國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

適用於行動式全球互通微波存取通訊協定之 2.37Gb/s LDPC-CC 編解碼器設計

A 2.37Gb/s Rate-Compatible LDPC-CC Codec Design for Mobile WiMAX Applications

研 究 生：林玉祥

指導教授：張錫嘉 教授

中 華 民 國 一 百 年 八 月

適用於行動式全球互通微波存取通訊協定之 2.37Gb/s LDPC-CC 編解碼器設計

A 2.37Gb/s Rate-Compatible LDPC-CC Codec Design for Mobile WiMAX Applications

研 究 生：林玉祥
指導教授：張錫嘉 博士

Student：Yu-Hsiang Lin
Advisor：Dr. Hsie-Chia Chang

國 立 交 通 大 學
電子工程學系 電子研究所碩士班
碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering

& Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao-Tung University

In partial Fulfillment of the Requirements

For the Degree of Master

In

Electronics Engineering

Aug. 2011

Hsinchu, Taiwan, Republic of China

中 華 民 國 一 百 年 八 月

適用於行動式全球互通微波存取通訊協定之 2.37Gb/s LDPC-CC 編解碼器設計

學生：林玉祥　　指導教授：張錫嘉 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘　要

行動通訊系統中，通道編解碼模組往往扮演關鍵角色，不僅要達到高吞吐量的傳輸需求，也必須降低伴隨而來的功率消耗，以提供具有技術競爭力的解決方案。低密度奇偶校驗區塊碼(LDPC block codes，簡稱 LDPC-BCs)因具有優越的錯誤更正能力與適合平行運算之架構而廣受矚目，然而，此解碼器在實作時面臨高繞線複雜度的困難，在設計支援多碼率之 LDPC-BCs 也面臨許多挑戰。低密度奇偶校驗迴旋碼（LDPC convolutional codes，簡稱 LDPC-CCs）於 1999 年提出，此碼可對任意長度的資料區塊做編解碼，且易於經由穿孔(puncturing)機制提供彈性的碼率。相較於傳統 LDPC-BCs，LDPC-CCs 具有簡單的編碼電路及較低的繞線複雜度，相較於渦輪碼(Turbo codes)，更易於實現高速解碼器架構並且降低功率消耗。

近來，IEEE 制定新一代無線寬頻技術標準 802.16m，又稱為行動式全球互通微波存取通訊協定(Mobile WiMAX)，提供更高的資料傳輸速率和較低的延遲以滿足下世代行動通訊，雖然 LDPC-CCs 具有符合未來傳輸需求的潛力，但目前卻因解碼延遲過高、解碼吞吐量偏低、功率消耗過高等困難而尚未被通訊標準採用。據此，本作品提出演算法、節點、位元等三個層級的最佳化架構來提升吞吐量、減少硬體花費及降低解碼延遲時間，並藉由混合分割式 FIFO 架構來降低功率消耗。此論文使用 Panasonic 針對 802.16m

標準所提出之提案中的規格來實作，一個碼率相容(rate compatible)週期為 3 之 LDPC-CC。演算法層級最佳化使用即時變數節點激活並隱藏通道值之解碼排程(on-demand variable node activation scheduling with concealing channel values)可加快一倍的解碼收斂速度，並省去 17%的暫存器使用量，節點層級最佳化使用暫存器摺疊法(Folding architecture)可將平行度提高到 12，並同時降低解碼延遲 12 倍，經由時序重排(Retiming)進一步減少所需位元儲存量，最後使用混合分割式 FIFO (Hybrid-partitioned FIFO)來實現同時具有高吞吐量且低功率消耗之解碼器架構。

經由 UMC 90 nm 製程下線，在 1.2V 電壓下晶片實際量測到 198 MHz，資料吞吐量高達 2.37 Gbps，解碼器共包含 5 個處理器，在整體晶片面積 2.7mm^2 中僅佔 2.24 mm^2，功率消耗為 284mW，能源效率為 0.024 nJ/bit/proc。所提出的解碼器在各方面都具有極高的競爭力，相當適合於未來使用手持行動裝置的高速網路傳輸需求。

A 2.37Gb/s Rate-Compatible LDPC-CC Codec Design for Mobile WiMAX Applications

Student: Yu-Hsiang Lin Advisor: Hsie-Chia Chang

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

Abstract

In mobile communication system, channel coding modules play an important role. To provide a highly competitive solution, both high data-transmission rate and low power consumption are required. LDPC block codes (LDPC-BCs) has attracted great interest recently due to its capacity-approaching performance and inherent parallel architecture. However, the problem of high routing complexity becomes a design challenge in decoder implementations. The complexity of designing multiple code-rates LDPC-BCs is increased because of different parity-check matrices are needed to be jointly considered. Low-density parity-check convolutional codes (LDPC-CCs) were introduced in 1999, which are not only capable of handling variable length of data frame but also possess flexible code-rates. Compared with LDPC-BCs, LDPC-CCs enjoy the advantages of simple encoding circuitry as well as low routing complexity. Compared to the Turbo decoder, the LDPC-CC decoder is more suitable for highly-parallel implementation and low-power architecture.

Recently, IEEE 802.16m standards, also known as Mobile WiMAX Release 2.0, are developed in order to provide higher data rates and lower latency for next generation mobile communication systems. Although the LDPC-CC has the potential to meet the high-speed requirements for the next generation communication systems, it rarely appear in system

specification for its bottlenecks of the long decoding latency, high power consumption, and low-to-moderate decoding throughput. Therefore, our work proposed three level optimization techniques, including algorithm-level, node-level and bit-level, to increase decoding throughput, lessen hardware costs, and reduce decoding latency. In particular, a hybrid-partitioned FIFO structure is presented to further reduce power consumption. We adopted the specification of rate-compatible (491, 3, 6) LDPC-CC with period of 3 proposed by Panasonic for the IEEE 802.16m standards. The on-demand variable node activation scheduling with concealing channel values is proposed for algorithm-level optimization. This technique not only allows twice faster decoding convergence speed than the standard decoding schedule, but also saves 17% message storage requirements. The node-level optimization enables the parallelism of 12, thus the throughput becomes twelve multiplying with clock frequency. In the meantime, the decoding latency is reduced by approximately 12 times. Also, the bit level optimization is utilized to retime the variable nodes in order to achieve higher clock frequency and around a 20% storage reduction.

Fabricated in UMC 90nm 1P9M CMOS process, the proposed LDPC-CC decoder chip could achieve maximum 2.37 Gb/s under 198MHz operating frequency. The decoder containing 5 processors only occupies an area of 2.24 mm 2 within the core area of total 2.37×1.14 mm^2. It draws 284mW of power with an energy efficiency of 0.024nJ/bit/proc. Besides, the power can be scaled down to 90.2mW at 0.8V supply with 1.58Gb/s information throughput. In conclusions, our proposed LDPC-CC decoder outperforms state-of-the-art designs and is suitable for the high-speed requirements of next-generation handheld mobile devices.

誌　謝

研究所是收穫豐富的兩年，首先要非常謝謝錫嘉老師，總是非常用心地幫我們解決研究上遇到的困難，在生活各方面也都能適時給予協助，感謝老師提供這麼好的研究環境以及一路上的照顧與包容，讓我在知識和態度上都學習了許多也成長許多。謝謝 OASIS 實驗室和 OCEAN 的每一位夥伴，有了你們的一起 meeting、聊天、吃飯、煮火鍋使得我的生活充滿歡樂，在未來的某天如果回想起來我一定會十分懷念的。特別要謝謝的是陳志龍學長，從做專題到研究所這段期間受到學長的照顧太多了，不管什麼困難都可以找學長討論，很高興也覺得非常幸運有個這麼好的直屬學長，如果沒有學長的幫忙和指導，我想我無法順利地完成這篇論文。感謝博班學長小肥、義閔、修齊、國光、佳龍、欣儒、其橫、振揚和渠的幫忙，謝謝學弟妹祐子、雞皮、奕勳、皮皮，跟你們相處的生活真的太有趣了，感謝 98 一起奮鬥 6 年的夥伴小朱哥、許智翔、vfo、印度、士家和大姊頭，特別是受傷時候對我的幫忙與照顧，以及無法在這邊一一感謝的人，你們都在我生命中扮演了重要的角色。

最後要很真心的感謝我們家人，謝謝爸媽的一路辛苦栽培，讓我可以順利地完成學業，你們的支持是我往前努力的最大動力，謝謝你們讓我有如此甜蜜的回憶。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Due to high demands on error-correcting performance and the decoding throughput, low-density parity-check block codes (LDPC-BCs) have attracted great research interests. Although LDPC block codes have capacity-approaching performance and inherent parallel architecture, the problem of high routing complexity becomes a design challenge in decoder implementations. Besides, the parity-check matrices of different block lengths are needed for LDPC block codes to encode and decode arbitrary lengths of data. The convolutional version of LDPC codes, namely the LDPC convolutional codes were proposed in 1999. As compared with LDPC block codes, LDPC convolutional codes possess many advantages. LDPC convolutional codes are capable of handling variable lengths of data frame. With simpler encoding circuitry, they can provide flexible code-rates easily through puncturing. The pipeline decoder architecture of LDPC convolutional codes can simplify the problem of routing congestion in the VLSI implementation. Compared with convolutional turbo codes (CTC), interleavers are not necessary in the LDPC convolutional codes encoder. Although the Turbo codes have excellent performance, the complexity of employing highly parallel turbo decoders will significantly increase. The decoder architecture of LDPC convolutional codes is suitable for parallel decoding process comparing to the turbo decoders. Moveover, LDPC convolutional codes can perform near Shannon limit performance as the same as LDPC block codes and turbo codes.

Based on the above discussions, LDPC convolutional codes have the potential to meet

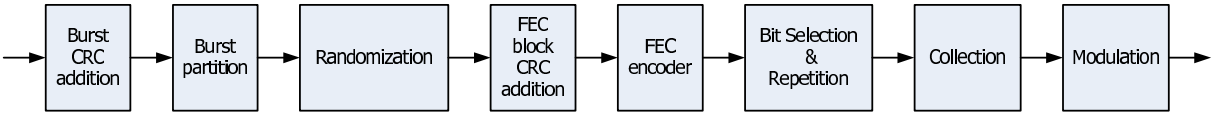


Figure 1.1: Block diagram of channel coding procedure in the IEEE 802.16m transmit chain [1].

the high-speed requirements in the next generation communication systems. However, LDPC convolutional codes rarely appear in system specification for its bottlenecks of the long decoding latency, high power consumption, and low-to-moderate decoding throughput. The maximum measured throughput in previous literatures was only 600Mb/s and difficult to compete with LDPC block codes. Therefore, these issues motivate the advances of the decoder architecture. Recently, IEEE 802.16m standards are developed in order to provide higher data rates and lower latency for next generation high-speed mobile communications. Figure. 1.1 shows the channel coding and modulation procedures in the IEEE 802.16m transmit chain [1]. Channel coding modules are indispensable to improve the performance and reliability of the overall systems. In this thesis, we adopted the specification proposed by Panasonic for the IEEE 802.16m standards [2]. To achieve high throughput, the parallel architecture for the encoder and decoder will be addressed. An improvement on the decoding schedule to reduce the decoding latency as well as hardware costs will be presented. Finally, we will propose a low-power decoder implementation with good error-correcting performance.

1.2 Thesis Organization

This thesis is organized as follows. In Chapter 2, we review the concept of low-density parity-check block codes and the iterative decoding algorithm. Chapter 3 gives the introduction of low-density parity-check convolutional codes. The code construction methods and the architectures of encoder and decoder are also provided in the chapter. In Chapter 4, the proposed techniques including algorithm-level, node-level, bit-level optimization and hybrid partitioned FIFO structure are described in detail. Chapter 5 gives the implementation results and summaries the architecture of the test chip. Comparisons with

the state-of-the-art designs are also provided. Conclusions and future works are given in Chapter 6.

Chapter 2

Background

In this chapter, we introduce the concept of low-density parity-check codes and Tanner graph representation. We also present the well-known iterative decoding algorithm and decoding procedure of low-density parity-check codes.

2.1 Low-Density Parity-Check Block Codes

Low-density parity-check (LDPC) block codes were first introduced by Gallager in 1960s [3]. However, these codes did not receive great interest at that time due to large computational complexity and difficulties in VLSI implementations. Rediscovered by MacKay and Neal [4], low-density parity-check codes were shown to have near Shannon limit bit error rate performance. Moreover, the structural regularity of low-density parity-check codes allows a highly-parallel decoder realization compared to the turbo decoder. As a consequence, low-density parity-check codes have attracted considerable attentions recently and have been widely adopted in many practical communication systems.

A binary low-density parity-check code is defined by a sparse parity-check matrix H, which contains a relatively low number of ones. For a regular (N, J, K) low-density parity-check block code, the block length is N and its parity-check matrix has exactly J ones in each column and K ones in each row. The parity-check matrix can be represented by a Tanner graph, or called a bipartite graph. We give the parity-check matrix of a $(6, 2, 3)$ regular LDPC block code as an example in (2.1). Each row of the matrix corresponds to a check node in the Tanner graph, and each column is mapped to a variable node. As the

example shown in Figure. 2.1 , the corresponding Tanner graph has 4 check nodes and 6 variable nodes, the number of variable nodes which are connected to the same check node is referred to the check node degree, and the number of check nodes which are connected to the same variable node is referred to the variable node degree. The one in the parity-check matrix is equal to an edge in the Tanner graph.

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{2.1}$$

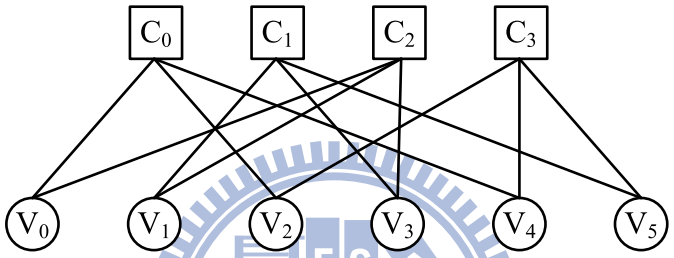


Figure 2.1: The Tanner graph of the parity check matrix given in (2.1).

Since the low-density parity-check codes are linear block codes, the encoding procedure is just like traditional linear block codes, we can use the generator matrix G to encode LDPC codes. Generally, the generator matrix G could be found by Gaussian elimination of the parity-check matrix H. For practical encoder realization, systematic encoding is often used to reduce encoding and decoding complexity. Therefore, the generator matrix can be simply represented as $G = [P|I]$, where I is the identity matrix. Since $GH^T = 0$, the parity-check matrix is $H = [I|P^T]$. For any valid codeword v, vH^T should be 0, and this property can be used for syndrome check in iterative decoding.

2.2 Iterative Decoding Algorithm

With the help of iterative message-passing decoding algorithm, low-density parity-check codes are capable of achieving near capacity performance. The best known iterative

decoding algorithm is belief propagation (BP) or called sum-product decoding algorithm. Simplified decoding algorithm such as min-sum algorithm and normalized min-sum algorithm reduce the decoder complexity with acceptable performance loss.

In iterative decoding, we are interested in the probability of the received symbol. These probabilities are usually represented in terms of log likelihood ratios (LLRs). We assume that the log likelihood ratio of bit n is

$$L_n = ln\frac{P(x=0)}{P(x=1)}.$$

The operation of iterative decoding can be described clearly using the Tanner graph. Figure. 2.2 and Figure. 2.3 show the Tanner graph and the notations we used in the following description. On the Tanner graph, the check nodes and variable nodes exchange messages along the edges iteratively. We illustrate the message passing operation of check node in Figure. 2.2, the outgoing message of the check node is computed from the other incoming messages. For variable node update shown in Figure. 2.3, the channel values are also participated in the outgoing message calculation. Let $\epsilon_{mn}^{(i)}$ be the message sent from check node m to variable node n, and let $z_{mn}^{(i)}$ denote the message sent from the variable node n to check node m. The a posterior LLR of bit n is denoted by $z_n^{(i)}$. The number of iterations is represented by i, and we also set the maximum iteration number as I_{Max}. The iterative decoding procedure of LDPC codes is described as follows.

1. **Initialization**

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

2. **Horizontal Step**

   check node to variable node update, for $1 \leq m \leq M$ and each $n \in N(m)$, where $N(m)$ represents the neighborhood of the m-th check node. For belief propagation (BP) decoding algorithm, compute

$$\epsilon_{mn}^{(i)} = 2\tanh^{-1}\Big(\prod_{n' \in N(m)\backslash n} \tanh\big(\frac{z_{mn'}^{(i-1)}}{2}\big)\Big) \tag{2.2}$$

   Otherwise, for min-sum algorithm, compute

$$\epsilon_{mn}^{(i)} \approx \Big(\prod_{n' \in N(m)\backslash n} sgn(z_{mn'}^{(i-1)})\Big) \cdot \min_{n' \in N(m)\backslash n}(|z_{mn'}^{(i-1)}|) \tag{2.3}$$

3. Vertical Step

variable node to check node update, for $1 \leq n \leq N$ and each $m \in M(n)$, where $M(n)$ denote the set of neighbors of the n-th variable node, compute

$$z_{mn}^{(i)} = L_n + \sum_{m' \in M(n) \backslash m} \epsilon_{m'n}^{(i)} \tag{2.4}$$

4. Decision Step and Stopping Criterion Test

Let $\hat{x}_n$ be the n-th bit of decoded codeword.

$$z_n^{(i)} = L_n + \sum_{m' \in M(n)} \epsilon_{m'n}^{(i)} \tag{2.5}$$

$$\hat{x}_n = \begin{cases} 0, & \text{if } z_n^{(i)} \geq 0 \\ 1, & \text{if } z_n^{(i)} < 0 \end{cases} \tag{2.6}$$

If $H \cdot \hat{x}_n^T = 0$ or the maximum iteration number I_{Max} is reached, the decoder stops the decoding process and outputs the decoded codeword. Otherwise, set $i = i + 1$ and the decoder repeats the decoding steps.

