

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

低密度同位元檢查碼於無線通訊網路之設計



The Design of LDPC Decoder for WiMAN

802.16e

研 究 生：嚴紹維

指 導 教 授：周世傑 博士

中 華 民 國 九 十 五 年 八 月

低密度同位元檢查碼於無線通訊網路之實現

The Implementation of LDPC code for WiMAN

802.16e

研究生：嚴紹維
指導教授：周世傑

Student: Shau-Wei Yen
Advisor: Dr. Shye-Jye Jou



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
in
Electronics Engineering

June 2006

HsinChu, Taiwan, Republic of China

中華民國九十五年七月

低密度同位元檢查碼於無線通訊網路之設計

研究生：嚴紹維

指導教授：周世傑

國立交通大學電子工程學系電子研究所

摘要

在本論文中，我們提出一個支援全模式的低密度同位元檢查碼解碼器的設計，此設計適用於無線通訊網路 WiMAN 802.16e 系統。為了簡化在解碼中執行的運算，採用 Min-sum algorithm 搭配常態化的方法以達到與理論值相同的位元錯誤率表現。此架構採用了一個重新安排的解碼流程來降低記憶體的使用量以及解碼的時間。此外更利用 802.16e 低密度同位元檢查碼中同位元檢查矩陣的特性去支援不同模式。經過 0.13um 製成實作晶片，所提出的部份平行解碼器於固定 20 次迴圈的解碼模式下，可以達到最高的傳輸速率為 20.3Mb，在迴圈的解碼過程當中，功率消耗為 700mW。

The Design of LDPC Decoder for WiMAN

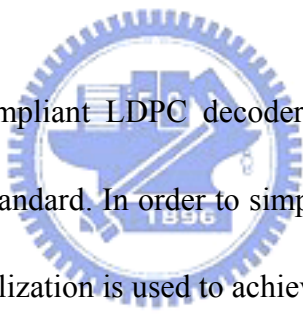
802.16e

Student : Shau-Wei Yen

Adviser:Shye-Jye Jou

Department of Electronics Engineering & Institute Electronics
National Chiao Tung University

ABSTRACT



In this thesis, a fully compliant LDPC decoder is presented. The LDPC decoder is applied for WiMAN 802.16e standard. In order to simplify the computation of decoding unit, Min-sum algorithm with normalization is used to achieve the BER performance the same with the theoretical performance. The architecture adopts a re-schedule decoding data flow to reduce memory usage and decoding latency. Also the characteristic of 802.16e LDPC parity check matrices is used to support different modes. After fabricated in 0.13um 1P8M process, the proposed partial parallel decoder can support 20.3Mb/s data rate under 20 decoding iteration. The power consumption is 700mW while iteration decoding.

誌 謝

這短短兩年的碩士生涯，讓我收穫許多，在許多師長的提攜以及同學朋友的鼓勵幫助下，讓我的研究路程順利且平穩，首先我要感謝的是我的指導教授周世傑老師，無論在研究上以及做人處事都給我許多的建議以及鼓勵，讓我有信心面對研究的挑戰，也提供我最好的研究資源讓我對於研究能夠更專注。

再來我要感謝我的學姐 momo，在這兩年他一直從旁鼓勵並幫助我，讓我對於研究上的問題都能迎刃而解，還要感謝實驗室的學長庭楨以及小胖給予我的幫助以及感謝學弟俊男幫助我在研究上有更快的進度，此外要感謝全實驗室的學長姐同學學弟們，讓我在這個實驗室能以愉快的心情去作研究。

再來要感謝 Ocean group 的全部成員，尤其是建青學長給予我許多研究上的建議以及研究的方向，讓我能夠更專注在我的研究，也感謝整個 Ocean group 提供了我許多研究相關的資訊以及建議。

最要感謝的就是我的家人，因為有你們在我背後全力的支持與付出，讓我能夠專心於我的研究。



Contents

Chapter 1 Introduction.....	1
1.1 Overview of Channel Codec for Wireless Metropolitan Area Network.....	1
1.2 Motivation	1
1.3 Thesis Organization.....	2
Chapter 2 LDPC Code.....	3
2.1 Concept of LDPC	3
2.1.1 Message Passing Algorithm	4
2.1.2 Decoding Concept	11
2.1.3 Decoding Flow	13
2.2 LDPC Code for 802.16e	16
2.2.1 Parity Check Matrix Definition.....	18
2.2.2 LDPC Encoder.....	20
2.2.3 Implementation Bottleneck.....	22
Chapter 3 Algorithm Optimization for Implementation.....	23
3.1 Min-Sum Algorithm	23
3.2 Simulation Result	30
Chapter 4 Architecture Design and Circuit Implementation	35
4.1 Decoder Design	35
4.1.1 Architecture Overview.....	35
4.1.2 Iterative Decoding Block.....	36
4.1.3 Memory Arrangement.....	44
4.1.4 Cyclic Shift Block	44
4.1.5 Shift Size ROM Table.....	48
4.2 Chip Implementation	50
Chapter 5 Conclusion and Future Work	53
5.1 Conclusion.....	53
5.2 Future Work.....	53
References.....	54

List of Figures

Fig. 2.1	An example of normal graph	5
Fig. 2.2	Graph representation of the extrinsic and the intrinsic probabilities.....	6
Fig. 2.3	Graph representation of the message passing between two vertices.....	8
Fig. 2.4	The corresponding Tanner graph	11
Fig. 2.4	The check node update for B_1	12
Fig. 2.5	Bit node update for B_1	13
Fig. 2.6	The example of definition for decoding procedure	14
Fig. 2.8	The definition of H_{b2}	19
Fig. 2.9	Decomposition of parity check matrix	21
Fig. 2.10	The architecture of encoder for LDPC codes	21
Fig. 3.1	The check node with degree d	24
Fig. 3.2	The bit node with degree k	25
Fig. 3.3	Plot of the $\Psi(x)$ function	28
Fig. 3.4	Simulation result (1): theoretical value for maximum code length	30
Fig. 3.5	Simulation result (2): theoretical value for minimum code length.....	31
Fig. 3.6	Simulation result (3): Normalization factor comparison	31
Fig. 3.7	Simulation result (4): Fixed-point simulation for integer part.....	32
Fig. 3.8	Simulation result (5): Fixed-point simulation for fraction part.....	33
Fig. 3.9	Simulation result (6): Fixed-point simulation for iteration.....	33
Fig. 4.1	LDPC decoding flow	35
Fig. 4.2	LDPC decoder block diagram	36
Fig. 4.2	Check Node Updating Memory Access Schedule.....	38
Fig. 4.3	One Data Sorter	39
Fig. 4.4	C2B Register Arrangement.....	39
Fig. 4.5	B2C block architecture	41
Fig. 4.6	XOR block architecture.....	42
Fig. 4.7	B2C memory block	43
Fig. 4.8	Deriving $q_{n,k}$ architecture	43
Fig. 4.9	5×5 identity matrix and its permutation	45

Fig. 4.10	C2B register arrangement.....	45
Fig. 4.11	C2B register arrangement after shifting.....	45
Fig. 4.12	B2C memory arrangement.....	46
Fig. 4.13	B2C memory arrangement after right shifting.....	46
Fig. 4.14	B2C memory arrangement after left shifting.....	46
Fig. 4.15	Sharing mechanism of cyclic shifter.....	47
Fig. 4.16	Two level cyclic shifter.....	48
Fig. 4.17	Overall architecture.....	49
Fig. 4.18	Chip layout of the LDPC decoder chip.....	50



List of Tables

Table 2.1	LDPC block size and code rate	17
Table 3.1	Parameter setting for implementation	34
Table 4.1	Summary of the LDPC decoder chip	51
Table 4.2	Gate count of functional block.....	51
Table 4.3	Comparison of LDPC chip	52



Chapter 1

Introduction

1.1 Overview of Channel Codec for Wireless Metropolitan Area Network

Wireless Metropolitan Area Network, allowing end-users to travel throughout a hot zone cell without losing connectivity, has been a very important technique in wireless communication. The services provide portability and mobility to make users more convenient to access information. For a high quality service, the channel capacity seems more important for WiMAN, therefore, the error correcting capability is a great issue in WiMAN. In WiMAN 802.16e standard [23], there are four channel coding methods: Convolutional coding (CC) [1], Reed-solomon coding (RS)[2], Turbo coding [3], and Low-Density Parity Check Coding (LDPC) [4]. The first three codes have been proposed in many application, such as DVB-T etc, and LDPC coding was rediscovered in recent years. Because LDPC can provide a better error correcting capability than the first three codes, so WiMAN adopts LDPC coding as an optional error correcting method.

1.2 Motivation

LDPC code was first proposed by Gallager [4] in 1953. It can provide a better performance in error correcting capability, but due to the difficulty of circuit implementation, LDPC code was not on the main stream until it was rediscovered by Mackay [5][6]. LDPC code provide a simple algorithm when decoding, however, the circuit implementation is still a

great challenge even the implementation technique has advanced a lot. In WiMAN 802.16e standard, it provides many different types of LDPC parity check matrix for users to choose according to the tradeoff between performance and cost. All types of parity check matrices make the architecture hard to design.

In this thesis, we propose an architecture that can support all types of parity check matrices in WiMAN 802.16e. The architecture provides a partial parallel computation unit method to accelerate the throughput rate, and employs the special characteristics of LDPC code in 802.16e to make the data paths and memory controls more simple to decrease the hardware complexity in circuit implementation. The detail discussion and architecture will be given in the following chapters.

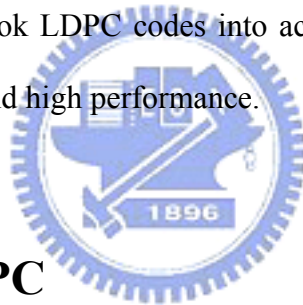
1.3 Thesis Organization

The remainder of this thesis is organized as follow. Chapter 2 describes the concept and the decoding algorithm of LDPC codes and the definition of parity check matrix of LDPC in WiMAN 802.16e standard. Some improved algorithms for LDPC codes and simulation results are introduced at Chapter 3. In Chapter 4, the proposed LDPC decoder architecture, including functional units, memory arrangement, are presented in detail. Besides, the chip implementation results and the summary will be described in Chapter 5. Finally, conclusions and future work are made in Chapter 6.

Chapter 2

LDPC Code

Low-density parity check (LDPC) code was first introduced by Gallager in the 1960s, but was almost forgotten until Mackay and Neal rediscovered. The most advantage of LDPC codes is it can achieve near Shannon limit error performance. Besides, its algorithm provides very simple arithmetic computations and parallelism to decrease the complexity of hardware design and increase the throughput rate. With these advantages, many applications, such as 802.16e and DVB-S2, have took LDPC codes into account for the forward error correction (FEC) to achieve high-speed and high performance.



2.1 Concept of LDPC

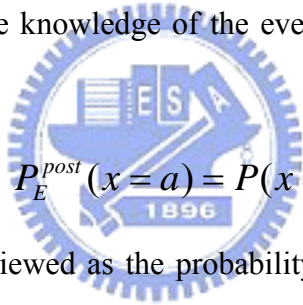
LDPC codes, just a linear block, is constructed by a sparse parity check matrix \mathbf{H} which means there are almost zeros and only a small number of ones in the entries. With the sparse matrix \mathbf{H} , the complexity of computation is reduced in decoding. LDPC codes can be divided into two types, one is regular LDPC code, the other is irregular LDPC code. The regular LDPC codes mean that each row has the same number of ones, and each column does so. For example, in a regular M -by- N LDPC code, there are λ ones in each of the M rows and ρ ones in each of the N columns. The irregular LDPC codes mean the numbers of ones in the rows and the columns are different.

2.1.1 Message Passing Algorithm

LDPC decoding algorithm is based on soft iterative decoding which relies on the message passing algorithm [8][9][16]. Thus, in this section, message passing algorithm which is based on the probabilistic decoding is introduced. For a variable x , there are three important probabilities in message passing algorithm. For the event (or called the constraint) $\{x=a\}$, suppose that E is an event affecting on the variable x . The **intrinsic probability** [10] represents the probability $P(x=a)$ that the variable x takes the value a . So for the variable x with respect to E , its intrinsic probability can be denoted by

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

On the other hand, the **posterior probability** is the conditional probability for the variable x taking the value a based on the knowledge of the event E . The posterior probability can be denoted by



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

The two probabilities can be viewed as the probability **before** and **after** taking the event E into account.

Besides, with Bayes' theorem, the posterior probability can be rewritten as follow :

$$P(x = a | E) = \frac{1}{P(E)} P(E | x = a) P(x = a) \quad (2.3)$$

The term at right-hand side of the equation $P(x = a)$ is the intrinsic probability. The term $P(E | x = a)$ is proportional to the **extrinsic probability**, which describes the probability that the new information for x obtained from the event E . The extrinsic probability can be denoted by

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

ρ_e represents the normalization constant to make the summation of the terms $P_E^{ext}(x = a')$

for $a' \in A$ equals to 1 (i.e., $\sum_{a' \in A} P_E^{ext}(x = a') = 1$), assuming a' takes values from the alphabet set A .

Then the relationship between the intrinsic, extrinsic and posterior probabilities in (2.3) can be rewritten as

$$P_E^{post}(x = a) = \rho_c P_E^{int}(x = a) P_E^{ext}(x = a) \quad (2.5)$$

Where ρ_c is a normalization constant as follow

$$\rho_c = \left(\sum_{a \in A} P_E^{int}(x = a) P_E^{ext}(x = a) \right)^{-1} \quad (2.6)$$

If $A = GF(2)$, $GF(2)$ denotes the set which only has two possible value 0 and 1, also called binary variables. The log-likelihood ratio representation for (2.5) is

$$L^{post}(x) = \ln \frac{P^{post}(x = 1)}{P^{post}(x = 0)} = \ln \frac{P^{int}(x = 1)}{P^{int}(x = 0)} + \ln \frac{P^{ext}(x = 1)}{P^{ext}(x = 0)} = L^{ext}(x) + L^{int}(x) \quad (2.7)$$

In the graph representation, we use a normal graph [11][12] which is an undirected graph, consisting of nodes, ordinary edges and left edges. The nodes denote the constraints and the ordinary edges denote the state variable for message passing and the left edges denote the symbol variables. Fig. 2.1 shows an example with three vertices :

The edges connecting two vertices are ordinary edges, and the edges connecting only one vertex is left edges

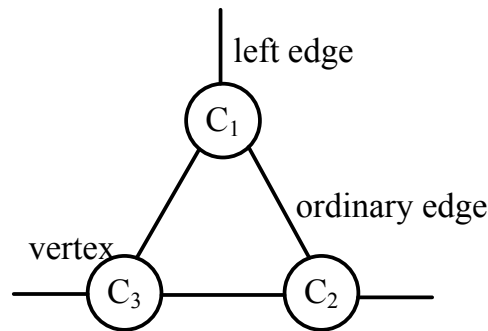


Fig. 2.1 An example of normal graph

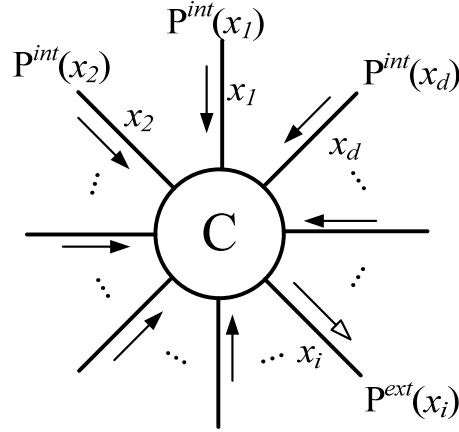


Fig. 2.2 Graph representation of the extrinsic and the intrinsic probabilities

One node

Now consider a single node C , with d edges, as shown in Fig. 2.2. There are $d-1$ left edges.

