M[r_1 + L_r, c_1 + L_c], Cb_2 \leftarrow M[r_2 + L_r, c_2 + L_c]$
    $C_1 \leftarrow max(Cb_1, Cb_2), C_2 \leftarrow min(Cb_1, Cb_2)$
end
```

The overall operations are as follows:

1. **Initialization**

   Insert $(n_s - 1)$ symbols in the first column and second symbol in the first row into the sorter. Because we want to ensure that there are at least two symbols in the first column and the first row. In Figure 3.10, the white circles represent the initialization when n_s is equal to 4.

2. **Output**

   Output two symbols with the first two largest possibilities in the sorter.

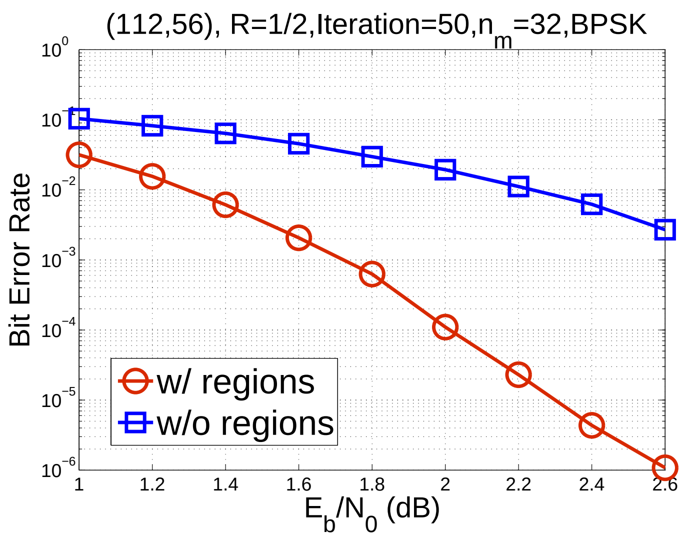


Figure 3.12: Performance comparison of defining regions

3. **Check**

   Check whether these two symbols are already in the output vector. If yes, give up the redundant symbols.

4. **Candidate choosing**

   (a) If output symbol is in region a, choose the candidates from the 2 candidates, C_{a_1} and C_{a_2}. (b) If output symbol is in region b, choosing the candidate followed predetermined path.

5. **Sorting**

   Insert two candidates C_1 and C_2 into the sorter, and C_1 is larger than C_2.

Using our proposed double throughput L-bubble check algorithm, there is no decoding performance loss compared with conventional one, and the processing time in a CES reduces to n_m cycles. The decoding performances of several decoding algorithms and our proposed one are depicted in Figure 3.13.

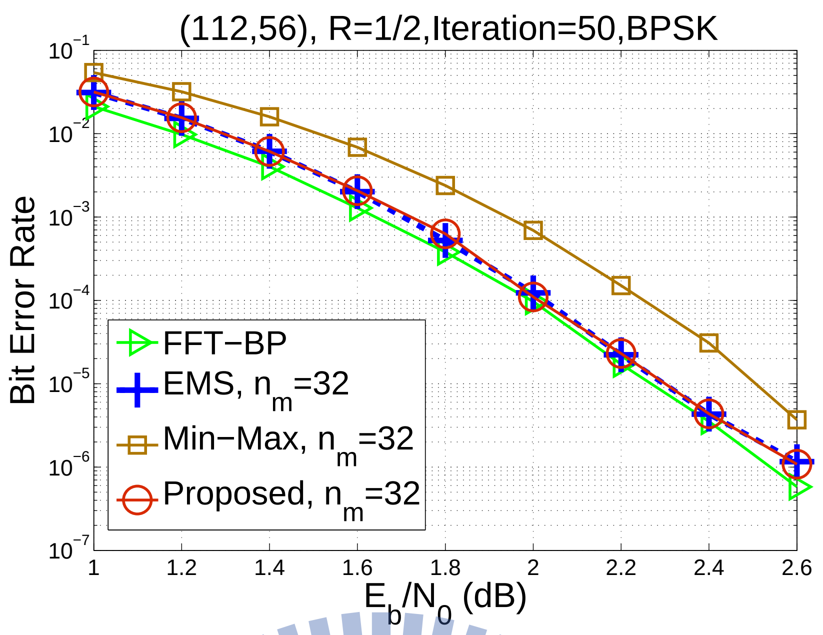


Figure 3.13: Comparison with other conventional algorithms

3.5 Variable Node Unit (VNU)

Target on the (2,d_c) non-binary LDPC codes, we proposed an efficient architecture to implement the variable node unit (VNU) with less memory usage. In the following sections, we first introduce the conventional variable elementary step (VES), and then the proposed VNU will be presented. The final parts in this section are the decision unit and proposed function unit which contains CES, VNU, and decision unit.

3.5.1 Variable Elementary Step (VES)

A VES is composed of two input vectors and one output vector, and the updating function is the following.

$$O[k] = \max_{\substack{I_1^{GF}[i]=I_2^{GF}[j]=O^{GF}[k] \\ i,j\in[1,n_m]}} \{I_1[i] + I_2[j]\} \tag{3.4}$$

The goal of a VES is to sort out the first n_m largest message possibilities among the $2n_m$ elements involved in the two input vectors. Assume that I_1 and I_2 are the input vectors and O is the output vector of the VES, and each vector size is equal to n_m. The updating process is described as the following:

1. **Candidates computation**

   First n_m cycles: According to the elements in I_1, search for the element with identical symbol from I_2 and combine their possibilities. If there is no corresponding identical symbol, adding the compensation value of I_2 to replace it.

   Second n_m cycles: Add the possibility of each symbol in I_2 with the compensation value of I_1.

$$
\begin{aligned}
&C[i] = I_1[i] + \begin{cases} I_2[j] \quad if \quad I_1^{GF}[i] = I_2^{GF}[j] \\ \gamma_{I_2} \quad if \quad I_1^{GF}[i] \notin I_2^{GF} \end{cases}, \quad \mathrm{C}^{GF}[i] = I_1^{GF}[i] \\
&C[i+n_m] = \gamma_{I_1} + I_2[i] \,, \quad \mathrm{C}^{GF}[i+n_m] = I_2^{GF}[i] \\
&i \in [1, 2, ..., n_m]
\end{aligned}
\tag{3.5}
$$

   The function of candidates computation is described in (3.5). Note that C represents as a vector which size is $2n_m$ for storing the candidates, and γ_1 and γ_2 stand for the compensation value of I_1 and I_2 respectively.

2. **Insert**

   According to the possibility calculated, insert the candidate into the output vector in decreasing order. If the identical symbol already exists in O, discard this candidate element. Repeat these two steps for $2n_m$ clock cycles, and the VES updating procedure is described in Algorithm 5.

In VES, it needs n_m "symbol matching" circuits to search identical symbol from n_m elements in I_2 as shown in Figure 3.14, and a symbol matching circuit is to check whether 2 inputs are the same or not. The conventional VES needs $2n_m$ cycles to process, and maybe several repeated symbols are accessed in these cycles. Therefore, we try to figure out the more efficient way to implement the VES without consuming redundant cycles on the repeated symbols.

