誌　謝

碩士班兩年的生活不算長也不算短，每天都有不同的新鮮事，其中有些帶給我歡笑，也有些讓人感覺到疲憊和崩潰，然而無論如何，走在邁向碩士學位的道路上學習到了很多新的事物，過得相當充實。

這一路上受到了很多的幫助，首先要感謝錫嘉老師在生活及研究上給與我許多的協助，因而能夠順利的完成學業。還有在學術研究上帶領我的阿龍學長以及佳龍學長，感謝有你們的指導與幫忙，讓我的研究能夠有所成果。感謝長宏學長在我們不眠不休忙著 Tape-out 時在旁細心的照料著我們。此外，感謝小肥學長時常煮東西給我們吃，讓我們了解到美味二字的意含。感謝渠學長的氣度，讓處在金字塔最底端的我還有人可以開玩笑。感謝其衡學長、欣儒學長在晶片量測時給與協助。

除了感謝老師及學長們外，我要謝謝這兩年來和我一起同甘苦、共福難的同學們：小朱哥、雞皮、奕勳、青春、皮皮、大師、浩文、哲維、浩呆、阿豪。感謝有你們的相伴，讓我不會感到孤單與寂寥，讓我在龐大的研究壓力下依然能夠展開笑靨，因為有你們我才能免於苦悶。當然，還有已畢業的學長姊：玉祥學長、士家學長、vfo、靜瑜學姊及學弟們：方舟、嚇嚇、蔡蔡、日和，與你們相處的點點滴滴都是美好的回憶。

最後，我要感謝我的家人，感謝爸爸媽媽將我養育成人，還有哥哥及大嫂及姊姊們在生活上所給與我的幫助，因為有你們才有今日的我。還有很多曾幫助過我但卻未被提及的人們，感謝你們，有你們的助力才能讓我在人生的道路上走得順遂，讓我能勇敢地踏出下一步。

中華民國　一〇一年　九月

藍祐誠

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the mobile communication systems, channel coding plays an important role to maintain the quality of transmission. And, for the good error-correcting performance and high decoding throughput purposes, low-density parity-check block codes (LDPC-BCs) attract high interests in research. Despite of the LDPC block codes have capacity-approaching performance, serious routing complexity problem is a great challenge in decoder implementations. Furthermore, the parity-check matrices of different block lengths are needed for LDPC block codes to encode and decode arbitrary lengths of data. To conquer these unfavorable difficulties, LDPC convolutional codes (LDPC-CCs) were proposed in 1999. This convolutioanl version can still perform near Shannon limit performance as well as the block code does. Compared to LDPC block codes, LDPC convolutional codes can deal with variable lengths of data frame, and the pipeline decoder architecture greatly mitigate the routing congestion problem of the VLSI implementation. Besides, LDPC convolutional codes have much simpler encoder architecture, and they also can provide flexible coding rates through puncturing easily.

Because of the advantages mentioned above, LDPC convolutional codes have great potential to meet the high-speed requirements in the next generation communication systems. However, there are still some challenges that LDPC convolutional codes need to face. The long decoding latency and high power consumption problems should be solved. In addition, for practical usage in the communication system, LDPC convolutional codes should be terminated with the termination sequence or converted to the tail-bitten version. The first technique will induce code-rate loss while padding the tail bits, and the latter one may not suitable for all lengths of data. In this thesis, we construct an LDPC convolutional code which has the tail-bitten version. And, at last, we proposed a high throughput and low power consumption tail-biting LDPC convolutional code decoder with memory-based architecture.

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, and the introduction of low-density parity-check convolutional codes is also provided. Furthermore, the architectures of encoder and decoder are also given in this chapter. In Chapter 3, a special case for constructing time-invariant tail-biting LDPC convolutional code is proposed. The memory-based decoder architecture is introduced in Chapter 4. Chapter 5 gives the implementation results. Conclusions and future works are given in Chapter 6.

Chapter 2

Low-Density Parity-Check Convolutional Codes (LDPC-CCs)

Low-density parity-check convolutional codes were first proposed in 1999 [1], 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 be encoded and decoded with variable length of data frame. It has been shown that these codes are suitable for certain applications such as video streaming and packet-switching networks [2].

In this chapter, the overview of LDPC convolutional codes is given. In the following sections, firstly, the low-density parity-check block code and iterative decoding algorithm are introduced. And then, 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.

2.1 Background

2.1.1 Low-Density Parity-Check Block Codes

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

A binary LDPC code is defined by a sparse parity-check matrix H, which contains a relatively low number of ones. For a regular (N, J, K) LDPC 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.

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

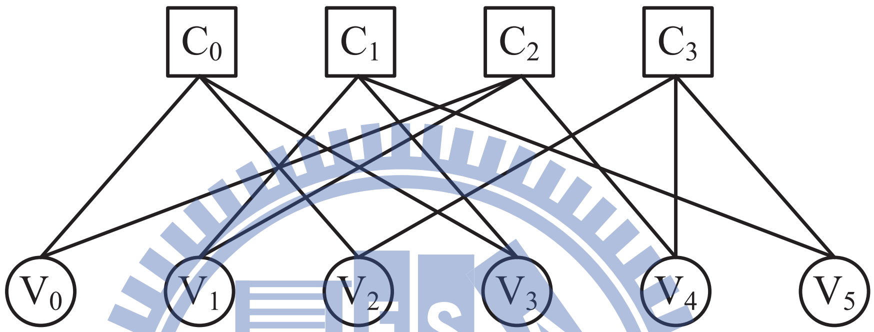


Figure 2.1: The Tanner graph of the parity check matrix given in (2.1).

Since the LDPC 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.

2.1.2 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 can 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 updates, 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 updates, 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.

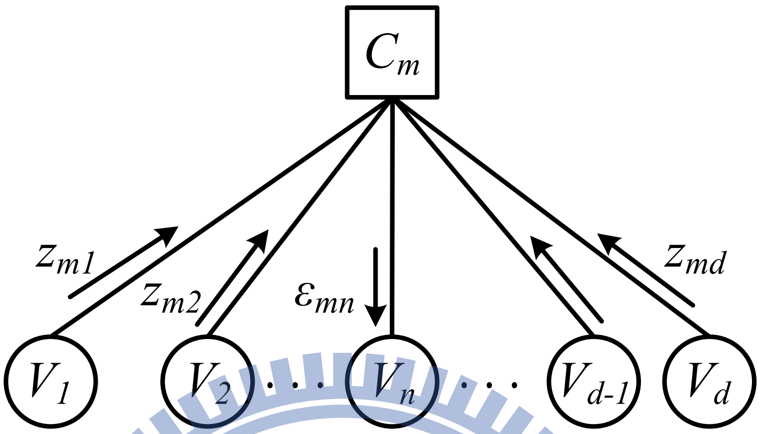


Figure 2.2: Message passing of check node.

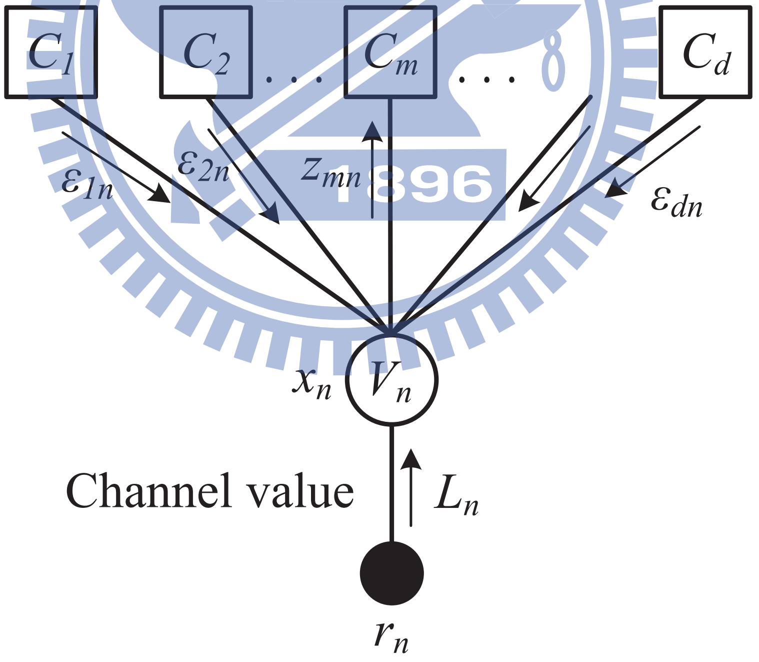


Figure 2.3: Message passing of variable node.

2.2 Encoding of LDPC Convolutional Codes

A binary LDPC convolutional code (LDPC-CC) 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

$$H^T = \begin{pmatrix} \ddots & & \ddots & & \\ H_0^T(t-m_s) & \dots & H_{m_s}^T(t) & & \\ & \ddots & \vdots & \ddots & \\ & & H_0^T(t) & \dots & H_{m_s}^T(t+m_s) \\ & & \ddots & & \ddots \end{pmatrix}. \tag{2.7}$$

The sub-matrices $H_i^T(t)$ at time instant t given in (2.8), $i = 0, 1, \dots, m_s$, 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, which is called fast encoding, and allow a register-based encoder implementation.

$$H_i(t) = \begin{pmatrix} h_i^{(1,1)}(t) & \dots & h_i^{(1,c)}(t) \\ \vdots & & \vdots \\ h_i^{(c-b,1)}(t) & \dots & h_i^{(c-b,c)}(t) \end{pmatrix} \tag{2.8}$$

The parameter m_s is called the code memory size 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 (2.9). It can be seen easily from (2.9) that the syndrome former has 3 ones in each row and 6 ones in each column.

$$H^T = \left(\begin{array}{cccccccccccc}
1 & 1 & 0 & 1 & 0 & 0 & & & & & & \\
1 & 0 & 0 & 1 & 0 & 1 & & & & & & \\
 & 1 & 1 & 1 & 0 & 0 & 0 & & & & & \\
 & 1 & 0 & 0 & 1 & 0 & 1 & & & & & \\
 & & 1 & 0 & 1 & 0 & 1 & 0 & & & & \\
 & & 1 & 0 & 0 & 1 & 0 & 1 & & & & \\
 & & & 1 & 1 & 1 & 0 & 0 & 0 & & & \\
 & & & 1 & 0 & 0 & 1 & 0 & 1 & & & \\
 & & & & 1 & 1 & 0 & 1 & 0 & 0 & & \\
 & & & & 1 & 0 & 0 & 1 & 0 & 1 & & \\
 & & & & & 1 & 1 & 1 & 0 & 0 & 0 & \\
 & & & & & 1 & 0 & 0 & 1 & 0 & 1 & \\
 & & & & & & \ddots & & & & & \ddots
\end{array} \right). \tag{2.9}$$

2.2.1 Syndrome Former Encoder

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

$$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1}) \tag{2.10}$$

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

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{t-1}), \tag{2.11}$$

where $\mathbf{v}_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(c)})$. The coded sequence $\mathbf{v}$ satisfies the constraint $\mathbf{v}\mathbf{H}^{\mathbf{T}} = \mathbf{0}$. This equation could be further decomposed into equation (2.12) 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 [3].

$$\mathbf{v}_t\mathbf{H}_0^T(t) + \mathbf{v}_{t-1}\mathbf{H}_1^T(t) + \ldots + \mathbf{v}_{t-m_s}\mathbf{H}_{m_s}^T(t) = 0 \tag{2.12}$$

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.

$$\begin{aligned} v_t^{(j)} &= u_t^{(j)}, & for \quad j = 1, \ldots, b \\ v_t^{(j)} &= \sum_{k=1}^{b} v_t^{(k)} h_0^{(j-b,k)}(t) + \sum_{i=1}^{m_s}\sum_{k=1}^{c} v_{t-i}^{(k)} h_i^{(j-b,k)}(t), & for \quad j = b+1, \ldots, c \end{aligned} \tag{2.13}$$

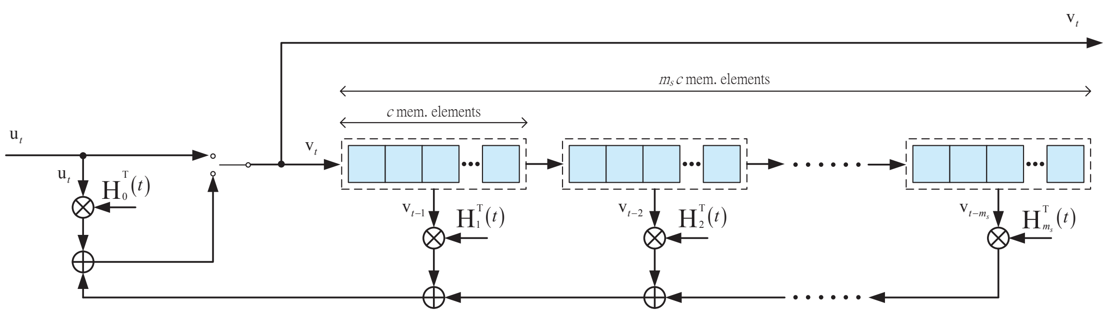


Figure 2.4: Generic syndrome former encoder for systematic LDPC convolutional codes.