We define a set S_c which is a subspace of the d -dimensional vector space A^d ($S_c \subset A^d$),

and any d -tuple $\mathbf{x} = (x_1, x_2, \dots, x_d) \in S_c$ will satisfy the constraint C . Each edge has the

intrinsic probability $P^{int}(x_j)$ associated with the symbol x_j for $j = 1 \sim d$, then a posteriori

probability of a symbol x_j with respect to C will be obtained from the combination of the

intrinsic probabilities and the extrinsic probability $P^{ext}(x_i)$. Therefore we have to evaluate

$P^{ext}(x_i)$ based on the constraint C and the intrinsic probabilities $P^{int}(x_j)$ with $j \neq i$. The

extrinsic probability $P^{ext}(x_i)$ is

$$P^{ext}(x_i) = \rho_e P(C | x_i) \quad (2.8)$$

To evaluate the extrinsic probability, we have to evaluate the conditional probability

$P(C | x_i)$, The conditional probability $P(C | x_i)$ can be evaluated as

$$\begin{aligned}
P(C | x_i) &= \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{Sc}}} P(C, \{x_j\}_{j=1}^d | x_i) \\
&= \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{Sc}}} P(C, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d | x_i) \\
&= \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{Sc}}} P(C | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d) P(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d | x_i)
\end{aligned} \tag{2.9}$$

The first term on the right-hand side of (2.9) is always equal to 1 because the constraint C is always true with given $\{x_j\}_{j=1}^d$ where $\{x_j\}_{j=1}^d$ belong to the constraint set \mathbf{Sc} . And the last term on the right-hand side is rewritten based on the independence of the variables $\{x_j\}_{j=1}^d$

$$P(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d | x_i) = \prod_{\substack{j=1 \\ j \neq i}}^d P^{int}(x_j) \tag{2.10}$$

The (2.8) can be rewritten as

$$P^{ext}(x_i) = \rho_e \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{Sc}}} \prod_{\substack{j=1 \\ j \neq i}}^d P^{int}(x_j) \tag{2.11}$$

And the posterior probability can be derived using (2.11) :

$$\begin{aligned}
P^{post}(x_i) &= \rho_c P^{int}(x_i) P^{ext}(x_i) \\
&= \rho_c \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{Sc}}} \prod_{j=1}^d P^{int}(x_j)
\end{aligned} \tag{2.12}$$

Where ρ_c is the normalization constant as

$$\rho_c = \left(\sum_{\substack{x_i \\ \mathbf{x} \in \mathbf{Sc}}} \prod_{j=1}^d P^{int}(x_j) \right)^{-1} \tag{2.13}$$

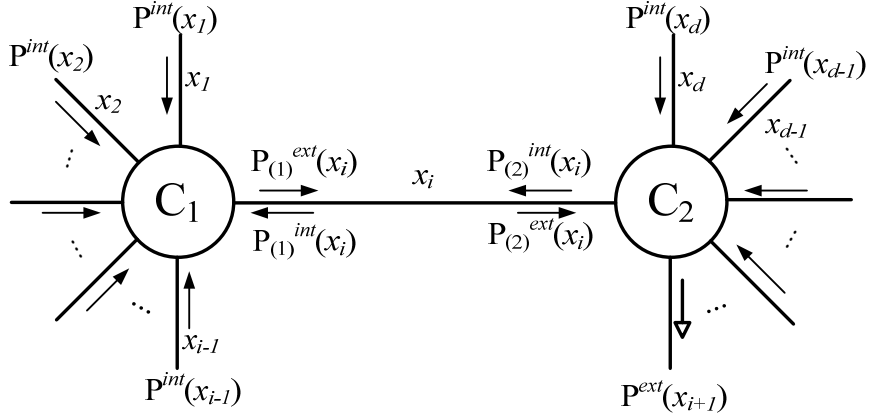


Fig. 2.3 Graph representation of the message passing between two vertices

Two node

Moreover, we consider the graph with two vertices (constraints), C_1 and C_2 , as shown in Fig. 2.3. The constraint C_1 has $i-1$ left edges and one ordinary edge, corresponding to the symbol $x_1 \sim x_{i-1}$ and x_i . On the other hand, $x_i \sim x_d$ are constrained by C_2 where only x_i is on the ordinary edge. Besides, the two vertices are associated to two constrain set, \mathbf{Sc}_1 and \mathbf{Sc}_2 such that any $\mathbf{x}_1 = (x_1, x_2, \dots, x_i) \in \mathbf{Sc}_1$ and $\mathbf{x}_2 = (x_i, x_{i+1}, \dots, x_d) \in \mathbf{Sc}_2$. As shown in Fig. 2.3, the symbol x_{i+1} is considered first, we have to evaluate the extrinsic probability for the left edge based on both C_1 and C_2 . First, we only consider the constrain C_2 , according to the result in (2.11), the extrinsic probability can be rewritten as

$$P^{ext}(x_{i+1}) = \rho_2 P(C_2 | x_{i+1}) = \rho_2 \sum_{\substack{\mathbf{x}_2 \setminus x_{i+1} \\ \mathbf{x}_2 \in \mathbf{Sc}_2}} P_{(2)}^{int}(x_i) \prod_{j=i+2}^d P^{int}(x_j) \quad (2.14)$$

The intrinsic probability $P_{(2)}^{int}(x_i)$ for C_2 in on the ordinary edge that is unable to be acquired from the inputs. Therefore, we evaluate the extrinsic probability based on both constrains C_1 and C_2 .

$$P^{ext}(x_{i+1}) = \rho_e P(C_1, C_2 | x_{i+1}) \quad (2.15)$$

We can rewrite the above probability as

$$\begin{aligned}
P(C_1, C_2 | x_{i+1}) &= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in \mathbf{Sc}_2}} P(C_1, C_2, x_i, x_{i+2}, \dots, x_d | x_{i+1}) \\
&= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in \mathbf{Sc}_2}} P(C_2 | C_1, \mathbf{x}_2) P(C_1, x_i, x_{i+2}, \dots, x_d | x_{i+1}) \\
&= \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in \mathbf{Sc}_2}} P(C_1, x_i, x_{i+2}, \dots, x_d | x_{i+1})
\end{aligned} \tag{2.16}$$

where the second equality comes from a Markov chain

$$P(C_1, C_2 | x_i) = P(C_1 | x_i) P(C_2 | x_i) \tag{2.17}$$

Such that the term

$$P(C_2 | C_1, \mathbf{x}_2) = P(C_2 | \mathbf{x}_2) = 1, \text{ for } \mathbf{x}_2 \in \mathbf{Sc}_2 \tag{2.18}$$

The term on the right-side of (2.16) can be continuously rewritten as

$$\begin{aligned}
P(C_1, x_i, x_{i+2}, \dots, x_d | x_{i+1}) &= P(C_1 | \mathbf{x}_2) P(x_i, x_{i+2}, \dots, x_d | x_{i+1}) \\
&= P(C_1 | x_i) \prod_{\substack{j=i \\ j \neq i+1}}^d P(x_j) \\
&= (\rho_1)^{-1} P_{(1)}^{ext}(x_i) P^{int}(x_i) \prod_{j=i+2}^d P(x_j)
\end{aligned} \tag{2.19}$$

From the Fig. 2.3,

$$P_{(1)}^{ext}(x_i) = \rho_1 P(C_1 | x_i) \tag{2.20}$$

is the extrinsic probability of x_i with respect to C_1 , and the intrinsic probability $P^{int}(x_i)$ is for the ordinary edge variable x_i . Since the ordinary edge connect C_1 and C_2 without any external input, the probability $P^{int}(x_i)$ can be initialized to be a constant. We set

$P^{int}(x_i) = \frac{1}{|\mathbf{A}|}$ for $x_i \in \mathbf{A}$. Therefore, the extrinsic probability in (2.15) can be expressed as

$$P^{ext}(x_{i+1}) = \rho_e' \sum_{\substack{x_2 \setminus x_{i+1} \\ x_2 \in \mathbf{Sc}_2}} P_{(1)}^{ext}(x_i) \prod_{j=i+2}^d P(x_j) \tag{2.21}$$

Where $\rho_e' = \rho_c / \rho_1 |\mathbf{A}|$, from the Fig. 2.3, we can know that

$$P_{(2)}^{int}(x_i) = P_{(1)}^{ext}(x_i) \quad (2.22)$$

if $P_{(1)}^{ext}(x_i)$ is available. So only constrain C2 is necessary for estimating $P^{ext}(x_{i+1})$. In the same way, $P^{ext}(x_j)$ for $j=(i+2) \sim d$ can also be derived. Moreover, $P^{ext}(x_l)$ with $l=1 \sim (i-1)$, the extrinsic probability $P_{(2)}^{ext}(x_i)$ with respected to C₂ is required. And the intrinsic probability is assume that

$$P_{(1)}^{int}(x_i) = P_{(2)}^{ext}(x_i) \quad (2.23)$$

The processes of (2.22) and (2.23) are the message passing between the vertex C₁ and C₂. With the message algorithm, we can simplify the problem of solving both C₁ and C₂ into the problem of solving the single vertex graph. The problem is more simple than the original problem. We concludes the message passed on the edge x_i as follow :

$$\mu_{C_1 \rightarrow C_2}(x_i) = P_{(1)}^{ext}(x_i) = \rho_1 \sum_{\substack{x_1 \setminus x_i \\ x_1 \in \mathbf{Sc}_1}} \prod_{j=1}^{i-1} P(x_j) \quad (2.24)$$

$$\mu_{C_2 \rightarrow C_1}(x_i) = P_{(2)}^{ext}(x_i) = \rho_2 \sum_{\substack{x_2 \setminus x_i \\ x_2 \in \mathbf{Sc}_2}} \prod_{j=i+1}^d P(x_j) \quad (2.25)$$

The operation in the message passing is the sum of products, thus the message passing algorithm is also called the **sum-product algorithm** [13].

Generally, if the graph consisting vertices, C₀, C₁, ..., C_d, the vertex C₀ has d ordinary edges that respectively connect to C₁, C₂, ..., C_d with symbol variables x_1, x_2, \dots, x_d . Assuming the message $\mu_{C_j \rightarrow C_0}(x_j)$ with $j=1 \sim d$ have been derived from C₁ ~ C_d, we can evaluate

$\mu_{C_0 \rightarrow C_i}$ by

$$\mu_{C_0 \rightarrow C_i} = \sum_{\substack{x \setminus x_i \\ x \in \mathbf{Sc}_0}} \prod_{\substack{j=1 \\ j \neq i}}^d \mu_{C_i \rightarrow C_0}(x_j) \quad (2.26)$$

Where \mathbf{S}_{C_0} is the constrain set for C_0 , and $\mathbf{x} = (x_1, x_2, \dots, x_d)$. And the message $\mu_{C_0 \rightarrow C_i}$ for $i=1 \sim d$ can be obtained and become the intrinsic probability inputs for the vertices $C_1 \sim C_d$.

2.1.2 Decoding Concept

Just like linear block codes, the goal of a M-by-N LDPC codes is, given a codeword $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$, to satisfy the equation $\mathbf{H}\mathbf{X} = \mathbf{0}$. LDPC codes can be represented by a Tanner graph [14][15]. Fig. 1 is an illustrative example of a 2×4 parity check matrix \mathbf{H} . There are four bit nodes, B_1, B_2, B_3, B_4 , (also called variable node), which represent the 4-bits codeword $\mathbf{X} = [x_1, x_2, x_3, x_4]^T$, and there are two check nodes, C_1, C_2 , which represent the two parity check equation of \mathbf{H} . The connections between check nodes and bit node means that there are ones at the corresponding positions in the parity check matrix \mathbf{H} . For example, the connection between C_1 and B_1 means that $H_{11} = 1$ in the parity check matrix \mathbf{H} , where H_{mn} denotes the element at the m^{th} row and the n^{th} column of \mathbf{H} . In the thesis, for simplicity, we only consider binary LDPC codes. So every addition actually denotes exclusive-or.

$$\mathbf{H}\mathbf{X} = \mathbf{0} \Rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{cases} x_1 \oplus x_2 = 0 \\ x_1 \oplus x_3 \oplus x_4 = 0 \end{cases} \quad (2.27)$$

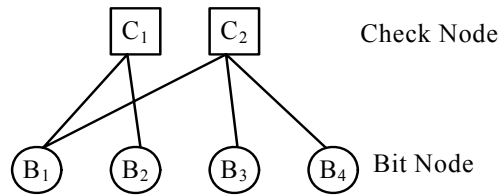


Fig. 2.4 The corresponding Tanner graph

LDPC decoding is based on the belief propagation (BP) algorithm, also called message passing algorithm, which provides an efficient and powerful approach to decode LDPC codes. Each bit node transmits its information to other bit nodes through the check node equation. The erroneous data can possibly be correct with iterative exchanging information between check nodes and bit nodes.

Now we first introduce the LDPC decoding algorithm in the simple view of probability. Take Fig. 1 as example, as Fig. 2, we first update the bit node information through the check node equation. We call this process check node update. Let us consider the first check node Eqn.(2.1) $x_1 \oplus x_2 = 0$. To obtain the probability of x_1 , assume we know the probability of $x_2 = 0$ is q_0 , denotes as $P(x_2 = 0) = q_0$, and $p(x_2 = 1) = q_1$ (The equation $q_0 + q_1 = 1$ is always true). We can know x_1 and x_2 must be the same to satisfy the equation $x_1 \oplus x_2 = 0$. So we can obtain $P(x_1 = 0) = q_0$ and $p(x_1 = 1) = q_1$.

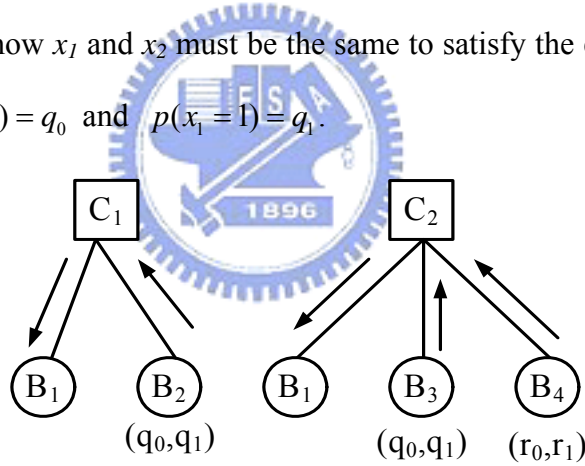


Fig. 2.4 The check node update for B_1

Now we consider the second check node equation $x_1 \oplus x_3 \oplus x_4 = 0$. To get the probability of x_1 , assume we know $P(x_3 = 0) = q_0$, $P(x_3 = 1) = q_1$, $P(x_4 = 0) = r_0$ and $P(x_4 = 1) = r_1$.