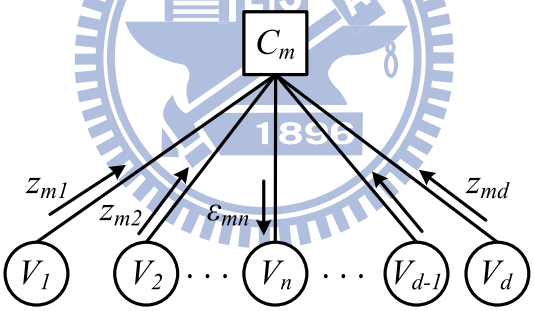


Figure 2.2: Message passing of check node.

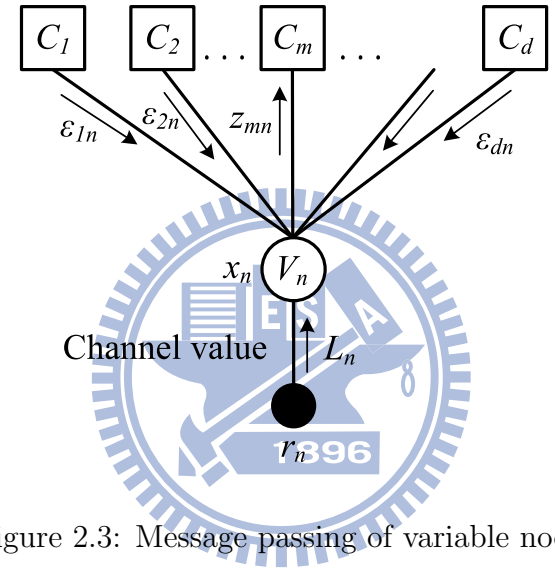


Figure 2.3: Message passing of variable node.

Chapter 3

Low-Density Parity-Check Convolutional Codes

Low-density parity-check convolutional codes were first proposed in 1999 [5], which are convolutional codes defined by sparse parity-check matrices and can be decoded using the iterative message-passing algorithm. Due to the properties of convolutional codes, LDPC convolutional codes can encode and decode variable length of data frame. It has been shown that these codes are suitable for certain applications such as streaming video and packet-switching networks [6].

In this chapter, the overview of LDPC convolutional codes is given. In the following sections, firstly, some important parameters of LDPC convolutional codes are defined, and then the methods for code construction in the literature are described. Moreover, the encoding procedure and encoder architecture are introduced. Finally, the pipeline decoder architecture and the corresponding Tanner graph of LDPC convolutional codes are also presented.

3.1 Definition and Code Constructions

A binary LDPC convolutional code is defined by a transposed semi-infinite parity-check matrix, or referred to the syndrome former H^T. For a rate $R = b/c$ $(b < c)$ LDPC convolutional code, the syndrome former can be described by the following form

$$H^T = \begin{pmatrix} \ddots & & \ddots & & \\ H_0^T(t-m_s) & \dots & H_{m_s}^T(t) & & \\ & \ddots & \vdots & \ddots & \\ & & H_0^T(t) & \dots & H_{m_s}^T(t+m_s) \\ & & \ddots & & \ddots \end{pmatrix}. \tag{3.1}$$

The sub-matrices $H_i^T(t)$ at time instant t given in (3.2), $i = 0, 1, \ldots, m_s$, are size of $c \times (c-b)$. In particular, the sub-matrix $H_0^T(t)$ are chosen to be full rank, this condition can make encoding easier and allow a register-based encoder implementation.

$$H_i(t) = \begin{pmatrix} h_i^{(1,1)}(t) & \dots & h_i^{(1,c)}(t) \\ \vdots & & \vdots \\ h_i^{(c-b,1)}(t) & \dots & h_i^{(c-b,c)}(t) \end{pmatrix} \tag{3.2}$$

The parameter m_s is called the syndrome former memory or the code memory of LDPC convolutional codes, and $v = c \cdot (m_s + 1)$ is defined as the constraint length. The number of ones in each row and each column in the syndrome former determine the variable node degree J and check node degree K respectively. For a regular (m_s, J, K) LDPC convolutional code, there are exactly J ones in each row and K ones in each column in the syndrome former; otherwise, it is an irregular code. If there exists a period T such that $H_i^T(t) = H_i^T(t+T)$, the code is periodically time-varying. For the period T equals 1, it is said to be a time-invariant LDPC convolutional code. We give an example of $(5, 3, 6)$ LDPC convolutional code with period $T = 4$ in (3.3). It can be seen easily from (3.3) that the syndrome former has 3 ones in each row and 6 ones in each column.

In addition, here we introduce some major techniques for the construction of LDPC convolutional codes in the literature. In these years, most LDPC convolutional codes are derived from the parity-check matrices of LDPC block codes. In [5], the authors firstly proposed the unwrapping procedure to obtain a time-varying periodical parity-check matrix of an LDPC convolutional code from a randomly constructed LDPC block code. In [7] and [8], the algebraically structured quasi-cyclic LDPC (QC LDPC) block codes were also applied to derive both time-invariant and time-varying LDPC convolutional codes. In addition, a construction method to design LDPC convolutional codes based on

protographs are proposed in [9]. The protograph-based LDPC convolutional codes enjoy several advantages, such as an effective pipeline decoding and thresholds close to capacity.

However, there are few researches on the constructions of low-density parity-check convolutional codes from the convolutional codes point of view. Until [10], a newly construction method based on the parity check polynomials of the convolutional codes was proposed. The parity check polynomials of the convolutional codes are generated randomly with predefined constraints. The simulation results in [10] have shown that the constructed time-varying LDPC convolutional codes with time period of 3 can provide good BER performances.

$$H^T = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & & & & & & \\
1 & 0 & 0 & 1 & 0 & 1 & & & & & & \\
 & 1 & 1 & 1 & 0 & 0 & 0 & & & & & \\
 & 1 & 0 & 0 & 1 & 0 & 1 & & & & & \\
 & & 1 & 0 & 1 & 0 & 1 & 0 & & & & \\
 & & 1 & 0 & 0 & 1 & 0 & 1 & & & & \\
 & & & 1 & 1 & 1 & 0 & 0 & 0 & & & \\
 & & & 1 & 0 & 0 & 1 & 0 & 1 & & & \\
 & & & & 1 & 1 & 0 & 1 & 0 & 0 & & \\
 & & & & 1 & 0 & 0 & 1 & 0 & 1 & & \\
 & & & & & 1 & 1 & 1 & 0 & 0 & 0 & \\
 & & & & & 1 & 0 & 0 & 1 & 0 & 1 & \\
 & & & & & & \ddots & & & & & \ddots
\end{pmatrix}. \tag{3.3}$$

3.2 LDPC Convolutional Codes for Mobile WiMAX

In this thesis, we adopt the specification proposed for the IEEE 802.16m standards by Panasonic [2], a rate compatible $(491, 3, 6)$ LDPC convolutional codes with time period of 3. IEEE 802.16 defines the air interface for fixed and mobile broadband wireless access systems. In order to meet the requirements of next generation mobile communications, IEEE 802.16m is developed to provide higher data rates and lower latency than 802.16e standards. IEEE 802.16m system, also known as Mobile WiMAX Release 2.0,

incorporates many innovative communication technologies with significant performance improvements than order releases [1]. Although LDPC block codes have been adopted in many communication standards, such as DVB-S2, IEEE 802.3an and IEEE 802.16e, current 802.16e LDPC block codes does not support IR type HARQ [2]. The proposed rate-compatible LDPC convolutional code is able to support IR type HARQ for the 16m FEC. In particular, LDPC convolutional codes are more suitable for parallel implementation than convolutional turbo codes (CTC). Compared with LDPC block codes, LDPC convolutional codes enjoys the advantages of encoder complexity as well as decoding latency. Therefore, the proposed LDPC convolutional code has the potential to be one candidate for next-generation communication systems.

The equations in (3.4), which are constructed based on the parity check polynomials define the time-varying convolutional encoder, where $u(D)$ and $v(D)$ represent the information polynomial and parity polynomial respectively. Due to the time period is only 3, this property leads to significantly reduced hardware implementation complexity. Besides, the proposed LDPC convolutional code could support 5 code rates through puncturing. The puncturing patterns are shown in Table 3.1, the value 1 indicate that bit will be transmitted. In the other hand, the value 0 indicates that bit will be discarded every L_{punc} bits. Note that the LLRs of punctured bits are assumed to 0 at the receiver and the decoding algorithm remains the same. Simulation results in [2] show that the BER performances are comparable to 16e Turbo decoder under AWGN channel for all code rates.

$$(D^{373} + D^{56} + 1)u(D) + (D^{406} + D^{218} + 1)v(D) = 0 \tag{3.4a}$$

$$(D^{457} + D^{197} + 1)u(D) + (D^{491} + D^{22} + 1)v(D) = 0 \tag{3.4b}$$

$$(D^{485} + D^{70} + 1)u(D) + (D^{236} + D^{181} + 1)v(D) = 0 \tag{3.4c}$$

Table 3.1: Puncturing patterns for (491,3,6) LDPC convolutional code.

Code Rate	Puncture Pattern		Puncture Length (L_{punc})
	Information	Parity	
1/2	1	1	1
2/3	110100	111111	6
3/4	010111	111010	6
4/5	1011	0110	4
5/6	1110001110	0100110111	10

3.3 Encoding of LDPC Convolutional Codes

The encoding procedure of a rate $R = b/c$ $(b < c)$ LDPC convolutional code is described as follows. Let

$$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1}) \tag{3.5}$$

be an information sequence, where $\mathbf{u}_i = (u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(b)})$. And assume that the coded sequence after encoding is

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{t-1}), \tag{3.6}$$

where $\mathbf{v}_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(c)})$. The coded sequence $\mathbf{v}$ satisfies the constraint $\mathbf{vH^T} = \mathbf{0}$. This equation could be further decomposed into equation (3.7) and directly used for encoding. Once all sub-matrices $\mathbf{H}_0^T(t)$ have full rank, the encoder can be realized as a shift-register based encoder. This property is called the fast encoding property, which guarantees that the constructed codes can be encoded continuously in real time [11].

$$\mathbf{v}_t\mathbf{H}_0^T(t) + \mathbf{v}_{t-1}\mathbf{H}_1^T(t) + \ldots + \mathbf{v}_{t-m_s}\mathbf{H}_{m_s}^T(t) = 0 \tag{3.7}$$

Low-density parity-check convolutional codes are commonly expressed in terms of the polynomial representation. The transposed polynomial parity-check matrix $H^T(D)$ of the convolutional code can be rewritten as

$$H^T(D) = H_0^T + H_1^T D + H_2^T D^2 + \ldots + H_M^T D^M. \tag{3.8}$$

We give an example of $R = 3/8$ time-invariant LDPC convolutional code with code memory $m_s = 203$ [12] in (3.9) to illustrate the encoding process. As long as $H_0^T(t)$ satisfy the full rank condition, the encoding equations can be easily derived from equation (3.7).

$$H(D) = \begin{pmatrix} 1+D^{194} & D^{158} & D^{166} & D^{144} & 0 & D^{65} & 0 & 0 \\ D^{97} & D^{49} & 0 & D^{203} & D^{65} & D^{37} & 1 & 0 \\ 0 & D^{106} & D^{83} & D^{138} & D^{48}+D^{132} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & D^{20} & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+D^{76} & 1 \end{pmatrix} \tag{3.9}$$

Let $v_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)}, v_t^{(3)}, v_t^{(4)}, v_t^{(5)}, v_t^{(6)}, v_t^{(7)})$ be the coded bits and $u_t = (u_t^{(0)}, u_t^{(1)}, u_t^{(2)})$ be the information bits. Given that $v_t^{(1)} = u_t^{(0)}$, $v_t^{(3)} = u_t^{(1)}$ and $v_t^{(4)} = u_t^{(2)}$. The parity bits shown in (3.10) can be determined by solving (3.7).

$$v_t^{(0)} = v_{t-194}^{(0)} \oplus v_{t-158}^{(1)} \oplus v_{t-166}^{(2)} \oplus v_{t-144}^{(3)} \oplus v_{t-65}^{(5)} \tag{3.10a}$$

$$v_t^{(2)} = v_{t-20}^{(6)} \oplus v_t^{(7)} \tag{3.10b}$$

$$v_t^{(5)} = v_{t-106}^{(1)} \oplus v_{t-83}^{(2)} \oplus v_{t-138}^{(3)} \oplus v_{t-48}^{(4)} \oplus v_{t-132}^{(4)} \tag{3.10c}$$

$$v_t^{(6)} = v_{t-97}^{(0)} \oplus v_{t-49}^{(1)} \oplus v_{t-203}^{(3)} \oplus v_{t-65}^{(4)} \oplus v_{t-37}^{(5)} \tag{3.10d}$$

$$v_t^{(7)} = v_t^{(6)} \oplus v_{t-76}^{(6)} \tag{3.10e}$$

A systematic encoder can be obtained if we let the bottom $(c-b)$ rows of the sub-matrices $H_0^T(t)$ are identity matrix of size $(c-b)\times(c-b)$. For this special case, the encoding equations can be simply summarized as follows.

$$v_t^{(j)} = u_t^{(j)}, j = 1, \ldots, b, \tag{3.11}$$

$$v_t^{(j)} = \sum_{k=1}^{b} v_t^{(k)} h_0^{(j-b,k)}(t) + \sum_{i=1}^{m_s} \sum_{k=1}^{c} v_{t-i}^{(k)} h_i^{(j-b,k)}(t), j = b+1, \ldots, c. \tag{3.12}$$

The encoding circuitry of an LDPC convolutional code is simple, analogous to the conventional convolutional code feedback encoder. The encoder can be implemented using multiplexers, XOR gates and c length (m_s+1) shift-registers. If the constructed code is time-invariant, the encoder connection is fixed. For the time-varying case which we are usually interested in, the encoder connections change periodically according to the check equations. It is worth to note that the encoding complexity of LDPC convolutional code is independent of the codeword length. Consequently, the encoder of LDPC convolutional

code enjoys the advantage of simple circuitry comparing to the traditional LDPC block code encoder. In addition, the ability to support arbitrary length of data frame provides more flexibility than LDPC block codes in many communication scenarios. The block diagram of a $R = 1/2$ systematic encoder for LDPC convolutional code is given in Figure. 3.1. The weight controller configures the encoder connections according to the parity-check polynomials. For low-cost implementation, the partial-syndrome former realization can be found in [13].

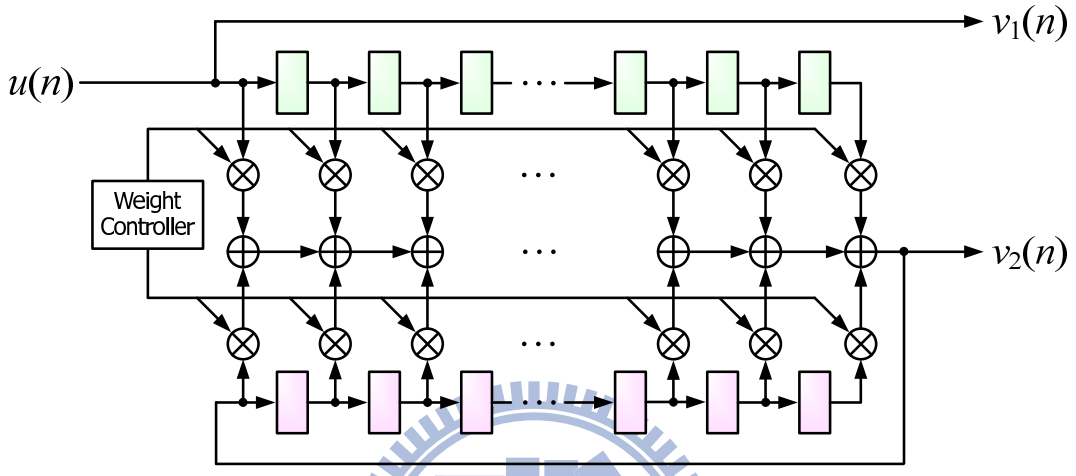


Figure 3.1: A rate 1/2 systematic LDPC convolutional code encoder.

3.4 Decoding of LDPC Convolutional Codes

The LDPC convolutional codes employ the same iterative message-passing algorithm as the decoding of LDPC block codes. Different from the LDPC block codes, the corresponding Tanner graph of an LDPC convolutional cods is infinite. Therefore, a pipeline decoder architecture is proposed as a finite sliding window to perform variable node and check node updates on the Tanner graph. Since the distance between two variable nodes that are connected to the same check node is at least $(m_s + 1)$ time instants apart, the decoding of two variable nodes can be performed independently. This concept allows the pipeline decoder to perform simultaneous decoding on different regions of the Tanner graph. The sliding window decoding on the Tanner graph is implemented by using a serial concatenation of I independent identical processors. The number of identical

processors corresponds to the number of iterative decoding iterations. Hence, the more processors are used, the more decoding performance increases. Studies have shown that LDPC convolutional codes could achieve the same near-Shannon-limit error performance as LDPC block codes. Recently, a comparison of LDPC block and LDPC convolutional codes indicates that, for the same decoding performance, the LDPC convolutional codes require less hardware costs than their corresponding block codes [14]. In addition, a detail investigation on several implementation issues such as stopping rules, decoding schedules, and several improvements to pipeline decoder architecture can be found in [15].

We give a $R = 1/2$ time-invariant $(14, 3, 6)$ LDPC convolutional code in (3.13) as an example to illustrate the operation of the pipeline decoder on the Tanner graph. Figure. 3.2 shows the corresponding trellis diagram and the illustration of pipeline decoding. Since we add additional stage of register for pipelining, the length of sliding window in Figure. 3.2 is $(m_s + 2)$ instead of $(m_s + 1)$.

$$(1 + D^5 + D^{11})u(D) + (1 + D^7 + D^{14})v(D) = 0 \tag{3.13}$$

For a $R = b/c$ $(b < c)$ LDPC convolutional code decoder, it consists I identical processors operating in parallel, and each processor consists $(J + 1) \times c \times (m_s + 1)$ first-in first-out (FIFO) shift registers, $(c - b)$ check node units and c variable node units. In particular, this regular architecture insures low routing complexity comparing to the traditional LDPC block code. The decoding procedure is described in the following steps. Initially, all the shift registers in the pipeline decoder are filled with infinite ∞ because of the dummy zeros are the initial values in the encoder. As a new channel LLR is received, it is shifted in all the $(J+1)$ shift registers. Note that the LLR stored in the first shift register indicates the intrinsic value in the message-passing decoding. The next step is to operate the check node computations and then update the corresponding symbols which are connected to the same check node unit. Here we use the decoding algorithms such as belief propagation (BP) algorithm or normalized min-sum (NMS) algorithm in the bit-error-rate simulations. For the preceding $(I - 1)$ processors, the variable node operations are performed just before the LLRs leave the processor. The last processor performs the hard decision for the information LLRs. Thus, the decoding procedure successively repeats the shifting step and appropriate node updates. As long as the initial decoding delay has elapsed, a total of $I \times (m_s + 1)$ time units, the pipeline decoder outputs a decoded data

stream continuously.