2.2.2 Partial Syndrome Encoder

Partial syndrome encoder is another solution for linear-time encoding circuit for LDPC convolutional code. Considering the equation

$$\mathbf{v}_{[0,t]}\mathbf{H}^T_{[0,t]} = \left(\mathbf{0}_{[0,t]}, \mathbf{p}_{[0,t]}\right) \tag{2.14}$$

is true for all the code sequence $\mathbf{v}_{[0,t]}$ from time instant 0 to t. And $\mathbf{H}^T_{[0,t]}$ is the syndrome former matrix from time instant 0 to t, $\mathbf{0}_{[0,t]}$ is a zero sequence of length $(c-b)(t+1)$, and the partial syndrome $\mathbf{p}_t$ is given by

$$\mathbf{p}_t = \left(\mathbf{p}_{t,0}, \mathbf{p}_{t,1}, \ldots, \mathbf{p}_{t,m_s-1}\right). \tag{2.15}$$

The elements of the partial syndrome $\mathbf{p}_t$ can be obtained from the recursive relationship

$$\mathbf{p}_{t,i} = \begin{cases} \mathbf{p}_{t-1,i+1} + \mathbf{v}_t\mathbf{H}^T_{i+1}(t+i+1), & 0 \leq i < m_s - 1 \\ \mathbf{v}_t\mathbf{H}^T_{m_s}(t+m_s), & i = m_s - 1 \end{cases} \tag{2.16}$$

However, similar to the syndrome former encoder, a systematic partial syndrome encoder can be obtained if the last $(c-b)$ rows of $\mathbf{H}^T_0(t)$ is an identity matrix, and then, the encoding equation can be summarized as follows.

$$\begin{aligned} v_t^{(j)} &= u_t^{(j)}, & for \quad j = 1, \ldots, b \\ v_t^{(j)} &= \sum_{k=1}^{b} v_t^{(k)} h_0^{(j-b,k)}(t) + p_{t-1,0}^{(j-b)}, & for \quad j = b+1, \ldots, c \end{aligned} \tag{2.17}$$

with the initial condition that $p_{t-1,1}^{(j-b)} = 0$ for $j = b+1, \ldots, c$ while $t = 0$.

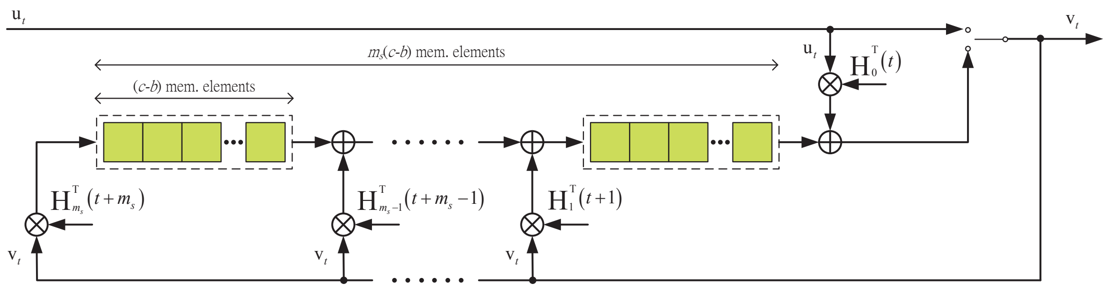


Figure 2.5: Generic partial syndrome encoder for systematic LDPC convolutional codes.

2.3 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 code is semi-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 time interval of the same variable node doing the update is at least $(m_s + 1)$ time instants apart, the decoding 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 processors corresponds to the number of iterations of iterative decoding. Hence, the decoding performance gets better when more processors are used. Studies have shown that LDPC convolutional code could achieve the same near-Shannon-limit error performance as the LDPC block code does. 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 [4]. 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 [5].

Here, we give a $R = 1/2$ time-invariant $(14, 3, 6)$ LDPC convolutional code in (2.18) as an example to illustrate the operation of the pipeline decoder on the Tanner graph. Figure 2.6 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 2.6 is (m_s+2) instead of $(m_s + 1)$.

$$(1 + D^5 + D^{11})u(D) + (1 + D^7 + D^{14})v(D) = 0 \tag{2.18}$$

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 value (∞) 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.

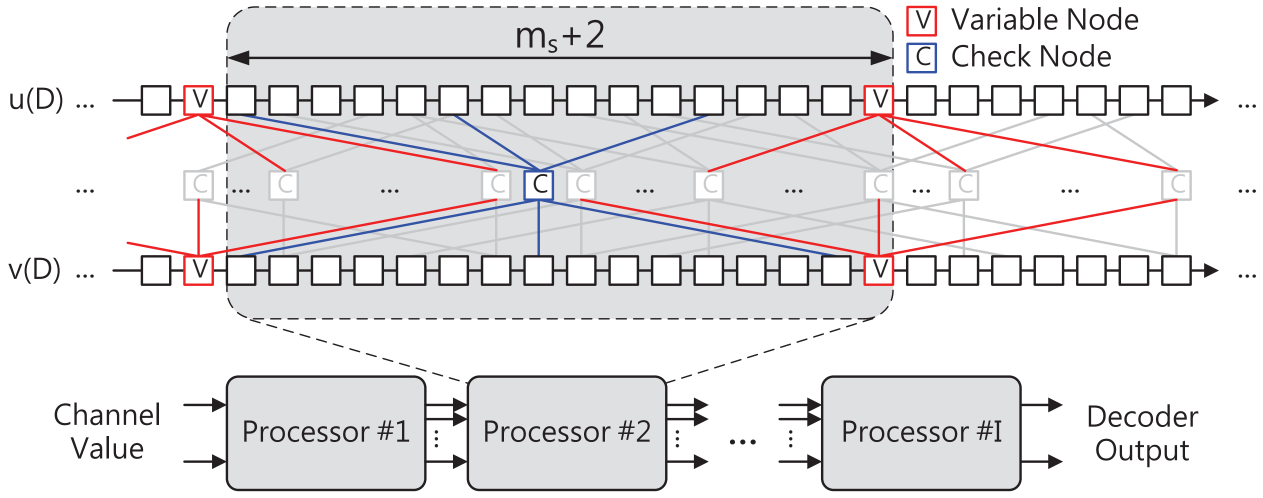


Figure 2.6: Trellis diagram and the illustration of pipeline decoding.

Chapter 3

Code Construction for Time-Invariant Tail-Biting LDPC-CCs

In this chapter, we proposed a method for constructing time-invariant LDPC convolutional code which could be tail-bitten when the encoding time $t = 2^n \geq m_s$ for n is a positive integer. At the beginning, we will introduce how the tail-biting technique works. Then, some special properties of the time-invariant LDPC convolutional code are discussed from the mathematical point of view. At last, we constructed a tail-biting LDPC convolutional code with the properties mentioned above.

3.1 Tail-Biting Technique for LDPC-CCs

Tail-biting could convert a convolutional code into a quasi-cyclic (QC) block code. The circular decoder architecture was proposed in [6]. Using tail-biting technique, 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 the tail bits are omitted.

The tail-bitten version parity-check matrix of the LDPC convolutional code is proposed in [6], which is obtained by wrapping the last $(c-b)\cdot m_s$ columns of the syndrome former after $t = t_N$ time instant. The wrapped syndrome former $\tilde{H}^T_{[0,t_N-1]}$ is shown in (3.1). This operation results in a circular Tanner graph. The encoding of tail-biting LDPC convolutional code (TB-LDPC-CC) can be achieved using a matrix multiplication circuitry, which is similar to the encoding of the LDPC block code.

$$\tilde{H}^T_{[0,t_N-1]} = \begin{pmatrix} H_0^T(0) & H_1^T(1) & \dots & H_{m_s}^T(m_s) & 0 & \dots & & 0 \\ 0 & H_0^T(1) & \dots & H_{m_s-1}^T(m_s) & H_{m_s}^T(m_s{+}1)\ 0 & \dots & & 0 \\ & \ddots & & & \ddots & & \ddots & \\ H_{m_s}^T(t_N) & 0 & & \dots & & 0\ H_0^T(t_N-m_s) & \dots & H_{m_s-1}^T(t_N-1) \\ H_{m_s-1}^T(t_N) & H_{m_s}^T(t_N{+}1)\ 0 & & & & & & \vdots \\ \vdots & & & & & 0 & H_0^T(t_N-2) & H_1^T(t_N-1) \\ H_1^T(t_N) & H_2^T(t_N{+}1) & \dots & H_{m_s}^T(t_N{+}m_s{-}1) & 0 & \dots & 0 & H_0^T(t_N-1) \end{pmatrix} \tag{3.1}$$

Another implementation for encoding the tail-biting LDPC convolutional code is proposed in [7]. The state-space variables are used to calculate the initial state of the encoder. This technique originally comes from [8], 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 $\mathbf{H}_0(t) = (\tilde{\mathbf{H}}_0(t), \mathbf{I}_{(c-b)})$. Moreover, considering a partial syndrome encoder, the correct initial state can be calculated from the following relationship

$$\mathbf{S}_{t+1} = \mathbf{A}(t)\cdot\mathbf{S}_t + \mathbf{B}(t)\cdot\mathbf{u}_t^T \tag{3.2a}$$

$$\mathbf{v}_t^T = \mathbf{C}(t)\cdot\mathbf{S}_t + \mathbf{D}(t)\cdot\mathbf{u}_t^T \tag{3.2b}$$

where $\mathbf{A}$(t) is the state matrix with size of $m_s(c-b)\times m_s(c-b)$, $\mathbf{B}$(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 feed-forward 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 **A**(t), **B**(t), **C**(t) and **D**(t) can be determined from the state transitions of the partial syndrome encoder. In ordered to proceed, we define the following decomposition

$$\mathbf{H}_i(t) = \left(\mathbf{H}_i^L(t), \mathbf{H}_i^R(t)\right), fori = 1, \ldots, m_s, \tag{3.3}$$

where $\mathbf{H}_i^L(t)$ contains the first b columns on the left hand side of $\mathbf{H}_i(t)$ and $\mathbf{H}_i^R(t)$ contains the remaining $(c-b)$ columns on the right hand side. Consequently, we can obtain the following equations.

