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 3GPP-LTE 規格的可重組渦輪解碼器之研究

Research on Reconfigurable Turbo Decoder for 3GPP-LTE Applications

學生：李永裕

指導教授：張錫嘉 教授

中華民國九十七年十月

適用於 3GPP-LTE 規格的可重組渦輪解碼器之研究

Research on Reconfigurable Turbo Decoder for 3GPP-LTE Applications

研　究　生：李永裕　　　　Student：Yung-Yu Lee

指導教授：張錫嘉教授　　Advisor：Hsie-Chia Chang

國　立　交　通　大　學

電子工程學系　電子研究所　碩士班

碩　士　論　文

A Thesis
Submitted to Department of Electronics Engineering & Institute Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
In Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Electronics Engineering
October 2008
Hsinchu, Taiwan, Republic of China

中華民國九十七年十月

適用於 3GPP-LTE 規格的可重組渦輪解碼器之研究

學生：李永裕　　　　　　　　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

本論文提出了一個支援全模式且可重組的渦輪解碼器。我們提出的解碼器可以支援所有定義在 3GPP-LTE 規格裡的編碼長度。而為了提高傳輸速度，遞迴解碼的平行架構更是受到關注。具有 contention-free 特性的二次多項式交錯器也被應用在渦輪解碼器的平行架構中。Max-Log MAP 演算法也被採用，可使在極小的效能損失下有效的減低硬體複雜度。此外，可重組的 1/2/4/8-MAP 解碼器設計也被提出，此設計可使在解碼過程中，根據所需的效能或傳輸速度，提供一個解碼器可重組的功能。而依據二次多項式交錯器的特性，一個稱為 residue-only interleaver 的方法可被用來減少記憶體的使用量。

根據在 90nm製程下的實驗結果，在 8 次迴圈的解碼模式下可以達到 130Mb/s 的傳輸速度，晶片面積是 2.10mm^2。此外，在 0.9V的供應電壓，277MHz的操作頻率且編碼長度 6144 下，功率消耗經量測過後為 149.03mW。

Research on Reconfigurable Turbo Decoder for 3GPP-LTE Applications

Student：Yung-Yu Lee　　　　Advisor：Dr. Hsie-Chia Chang

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

In this thesis, a fully compliant and reconfigurable turbo decoder is presented to support all block lengths specified in 3GPP-LTE system. The contention-free quadratic permutation polynomial (QPP) interleaver is also introduced for parallel architecture of turbo codes. The parallel processing of iterative decoding is of interest for throughput increasing. The Max-Log MAP algorithm is used to reduce the hardware complexity with the minimized performance loss. Moreover, the reconfigurable 1/2/4/8-MAP decoders is proposed to decode the received codewords based on performance or throughput expected in different conditions. Based on QPP characteristic, the residue-only interleaver is adopted to reduce the memory storage.

After implementation in a 90-nm 1P9M technology, the 130Mb/s data rate with 8 decoding iterations can be achieved in the 2.10 mm^2 core area containing 602K gates. According to the post-layout simulation, the power consumption is 149.03mW worked at supply voltage 0.9V and clock rate 277MHz with block length 6144.

Acknowledgement

Firs of all, I would like to express my deepest gratitude to my advisor, Dr. Hsie-Chia Chang, for his enthusiastic guidance and great patience. I learn a lot from his positive attitude in way of research and many other areas. Secondly, I really appreciate Dr. Shyue-Win Wei, Dr. Chun-Hsiung Chuang and Dr. Mao-Ching Chiu for serving as my committees. Heartfelt thanks are also offered to all members in the OCEAN and OASIS group for their constant encouragement and assistance whenever I need.

I would like to show my sincere thanks to my parents for their invaluable care and love. When I was depressed, they comforted and encourage me, and whenever I met difficulty, their heart and mind were always by my side. I also appreciate for their supply to every thing I need without any complaining. Without them, I would not exist in this world. Without them, I would not live to this age. Without them, I would not be here today. Sincerely, it is not enough to give all my appreciation to my dear parents. I would like to attribute all my achievement and honor to them. Thank you mom and dad, I love you forever.

Finally, I appreciate all my true friends for their immediately encouraging when I was down. They let me know that someone other than my parents in this world are caring me, I am not alone to fight. I am willing to share my honor with you guys, thank you every one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The fourth-generation (4G) is a term used to describe the next complete evolution in wireless communications. Several communication applications in the future may demand for a higher speed channel coding scheme. The approaching 4G wireless systems are projected to achieve 100 Mbps to 1Gbps data rate by 2010. The high throughput turbo codes are required in many 4G communication systems. The two significant examples are 3GPP Long Term Evolution (LTE) and IEEE 802.16e WiMax. Some 4G systems use different types of turbo coding schemes. That is, binary codes is used in 3GPP-LTE and duo-binary codes is utilized in WiMax. Although the 4G is now still in formative stages. They may become commercially in the following years.

1.2 Motivation

The fundamental block diagram of traditional digital communication system is illustrated in Fig. 1.1. The communication system conveys a information source to a destination through a channel. Generally, the system consists of transmitter and receiver via a channel. The transmitter includes source encoder, channel encoder and modulator, is to transform the information into a form that can withstand the effect of noise over the transmission media. Furthermore, the receiver will reverse the signal transformation by demodulator, channel decoder and source decoder. Since the channel impairments such as noise, interference and distortion may cause the error in the received signal, the channel encoder is incorporated in the system to add certain structural redundancy to the source codeword to minimize the transmission errors. These redundant bits can be used for error detecting and correcting. The channel coding eliminate the effects of noise disturbances and can improve the performance, compared with an uncoded system.

Turbo code is one of the most popular channel coding technique for modern digital communication systems. It is impressive by its excellent error correction ability based on the soft iterative decoding. In the Third Generation Partnership Projects 3GPP and 3GPP2 defined detail standard, they both adopt turbo codes as their channel coding scheme and provide the maximum data rate of about 2 Mb/s and 3 Mb/s, respectively. However, the applications in the future may require higher data rate. Therefore, how to increase the throughput and reduce decoding latency may become obstacles for the turbo code in the hardware implementation.

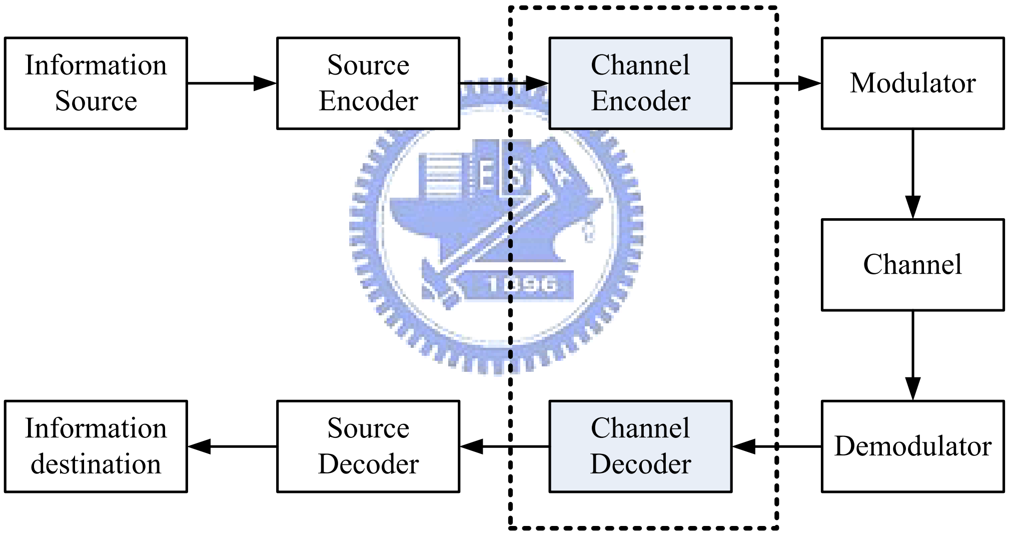


Figure 1.1: Block diagram of digital communication system.

In this thesis, our work is motivated to design a turbo decoder for 3GPP-LTE applications. We attempt to achieve two targets : The first one is to support all block lengths in the standard. The other is to achieve the throughput requirement of 3GPP-LTE specification. Therefore, we employ a contention-free interleaver that fits for parallel decoding architecture to reduce the latency and increase the throughput. Finally, we propose a reconfigurable turbo decoder that fits for whole code lengths and present a practical hardware architecture for the complete turbo decoder with the modest hardware cost.

1.3 Thesis Organization

This thesis consists of 6 chapters. In chapter 2, the concept of turbo principle and iterative decoding algorithm of turbo codes will be introduced. The turbo codes applied in 3GPP-LTE system, which includes encoding, interleaver and decoding procedure are presented in chapter 3. Furthermore, the simulation analysis and parameter decision are also described. Chapter 4 introduces the design of reconfigurable 3GPP-LTE turbo decoder, including the hardware architecture and the characteristic of proposed decoder. In chapter 5, the chip implementation result will be shown in detail. Finally, the conclusion is given in chapter 6. The parameters used in 3GPP-LTE internal interleaver are illustrated in Appendix A.

Chapter 2

Turbo Code

The parallel concatenated convolutional code (PCCC), named turbo code, was first proposed by C. Berrou, A. Glavieux, and P. Thitimajshima in 1993. It has been proved to have a performance near Shannon limit with simple constituent codes concatenated by an interleaver. Turbo code is adopted in 3GPP, 3GPP2 and WiMAX standards due to its excellent error correction ability. In this chapter, both turbo encoding and turbo decoding methods will be described. The error floor effect in turbo decoding and some decoding techniques will also be interpreted.