Algorithm 5: Variable Elementary Step

```
Input: $I_1$ and $I_2$ (input vectors)
Output: $O$ (output vector)
for $i = 1$ to $2n_m$ do
    Candidate Computation:
    if $i \leq n_m$ then
        if the identical symbol exists, and $I_1^{GF}[i] = I_2^{GF}[j]$ then
            $Candidate \leftarrow I_1[i] + I_2[j]$
            $Candidate^{GF} \leftarrow I_1[i]^{GF} = I_2[j]^{GF}$
        else
            $Candidate \leftarrow I_1[i] + \gamma_{I_2}$
            $Candidate^{GF} \leftarrow I_1[i]^{GF}$
        end
    else
        $Candidate \leftarrow \gamma_{I_1} + I_2[i - n_m]$
        $Candidate^{GF} \leftarrow I_2[i - n_m]^{GF}$
    end
    Insertion:
    forall the $j$ such that $n_m \leq j \leq 1$ do
        $Diff[j] = Candidate - S[j]$
    end
    if $Diff[k] > 0$ and $Diff[k-1] \leq 0$ then
        forall the $j$such that $n_m \leq j \leq k+1$ do
            $S[j] \leftarrow S[j-1]$
            $S^{GF}[j] \leftarrow S^{GF}[j-1]$
        end
        $S[k] \leftarrow Candidate$
        $S^{GF}[k] \leftarrow Candidate^{GF}$
    end
    $i = i + 1$
end
```

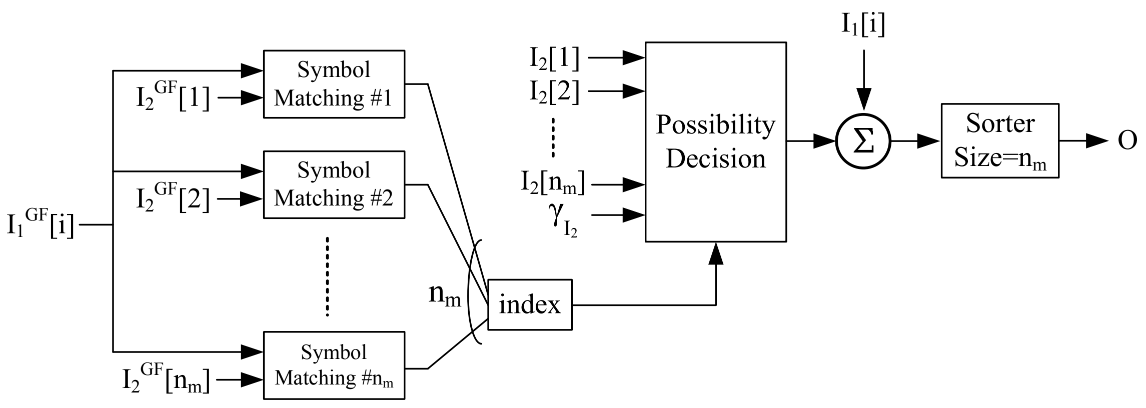


Figure 3.14: Conventional VES in i_{th} clock cycle

3.5.2 Proposed Variable Node Unit

Because the code we used is ultra sparse (2,d_c) non-binary LDPC code, we do not need to apply the forward and backward algorithm to implement a VNU. Furthermore, one of the inputs of each VNU is always the channel value. In order to efficiently improve the processing cycles and reduce the storage element for channel values, we change the approach of storing channel values from conventional one. Instead of storing the elements with the first n_m largest channel possibilities and corresponding symbols, the proposed VNU only stores the binary Log-Likelihood-Ratios (LLRs) of each variable node. According to the symbol of the input element from edge message, the corresponding LLR value is computed immediately from the channel value calculator (CVC). A CVC calculates the summation of binary LLRs by matching the symbol in binary representation as shown in Figure 3.15. Note that C is denoted as the channel value immediately computed from CVC, and it is defined over GF(2^p).

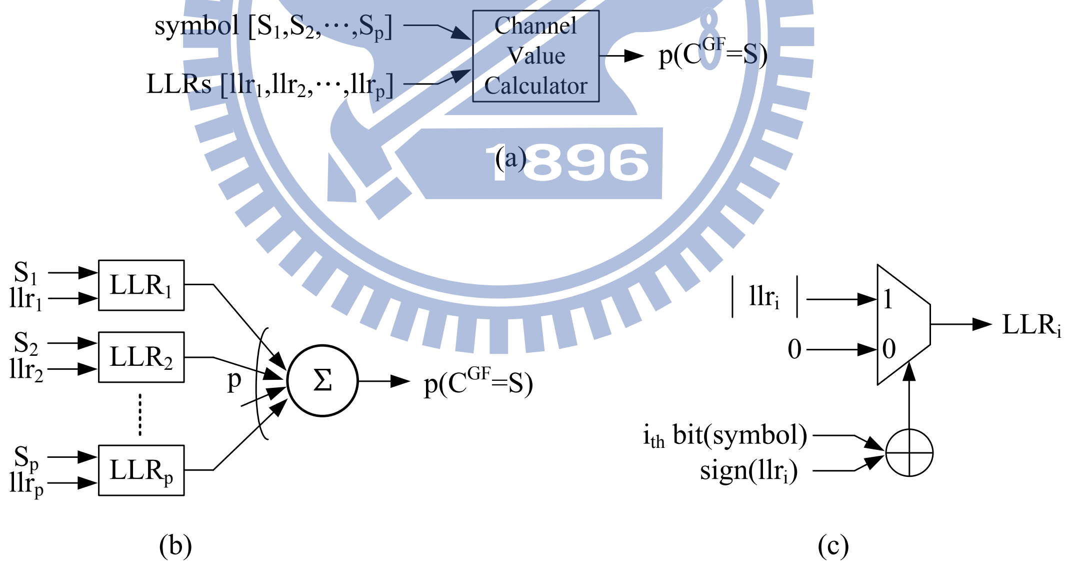


Figure 3.15: (a) Channel Value Calculator (b) Architecture of Channel Value Calculator (c) Architecture of calculating the i_{th} bit Log-Likelihood-Ratio

Updating process of proposed VNU are illustrated in Figure 3.16. Without considering the $2n_m$ elements involved in two incoming vectors, the proposed VNU only take account of n_m different symbols in the edge vector I_1. For this reason, using this method may miss several

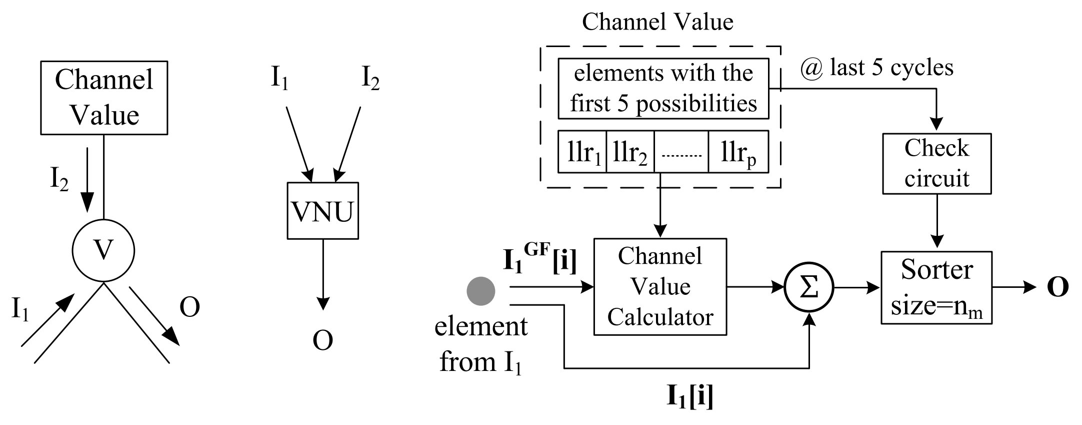


Figure 3.16: (a) Variable node degree is 2 (b) Variable node unit (c) Proposed VNU processing in i_{th} clock cycle

symbols not included in the edge vector, but they have non-negligible possibilities. Lack of considering these symbols with high weight will result in the decoding performance loss. Therefore, we need the extra memory to store some elements with the first several largest possibilities in channel value vector for compensating the performance loss. After performance simulation, the elements with the first 5 largest possibilities of channel values should be additionally stored in our design. Therefore, the processing cycles in proposed VNU is $n_m + 5$ which is fewer than $2n_m$ cycles required in conventional VES. The overall operation in a proposed VNU is described in Algorithm 6. Note that the notation of CVC stands for channel value calculator, and channel value vector stores elements with the first 5 largest possibilities.