$$\mathbf{A}(t) = \begin{pmatrix} \mathbf{0}_{(c-b)\times(c-b)} & \cdots & \cdots & \cdots & \mathbf{0}_{(c-b)\times(c-b)} & \mathbf{H}_{m_s}^R(t+m_s) \\ \mathbf{I}_{(c-b)} & \mathbf{0}_{(c-b)\times(c-b)} & \cdots & \cdots & \mathbf{0}_{(c-b)\times(c-b)} & \mathbf{H}_{m_s-1}^R(t+m_s-1) \\ \mathbf{0}_{(c-b)\times(c-b)} & \mathbf{I}_{(c-b)} & \mathbf{0}_{(c-b)\times(c-b)} & \cdots & \mathbf{0}_{(c-b)\times(c-b)} & \mathbf{H}_{m_s-2}^R(t+m_s-2) \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ \mathbf{0}_{(c-b)\times(c-b)} & \cdots & \cdots & \mathbf{0}_{(c-b)\times(c-b)} & \mathbf{I}_{(c-b)} & \mathbf{H}_1^R(t+1) \end{pmatrix} \tag{3.4}$$

$$\mathbf{B}(t) = \begin{pmatrix} \mathbf{H}_{m_s}^L(t+m_s) + \mathbf{H}_{m_s}^R(t+m_s)\cdot\tilde{\mathbf{H}}_0(t) \\ \mathbf{H}_{m_s-1}^L(t+m_s-1) + \mathbf{H}_{m_s-1}^R(t+m_s-1)\cdot\tilde{\mathbf{H}}_0(t) \\ \mathbf{H}_{m_s-2}^L(t+m_s-2) + \mathbf{H}_{m_s-2}^R(t+m_s-2)\cdot\tilde{\mathbf{H}}_0(t) \\ \vdots \\ \mathbf{H}_1^L(t+1) + \mathbf{H}_1^R(t+1)\cdot\tilde{\mathbf{H}}_0(t) \end{pmatrix} \tag{3.5}$$

$$\mathbf{C}(t) = \begin{pmatrix} \mathbf{0}_{b\times(c-b)} & \mathbf{0}_{b\times(c-b)} & \cdots & \mathbf{0}_{b\times(c-b)} \\ \mathbf{I}_{(c-b)} & \mathbf{0}_{(c-b)\times(c-b)} & \cdots & \mathbf{0}_{(c-b)\times(c-b)} \end{pmatrix} \tag{3.6}$$

$$\mathbf{D}(t) = \begin{pmatrix} \mathbf{I}_b \\ \tilde{\mathbf{H}}_\mathbf{0}(\mathbf{t}) \end{pmatrix} \tag{3.7}$$

The complete solution of (3.2) is given by the superposition of the zero-input solution $\mathbf{S}_{t_N}^{[zi]}$ and the zero-state solution $\mathbf{S}_{t_N}^{[zs]}$.

$$\mathbf{S}_{t_N} = \mathbf{S}_{t_N}^{[zi]} + \mathbf{S}_{t_N}^{[zs]} \tag{3.8}$$

We can derive the zero-input solution and the zero-state solution by applying (3.2) recursively. The zero-input solution $\mathbf{S}_{t_N}^{[zi]}$ is the state achieved after $t = t_N$ time instants if the encoding started in an arbitrary state $\mathbf{S}_0$ and all input bits are zero.

$$\mathbf{S}_{t_N}^{[zi]} = \left(\prod_{i=0}^{t_N-1} \mathbf{A}(t_N - i) \right) \cdot \mathbf{S}_0 \tag{3.9}$$

The zero-state solution $\mathbf{S}_{t_N}^{[zs]}$ 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 $\mathbf{u}$.

$$\mathbf{S}_{t_N}^{[zs]} = \sum_{j=0}^{t_N-1} \left[\left(\prod_{i=0}^{t_N-j-2} \mathbf{A}(t_N - i) \right) \cdot \mathbf{B}(j+1) \cdot \mathbf{u}_j^T \right] \tag{3.10}$$

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

$$\left(\mathbf{I} + \prod_{i=0}^{t_N-1} \mathbf{A}(t_N - i) \right) \cdot \mathbf{S}_0 = \mathbf{S}_{t_N}^{[zs]}, \tag{3.11}$$

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

$$\mathbf{S}_0 = \left(\mathbf{I} + \prod_{i=0}^{t_N-1} \mathbf{A}(t_N - i) \right)^{-1} \cdot \mathbf{S}_{t_N}^{[zs]}. \tag{3.12}$$

Given the matrix in (3.12) is invertible, the encoding procedure for tail-biting LDPC convolutional code can be summarized as the following steps [7].

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

   Determine the zero-state solution $\mathbf{S}_{t_N}^{[zs]}$ 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 (3.12) 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 $\mathbf{v}$.

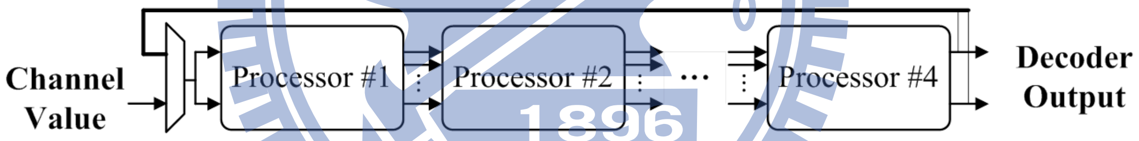


Figure 3.1: The illustration of a tail-biting LDPC convolutional code decoder.

3.2 Mathematical Properties of Time-Invariant LDPC-CCs

From equation (3.12), an LDPC convolutional code could have the tail-bitten version if and only if the matrix $\left(\mathbf{I} + \prod_{i=0}^{t_N-1} \mathbf{A}(t_N - i)\right)$ is invertible. However, whether the matrix is invertible or not depends on the length of the encoding time duration. As a result, we should try the encoding time duration one-by-one to test the constraint is met or not. It is very inconvenient and may take a lot of time to do the test. For the reasons, we discovered a mathematical property of the time-invariant LDPC convolutional code which can help us to find suitable encoding time duration to meet the tail-biting constraint. In fact, we focus on the time-invariant code because the state transition matrix is simpler. And fortunately, the time-variant LDPC convolutional code can be transformed into a time-invariant one by the folding technique proposed in our previous work [9]. Anyway, once a time-invariant LDPC convolutional code is constructed, no matter it is directly constructed or transformed from a time-variant code through the folding technique, it is surely have the same properties below. The derivation is as follows.

First of all, for a time-invariant LDPC convolutional code, the state matrix is irrelevant to time, in other words, the state matrix is no longer a function of time. Consequently, the equation (3.12) can be rewritten to

$$\mathrm{S}_0 = \left(\mathbf{I} + \mathbf{A}^{t_N}\right)^{-1} \cdot \mathbf{S}_{t_N}^{[zs]}. \tag{3.13}$$

Seeing that the state matrix $\mathbf{A}$ is binary, it is apparently that we can absolutely find a matrix $\mathbf{Z}$ such that

$$\begin{aligned} \mathbf{A} &= \mathbf{I} + \mathbf{Z} \\ \text{or} \quad \mathbf{Z} &= \mathbf{I} + \mathbf{A} \end{aligned} \tag{3.14}$$

Therefore, the matrix

$$\begin{aligned} \mathbf{I} + \mathbf{A}^{t_N} &= \mathbf{I} + (\mathbf{I} + \mathbf{Z})^{t_N} \\ &= \mathbf{I} + \mathbf{I} + \sum_{i=1}^{t_N} \alpha_i \mathbf{Z}^i \\ &= \sum_{i=1}^{t_N} \alpha_i \mathbf{Z}^i \\ &= \sum_{i=1}^{t_N} \alpha_i \left(\mathbf{I} + \mathbf{A}\right)^i \end{aligned} \tag{3.15}$$

where $\alpha_i \in \{0, 1\}$. It is obvious that matrix $\mathbf{I} + \mathbf{A}$ is a factor of matrix $\mathbf{I} + \mathbf{A}^{t_N}$, which means that the matrix $\mathbf{I} + \mathbf{A}^{t_N}$ is invertible only if $\mathbf{I} + \mathbf{A}$ is invertible. From this relationship, we can infer that a time-invariant LDPC convolutional code with the state matrix $\mathbf{A}$ cannot have the tail-bitten version with any length of encoding time duration if $\mathbf{I}+\mathbf{A}$ is not invertible. However, we cannot know if this code could be tail-bitten for any time duration or not. Fortunately, if $\mathbf{I}+\mathbf{A}$ is invertible, then the time-invariant LDPC convolutional code can surely be tail-bitten when the encoding time duration $t_N = 2^n$. In this case, the matrix

$$\begin{aligned} \mathbf{I}+\mathbf{A}^{t_N} &= \mathbf{I}+(\mathbf{I}+\mathbf{Z})^{2^n} \\ &= \mathbf{I}+\mathbf{I}+\mathbf{Z}^{2^n} \\ &= (\mathbf{I}+\mathbf{A})^{2^n}. \end{aligned} \tag{3.16}$$

The equation above can be easily derived from the mathematical induction. Seeing that $\mathbf{I}+\mathbf{A}$ is invertible, the matrix $\mathbf{I}+\mathbf{A}^{2^n}$, or $(\mathbf{I}+\mathbf{A})^{2^n}$, is definitely invertible. Hence, this time-invariant LDPC convolutional code can have the tail-bitten version when the encoding time duration is equal to 2^n.

3.3 Time-Invariant TB-LDPC-CCs Construction

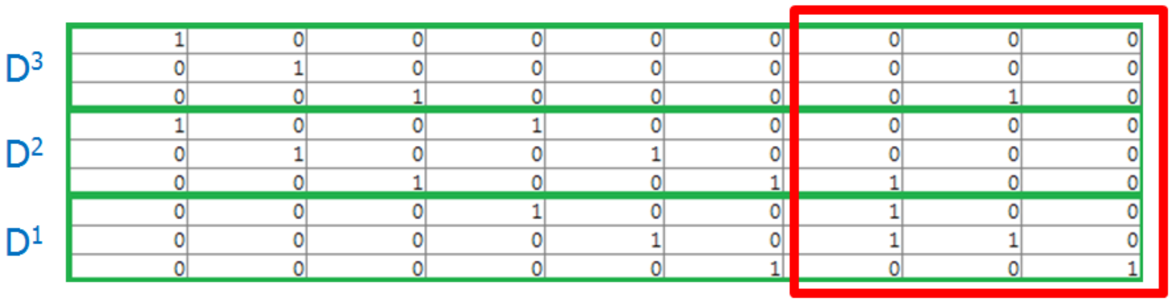
From the last section, we know that if a time-invariant LDPC convolutional code could have the tail-bitten version, the matrix $\mathbf{I}+\mathbf{A}$ should be invertible. As a result, we want to find a code construction method which insure that code constructed with the method have the tail-bitten version. In the past, most LDPC convolutional codes are derived from the parity-check matrices of LDPC block codes. In [1], 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 [10] and [11], the algebraically structured quasi-cyclic LDPC (QC-LDPC) block codes were also applied to derive both time-invariant and time-varying LDPC convolutional codes. Besides, a construction method to design LDPC convolutional codes based on protographs were proposed in [12]. In [13], 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. Here, we use the polynomial parity-check matrix $\mathbf{H}(D)$ to construct the LDPC convolutional code.

In our work, we choose the partial syndrome encoder as our encoder type. For the fast encoding purpose, we restrict the diagonal entry of the parity part to be 1 in the polynomial party-check matrix. Moreover, we set the parity part in $\mathbf{H}(D)$ to be lower triangular; therefore, the tail-biting examination matrix $\mathbf{I} + \mathbf{A}$ will be invertible. In other words, the time-invariant LDPC convolutional code, which was constructed with this method, can have the tail-bitten version when the encoding time duration $t_N = 2^n$. Now, we give an example to see how it happened.

$$H(D) = \left(\begin{array}{ccc|ccc} D^{\alpha_{11}} & D^{\alpha_{12}} & D^{\alpha_{13}} & 1 & 0 & 0 \\ D^{\alpha_{21}} & D^{\alpha_{22}} & D^{\alpha_{23}} & D^1 & 1 & 0 \\ D^{\alpha_{31}} & D^{\alpha_{32}} & D^{\alpha_{33}} & D^2 & D^3 & 1 \end{array} \right) \tag{3.17}$$

In the equation (3.17), the left part is the information part, and the right part is the parity part. Because the state matrix $\mathbf{A}$ is irrelevant to the information part in (3.4), we only focus on the parity part in this example. The matrix $\mathbf{I} + \mathbf{A}$ is as follow. From the matrix above, we can discover that it is always invertible because of the last column of $degree - one$.