2.1 Turbo principle

2.1.1 Turbo encoding

The turbo encoder is composed of two recursive systematic convolutional (RSC) encoders, which are connected in parallel but separated by an interleaver. The block diagram of the turbo encoder is illustrated in Fig. 2.1. In the first encoder, the information symbols are encoded to the systematic part $c_0(D)$ and the parity $c_1(D)$; thus, $c_0(D) = x(D)$. The second encoder encodes $\tilde{x}(D)$, the information sequence $x(D)$ after interleaving. However, the systematic part which is also $\tilde{x}(D)$ will be discarded during transmission because x_0 has carried the information sequence. If the code rates of encoder 1 and encoder 2 are R_1 and R_2 respectively, the overall code rate R in Fig. 2.1 will satisfy

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1. \tag{2.1}$$

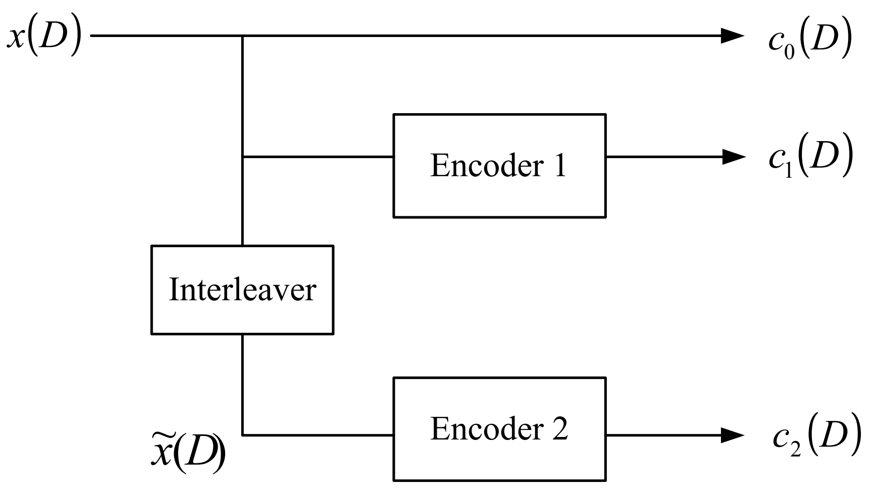


Figure 2.1: Turbo encoder.

In 3GPP2 standard [1], each input bit is encoded as one systematic bit and two parity-check bits for each RSC encoder. Thus, the code rate of each component encoder is 1/3. In order to increase the code rate of turbo code, the systematic bits of the second RSC encoder are not transmitted. Therefore, the output should be $\{X, Y_0, Y_1, Y_0', Y_1'\}$, and the overall code rate is 1/5. It will be shown in Fig. 2.2.

After encoding all information sequence, several tail bits have to be generated to set both component encoders back to zero state. However, it's impossible for a RSC encoder to return zero state by inserting dummy zeros into the encoder directly. Thus, a solution is provided in Fig. 2.3. While encoding input information, the switch is set to position "A". When whole block are encoded, the position of switch is changed to "B" for three additional cycles. This will force all registers back to zero state.

2.1.2 Turbo interleaver

The interleaver is a critical component for channel coding performance of turbo codes. First of all, a proper coding gain can be achieved with small memory RSC encoders since the interleaver scramble a long block message. Besides, the interleaver de-correlates the input of two RSC encoders so that iterative decoding algorithm can be applied between two component decoders. Theoretically, the block size of interleaver is one of the major factors to lower the upper-bound on bit error probability of the turbo code system. The performance upper-bound of turbo code corresponding to a uniform random interleaver

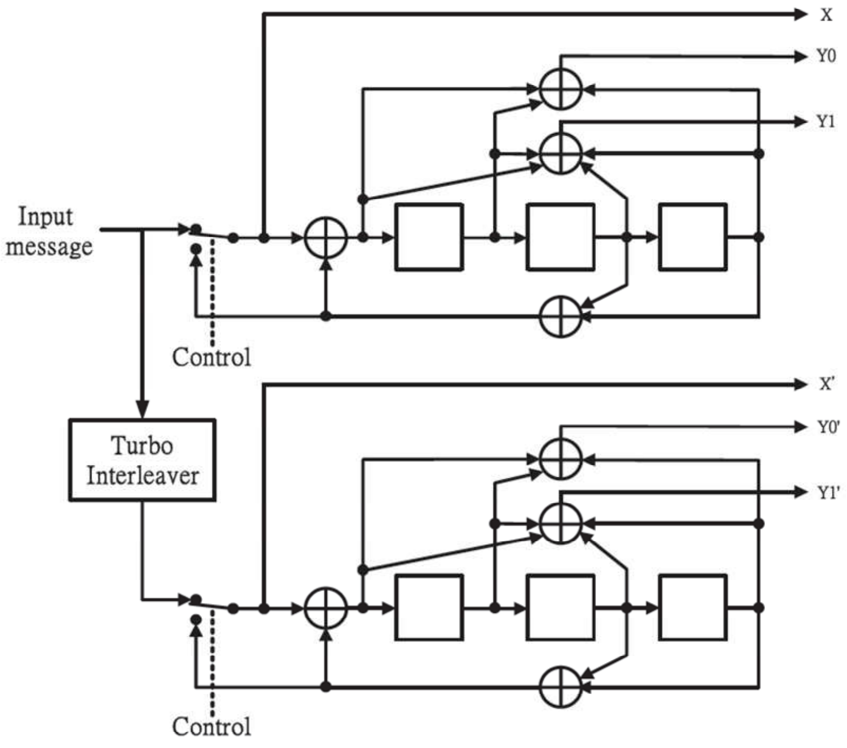


Figure 2.2: Turbo encoder for 3GPP2 standard.

has been evaluated in [2]. The bit-error-probability upper bound of turbo code is approximately proportional to 1/N, where N is the interleaver size. The factor 1/N is also called the interleaver gain.

2.1.3 Turbo decoding

A general idea for iterative turbo decoding is illustrated in Fig. 2.4, where r_s is the received systematic information, r_{p_1} is the received parity information generated by the first RSC encoder, and r_{p2} is the received parity information generated by the second RSC encoder. The iterative turbo decoding consists of two constituent SISO (soft in/soft out) decoders, which are concatenated serially via the interleaver and de-interleaver. An additional interleaver is used to interleave the input systematic information and then provides the interleaved data to the second SISO decoder. During iterative decoding process, each constituent decoder delivers the extrinsic information L_{ex} which is taken as *a priori* information L_{in} for the other constituent decoder after the interleaver and de-interleaver process. That is, $L_{in}^1 = L_{ex}^2$ and $L_{in}^2 = L_{ex}^1$. As the number of iteration

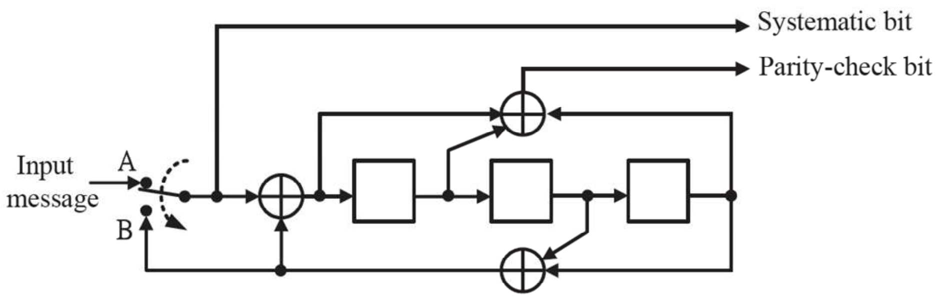


Figure 2.3: Trellis termination.

increases, better coding gain is expected. However, there is no significant performance improvement if a threshold of the iteration numbers has be reached.

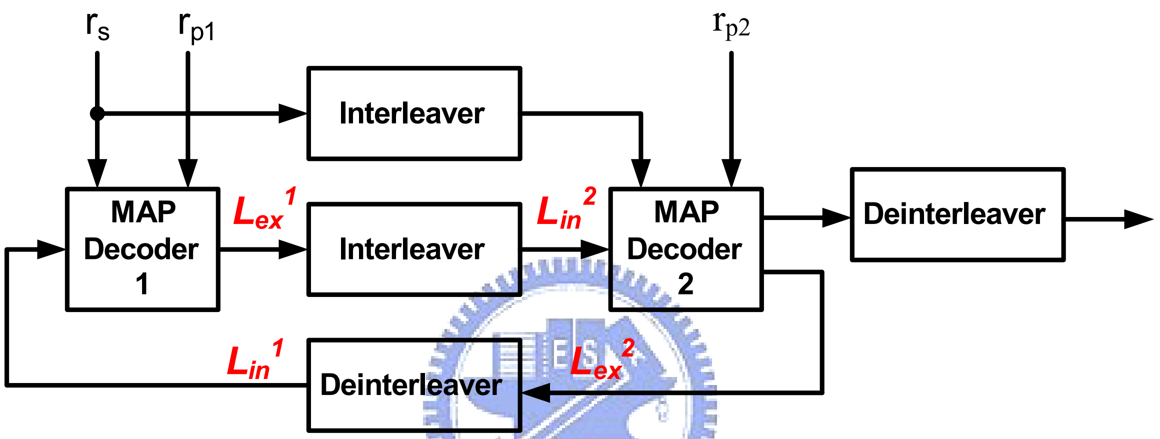


Figure 2.4: Conventional Turbo decoder.

2.1.4 Error floor effect

Although turbo coding provides an excellent performance, the bit-error-rate (BER) certainly decrease quite slowly at high signal-to-noise ratio (SNR). This phenomenon is due to relative small free distance of turbo codes, and is called an *error floor* [3]. Consider the relation of minimum free distance and the bit error probability in turbo coding, which can be expressed by

$$P_b \propto Q\left(\sqrt{2d_{free}R\frac{E_b}{N_0}}\right), \tag{2.2}$$

where d_{free} is the minimum free distance, R is the code rate, and E_b/N_0 is the SNR. At low SNR, the major part of errors can be corrected by iterative decoding since systematic information and parity information can be regarded as highly independent events. However, as the channel provides a reliable transmission, the dependency of the systematic and parity information grows up and the interleaver does little contribution on iterative decoding. To overcome this issue, the interleaver size should be enlarged to lower the error floor.

2.2 Decoding algorithm

The turbo code is composed of two SISO constituent codes that communicate iteratively through an interleaver. The maximum a posteriori probability (MAP) [4] algorithm and soft-output Viterbi algorithm (SOVA) [5] are commonly employed for the SISO decoders. Unlike the SOVA which exploits maximum likelihood (ML) algorithm to minimize the word error probability, the MAP algorithm minimizes the symbol (or bit) error probability. In this section, we will focus on introducing the turbo decoding based on MAP algorithm, because it has been proved that the MAP algorithm is the optimal decoding method for turbo codes while comparing with SOVA [6]. Furthermore, the Log MAP and Max-Log MAP algorithms will also be introduced, which reduce the hardware complexity and are widely used for implementation.