Through the second check node equation, we can derive the following equation :

$$\begin{cases} P(x_1 = 0) = P(x_3 \oplus x_4 = 0) = P(x_3 = 0)P(x_4 = 0) + P(x_3 = 1)P(x_4 = 1) = q_0r_0 + q_1r_1 \\ P(x_1 = 1) = P(x_3 \oplus x_4 = 1) = P(x_3 = 1)P(x_4 = 0) + P(x_3 = 0)P(x_4 = 1) = q_1r_0 + q_0r_1 \end{cases} \quad (2.2)$$

Here we define $CHK(q_0, q_1, r_0, r_1) = (q_0r_0 + q_1r_1, q_1r_0 + q_0r_1)$. Then we obtain each bit node's

probability through the check nodes that connects to it.

We will do the next process called bit node update. This process is for bit nodes to gather all probability from the check nodes connecting to them. Fig. 3 is the illustration for bit node update. There are two check nodes connecting to bit node 1, so bit node 1's probability can be calculate as follow :

$$\begin{aligned} P(x_1 = 0) &\propto P(C_1 = 0 \text{ and } x_1 = 0)P(C_2 = 0 \text{ and } x_1 = 0) = q_0 r_0 \\ P(x_1 = 1) &\propto P(C_1 = 0 \text{ and } x_1 = 1)P(C_2 = 0 \text{ and } x_1 = 1) = q_1 r_1 \end{aligned} \quad (2.28)$$

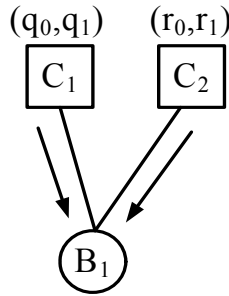


Fig. 2.5 Bit node update for B₁

To ensure $P(x = 0) + P(x = 1) = 1$, we normalize the probability of x_1 . So we can obtain that

$$P(x_1 = 0) = \frac{q_0 r_0}{q_0 r_0 + q_1 r_1} \quad P(x_1 = 1) = \frac{q_1 r_1}{q_0 r_0 + q_1 r_1} \quad (2.29)$$

Here we define

$$VAR(q_0, q_1, r_0, r_1) = \left(\frac{q_0 r_0}{q_0 r_0 + q_1 r_1}, \frac{q_1 r_1}{q_0 r_0 + q_1 r_1} \right) \quad (2.30)$$

2.1.3 Decoding Flow

Based on the above equation we can roughly construct the LDPC decoding algorithm. In order to decrease complexity of computations, we represent the probabilistic messages by Log-Likelihood Ratios (LLR), the LLR is defines as

$$L(U) @ \ln \frac{P(U = 0)}{P(U = 1)} = \ln \lambda \quad (2.31)$$

Then the equation can be rewritten as

$$\begin{aligned}
CHK(U_1, U_2) &= CHK(U_1 \oplus U_2) = \ln \frac{1 + \lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \\
&= \ln \frac{1 + e^{U_1} e^{U_2}}{e^{U_1} + e^{U_2}} = \ln \frac{e^{\frac{U_1+U_2}{2}} + e^{\frac{U_1+U_2}{2}}}{e^{\frac{U_1-U_2}{2}} + e^{\frac{U_1-U_2}{2}}}
\end{aligned} \tag{2.32}$$

$$VAR(U_1, U_2) = \ln(\lambda_1 \lambda_2) = \ln \lambda_1 + \ln \lambda_2 = L_1 + L_2 \tag{2.33}$$

These two equations can be computed in Log-Likelihood Ratios form to reduce the numbers of computation parameters and VAR equation only needs addition operation instead of multiplication and division. If we construct the general form of CHK and VAR , we can get that

$$\begin{aligned}
CHK(U_1, U_2, \dots, U_N) &= CHK(U_1 \oplus U_2 \oplus \dots \oplus U_N) \\
&= CHK(CHK(\dots CHK(CHK(U_1 \oplus U_2) \oplus U_3) \dots) \oplus U_N)
\end{aligned} \tag{2.34}$$

$$\begin{aligned}
VAR(U_1, U_2, \dots, U_N) &= \ln(\lambda_1 \lambda_2 \dots \lambda_N) \\
&= \ln \lambda_1 + \ln \lambda_2 + \dots + \ln \lambda_N = L_1 + L_2 + \dots + L_N
\end{aligned} \tag{2.35}$$

Before we describe the BP decoding algorithm, we define some parameters for simplifying decoding procedure. Take Fig. 4 for example, Fig. 4 is a 4×6 parity check matrix.

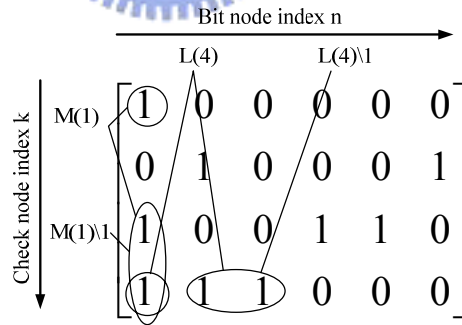


Fig. 2.6 The example of definition for decoding procedure

$M(n)$ denotes the set of check nodes that connect to the bit node n , where n represents the n^{th} column of the parity check matrix, i.e., $M(n)$ represents the positions of “1”s in the n^{th} column. $L(k)$ denotes the set of bit nodes that connect to the check node k , where k represents the k^{th} row of the parity check matrix, i.e., $L(k)$ represent the positions of “1”s in the k^{th} row. $M(n)\setminus k$ denotes the set of $M(n)$ excluding the k^{th} check node, and $L(k)\setminus n$ denotes the set of $L(k)$

excluding the n^{th} bit node. “ $q_{n,k}$ ” denotes the probability of information that the bit node n transmits to the check node k , “ $r_{k,n}$ ” denotes the probability of information that the check node k transmits to the bit node n . BP algorithm is based on iterative decoding procedure, the iterative decoding procedure is shown below.

Initialization

We assume the channel is AWGN channel, BPSK mapping (0 mapped to +1 and 1 mapped to -1) is used. u_n is the channel’s input, and y_n is the channel’s output. The channel transition probabilities are shown below

$$p_n = p(u_n = -1 | y_n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_n+1)^2}{2\sigma^2}}$$

$$(1 - p_n) = p(u_n = +1 | y_n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_n-1)^2}{2\sigma^2}}$$
(2.36)

In the LLR form the probability is rewritten as

$$L(u_n) = \ln \frac{p(u_n = +1 | y_n)}{p(u_n = -1 | y_n)} = \ln e^{-\frac{1}{2\sigma^2}((y_n-1)^2 - (y_n+1)^2)} = \frac{2}{\sigma^2} y_n$$
(2.37)

We set the initial probabilities of $q_{n,k}$ as $L(u_n)$ and $r_{k,n}$ as zero.

Message passing

1st step : check node updating, i.e., information passing from check nodes to bit nodes by collecting the incoming information $q_{n,k}$ ’s. Then we update the probabilities $r_{k,n}$ ’s for the next step.

$$r_{k,n} = \text{CHK} \left(\sum_{n' \in L(k) \setminus n} \oplus q_{n',k} \right)$$
(2.38)

2nd step: bit node updating, i.e., information passing from each bit node to check nodes by collecting the incoming information $r_{k,n}$ ’s. We update the probabilities $q_{n,k}$ ’s for next

iteration decoding and make decision at next step.

$$q_{n,k} = \text{VAR}\left(\text{VAR}_{k' \in M(n) \setminus k}(r_{k',n}), L(n)\right) = L(n) + \sum_{k' \in M(n) \setminus k} r_{k',n} \quad (2.39)$$

3rd step: summing up, for each bit node n, we sum up all information from all the check nodes connecting to the bit node n. We define q_n 's as the summation results.

$$q_n = \text{VAR}\left(\text{VAR}_{k \in M(n)}(r_{k,n}), L(n)\right) = L(n) + \sum_{k \in M(n)} r_{k,n} \quad (2.40)$$

Decision

We decode \hat{u}_n by analyzing q_n , $q_n \geq 0$ represents that the probability of $\hat{u}_n = 1$ is larger than that of $\hat{u}_n = 0$, so we can derive the following equation :

$$\hat{u}_n = \begin{cases} 0 & \text{if } q_n \geq 0 \\ 1 & \text{otherwise} \end{cases} \quad (2.41)$$

According to the \hat{u}_n 's we obtain above, we check whether \hat{u}_n satisfies the parity check equation $\mathbf{H}\hat{\mathbf{u}} = 0$. If yes, \hat{u}_n is a legal codeword, then the iterative decoding stop. If not, return back to the message passing procedure until the legal codeword is obtained or the maximum iteration number is achieved.

2.2 LDPC Code for 802.16e

In WiMAX 802.16e , there are 19 types of block size, from 576 to 2304, each block size is a multiple of 24. There are six types of code rate, including 1/2, 2/3 (A,B), 3/4 (A,B), 5/6. Table 1 shows the 19 types block size and their corresponding parameters. “z factor” represents a shift size factor according to different block sizes. Because LDPC code in 802.16e is a systematic code. Codeword is composed of original information bits and parity check bits. “k” represents the original information size without parity check bits. The users have to adapt

block size and code rate according to the channel situation to achieve high-speed with satisfying the error correction capacity.

Table 2.1 LDPC block size and code rate

n (bit)	n (bytes)	z factor	k (bytes)			
			R=1/2	R=2/3	R=3/4	R=5/6
576	72	24	36	48	54	60
672	84	28	42	56	63	70
768	96	32	48	64	72	80
864	108	36	54	72	81	90
960	120	40	60	80	90	100
1056	132	44	66	88	99	110
1152	144	48	72	96	108	120
1248	156	52	78	104	117	130
1344	168	56	84	112	126	140
1440	180	60	90	120	135	150
1536	192	64	96	128	144	160
1632	204	68	102	136	153	170
1728	216	72	108	144	162	180
1824	228	76	114	152	171	190
1920	240	80	120	160	180	200
2016	252	84	126	168	189	210
2112	264	88	132	176	198	220
2208	276	92	138	184	207	230
2304	288	96	144	192	216	240

2.2.1 Parity Check Matrix Definition

Each LDPC code is defined by a matrix \mathbf{H} of size m -by- n , where n is as previously defined-the length of the code, and m is the number of the number of parity check bits in the code.

The parity check matrix \mathbf{H} is defined as :

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \mathbf{P}_{0,2} & \text{L} & \mathbf{P}_{0,n_b-2} & \mathbf{P}_{0,n_b-1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \text{L} & \mathbf{P}_{1,n_b-2} & \mathbf{P}_{1,n_b-1} \\ \mathbf{P}_{2,0} & \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \text{L} & \mathbf{P}_{2,n_b-2} & \mathbf{P}_{2,n_b-1} \\ \text{L} & \text{L} & \text{L} & \text{L} & \text{L} & \text{L} \\ \mathbf{P}_{m_b-1,0} & \mathbf{P}_{m_b-1,1} & \mathbf{P}_{m_b-1,2} & \text{L} & \mathbf{P}_{m_b-1,n_b-2} & \mathbf{P}_{m_b-1,n_b-1} \end{bmatrix} = \mathbf{P}^{H_b}$$

Fig 2.7 the parity check matrix of LDPC codes

where $\mathbf{P}_{i,j}$ is one of a set of z -by- z permutation matrices or a z -by- z zero matrix. The matrix

\mathbf{H} is expanded from a binary base matrix \mathbf{H}_b of size of m_b -by- n_b . where $m_b = m / z_f$ and

$n_b = n / z_f$, with z_f is the z factor corresponding to the code length. The base matrix \mathbf{H}_b is

expanded by replacing each 1 with a z -by- z permutation matrix and each 0 with a z -by- z zero

matrix. The base matrix size n_b is always equal to 24 and m_b is set according to the code rate

as follow :

Table 2.2 Row number of code rate

Code rate	1/2	2/3	3/4	5/6
m_b	12	8	6	4

The matrix \mathbf{H}_b only has information about whether the $\mathbf{P}_{i,j}$ is a permutation matrix or a

zero matrix, it doesn't contain shift size of permutation matrix. The permutations are circular

right shift, and the set of permutation matrices contains the z -by- z identity matrix and circular

right shifted of the identity matrix. Each permutation matrix is specified a single circular right

shift factor, so the binary base matrix information and permutation shift information can be

combined into a compact model matrix \mathbf{H}_{bm} . The matrix \mathbf{H}_{bm} is the same size as the binary

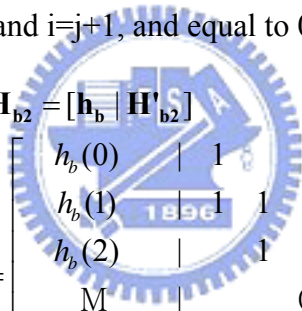
base matrix \mathbf{H}_b , and each entry (i,j) of \mathbf{H}_b is replaced by the permutation information to generate the model matrix \mathbf{H}_{bm} . Each 0 in \mathbf{H}_b is replaced by a negative value (e.g. by -1) to denote a z -by- z zero matrix and Each 1 in \mathbf{H}_b is replaced by a circular shift factor $P(i, j)$ ($P(i, j)$ is a positive integer).

\mathbf{H}_b can be partitioned into two parts- \mathbf{H}_{b1} and \mathbf{H}_{b2} , where \mathbf{H}_{b1} corresponds to the systematic bits and \mathbf{H}_{b2} corresponds to the parity check bits as follow:

$$\mathbf{H}_b = [(\mathbf{H}_{b1})_{m_b \times k_b} \mid (\mathbf{H}_{b2})_{m_b \times m_b}] \quad (2.42)$$

where $k_b = k / z_f$.

\mathbf{H}_{b2} also can be partitioned into two parts- \mathbf{h}_b and \mathbf{H}'_{b2} , where vector \mathbf{h}_b has odd weight and \mathbf{H}'_{b2} has a dual-diagonal structure, where the matrix element (i,j) (i denotes row and j denotes column) is equal to 1 when $i=j$ and $i=j+1$, and equal to 0 elsewhere:



$$\mathbf{H}_{b2} = [\mathbf{h}_b \mid \mathbf{H}'_{b2}] = \begin{bmatrix} h_b(0) & | & 1 & & & \\ h_b(1) & | & 1 & 1 & & 0 \\ h_b(2) & | & & 1 & & \\ \vdots & | & & & \ddots & \\ M & | & & & 0 & \\ \vdots & | & & & & \ddots \\ M & | & & & 0 & 0 & 1 \\ h_b(m_b-1) & | & & & & & 1 & 1 \end{bmatrix}$$

Fig. 2.8 The definition of \mathbf{H}_{b2}

The matrix \mathbf{h}_b has a characteristic that $h_b(0)$ and $h_b(m_b-1)$ are equal to 1, one of $[h_b(1), h_b(2), \dots, h_b(m_b-1)]$ is equal to 1, and others are equal to 0.