And, however, it is difficult to fill all the elements in the polynomial parity-check matrix $\mathbf{H}(D)$ to perform good error-correcting performance with low hardware complexity. Inspired from [14], we construct a smaller $\mathbf{H}(D)$ first. And then, we take it as the base matrix and span it with the quasi-cyclic (QC) transformation to get a larger parity-check matrix. For example, we now have a polynomial parity-check matrix as follows.



D^3	1	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	1	0
D^2	1	0	0	1	0	0	0	0	0
	0	1	0	0	1	0	0	0	0
	0	0	1	0	0	1	1	0	0
D^1	0	0	0	1	0	0	1	0	0
	0	0	0	0	1	0	1	1	0
	0	0	0	0	0	1	0	0	1

Figure 3.2: The tail-biting examination matrix in the example above.

$$\mathbf{H}(D) = \begin{pmatrix} 1 & D^{11} & D^4 & D^{16} & 1 & 0 & 0 & 0 \\ D^5 & 1 & D^2 & D^{18} & D^2 & 1 & 0 & 0 \\ D^7 & D^9 & 1 & D^7 & D^{15} & D^{21} & 1 & 0 \\ D^{18} & D^{16} & D^8 & 1 & D^4 & D^8 & D^5 & 1 \end{pmatrix} \tag{3.18}$$

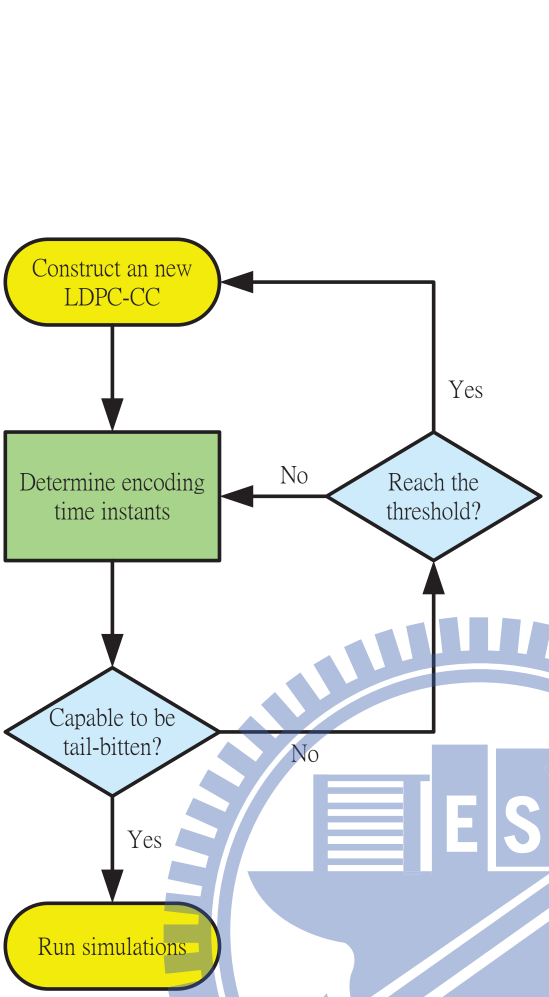
After the QC transformation, the polynomial parity-check matrix is

$$\mathbf{H}(D) = \begin{pmatrix} \mathbf{I}_{6\times6} & D^{11}\cdot\mathbf{P}^4 & D^4\cdot\mathbf{P}^2 & D^{16}\cdot\mathbf{P}^1 & \mathbf{I}_{6\times6} & \mathbf{0}_{6\times6} & \mathbf{0}_{6\times6} & \mathbf{0}_{6\times6} \\ D^5\cdot\mathbf{P}^2 & \mathbf{I}_{6\times6} & D^2\cdot\mathbf{P}^5 & D^{18}\cdot\mathbf{P}^4 & D^2\cdot\mathbf{P}^2 & \mathbf{I}_{6\times6} & \mathbf{0}_{6\times6} & \mathbf{0}_{6\times6} \\ D^7\cdot\mathbf{P}^3 & D^9\cdot\mathbf{P}^1 & \mathbf{I}_{6\times6} & D^7\cdot\mathbf{P}^3 & D^{15}\cdot\mathbf{P}^4 & D^{21}\cdot\mathbf{P}^5 & \mathbf{I}_{6\times6} & \mathbf{0}_{6\times6} \\ D^{18}\cdot\mathbf{P}^5 & D^{16}\cdot\mathbf{P}^3 & D^8\cdot\mathbf{P}^3 & \mathbf{I}_{6\times6} & D^4\cdot\mathbf{P}^2 & D^8\cdot\mathbf{P}^2 & D^5\cdot\mathbf{P}^4 & \mathbf{I}_{6\times6} \end{pmatrix} \tag{3.19}$$

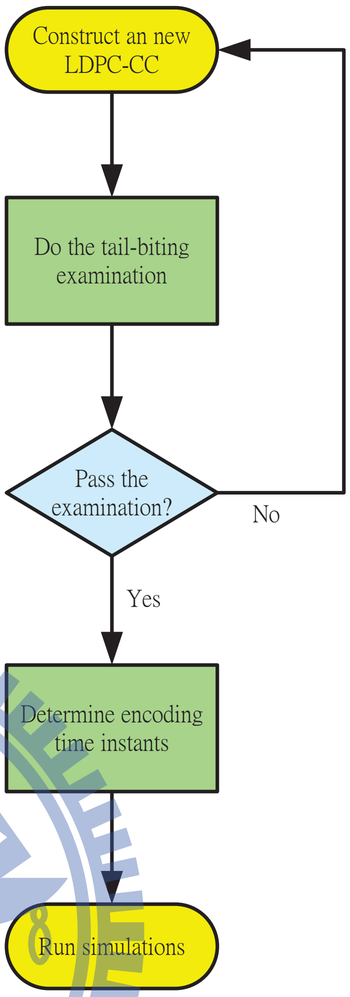
where

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.20}$$

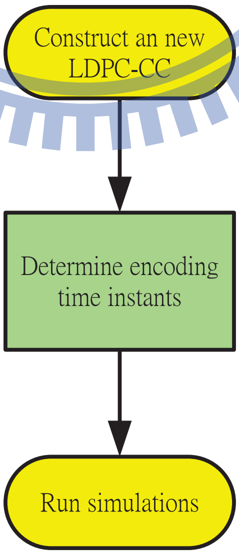
is the left-shift permutation matrix. However, while applying the tail-biting scheme, the performance will be degraded with the dependence on the length of encoding time duration. The shorter encoding time, the poor error-correcting performance. In addition, from [15], we forced the each elements in the polynomial parity-check matrix to be monomial in order to reduce the number of short cycles. Figure 3.3 shows the difference between our construction method and the traditional one.



(a) Conventional TB-LDPC-CC construction.



(b) TB-LDPC-CC construction with the proposed examination.



(c) The proposed TB-LDPC-CC construction.

Figure 3.3: The TB-LDPC-CC construction procedure.

Chapter 4

Proposed TB-LDPC-CC Decoder Architecture

In this chapter, we are going to introduce the decoder architecture in our work. At the first, we modify the OVA scheduling to lower the storage capacity and improve the error-correcting performance compared to the traditional flooding scheduling. Besides, we adopt the memory-based architecture in our design in order to lower the power consumption. However, for the throughput purpose, we also split the decoder into a 4-stage pipeline architecture.

4.1 Scheduling 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 processor architecture in a conventional decoder for the $(14, 3, 6)$ LDPC convolutional code given in (2.18). We can see that the variable nodes are performed until the messages are all transferred into the check-to-variable messages in the processor.

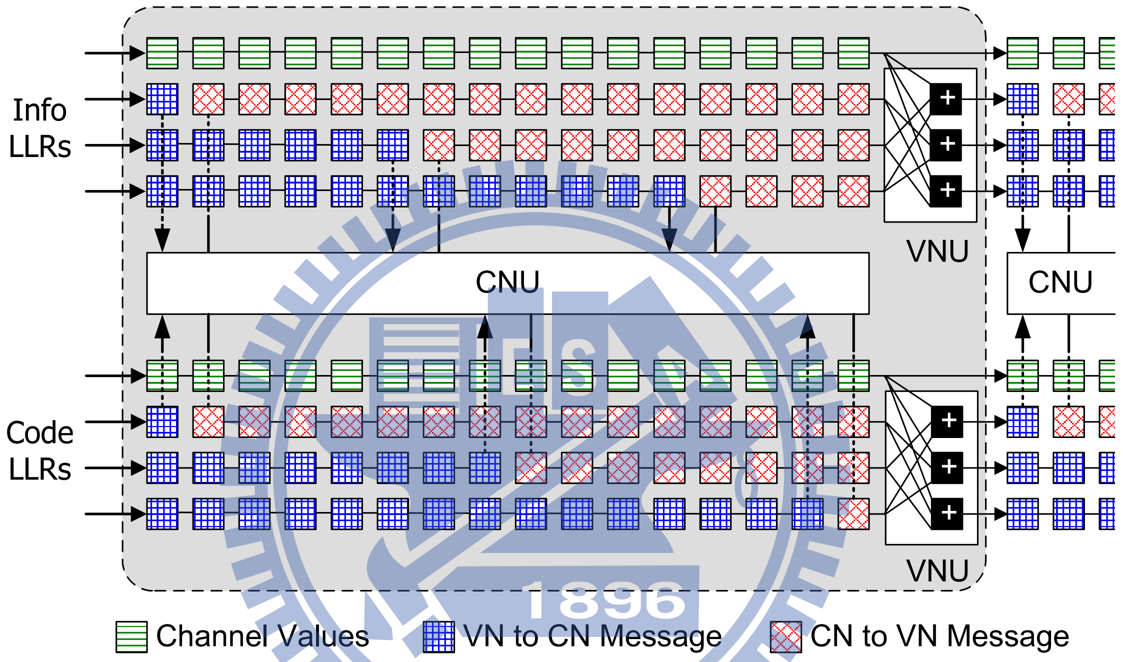


Figure 4.1: Conventional processor architecture.

4.1.2 On-Demand Variable Node Activation (OVA) Scheduling

Studies have shown that the message-passing schedule affects the decoding convergence rate 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 [16] 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 [17] [18] that check nodes could access the most recent messages. We use the same example given in (2.18) to demonstrate the decoding scheduling 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.

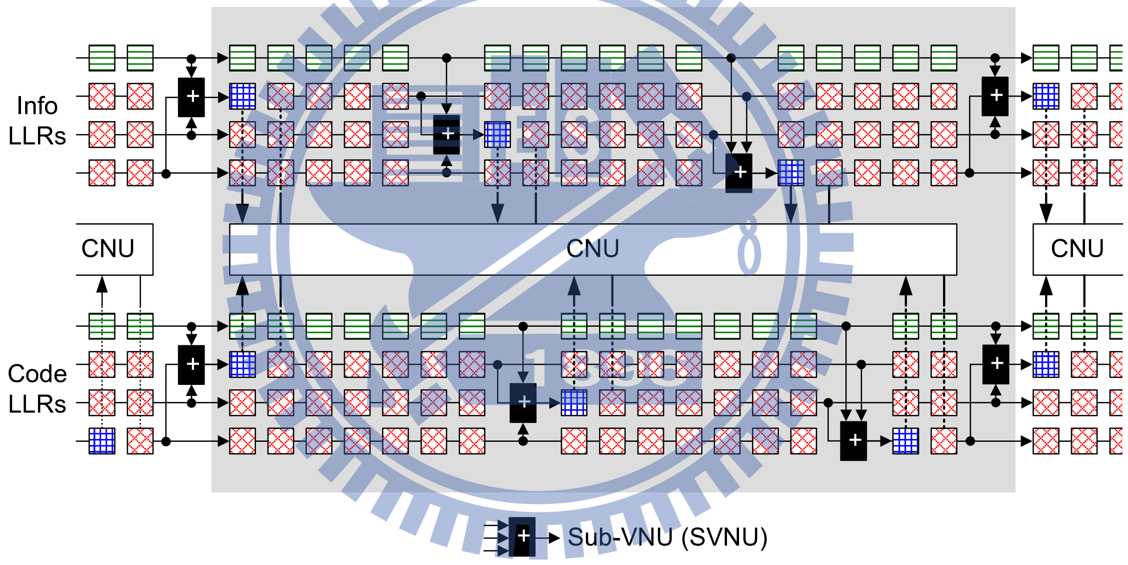


Figure 4.2: Processor architecture of on-demand variable node activation scheduling.