2.2.1 The MAP algorithm

The MAP decoding algorithm, termed as *BCJR algorithm*, is developed by Bahl, Cocke, Jelinek, and Raviv [4] in 1974. For each transmitted information bit u_t, the MAP algorithm estimates the *a posteriori* probabilities (APP) based on the received code sequence $\mathbf{r}$ over a discrete memoryless channel (DMC). It computes the log-likelihood ratio (LLR)

$$L(u_t) = \log \frac{P(u_t = +1|\mathbf{r})}{P(u_t = -1|\mathbf{r})}, \tag{2.3}$$

for $1 \leq t \leq N$, where N is the received sequence length, and compares this value to a zero threshold to determine the hard estimate u_t as

$$u_t = \begin{cases} +1, & \text{if } L(\hat{u}_t) \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.4}$$

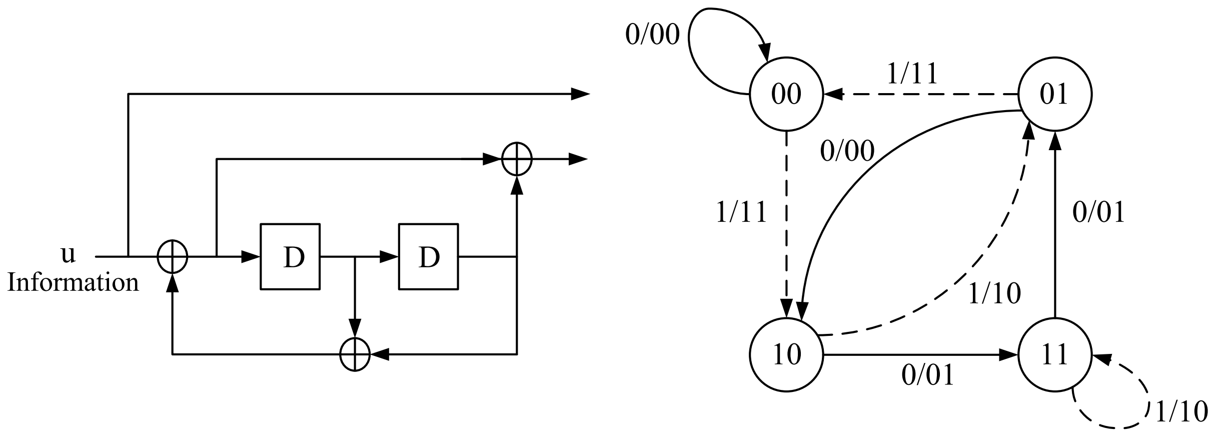


Figure 2.5: A (2,1,2) RSC encoder and its state transition diagram.

As an example, a rate 1/2 memory order 2 RSC encoder and its state transition are illustrated in Fig. 2.5. Note that the solid lines represent the state transitions corresponding to an information bit u_t of -1, while the dotted lines represent the state transitions corresponding to an information bit u_t of $+1$. Its decoding trellis diagram is shown in Fig. 2.6. In Fig. 2.6, the APP's in (2.3) can be computed by the summation of state transition probabilities. Therefore, the equation can be further expressed as

$$\begin{aligned} L(\hat{u}_t) &= \log \frac{P(u_t = +1|\mathbf{r})}{P(u_t = -1|\mathbf{r})} \\ &= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} P(S_{t-1} = m', S_t = m|\mathbf{r})}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} P(S_{t-1} = m', S_t = m|\mathbf{r})} \\ &= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} P(S_{t-1} = m', S_t = m, \mathbf{r})}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} P(S_{t-1} = m', S_t = m, \mathbf{r})}, \end{aligned} \tag{2.5}$$

where $P(S_{t-1} = m', S_t = m, \mathbf{r})$ represents the joint probability for the existing transition from S_{t-1} at time t to S_t at time $t+1$. $\mathbf{B}_t^{+1}$ and $\mathbf{B}_t^{-1}$ is the sets of (m', m), denoted the state transitions which are due to input bit $u_t = +1$ and $u_t = -1$ respectively.

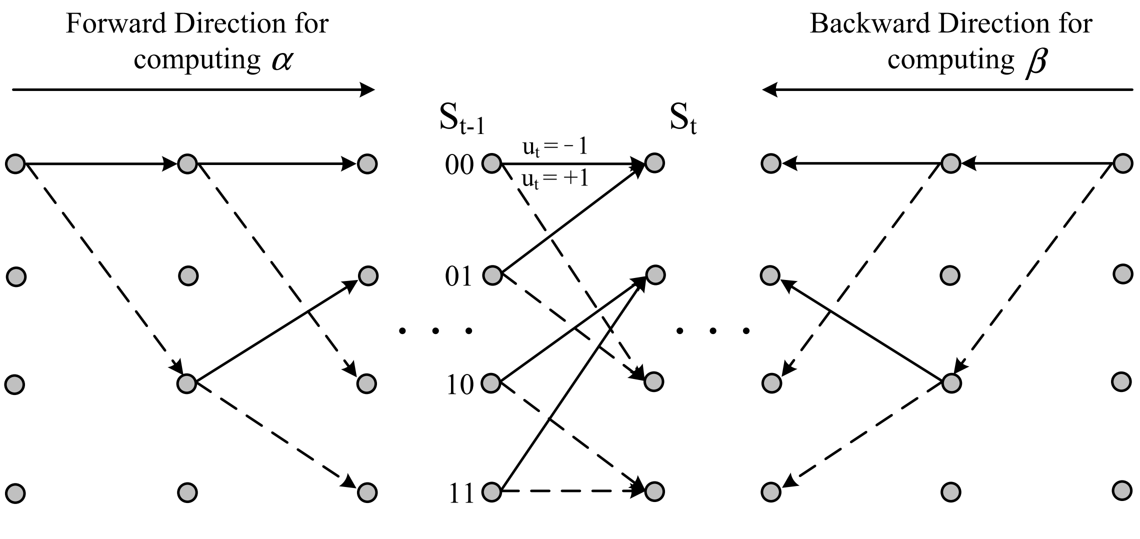
In order to compute joint probability required for calculation of $L(u_t)$ in (2.5), we define the following metrics:

$$\lambda_t(m', m) = P(S_{t-1} = m', S_t = m, \mathbf{r}) \tag{2.6}$$

$$\alpha_t(m) = P\{S_t = m, \mathbf{r}_0^t\} \tag{2.7}$$

$$\beta_t(m) = P\{\mathbf{r}_{t+1}^{N-1}|S_t = m\} \tag{2.8}$$

$$\gamma_t(m', m) = P\{S_t = m, r_t|S_{t-1} = m'\} \tag{2.9}$$



Forward Direction for computing α
Backward Direction for computing β
S_{t-1}
S_t
$u_t = -1$
$u_t = +1$
00
01
10
11

Figure 2.6: The decoding trellis diagram of the (2,1,2) RSC encoder.

Since we assume the code sequence after encoding is transmitted through discrete memoryless channel, the joint probability can be expressed as

$$\begin{aligned}\lambda_t(m', m) &= P(S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}, r_t, \mathbf{r}_{t+1}^{N-1}) \\ &= P(\mathbf{r}_{t+1}^{N-1} | S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}, r_t) \\ &\quad \cdot P(S_t = m, r_t | S_{t-1} = m', \mathbf{r}_0^{t-1}) \\ &\quad \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-1}) \\ &= P(\mathbf{r}_{t+1}^{N-1} | S_t = m) \cdot P(S_t = m, r_t | S_{t-1} = m') \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-1}).\end{aligned} \tag{2.10}$$

Here $\mathbf{r}_0^{t-1}$ represents the received code sequence from time instance 0 to $t-1$, while $\mathbf{r}_{t+1}^{N-1}$ is from time instance $t+1$ to the end of sequence. Note that the second equation of (2.10) comes from Bayes' rule, and the third equation is due to the Markov process in the state transitions. Therefore, compared with the definition of (2.7), (2.8) and (2.9), the joint probability defined in (2.6) can be rewritten as

$$\lambda_t(m', m) = \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m). \tag{2.11}$$

Now we will derive the equations (2.7), (2.8) and (2.9) as follow:

$$\begin{aligned}
\alpha_t(m) &= P(S_t = m, \mathbf{r}_0^{t-1}) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}) \\
&= \sum_{m' \in \mathbf{S}} P(S_t = m, r_{t-1} | S_{t-1} = m', \mathbf{r}_0^{t-2}) \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-2}) \\
&= \sum_{m' \in \mathbf{S}} P(S_t = m, r_{t-1} | S_{t-1} = m') \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-2}) \\
&= \sum_{m' \in \mathbf{S}} \alpha_{t-1}(m') \cdot \gamma_t(m', m).
\end{aligned} \tag{2.12}$$

Similarly, we have

$$\begin{aligned}
\beta_t(m) &= P(\mathbf{r}_{t+1}^{N-1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t+1} = m', \mathbf{r}_{t+1}^{N-1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t+1} = m', r_{t+1}, \mathbf{r}_{t+2}^{N-1}, S_t = m) \;/\; P(S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(\mathbf{r}_{t+2}^{N-1} | S_{t+1} = m', r_{t+1}, S_t = m) \cdot P(S_{t+1} = m', r_{t+1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(\mathbf{r}_{t+2}^{N-1} | S_{t+1} = m') \cdot P(S_{t+1} = m', r_{t+1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} \gamma_{t+1}(m, m') \cdot \beta_{t+1}(m'),
\end{aligned} \tag{2.13}$$

where $\mathbf{S}$ represent the set of all states. Note that the forward metric α in (2.12) and the backward metric β in (2.13) are computed recursively in opposite direction. If the trellis of encoding diverges from zero state at $t = 0$ and converges to zero state at $t = N - 1$, as shown in Fig. 2.6, the following initial conditions are satisfied:

$$\begin{aligned}
&\alpha_0(0) = 1, \quad \alpha_0(m) = 0 \quad \text{for } m \neq 0 \\
&\beta_N(0) = 1, \quad \beta_N(m) = 0 \quad \text{for } m \neq 0
\end{aligned} \tag{2.14}$$

Furthermore, for any existing transitions from state m' to m, the branch transition probability $\gamma_t(m',m)$ can be decomposed as

$$\begin{aligned}\gamma_t(m',m) &= P(S_t = m, r_t | S_{t-1} = m') \\ &= \frac{P(S_{t-1} = m', S_t = m, r_t)}{P(S_{t-1} = m')} \\ &= \frac{P(S_{t-1} = m', S_t = m)}{P(S_{t-1} = m')} \cdot \frac{P(S_{t-1} = m', S_t = m, r_t)}{P(S_{t-1} = m', S_t = m)} \\ &= P(S_t = m | S_{t-1} = m') \cdot P(r_t | S_{t-1} = m', S_t = m) \\ &= P(u_t) \cdot P(r_t | \mathbf{v_t}),\end{aligned} \tag{2.15}$$

where $P(u_k)$ is well-known as a prior probability of u_k and $\mathbf{v_t}$ is the codeword associated with the transition $S_{t-1} = m'$ to $S_t = m$ corresponding to encoder input u_t.