In model matrix \mathbf{H}_{bm} , each 1 in \mathbf{H}'_{b2} has a shift size of 0, indicating that there is a z -by- z identity matrix when expanding to \mathbf{H} , and $h_b(0)$ and $h_b(m_b-1)$ have the same shift sizes.

The base model matrix \mathbf{H}_{bm} is defined for the largest code length ($n=2304$) of all code rate. For other code lengths, the shift sizes have to be changed according to the code length. The set of shifts $P(i, j)$ in the base model matrix \mathbf{H}_{bm} are still used for other code length of

the same code rate.

For code rates 1/2 code, 2/3A and B code, 2/3B code and 5/6 code, the shift sizes $\{P(f, i, j)\}$ for a code size corresponding to the expansion factor z_f are derived from $P(i, j)$ by scaling $P(i, j)$ proportionally:

$$p(f, i, j) = \begin{cases} p(i, j), & p(i, j) \leq 0 \\ \left\lfloor \frac{p(i, j)z_f}{z_0} \right\rfloor, & p(i, j) > 0 \end{cases} \quad (2.43)$$

Where f denotes the index of 19 types of code length for a giving code rate, $f=0,1,2,\dots,18$, $f=0$ denotes the largest code length ($n=2304$). z_f denotes the z factor corresponding to the code length, so z_0 represents the z factor of the largest code length ($n=2304$) and is set to 96. Besides, the operation $\lfloor x \rfloor$ denotes that it only gets the integer part of x when x is positive.

For code rate 2/3A code, the shift sizes $\{P(f, i, j)\}$ for a code size corresponding to the expansion factor z_f are derived from $P(i, j)$ by using a modulo function:

$$p(f, i, j) = \begin{cases} p(i, j), & p(i, j) \leq 0 \\ \text{mod}(p(i, j), z_f), & p(i, j) > 0 \end{cases} \quad (2.44)$$

2.2.2 LDPC Encoder

For efficiency and memory saving, LDPC encoder generates codeword with parity check matrix \mathbf{H} instead of generator matrix \mathbf{G} by Richardson [7]. Because parity check matrix \mathbf{H} is in an approximate lower triangular form, so the matrix can be written in the form:

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

where \mathbf{A} is $(m-z) \times k$, \mathbf{B} is $(m-z) \times z$, \mathbf{T} is $(m-z) \times (m-z)$, \mathbf{C} is $z \times k$, \mathbf{D} is $z \times z$ and finally \mathbf{E} is $z \times (m-z)$, here z is the same as z factor, the $\begin{pmatrix} \mathbf{B} \\ \mathbf{D} \end{pmatrix}$ and \mathbf{D} denote the expansion of \mathbf{h}_b and $\mathbf{h}_b(m_b-1)$ respectively, the Fig. 2.9 shows the parity check matrix composition.

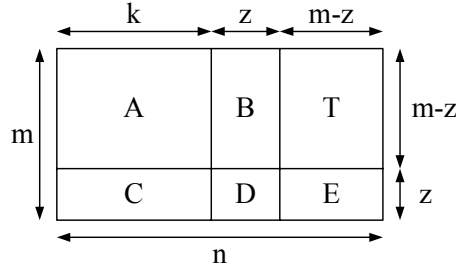


Fig. 2.9 Decomposition of parity check matrix

Let codeword $\mathbf{v} = [\mathbf{u}, \mathbf{p}_1, \mathbf{p}_2]$, where \mathbf{u} denotes the systematic part-original message, \mathbf{p}_1 and \mathbf{p}_2 denote the parity part, \mathbf{p}_1 has length z and \mathbf{p}_2 has length $(m-z)$. According to the definition of parity check matrix \mathbf{H} , each codeword of \mathbf{H} must satisfy the equation $\mathbf{H} \cdot \mathbf{v}^T = 0$. So the equation can be rewritten as follow:

$$\begin{cases} Au^T + Bp_1^T + Tp_2^T = 0 \text{ L L L L L L } (1) \\ Cu^T + Dp_1^T + Ep_2^T = 0 \text{ L L L L L L } (2) \end{cases} \quad (2.45)$$

According to equation (1), p_2^T can be rewritten as $p_2^T = T^{-1}(Au^T + Bp_1^T)$ and replace p_2^T in the equation (2), then the equation (2) becomes

$$(ET^{-1}A + C)u^T + (ET^{-1}B + D)p_1^T = 0 \quad (2.46)$$

Because $ET^{-1}B + D$ is an identity matrix, so p_1^T can be derived from

$$p_1^T = (ET^{-1}A + C)u^T \quad (2.47)$$

And p_2^T can be derived from $p_2^T = T^{-1}(Au^T + Bp_1^T)$ (2.48)

The fig is the block diagram of encoder architecture:

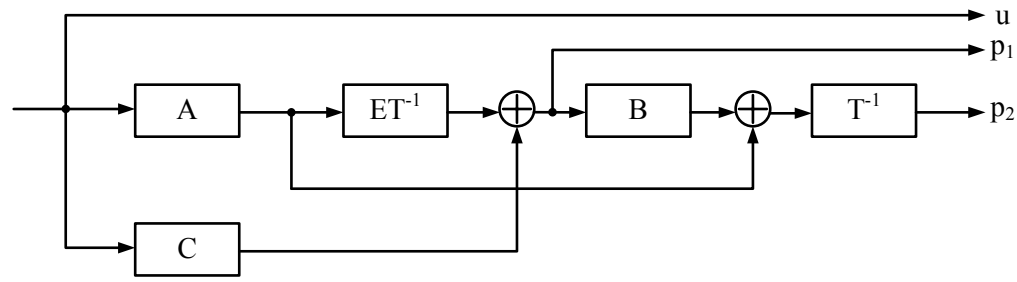


Fig. 2.10 The architecture of encoder for LDPC codes

All the matrix that the encoder needs can be obtained from parity check matrix, only T^{-1} seems hard to obtain. But T is a dual diagonal matrix so it has a characteristic that T^{-1} is a lower triangular matrix. This characteristic can be easily verified.

2.2.3 Implementation Bottleneck

To construct the LDPC decoder for 802.16e, there are some problems should be considered :

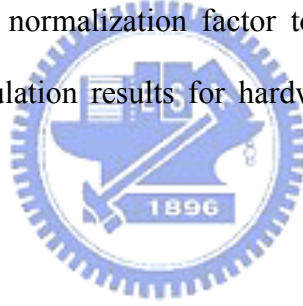
- 1、 There are 5 code rates, according to different code rates, the shift size tables are different, and the row numbers are different, too. Thus, to construct a LDPC decoder that can change modes by a simple control logic and share the computation units in an efficient method is an important task.
- 2、 For the same mode, there are 19 block lengths, according to different block lengths, the compositions of the parity check matrices have different z factor sub-matrices, it is important to decrease the area for cyclic shifters when doing permutation.
- 3、 The computation of check node is difficult in hardware implementation. To simplify the computation in a easier form to reduce chip area and decoding latency and in the same condition to maintain the decoding performance is the most important task.

The maximum iteration number is also an important index while considering the latency of the decoding flow. A smaller maximum iteration number can have better throughput rate. In conventional LDPC decoding algorithm, it takes larger maximum iteration number to achieve the required bit-error-rate performance. By adopting an improved algorithm, it only needs smaller maximum iteration number to achieve the same performance.

Chapter 3

Algorithm Optimization for Implementation

In Chapter 2 we know that in the LDPC decoding algorithm, the check node update processing occupies the most calculation latency. In this chapter, we analyze the check node update equation based on message passing algorithm and at the view of probability to form an approximation equation called **Min-sum algorithm** [25] that is much easier to implement. Besides, we adopt a dynamic normalization factor to improve the decoding performance. Finally, we present some simulation results for hardware implementation and performance comparison.



3.1 Min-Sum Algorithm

Consider the check node update process, Fig. 3.1 is the check node with d degrees, the constrain with respect to the check node is

$$\mathbf{Sc}_j = \{(x_1, x_2, \dots, x_d) \mid x_1 + x_2 + \dots + x_d = 0\} \quad (3.1)$$

If we want to evaluate the message with respect to the constrain for the edge x_i , the message should be

$$\mu_{c_j \rightarrow x_i}(x_i) = P^{ext}(x_i) = P(x_1 + \dots + x_{i-1} + x_{i+1} + \dots + x_d = -x_i) \quad (3.2)$$

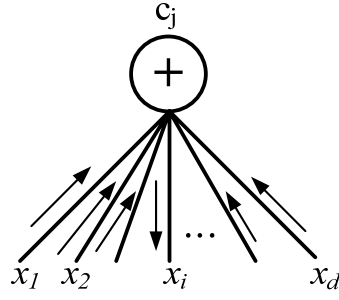


Fig. 3.1 The check node with degree d

To derive the (3.2), we consider the following equation first

$$\begin{aligned} P(x_1 + x_2 = 0) &= P^{int}(x_1 = 0)P^{int}(x_2 = 0) + P^{int}(x_1 = 1)P^{int}(x_2 = 1) \\ &= (1 - p_1)(1 - p_2) + p_1 p_2 \end{aligned} \quad (3.3)$$

Where p_i denotes the intrinsic probability $P^{int}(x_i = 1)$ and (3.3) can be expressed in other form

$$2P(x_1 + x_2 = 0) - 1 = (1 - 2p_1)(1 - 2p_2) \quad (3.4)$$

If we assume the equation

$$\begin{aligned} 2P(x_1 + x_2 + \dots + x_j = 0) - 1 &= 2\Pi_j - 1 \\ &= (1 - 2p_1)(1 - 2p_2)\dots(1 - 2p_j) \\ &= \prod_{i=1}^j (1 - 2p_i) \end{aligned} \quad (3.5)$$

is true. The following equation can be derived by

$$\begin{aligned} \Pi_{j+1} &= P(x_1 + x_2 + \dots + x_j + x_{j+1} = 0) \\ &= P(x_1 + x_2 + \dots + x_j = 0)P(x_{j+1} = 0) + P(x_1 + x_2 + \dots + x_j = 1)P(x_{j+1} = 1) \\ &= \Pi_j(1 - p_{j+1}) + (1 - \Pi_j)p_{j+1} \end{aligned} \quad (3.6)$$

According to (3.6), we can derive

$$\begin{aligned} 2\Pi_{j+1} - 1 &= 2[\Pi_j(1 - p_{j+1}) + (1 - \Pi_j)p_{j+1}] - 1 \\ &= (2\Pi_j - 1)(1 - 2p_{j+1}) \\ &= \prod_{i=1}^{j+1} (1 - 2p_i) \end{aligned} \quad (3.7)$$

By induction, we conclude from (3.7) that

$$\begin{aligned}\Pi_d &= P(x_1 + x_2 + \dots + x_d = 0) \\ &= \frac{1}{2} \left[1 + \prod_{i=1}^d (1 - 2p_i) \right]\end{aligned}\quad (3.8)$$

Then (3.2) can be obtained from

$$\mu_{c_j \rightarrow x_i}(x_i = 0) = \frac{1}{2} \left[1 + \prod_{l=1, l \neq i}^d (1 - 2\mu_{x_i \rightarrow c_j}(x_l = 1)) \right] \quad (3.9)$$

$$\mu_{c_j \rightarrow x_i}(x_i = 1) = \frac{1}{2} \left[1 - \prod_{l=1, l \neq i}^d (1 - 2\mu_{x_i \rightarrow c_j}(x_l = 1)) \right] \quad (3.10)$$

Where the probability $\mu_{x_i \rightarrow c_j}(x_l = 1)$ is the message from the bit node x_i as Fig. 3.2.

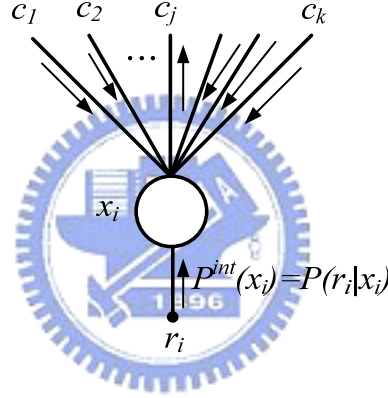


Fig. 3.2 The bit node with degree k

It received the message from the check nodes connecting to it excluding c_j and sent the

message $\mu_{x_i \rightarrow c_j}(x_i)$ to the check node c_j as

$$\mu_{x_i \rightarrow c_j}(x_i = 0) = \rho_b g P^{int}(x_i = 0) \prod_{l=1, l \neq j}^k \mu_{c_l \rightarrow x_i}(x_i = 0) \quad (3.11)$$

$$\mu_{x_i \rightarrow c_j}(x_i = 1) = \rho_b g P^{int}(x_i = 1) \prod_{l=1, l \neq j}^k \mu_{c_l \rightarrow x_i}(x_i = 1) \quad (3.12)$$

Where

$$\rho_b = \sum_{x_i} P^{int}(x_i = 1) \prod_{l=1, l \neq j}^k \mu_{c_l \rightarrow x_i}(x_i = 1) \quad (3.13)$$

The intrinsic probability $P^{int}(x_i)$ comes from the received symbol r_i .