It can be seen from Figure 4.2, 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 4.3, 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.

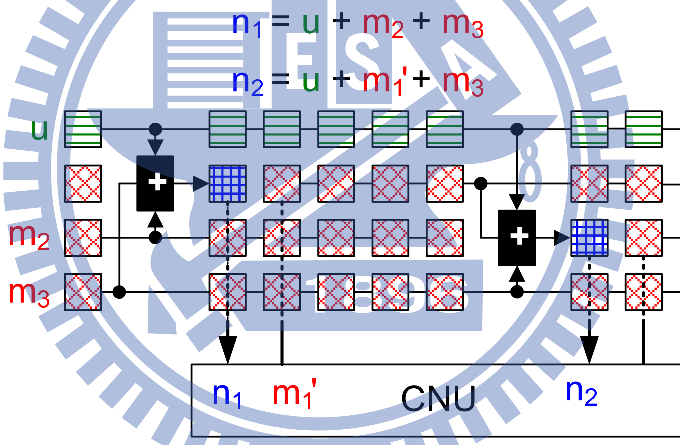


Figure 4.3: The illustration of on-demand variable node activation scheduling.

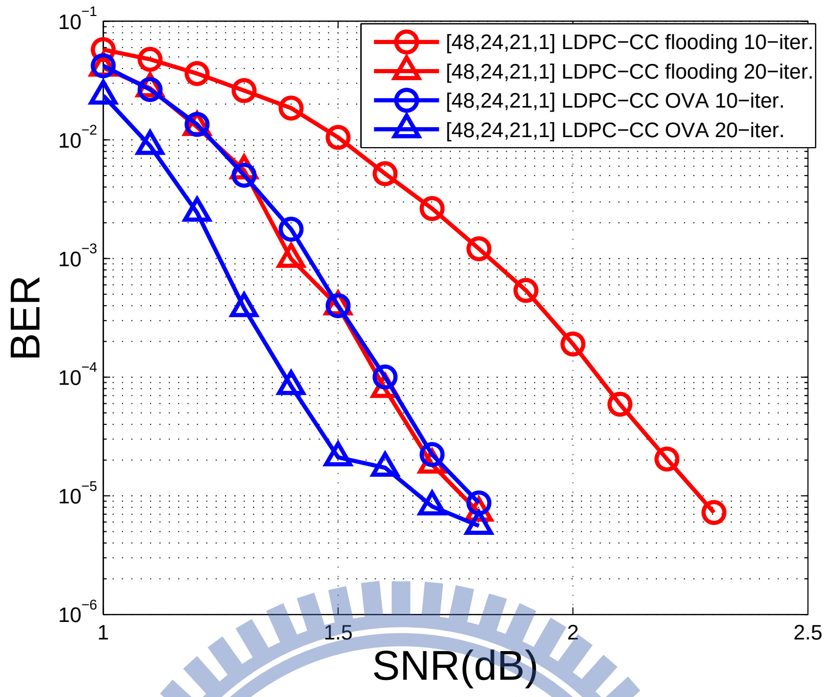


Figure 4.4: BER performance of OVA scheduling and flooding scheduling. [floating-point]

4.1.3 Modified OVA Scheduling

Seeing that the variable-to-check message n_2 is irrelevant to m_2 in the equation in Figure 4.3, it is not necessary to keep the value after the variable node updating of n_1 any more. Hence, we can put the received channel value into the storage row of second degree instead of the original m_2 without break the regularity in the decoder architecture. Figure 4.8 shows the new processor architecture. However, in our implementation, it can reduce 30.77% storage requirements. This modified OVA scheduling is similar to the OVA scheduling with channel value concealing which is proposed in [9], but this method is more suitable to the memory-based design because the quantization bit number for every time instant are all the same. But the shortcoming is that it can't employ the re-timing technique.

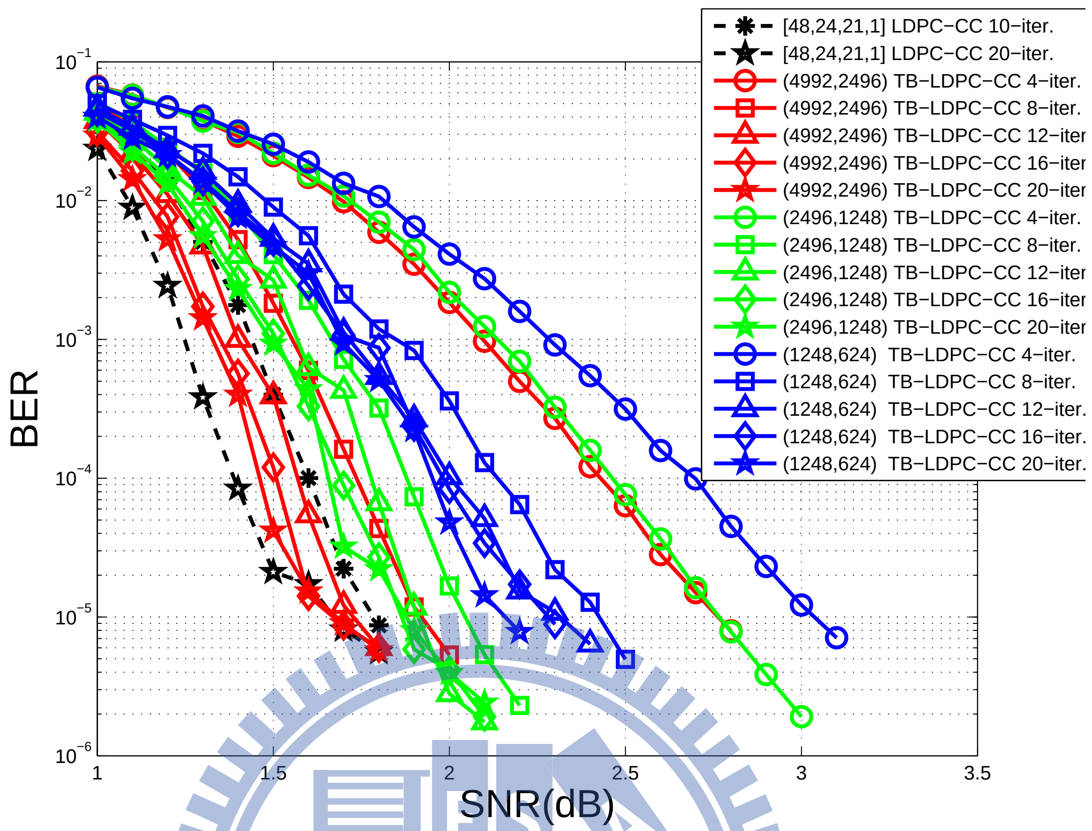


Figure 4.5: Performance curve of the constructed TB-LDPC-CC with OVA scheduling. [floating-point]

4.2 Memory-Based Architecture

For the low-power issue, we adopt a memory-based architecture in our VLSI implementation. In addition, using the memories instead of the registers to store the messages on Tanner graph can reduce the loading of the clock tree, which can benefit the routing complexity. In this section, we are going to introduce the distinguishing features in our work.

1. **Memory Address Transferring Instead of Data Shifting**

   Seeing that all the messages is stored in the memories, we can use the address to indicate the head of data, namely the time indication. Since there is no data shifting between processors, the number of iterations is not relevant to the number of processors now. Hence, we can arbitrarily alter the decoding iterations with the trade off between error-

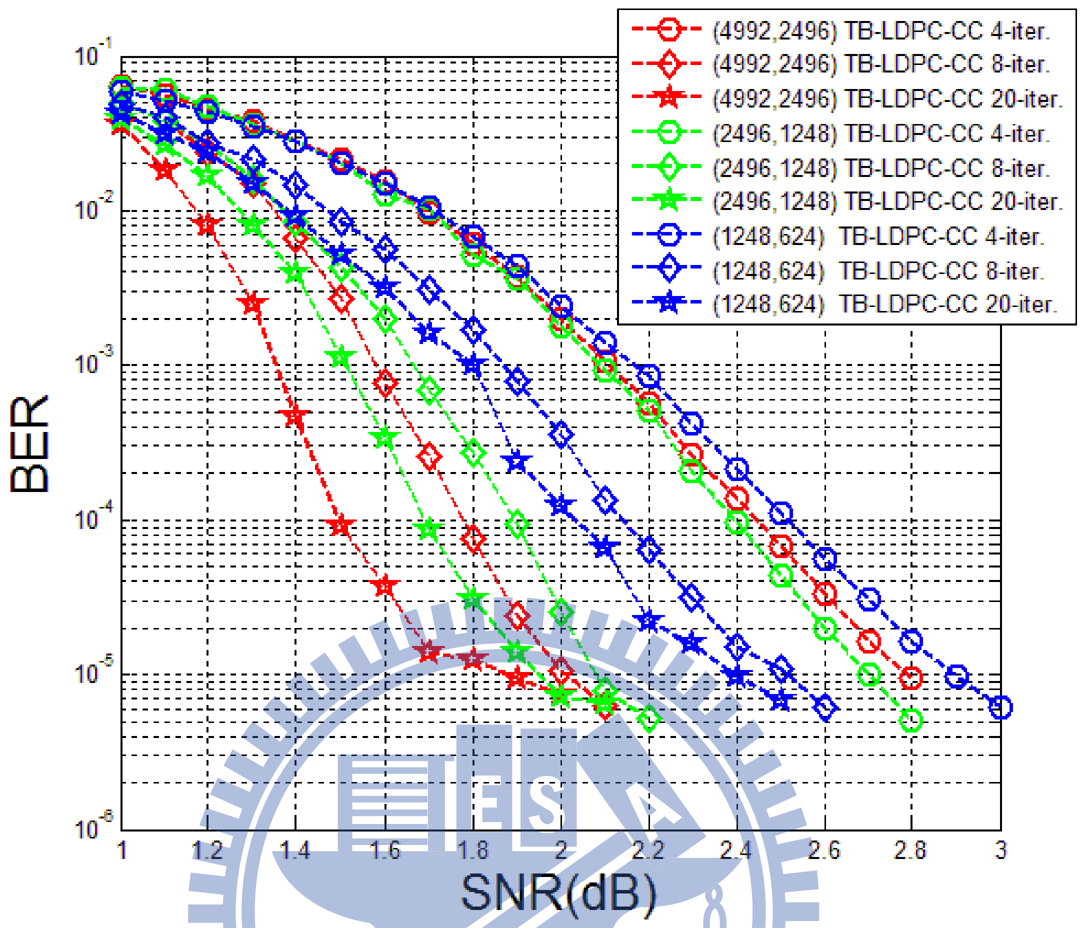


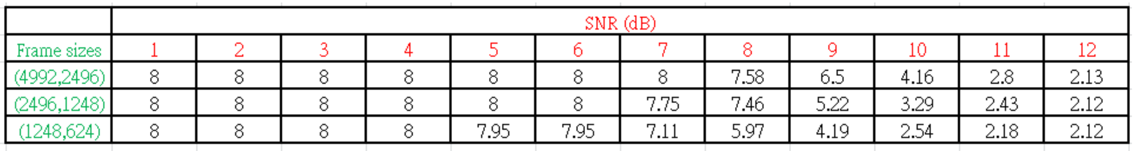
Figure 4.6: Performance curve of the constructed TB-LDPC-CC with OVA scheduling. [(6,2)fixed-point]

correcting performance and decoding throughput.

2. Channel Value Location Pre-Computation

While the channel value and the variable-to-check message are stored in the same memory bank, the channel value location should be pre-computed in the initial stage of decoding procedure due to the modified OVA scheduling, or the decoder architecture regularity would be broken. The corresponding degree of the channel value is dependent on the time instant, namely, the address of the memory. In this case, we need little circuitry overhead to determine the correct location of channel value.