Compared with traditional VES, the memory usage for channel value change from $(b_s+b_p)*n_m$ to $(b_{LLR})*p+(b_s+b_p)*5$ over GF(2^p). Note that (b_s,b_p,b_{LLR}) represent the quantization bits of symbol, possibility, and LLR value respectively. In Table 3.2, the channel value memory reduction in different cases are listed, and it is assumed that (b_s,b_p,b_{LLR}) is (6,7,6) and defined over GF(2^6). The proposed method can reduce the storage element for channel value especially when n_m is larger. When n_m is equal to 32, the reduction can reach about 75%. But if n_m is equal to 8, there is no remarkable improvement in the memory reduction of channel values.

Algorithm 6: Proposed Variable Node Unit

```
Input: $I$ ($C2V$ message vector), $LLRs$ and $C$ (channel value vector which store the first
       5 largest elements)
Output: $O$ (output vector)
for $i = 1$ to $(n_m + 5)$ do
    Channel Value Calculation:
    if $i \leq n_m$ then
        $Candidate = I[i] + CVC(I^{GF}[i])$
        $Candidate^{GF} = I^{GF}[i]$
    else
        $Candidate = \gamma_I + C[i - n_m]$
        $Candidate^{GF} = C^{GF}[i - n_m]$
    end
    Insertion:
    forall the $j$ such that $n_m \leq j \leq 1$ do
        $Diff[j] = Candidate - S[j]$
    end
    if $Diff[k] > 0 and Diff[k-1] \leq 0$ then
        $S[k] \leftarrow Candidate$
        $S^{GF}[k] \leftarrow Candidate^{GF}$
    end
end
```

Table 3.2: Comparison of the memory usage in channel values

Algorithm	n_m	Memory bits/Variable	Normalization
Traditional	32	$(6+7)*32 = 416$	1
Proposed	32	$6*6+(6+7)*5 = 101$	0.24
Traditional	8	$(6+7)*8 = 104$	0.25
Proposed	8	$6*6+(6+7)*5 = 101$	0.24

3.5.2.1 Decision Unit

In proposed VNU, the decision unit is included, and it operates simultaneously with VNU. Decision unit is to calculate the posterior probability by means of all incoming messages of the variable node, and choose the symbol with the largest posterior probability as the decoded result. In our work, the variable node degree is 2. There are 3 incoming vectors should be considered, 2 edge messages and 1 channel value. The edge memories stored are the messages from variable nodes to check nodes, and it contains the information of one of the C2V edge messages and channel value as shown in Figure 3.17 (a).

Therefore, we can use the C2V message immediately computed from the second stage CES

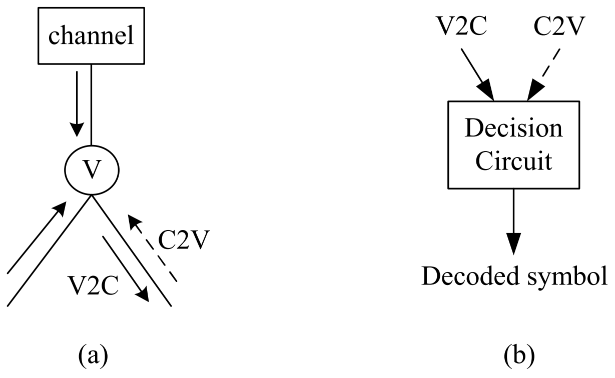


Figure 3.17: (a) Posterior probability of each variable is computed from C2V and V2C messages (b) Decision circuit

and the stored V2C message to calculate the posterior probability of each variable node. The Figure 3.18 illustrates the part of calculating the posterior probability of each variable in decision circuit. In addition, the main operation components in design unit is the same with conventional VES.

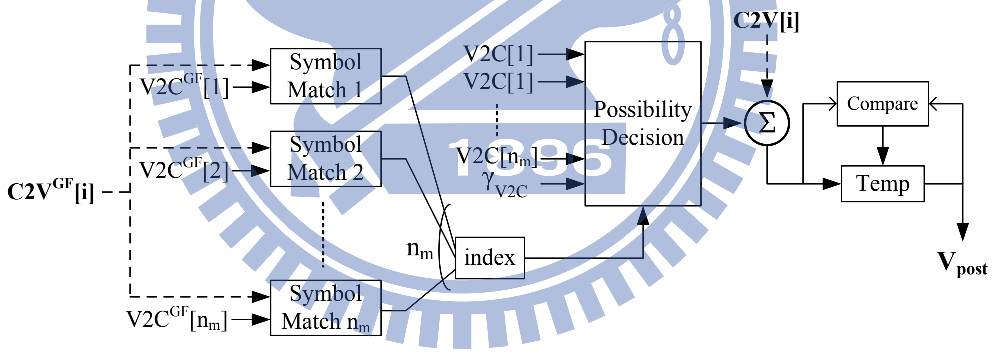


Figure 3.18: Calculating posterior probability in the decision unit

3.5.2.2 Proposed Processing Element (PE)

In our decoder architecture, we concatenate the second stage CES and one VNU to form a function unit for reducing the internal buffer size and sharing the check circuit. In this function unit, the output elements computed from the second stage CES will operate VNU simultaneously, and the internal buffer can reduce to half size compared with the original one.

As described in previous section, our proposed CES output 2 elements at a time, but the

VNU operates 1 element once. For this reason, the internal buffer is needed to temporarily store the elements. In Figure 3.19, there are three kinds of situation of the output elements from CES and the corresponding operations for controlling the internal buffer. Note that the meanings of ”redundant” and ”unique” stand for whether the element is already in the output vector or not.

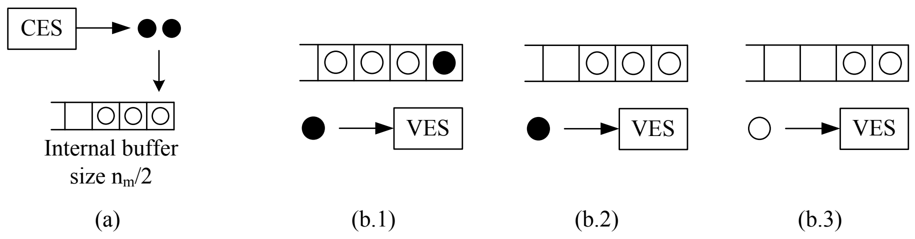


Figure 3.19: (a) Two output elements from CES prepare to operate VNU (b) Three cases for arranging internal buffer and deciding the input element of VNU

1. Figure 3.19 (b.1)

   The two output elements are all unique, one operates VNU and another one put into the internal buffer.

2. Figure 3.19 (b.2)

   If one of the two output symbols is redundant, using the unique one to executes VNU, and discard the redundant one.

3. Figure 3.19 (b.3)

   If both two output symbols are redundant, taking the symbol in the internal buffer to do VNU.

The overall block diagram which contains the second stage CES, a VNU, and decision unit is depicted in Figure 3.20.