As a summary of the MAP algorithm, with computation of $\gamma_t(m',m)$ in (2.15), we can derive α and β for each state at different time instances. As a result, the joint probability in (2.11) is also available for $t = 0, 1, \cdots, N-1$. The log-likelihood ratio $L(u_t)$ can be calculated by

$$L(u_t) = \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \alpha_{t-1}(m') \cdot \gamma_t(m',m) \cdot \beta_t(m)}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \alpha_{t-1}(m') \cdot \gamma_t(m',m) \cdot \beta_t(m)}. \tag{2.16}$$

2.2.2 The Log-MAP algorithm

The MAP algorithm requires large memory and a large number of operations involving exponentiations and multiplications. The hardware realization of MAP decoder will be quite complex and difficult. Therefore, the Log-MAP algorithm is proposed to solve this problem. First, we transfer the branch metrics defined in the MAP algorithm to the logarithmic domain; that is

$$\bar{\gamma}_t(m',m) = \log \gamma_t(m',m). \tag{2.17}$$

Referring to (2.12) and (2.13), the forward path metric $\bar{\alpha}_t$ can be expressed as

$$\begin{aligned}\bar{\alpha}_t(m) &= \log \alpha_t(m) \\ &= \log \sum_{m'\in \mathbf{S}} e^{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m)},\end{aligned} \tag{2.18}$$

and the backward path metric $\bar{\beta}_t$ can be expressed as

$$\begin{aligned}\bar{\beta}_t(m) &= \log \beta_t(m) \\ &= \log \sum_{m'\in \mathbf{S}} e^{\bar{\gamma}_{t+1}(m,m') + \bar{\beta}_{t+1}(m')}.\end{aligned} \tag{2.19}$$

Note that the initial conditions of path metrics also have changed, since all computations work with the logarithm domain.

$$\begin{aligned} &\bar{\alpha}_0(0) = 0, \quad \bar{\alpha}_0(m) = -\infty \quad \text{for } m \neq 0 \\ &\bar{\beta}_N(0) = 0, \quad \bar{\beta}_N(m) = -\infty \quad \text{for } m \neq 0 \end{aligned} \tag{2.20}$$

After substituting (2.17), (2.18) and (2.19), the APP information $L(\hat{u}_t)$ in (2.16) can be rewritten as

$$L(u_t) = \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}}. \tag{2.21}$$

Considering the following Jocobian algorithm [7]

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|e^{\delta_2} - e^{\delta_1}|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|), \end{aligned} \tag{2.22}$$

where $f_c(\cdot)$ is a compensation function and thus the performance can be improved. By a recursive procedure of (2.22), the expression $\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n})$ can be computed exactly, as follows

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) &= \log(\Delta + e^{\delta_n}), \quad \Delta = e^{\delta_1} + \cdots + e^{\delta_{n-1}} = e^{\delta} \\ &= \max(\log \Delta, \delta_n) + f_c(|\log \Delta - \delta_n|) \\ &= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|). \end{aligned} \tag{2.23}$$

Now we can use (2.22) to represent forward metrics in (2.18) and backward metrics in (2.19) as

$$\bar{\alpha}_t(m) = \max_{m' \in \mathbf{S}}{}^* \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m', m)\}, \tag{2.24}$$

and

$$\bar{\beta}_t(m) = \max_{m' \in \mathbf{S}}{}^* \{\bar{\gamma}_{t+1}(m, m') + \bar{\beta}_{t+1}(m')\}, \tag{2.25}$$

where the $\max^*(\cdot)$ operation is defined as

$$\max{}^*(\cdot) = \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|). \tag{2.26}$$

Therefore, the (2.29) can be expressed as

$$\begin{aligned} L(\hat{u}_t) = &\max_{(m',m)\in \mathbf{B}_t^{+1}}{}^* \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m', m) + \bar{\beta}_t(m)\} \\ &- \max_{(m',m)\in \mathbf{B}_t^{-1}}{}^* \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m', m) + \bar{\beta}_t(m)\}. \end{aligned} \tag{2.27}$$

The performance of the Log-MAP algorithm is equivalent to the performance of the MAP algorithm but the complexity has been reduced considerately. However, some difficulty for hardware implementation still exists since computing $f_c(\cdot)$ also involves exponentiations and multiplications. For simplified the computation of correction function, it is usually stored in a pre-computed table. This table is only one dimensional due to the correction only depends on $|\delta_2 - \delta_1|$. Thus, the Log-MAP algorithm can be implemented with max function as well as a lookup table.

2.2.3 The Max-Log-MAP algorithm

In order to further simplify the complexity, another approximation of MAP algorithm termed Max-Log-MAP algorithm is derived. Considering the following approximation formula **max** function

$$\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) \approx \max_{i \in \{1,2,\cdot,n\}} \delta_i. \tag{2.28}$$

Note that the term $f_c(\cdot)$ is ignored in comparison with (2.23). Then we can simplify the equation (2.21) as follows:

$$\begin{aligned} L(u_t) = &\max_{(m',m)\in \mathbf{B}_t^{+1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \\ &- \max_{(m',m)\in \mathbf{B}_t^{-1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\}. \end{aligned} \tag{2.29}$$

Similarly, the forward recursive and backward recursive metrics in (2.18) and (2.19) can be individually expressed as

$$\bar{\alpha}_t(m) = \max_{m' \in \mathbf{S}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m)\}, \tag{2.30}$$

and

$$\bar{\beta}_t(m) = \max_{m' \in \mathbf{S}} \{\bar{\gamma}_{t+1}(m,m') + \bar{\beta}_{t+1}(m')\}. \tag{2.31}$$

The computations of $\bar{\alpha}$ and $\bar{\beta}$ are reduced to simple add-compare-select (ACS) operations, which are equivalent to the path metric updating of Viterbi algorithm. Therefore, compared with the MAP algorithm, the Max-Log-MAP algorithm utilizes additions to replace the multiplications and avoids the complicated exponentiations. However, the performance would degrade because of the information loss in (2.28).

2.3 Sliding window approach

In the conventional MAP-series decoding algorithm(including MAP algorithm, Max-Log MAP algorithm and Log-MAP algorithm), the LLR computation requires the path metric values generated by the forward and backward processes. Furthermore, since the backward recursive computation initials from the end of decoding trellis, as shown in Fig. 2.6, the decoding process can be started after the entire block message to be received. If the sequence length is large, it will lead to long output latency and huge memory requirement for hardware implementation. For example, the maximum block length of 3GPP standard is 5114, which means 5114 LLR values and path metrics should be stored. It is the main disadvantage of turbo code for real applications.

The main problem is that long block length can not be divided into several short sub-block immediately, since the unknown initial condition of backward recursive metrics computations will damage the performance of turbo codes. Therefore, the sliding window approach was proposed [8] to overcome it. This algorithm utilizes the fact that the backward metrics can be highly reliable even without the initial condition if the backward recursion goes long enough. Fig. 2.7 shows the process of the sliding window algorithm and will be further illustrated as follows. First, the received codeword sequence is divided

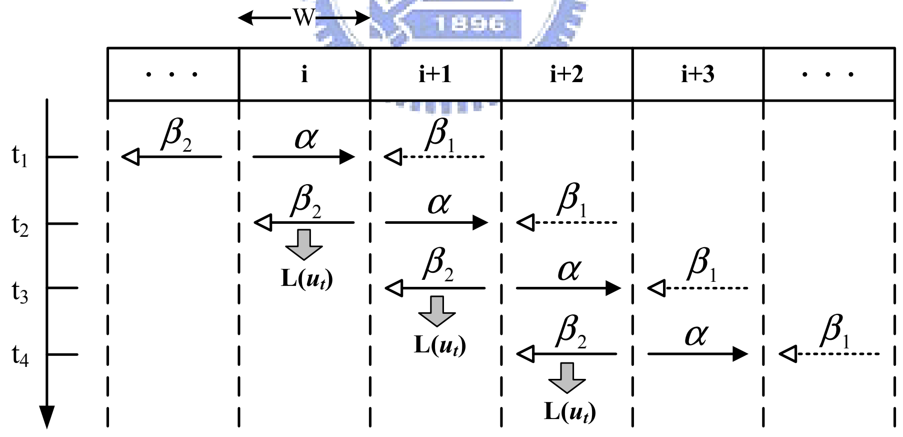


Figure 2.7: The process diagram of sliding window algorithm.

into several sub-blocks of length of W. W is called the convergence length, which normally is set to be five times the constraint length of component encoder in turbo code to ensure the reliable initialization. In the sliding window approach, the end of sub-block is the initial of next sub-block whether the forward or backward recursive operation. Thus, the initial metric values are inherited from the last metrics calculated in the previous sub-block. Note that the dummy backward recursion β_1 is employed to establish the initial condition for the true backward recursion β_2. Although the initial condition for the β_1 is unknown except the last sub-block, we utilize the equally likely condition for the β_1 values at time instance $(i+1)\cdot W$:

$$\beta_1(m) = \frac{1}{M}, \quad \text{for all } m \in \mathbf{S} \tag{2.32}$$

where $\mathbf{S}$ represents all possible states and M is equal to the total state number. During the forward recursion α proceeds in the i-th sub-block and stores these values into memory, the dummy backward recursion β_1 is performed in the $i+1$ sub-block concurrently. As soon as the β_1 computation is finished, the initial metrics in the i-th sub-block are available for the β_2 recursion. And $L(\hat{u}_t)$ can be calculated based on the α metrics in the memory, the β_2 metrics in computation, and the corresponding branches metrics in the i-th sub-block.

Chapter 3

Turbo Code in 3GPP-LTE system

The 3rd Generation Partnership Project (3GPP) is a collaboration between groups of telecommunications associations, to make a globally applicable third generation (3G) mobile phone system specification within the scope of the International Mobile Telecommunications 2000 project of the International Telecommunication Union (ITU). 3GPP specifications are based on evolved Global System for Mobile Communications (GSM) specifications. The project was established in December 1998.

3GPP-LTE (Long Term Evolution) is the name given to a project within the 3GPP to improve the Universal Mobile Telecommunications System (UMTS) mobile phone standard to cope with future technology evolutions. Furthermore, the channel coding Turbo Code with an interesting Quadratic Permutation Polynomial (QPP) interleaver is applied in 3GPP-LTE system [9]. In this chapter, the 3GPP-LTE encoder, interleaver and decoding algorithm used in our design will be introduced in detail. The simulation result will also be shown later.