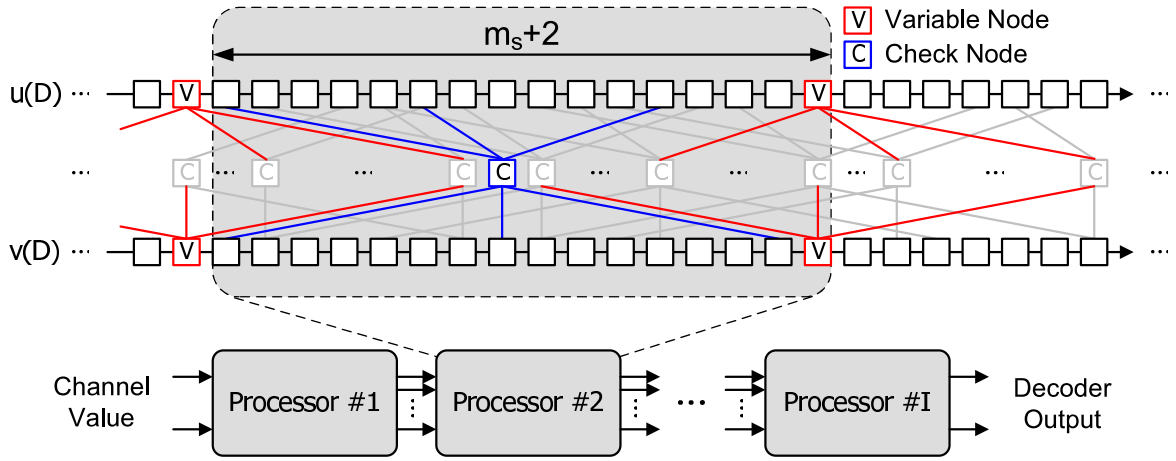


Figure 3.2: Trellis diagram and the illustration of pipeline decoding.

Chapter 4

Proposed Techniques for LDPC Convolutional Codes

In previous chapter, we have introduced many important properties of LDPC convolutonal codes, including near Shannon limit performance, variable length of data frame, flexible code-rate through puncturing, simple encoding circuitry, and low routing complexity in the decoder architecture. However, LDPC convolutional codes rarely appear in system specification due to its bottlenecks of the long decoding latency, high power consumption, and low-to-moderate decoding throughput. The maximum measured throughput in previous literatures was only 600Mb/s and difficult to compete with LDPC block codes [16] [17].

In this thesis, we focus on the following three critical issues to design a high performance LDPC convlutional code decoder. Firstly, more processor numbers can obtain better decoding performance but extend longer latency with similar throughput. Secondly, high node-level parallelism can achieve higher throughput but cause larger message bandwidth and severe memory conflict in memory-based FIFO architecture. Thirdly, a register-based FIFO architecture can support unlimited bandwidth but yield expensive power and area costs. To overcome these three issues, this thesis propose a new design approach which combines algorithm-level, node-level, bit-level optimization, and a hybrid-partitioned FIFO to achieve over 2Gb/s throughput with competitive power consumption and chip area.

4.1 Algorithm-Level Optimization

4.1.1 Flooding Scheduling

The message passing schedule is the order of propagating messages between check nodes and variable nodes over the Tanner graph. For decoding LDPC codes, the standard message-passing decoding schedule is the flooding schedule. According to the flooding schedule, the messages are passed in parallel between nodes. In each iteration, all the check nodes are updated simultaneously using the variable-to-check messages. Then, followed by updating all the variable nodes using the check-to-variable messages. Although the flooding scheduling is suited for parallel implementation, it is inefficient due to its low decoding convergence speed. Figure. 4.1 shows the conventional decoder architecture of $(14, 3, 6)$ LDPC convolutional code given in (3.13). We can see that the variable nodes are performed until the messages are all transferred into the check-to-variable messages in the decoder.

We plot the floating point simulation results of $(491, 3, 6)$ LDPC convolutional code with flooding schedule in the following figures. Figure. 4.2 depicts the BER performance under different decoding iterations at $R = 1/2$. We also compare the performance using different decoding algorithms such as log-BP algorithm, min-sum algorithm, and normalized min-sum algorithm. In Figure. 4.3, the dependence of the BER on the number of processors I at different SNRs using flooding schedule is shown. The results are simulated using the NMS algorithm with scaling factor 0.75. In Figure. 4.4, we present the BER performance under different code-rates through puncturing with 50 iterations. Comparisons of log-BP algorithm and normalized min-sum algorithm with scaling factor 0.75 are also given.

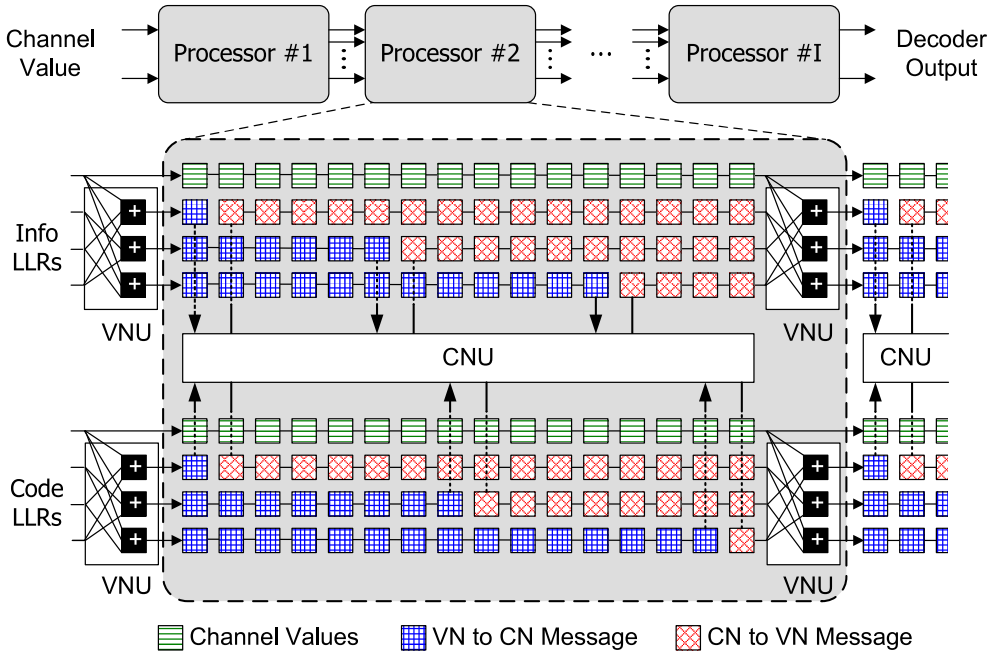


Figure 4.1: Conventional decoder architecture.

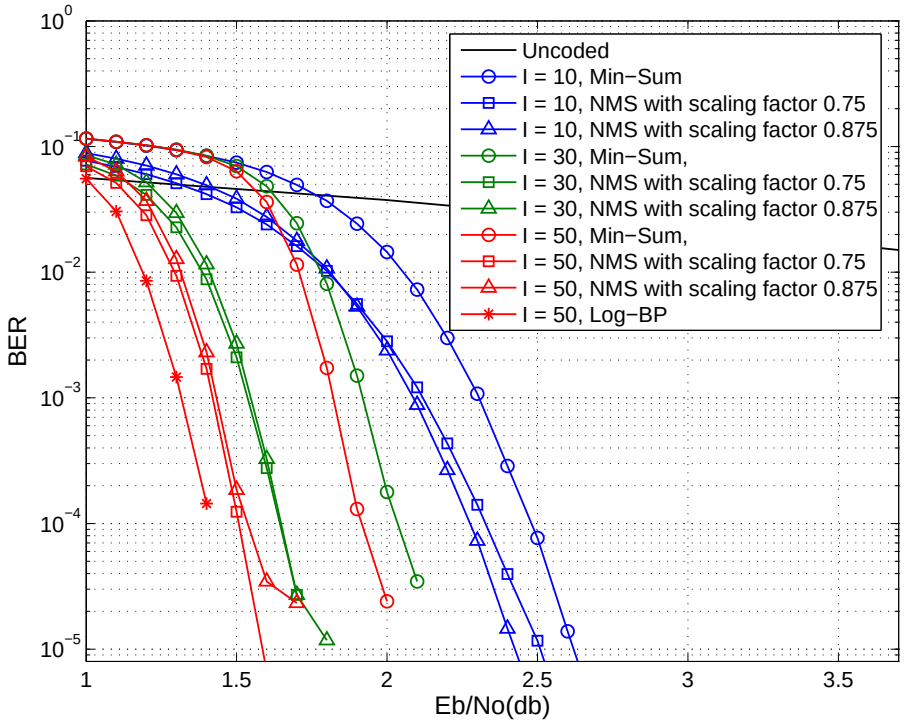


Figure 4.2: Simulation results under different number of iterations.

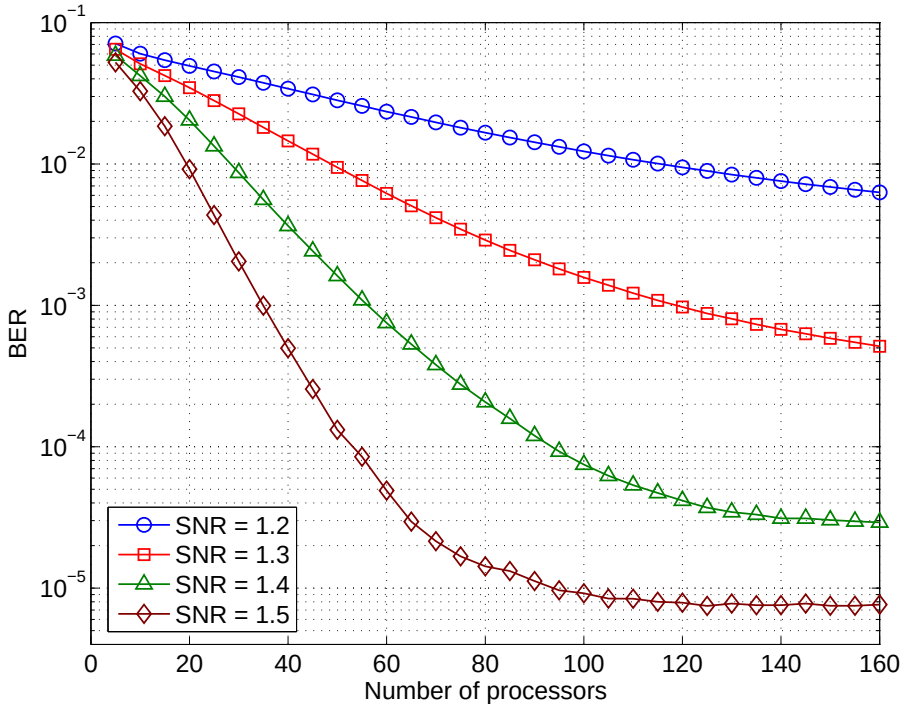


Figure 4.3: The dependence of the BER on the number of processors I at different SNRs.

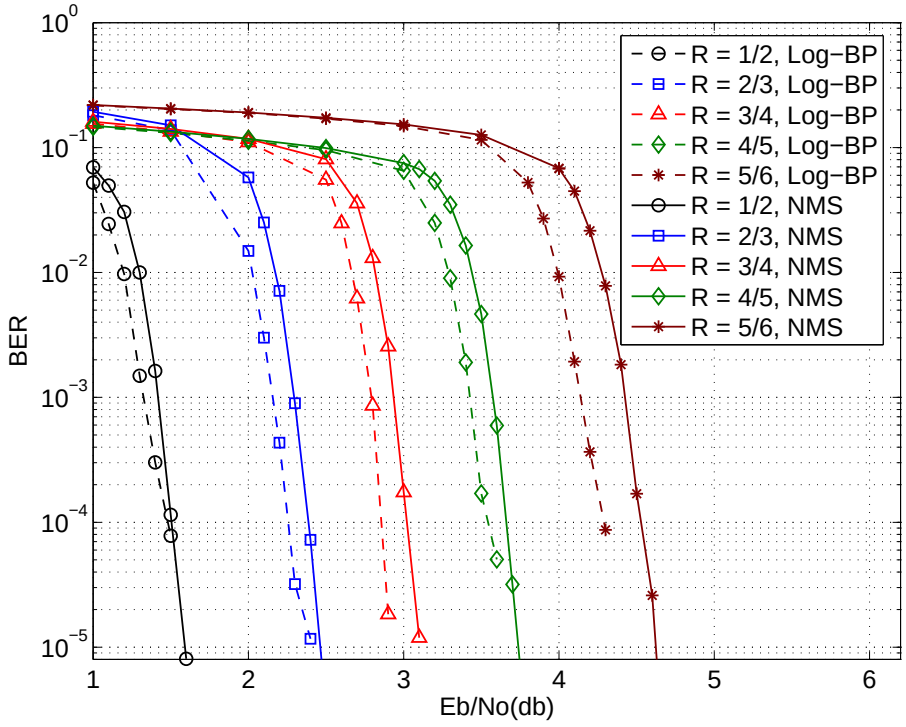


Figure 4.4: BER performance under different code-rates with 50 iterations using flooding schedule.

4.1.2 On-Demand Variable Node Activation (OVA) Scheduling

Studies have shown that the message-passing schedule affects the rate of decoding convergence and computational complexity. If we observe the standard decoding schedule, namely, the flooding scheduling, the variable node units are activated only once before the values leave the processor. Most messages are shifting while few messages are updating in a processor. Thus, this scheduling is inefficient without utilizing the recent updated information. For increasing the convergence speed, sequential scheduling are introduced in decoding LDPC block codes. Recently, an on-demand variable node activation schedule is proposed in [18] to accelerate the decoding convergence speed for LDPC convolutional code. The main idea is to change the variable node activation location leaving from the processor to the position right before each check node input. This on-demand variable node activation scheduling is very similar to the layered decoding in LDPC block codes [19] [20] that check nodes could access the most recent messages. We use the same example given in (3.13) to demonstrate the algorithm-level optimization by using the on-demand variable node activation scheduling technique. Although this example is a time-invariant code, the on-demand scheduling can be applied directly to the case of time-varying codes.

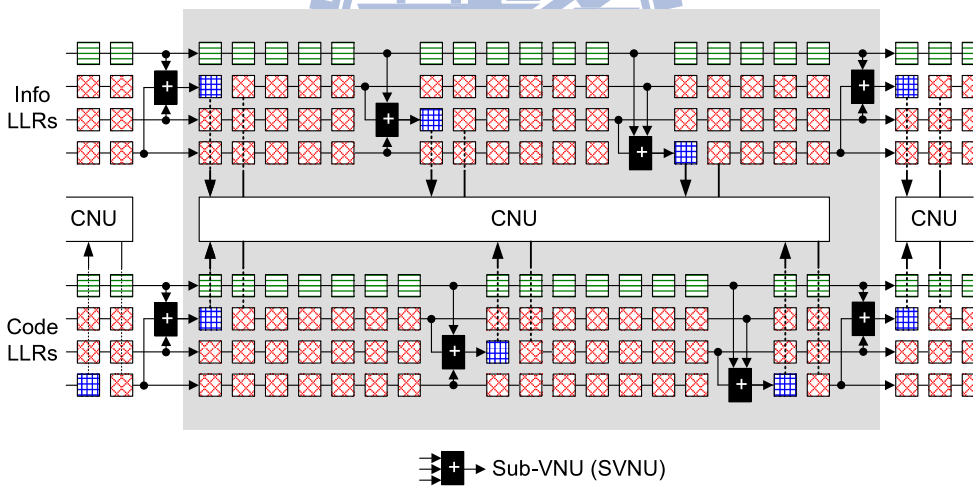


Figure 4.5: Processor architecture of on-demand variable node activation scheduling.

It can be seen from Figure. 4.5, a variable node unit (VNU) can be disassembled into J sub-VNUs (SVNUs) and distributed within a processor. Before each check node unit is activated, the sub-VNU calculates single variable-to-check message instead of calculating J variable-to-check messages in parallel. Therefore, the check-to-variable messages could

be obtained by the partial computation of the variable nodes, and then these updated messages could be used to compute the remaining variable-to-check messages. The major difference from the standard decoding schedule is the order of updating procedures, and it is worth to note that the both computational complexity is the same.

To be more specific, in Figure. 4.6, n_1 is a variable-to-check message which is obtained by the summation of the intrinsic channel value u and two check-to-variable messages m_2 and m_3. Then this recently generated n_1 is accessed by the check node unit immediately to compute new check-to-variable message $m_1^{'}$. Also, the message $m_1^{'}$ is available for next sub-VNU to calculate a new variable-to-check message n_2 for further check node updating within the same iteration. It is clear that the frequency of message passing between check node and variable node is significantly increased. Using this scheduling, the messages can flow faster through the Tanner graph.

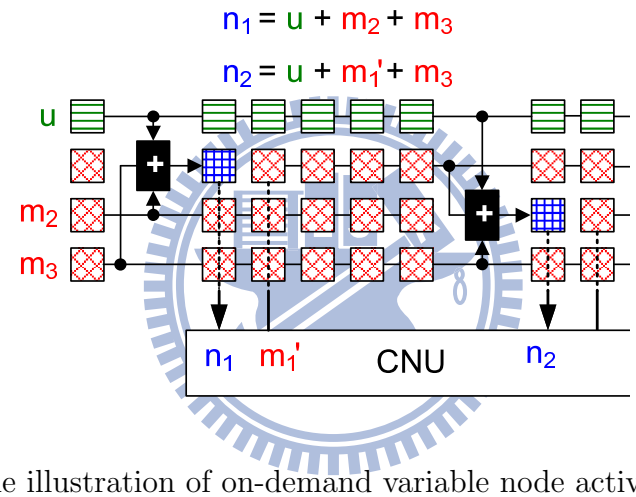


Figure 4.6: The illustration of on-demand variable node activation scheduling.