3. Memory Banks Sharing Between Decoding Processors



	SNR (dB)											
Frame sizes	1	2	3	4	5	6	7	8	9	10	11	12
(4992,2496)	8	8	8	8	8	8	8	7.58	6.5	4.16	2.8	2.13
(2496,1248)	8	8	8	8	8	8	7.75	7.46	5.22	3.29	2.43	2.12
(1248,624)	8	8	8	8	7.95	7.95	7.11	5.97	4.19	2.54	2.18	2.12

Figure 4.7: Average iterations with early termination at different SNRs.

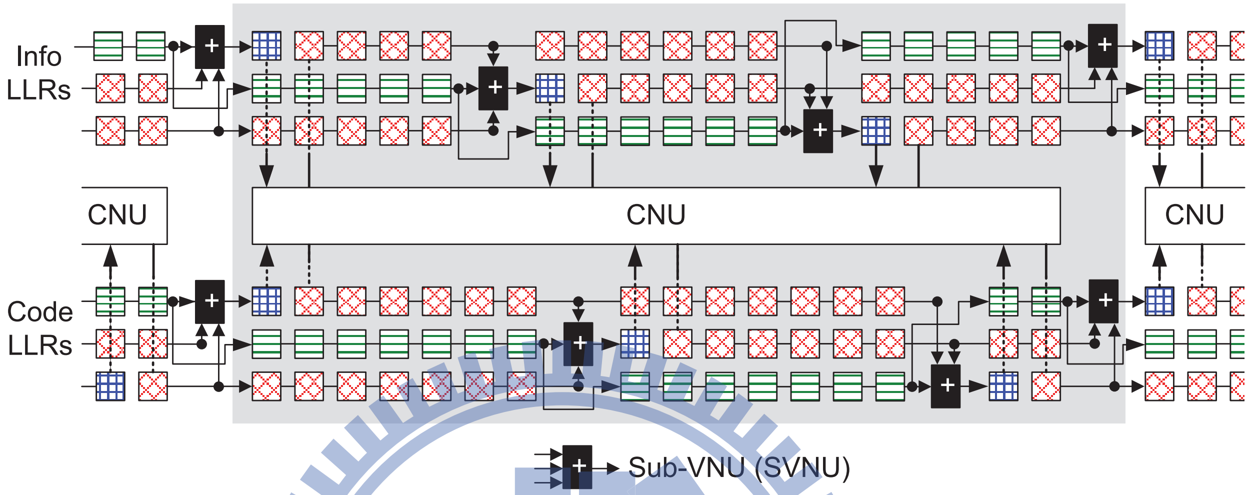


Figure 4.8: Processor architecture of modified OVA scheduling.

As the architecture of the processors are identical, the corresponding memories can be merged together into a larger one due to the same behavior of the individual processors. Therefore, the decoder area can be decreased. However, because the number of bits of a word in the memory is fixed to a maximum value, it is not true that all the corresponding memory banks can be combined together. In our decoder implementation, only the memory banks corresponding to the variable nodes with $degree - 1$ and $degree - 2$ in the polynomial parity-check matrix can be merged through different processors.

4. 4-Stage Pipeline to Enhance the Data Throughput

Considering the memory bandwidth in our work is much less than the register-based design, we insert pipeline registers to break up the critical path into small pieces for improving the throughput by increasing clock frequency. Note that the pipeline register inserted may discontinue the connection between processors, it will result in much inconvenience.

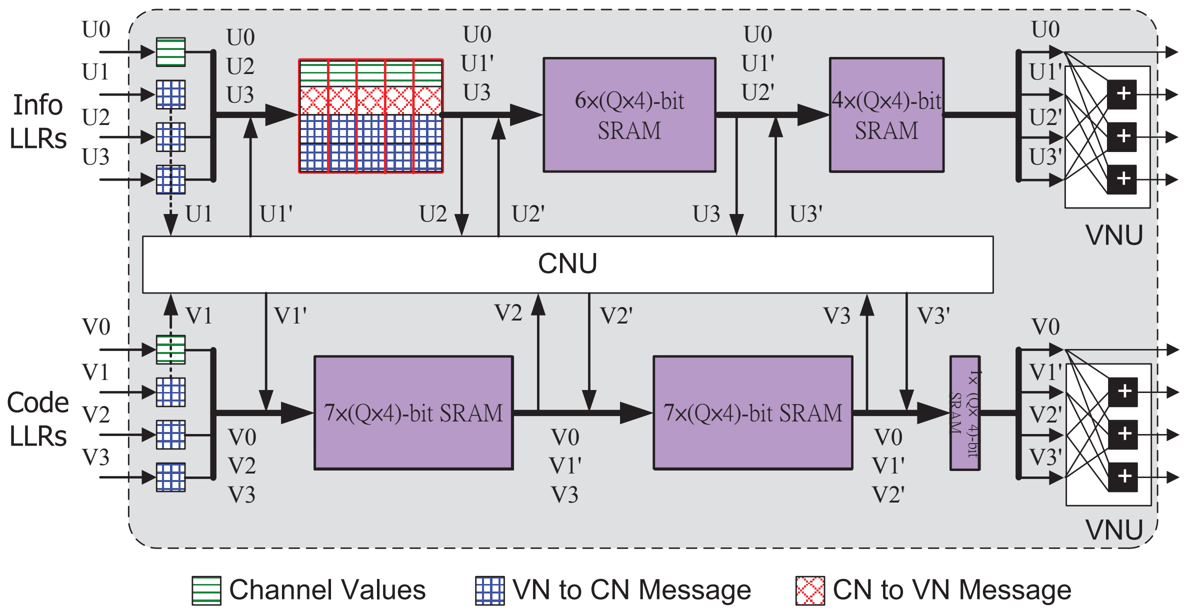


Figure 4.9: Memory-based processor architecture of modified OVA scheduling.

Fortunately, we can solve this problem by preventing it at the encoding stage.

5. **Parallel Decoding with Quasi-Cyclic Characteristic**

Due to the QC characteristic, the elements in the same circulant can be stored in the same word of the memory. This is an amazing property for parallel operations because the connections between variable and check nodes are regular and the memory bandwidth efficiency is better. Besides, the connections between the variable and check nodes are pre-determined instead of using the barrel shifters [19] because the connected ”variable nodes” are not the same between the layers even though they are in the same group of the columns in the polynomial parity-check matrix.

6. **4-Layers Decoding for Avoiding Memory Conflict**

Because all the variable nodes which are belong to the same column in the polynomial parity-check matrix are stored in the same memory bank, it will need to access different memory address simultaneously if two or more ”layers” are activated. As a result, we

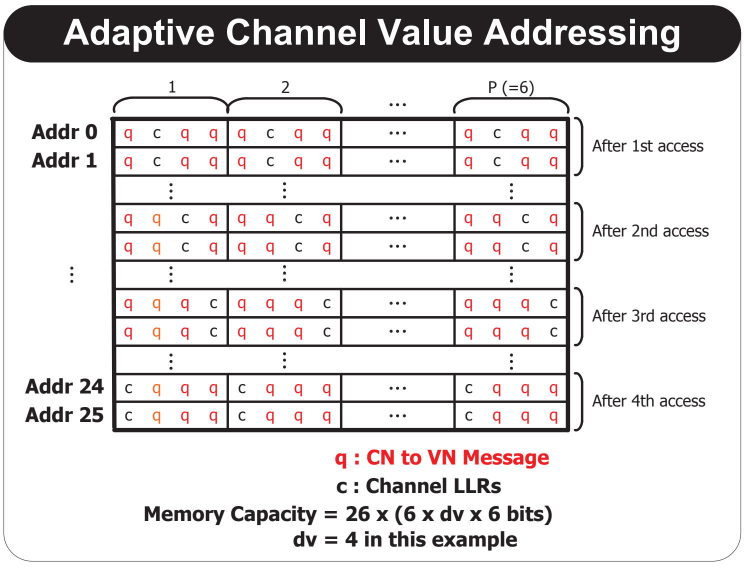


Figure 4.10: Memory alignment of the channel value.

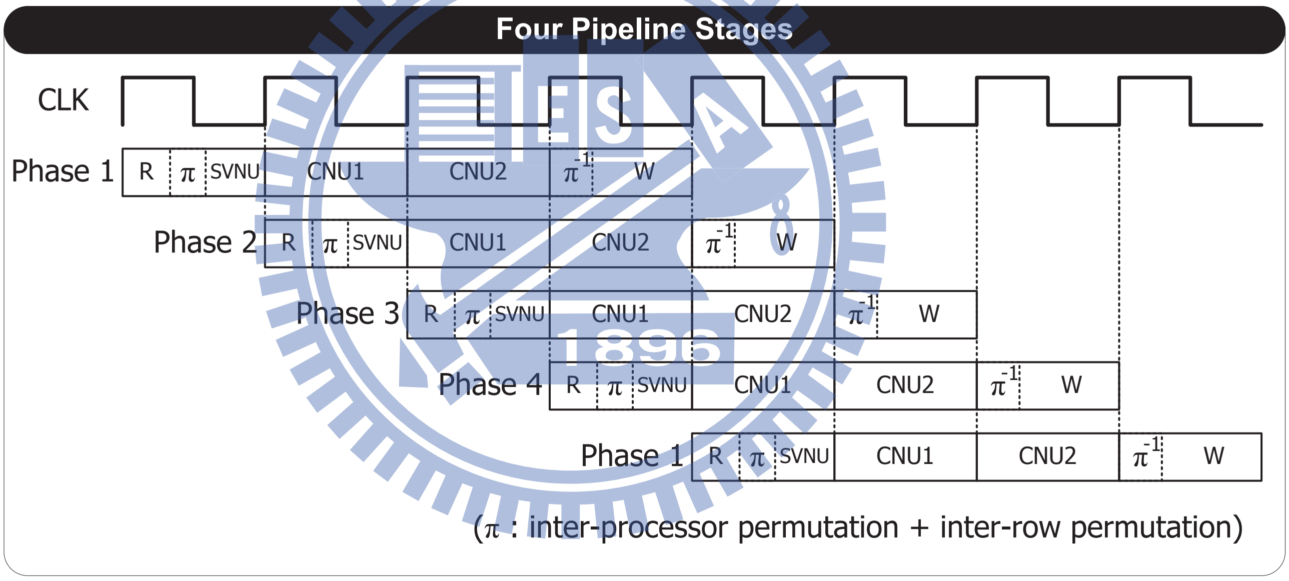


Figure 4.11: The timing diagram of the 4-stage pipeline.

break the code into four layers to perform the decoding procedure owing to the properties of the code structure itself.

7. Control Signal Duplication

While the bandwidth of data bit transferring is quite large, the fanout of the control signal is extremely high. Consequently, we have to duplicate the control signals to lower the fanout; otherwise, the critical path will be increased due to the logic effort imbalance. Basically, we duplicate the control modules until the worst paths are not caused by the control signals. However, it is amazing that the timing path is relieved and the area is reduced even the number of control modules is increased.