3.1 Encoding procedure

In 3GPP-LTE specification, the scheme of turbo encoder is a Parallel Concatenated Convolutional Code (PCCC) with 8-state constituent encoders and one internal interleaver. The coding rate of turbo encoder is 1/3. The structure of turbo encoder is illustrated in Fig. 3.1

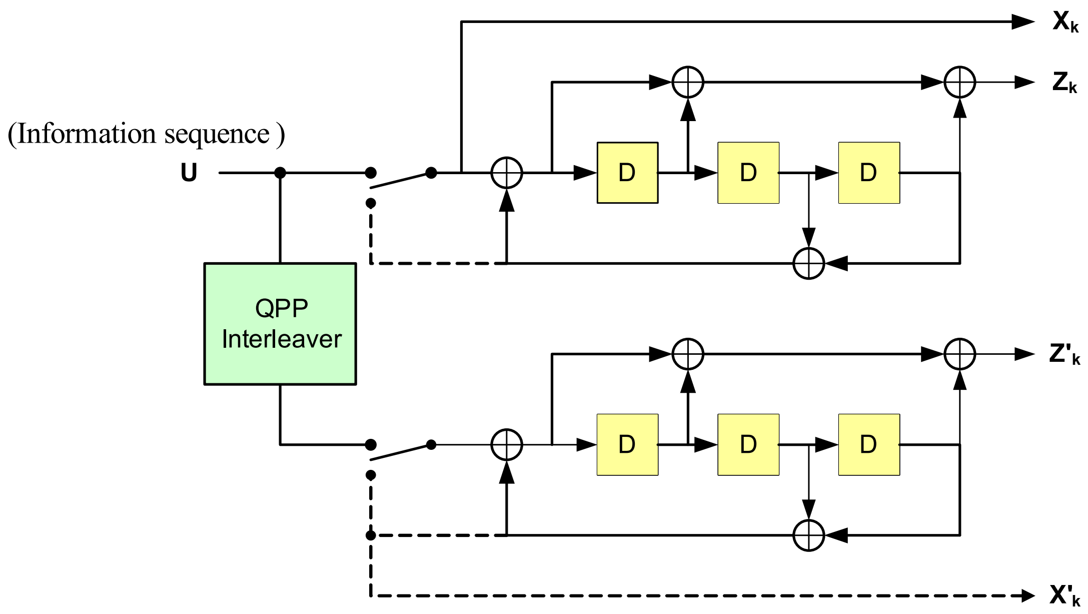


Figure 3.1: Turbo encoder for 3GPP-LTE.

The transfer function of the 8-state constituent code for the PCCC is

$$G(D) = \left[1, \frac{1 + D + D^3}{1 + D^2 + D^3}\right]. \tag{3.1}$$

The bit sequence input for a given code block to channel coding by c_0, c_1, c_2, c_3, ..., c_{K-1}, where K is the number of bits to encode. After encoding, the bits are denoted by $d_0^{(i)}$, $d_1^{(i)}$, $d_2^{(i)}$, $d_3^{(i)}$, ..., $d_{D-1}^{(i)}$, where D is the number of encoded bits per output stream and i indexes the encoder output stream.

The initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits. The output from the turbo encoder is $d_k^{(0)} = x_k$, $d_k^{(1)} = z_k$, $d_k^{(2)} = z_k^{'}$ for $k = 0, 1, 2, ..., K-1$. The bits input to the turbo encoder are denoted by c_0, c_1, c_2, c_3, ..., c_{K-1}, and the bits output from the first and second 8-state constituent encoders are denoted by z_0, z_1, z_2, z_3,..., z_{K-1} and $z_0^{'}$, $z_1^{'}$, $z_2^{'}$, $z_3^{'}$, ..., $z_{K-1}^{'}$, respectively. The bits output from the turbo code internal interleaver are denoted by $c_0^{'}$, $c_1^{'}$, $c_2^{'}$, $c_3^{'}$, ..., $c_{K-1}^{'}$, and these bits are to be the input to the second encoder.

Moreover, trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits. The transmitted bits for trellis termination shall be:

$$d_K^{(0)} = x_K, d_{K+1}^{(0)} = z_{K+1}, d_{K+2}^{(0)} = x_K^{'}, d_{K+3}^{(0)} = z_{K+1}^{'}$$
$$d_K^{(1)} = z_K, d_{K+1}^{(1)} = x_{K+2}, d_{K+2}^{(1)} = z_K^{'}, d_{K+3}^{(1)} = x_{K+2}^{'}$$
$$d_K^{(2)} = x_{K+1}, d_{K+1}^{(2)} = z_{K+2}, d_{K+2}^{(2)} = x_{K+1}^{'}, d_{K+3}^{(2)} = z_{K+2}^{'}$$

3.2 QPP interleaver

Interleavers for turbo codes have been extensively investigated. In particular, quadratic polynomials were emphasized. The algebraic approach was shown to admit analytical design of an interleaver matched to the constituent codes. The interleaver used in 3GPP-LTE standard is quadratic permutation polynomial (QPP) interleaver [2] [10]. Moreover, the performance using QPP interleaver was shown to be better than S-random interleavers [11] for short-to-medium block lengths. The formula of this interleaver is

$$\pi(x) = f_1 x + f_2 x^2 (\text{mod} N), \tag{3.2}$$

where N is the block length and the f_1, f_2, are the non-negative integers of the QPP interleaver of different block length. The interleaver parameter f_1, f_2, will be shown in detail in Appendix A. Besides, x is the original address and $\pi(x)$ is the address after interleaving. The algebraic construction can make a contention-free condition.

In recent years, the parallel processing of iterative decoding of turbo codes is of interest for high-speed decoders. Contention-free interleavers have recently been shown to be suitable for parallel decoding of turbo codes. Furthermore, in [12] it was shown that all quadratic permutation polynomials generate maximum contention-free interleavers, i.e., every factor of the interleaver length becomes a possible degree of parallel processing of the decoder. Thus, this class of interleaver is very interesting from an implementation point of view. That is, when multiple MAP decoders try to access multiple memories concurrently, there are no hazards happened.

3.2.1 Contention-free interleaver

To increase throughput, a MAP decoder is parallelized by dividing a size-N trellis into M size-W windows (N=MW) and employing M synchronous MAP-based decoders with M separate memory banks. Interleavering latency is eliminated by writing the M values generated each clock cycle directly to their interleaved position. However, if the interleaver is not designed well, two or more MAP-based decoders may require access to the same memory bank on a given clock cycle, resulting in a memory contention.

Several interleaving algorithms with contention-free properties [13] have been published, and the QPP interleaver has the contention-free property. Fig.3.2 shows an example of memory contention problem in a parallel decoding structure. A codeword sequence is stored in order in four different memory banks. It is obvious that it is a contention-free access at all different timing with pre-permutation order. But it will have the memory contention problem if we apply post-permutations. The post-permutation 1 is a contention-free interleaver design. Because every time we access four symbols, they come from different memory banks. The interleaver design of post-permutation 2 suffers two access collisions at time T_0 and T_2. Due to the good property mentioned above, the QPP interleaver is suitable for parallel decoding of turbo code.

3.2.2 Performance comparison

The design of the interleaver is critical to the performance of the Turbo Code. Interleavers can in general be separated into two classes: random interleavers and deterministic interleavers. The basic random interleavers permute the information bits in a pseudorandom manner. It was demonstrated in [14] that near Shannon limit performance can be achieved with these interleavers for large frame sizes. The S-random interleaver proposed in [15] is an improvement to the random interleaver. The S-random interleaver is a pseudo-random interleaver with the restriction that any two input positions within distance S cannot be permuted to two output positions within distance S.

QPP interleaver is a class of deterministic interleavers based on a quadratic congruence in (3.2). The interleaver depends on permutation polynomials over the ring of integers modulo N. By carefully selecting the coefficients of the polynomial, we can achieve a performance close to, or in some case, even better than S-random interleavers. The performance result is illustrated in Fig. 3.3.

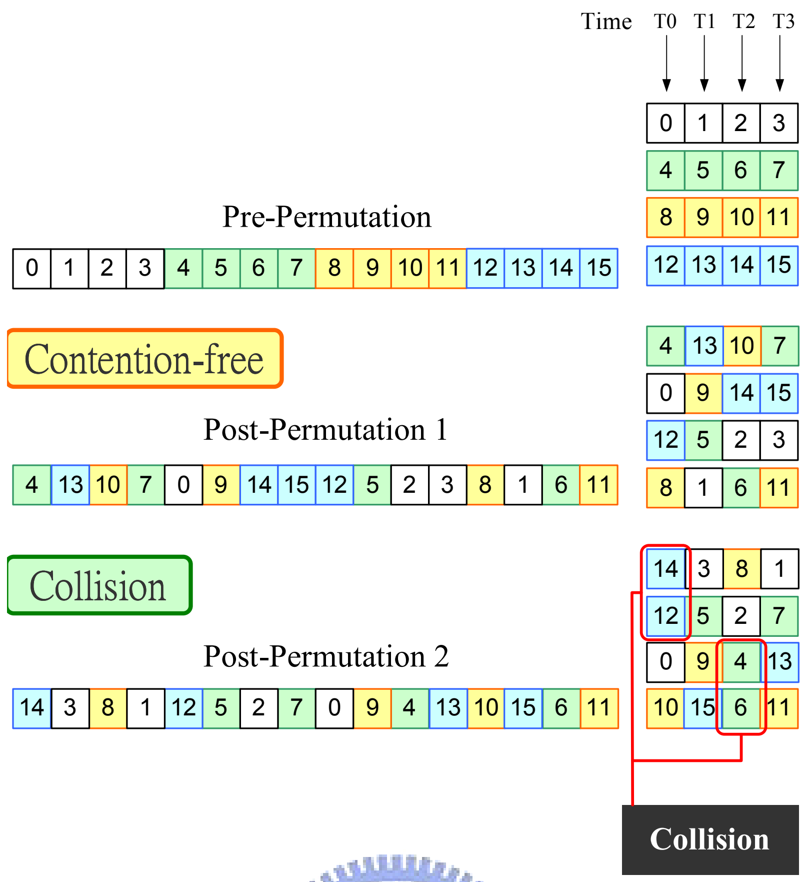


Figure 3.2: Example of a contention-free permutation.