In Figure. 4.7, we show the floating point BER performance comparisons of standard schedule versus on-demand schedule for a $(491, 3, 6)$ time-varying LDPC convolutional code. It can be seen that the performance which uses on-demand schedule with 25 iterations is almost identical to the performance that uses standard schedule with 50 iterations. This indicates that the on-demand schedule converges twice faster than the standard schedule due to making use of the most recent information. In other words, the on-demand variable node activation scheduling allows the decoder to improve performance and achieve a lower BER for the same number of iterations. Therefore, for a given BER, once on-demand scheduling is employed, the required processor numbers

and initial decoding delay are reduced nearly by half comparing to the standard scheduling. Figure. 4.8 shows that on-demand variable node activation schedule has a faster convergence speed than flooding schedule for all iterations. Figure. 4.9 gives the BER performance of NMS algorithm with scaling factor 0.75 using on-demand variable node activation schedule with the rates ranging from 1/2 to 5/6.

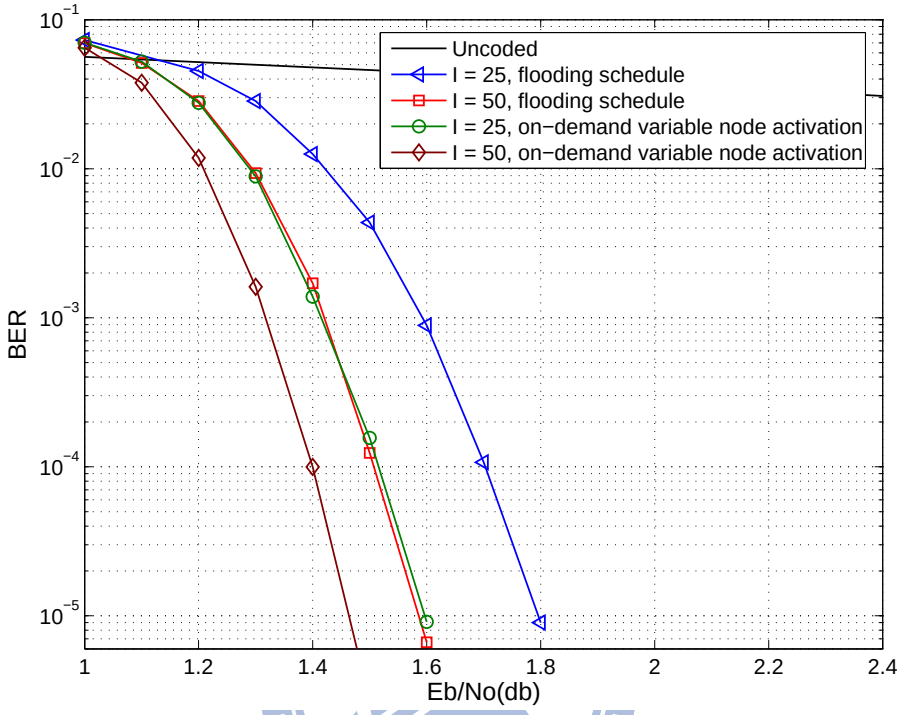


Figure 4.7: BER performance of on-demand variable node activation scheduling.

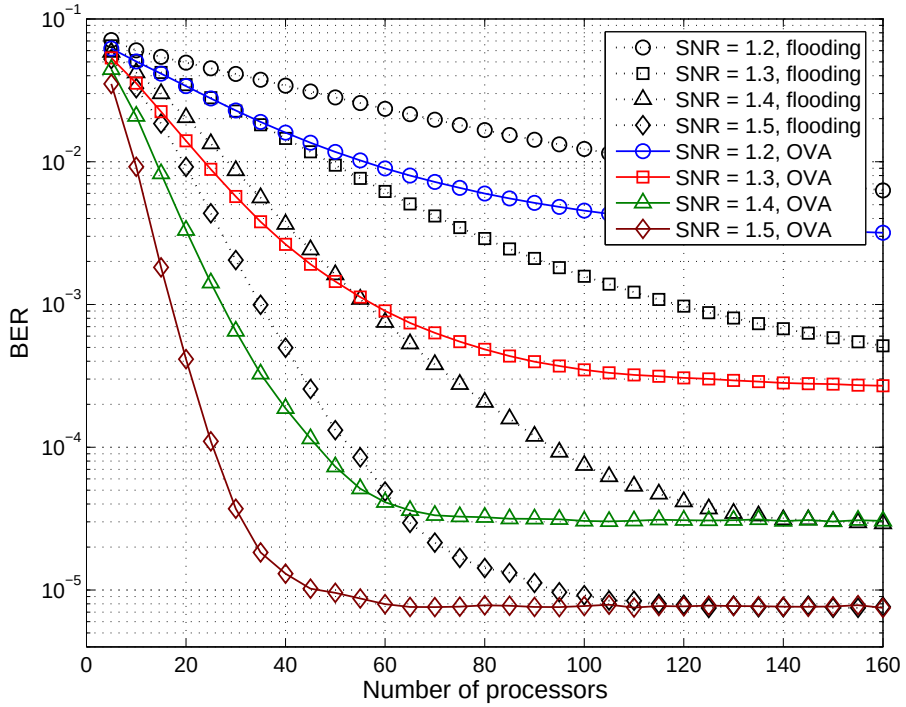


Figure 4.8: The dependence of the BER on the number of processors I at different SNRs using on-demand variable node activation schedule.

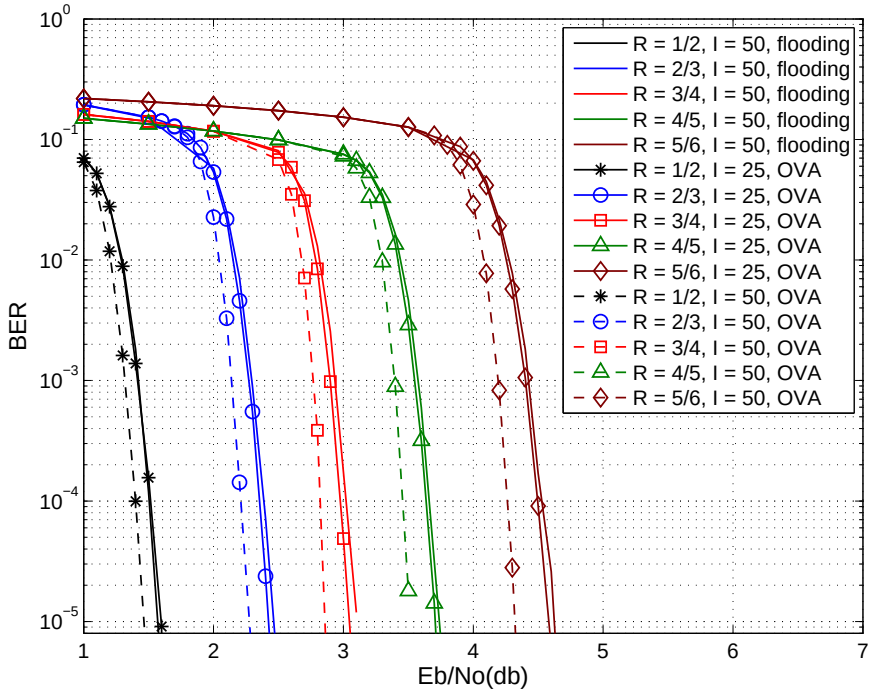


Figure 4.9: BER performance under different code-rates using on-demand variable node activation schedule.

4.1.3 OVA Scheduling with Concealing Channel Values

Since the equations in Figure. 4.6 of these two variable-to-check messages n_1 and n_2 have common terms u, we may calculate n_2 from n_1 by deducting m_2 and adding $m_1^{'}$. Each sub-VNU can be further disassembled into a pre-SVNU and a post-SVNU. The pre-SVNU performs the subtraction operation, and the post-SVNU performs the addition operation. Consequently, the channel values are concealed in variable-to-check messages. We named this newly proposed technique the on-demand variable node activation scheduling with concealing channel values. Figure. 4.10 illustrates the procedure to conceal the channel values and the corresponding pipeline decoder architecture is presented in Figure. 4.11.

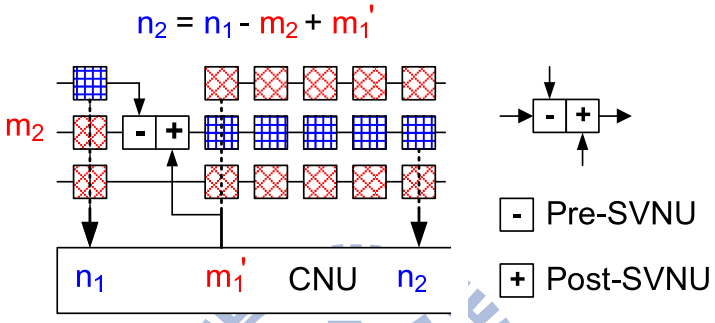


Figure 4.10: The illustration of on-demand variable node activation scheduling with concealing channel values.

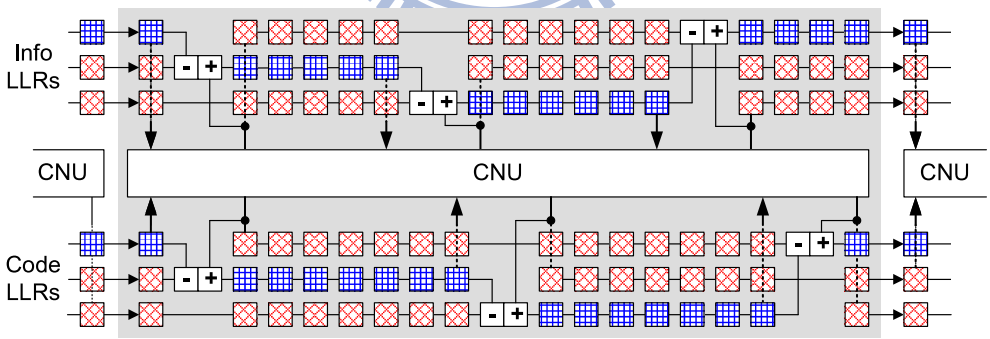


Figure 4.11: Processor architecture of on-demand variable node activation scheduling with concealing channel values.

The original processor architecture requires $c \times (J+1) \times (m_s+2)$ shift registers. With the help of this technique, the number of required shift registers is reduced to

$c \times J \times (m_s + 2)$. Assume that the quantization of messages is chosen as 6 bits and J equals 3 in the decoder implementation. The storage requirement of original processor architecture is

$$6\ bits \times 8\ rows \times (m_s + 2) = 48 \times (m_s + 2)\ bits.$$

For example, in Figure. 4.11, the register in red requires 6 bits and the register in blue requires 8 bits. The storage requirement of our proposed processor architecture is

$$(6\ bits \times 4\ rows + 8\ bits \times 2\ rows) \times (m_s + 2) = 40 \times (m_s + 2)\ bits.$$

Thus, the storage space of channel values can be removed from processors to save 17% memory. Although the channel values are concealed, the decoding results are identical to the scheduling without concealing channel values. Moreover, with our proposed concealing channel values technique, the storage requirements of the decoder are further reduced.

Figure. 4.12 shows the fixed point simulation of the $(491, 3, 6)$ LDPC convolutional code with 5 iterations using normalized min-sum algorithm with scaling factor 0.75. We can see that the quantization $(6, 2)$ for the LLRs, namely 4-bit for integer part and 2-bit for fraction part, has the minimum costs but with acceptable performance loss. Finally, we implemented 5 processors in our decoder chip. Figure. 4.13 is the BER performance of the rate-compatible $(491, 3, 6)$ time-varying LDPC convolutional code under AWGN channel. In contrast to log-BP algorithm with 10 processors, the proposed algorithm with 5 processors can achieve similar or even better performance in all code-rates. Therefore, only half number of processors are required under the same performance, leading to half decoding latency reduction as well.

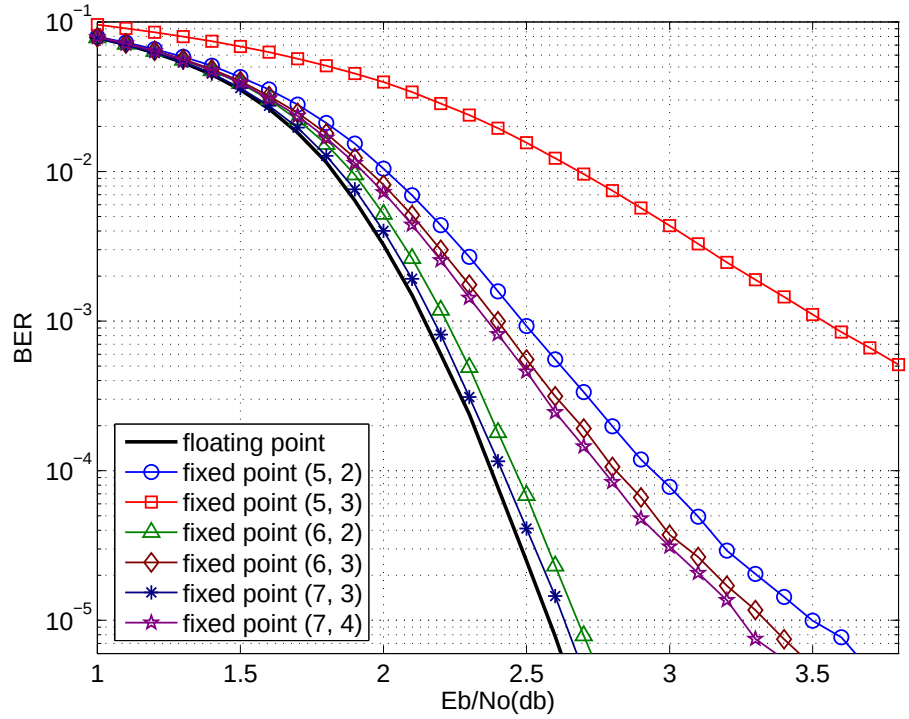


Figure 4.12: Fixed-point simulation results using OVA scheduling with 5 iterations.

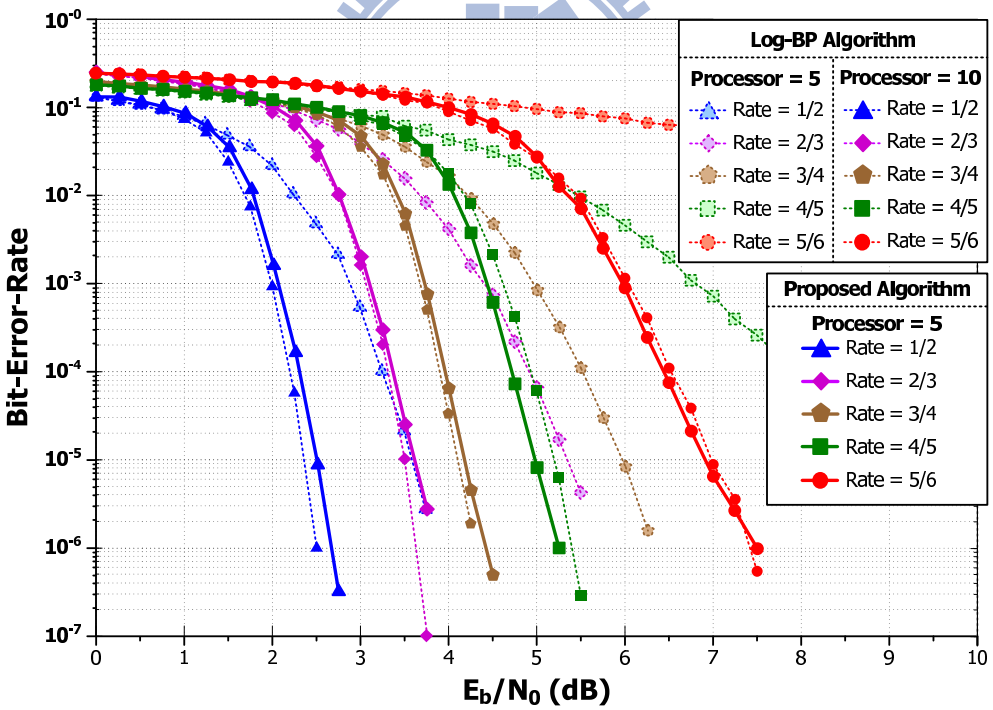


Figure 4.13: BER performance of log-BP algorithm (floating-point) and our proposed scheduling in normalized min-sum algorithm with scaling factor 0.875 (fixed-point (6,2)) under AWGN channel.

4.2 Node-Level Optimization

4.2.1 Folding Architecture

In the original pipeline decoder architecture, a number of processors are concatenated together to decode on different regions over the Tanner graph simultaneously, thus the decoding is parallel in the iteration dimension. Assume the decoder can operate at f_{clk} MHz clock frequency, since the decoder can only decode one bit in one cycle, the information throughput will be limited to only f_{clk} Mb/s. For high-speed applications, the parallelization for LDPC convolutional code encoder and decoder is desirable. In the literatures, the concepts of node level parallelization are proposed in [21]. However, their decoder architecture requires the shuffle networks to overcome the problem of memory misalignments. Furthermore, the parallelism over a hundred is necessary to achieve the decoding throughput of 1 Gbps.

In order to provide a solution with lower complexity, we propose the folding technique for node level parallelization to design high throughput LDPC convolutional code encoder and decoder. The idea of folding technique is to look ahead the bits that are going to participate in the encoding or decoding operations. For parallel encoder using folding technique, total ρ information bits can be encode at the same time instant, where ρ is defined as the folding factor or the parallelization factor. The length of delay lines in the encoder are folded by a factor of ρ. And the XOR gate for encoding operation have to duplicate to ρ units. Although the similar parallel encoder architecture has been proposed in [22], the parallel encoder architecture they proposed requires large multiplexers for phase selection. For our folding technique, the parallelism is chosen as the multiple of time period. This allows the original time-varying encoding connections to transfer to fixed time-invariant connections. Thus, the multiplexers are no longer required in our encoder architecture, which is the major difference from [22].