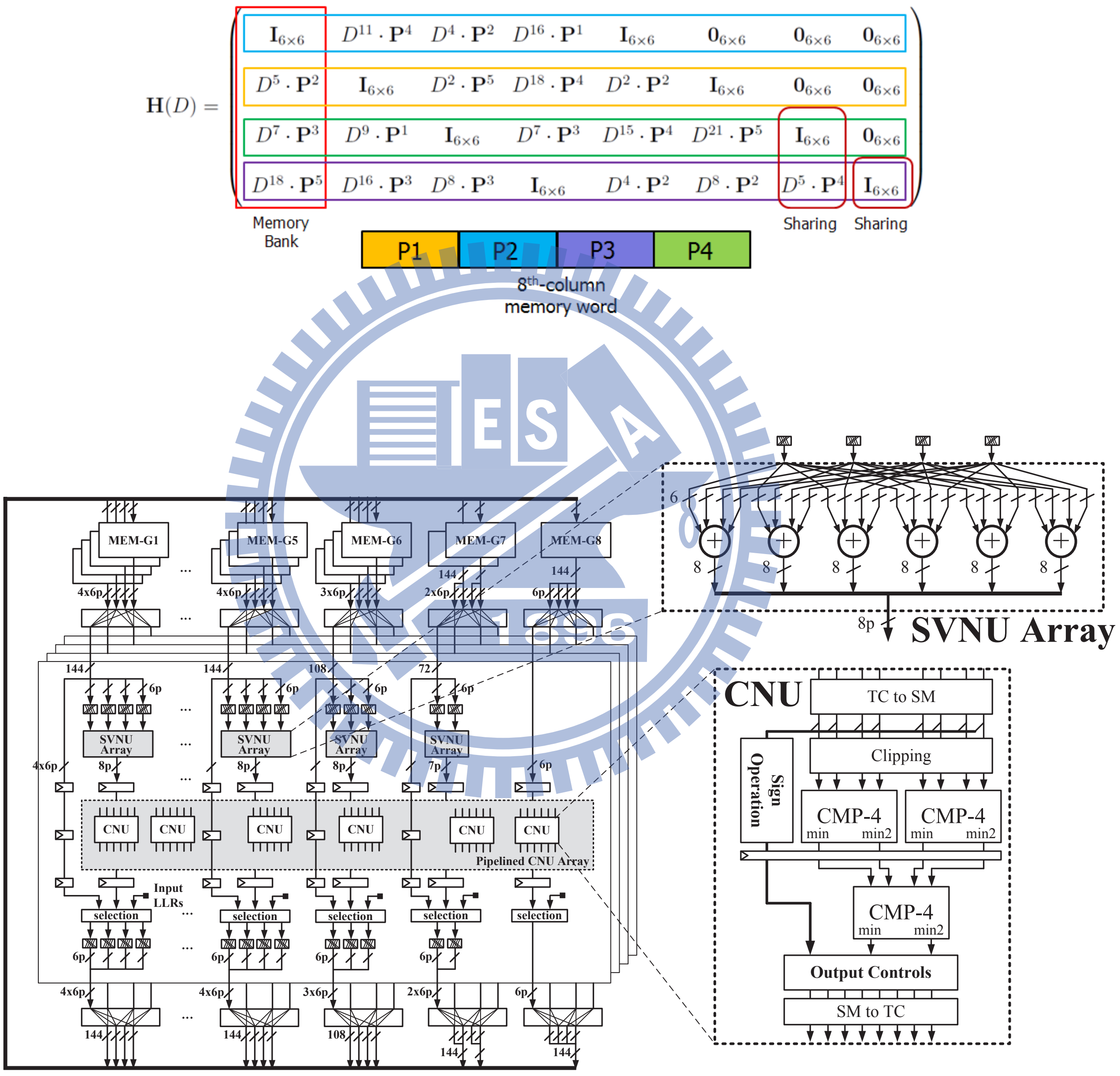


Figure 4.12: Memory-based decoder architecture with modified OVA scheduling.

However, there are some design challenges we have to face. First of all, since the memory bandwidth is limited, we need very large amount of memory banks to maintain the throughput.

But, it will cause huge difficulty while at floor planning stage in automatic place and route (APR). How to place the memory banks is going to be very important. Secondly, in the code construction stage, the polynomial coefficient difference between ”layers” should be greater than 1 to avoid the memory conflict. Third, the power planning should be strong enough to relief the IR-drop.

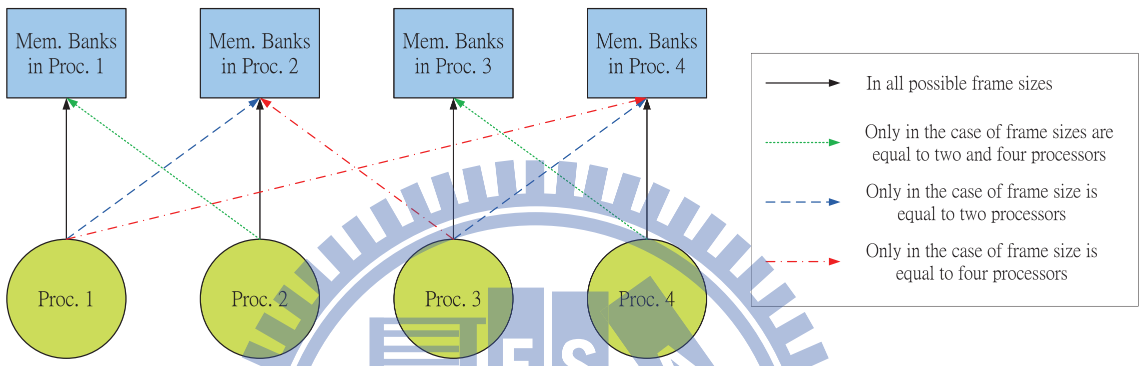
Multiple Code Lengths Handling Capability

Besides, thanks to the tail-biting scheme, we also provide multiple code length handling capability; in other words, the decoder is able to decoding codewords with different frame sizes. We can change the mode to decode more smaller codewords concurrently, or to decode a larger size codeword one at a time. For an instance, while the channel is noisy, we can use the larger code size one to perform better performance. On the contrary, we use the smaller codewords for lower power consumption when the channel situation is pleasing. Moreover, unlike the LDPC block code with the single frame size, due to the capability of decoding different frame sizes, it will more suitable for different applications, too.

It is worth to mention that the tail-biting LDPC convolutional code is a QC block code. The decoder architecture can be implemented with the conventional technique which is often utilized in LDPC block code to obtain higher data rate. Nevertheless, this will be extremely hard to fulfil the capability of dealing with multiple code lengths. Consequently, we choose to preserve this admiring capability in our design.

Permutation Network

As shown in Figure 4.12, the inter-processor and inter-row permutation network are introduced in out design. Both of them are composed of a series of multiplexers circuitry. The inter-row permutation network is used to solve the different cyclic shift between the layers. On the other hand, the inter-processor permutation network deals with the different data resources and data destinations such that our design can handle different frame sizes. In fact, in our implementation, there are three frame sizes available; moreover, the frame sizes are referred to the length of one processor, two processors, and four processors, respectively. Hence, there are some simplifications adopted in the inter-processor permutation network. The detail connections are described in Figure 4.13.



Mem. Banks in Proc. 1
Mem. Banks in Proc. 2
Mem. Banks in Proc. 3
Mem. Banks in Proc. 4
Proc. 1
Proc. 2
Proc. 3
Proc. 4
In all possible frame sizes
Only in the case of frame sizes are equal to two and four processors
Only in the case of frame size is equal to two processors
Only in the case of frame size is equal to four processors

Figure 4.13: The description of inter-processor permutation network.

Chapter 5

Simulation and Implementation Result

Our proposed time-invariant irregular LDPC convolutional code decoder chip integrates the following techniques. For scheduling optimization, the modified on-demand variable node activation scheduling is presented to improve decoding performance and reduce storage requirements. Memory-based design can lower the routing complexity and the power consumption. The capability of handling multiple frame sizes can further lower the power consumption and apply to various applications.

The test chip was implemented with UMC 90nm 1P9M CMOS process.

5.1 System Specification

With the post-PC era, all the personal, family or business electrical equipment are toward the way to the development of the portable communication device. As a result, Wireless Personal Area Network (WPAN), including UWB (Ultra Wide Band) system, Bluetooth, ZigBee and NFC (Near Field Communication) applications, plays an important role in our daily life. The IEEE 802.15.3c standard [20] is developed for WPAN while focusing on the indoor over Gb/s data rate transmission for operation in the 60 GHz radio frequency band. In fact, our proposed code was constructed on the base of the parity-check matrix of the channel decoder in IEEE 802.15.3c standard. Hence, we are focus our work on its applications. However, in this standard, the highest data rate request is greater than 5 Gb/s, and the demanding data rate for code-rate $R = 1/2$ is 1.76 Gb/s. There are three modes, single-carrier (SC), high-speed interface (HSI) and audio/video (AV) modes respectively, and one common mode, for enabling the switching among different modes, specified in the standard. In this thesis, we simulate the code bit-error-rate (BER) performance with the binary-input additive white Gaussian noise (BI-AWGN) channel model.

5.2 Parameter Determination

When we are going to implement the decoder, there are some parameters we have to discuss and make the choice. In our work, we adopt the normalized min-sum algorithm to perform the message-passing decoding. While the complexity of log-BP algorithm is too huge and the BER performance of conventional min-sum algorithm is unfavorable, the normalized min-sum algorithm possesses good performance and low complexity. However, the scaling factor is chosen to be 0.75 for the error-correcting performance and hardware implementation concern.

Furthermore, the log-likely-hood ratio should be quantized to be fixed-point value. From the simulation result, we choose the 6-bit quantization, 4-bit integer part and 2-bit fraction part, in our design with tolerable performance loss. Although the performance is better with 7-bit quantization, it doesn't have large difference. But in the view of hardware cost, the cost of 7-bit quantization decoder is much higher. Figure 5.1 shows the LDPC-CC BER performance with different bits quantization using normalized min-sum with scaling factor 0.75.

After the quantization bit number is decided, we are going to determine the parallelization factor to enhance both the throughput and error-correcting performance. Whereas the maximum

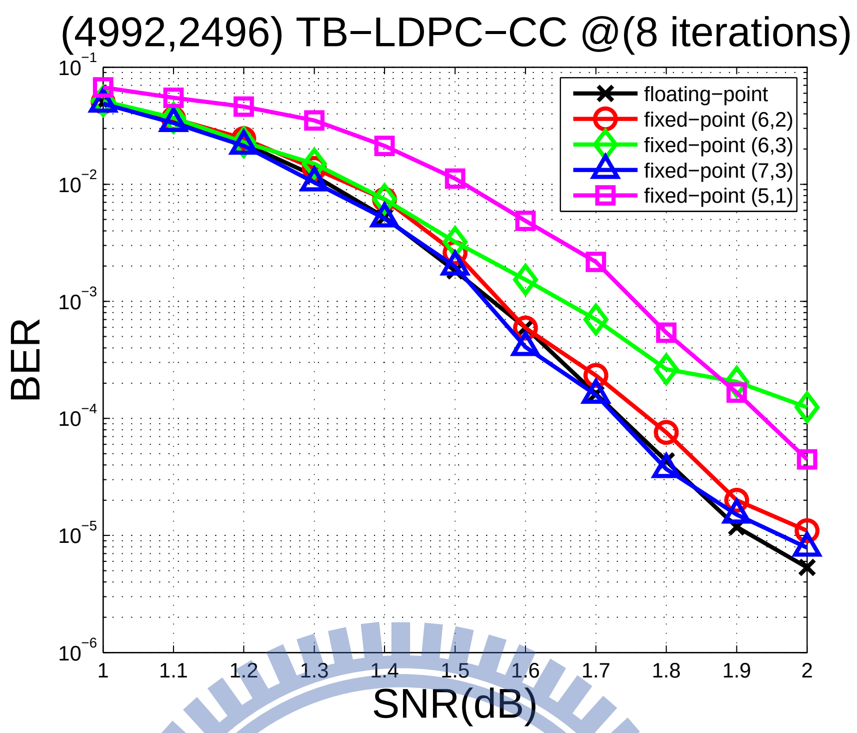


Figure 5.1: Fixed-point simulation results using OVA scheduling with 8 iterations.

bit number in a word of the memory is 144, we choose the parallelization factor as 6 which will exactly meet the maximum bit number. The reason why we choose this number is that if the parallelization factor is greater, it will need more memory banks to store the messages. On the other hand, if the parallelization factor is smaller, the cost is just little reduced. Hence, the factor which meet the maximum word bit number is the best choice.