3.3 Decoding procedure

According to the iterative decoding algorithm of turbo codes in section 2.2, we realize that the goal of the MAP algorithm is to derive the LLR and extrinsic values. Therefore, for the input signal u_t, the LLR can be represented as

$$L(\hat{u}_t) = \log \frac{P(u_t = +1|\mathbf{r})}{P(u_t = -1|\mathbf{r})}, \tag{3.3}$$

where u_t is defined as the collection of input symbols $(u_{0,t}, u_{1,t})$ from time $(t-1)$ to time t, and $\mathbf{r}$ is the received symbol after BPSK mapping. The decomposition of the equation

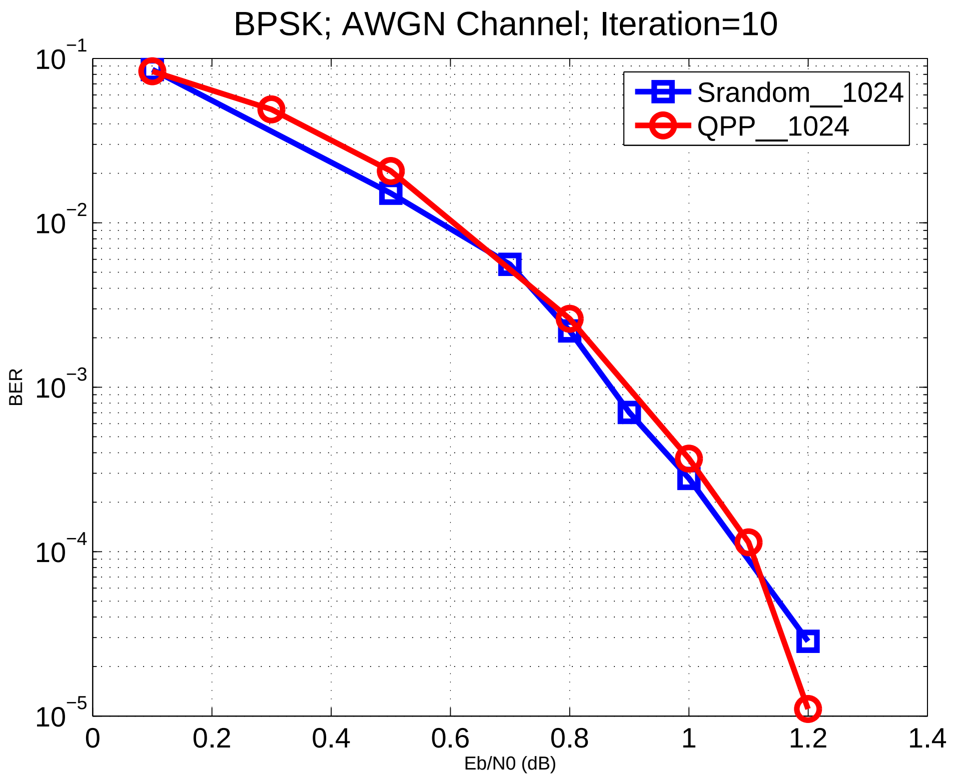


Figure 3.3: Performance comparison.

(3.3) will be

$$
\begin{aligned}
L(\hat{u}_t) &= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} P(S_{t-1}=m', S_t=m|\mathbf{r})}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} P(S_{t-1}=m', S_t=m|\mathbf{r})} \\
&= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} P(S_{t-1}=m', S_t=m, \mathbf{r})}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} P(S_{t-1}=m', S_t=m, \mathbf{r})} \\
&= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \alpha_{t-1}(m')\cdot\gamma_t(m',m)\cdot\beta_t(m)}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \alpha_{t-1}(m')\cdot\gamma_t(m',m)\cdot\beta_t(m)},
\end{aligned} \tag{3.4}
$$

where $\mathbf{B}_t^{+1}$ is the set of (m',m) that indicate the state transitions which are caused by $u_t = +1$, and $\mathbf{B}_t^{-1}$, the set of (m',m), denoted the state transitions are due to $u_t = -1$.

Applying the Log-MAP algorithm to the (3.4), the LLR can be rewritten to

$$
\begin{aligned}
L(\hat{u}_t) &= \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}} \\
&= \max_{(m',m)\in \mathbf{B}_t^{+1}}{}^{*}\{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)\} \\
&\quad - \max_{(m',m)\in \mathbf{B}_t^{-1}}{}^{*}\{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)\},
\end{aligned} \tag{3.5}
$$

and the Max-Log MAP approximation will become

$$\begin{aligned} L(\hat{u}_t) = &\max_{(m',m)\in \mathbf{B}_t^{+1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \\ &- \max_{(m',m)\in \mathbf{B}_t^{-1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\}. \end{aligned} \tag{3.6}$$

Furthermore, the initial condition of branch metrics become

$$\begin{aligned} \bar{\alpha}_0(0) = 0, \quad \bar{\alpha}_0(m) = -\infty \quad \text{for } m \neq 0 \\ \bar{\beta}_N(0) = 0, \quad \bar{\beta}_N(m) = -\infty \quad \text{for } m \neq 0 \end{aligned} \tag{3.7}$$

For a rate $1/n$ RSC encoder, each codeword frame consists of one systematic bit and $(n-1)$ parity bits. In the receiver, the received codeword has the systematic symbol $r_t^{(0)}$ abd parity symbols $r_t^{(1)} \sim r_t^{(n-1)}$. Moreover, in order to reduce the computational complexity, we could further simplify the path metrics into

$$\bar{\gamma}_t(m',m) = \frac{1}{2}(u_t L_a(u_t) + L_c u_t r_t^{(0)} + \sum_{i=1}^{n-1} L_c r_t^{(i)} \mathbf{v_t}^{(i)}), \tag{3.8}$$

where the value of $\mathbf{v_t}^{(i)} \in \{+1, -1\}$ depends on the encoding generator matrix after BPSK mapping. L_c is the channel reliability value for the AWGN channel. The *a priori* information in (3.8) is represented by

$$L_a(u_t) \triangleq \log \frac{P(u_t = +1)}{P(u_t = -1)}. \tag{3.9}$$

From the decoding flow shown in Fig. 2.4, the extrinsic information corresponding to the information bit u_t for the next stage can be calculated as

$$L_e(u_t) = L(u_t) - L_c\, r_{0,t} - L_a(u_t). \tag{3.10}$$

Computer symbol probabilities for the next decoder from previous decoder as

$$L_a(u_t) = L_e(\tilde{u}_t) = \log \frac{P(u_t = +1)}{P(u_t = -1)}. \tag{3.11}$$

Assume the information symbols are equal probability, so the *a priori* information can be initialized for the first iteration:

$$\begin{cases} \log P(u_t = +1) = 0 \\ \log P(u_t = -1) = 0 \end{cases} \tag{3.12}$$

The turbo decoding proceeds iteratively with the extrinsic information passing between the two SISO decoders. When the stopping criteria is reached, which may be the maximum iteration number or a correctly decoded codeword, the final decisions after de-BPSK are

$$u_t = \begin{cases} 0, & \text{if } L(u_t) \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{3.13}$$

3.4 Design parameter analysis

In this section, we will present the simulation results and some parameters setting for implementation. All the simulation results are signal-to-noise (SNR) versus BER under BPSK modulation and AWGN channel.

In order to determine appropriate design parameters such as the bit widths of the input symbol, the branch metric, and the path metric, the performance evaluation through simulations are necessary. The iteration number and the sliding window size will directly influence not only the performance of turbo coding but also the hardware cost of the design. The BER curves of the floating point decoders under BPSK modulation and AWGN channel with block length of 2048 are presented in Fig. 3.4. In Fig. 3.4, there is a 0.05dB loss between the sliding window size of 16 and 32 at the BER=10^{-5} under the fixed 8 iterations. Therefore, we choose the window size 16 in our design.

However, the Max-Log MAP decoding algorithm can reduce the decoding complexity, it causes the performance loss due to the approximation of max function. The scaling factor approach [16] is applied as compensation for the performance loss due to the Max-Log-MAP algorithm. That is, a scaling factor to scale down the extrinsic information is introduced. Therefore, the intrinsic information $L_a(u_t)$ can be formulated as :

$$L_a(u_t) = \beta \times L_e(\tilde{u}_t), \tag{3.14}$$

where β is the scaling factor. We choose the scaling factor β=0.75 in our design for performance compensation. This approach not only improves the performance but also cost a little gate counts.

The fixed point representation of the internal variable in the SISO decoder is determined from the received symbol quantization. Fig. 3.5 shows the simulation result with different input symbol quantization under BPSK modulation and AWGN channel, and we

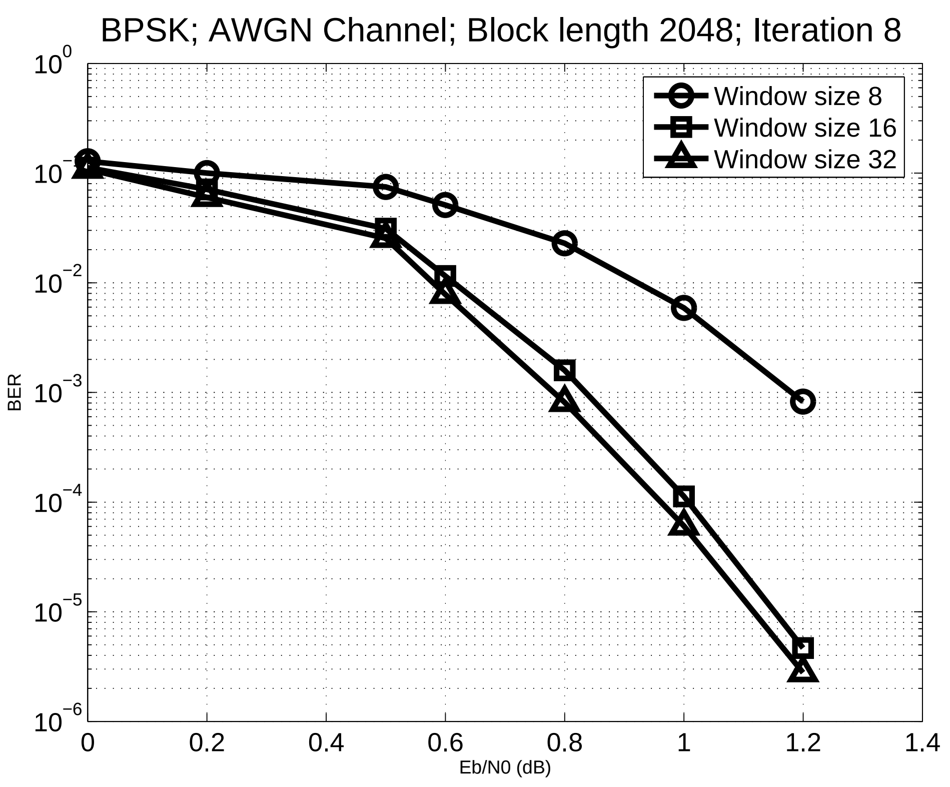


Figure 3.4: Comparison of window size.

choose block length 2048, window size 16, and iteration number 8. Note that $a.b$ in the figure denotes the quantization scheme where a is the integer part, and b is the fractional part. Furthermore, the primary specifications of MAP decoder are given in Table 3.1, where the code polynomial is follow 3GPP-LTE system.