Figure. 4.14 shows the parallel decoder architecture using folding technique. It can be seen that each FIFO delay line in the conventional processor is folded to ρ FIFO delay lines. In other words, each FIFO delay line is segmented by ρ factor to support required bandwidth. With this modified FIFO structure, sufficient input data could be provided for operation units. For instance, in Figure. 4.14, given that the folding factor $\rho = 3$, each

shift register of length 16 is replaced by 3 shift registers of length 6. Namely, the decoding delay is reduced from 16 clock cycles to 6 clock cycles for a unit processor. Also, both check node units and variable node units are duplicated to ρ units. Using this approach, the decoder throughput becomes $(\rho \times f_{clk})$ Mb/s. However, the maximum value of folding factor is restricted by the code structure. Generally, the larger constraint length LDPC convolutional codes with careful code constructions would allow higher folding factor. To be mentioned that folding technique primarily duplicates the combinational logic while the sequential circuits are only slightly increased. It is evident that the folding technique not only increases the decoder throughput, but also reduces the decoding delay by a factor of ρ at the same time.

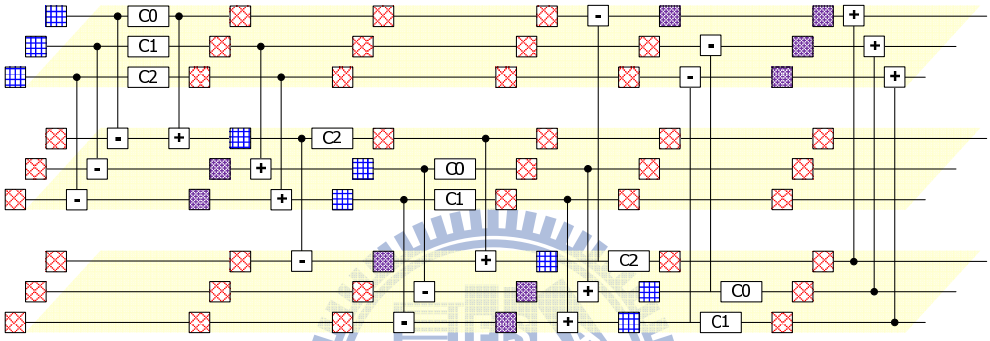


Figure 4.14: Folding technique (only information part in shown).

Based on the conventional decoder architecture, Table. 4.1 presents a comparison of storage requirements and decoding latency for a unit processor with different folding factors. It can be seen easily that folding technique not only directly duplicates the throughput, but also significantly reduces the decoding latency. Moreover, this technique only slightly increases the hardware costs. In particular, the overhead of the duplication of check node units and variable node units is minor comparing to the overall cost of a processor. We apply the folding technique to the time-varying $(491, 3, 6)$ LDPC convolutional code with period of 3. Since the shortest distance of adjacent check node accessing positions is $70 - 56 = 14$, the maximum folding factor of this code is 12. Therefore, a 12 times decoding throughput increase while the decoding delay is reduced from 493 clock cycles to 43 clock cycles for single processor.

Table 4.1: Comparison of hardware cost and decoding latency with different folding factors based on the conventional decoder architecture.

Folding factor ρ	1	3	12
Required bits for storage	23664	23904	24768
Number of CNUs	1	3	12
Number of VNUs	1	3	12
Throughput	f_{clk}	$3 \times f_{clk}$	$12 \times f_{clk}$
Decoding latency for a unit processor (cycles)	493	166	43

In addition, the concepts of parallelization can be described mathematically. Figure. 4.15 shows the parity-check matrix of $(14, 3, 6)$ LDPC convolutional code given in (3.13). Given that folding factor $\rho = 3$, every 3 rows in the parity-check matrix can be grouped to form a rate $R = 3/6$ LDPC convolutional code with syndrome former memory $m_s = 5$.

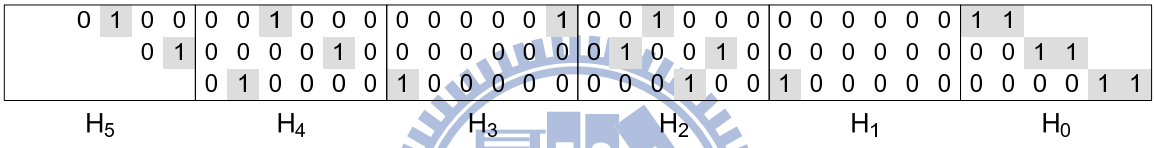


Figure 4.15: Parity-check matrix of $(14, 3, 6)$ LDPC convolutional code.

From the graph illustration in Figure. 4.15, we can see that the polynomial parity-check matrix becomes

$$H(D) = \begin{pmatrix} 1 & 1 & D^2 + D^4 & D^5 & 0 & D^3 \\ 0 & D^2 & 1 & 1 & D^2 + D^4 & D^5 \\ D + D^3 & D^4 & 0 & D^2 & 1 & 1 \end{pmatrix}. \tag{4.1}$$

The same result can be derived from the parity check polynomials of the LDPC convolutional code. Using the parity check polynomial representation, the parity-check matrix can be described as

$$(D^2 + D^7 + D^{13})u_0(D) + (D^2 + D^9 + D^{16})v_0(D) = 0 \tag{4.2a}$$

$$(D + D^6 + D^{12})u_1(D) + (D + D^8 + D^{15})v_1(D) = 0 \tag{4.2b}$$

$$(1 + D^5 + D^{11})u_2(D) + (1 + D^7 + D^{14})v_2(D) = 0. \tag{4.2c}$$

Let $X = D^3$, we can rewrite these equations as

$$(D^2 + X^2 \cdot D + X^4 \cdot D)u_0(D) + (D^2 + X^3 + X^5 \cdot D)v_0(D) = 0 \tag{4.3a}$$

$$(D + X^2 + X^4)u_1(D) + (D + X^2 \cdot D^2 + X^5)v_1(D) = 0 \tag{4.3b}$$

$$(1 + X \cdot D^2 + X^3 \cdot D^2)u_2(D) + (1 + X^2 \cdot D + X^4 \cdot D^2)v_2(D) = 0. \tag{4.3c}$$

Given in (4.4), the polynomial parity-check matrix is the same as (4.1) if X is replaced by D.

$$H(X) = \begin{pmatrix} 1 & 1 & X^2 + X^4 & X^5 & 0 & X^3 \\ 0 & X^2 & 1 & 1 & X^2 + X^4 & X^5 \\ X + X^3 & X^4 & 0 & X^2 & 1 & 1 \end{pmatrix}. \tag{4.4}$$

We apply this procedure on the time-varying $(491, 3, 6)$ LDPC convolutional code with period of 3. Let folding factor $\rho = 3$, the corresponding parity-check polynomials are listed in (4.5).

$$(D^2 + D^{58} + D^{375})u_0(D) + (D^2 + D^{220} + D^{408})v_0(D) = 0 \tag{4.5a}$$

$$(D + D^{198} + D^{458})u_1(D) + (D + D^{23} + D^{492})v_1(D) = 0 \tag{4.5b}$$

$$(1 + D^{70} + D^{485})u_2(D) + (1 + D^{181} + D^{236})v_2(D) = 0 \tag{4.5c}$$

With $X = D^3$, we can rewrite the equations as

$$(D^2 + X^{19} \cdot D + X^{125})u_0(D) + (D^2 + X^{73} \cdot D + X^{136})v_0(D) = 0 \tag{4.6a}$$

$$(D + X^{66} + X^{152} \cdot D^2)u_1(D) + (D + X^7 \cdot D^2 + X^{164})v_1(D) = 0 \tag{4.6b}$$

$$(1 + X^{23} \cdot D + X^{161} \cdot D^2)u_2(D) + (1 + X^{60} \cdot D + X^{78} \cdot D^2)v_2(D) = 0. \tag{4.6c}$$

Finally, we can obtain a rate $R = 3/6$ time-invariant $(164, 3, 6)$ LDPC convolutional code, whose polynomial parity-check matrix is shown in (4.7). The columns of the parity-check matrix are rearranged to ensure systematic encoding. If the folding factor is chosen as a multiple of time period , the folding technique allows the original time-varying code to transform into a time-invariant code. Thus, the multiplexers for the configuration of time-varying connection are saved. Moreover, if the time-invariant code has quasi-cyclic symmetries, the encoder complexity of tail-biting LDPC convolutional codes may be reduced. We simulate the performance of a family of LDPC convolutional codes derived from (491,3,6) LDPC convolutional code with folding factors 3, 6, 9 and 12. The BER

performance of these codes is shown in Figure. 4.16. We can see that these codes perform very similarly even if the syndrome former memories vary greatly. Also given in (4.8) is the parity-check matrix of the $R = 12/24$ $(41, 3, 6)$ LDPC convolutional code.

$$H(D) = \begin{pmatrix} 1 & D^{19} & D^{125} & 1 & D^{73} & D^{136} \\ D^{152} & 1 & D^{66} & D^{7} & 1 & D^{164} \\ D^{161} & D^{23} & 1 & D^{78} & D^{60} & 1 \end{pmatrix} \tag{4.7}$$

$$H(D)= \begin{pmatrix} 1 & 0 & 0 & 0 & D^{5} & 0 & 0 & 0 & 0 & 0 & 0 & D^{32} & 1 & 0 & D^{34} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^{19} & 0 \\ D^{38} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & D^{17} & 0 & 0 & 0 & 0 & 1 & D^{41} & D^{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & D^{6} & 0 & 0 & 0 & 0 & D^{41} & 0 & 0 & 0 & D^{15} & 1 & 0 & 0 & 0 & D^{20} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & D^{31} & 1 & 0 & 0 & 0 & D^{5} & 0 & 0 & 0 & 0 & 0 & D^{18} & 0 & 1 & 0 & D^{34} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & D^{38} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & D^{17} & 0 & 0 & 0 & 0 & 1 & D^{41} & D^{2} & 0 & 0 & 0 & 0 & 0 \\ D^{40} & 0 & 0 & 0 & 0 & 1 & 0 & D^{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^{15} & 1 & 0 & 0 & 0 & D^{20} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & D^{31} & 1 & 0 & 0 & 0 & D^{5} & 0 & 0 & 0 & 0 & 0 & D^{18} & 0 & 1 & 0 & D^{34} & 0 & 0 & 0 \\ 0 & 0 & D^{16} & 0 & 0 & 0 & D^{38} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & D^{41} & D^{2} & 0 & 0 \\ 0 & 0 & 0 & D^{40} & 0 & 0 & 0 & 0 & 1 & 0 & D^{6} & 0 & D^{19} & 0 & 0 & 0 & 0 & 0 & 0 & D^{15} & 1 & 0 & 0 & 0 \\ 0 & D^{4} & 0 & 0 & 0 & 0 & 0 & 0 & D^{31} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & D^{18} & 0 & 1 & 0 & D^{34} \\ 0 & 0 & 0 & 0 & 0 & D^{16} & 0 & 0 & 0 & D^{38} & 1 & 0 & D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & D^{41} \\ 0 & D^{5} & 0 & 0 & 0 & 0 & D^{40} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & D^{19} & 0 & 0 & 0 & 0 & 0 & 0 & D^{15} & 1 \end{pmatrix} \tag{4.8}$$

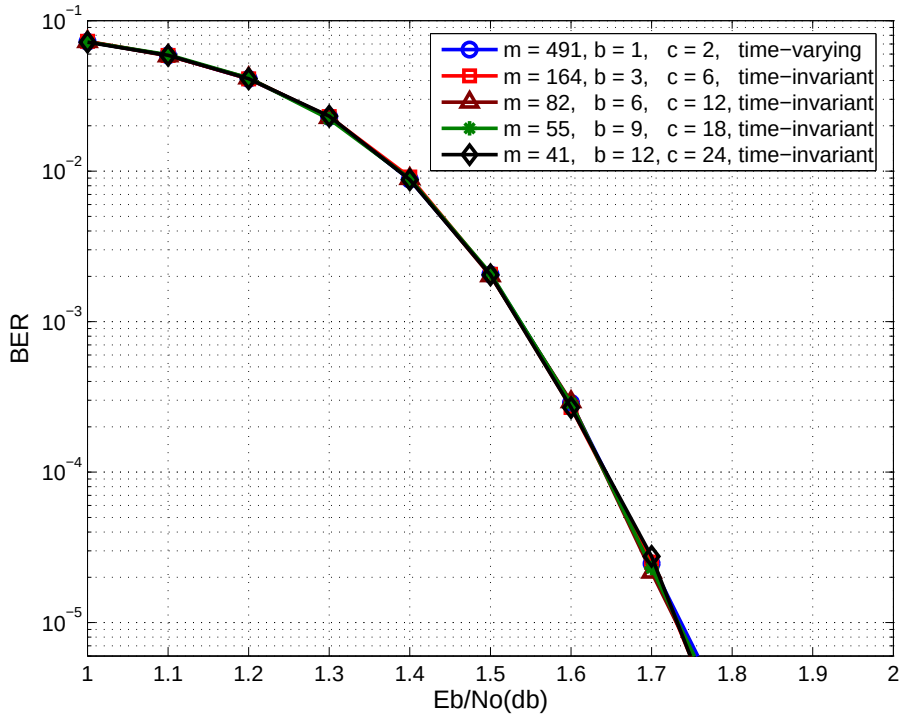


Figure 4.16: Performance of the $(491, 3, 6)$ LDPC convolutional code and the associated LDPC convolutional codes with different folding factors.

4.3 Bit-Level Optimization

4.3.1 Retiming Technique

According to the previous mentioned on-demand variable node activation scheduling with concealing channel values, the channel values are concealed in the variable-to-check messages. When the channel values are concealed within the summation values, the bit-width of each message should be adjusted to avoid truncation error. In the situation of w-bit channel value, the summation of one channel value and two check-to-variable messages needs $(w+2)$-bit. Since the operations of pre-SVNU and post-SVNU are independent, they can be re-timed such that the messages between them only need $(w+1)$-bit. From the illustration in Figure. 4.17, we observed that, as long as the computation of sub-VNU is completed before the check node accesses the messages, the result is identical to the original operation. In order to achieve a maximum saving in hardware cost, we let the computation of post-SVNU to perform at the position just before check node accessing the messages.

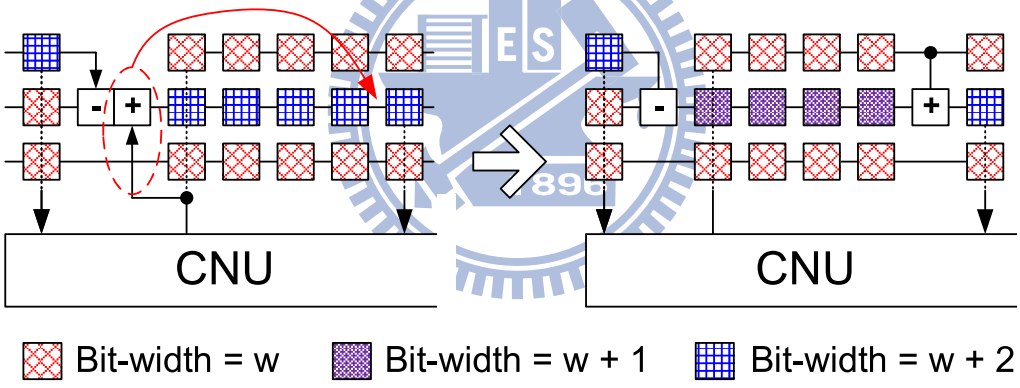


Figure 4.17: Retiming of sub-VNUs.

Figure. 4.18 depicts the bit-level optimized processor architecture, the message output from the check node unit is w-bit. The message between the subtraction and addition needs $(w+1)$-bit. And the message output from the post-SVNU which concealed channel values requires $(w+2)$-bit. In particular, the critical path of the conventional processor is dominant by the check node unit due to large sorters are required. Although the on-demand variable node activation scheduling with concealing channel values can accelerate the decoding convergence speed, it induces one more adder delay and results to a longer

critical path. With the retiming technique for sub-VNUs, the critical path from check node unit to post-SVNU could be diminished by one adder delay. As a consequence, the retiming of sub-VNUs causes that the critical path of a unit processor remains the same while reducing memory requirements. This technique is especially useful to large constraint length LDPC convolutional codes for the long distance between two check node inputs.

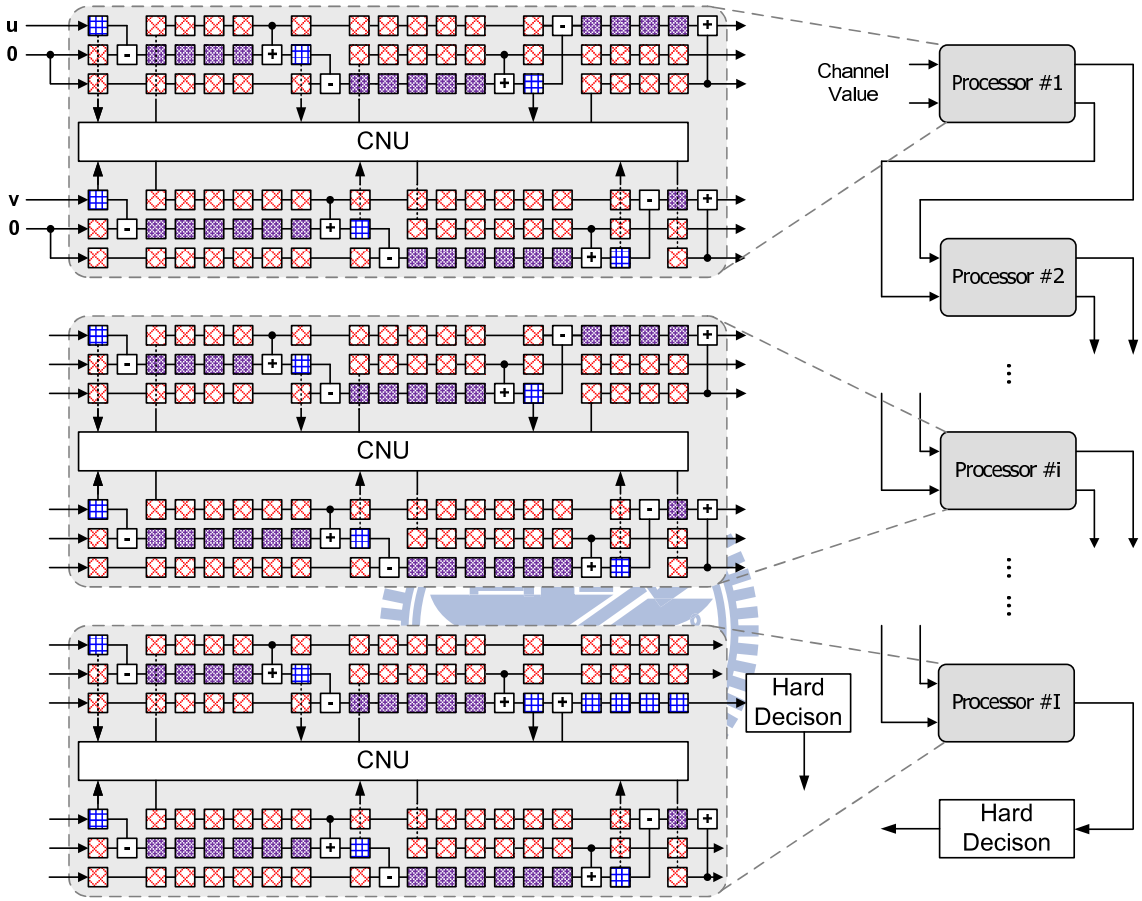


Figure 4.18: Processor architecture with retimed sub-VNUs.