Now, seeing that the parallelization factor is determined, the next step is to determine the memory size, the length of the FIFO of encoder, of the LDPC convolutional code. However, in order to compare to the performance of the $(491, 3, 6)$ LDPC convolutional code in [13], we choose the memory size to be 21 such that the constraint length of our code is closed to that the $(491, 3, 6)$ LDPC convolutional code has. From the simulation result, we find out that the LDPC-CC we constructed has the better performance while in the low SNR situation. In high SNR condition, our code performance is worse than the $(491, 3, 6)$ LDPC-CC because of the error floor phenomenon. However, we take the analysis of the error floor, and we discover that it is caused by the $degree - one$ components in the parity-check matrix which cannot be updated through the iterations. Besides, it is worth to mention that the memory size is small, thus the hybrid-partitioned FIFO architecture in our previous work [9] is no longer suitable to use. In other words, this code is suitable for memory-based design.

Finally, concerning with the decoder area and the available frame sizes, the number of processors is chosen to be 4. And, memory bank sharing technique is performed efficiently in this case, too. The codeword lengths then have 3 choices as the length of $1, 2, 4$ processors. However, the error-correcting performance insufficiency can be overcome by the circular decoder architecture due to the tail-biting technique. Besides, the early termination scheme is introduced to improve the data rate in our design.

5.3 Chip Implementation

5.3.1 Testing Consideration

In our design, we have the following testing mode to verify the chip functionality.

- **Normal Function Operation**

  In this mode, we can get the corresponding decoded output of the input pattern.

- **Build-In Automatic Output Comparison**

  In this mode, we can input the correct output into the decoder; therefore, the decoder compare that the correct sequence is consistent with the decoded output or not. This can check the functionality with higher speed and prevent from the failure occured at the output pad.

- **Random Number Generation Test**

  In case of the failure occured at the input pad, we use this mode to see if the decoder works well.

5.3.2 Measurement Result

From the measurement result, our decoder's maximum clock operating frequency is up to 305 MHz, and the throughput is 1.83 Gb/s at 4 iterations. The power consumption is $275mW$ under the condition that the voltage supply is 0.8 V at 305 MHz. When supply voltage is scaled to 1.2V, the throughput can be enhanced to 2.1 Gb/s with larger power consumption.

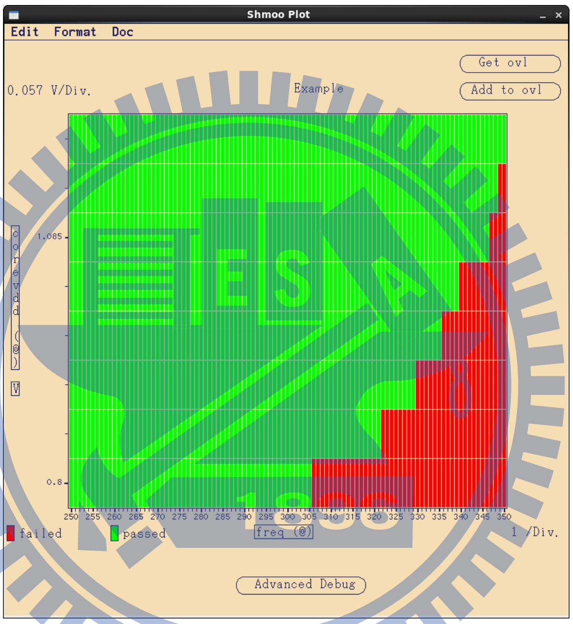
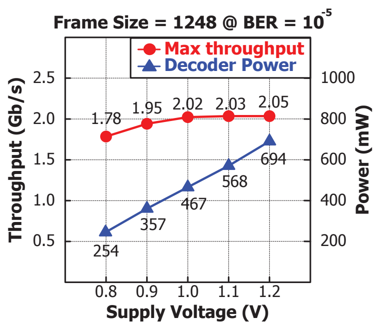


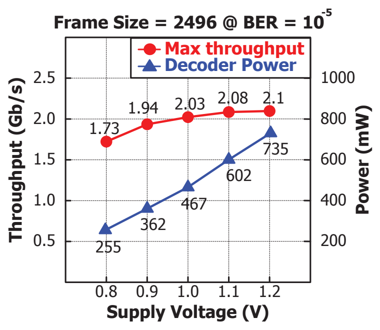
Figure 5.2: The Shmoo plot of the test chip.

5.4 Implementation Result and Comparison

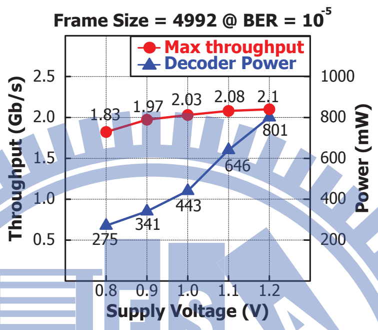
The total core area in our design is $2.27mm^2$ (I/O buffer included), and the chip utilization is up to 90.2%. Table 5.1 present a brief summary of the proposed time-invariant tail-biting LDPC convolutional code.



(a) Mcode frame size = 1248.



(b) code frame size = 2496.



(c) code frame size = 4992.

Figure 5.3: Measured throughput and power at different frame sizes.

Table 5.1: Chip Summary.

Process	UMC 90nm 1P9M
FEC	TB-LDPC-CC
Constraint Length	$(21 + 1) * 6 * 8 = 1056$
Input Quantization	6 bits
Chip Utilization	90.2%
Parallelization Factor	6
Processor Number	4
Memory	104.8 Kbits
Decoder Area	2.18 mm^2
Max. Clock Frequency	305 MHz
Max. Data Rate	1.83 Gb/s
Decoder Power	275mW

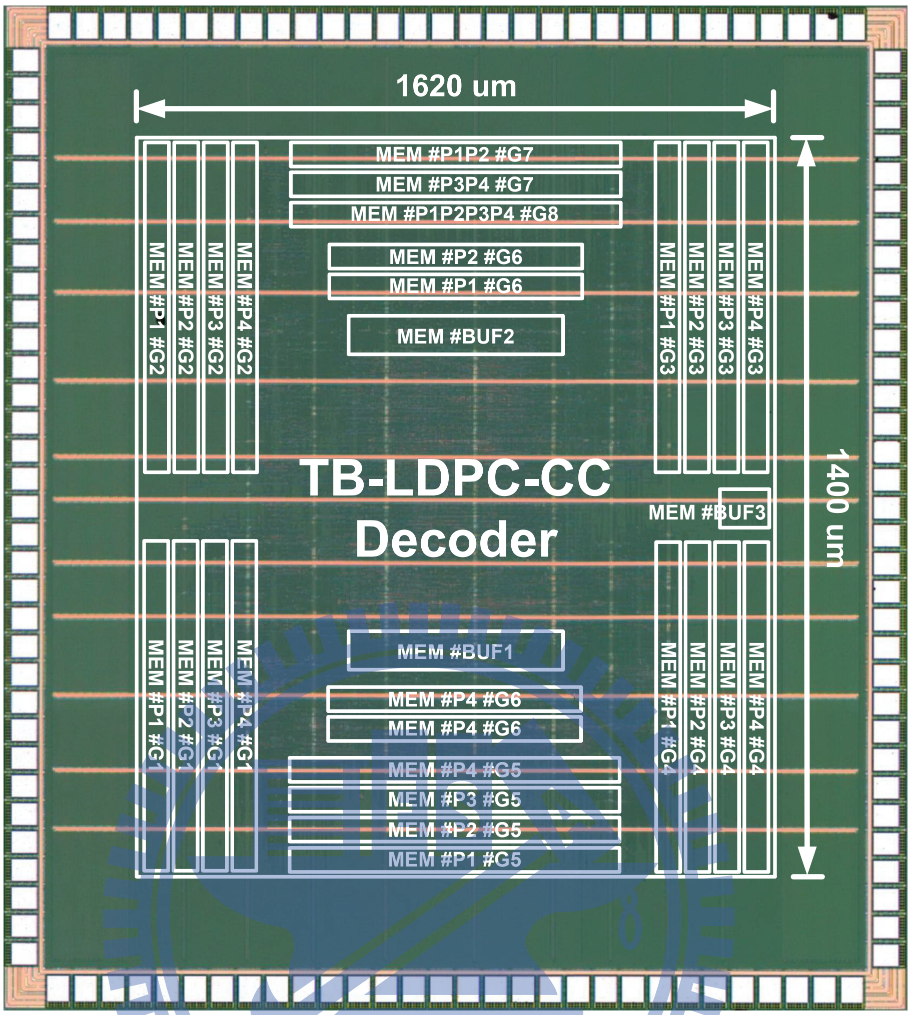


Figure 5.4: Chip micrograph.

Table 5.2: Comparison with state-of-the-art.

	This work	[9]	[21]	[22]	[14]	[23]
FEC Type	TB-LDPC-CC	LDPC-CC	LDPC-CC	LDPC-CC	LDPC-CC	LDPC-BC
Constraint Length / Block Size	1056	984	258	960	946	672
Code-Rate	$1/2$	$1/2, 2/3,$ $3/4, 4/5, 5/6$	$1/2$	$1/2$	$1/3$	$1/2, 5/8,$ $3/4, 7/8$
CMOS Technology (nm)	90	90	90	90	90	65
Input Quantization (bit)	6	6	8	6	6	6
Processor / Iteration	4	5	3	11 (10)	1	5
Memory (Kb)	104.8	52.5	-	221.28	-	0
Chip Utilization (%)	90.2	87.8	-	-	-	73.3
Decoder Area (mm^2)	2.18	2.24	1.5	5.363	0.924	1.4
Max. Frequency (MHz)	305	198	600	256.4	250	197
Max. Data Rate (Gb/s)	1.83	2.37	0.6	1.025	2	3.57
Power (mW)	275	287	368.7	682	-	469.7
Energy Efficiency (nJ/bit/proc)	0.0376	0.024	0.2048	0.0665	-	0.0263
Chip Status	Measurement	Measurement	Measurement	Post-Layout	Synthesis	Measurement

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed a time-invariant tail-biting LDPC convolutional code chip design. With decoding scheduling optimization, the modified OVA scheduling is used to achieve twice faster decoding convergence speed than the standard decoding schedule. Furthermore, this technique also saves 30.77% storage requirements. And, the memory-based design are adopted to save the power consumption. And, the multiple frame sizes handling capability can lower the power and adapt to multiple applications.

Integrated with these schemes, the test chip is implemented in a UMC 90nm 1P9M CMOS process. The decoder area is $2.18mm^2$ with 90.2% chip utilization. The maximum clock frequency is 305 MHz, and the data rate is 1.83 Gb/s at 4 iterations.

6.2 Future Work

In this thesis, the code performance has obvious error floor phenomenon. Hence, to lower the error floor is the problem for future studies. And, applying the tail-biting scheme to a time-variant code is another challenge for general LDPC convolutional code construction. In addition, since the LDPC convolutional code which is constructed with the polynomial parity-check method has some connections with the QC-LDPC block code, the dual mode implementation handling both block and convolutional code is possible. A decoder architecture which can support convolutional code and block code is our next goal.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] C.-L. Chen, Y.-H. Lin, H.-C. Chang, and C.-Y. Lee, "A 2.37Gb/s 284.8mW rate-compatible (491,3,6) LDPC-CC decoder," in *2011 Symposium on VLSI Circuits (VLSIC)*, june 2011, pp. 134 –135.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] H. Zhou and N. Goertz, “Unavoidable cycles in polynomial-based time-invariant ldpc convolutional codes,” *Wireless Conference 2011 - Sustainable Wireless Technologies (European Wireless), 11th European*, 2011.

[16] 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.

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

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

[19] D. Oh and K. Parhi, “Area efficient controller design of barrel shifters for reconfigurable LDPC decoders,” in *IEEE International Symposium on Circuits and Systems 2008. ISCAS 2008.*, may 2008, pp. 240 –243.

[20] “IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs) Amendment 2: Millimeter-wave-based Alternative Physical Layer Extension,” *IEEE Std 802.15.3c-2009 (Amendment to IEEE Std 802.15.3-2003)*, pp. c1 –187, 12 2009.

[21] 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.

[22] Y.-L. Ueng, Y.-L. Wang, L.-S. Kan, C.-J. Yang, and Y.-H. Su, “Jointly Designed Architecture-Aware LDPC Convolutional Codes and Memory-Based Shuffled Decoder Architecture,” *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 4387 –4402, aug. 2012.

[23] 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.