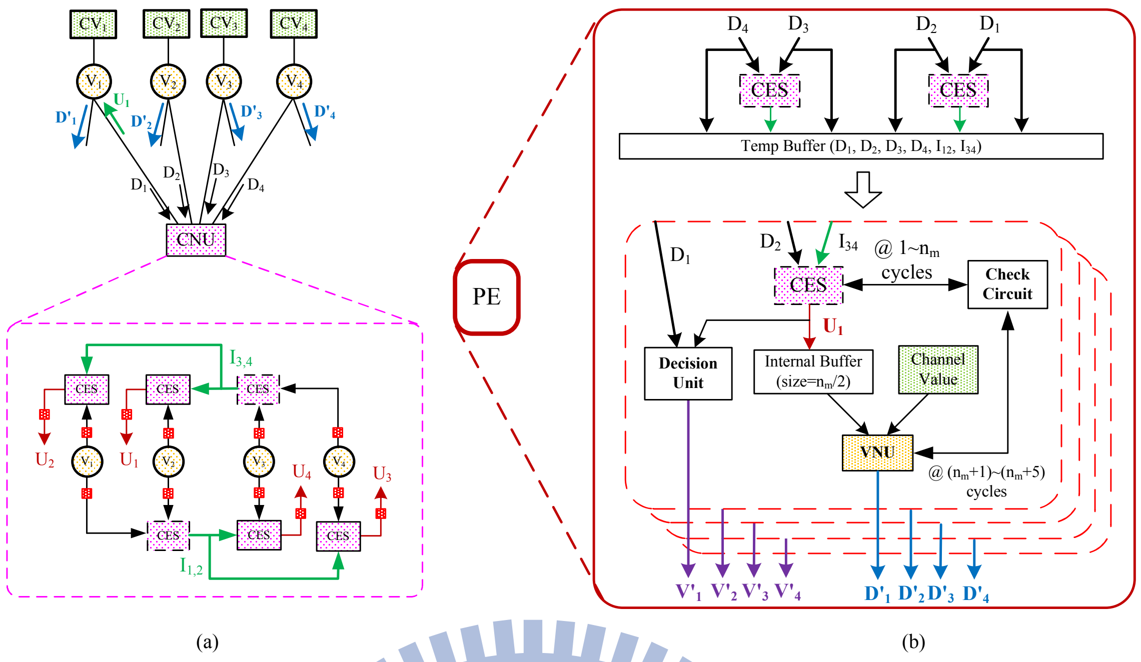


Figure 3.20: (a) Update the V2C message D'_1 (b) Block diagram of operating the second stage CES, VNU, and decision unit

3.6 Scheduling

3.6.1 Layered Scheduling

The key feature of layered scheduling is to use the immediately updated results from previous layers within the same iteration. In [27], it was shown that nearly half number of iterations improvement when applying the layered scheduling. Based on the code structure of the quasi-cyclic code, it is suitable to apply the layered decoding. In layered decoding scheduling, the overall check equations separate to several groups, and the group size stands for the number of check equations included.

Note that there is no specific restriction on grouping the check equations and the size of a group, and there are two kinds of common manners. One is to take the block row as a group, and the group size is the number of rows in a sub-matrix. Another one is composed of one row in each block row, and the group size is equal to the number of block rows in **H**.

3.6.1.1 Constraints of H

The methods of saving storage elements and controlling the memory access for making this decoder be an efficient design are based on some significant presuppositions. Here we conclude some requirements for applying the proposed decoder architecture in other non-binary LDPC codes.

1. Variable node degree d_v should be 2

   Because our proposed VNU is only for the code which the column degree is 2, and the half edge messages reduction and efficient VNU design are based on this property.

2. Arrange the order of accessing rows appropriately

   Since we only store one edge memory of each variable node, it should be prevented from accessing the variable node which has not finish updated yet.

3.6.2 Early Termination

In our proposed decoder architecture, the early termination design is included. It can reduce the redundant iteratively decoding, and our decoder can finish the decoding process in 4 iterations when the bit error rate is equal to 10^{-5}.

After operating the decision units, the decoded symbols are stored in the buffer. Until accomplish one iteration, computing the syndrome check to decide whether terminating the decoding process or not.

3.6.3 Memory Configuration

In the proposed decoder depicted in Figure 3.1, we use two main memory banks. V2C memory is used to store edge messages from variable node to check node. The binary LLRs of channel and the first n_c largest possibilities of each variable are included in LLR_{CV} memory bank. In addition, we separate these memory banks into several smaller memory blocks because of the limitation in the bandwidth. The configuration of the memory banks is illustrated in

Figure 3.21.

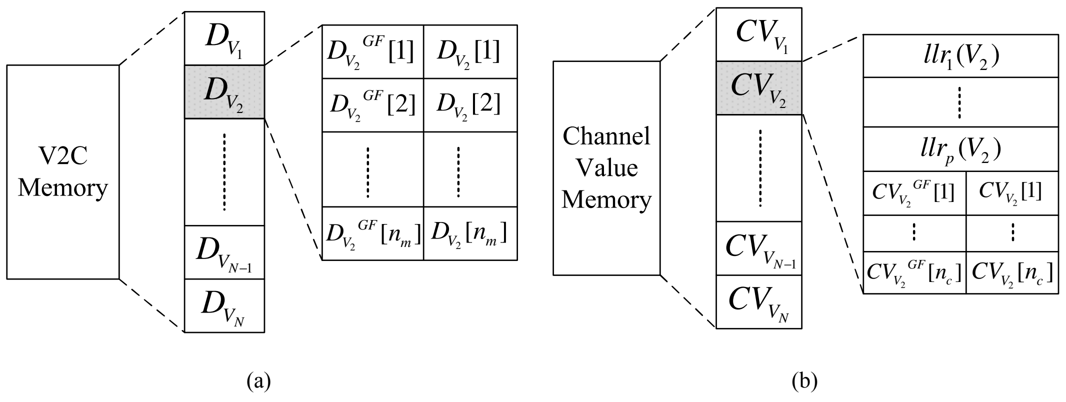


Figure 3.21: (a) Memory bank of V2C messages (b) Memory bank of channel values

3.6.3.1 Memory Collision Problem

Because of using the forward and backward algorithm, the $(2, 4)$ code needs 2 stages CESs to accomplish a check node updating function. Assumed that the processing time in a CES is n_t clock cycles. The calculation time of one check node includes the memory accessing time totally needs $4n_t$ cycles. Based on the memory configuration for V2C messages of the proposed decoder, there are some memory collision problem can not prevent from arranging the rows. In the following cases, assume that some associated variable nodes of the updating check nodes in $group_1$ are the same with in $group_2$, $group_3$ and $group_4$ as shown in Figure 3.22.

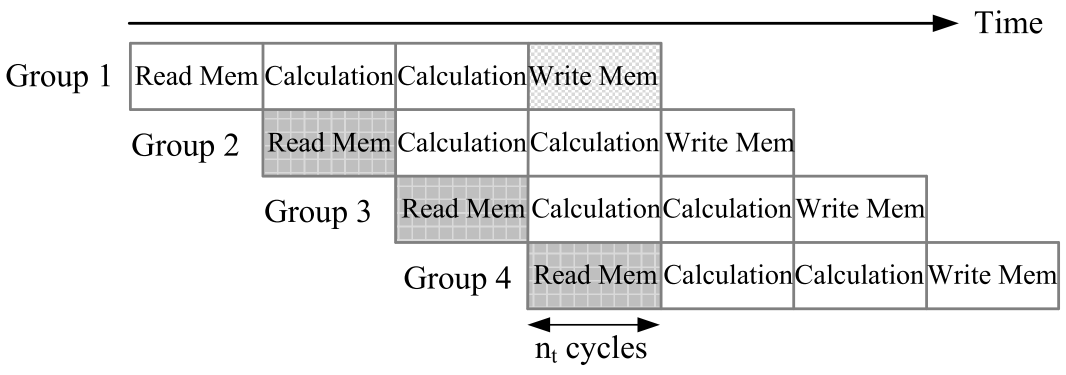


Figure 3.22: Every possible memory collision problems

1. Case 1

   Before completing the second edge message of the variable node in $group_1$, the V2C memory read by $group_2$ will be the wrong edge. And there is no way to solve this situation instead of changing the group decoding order.