In Table. 4.2, we give a comparison of the storage requirements of three techniques which have been introduced so far. Assume that the quantization of LLRs is 6 bits, the required numbers of 6-bit, 7-bit and 8-bit registers for the $(491, 3, 6)$ LDPC convolutional code with folding factor $\rho = 12$ are compared. When the OVA schedule with concealing channel values is adopted, the storage requirements is reduced by around 17%. With retimed sub-VNUs, the required number of 8-bit registers is minimized, a 20% storage reduction is reached.

Table 4.2: Comparison of the storage requirements with different techniques.

	6-bit reg.	7-bit reg.	8-bit reg.	Total required bits
Standard schedule	4128	0	0	24768
OVA scheduling with concealing channel values	2064	0	1032	20640
Retiming the sub-VNUs	2064	960	72	19680

4.4 Hybrid-Partitioned FIFO

There are two kinds of architectures for implementation of the LDPC convolutional code decoder in the literature, register-based and memory-based architecture. The register-based architecture enjoys the advantage of bandwidth flexibility, thus higher throughput can be easily achieved using folding technique. However, the large numbers of required registers would cause large hardware cost and high power consumption. The first ASIC realization using register-based decoder architecture is proposed in [23]. Although the memory-based architecture saves the power consumption and silicon area, the problem of memory access collisions is serious when high node parallelization is used. In the meantime, folding technique could increase throughput, however it will divide the FIFOs into more pieces and make it difficult to use memory-based decoder architecture. In [24], the memory-base decoder architecture are used for FPGA implementations.

For time-varying LDPC convolutional codes with large folding factor, neither register-based FIFO nor memory-based FIFO is suitable. Therefore, we present a hybrid-partitioned FIFO structure to support large bandwidth requirement and also minimize the power consumption.

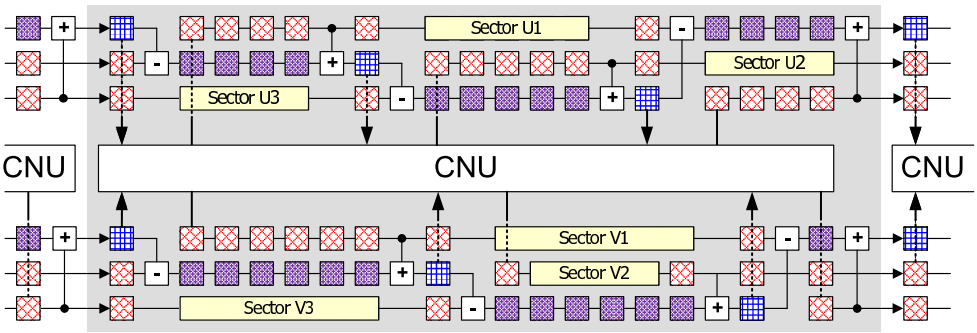


Figure 4.19: Finding longest continuous sectors in each FIFO.

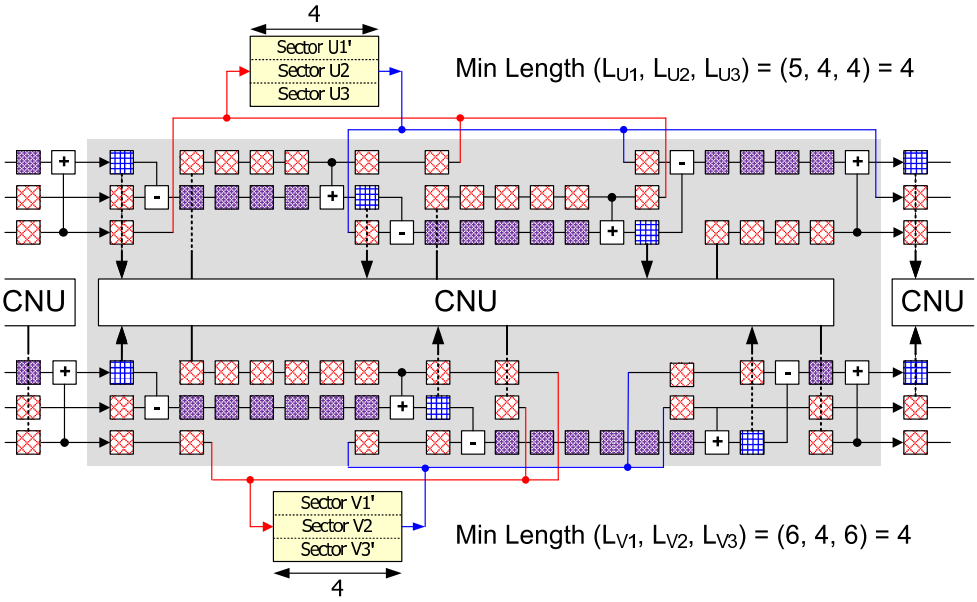


Figure 4.20: Processor architecture after merging sectors into memories.

Figure. 4.19 shows the illustration of the hybrid-partitioned FIFO structure. The first step of this technique is to calculate the length of the longest continuous sectors of every folded row. We only consider the continuous section of shift registers without messages accesses. Then the sectors are to be merged into one memory bank together, where the depth of the memory bank is the minimum value of the sector lengths. If the original sector is larger than the memory depth, the excess part is still stored in registers. This procedure continues to merge sectors until the memory depth is less than a pre-defined parameter. As shown in Figure. 4.20, the longest lengths of continuous sectors within the information part of the processor are 5, 4 and 4. Hence, the depth of the memory is 4 because of the minimum length of these 3 sectors is 4.

Note that this simple example is only for illustration. For the LDPC convolutional code with larger constraint length, the lengths of continuous sector within a processor will be longer. Large amounts of data are saved in the memory banks instead of registers, thus leads to a significant saving in power consumption. Besides, if large folding factor is employed, the number of continuous sectors in a processor will increase, and the lengths of continuous sectors will shorten. However, these continuous sectors can still be merged into several memory banks. This merge operation allows that the segmented and shorten sectors being integrate to a unified memory bank. The memory bank is implemented as

a circular buffer whose positions for read and write operations are tracked by address pointers. We use two-port memories such that read and write operation can perform in the same clock cycle. When a new message is received, this newest message is written into a proper position of the circular buffer and then the circular buffer outputs the oldest message. Therefore, the shifting operations in the FIFOs no longer exist to achieve a low-power implementation.

In our work of the $(491, 3, 6)$ time-varying LDPC convolutional code decoder with parallelism of 12, about 50% of messages in a unit processor are partitioned into 3 two-port memories. The sizes of these 3 two-port memories are listed in Table. 4.3 and the total storage space is

$$(36 \times 144) + (20 \times 144) + (32 \times 76) = 10.5\ Kbits.$$

For chip design, sink is the number of registers which are connected in the end of the clock tree. The less the sink number is, the less power the chip consumes. In Table. 4.4, we give the comparisons of clock tree loading using register-based FIFO and hybrid-partitioned FIFO. It can be seen that, in the situation of similar single processor area, the hybrid-partitioned FIFO structure can reduce the numbers of clock buffers and sinks efficiently. The effects of reducing the number of clock buffers and sinks can greatly minimize the clock tree loading during the physical design stage. This improvement can be translated into reduced power consumption. Moreover, the power reduction of merging the continuous sectors to several memory banks is more than merging sectors to single large memory bank. Table. 4.4 shows that, compared to register-based FIFO, the hybrid-partitioned FIFO structure using 3 memory banks achieves a 54% reduction of clock buffers.

Table 4.3: The size of memory banks in each processor.

	Memory size
Memory 1	20 words × 144 bits
Memory 2	32 words × 76 bits
Memory 3	36 words × 144 bits

Table 4.4: Comparison of clock tree loading.

Folding factor (ρ)	12	12	12
Number of processors	1	1	1
Number of memories	0	1	3
Clock buffer	359	278	194
Sink	17968	13676	9068
Area (um^2)	750×520	750×520	750×540

Chapter 5

Implementation Result

Our proposed rate-compatible time-varying (491, 3, 6) LDPC convolutional code decoder chip integrates the fallowing techniques. For algorithm-level optimization, the on-demand variable node activation scheduling with concealed channel values is presented to improve decoding performance and reduce storage requirements. For node-level optimization, the folding technique significantly increases the decoding throughput and reduces the decoding latency at the same time. For bit-level optimization, the retiming technique is applied to SVNU, and this approach can reduce the critical path of a unit processor while further reducing the memory requirements. Finally, hybrid-partitioned FIFO structure is employed to save power consumption.

The test chip was implemented in a UMC 90nm 1P9M CMOS process and measured using the Agilent 93000 SOC Series provided by CIC. The test chip includes the proposed encoder, proposed decoder with 5 processors, random number generators, and additive white Gaussian noise (AWGN) engines. In section 5.1, we show the architecture of the test chip that enables high-speed on-chip testing. Chip measurement results of power consumption and operating frequency are given in section 5.2. And then the key features of test chip are summarized in section 5.3. We compare our design in this section with the state-of-the-art decoder implementations.

5.1 Testing Consideration

The block diagram of proposed test chip is shown in Figure. 5.1. This test chip comprises the random number generators, the encoder with folding factor of 3, the decoder with folding factor of 12, additive white Gaussian noise (AWGN) channel module, puncture and de-puncture modules, SRAM for data buffering, and control module for performing a number of testing operations. For chip measurement consideration, the random number generators and AWGN engines are embedded in the test chip for built-in-self-test (BIST). Consequently, the random number generator can provide sufficient test patterns on-chip for real-time decoding. The output signals can be used to verify the functionality of the test chip. Here we use two identical random number generators to avoid using large FIFO for data buffering. Besides, the puncture and de-puncture block allows the LDPC convolutional code to support 5 different code-rates. In order to handle chip failures when unexpected errors occurred in any module, we designed several testing modes to identify the error location. Once the error location is identified, the control circuit will bypass the error module during chip measurement.

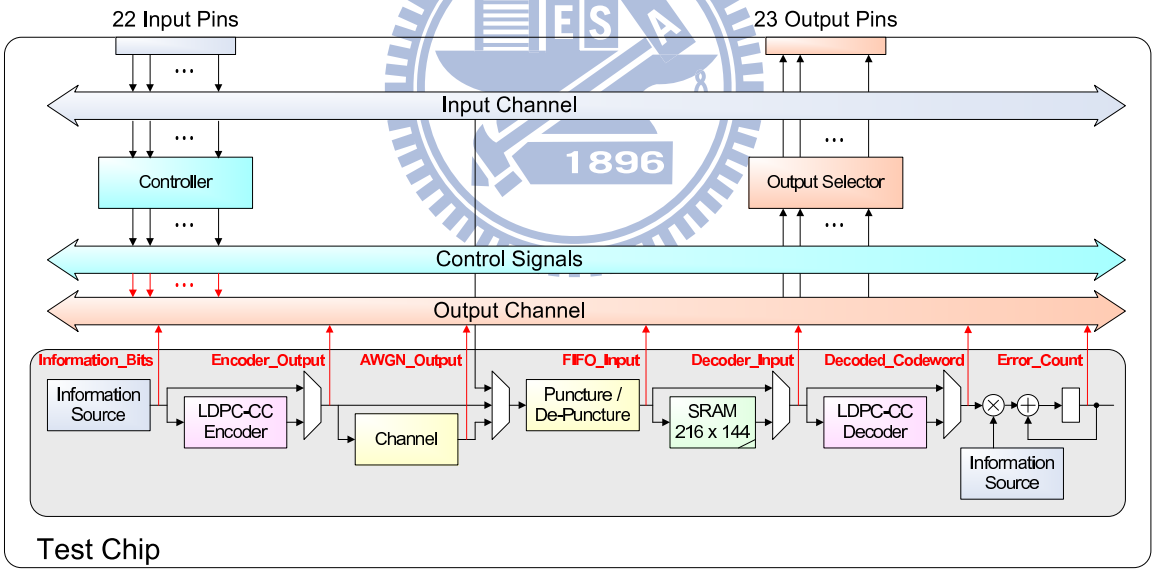


Figure 5.1: Block diagram of test chip.

Our test chip contains the following testing modes.

- **Normal function operation**

  In this mode, we can simulate the error correcting performance and measure the

power consumption of proposed LDPC convolutional codec using 5 processors under different signal-to-noise ratios. In addition, the error correcting performance with different code-rates can be also obtained.

- **Normal function without noise**

  This operation bypasses the AWGN engine for functionality debugging. The encoded data will be transmitted directly to the decoder, if there is no failure in any one of encoder and decoder, the encoded data will be correctly decoded.

- **Uncoded**

  In this mode, the encoder and decoder are bypassed. We can simulate the uncoded BER performance and measure the power consumption when LDPC convolutional codec is disable.

- **Normal function but bypass FIFO**

  Since the decoder is composed by a number of concatenated processors, a failure occurs in any one of them would cause a complete failure of the decoder. To solve this problem, Figure. 5.2 shows our proposed test circuits for the decoder. If a processor failure occurs, the BYPASS signals will control the configuration of processors, and FINAL signals will determine the location of the last processor to perform hard decision operation. Thus, the measurements can still work successfully when malfunction happened in the processor.

- **Testing mode**

  The test patterns are generated on-chip by random number generator. In addition, we designed a testing mode to feed test patterns from the input pins of the test chip to de-puncture module for decoding.

- **External control**

  Since the control module plays an important role in improving design testability, any defect occurs in the controller will lead to a catastrophic error. With this testing mode, the control signals of the system operations can be configured from the input pins of the test chip.

- **Repeat mode**

For accurate power measurements, the decoder is required to run long periods of time to obtain average power dissipation. This mode allows the decoder to run and repeat the decoding procedure. Hence, the operation time of decoding is extended, a accurate power measurement can be achieved.

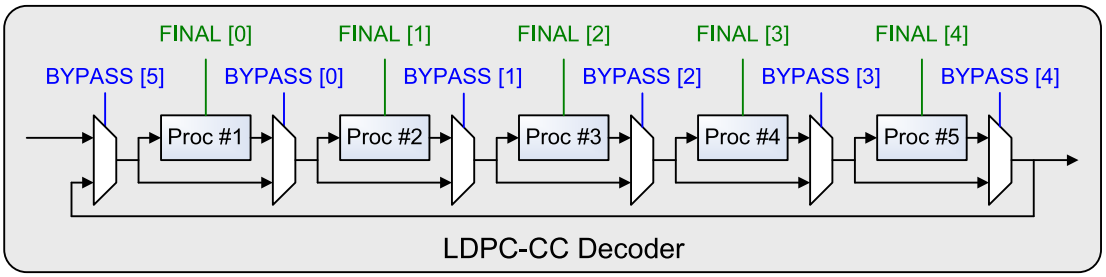


Figure 5.2: Testing circuits of proposed decoder.

5.2 Chip Measurement Results

The measurement result of the decoder operating at an SNR of 2.5 dB under different supply voltages is shown Figure. 5.3. It is clear that the information throughput increases as the supply voltage increases. The result shows that the decoder draws 284 mW under 1.2V supply voltage while running at 198 MHz. Since the folding factor equals 12, the information throughput of proposed decoder achieves $0.198 \times 12 = 2.37$ Gb/s. When supply voltage is scaled down to 0.8V, the power is reduced to 90.2 mW with an energy efficiency of 0.0114 nJ/bit/proc. Besides, the Shmoo plot is shown in Figure. 5.4. We choose the SNR of 2.5 dB which can achieve a BER of 10^{-5} to simulate the valid range of operating frequency and supply voltage. The simulated frequencies range from 120 MHz to 220 MHz with a step of 2 MHz. And the supply voltages range from 0.8V to 1.2V with a step of 0.01V. The green blocks in the Shmoo plot indicate that the decoder is capable of decoding the encoded patterns correctly. As shown in Figure. 5.4, the measured maximum operating frequency of the decoder is 198 MHz, hence the information throughput can reach 2.37 Gb/s.

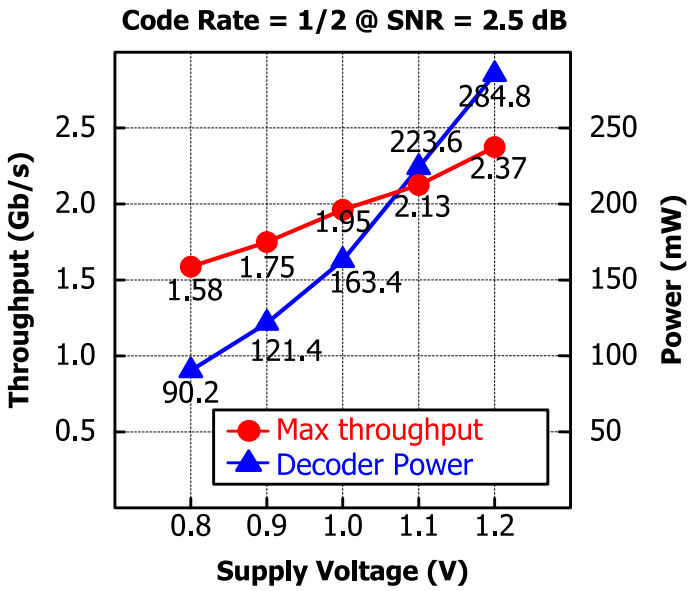


Figure 5.3: Measurement results under different supply voltages.