Table 3.1: MAP Decoder Specification

generator matrix	$[\ 1\ \ \frac{1+D+D^3}{1+D^2+D^3}\ ]$
code rate	1/3
sliding window size	16

We can observe that the quantized format 6 bits (3.3) is suitable scheme for our design. In addition, the width of extrinsic information, branch metric, and path metric can be derived and we summarize the fixed representations in Table 3.2.

In 3GPP-LTE specification, there are 188 different block lengths based on particular QPP parameters. The shortest size and the longest one are 40 and 6144, respectively. The

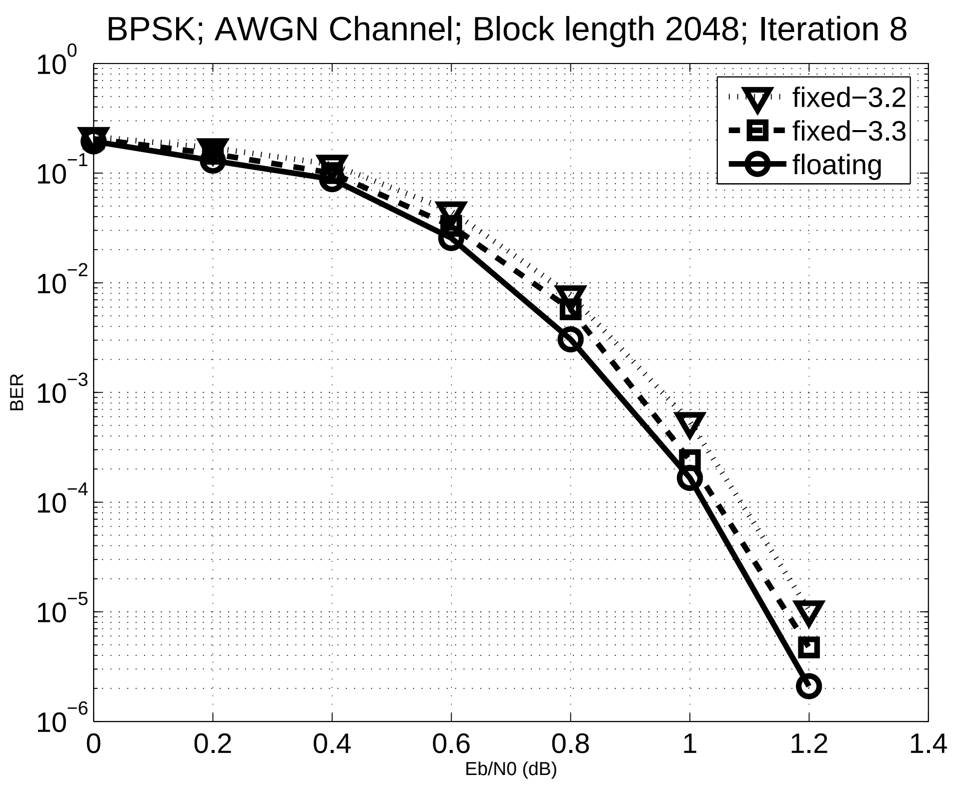


Figure 3.5: Fixed point comparison.

performance of different size is apparently distinct during simulation and is illustrated in Fig. 3.6. The parameters of simulation is under 8 iterations and sliding window size 16. In Fig. 3.6, we can see that the worst case (N=40) can achieve BER=10^{-5} at SNR about 4.5 $\sim$ 5.0 dB and the best one (N=6144) can reach the same BER under 1.0 dB.

Table 3.2: Summary of fixed representation in MAP decoder

quantities	Input symbols	Extrinsic information	Branch metrics	Path metrics	LLR values
width	6	6	9	9	10

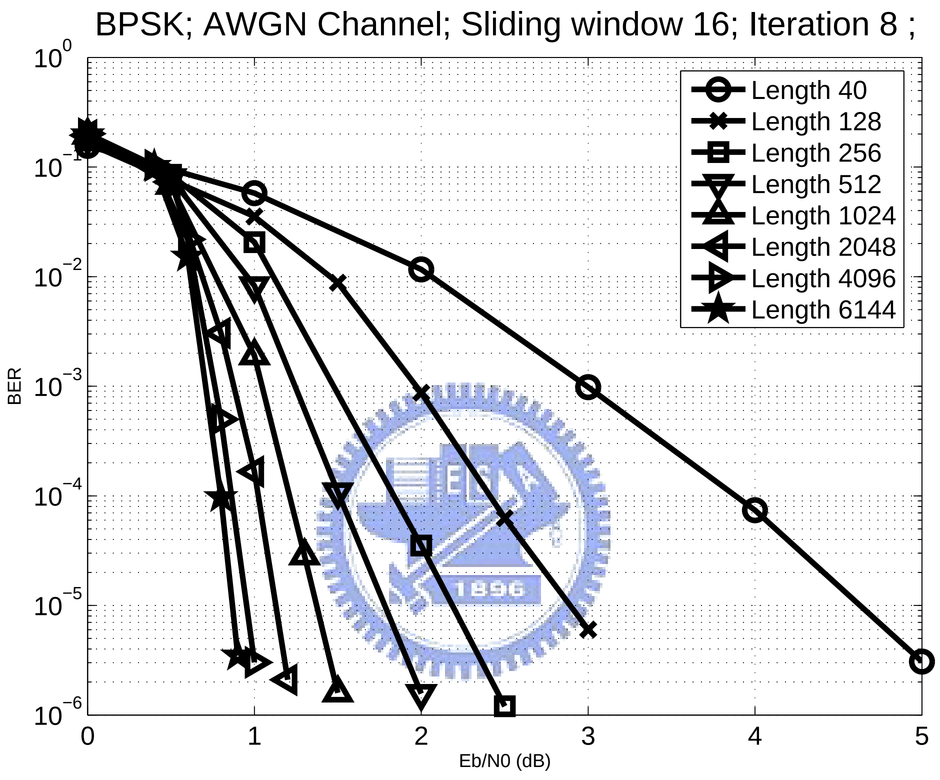


Figure 3.6: Performance of 3GPP-LTE turbo decoder.

Chapter 4

Reconfigurable 3GPP-LTE Turbo Decoder

The turbo codes have found applications in several standards listed in Table 4.1 due to its outstanding error correction ability. However, several communication application in the future may demand for a higher speed channel coding scheme. It may become a obstacle for turbo codes in hardware implementations.

To meet the future applications, we proposed a parallel architecture of turbo decoder for improving the throughput. In this chapter, not only the proposed turbo decoder but also the MAP decoder utilized in our architecture will be described in detail. Moreover, since the block length ranges from 40 to 6144, the memory requirement is also a great issue. A residue-only interleaver is adopted to reduce storage memory. Furthermore, reconfigurable property of our decoder designed for applying in various situation will also

Table 4.1: Standard specifications for turbo coding

Standard	**Application**	**Iterative Code**	**Max. Throughput**
3GPP UMTS	Wireless cellular	Parallel conc. of 8-state conv. codes	2 Mb/s
3GPP2 CDMA2000	Wireless cellular	Parallel conc. of 8-state conv. codes	3.09 Mb/s
IEEE 802.16e	Wireless networking (WiMAN)	Double binary conv. turbo code	30 Mb/s

be introduced later. Finally, how to compute the throughput in our parallel architecture will be interpreted clearly.

4.1 Architecture overview

According to the turbo decoding flow shown in Fig. 4.1, each iteration computes the extrinsic information which is an *a priori* value estimation for the next iteration. The next iteration remains idle until the interleaving step reorders all decoding results. The latency and the performance are determined by the iteration number and the block size. In the conventional turbo decoder, lower error rate can be achieved with large block size, but it also requires more decoding time. Hence, the application of turbo code provides a trade-off between performance and speed.

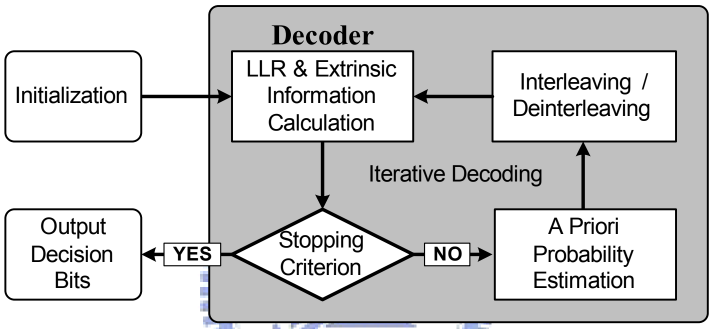

Decoder
Initialization
LLR & Extrinsic Information Calculation
Interleaving / Deinterleaving
Iterative Decoding
Output Decision Bits
YES
Stopping Criterion
NO
A Priori Probability Estimation

Figure 4.1: Iterative decoding flow of turbo decoder.

Fig. 4.2 shows the simple block diagram of proposed parallel decoder. There are 8 radix-2 MAP decoders and 8 memory banks connected with QPP network and parallel network [17]. Each memory stores the required data for decoding, including the received LLR values from channel and the extrinsic informations from MAP decoder. Each MAP decoder computes the a priori probability which is used in the next decoding round and makes decision for received symbols. The interleaver is implemented with the address generators inside each memory and the global network controller, then the pseudo random data will be fed into the MAP decoders. In addition, the decoder supports 188 kinds of block length, 40 $\sim$ 6144, and all the QPP interleaver parameters f_1 and f_2. Iteration numbers in this design also can be worked from 1 $\sim$ 8. The detail process and architecture is explained in the following subsection.