For simplifying the equation, we use log-likelihood ratio to represent the messages. The ratio is defined before as (2.6)

$$L(x) = \ln \frac{P(x=0)}{P(x=1)} = \ln \frac{1-P(x=1)}{P(x=1)}$$

Rewriting the above equation it will become

$$P(x=1) = \frac{1}{e^{L(x)} + 1} \quad (3.14)$$

Then we can write

$$1 - 2P(x=1) = \frac{e^{L(x)} - 1}{e^{L(x)} + 1} = \tanh\left(\frac{L(x)}{2}\right) \quad (3.15)$$

Where the hyperbolic tangent is defined as

$$\tanh\left(\frac{x}{2}\right) = \frac{e^x - 1}{e^x + 1} \quad (3.16)$$

According to (3.15), the (3.9) and (3.10) can be reformulated with log-likelihood ratio as

$$\begin{aligned} L_{c_j \rightarrow x_i}(x_i) &= \ln \frac{1 + \prod_{l=1, l \neq i}^d (1 - 2\mu_{x_i \rightarrow c_j}(x_l = 1))}{1 - \prod_{l=1, l \neq i}^d (1 - 2\mu_{x_i \rightarrow c_j}(x_l = 1))} \\ &= \ln \frac{1 + \prod_{l=1, l \neq i}^d \tanh\left(\frac{L_{x_i \rightarrow c_j}(x_l)}{2}\right)}{1 - \prod_{l=1, l \neq i}^d \tanh\left(\frac{L_{x_i \rightarrow c_j}(x_l)}{2}\right)} \\ &= 2 \tanh^{-1}\left(\prod_{l=1, l \neq i}^d \tanh\left(\frac{L_{x_i \rightarrow c_j}(x_l)}{2}\right)\right) \end{aligned} \quad (3.17)$$

Where the inverse hyperbolic tangent is

$$\tanh^{-1}(y) = \frac{1}{2} \ln \frac{1+y}{1-y} \quad (3.18)$$

Furthermore, we define a new function for $x > 0$

$$\Psi(x) = \Psi^{-1}(x) = \ln \frac{1+e^{-x}}{1-e^{-x}} = -\ln\left(\tanh\left(\frac{x}{2}\right)\right) \quad (3.19)$$

Then we decompose the term in (3.17)

$$\begin{aligned}
\prod_{l=1, l \neq i}^d \tanh\left(\frac{L_{x_l \rightarrow c_j}(x_l)}{2}\right) &= \prod_{l=1, l \neq i}^d A_l \\
&= \left(\prod_{l=1, l \neq i}^d \text{sign}(A_l)\right) \exp\left(\sum_{l=1, l \neq i}^d \ln|A_l|\right) \\
&= \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l))\right) \exp\left(\sum_{l=1, l \neq i}^d \ln\left[\tanh\left(\frac{|L_{x_l \rightarrow c_j}(x_l)|}{2}\right)\right]\right)
\end{aligned} \tag{3.20}$$

The sign magnitude of A_l is the same with $L_{x_l \rightarrow c_j}(x_l)$. And we note that for any integer t

$$(-1)^t \Psi^{-1}(x) = \ln \frac{1 + (-1)^t e^{-x}}{1 - (-1)^t e^{-x}} \tag{3.21}$$

Then we replace the x in (3.21) as

$$x = -\sum_{l=1, l \neq i}^d \ln\left[\tanh\left(\frac{|L_{x_l \rightarrow c_j}(x_l)|}{2}\right)\right] \tag{3.22}$$

(3.17) is rewritten as

$$\begin{aligned}
L_{c_j \rightarrow x_i}(x_i) &= \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l))\right) \Psi^{-1}\left(-\sum_{l=1, l \neq i}^d \ln\left[\tanh\left(\frac{|L_{x_l \rightarrow c_j}(x_l)|}{2}\right)\right]\right) \\
&= \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l))\right) \Psi^{-1}\left(\sum_{l=1, l \neq i}^d \Psi(|L_{x_l \rightarrow c_j}(x_l)|)\right)
\end{aligned} \tag{3.23}$$

Compared to (3.17), the multiplications are replaced with additions, it is easier for

implementation. And the message from bit node x_i to check node c_j can be represented as

$$\begin{aligned}
L_{x_i \rightarrow c_j}(x_i) &= \ln \frac{\rho_b g P^{int}(x_i = 0) \prod_{l=1, l \neq j}^k \mu_{c_l \rightarrow x_i}(x_i = 0)}{\rho_b g P^{int}(x_i = 1) \prod_{l=1, l \neq j}^k \mu_{c_l \rightarrow x_i}(x_i = 1)} \\
&= L^{int}(x_i) + \sum_{l=1, l \neq j}^k L_{c_l \rightarrow x_i}(x_i)
\end{aligned} \tag{3.24}$$

For hardware implementation, the function $\Psi(x)$ in (3.23) is often constructed by the table look-up approach because the operation of $\Psi(x)$ is too complex.

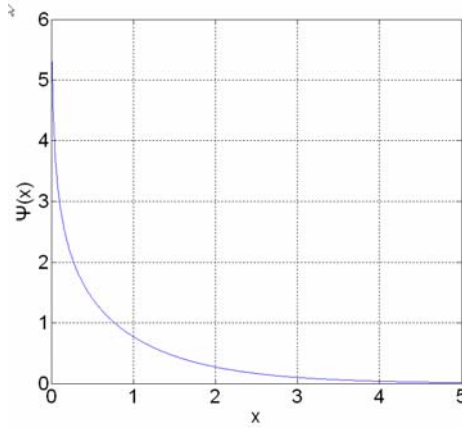


Fig. 3.3 Plot of the $\Psi(x)$ function

Then we analyze the function $\Psi(x)$ as Fig. 3.3, from Fig. 3.3, we can find the property of $\Psi(x)$ that the smaller x has larger result of $\Psi(x)$. In the (3.23), the summation will be dominated by the smaller $|L_{x_l \rightarrow c_j}(x_l)|$, therefore (3.23) can be simplified by an approximate equation as

$$L_{c_j \rightarrow x_i}(x_i) \approx \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l)) \right) \min_{l \in L(j) \setminus i} |L_{x_l \rightarrow c_j}(x_l)| \tag{3.25}$$

The set $L(j) \setminus i$ is defined the same as section 2.1.3. The decoding procedure based on (3.24) and (3.25) is referred to **min-sum algorithm**. We can implement the check node update with the comparison unit instead of table look-up method. But it will lose some bit-error-rate (BER) performance if we adopt (3.25) compared to (3.23). There are some popular compensating

methods for min-sum algorithm such as offset compensation [17][18][19] and normalization compensation [20][21][22] :

$$L_{c_j \rightarrow x_i}(x_i) \approx \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l)) \right) \min_{l \in L(j) \setminus i} |L_{x_l \rightarrow c_j}(x_l)| - \alpha \quad (3.26)$$

or

$$L_{c_j \rightarrow x_i}(x_i) \approx \beta \times \left(\prod_{l=1, l \neq i}^d \text{sign}(L_{x_l \rightarrow c_j}(x_l)) \right) \min_{l \in L(j) \setminus i} |L_{x_l \rightarrow c_j}(x_l)| \quad (3.27)$$

where α and β are compensation factor with $\alpha > 0$ and $0 < \beta \leq 1$. The (3.27) often has better performance than (3.26) because the compensation factor α in (3.26) is constant, it won't change depending on the value of $|L_{x_l \rightarrow c_j}(x_l)|$. If we don't know the range of

$|L_{x_l \rightarrow c_j}(x_l)|$, we can't derive a optimum α . Considering the compensation factor β in

(3.27), although it is constant, but the operation is multiplication, the result will depend on the value of $|L_{x_l \rightarrow c_j}(x_l)|$, so we can derive a optimum value for β . For the same check node, we provide two different values β for the compensation.

Compared with the Eqn.(2.32) , Eqn.(3.17), Eqn.(3.23), The last three equations need complex computation units to derive the exponential function, the hyper-tangent function, the $\Psi(x)$. The look-up table (LUT) is mostly used to derive the approximation value to reduce the computation units. But the LUT's size determined the correcting capability of these function, if we use a detailed LUT for deriving the approximation value, the size of LUT occupies a lot of area, otherwise, if we use a gross LUT, the error with respect to the theoretical increase and we will loss BER performance. Min-sum algorithm using normalization factor can almost achieve the same BER performance with the theoretical one with a sorting unit to find the minimum value of relative input data. The area for check node updating can be saved a lot.

3.2 Simulation Results

In this section, we will present the simulation results and some parameters setting for implementation. The simulation environment is set by the C code. All the simulation results are signal-to-noise ratio (SNR) versus BER through changing some parameters required for implementation.

We set the simulation environment is BPSK modulation and AWGN noise channel, according to (3.17), we can derive the theoretical performance for 802.16e, Fig. 3.4 illustrate the SNR versus BER.

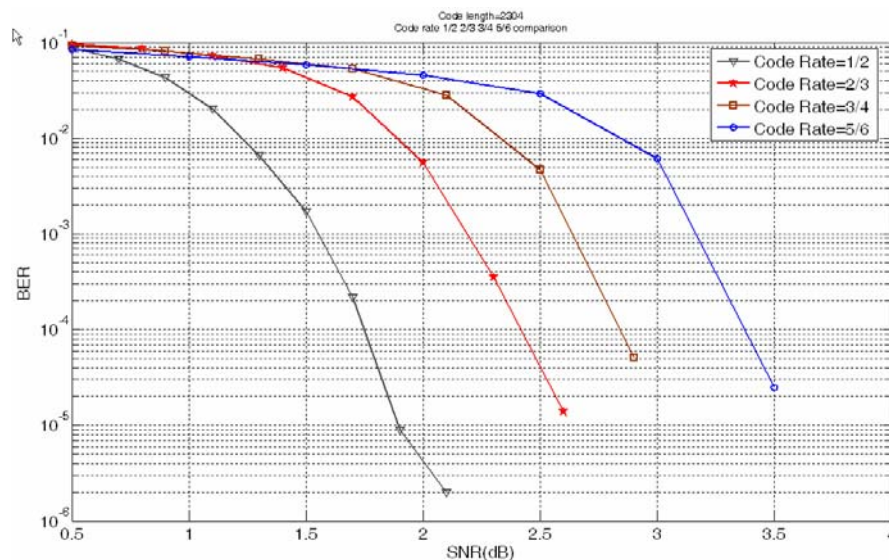


Fig. 3.4 Simulation result (1): theoretical value for maximum code length

The simulation result is based on the maximum iteration number=20 and compare the code 1/2、2/3B、3/4A and 5/6 of the maximum code length=2304, all code rates can achieve $BER = 10^{-5}$ before $SNR = 3.5dB$.

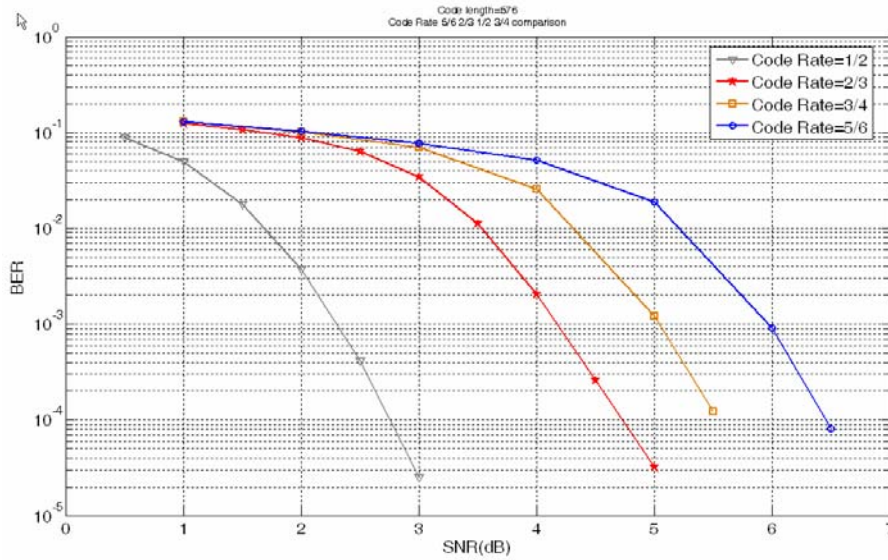


Fig. 3.5 Simulation result (2): theoretical value for minimum code length

The Fig. 3.5 shows the simulation result based on the minimum code length=576, other parameter setting is the same with Fig. 3.4. From Fig. 3.3 and 3.4, we can derive that the parity check matrices defined in 802.16e can provide the performance of $BER = 10^{-4}$ between $SNR = 2 \sim 7dB$

Now for optimization we use min-sum algorithm with different normalize factor, the Fig. 3.5 shows the simulation result. At the simulation we take code rate=1/2 and code length=576, the iteration number is 20.

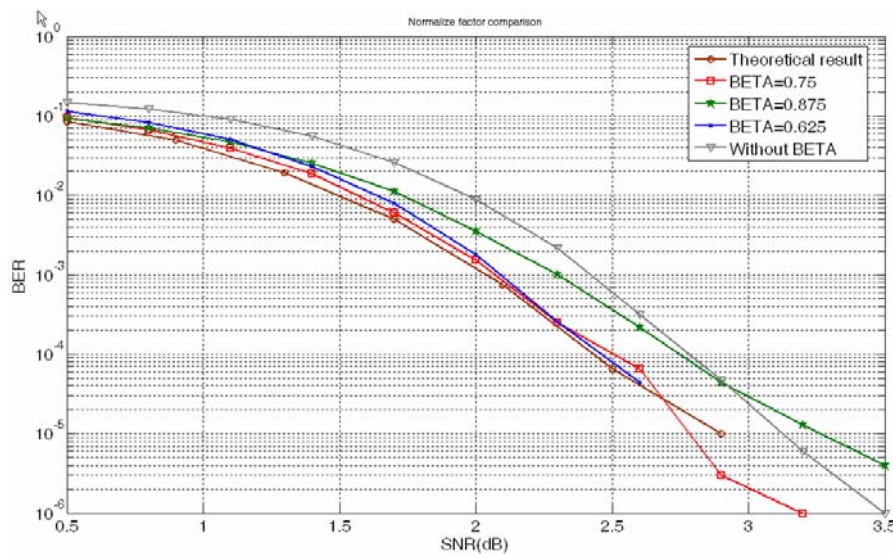


Fig. 3.6 Simulation result (3): Normalization factor comparison

For easier implementation, we choose normalize factor based on the power of 2 such as 0.75、0.625 and 0.875. We can see that the normalize factor=0.75 can achieve almost the theoretical result, and so as normalize factor=0.625.

For implementation, we have to process fix-point simulation to simulate the hardware processing, The number of bits needed to present the message also requires simulation to derive. We simulate the code length=576 and provide min-sum algorithm with normalization factor=0.75 and iteration number is 20. First we fix the fraction part at 1bit and simulate with different integer part bits shown as Fig. 3.6 and from the figure we can find that the integer part bits equal to 4bits is the best choice. Then we fix the integer part at 4bits and simulate different fraction part bits shown as Fig. 3.7. And from the figure the fraction part has best choice of 2bits.