2. Case 2

   The V2C messages are calculating when the $group_3$ want to access them. The solution is using the extra circuit to bypass the data from the output buffer to the input buffer of the computation units. The signal path covered in dashed line in Figure 3.1 is represented as the bypass path for this case.

3. Case 3

   The updated messages are already in the output buffer, and prepared to write into the V2C memory. The order of writing and reading data should be careful. It needs to check the updated message is already written into the memory before the next group reads it.

Chapter 4

Implementation Results

Using the efficient decoder architecture and improved CNU algorithm described in chapter 3, we implement a (2,4)-regular non-binary QC-LDPC decoder using proposed double throughput L-bubble check algorithm over GF(2^6). In this chapter, we will first discuss the architecture for this non-binary LDPC test chip. Then, we will illustrate the hardware implementation plan and the post-layout results of our chip. The comparison in hardware efficiency and decoding performance with other related works are presented. Finally, we apply the proposed decoder on the baseband simulation platform.

4.1 Chip Plan

Based on our proposed non-binary LDPC decoder architecture, we design 2 kinds of different n_m size, 8 and 32 respectively. According to the consideration of the hardware efficiency, we choose the decoder with n_m=8 to be realized in chip. The overall chip plan is depicted in Figure 4.1, and it can be separated into three major parts.

1. **Docoder**

   In the proposed decoder, We partial-parallel with 7 Processing Elements(PEs), and the overall architecture is illustrated in Figure 4.2. Each component for function unit and memory access in proposed decoder are illustrated, and the number in every function

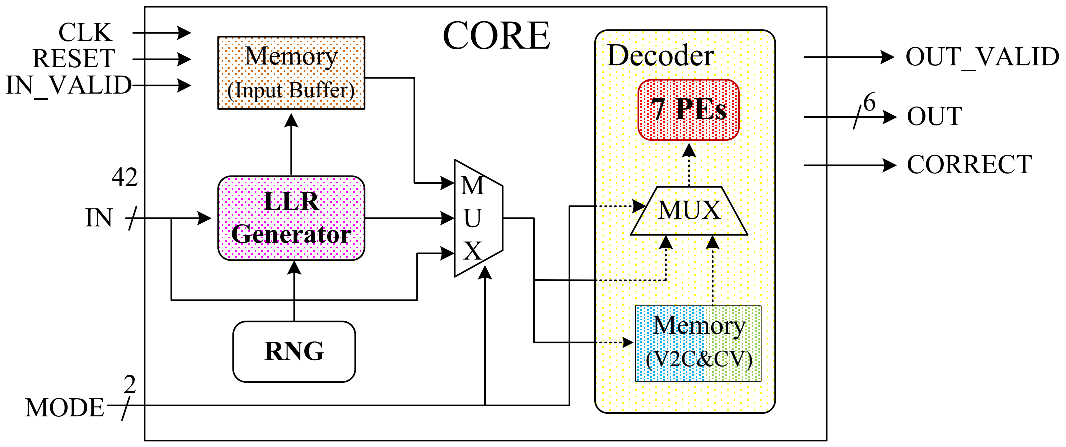


Figure 4.1: Chip plan

unit stands for the gate count. The total gate count of the core (includes decoder, LLR generator, and input buffer) is 655 K, and the proposed decoder accounts for 564 K.

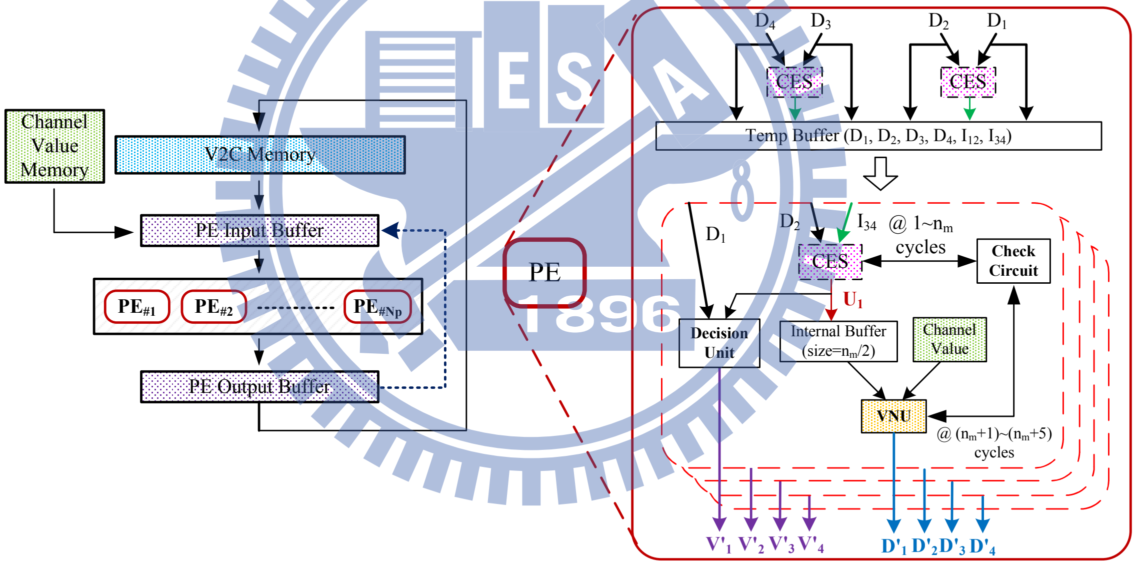


Figure 4.2: Components in the non-binary LDPC decoder

2. LLR Generator [28]

By means of the binary LLR values from channel, LLR generator is used to compute the first n_m largest LLR values. With LLR generator over GF(2^p), the required bits of input information for each variable node change from $b_p * 2^p$ to $b_{LLR} * p$, and the reduced amount can improve the processing time of initialization and reduce the number of input pins. Note that (b_p,b_{LLR}) stand for the quantization bits of possibility and binary LLR value respectively.

3. **Input Buffer**

   It is used to store the LLR values calculated from LLR generators for the next decoding process when the decoder is doing the iterative decoding. For this reason, the decoder can operate the decoding procedure continuously without wasting time on accessing the input information.

Note that we also design 4 kinds of test plan to test functions of certain modules for avoiding fabrication uncertainty. These 4 test mode are controlled by 2 multiplexers as shown in Figure 4.1, and the testing targets are as follows

1. Normal operation
2. Test for input buffer
3. Test for one PE
4. Test for one PE without any input information

In mode 4, using the Random Number Generator (RNG) to generate the values as the binary LLRs, and it is designed for the case when some of the input pins in chip malfunction.

4.2 Post-layout Results

With 90-nm CMOS process technology, the proposed (112,56) non-binary QC-LDPC layered decoder over GF(2^6) is implemented, and the number of n_m=8. In the following, we list the post-layout results of our test chip in Table 4.1, and show the layout picture in Figure 4.3.