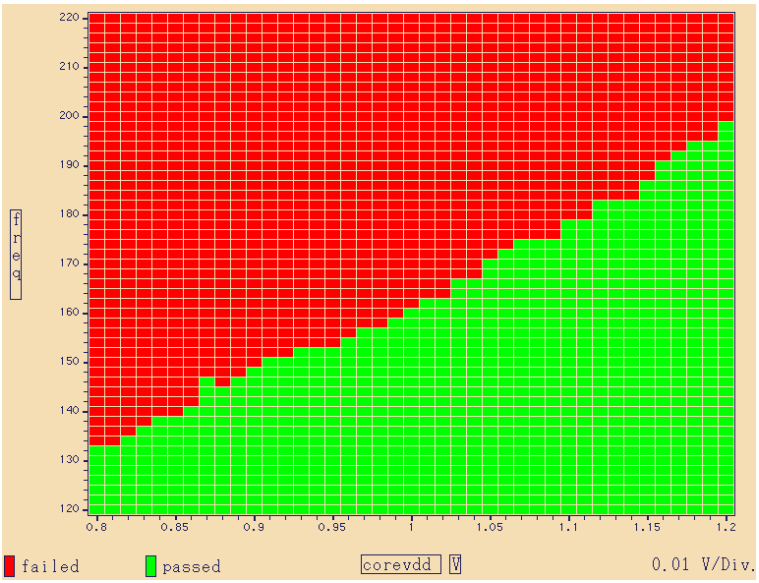


Figure 5.4: Shmoo plot of test chip.

5.3 Summary and Comparison

Table. 5.1 presents a brief summary of the proposed $(491, 3, 6)$ time-varying LDPC convolutional code test chip. Implemented in UMC 90nm process, the chip area including testing circuits is $2.37 \times 1.14 = 2.7mm^2$. With an 87.8% chip utilization, the decoder chip only occupies 2.24 mm^2 area. In addition, the proposed decoder can support 5 code-

rates from 1/2 to 5/6 through puncturing technique. The messages for iterative decoding are quantized to 6 bits. We implemented 5 processors in the decoder, and each processor contains 3 two-port memories. Therefore, the test chip totally contains 15 memory banks, and the size of these memories is 52.5 Kb. The chip micrograph is shown in Fig. 5.5. Figure. 5.6 lists the gate-count profile of the test chip. Test chip totally contains 867K gate counts, and each processor contains 145K gate counts. The AWGN modules and decoder are 8% and 84% of the total gate counts respectively. A comparison of post-layout results and measurement results of the proposed decoder chip is given in Table. 5.2, and the corresponding illustration is shown in Figure. 5.7.

Table 5.1: Chip summary.

Process	UMC 90nm 1P9M	
Code	$(491, 3, 6)$ LDPC-CC with T=3	
Code Rate	1/2, 2/3, 3/4, 4/5, 5/6	
Constraint Length	984	
Input Quantization	6 bits	
Chip Utilization	87.8%	
Parallelization Factor	12	
Gate Count	867 K	
Processor Number	5	1
Memory	52.5 Kb	10.5 Kb
Decoder Area	2.24 mm^2	0.448 mm^2
Max. Clock Frequency [()]	198 MHz	207 MHz
Max. Data Rate [(1)]	2.37 Gb/s	2.48 Gb/s
Decoder Power [(1)]	284 mW	-
Energy Efficiency [(1)]	0.024 nJ/bit/proc.	

[(1)] measured at BER=10^{-5} without early-termination at 1.2V supply for $R = 1/2$

A comparison with state-of-the-art designs is given in Table. 5.3. We compare the throughput of our decoder with other LDPC convolutional code decoders. In [16], the measured operating frequency is 600 Mb/s. Since the parallelism of this implementation

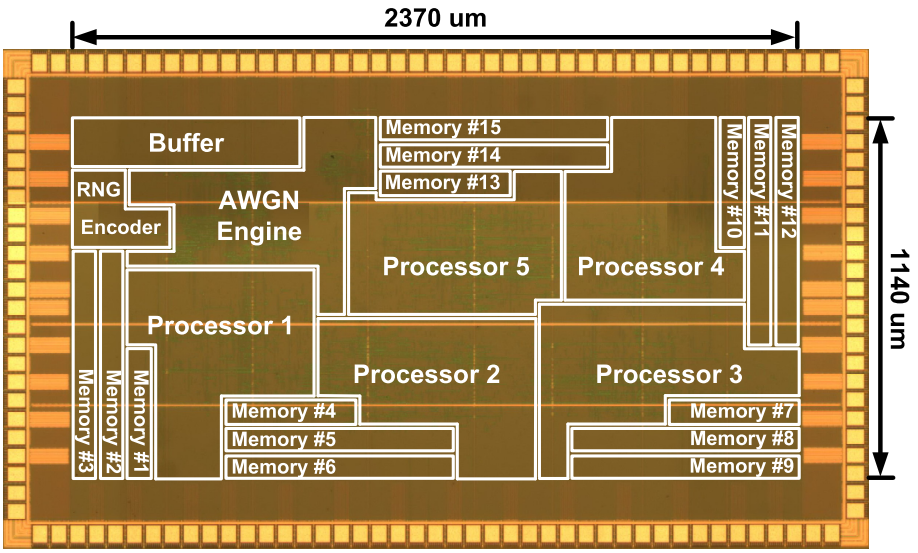


Figure 5.5: Chip micrograph.

Module	Gate Count	Unit
AWGN	23.3 K	3
Processor	145.0 K	5
Buffer	42.3 K	1
Chip	867 K	1

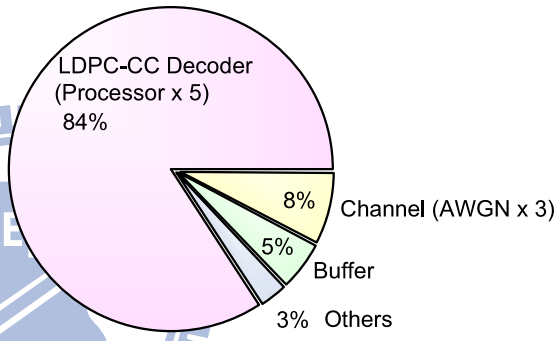


Figure 5.6: Gate-count profile.

is 1, the maximum throughput is 0.6 Gb/s. In [17], the parallelism is 8. Synthesis results show that the decoder achieves 2.0 Gb/s throughput at 250 MHz clock frequency. For our design, the measured maximum operating frequency is 198 MHz. With a parallelism of 12, the maximum throughput reaches 2.37 Gb/s. To compare the area of a unit processor with other designs, the silicon area occupied by 3 processors in [16] is 1.507 mm^2. Thus, the area of a unit processor is 0.502 mm^2. In [17], the area of a unit processor is 0.924 mm^2. Although the constraint length of our implemented code is larger than other designs, each processor only takes 0.448 mm^2. Therefore, this work provides higher throughput, less area, and better energy efficiency when compared with previously reported LDPC convolutional code decoders.

Compared with the LDPC block code decoder in [25], this work has similar energy efficiency. For a fair comparison, the LDPC block code decoder area is normalized to 2.68 mm^2 due to different CMOS technologies are used. Our decoder has a 17% less normalized area with higher chip utilization. Compared with the Turbo decoder in [26], this work achieves much higher throughput with lower power and less die area.

Table 5.2: Comparison of post-layout results and measurement results.

	Post-Layout	Measurement	
Max. Clock Frequency (MHz)	284	198	132
Max. Data Rate (Gb/s)	3.40	2.37	1.58
Decoder Power (mW)	371	284	90.2
Energy Efficiency (nJ/bit/proc.)	0.021	0.024	0.011
Supply (V)	0.9	1.2	0.8

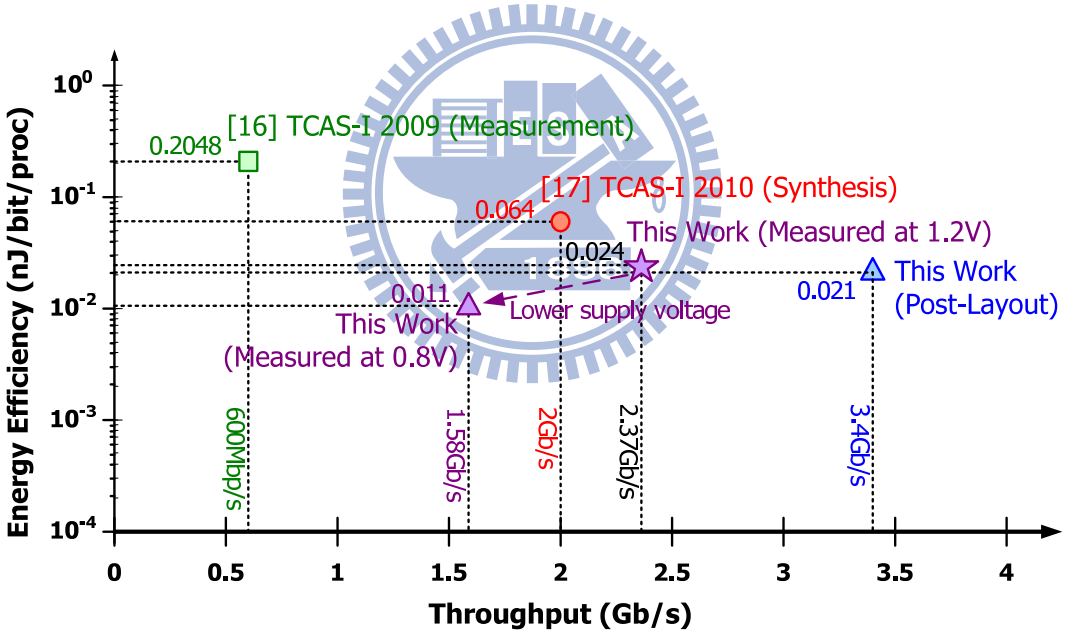


Figure 5.7: Comparison of throughput and energy efficiency with other LDPC convolutional code decoders.

Table 5.3: Comparison with state-of-the-art.

	This work	[16]	[17]	[25]	[26]
FEC Type	LDPC-CC	LDPC-CC	LDPC-CC	LDPC-BC	Turbo Code
Constraint Length / Block Size	984	258	960	672	3200
Code-Rate	1/2, 2/3, 3/4, 4/5, 5/6	1/2	1/2	1/2, 5/8, 3/4, 7/8	1/3
CMOS Technology (nm)	90	90	90	65	130
Input Quantization (bit)	6	8	6	6	-
Processor / Iteration	5	3	1	5	5.5
Memory (kb)	52.5	-	23.04	0	129
Chip Utilization (%)	87.8	-	-	73.3	-
Decoder Area (mm^2)	2.24	1.5	0.924	1.4	3.57
Max. Frequency (MHz)	198 [b]	600	250	197 [c]	302
Max. Data Rate (Gb/s)	2.37 [b]	0.6	2.0	3.57 [c]	0.39
Power (mW)	284 [b]	368.7	-	469.7 [c]	788.9
Energy Efficiency (nJ/bit/proc)	0.024 [b]	0.2048	0.064	0.0263 [a][c]	0.37 [a]
	Measurement	Measurement	Synthesis	Measurement	Measurement

[a] this unit is nJ/bit/iter in LDPC-BC and Turbo code

[b] measured at BER=10^{-5} without early-termination under 1.2V supply voltage for code-rate 1/2

[c] measured at BER=10^{-6} without early-termination for code-rate 1/2

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a rate-compatible $(491, 3, 6)$ time-varying LDPC convolutional code chip design. With algorithm optimized, the on-demand variable node activation scheduling with concealing channel values is used to achieve twice faster decoding convergence speed than the standard decoding schedule. Moreover, this technique also saves 17% message storage requirements. For the node level optimization, the folding technique is employed to reach a 12 times throughput increase while reducing the decoding latency by approximately 12 times at the same time. Also, the bit level optimization is utilized to retime the variable nodes in order to achieve higher clock frequency and less chip area. In particular, a hybrid-partitioned FIFO is introduced into the decoder implementation to avoid memory access collisions and lower power consumption.

Integrated with these schemes, the test chip of the proposed $(491, 3, 6)$ LDPC convolutional code decoder is implemented in a UMC 90nm CMOS process. The decoder part occupies 2.24 mm^2 within the core area of total 2.37×1.14 mm^2. Measurement results show that the decoder can provide a maximum throughput of 2.37 Gb/s under 1.2V supply voltage with a 0.024 nJ/bit/proc energy efficiency. It consumes 284mW of power for a 1.2V supply while running at 198 MHz. The power can be scaled down to 90.2mW at 0.8V supply with 1.58 Gb/s information throughput. Compared with previous LDPC convolutional code decoders, this work outperforms state-of-the-art designs in both throughput and energy efficiency. In conclusion, our proposed LDPC convolutional

code decoder has the potential to be one candidate for next-generation communication systems.

6.2 Future Work

In this implementation, the maximum folding factor of the $(491, 3, 6)$ LDPC convolutional code is only 12. Because the maximum folding factor is limited by the code structure, developing a new code construction algorithm for LDPC convolutional code is necessary. According to the simulations, the $(491, 3, 6)$ LDPC convolutional code cannot be terminated. Although tail-biting can be applied to this specification, it cannot be applied to the corresponding LDPC convolutional codes derived from the $(491, 3, 6)$ LDPC convolutional code using multiple of period as folding factors. Therefore, a construction algorithm that jointly considers the node parallelization, decoding performance, termination and tail-biting problem can be investigated. In particular, how to construct a code which is capable of exploiting high node parallelization while maintaining good error-correcting performance is a problem for future studies. A efficient high-speed implementation of the tail-biting LDPC convolutional code encoder and decoder which can handle data frames of variable length is also a research topic.

In addition, it is known that many factors affect the performance of the LDPC codes, such as the cycle in the Tanner graph, girth, trapping sets and minimum or free distance. Therefore, further research includes the study of the relationship between these factors and the performance of LDPC convolutional codes.

Appendix A

Termination of LDPC Convolutional Codes

In many communication systems and standards, the data bits are transmitted as packets with finite frame lengths. In order to ensure good error correcting performance near the end of the data frame, termination of LDPC convolutional code encoder is required to provide equal protection for each transmitted data bit. Termination is achieved by appending termination sequence, this sequence allows the encoder to return to the initial state, usually the all-zero state. For conventional feedforward convolutional code encoder, the encoder can be driven to all-zero state through all-zero sequence. Finding termination sequence of LDPC convolutional code is more complex than traditional convolutional codes due to its feedback encoder architecture. The method for termination of LDPC convolutional codes is proposed in [27]. However, the implementation in [27] can only terminate the encoder from a specific phase. Padding is required when the encoder stops at a different phase. All-phase termination for LDPC convolutional code is proposed in [28]. This new approach could terminate the encoder from all possible states of any phase. Termination will induce code rate loss, in particular, this problem will be serious for short data frame length. Furthermore, the encoder complexity is also increased.

The procedure of finding termination sequence is described as follows. For more detail description, please refer to [28]. Consider a $R = 1/2$ systematic LDPC convolutional code. Let $\mathbf{v}$ be the encoded frame and $\mathbf{x}$ be the encoded termination sequence. Assume that the length of data frame is n and the termination length is L. The following equation

must be satisfied.

$$[\mathbf{v}_{1\times 2n}, \mathbf{x}_{1\times 2L}, \mathbf{0}_{1\times 2m_s}]\mathbf{H}^T_{[(0,n+L+m_s-1]} = 0_{1\times(n+L+m_s)} \tag{A.1}$$

Then the syndrome former $\mathbf{H^T}$ can be partitioned into several sub-matrices.

$$[\mathbf{v}_{1\times 2n}, \mathbf{x}_{1\times 2L}, \mathbf{0}_{1\times 2m_s}] \begin{pmatrix} \mathbf{A}^T_{2n\times n} & \mathbf{B}^T_{2n\times(L+m_s)} \\ \mathbf{0}_{2L\times n} & \mathbf{D}^T_{2L\times(L+m_s)} \\ \mathbf{0}_{2m_s\times n} & \mathbf{F}^T_{2m_s\times(L+m_s)} \end{pmatrix} = 0_{1\times(n+L+m_s)} \tag{A.2}$$

From above equation, we can obtain $\mathbf{v}_{1\times 2n}\mathbf{A}^T_{2n\times n} = \mathbf{0}_{1\times n}$ and

$$\mathbf{x}_{1\times 2L}\mathbf{D}^T_{2L\times(L+m_s)} = \mathbf{v}_{1\times 2n}\mathbf{B}^T_{2n\times(L+m_s)}. \tag{A.3}$$

Since only the last $2m_s$ bits of $\mathbf{v}$ are participated in the computation, we can let $\mathbf{vB^T} = \mathbf{sP^T}$, where $\mathbf{s}$ represents the ending state vector of the encoder and $\mathbf{P^T}$ comprises the last m_s row of $\mathbf{B^T}$. The sub-matrices $\mathbf{D^T}$ and $\mathbf{P^T}$ relate to the starting phase ϕ of the termination sequence and the termination length L. After taking the tranpose, the above equation can be rewritten as

$$\mathbf{D}_{\phi,L}\mathbf{x}^T = \mathbf{P}_{\phi,L}\mathbf{s}^T. \tag{A.4}$$

Therefore, the termination sequence can be derived by solving the linear equations. As long as $\mathbf{D}_{\phi,L}$ is full rank, this equation will have one or more solutions. The matrix $\mathbf{F}_{\phi,L} = \mathbf{D}^{-1}_{\phi,L}\mathbf{P}_{\phi,L}$ is defined as the termination matrix.

$$\mathbf{x}^T = \mathbf{D}^{-1}_{\phi,L}\mathbf{P}_{\phi,L}\mathbf{s}^T = \mathbf{F}_{\phi,L}\mathbf{s}^T \tag{A.5}$$

We summarize the procedure of generating termination connections of the encoder [28].

1. **Determine the Termination Length**

   (a) Initialize L: $L = m_s$.