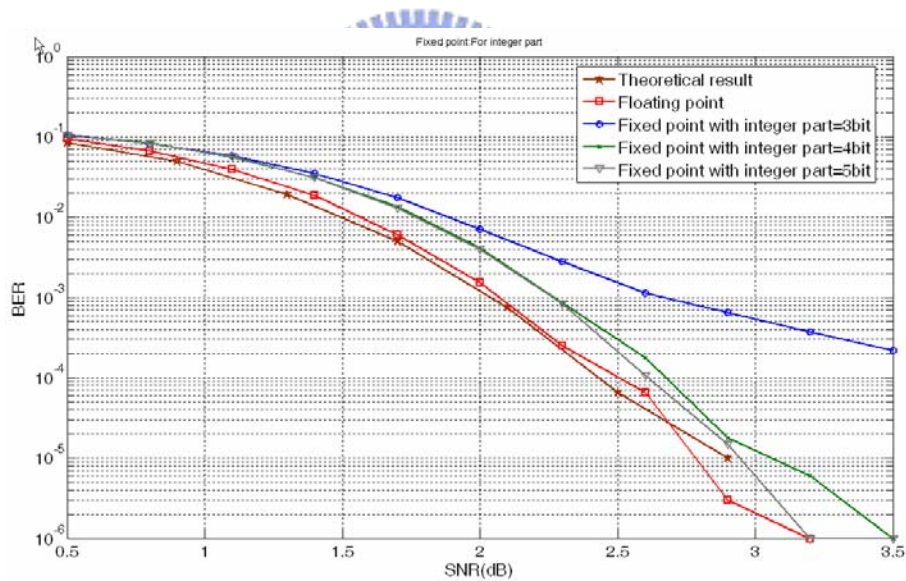


Fig. 3.7 Simulation result (4): Fixed-point simulation for integer part

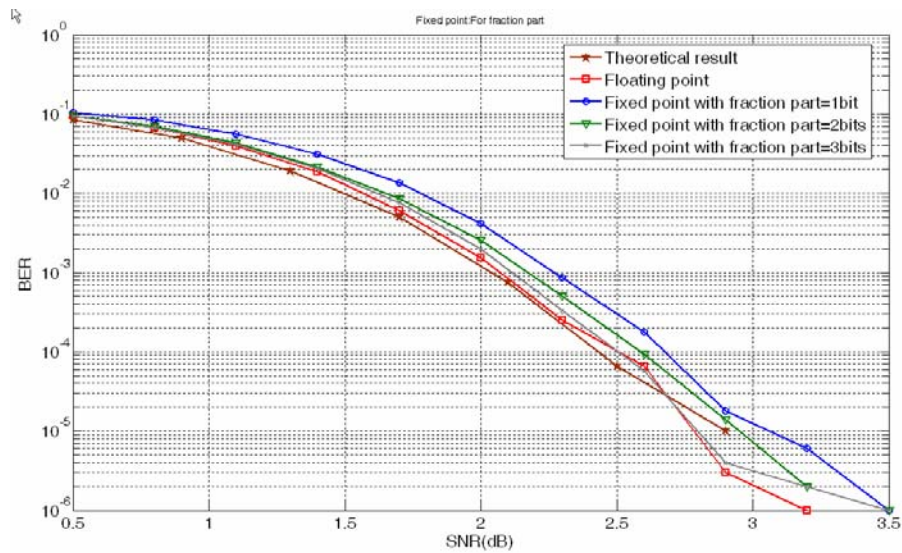


Fig. 3.8 Simulation result (5): Fixed-point simulation for fraction part

The maximum iteration number will affect the performance of BER. But the performance will saturate with the maximum iteration number becomes larger because the dependency of message. After iterative decoding, the assumption of dependency won't be always true, then the decoding equation we use will has some error with respect to the actual condition. Fig. 3.8 shows the fixed point simulation with different maximum iteration number. The simulation parameter is based on the maximum code length=2304 of code rate 1/2. We can find that the iteration number=20 is a good choice between latency and performance.

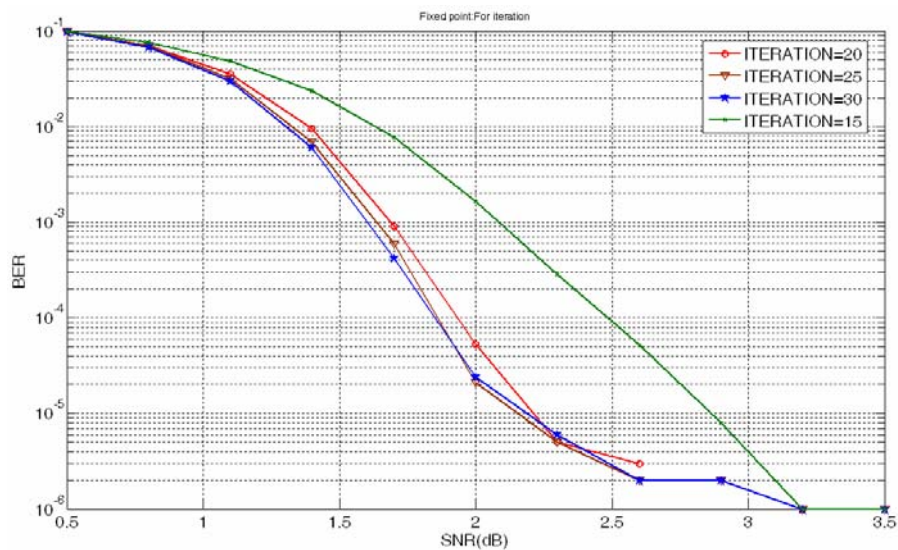


Fig. 3.9 Simulation result (6): Fixed-point simulation for iteration

According to the simulation results above, we set out proposed architecture based on the parameter as Table 3.1

Table 3.1 Parameter setting for implementation

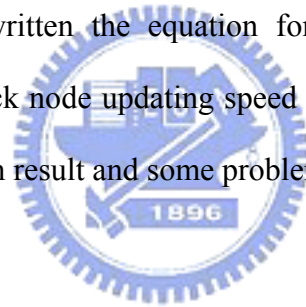
Maximum iteration number		20
Normalization factor		0.75
Bit length	Integer	4 bits
	Fraction	2 bits



Chapter 4

Architecture Design and Circuit Implementation

According to our analysis and simulation, we propose a LDPC decoder architecture for 802.16e. In this LDPC decoder architecture we propose a hierarchical cyclic shifter block for cyclic shifter. Besides, with the characteristic of LDPC decoding algorithm, the memory arrangement is also an important point for improvement. To reduce the complexity of computation element, we rewritten the equation for check node updating and bit node updating to accelerate the check node updating speed to reduce the memory usage. Final we will present the implementation result and some problems we met at backend APR process.



4.1 Decoder Design

4.1.1 Architecture Overview

According to the LDPC decoding flow shown in Fig. 4.1, the Fig. 4.2 is the block diagram of the LDPC decoder, the decoder is partitioned into several block. Each block processes based on the sub-matrix defined in the Section 2.2.1.

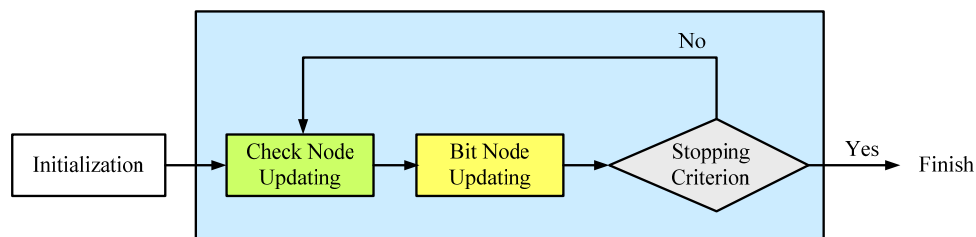


Fig. 4.1 LDPC decoding flow

The decoder is designed based on the maximum input number 2304 and the maximum column number 1152 (code rate=1/2) to support all types of LDPC code in 802.16e. Each process is controlled by the decoder controller. The Input Buffer stores the input data and for every Z_f input data received, the Input Buffer will store the data into Channel Value Memory. Beside, it also stores the data into the B2C memory for the first iteration check node updating. The C2B update block reads data from the B2C memory to process check node updating. The check node output registers store the check node updating result. Then the B2C update block reads data from the C2B registers and stores the results at B2C memory. The iteration repeats between C2B update and B2C update, the iteration decoding processes until the maximum iteration is reached. Final, the Decision block decides whether the data is one (or zero) based on the result of B2C update. The detail process and architecture is explained at the following sub section.

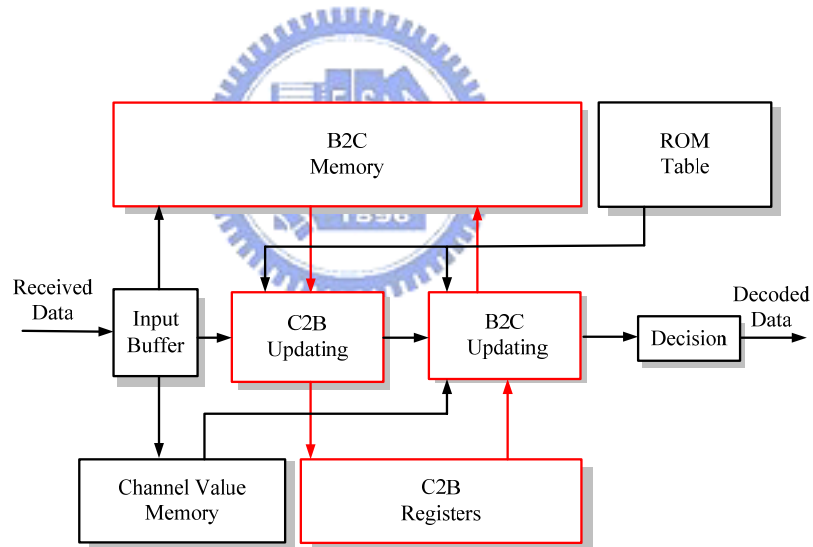


Fig. 4.2 LDPC decoder block diagram

4.1.2 Iterative Decoding Block

In this section we will introduce the proposed iterative decoding block architecture, in order to save memory usage, some computations of LDPC decoding algorithm are moved to different block, final we will explain the overall architecture for this block.

C2B block (I) and C2B registers

The C2B block processes check node updating, we choose Min-Sum algorithm with normalized scaling method. According to the Min-Sum algorithm, the minimum value of bit node message determines the absolute value of the result. We can use normalized scaling method to compensate the error relative to the accurate result. The equation is denoted as following :

$$r_{k,n} \approx \beta \prod_{n' \in L(k) \setminus n} \text{sign}(q_{n',k}) \min(q_{n',k}) \quad 0 < \beta \leq 1 \quad (4.1)$$

For each $r_{k,n}$, the component of $L(k) \setminus n$ is $L(k)$ excluding n . According to the above equation, with the row degree t , we have to sort the $q_{n',k}$ where $n' \in L(k) \setminus n$ to find the minimum value of them t times for the same k of $r_{k,n}$ s, the latency is very long and the control logic is very complicated. We find that, for different $r_{k,n}$ s, their components of $L(k) \setminus n$ are almost the same. So we can sort all the elements of the $L(k)$ first, with the sorting result we will rapidly find each n of $r_{k,n}$ at more simple control logic. The algorithm is present as below :

$$r_{k,n} = \begin{cases} \beta m_1 \prod_{n \in \{L(k)\}} \text{sgn}(q_{n,k}) & \text{if } |q_{n,k}| \neq m_1 \\ \beta m_2 \prod_{n \in \{L(k)\}} \text{sgn}(q_{n,k}) & \text{otherwise} \end{cases} \quad (4.2)$$

Where m_1 denotes the minimum value of $q_{n,k}$ in the set of $L(k)$, and m_2 denotes the second minimum value of $q_{n,k}$ in the set of $L(k)$. With this algorithm, we sort all the $q_{n,k}$ of the sets $L(k)$ one time, we only have to find out the minimum value and second minimum value.

Before introduce the sorting architecture, we present the memory access schedule for check node updating. By the same method with channel value memory arrangement, we divided

memory bank the same as sub-matrix defined in the section 2.2.1, the sub-matrix is based on the maximum matrix which is a 96×96 matrix.

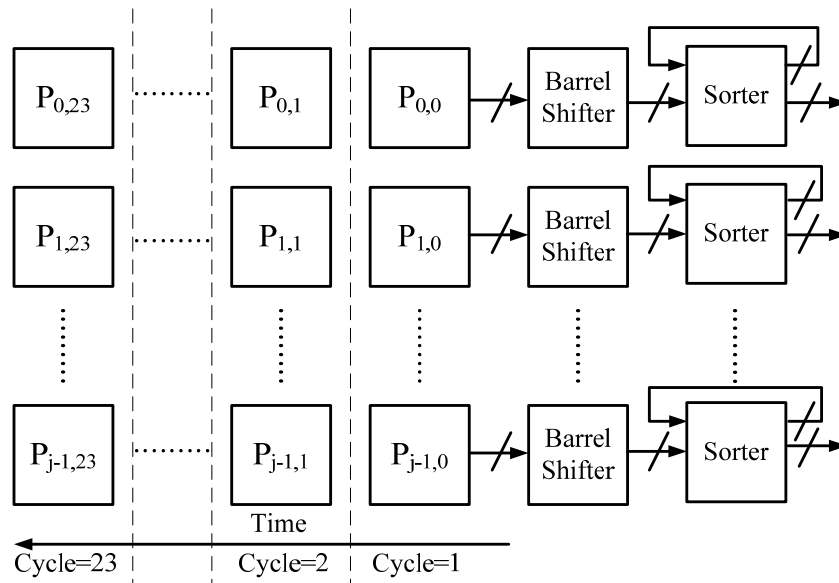


Fig. 4.2 Check Node Updating Memory Access Schedule

Fig. 4.2 is the check node updating memory access schedule, each row has 24 sub-matrices, the sub-matrices are the length of 96. We adopt partial parallel computation units, j is the partial parallel number. The barrel shifters can rotate the data in the memory to the correct position for check node updating, the rotation range is 0 to 95. The detail architecture of the barrel shifter will be discussed in the section 4.1.5. With the limitation of memory bus, every cycle we only access one column sub-matrices data and access the sub-matrices in column order at following cycle. The Sorter is composed of 96 data sorters, they reads the data after barrel shifter and the previous data. The data sorter architecture is as Fig. 4.3.

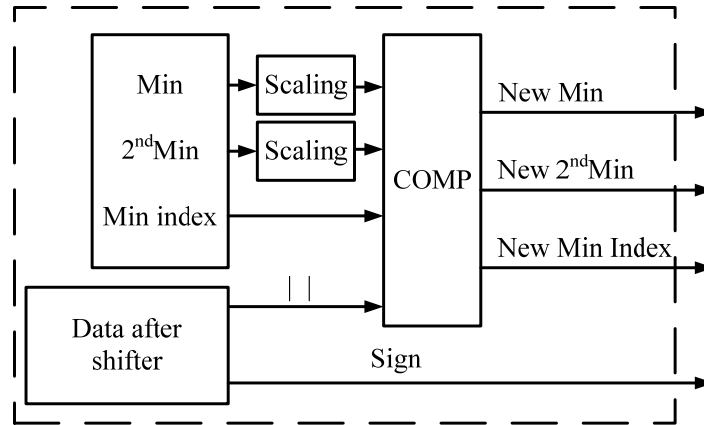


Fig. 4.3 One Data Sorter

The data sorter store the previous minimum value (Min)·second minimum value (Min2nd) and the minimum value location (Min index). There is one thing to know, Min and Min2nd are all absolute values. The minimum value location ranges from 1 to 24, the block COMP compares Min · Min2nd and new input data after barrel shifter to find new Min · new Min2nd and new Min index, also the new input data is taking absolute value from the data. Our check node updating processing element doesn't directly calculate $r_{k,n}$ out, we just find out the required information. The task of getting $r_{k,n}$ will hold at bit node updating.

The C2B register has to store four types values, Fig. 4.4 shows the register arrangement :

Check 0 Min	Check 1 Min	Check 2 Min	Check 1151 Min	Each block 5 bits
Check 0 2 nd Min	Check 1 2 nd Min	Check 2 2 nd Min	Check 1151 2 nd Min	Each block 5 bits
Check 0 Index	Check 1 Index	Check 2 Index	Check 1151 Index	Each block 5 bits
Check 0 Sign	Check 1 Sign	Check 2 Sign	Check 1151 Sign	Each block 24 bits

Fig. 4.4 C2B Register Arrangement

1. Minimum value : For 1152 rows of parity check matrix, there are 1152 minimum values we called Check Min for each row, because minimum value is absolute value, it require 5 bits to

represent the absolute value. Total register requirement is $1152 \times 5 = 5760$ bits.