4.3 Comparisons

Table 4.2 compares the proposed decoder with other related works [4] [5], and the synthesis result with n_m=32 is also listed. Because there is no other synthesis or post-layout result of

Table 4.1: Post-layout results of (112,56) non-binary QC-LDPC codes decoder

Technology	90-nm CMOS process
Algorithm	EMS
Scheduling	Layered
n_m,n_s	8,5
Quantization bits	7
Logic gate count	580 K
Memory bits	42 K
Frequency	277 MHz
Throughout	124.6 Mb/s
Core area	2.24 mm^2
Power	274 mW

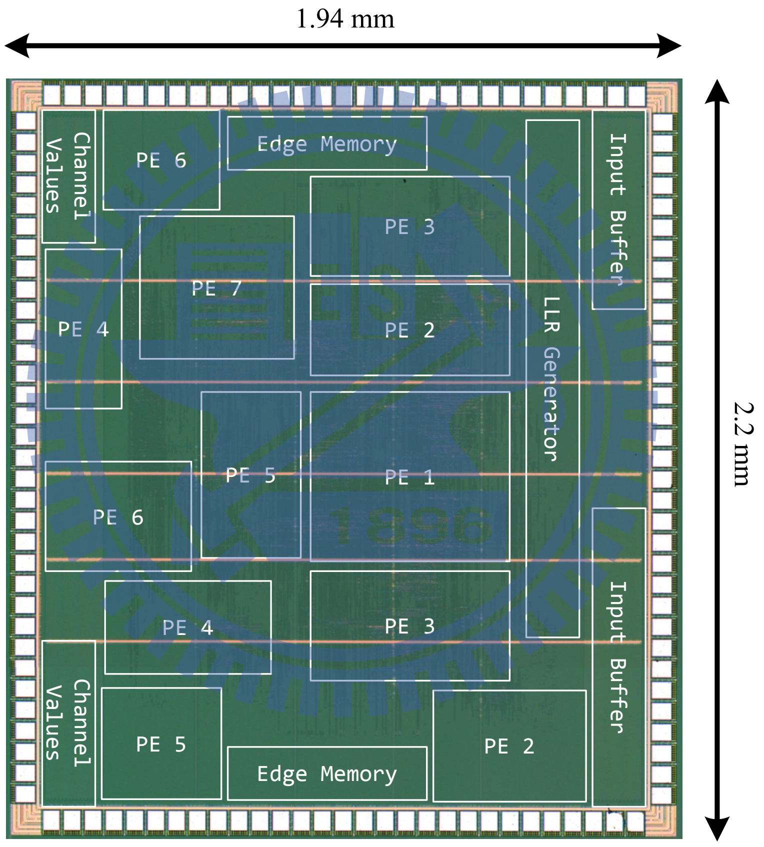


Figure 4.3: Post-layout photo

non-binary LDPC decoder with EMS decoding algorithm, the related works listed in Table 4.2 are implemented using Min-Max decoding algorithm. In non-binary LDPC, the hardware performance is usually represented as the parameter denoted hardware efficiency, which is defined of throughput-to-gate-count-ratio. From the value of hardware efficiency in the comparison table, our proposed decoder is an hardware efficient implementation. Note that the EMS decoding algorithm we used is more complicated than Min-Max algorithm, and the field size we applied is also the highest compared with others. Figure 4.4 shows the decoding performance with [5]. Based on the competitive decoding performance with [5], our design have 4.3 times improvement in hardware efficiency.

Table 4.2: Comparison Table

	[4] T-CASI-2012 Synthesis	[5] T-CASI-2011 Post-layout	Proposed Synthesis	Proposed Post-layout
Code length	640	248	112	112
Code rate	0.5	0.55	0.5	0.5
Galois Field / n_m	GF(32)/32	GF(32)/8	GF(64)/32	GF(64)/8
(d_v, d_c)	(3,6)	(4,8)	(2,4)	(2,4)
Algorithm	Min-Max	Min-Max	EMS	EMS
Process	180 nm	90 nm	90 nm	90 nm
Quantization bit	7b	7b	7b	7b
Frequency	200 MHz	260 MHz	312 MHz	277 MHz
Iterations	10	10	10	10
Throughput	31.2 Mb/s	47.69 Mb/s	57 Mb/s	124.6 Mb/s
Decoder gate count	1.24 M	1.92 M	1.42 M	564 K
Area (mm^2)	N/A	10.33	N/A	2.24
Power (mW)	N/A	479.8	N/A	274
Hardware Efficiency (M bps/M gates)	50.4	24.84	40.1	220.9
Energy Efficiency (nJ/bit/iter)	N/A	1.01	N/A	0.22

In Figure 4.4, we also display the decoding result with FFT-SPA decoding algorithm, which represents the curve without performance loss. Therefore, we can find out the decoder which n_m is 32 is very close to the curve of FFT-SPA result. Because the code length we used is relatively small, it has the error floor problem at higher signal-to-noise-ratio (SNR). But if we apply to other $(2, 4)$ ultra sparse code with longer code length, the hardware efficiency is still the same.

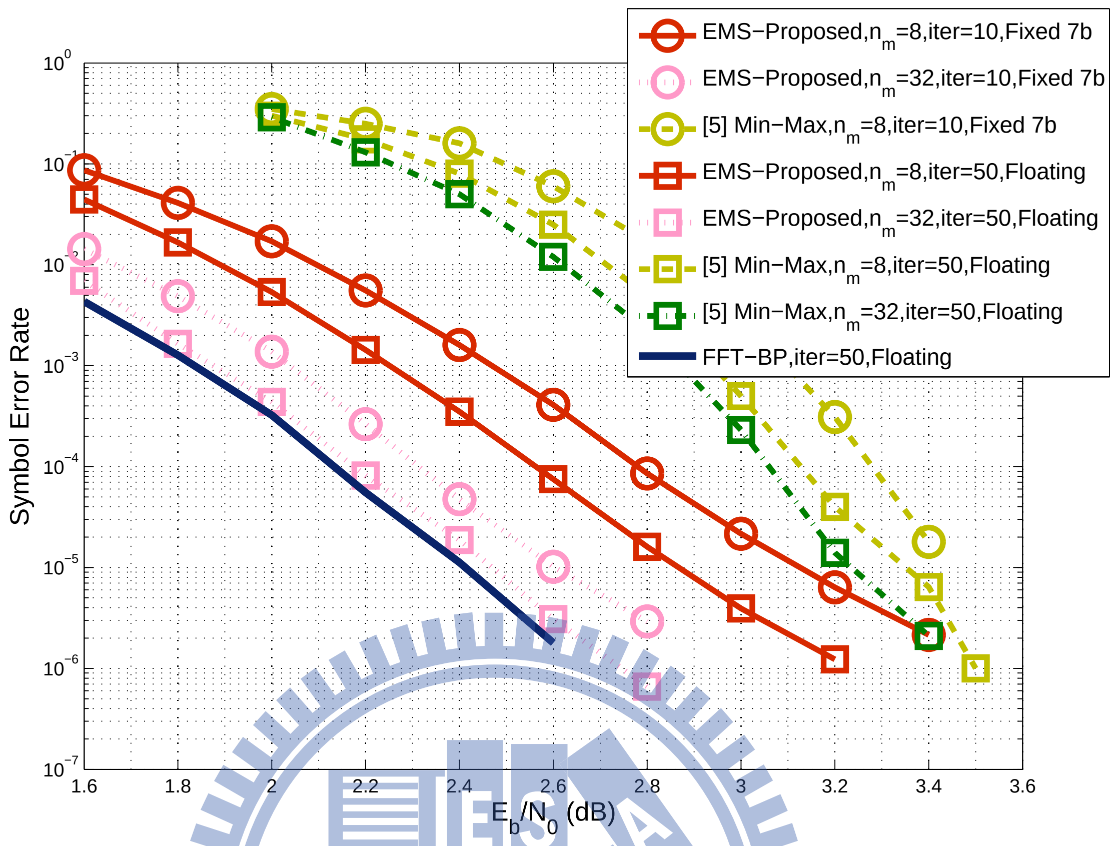


Figure 4.4: Symbol error performances comparison of proposed (112,56) non-binary LDPC decoder using EMS algorithm over GF(2^6) and [5] (248,137) non-binary LDPC decoder using Min-Max algorithm over GF(2^5). The simulation is performed under BPSK modulation and AWGN channel

4.4 Application

Applying our design on the simulation platform which supports dual SC/HSI modes of IEEE 802.15.3c applications [29] [30]. In Single Carrier (SC) and High Speed Interface (HSI) modes, the simulation results of the proposed decoders which n_m is 32 and 8 are shown in Figure 4.5 and Figure 4.6 respectively. In addition, the simulation result of (672,336) binary LDPC code which is well-chosen for 802.15.3c specification is also depicted.