   (b) Obtain $D_{0,L}$, $D_{1,L}$, ... ,$D_{T_s-1,L}$ from H^T. Check the rank of these matrices. If all of them have rank $L + m_s$ (full rank), stop the search and choose the current value of L. Otherwise, go to (c).

   (c) Increase L by 1. Then, go to (b).

2. **Find Termination Matrices for All the Phases**

(a) Find the inverse of $D_{\phi,L}$ using Gaussian-Jordan elimination.

(b) Obtain $P_{\phi,L}$ for all phases from H^T.

(c) Get the termination matrix by multiplication. $F_{\phi,L} = D_{\phi,L}^{-1} P_{\phi,L}$.

3. Generate the Termination Connections

(a) The first row of each termination matrix is taken and combined to form the all-phase termination matrix $\mathcal{F}$.

Once the all-phase termination matrix $\mathcal{F}$ is determined, it needs to be stored and used in the implementation. The matrix $\mathcal{F}$ determines the connections of the termination circuit. Using this approach, the termination bits can be generated on demand with termination circuit embedded in the encoder. When all the termination bits have been sent to the encoder, the encoder will reach a partial-zero state. Then $b \cdot m_s$ zeros are needed to flush the encoder to the all-zero state.

We simulated the time-varying $(491, 3, 6)$ LDPC convolutional code with termination length from 491 to 2608. The results in Table. A.1 show that the termination sequence cannot be found because the matrices $\mathbf{D}_{\phi,L}$ are not full rank for all phases. For the cases we simulated, we observed that the difference between the rank of matrix $\mathbf{D}_{0,L}$ and the full rank condition $(L + m_s)$ is always 33. Therefore, we conclude that the $(491, 3, 6)$ LDPC convolutional code cannot be terminated.

Table A.1: Simulation results of the (491,3,6) LDPC convolutional code using all-phase termination.

Termination Length (L)	Rank of $\mathbf{D}_{0,L}$	Full rank condition $(L + m_s)$
491	949	982
492	950	983
493	951	984
⋮	⋮	⋮
2606	3064	3097
2607	3065	3098
2608	3066	3099

Appendix B

Tail-Biting LDPC Convolutional Codes

Tail-biting could convert a convolutional code into a block code. Using tail-biting, the starting state of the encoder is forced to be the same state as its ending state. The beginning state of the encoder does not need to start in the all-zero state, it is determined from the information sequence. Compared to the encoder termination, tail-biting avoids code-rate loss due to tail bits are not needed. The tail-biting version of the LDPC convolutional code is proposed in [29], which is obtained by wrapping the last $(c-b)\cdot m_s$ columns of the syndrome former after $t = N$ time instant. The wrapped syndrome former $\tilde{H}^T_{[0,N-1]}$ is shown in (B.1). This operation results in a circular Tanner graph. The encoding of tail-biting LDPC convolutional code can be achieved using a matrix multiplication circuitry, which is similar to the encoding of the LDPC block codes. However, the complexity of this encoder implementation is dependent of N^2.

$$\tilde{H}^T_{[0,N-1]} = \begin{pmatrix} H_0^T(0) & H_1^T(1) & \dots & H_{m_s}^T(m_s) & 0 & \dots & & 0 \\ 0 & H_0^T(1) & \dots & H_{m_s-1}^T(m_s) & H_{m_s}^T(m_s+1) & 0 \dots & & 0 \\ & \ddots & & & \ddots & & \ddots & \\ H_{m_s}^T(N) & 0 & & \dots & & 0\ H_0^T(N-m_s) & \dots & H_{m_s-1}^T(N-1) \\ H_{m_s-1}^T(N) & H_{m_s}^T(N+1) & 0 & & & & & \vdots \\ \vdots & & & & & 0 & H_0^T(N-2) & H_1^T(N-1) \\ H_1^T(N) & H_2^T(N+1) & \dots & H_{m_s}^T(N+m_s-1) & 0 & \dots & 0 & H_0^T(N-1) \end{pmatrix} \tag{B.1}$$

Another implementation for encoding the tail-biting LDPC convolutional code is proposed in [30]. The state-space variables are used to calculate the initial state of the encoder.

This technique originally comes from [31], which presented the encoding of tail-biting codes with feedback encoder. Let $\mathbf{S}_t$ denote the state vector at time instant t. Let $\mathbf{u_t}$ be the information sequence and $\mathbf{v_t}$ be the code bits. To simplify the situation, here we consider the case of systematic encoding, where the submatrices $\mathbf{H}_0(t) = (\tilde{\mathbf{H}}_0(t), \mathbf{I}_{(c-b)})$. The correct initial state can be calculated from the following relationship

$$\mathbf{S}_{t+1} = \mathbf{A}(t) \cdot \mathbf{S}_t + \mathbf{B}(t) \cdot \mathbf{u}_t^T \tag{B.2a}$$

$$\mathbf{v}_t^T = \mathbf{C}(t) \cdot \mathbf{S}_t + \mathbf{D}(t) \cdot \mathbf{u}_t^T \tag{B.2b}$$

where $\mathbf{A}$(t) is the state matrix with size of $(m_s + b) \times (m_s + b)$, $\mathbf{B}$(t) denotes the input matrix with size of $(m_s c + b) \times b$, $\mathbf{C}$(t)is the output matrix with size of $c \times (m_s c + b)$, and $\mathbf{D}$(t) denotes the feedforward matrix with size of $c \times b$. Note that the variables in the above equations are functions of time due to time-varying are considered. These matrices $\mathbf{A}$(t), $\mathbf{B}$(t), $\mathbf{C}$(t) and $\mathbf{D}$(t) can be determined from the state transitions of the syndrome former encoder.

$$\mathbf{A}(t) = \begin{pmatrix} \mathbf{H}_1(t) & \mathbf{H}_2(t) & \mathbf{H}_3(t) & \dots & \mathbf{H}_{m_s-2}(t) & \mathbf{H}_{m_s-1}(t) & \mathbf{H}_{m_s}(t) & 0_{(c-b)\times b} \\ 0_{b\times c} & 0_{b\times c} & 0_{b\times c} & \dots & 0_{b\times c} & 0_{b\times c} & 0_{b\times c} & 0_{b\times b} \\ \mathbf{I}_c & 0_{c\times c} & 0_{c\times c} & \dots & 0_{c\times c} & 0_{c\times c} & 0_{c\times c} & 0_{(c-b)\times b} \\ 0_{c\times c} & \mathbf{I}_c & 0_{c\times c} & \dots & 0_{c\times c} & 0_{c\times c} & 0_{c\times c} & 0_{(c-b)\times b} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0_{c\times c} & 0_{c\times c} & 0_{c\times c} & \dots & 0_{c\times c} & \mathbf{I}_c & 0_{c\times c} & 0_{(c-b)\times b} \\ 0_{b\times c} & 0_{b\times c} & 0_{b\times c} & \dots & 0_{b\times c} & 0_{b\times c} & (\mathbf{I}_b, 0_{b\times c(c-b)}) & 0_{b\times b} \end{pmatrix} \tag{B.3}$$

$$\mathbf{B}(t) = \begin{pmatrix} \tilde{\mathbf{H}}_0(\mathbf{t}) \\ \mathbf{I}_b \\ 0_{((m_s-1)c+b)\times b} \end{pmatrix} \tag{B.4}$$

$$\mathbf{C}(t) = \begin{pmatrix} 0_{b\times c} & 0_{b\times c} & \dots & 0_{b\times c} & 0_{b\times c} & 0_{b\times b} \\ \mathbf{H}_1(t) & \mathbf{H}_2(t) & \dots & \mathbf{H}_{m_s-1}(t) & \mathbf{H}_{m_s}(t) & 0_{(c-b)\times b} \end{pmatrix} \tag{B.5}$$

$$\mathbf{D}(t) = \begin{pmatrix} \mathbf{I}_b \\ \tilde{\mathbf{H}}_0(\mathbf{t}) \end{pmatrix} \tag{B.6}$$

The complete solution of (B.2) is given by the superposition of the zero-input solution $\mathbf{S}_N^{[zi]}$ and the zero-state solution $\mathbf{S}_N^{[zs]}$.

$$\mathbf{S}_N = \mathbf{S}_N^{[zi]} + \mathbf{S}_N^{[zs]} \tag{B.7}$$

We can derive the zero-input solution and the zero-state solution by applying (B.2) recursively. The zero-input solution $\mathbf{S}_N^{[zi]}$ is the state achieved after t time instants if the encoding started in an arbitrary state $\mathbf{S}_0$ and all input bits are zero.

$$\mathbf{S}_N^{[zi]} = \left(\prod_{i=0}^{N-1} \mathbf{A}(N-i) \right) \cdot \mathbf{S}_0 \tag{B.8}$$

The zero-state solution $\mathbf{S}_N^{[zs]}$ denotes the state achieved after t time instants if the encoding started in the all-zero state $\mathbf{S}_0 = \mathbf{0}$ and input is the information sequence $\mathbf{u}$.

$$\mathbf{S}_N^{[zz]} = \sum_{j=0}^{N-1} \left[\left(\prod_{i=0}^{N-j-2} \mathbf{A}(N-i) \right) \cdot \mathbf{B}(j+1) \cdot \mathbf{u}_j^T \right] \tag{B.9}$$

Then we let the state at time $t = N$ is equal to the initial state $\mathbf{S}_0$, we can obtain

$$\left(\mathbf{I} + \prod_{i=0}^{N-1} \mathbf{A}(N-i) \right) \cdot \mathbf{S}_0 = \mathbf{S}_N^{[zs]}, \tag{B.10}$$

where $\mathbf{I}$ denotes the $(m_s c + b) \times (m_s c + b)$ identity matrix. Therefore, the initial state of the encoder is

$$\mathbf{S}_0 = \left(\mathbf{I} + \prod_{i=0}^{N-1} \mathbf{A}(N-i) \right)^{-1} \cdot \mathbf{S}_N^{[zs]}. \tag{B.11}$$

Given the matrix in (B.11) is invertible, the encoding procedure for tail-biting LDPC convolutional code can be summarized as the following steps [30].

1. **Determine the Zero-State Response**

   Determine the zero-state solution $\mathbf{S}_N^{[zs]}$ by encoding the information sequence $\mathbf{u}$ with the encoder starting from the zero-state $\mathbf{S}_0 = \mathbf{0}$. At this step, the sequence obtained at the output of the encoder is ignored.

2. **Calculate the Initial State**

   Calculate $\mathbf{S}_0$ using (B.11) and initialize the encoder accordingly.

3. Perform the Actual Encoding

With $\mathbf{S}_0$ correctly set, perform the actual encoding for $\mathbf{u}$. In this case, the sequence obtained at the output of the encoder is the valid code sequence $\mathbf{v}$.

Using this technique, the inverse matrices need to be stored for different block lengths. Moreover, the same encoder can be used to encode the tail-biting codes with different block lengths. In [29] and [30], a circular pipeline decoder architecture for the decoding of the tail-biting LDPC convolutional code is proposed. For more detail description, please refer to [29] and [30]. Then we applied this technique to the $(491, 3, 6)$ LDPC convolutional code. The simulated block lengths range from $N = 491$ to $N = 6401$. There are 1454 samples satisfy the tail-biting constraint, namely, the matrix in (B.11) is invertible. However, when we applied this technique to the $R = 3/6$ $(164, 3, 6)$, $R = 6/12$ $(82, 3, 6)$, $R = 9/18$ $(55, 3, 6)$, and $R = 12/24$ $(41, 3, 6)$ LDPC convolutional code shown in Figure. 4.16. Unfortunately, the tail-biting constraint cannot be satisfied for all the block lengths we simulated.

Bibliography

[1] *IEEE 802.16m System Description Document (SDD)*, IEEE Std. 802.16m-09/0034r4, 2010.

[2] *Amendment Text Proposal on Rate Compatible LDPC-Convolutional Codes*, IEEE Std. C802.16m-09/0339, 2009.

[3] R.G.Gallager, "Low-Density Parity-Check Codes," in *MA: MIT Press*, 1963.

[4] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, March 1997.

[5] A. Jimenez Felstrom and K. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 2181 –2191, Sep. 1999.

[6] Z. Chen, S. Bates, and X. Dong, "Low-density parity-check convolutional codes applied to packet based communication systems," in *Proc. IEEE Global Telecommun. Conf.*, vol. 3, Nov. 2005, pp. 1250–1254.

[7] R. Tanner, D. Sridhara, A. Sridharan, T. Fuja, and J. Costello, D.J., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966 – 2984, Dec. 2004.

[8] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, "On deriving good LDPC convolutional codes from QC LDPC block codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2007, pp. 1221 –1225.

[9] G. Richter, M. Kaupper, and R. Zigangirov, "Irregular low-density parity-check convolutional codes based on protographs," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2006, pp. 1633 –1637.

[10] Y. Murakami, S. Okamura, S. Okasaka, T. Kishigami, and M. Orihashi, "LDPC convolutional codes based on parity check polynomials with a time period of 3," *IEICE Trans. Fundamentals of Electronics Communications and Computer Sciences*, vol. E92-A, no. 10, pp. 2479–2483, Oct. 2009.

[11] A. Pusane, K. Zigangirov, and D. Costello, "Construction of irregular LDPC convolutional codes with fast encoding," in *Proc. IEEE Int. Conf. Commun.*, vol. 3, Jun. 2006, pp. 1160 –1165.

[12] J.-J. Weng, C.-C. Lai, and C.-H. Wang, "Decoding of LDPC convolutional codes with rational parity-check matrices from a new graphical perspective," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2010, pp. 789 –793.

[13] Z. Chen, R. Swamy, and S. Bates, "A new encoder implementation for low-density parity-check convolutional codes," in *Proc. IEEE Midwest Symp. Circuits Syst.*, Aug. 2007, pp. 883 –886.

[14] D. J. Costello, A. E. Pusane, S. Bates, and K. S. Zigangirov, "A comparison between LDPC block and convolutional codes," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, Feb. 2006.

[15] A. Pusane, A. Feltstrom, A. Sridharan, M. Lentmaier, K. Zigangirov, and D. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 56, no. 7, pp. 1060 –1069, Jul. 2008.

[16] T. Brandon, J. Koob, L. van den Berg, Z. Chen, A. Alimohammad, R. Swamy, J. Klaus, S. Bates, V. Gaudet, B. Cockburn, and D. Elliott, "A compact 1.1-Gb/s encoder and a memory-based 600-Mb/s decoder for LDPC convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 5, pp. 1017 –1029, May 2009.

[17] Z. Chen, T. Brandon, D. Elliott, S. Bates, W. Krzymien, and B. Cockburn, "Jointly designed architecture-aware LDPC convolutional codes and high-throughput parallel

encoders/decoders," *IEEE Trans. Circuits and Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 836 –849, Apr. 2010.

[18] A. Pusane, M. Lentmaier, K. Zigangirov, and J. Costello, D.J., "Reduced complexity decoding strategies for LDPC convolutional codes," in *Proc. IEEE Intl. Symposium on Inform. Theory*, 2004, p. 490.

[19] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. SIPS*, Oct. 2004, pp. 107 – 112.

[20] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209 – 213, Feb. 2005.

[21] E. Matus, M. Tavares, M. Bimberg, and G. Fettweis, "Towards a Gbit/s programmable decoder for LDPC convolutional codes," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2007, pp. 1657 –1660.

[22] S. Bates and R. Swamy, "Parallel encoders for low-density parity-check convolutional codes," in *Proc. IEEE Int. Symp. Circuits Syst.(ISCAS)*, May 2006.

[23] R. Swamy, S. Bates, and T. Brandon, "Architectures for ASIC implementations of low-density parity-check convolutional encoders and decoders," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, May 2005, pp. 4513 – 4516.

[24] S. Bates and G. Block, "A memory-based architecture for FPGA implementations of low-density parity-check convolutional decoders," in *Proceedings of IEEE Symposium on Circuits and Systems (ISCAS)*, May 2005.

[25] S.-Y. Hung, S.-W. Yen, C.-L. Chen, H.-C. Chang, S.-J. Jou, and C.-Y. Lee, "A 5.7Gb/s row-based layered scheduling LDPC decoder for IEEE 802.15.3c applications," in *IEEE Asian Solid-State Circuits Conference (ASSCC)*, Nov. 2010.

[26] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "A 390Mb/s 3.57mm^2 3GPP-LTE turbo decoder ASIC in 0.13μm CMOS," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2010.

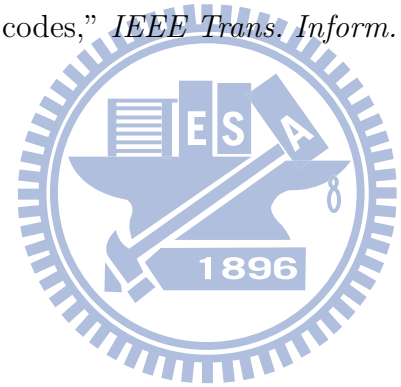
[27] S. Bates, D. Elliott, and R. Swamy, "Termination sequence generation circuits for low-density parity-check convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 9, pp. 1909 –1917, Sep. 2006.

[28] Z. Chen, T. Brandon, S. Bates, D. Elliott, and B. Cockburn, "Efficient implementation of low-density parity-check convolutional code encoders with built-in termination," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3628 –3640, Dec. 2008.

[29] M. B. Tavares, K. S. Zigangirov, and G. P. Fettweis, "Tail-biting LDPC convolutional codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2007, pp. 2341 –2345.

[30] M. B. Tavares, *On Low-Density Parity-Check Convolutional Codes: Constructions, Analysis and VLSI Implementation.* Jorg Vogt Verlag, 2010.

[31] C. Weiss, C. Bettstetter, and S. Riedel, "Code construction and decoding of parallel concatenated tail-biting codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 1, pp. 366 –386, Jan. 2001.



作 者 簡 歷

姓　　名：林玉祥

出 生 地：台灣 台北市

出生日期：1987.06.25

學歷:

1993.9 ~ 1999.6 雲林縣 麥寮鄉 橋頭國小

1999.9 ~ 2002.6 雲林縣 正心中學 國中部

2002.9 ~ 2005.6 雲林縣 正心中學 高中部

2005.9 ~ 2009.6 國立交通大學 電子工程學系 學士

2009.9 ~ 2011.8 國立交通大學 電子研究所 系統組 碩士