2. Second minimum value : The same with minimum value, for 1152 rows of parity check matrix, there are 1152 second minimum values called Check 2ndMin for each row. Total register requirement is also $1152 \times 5 = 5760$ bits.

3. Minimum value index : Each row is divided into 24 arrays, each array only has one 1, so the minimum value location called Check Index has 24 possible locations. It requires 5 bits to represent 24. Total register requirement is $1152 \times 5 = 5760$ bits.

4. Sign magnitude : The registers stores the sign magnitudes for the term $sign(q_{n,k})$.

One row has to store 24 sign magnitude called Check Sign, so total register require $1152 \times 24 = 27648$ bits,

The total register requirement is **44928**bits.

B2C Block and B2C memory

The B2C block processes bit node updating, Originally it just sums up the required $r_{k,n}$ and channel value, but the C2B block doesn't get $r_{k,n}$, it only stores the required information for $r_{k,n}$ because we move this task for B2C block to execute. In order to reduce the complexity of control logic, we move some computation to C2B block. The B2C block contains two procedures : one is deriving $r_{k,n}$, the other is summing up. The Fig. 4.5 is the architecture for B2C block :

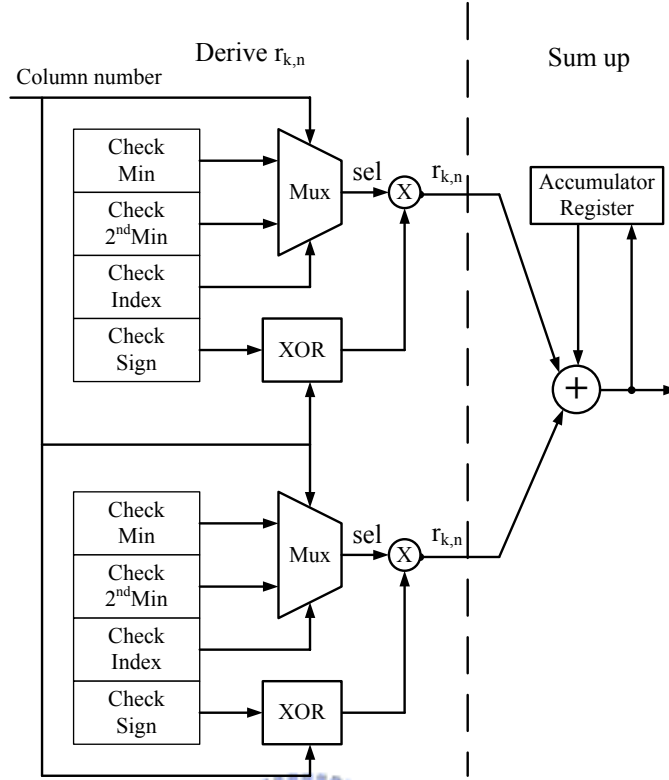


Fig. 4.5 B2C block architecture

The first procedure is to derive the $r_{k,n}$, according to the sub-matrix column number, the algorithm is as below :

$$\begin{cases} sel = Min2^{nd}, & \text{if } Column \text{ number} = Index \\ sel = Min, & \text{otherwise} \end{cases}$$

The multiplexer follows this algorithm to select the correct value. Now let's consider the check sign part, each check sign block contains 24 bits, each bit denotes each sub-matrix column value's sign magnitude. But some sub-matrices are zero matrices, so these matrices' sign magnitude we set to 0. In the Eqn.(4.2) of deriving $r_{k,n}$, the term $\prod_{n'=\{L(k)\setminus n\}} \text{sgn}(q_{n'k})$ requires the block check sign, the operation \prod is realized by exclusive-or. The set $n' = \{L(k) \setminus n\}$ makes the control logic more complex, we decide to XOR 25 bits, the first 24 bits are the all check sign bits in one check sign block, and the 25th bit is the decided by the

column number to xor the sign magnitude of $q_{n,k}$, which is also in the check sign block. The architecture is as below :

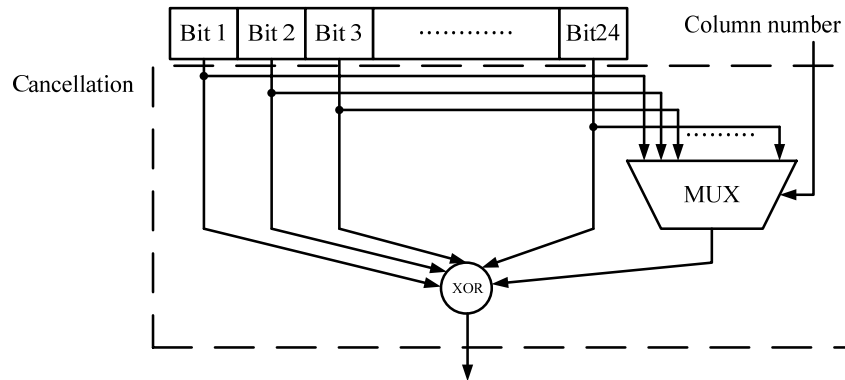


Fig. 4.6 XOR block architecture

Now compare Eqn.(2.14) and Eqn.(2.15) :

$$q_{n,k} = VAR(VAR_{k' \in M(n) \setminus k} (r_{k',n}), L(n)) = L(n) + \sum_{k' \in M(n) \setminus k} r_{k',n}$$

$$q_n = VAR(VAR_{k \in M(n)} (r_{k,n}), L(n)) = L(n) + \sum_{k \in M(n)} r_{k,n}$$

We can find

$$q_{n,k} = q_n - r_{k,n} \tag{4.3}$$

The term $r_{k,n}$ we can derive from the B2C block. In order to save memory usage, we just store the information of q_n , we don't store the information of $q_{n,k}$. For one row there are 24 possible $q_{n,k}$ s, it needs 6 bits to represent $q_{n,k}$, originally we need $1152 \times 24 \times 6 = 165888$ bits. Now we only need $1152 \times 6 = 6912$ bits, the total memory usage reduction is about 95%. The B2C memory block is as Fig. 4.7 :

Column 1	Column 2	Column 3	Column 96
Column 97	Column 98	Column 99	Column 192
⋮	⋮	⋮	⋮	⋮
Column 2209	Column 2210	Column 2211	Column 1152

Each block is 6bits

Fig. 4.7 B2C memory block

C2B block (II)

Because the B2C memory doesn't store the information of $q_{n,k}$, it only store the information of q_n , we have to rewritten the Eqn.(4.1). According to Eqn.(4.3), Eqn.(4.2) is rewritten as follow :

$$r_{k,n}^t = \begin{cases} \beta \times \min_{n=\{L(k)\}} (q_n^{t-1} - r_{k,n}^{t-1}) \prod_{n=\{L(k)\}} \text{sgn}(q_n^{t-1} - r_{k,n}^{t-1}) & \text{if } q_n^{t-1} \neq m_1 \\ \beta \times 2^{nd} \min_{n=\{L(k)\}} (q_n^{t-1} - r_{k,n}^{t-1}) \prod_{n=\{L(k)\}} \text{sgn}(q_n^{t-1} - r_{k,n}^{t-1}) & \text{otherwise} \end{cases} \quad (4.4)$$

The superscript of r and q denotes the iteration number, $r_{k,n}^t$ denotes $r_{k,n}$ derived from C2B block (in our design, it is actually derived at B2C block) at t^{th} iteration, and q_n^{t-1} denotes q_n derived from B2C block at $(t-1)^{\text{th}}$ iteration. In order to reduce memory usage, we move the task of deriving $q_{n,k}$ to C2B block to complete. Before find the minimum and second minimum of $q_{n,k}$, we have to find out the $q_{n,k}$ first by subtracting $r_{k,n}^t$ from q_n . So the architecture is as Fig. 4.8 :

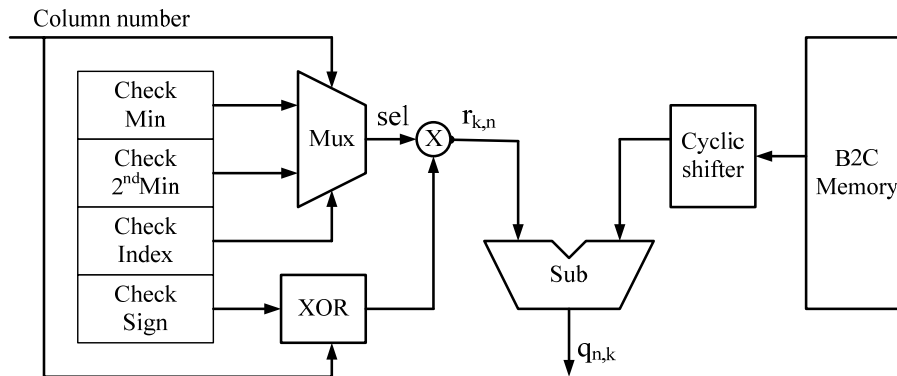


Fig. 4.8 Deriving $q_{n,k}$ architecture

The Cyclic shifter block will be introduced at next section.

4.1.3 Memory Arrangement

The memory arrangement can be divided into three parts : one is the channel value memory, another is the C2B register, and the other is the B2C memory. For convenient, the C2B register and B2C memory are introduced before with their relative processing elements. The channel value memory arrangement is very simple. Its task is to store the input data, the maximum input number is 2304, but in order to support full modes of LDPC code in 802.16e, we partition 2304 into 24 arrays which is the same with sub-matrix definition, so each array can store maximum 96 input data. When the mode is defined by $Z_f \times Z_f$ sub-matrices where Z_f is less than 96, every Z_f input data are stored at the array in order, Z_f denotes the z factor. Take block length=1152 for example, its z factor is 48, the 1st to 48th data are stored in the 1st array, the 49th to 96th data are store in the 2nd array and so on, and final the 1105th to 1152th data are store in the 24th array.

4.1.4 Cyclic Shift Block

There are two blocks requiring cyclic shifter : C2B block for C2B register and B2C block for B2C memory, but the two blocks require different direction cyclic shifter, and shift direction is different from the direction of matrix. We will explain why this happens.

Now we consider the C2B register first, C2B register number is labeled by the row number in parity check matrix. So there are 1152 block, each block contains one 5 bits Check Min 、 one 5 bits Check 2nd Min 、one 5 bits Check Index and one 24 Check Sign. But the C2B register is for B2C block to read, we have to convert the row arrangement to the column arrangement, so we need cyclic shifter to help us. Now we take a 5 bits cyclic shifter for example as shown in

Fig. 4.9,

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} & & & 1 & \\ & & & & 1 \\ & & & & & 1 \\ 1 & & & & \\ & 1 & & & \end{bmatrix}$$

Fig. 4.9 5×5 identity matrix and its permutation

The left hand matrix is a 5×5 identity matrix, after permutation the matrix becomes the right hand form, the shift size is 2. C2B register arrangement is shown in Fig. 4.10

Row 1	Row 2	Row 3	Row 4	Row 5
----------	----------	----------	----------	----------

Fig. 4.10 C2B register arrangement

After right shifting 2, the register arrangement is shown in Fig. 4.11 :

Row 4	Row 5	Row 1	Row 2	Row 3
----------	----------	----------	----------	----------

Fig. 4.11 C2B register arrangement after shifting

So the cyclic shifter satisfies the B2C block process's requirement, it can shift the data according to the shift size to the correct position for the B2C block read.

Then we consider the B2C memory, the B2C memory number is labeled by the column number in the parity check matrix. So there are 2304 block, each block contains one 6 bits q_n value. But the B2C memory is used for C2B block reading to derive $q_{n,k}$, we should convert the data order from column order to row order. As in the C2B case we take a 5 bits cyclic shifter for example as shown below.

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} & & & 1 & \\ & & & & 1 \\ & & & & & 1 \\ 1 & & & & \\ & 1 & & & \end{bmatrix}$$

The same with the Fig. 4.8, according to our memory arrangement is shown as Fig. 4.12 :

Column 1	Column 2	Column 3	Column 4	Column 5
-------------	-------------	-------------	-------------	-------------

Fig. 4.12 B2C memory arrangement

If we do the same process as C2B register, the memory arrangement after shifting becomes as follow is shown as Fig. 4.13 :

Column 4	Column 5	Column 1	Column 2	Column 3
-------------	-------------	-------------	-------------	-------------

Fig. 4.13 B2C memory arrangement after right shifting

Originally the column 1 message should be passed to row 4, but the right shift 2 doesn't make the goal we wish. The column 1 message was passed to row 3, this arrangement is incorrect. We can find that for B2C memory we need left shift instead of right shift, if we left shift 2, the memory arrangement is shown as Fig. 4.14 :

Column 3	Column 4	Column 5	Column 1	Column 2
-------------	-------------	-------------	-------------	-------------

Fig. 4.14 B2C memory arrangement after left shifting

This memory arrangement satisfies our row order. So for C2B register and B2C memory, we need one right cyclic shifter and one left cyclic shifter and these operations will require many hardware area, so for the left shifter, we try to use the right shifter by rewriting the shifting size. Comparing Fig. 4.13 and Fig. 4.11, for a t bits left shifter with shift size of r is equal to a t bits right shifter with shift size of $t-r$. So we can add an adder and a 2 input multiplexer to share the right shifter for both blocks. In our design, t denotes the z factor Zf . Fig. 4.14 shows the architecture of the sharing mechanism,

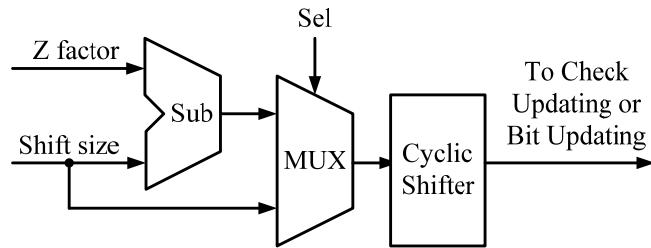


Fig. 4.15 Sharing mechanism of cyclic shifter

The cyclic shift block is a very important module because there are many modes for 802.16e. Now consider the size of sub-matrix, due to different modes, the shift size varies from 24 to 96, the interval is 4. A t -bits cyclic shifter can support $0 \sim t-1$ shift size of cyclic shift. In conventional cyclic shifter design, for a shift size=24 barrel shifter, it is constructed by a 24 bits cyclic shifter. In our design, directly constructing a t -bits cyclic shifter will occupies a lot of area because of a lot of multiplexers. For example, a 96 bits cyclic shifter needs a 96 inputs multiplexer. We construct a hierarchical cyclic shifter for our architecture, we cut the shifter into two part : one is at the sight of macroscopic of cyclic shifter and the other is fine-tuning. We employ the characteristic of all modes shift sizes, the shift sizes have the same submultiples. So we shift the data at higher interval to reduce multiplexer inputs and complexity, and then at next level we fine-tune the data to the correct position. Therefore, we develop a two-level cyclic shifter, the first level shifter shifts data at interval 4, that is, the shift size is the multiple of 4. The second level shifter shift data at interval 1, the shift range is from 0 to 3. For a 96 bits data, the first level cyclic shifter requires a 24 inputs multiplexer, the second level cyclic shifter requires a 4 inputs multiplexer. Compared to the conventional cyclic shifter, we can save a lot of architecture area for cyclic shifter. The Fig. 4.15 shows the architecture of two level cyclic shifter :