In SC mode, there are two kinds of modulations (8PSK and 16QAM) to simulate, and the dashed line represents 16 QAM modulation in Figure 4.5. With the 16QAM modulation, our proposed decoder can outperform the binary code more than 1 dB. But in the 8PSK modulation, the performance result with n_m=8 is worse than the binary case about 0.2 dB.

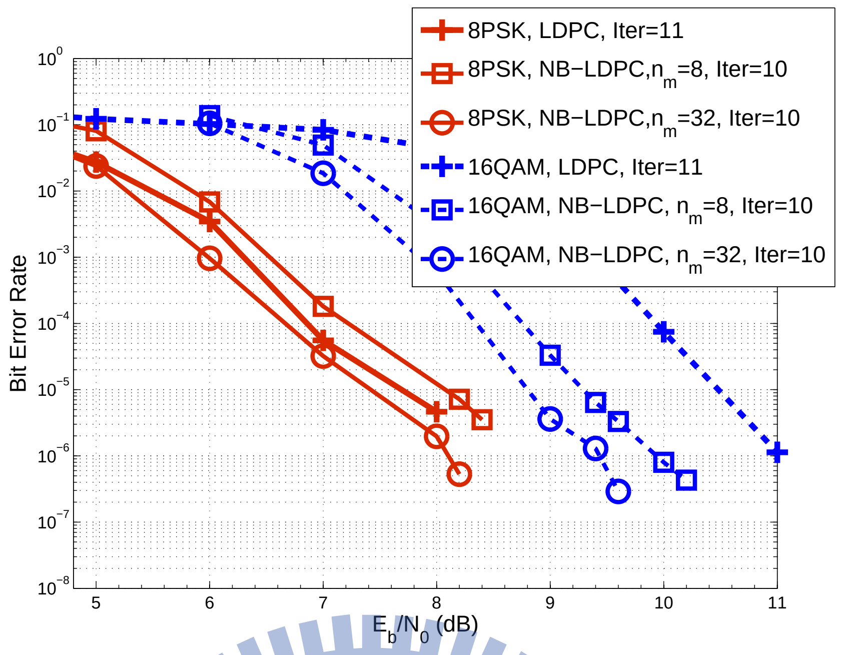


Figure 4.5: Performance simulations for (112,56) non-binary LDPC codes and (672,336) binary LDPC codes in SC mode.

HSI mode has 16QAM and 64QAM modulations, and the dashed line in Figure 4.6 stands for 16QAM. From the performance curve, the implementation with n_m=32 is better than the binary case about 0.2 dB in both modulations. When n_m is equal to 8, the simulation results worse than the binary one. But in the 64QAM modulation, the performance curve of n_m=8 is overlapping with the binary code at low error rate.

The proposed decoder design which n_m is 32 has good decoding ability, and the performance loss is negligible ($<$ 0.1 dB). Considering from the simulation results with no performance loss, the (112,56) non-binary LDPC code can better than the (672,336) binary LDPC code which is well-chosen for the 802.15.3c specification about 0.2 dB in average. Because the implementation that n_m is 8 has about 0.4 dB performance degradation, it results in the worse decoding performance than the binary case.

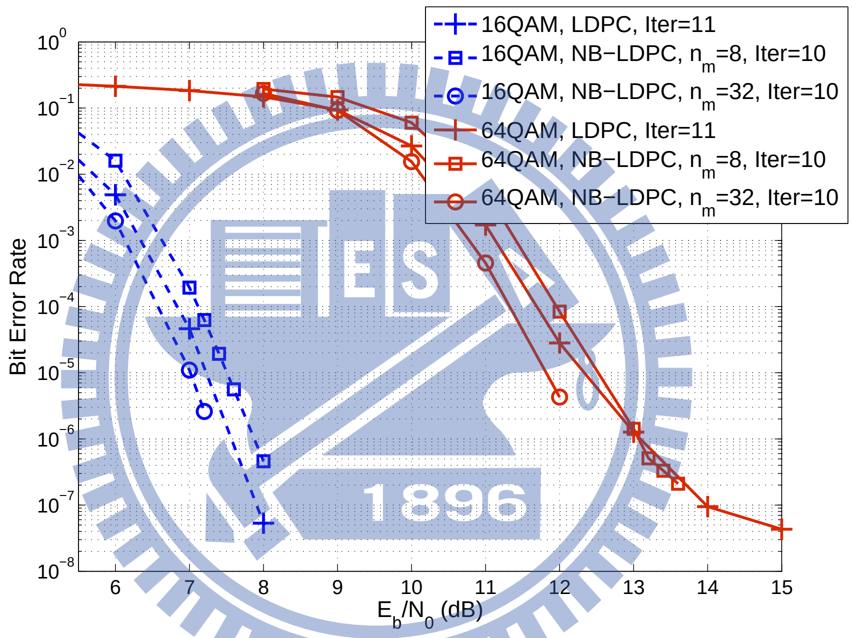


Figure 4.6: Performance simulations for (112,56) non-binary LDPC codes and (672,336) binary LDPC codes in HSI mode.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, a novel decoder architecture for non-binary QC-LDPC codes with improved EMS decoding algorithm was proposed. Using 90-nm CMOS process, a (2,4)-regular non-binary QC-LDPC decoder over GF(2^6) is implemented. To the best of our knowledge, this is the first chip of non-binary LDPC decoder using EMS decoding algorithm and high order finite field ($\geq$ GF(2^6)).

Compared with state-of-the-art, our design has 5 advantages. First, we enhance the throughout in CES and VES, which are the main computation units in the decoder. In the implementation of $n_m = 8$, it can reach over 100 Mbps throughout with only 655 K gate counts (564 K gate count only for decoder). Second, because of the code property that variable node degree is 2, we can reduce half storage elements for edge messages. Third, the architecture in proposed VES can efficiently save the memory usage in channel values, and it has about 75% reduction when the number of n_m is equal to 32. Forth, we use EMS decoding algorithm which is more complicated than Min-Max decoding algorithm but it has better decoding performance. Although we use very short code length (672 bits), the decoding performance are still competitive with other designs based on the higher finite field size (GF(2^6)). Especially when choosing n_m=32 to implement, the decoding performance loss is negligible ($<$ 0.1 dB) compared with FFT-SPA decoding algorithm. Fifth, the hardware efficiency of our design is better than other existing works, and it has at least 4.3 times improvement.