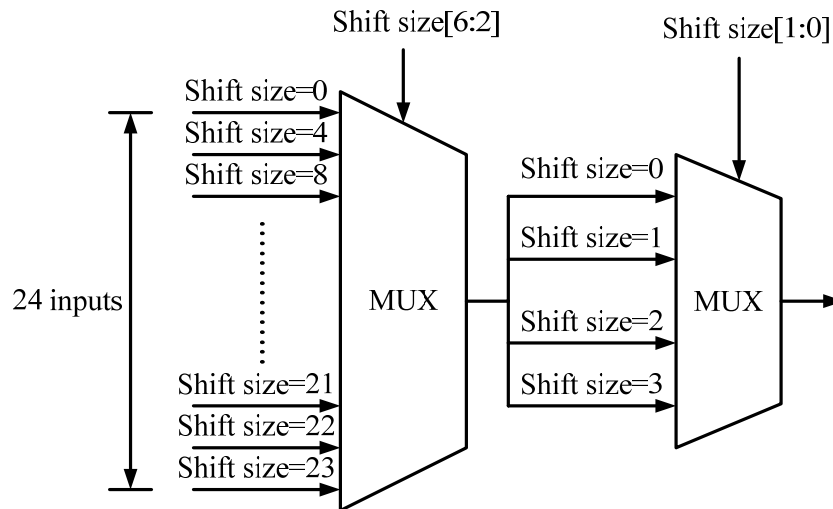


Fig. 4.16 Two level cyclic shifter

4.1.5 Shift Size ROM Table

We construct a ROM to store the value of the shift size for different code rate and length, besides. In order to separate zero matrix and identity matrix, we additionally store enable signals for sub-matrices. The enable signal is zero if the sub-matrix is zero matrix, otherwise, the enable signal is set to 1. Besides, to increase the bandwidth of the data bus, we cut the shift size ROM table into two parts, odd ROM and even ROM. At the same time, we read one shift size from each ROM to support shifter to process permute. It will decrease the latency of accessing the ROM table and make the control signal simpler.

4.1.6 The Overall architecture

The Fig 4.17 illustrates the overall architecture for proposed decoding process.

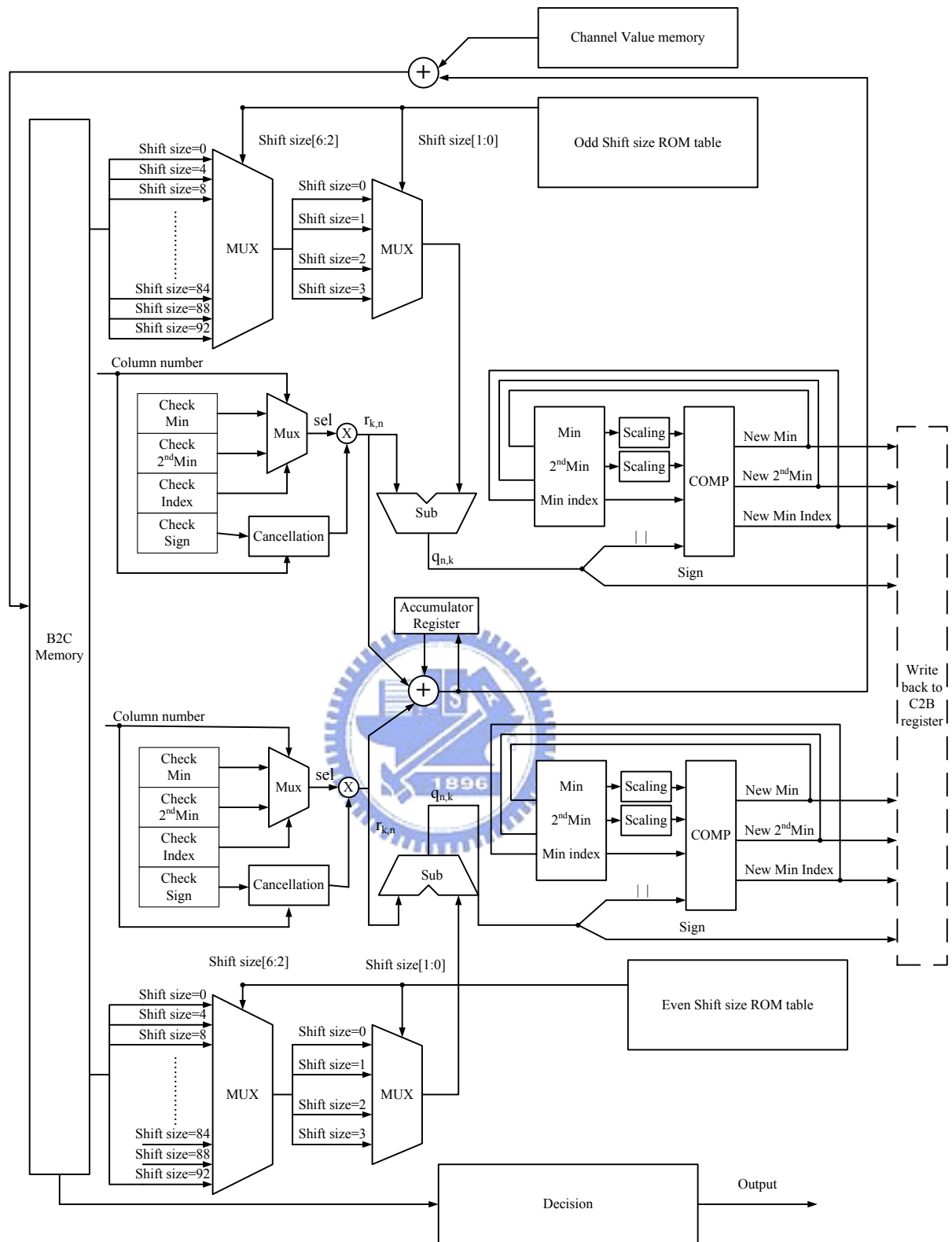


Fig. 4.17 Overall architecture

We adopt partial parallel of 2 to construct our decoding process, each parallel element process 96 computation units

4.2 Chip Implementation

The proposed LDPC decoder was implemented with the 0.13 μm 1P8M standard CMOS process. The chip layout of the LDPC decoder including the B2C memory, the shift size ROM table, ADDLL and the other control logic and processing blocks is shown in Fig. 4.16. The decoder die size is $13.69 \mu\text{m}^2$. The total gate count of the LDPC decoder is 1265K, where 194K is for B2C memory. We use 184 pins for pads and 67 pins for I/O pads and 114 pins for VDD/GND pads. The SRAM contains the B2C Memory and Channel Memory, and the ROM1 and ROM2 are the shift size rom table.

The post simulation result is tested to verify the functional correctness. The maximum iteration number is 20. The maximal data rate of the decoder is 28.3 Mb/s at typical case while working at 198MHz and 20.3 Mb/s at worst case. The power consumption is 700 mW. This power consumption analyzes only on iteration decoding excluding receiving data and outputting result.

The throughput rate is mostly constrained by the cyclic shifters latency and the data bus bandwidth for message passing. Although we simplify the operation of iteration decoding, a lot of accessing memory operations makes our decoding latency longer.

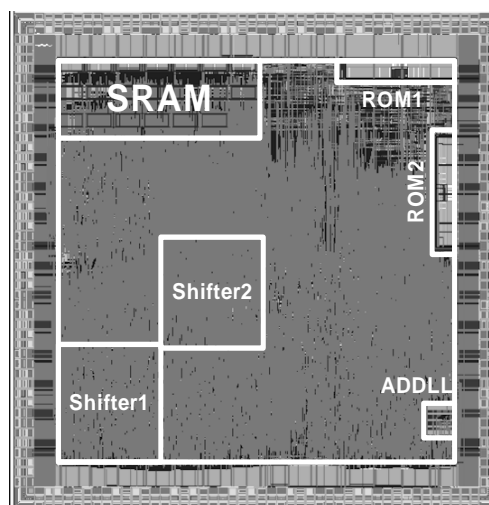


Fig. 4.18 Chip layout of the LDPC decoder chip

Table 4.1 Summary of the LDPC decoder chip

Technology	Standard 0.13-um CMOS 1P8M
Core size	3.0 um × 3.0 um
Chip size	3.7 um × 3.7 um
Gate count	1265K
Power dissipation	700mW @ 198MHz *
Maximum data rate	20.3Mb/s @ 20iterations **

* : The simulation environment is set at typical speed corner (1.2V Supply voltage) and the power consumption analyzes only on the iteration decoding excluding receiving data and outputting result.

** : The simulation environment is set at worst speed corner (1.08V Supply voltage) with considering the coupling noise due to crosstalk effect on signal wires. The maximum data rate is 28.3Mb/s at typical case, the worst IR drop=0.04V.

Table 4.2 shows the gate count of each functional block, “Control + C2B Register” denotes the control logic for the whole decoder and the registers for C2B block storage.

Table 4.2 Gate count of functional block

Function Block	Total gate count
Cyclic Shifter	135K
C2B Block	176K
B2C Block	37K
Control + C2B Register	758K
RAM	91K
ROM + Asynchronous	73K
Total	1270K

4.3 Comparison

The comparison of our proposed LDPC code decoder with state-of-the-arts are listed in Table

4.2.

Table 4.3 Comparison of LDPC chip

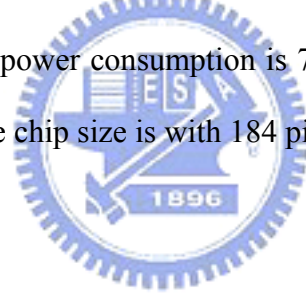
	Proposed	[24]	
Block length	576~2304 (19 types)	1200	
Code structure	Irregular	irregular	
Code rate	1/2, 2/3, 3/4, 5/6	3/5	
Silicon proven	No	Yes	No
Technology	0.13-um	0.18-um	0.13-um
Supply voltage	1.2V	1.8V	1.2V
Clock freq.	142MHz	83MHz	145MHz
Chip size	13.69mm ²	25mm ²	13.47mm ²
Gate count	1.265M	1.15M	
Power dissipation	700mW@198MHz	644mW	299mW
Data rate	20.3Mb/s@1.08V	3.33Gb/s	5.8Gb/s
Decoding iteration	20	8	

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we analyze the LDPC code for 802.16e based on the BER performance and propose an efficient architecture for 802.16e. In the proposed architecture we reschedule the process task for reducing memory usage and decreasing the latency. According our post simulation, this LDPC decoder can achieve the data rate to 20.3Mb/s using 0.13um, 1.08V, 1P8M CMOS process and the power consumption is 700mW at iterative decoding. The core occupies $3.0\mu\text{m} \times 3.0\mu\text{m}$ and the chip size is with 184 pins



5.2 Future Work

For our proposed architecture, the area for processing element is still too large, and the throughput rate is a little low to achieve the standard requirement. Our future work is to optimize the area and throughput rate. We will try to reuse the processing element to decrease the area and try to reschedule the processing element to achieve high clock rate.

References

- [1] P. Elias, "Coding for noisy channels", *IRE. Conv. Rec.* , pt.4, pp.37-47, 1955.
- [2] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Fields," *J. Soc. Ind. Appl. Math.*, 8: 300-304, June 1960.
- [3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.
- [4] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21-28, Jan. 1962.
- [5] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.
- [6] D. J. C. Mackay and R. M. Neal, "Near Shannon limit performance of low-density parity-check codes," *Electron. Lett.*, vol. 32, pp. 1645-1646, Aug. 1996.
- [7] T. J. Richardson and R. L. Urbanke, "Efficient encoding of Low-Density Parity-Check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638-656, Feb. 2001.
- [8] J. Pearl, *Probabilistic Reasoning in intelligent systems: networks of plausible inference*. San Mateo: Morgan Kaufmann, 1988.
- [9] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo decoding as a instance of Pearl's belief propagation algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 140-152, Feb. 1998.
- [10] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 219-230, Feb. 1998.
- [11] J. L. Fan, *Constrained coding and soft iterative decoding*. Netherlands: Kluwer Academic, 2001.

- [12] G. D. Forney, Jr., “Codes on graphs: Normal realizations,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.
- [13] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.
- [14] R. M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inform. Theory*, vol. IT-27, no. 5, pp. 399–431, Sept. 1981.
- [15] D. B. West, *Introduction to graph theory*, 2nd ed. NJ: Prentice-Hall, 2001.
- [16] J. L. Fan, *Constrained Coding and Soft Iterative Decoding*. Kluwer Academic Publishers, 2001
- [17] A. Anastasopoulos, “A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution,” in *IEEE GLOBECOM’01*, vol. 2, Nov. 2001, pp. 1021 – 1025
- [18] X. Y. Hu, Eleftheriou, D. M. Arnold, and A. Dholakia, “Efficient implementation of the sum-product algorithm for decoding ldpc codes,” in *IEEE GLOBECOM’01*, vol. 2, Nov. 2001, pp. 25–29
- [19] H. S. Song and P. Zhang, “Very-low-complexity decoding algorithm for low-density parity-check codes,” in *IEEE PIMRC’03*, vol. 1, Sep. 2003, pp. 161 – 165
- [20] J. Chen and M. P. C. Fossorier, “Near optimum universal belief propagation based decoding of low-density parity check codes,” *IEEE. Trans. Commun.*, vol. 50, pp. 406–414, Mar. 2002
- [21] H. Jun and K. M. Chugg, “Optimization of scaling soft information in iterative decoding via density evolution methods,” in *IEEE. Trans. Commun.*, vol. 6, Jun. 2005, pp. 957 – 961.
- [22] J. Chen and M. P. C. Fossorier, “Density evolution for two improved bp-based decoding algorithms of ldpc codes,” *IEEE. Communications Letters*, vol. 6, pp. 208 – 210, May 2002

- [23] *IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1*, 2006.
- [24] Chien-Ching Lin, Kai-Li Lin, Hsie-Chia Chang and Chen-Yi Lee, "A 3.33Gb/s (1200,720) Low-Density Parity Check Code Decoder," *IEEE ESSCIRC*, pp. 211 - 214 Sep. 2005.
- [25] N.Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Univ. Linkoping, Sweden, 1996.



作者簡歷

姓名：嚴紹維

出生地：台灣省高雄市

出生日期：1982.7.29

學歷：1988.9~1994.6 高雄市立四維國小

1994.9~1997.6 高雄市立五福國中

1997.9~2000.6 高雄市立高雄高級中學

2000.9~2004.6 國立交通大學 電子工程學系 學士

2004.9~2006.8 國立交通大學 電子研究所 系統組 碩士