Based on the improvements in our design above, we can really enhance the hardware efficiency of the non-binary LDPC decoder, and have well enough decoding performance. Using a 90-nm CMOS process, we implemented the proposed non-binary LDPC decoder with n_m=8, and the throughput can reach over 100 Mbps.

5.2 Future Work

In proposed decoder, we double the throughput and reduce the memory usage by the code property that the column degree is 2. Applying these techniques, non-binary LDPC decoder design for two different size of n_m (8 and 32) are provided. In non-binary LDPC decoding algorithm, the larger size of n_m can have better decoding performance, but the computational complexity and memory usage are also increased. Based on the consideration of hardware cost, we choose the decoder with n_m=8 to implement, but its decoding performance is worse than the case of n_m=32 about 0.4 dB. For this reason, we can try to figure out some approaches to decrease the performance loss when changing the value of n_m. Besides, as we only store half edge messages in our decoder in layered scheduling, grouping and processing order are important in decoding. To avoid the memory collision problems (in 3.6.3.1), we rearrange the processing order by manual operation. For more general implementation, the relation between the accessing order of groups and the number of parallelism in the decoder should be made by a valid inference.

Bibliography

[1] M. Davey and D. MacKay, “Low-density parity check codes over gf(q),” *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165 –167, June 1998.

[2] J. Chen, L. Wang, and Y. Li, “Performance comparison between non-binary ldpc codes and reed-solomon codes over noise bursts channels,” in *Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on*, vol. 1, may 2005, pp. 1 – 4 Vol. 1.

[3] A. Marinoni, P. Savazzi, and S. Valle, “Efficient design of non-binary ldpc codes for magnetic recording channels, robust to error bursts,” in *in Proc. ISTC*, Sep. 2008, pp. 288 –293.

[4] X. Chen, S. Lin, and V. Akella, “Efficient configurable decoder architecture for nonbinary quasi-cyclic ldpc codes,” *IEEE Trans. On Circuit and Systems-I*, vol. 59, no. 1, pp. 188 –197, Jan 2012.

[5] Y.-L. Ueng, C.-Y. Leong, C.-J. Yang, C.-C. Cheng, K.-H. Liao, and S.-W. Chen, “An efficient layered decoding architecture for nonbinary qc-ldpc codes,” *IEEE Trans. On Circuit and Systems-I*, vol. 59, no. 2, pp. 385 –398, Feb. 2012.

[6] X. Zhang and F. Cai, “Reduced-complexity decoder architecture for non-binary ldpc codes,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 7, pp. 1229 –1238, July 2011.

[7] J. Lin, J. Sha, Z. Wang, and L. Li, “Efficient decoder design for nonbinary quasicyclic ldpc codes,” *IEEE Trans. On Circuit and Systems-I*, vol. 57, no. 5, pp. 1071 –1082, May 2010.

[8] I. Gutierrez, G. Bacci, J. Bas, A. Bourdoux, H. Gierszal, A. Mourad, and S. Pleftschinger, “Davinci non-binary ldpc codes: Performance and complexity assessment,” in *Future Network and Mobile Summit, 2010*, June 2010, pp. 1 –8.

[9] X. Jiang, Y. Yan, X. gen Xia, and M. H. Lee, “Application of nonbinary ldpc codes based on euclidean geometries to mimo systems,” in *in Proc. WCSP*, Nov. 2009, pp. 1 –5.

[10] L. Costantini, B. Matuz, G. Liva, E. Paolini, and M. Chiani, “On the performance of moderate-length non-binary ldpc codes for space communications,” in *2010 5th Advanced Satellite Multimedia Systems Conference and the 11th Signal Processing for Space Communications Workshop*, Sep. 2010, pp. 122 –126.

[11] G. Calzolari, M. Chiani, F. Chiaraluce, R. Garello, and E. Paolini, “Channel coding for future space missions: New requirements and trends,” *Proceedings of the IEEE*, vol. 95, no. 11, pp. 2157 –2170, Nov. 2007.

[12] L. Barnault and D. Declercq, “Fast decoding algorithm for ldpc over gf(2q),” in *IEEE Information Theory Workshop*, march-4 April 2003, pp. 70 – 73.

[13] H. Wymeersch, H. Steendam, and M. Moeneclaey, “Log-domain decoding of ldpc codes over gf(q),” in *in Proc. IEEE ICC*, vol. 2, June 2004, pp. 772 – 776 Vol.2.

[14] ——, “Computational complexity and quantization effects of decoding algorithms for non-binary ldpc codes,” in *in Proc. ICASSP*, vol. 4, May 2004, pp. iv–669 – iv–672 vol.4.

[15] D. Declercq and M. Fossorier, “Decoding algorithms for nonbinary ldpc codes over gf(q),” *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633 –643, Apr. 2007.

[16] ——, “Extended minsum algorithm for decoding ldpc codes over gf(q),” in *on Proc. ISIT*, Sep. 2005, pp. 464 –468.

[17] V. Savin, “Min-max decoding for non binary ldpc codes,” in *in Proc. ISIT*, July 2008, pp. 960 –964.

[18] A. G. A. Conde-Canencia, L. and E. Boutillon, “Complexity comparison of non-binary ldpc decoders,” June 2009.

[19] T. Lehnigk-Emden and N. Wehn, “Complexity evaluation of non-binary galois field ldpc code decoders,” in *in Proc. ISTC*, Sep. 2010, pp. 53 –57.

[20] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard, “Architecture of a low-complexity non-binary ldpc decoder for high order fields,” in *in Proc. ISCIT*, Oct. 2007, pp. 1201 –1206.

[21] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, “Construction of non-binary quasi-cyclic ldpc codes by arrays and array dispersions,” *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1652 –1662, June 2009.

[22] Y.-K. Lin, “Design of structured cp-peg ldpc codes with low error floor,” Master’s thesis, National Chiao Tung University, Nov. 2007.

[23] X.-Y. Hu and E. Eleftheriou, “Binary representation of cycle tanner-graph gf(2b) codes,” in *in Proc. IEEE ICC*, vol. 1, June 2004, pp. 528 – 532 Vol.1.

[24] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, “Low-complexity decoding for non-binary ldpc codes in high order fields,” *IEEE Trans. Commun.*, vol. 58, no. 5, pp. 1365 –1375, May 2010.

[25] E. Boutillon and L. Conde-Canencia, “Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary ldpc decoders,” *Electronics Letters*, vol. 46, no. 9, pp. 633 –634, Apr. 2010.

[26] ——, “Simplified check node processing in nonbinary ldpc decoders,” in *in Proc. ISTC*, Sep. 2010, pp. 201 –205.

[27] D. Hocevar, “A reduced complexity decoder architecture via layered decoding of ldpc codes,” in *in Proc. IEEE SIPS*, Oct. 2004, pp. 107 – 112.

[28] A. Ghouwayel and E. Boutillon, “A systolic llr generation architecture for non-binary ldpc decoders,” *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 851 –853, august 2011.

[29] Y.-S. Huang, W.-C. Liu, and S.-J. Jou, “Design and implementation of synchronization detection for ieee 802.15.3c,” in *in Proc. VLSI-DAT*, Apr. 2011, pp. 1 –4.

[30] F.-C. Yeh, T.-Y. Liu, T.-C. Wei, W.-C. Liu, and S.-J. Jou, “A sc/ofdm dual mode frequency-domain equalizer for 60ghz multi-gbps wireless transmission,” in *in Proc. VLSI-DAT*, Apr. 2011, pp. 1 –4.