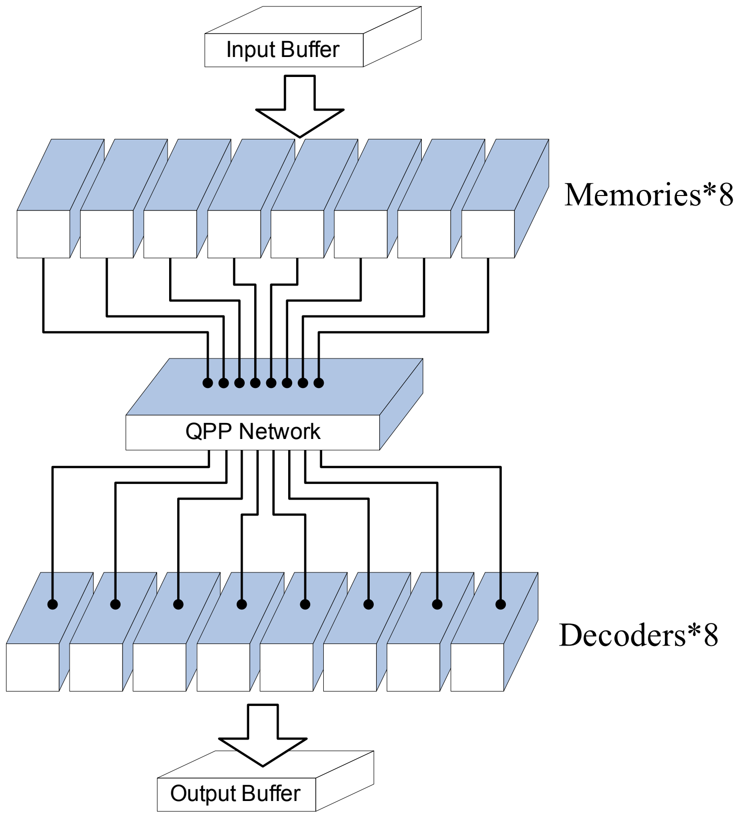


Figure 4.2: Block diagram of proposed turbo decoder.

4.2 MAP decoder

4.2.1 Decoding schedule

Fig. 4.3 presents the radix-2 MAP decoder, which consists of branch metric unit(BMU), add-compare-select (ACS) unit, log-likelihood-ratio (LLR) unit and input buffers (IBUF). The input buffers are used to store the input symbols. The BMUs calculate the branch metrics for α-ACS, β-ACS, and β_d-ACS units and each ACS unit according to the trellis structure performs addition, comparison, and selection. The α-ACS unit carries out the forward recursion and saves the result in the α-buffer. β-ACS unit begins the backward recursion from the initial conditions determined by the β_d-ACS unit calculation. Moreover, the α-buffer performs the Last-In/First-Out (LIFO) in order to reorder the α value for LLR calculation. The LLR unit determines the LLR values $L(u_t)$ used to make decision and the extrinsic informations $L_e(u_t)$ that store in the memory for the next decoding round [18].

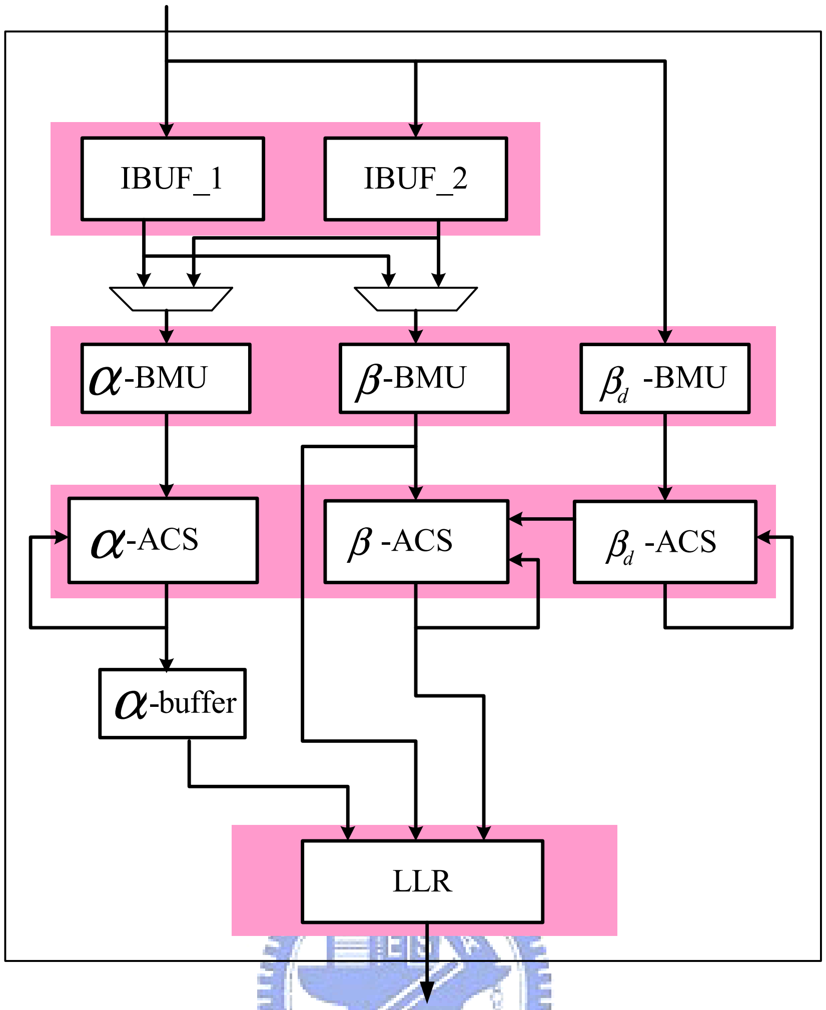


Figure 4.3: Block diagram of MAP decoder.

To consider the sliding window approach shown in Fig. 2.7, the backward metric β evaluation can be started until the required window of data have been stored. However, if we reverse the order of input sequence within a sub-block, the input buffer of the β_d computation can be saved [19]. Fig. 4.4 is the decoding schedule of the MAP decoder. In order to reduce the IBUF for β_d-ACS, the input sequence order of decoding is from the end of the sliding window to the beginning of the window. That is, the input sequences are written into the input buffer in the reversed order. In Fig. 4.4, the β_d computation can be executed immediately at the first time interval since the reversed sequence. Then the α recursion is performed on the previous stored data at the next time interval and the second window data will be written into the other input buffer. Finally, the first window data will be read at the third time interval for backward β calculation at the same and the third window data will be written into the input buffer simultaneously. When the first window β starts to calculate, the LLR computation also calculates in a neglected latency. As a result, the latency of MAP decoder is about two sliding window size.

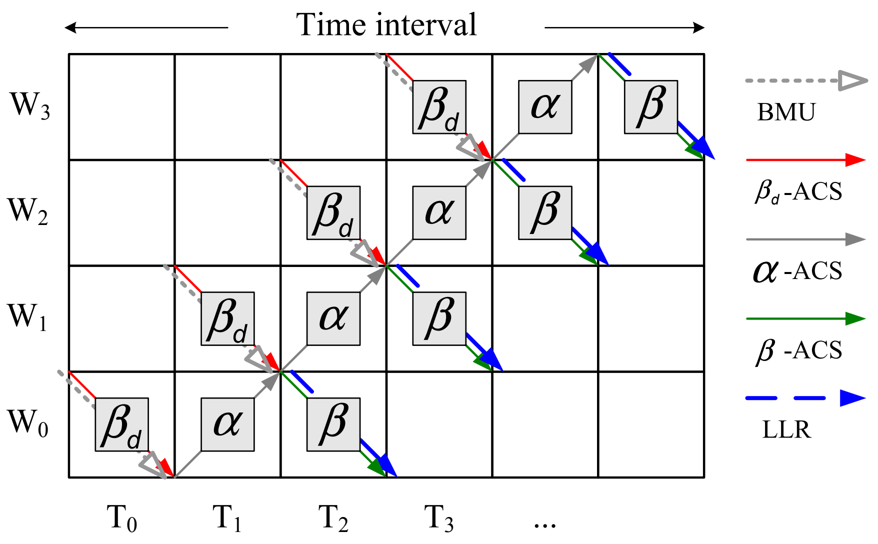


Figure 4.4: Decoding schedule of MAP decoder.

4.2.2 The circuit for LLR calculation

Our design adopts the modulo normalization to avoid overflow of path metric [20]. This methods requires only one more bit in the ACS unit and a simple modification inside the LLR unit. Only the differences between both forward path metric and backward path metric are significant in modulo normalization, so the LLR unit has to use these differences to calculate the log-likelihood value. We rearrange the computation order of log-likelihood value from circuit in Fig. 4.5(a) to Fig. 4.5(b). Although the two circuits have the same function, the original circuit may result in overflow due to the limited data width. The modified circuit could guarantee the correctness and cause no extra path delay.

4.3 Proposed Turbo decoder design

Fig. 4.6. shows the block diagram of proposed decoder, which consists of 8 parallel decoders and 8 memory sets. For an example of block length 1024, we separate a codeword into 8 sub-blocks with length 128. Each sub-block is assigned to one decoder and

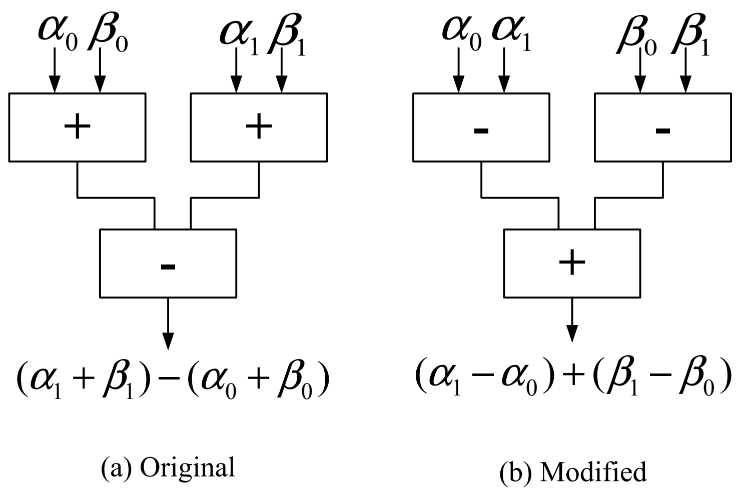


Figure 4.5: The circuit for LLR unit.

decoded separately. These sub-blocks are connected by a well-designed QPP interleaver. This method avoids the forward and backward recursion problem while using parallel architecture. The decoding process is described as follows: each memory will collect a 128-bits sub-block from input buffer until the whole 1024-bits codeword is received. The memory stores the received symbols and extrinsic information, and the 8 memories will deliver the required data to the 8 MAP decoders through QPP network. The interleaver is implemented with the address generators in each memory and the network controller. The MAP decoders perform the primary decoding procedures, and each one decodes the sub-blocks. After the required iterations, our design will output the decisions of current block and start to decode next block.

As described in the previous section, the quadratic permutation polynomial interleaver interchanges the information between each blocks to improve performance. In addition, the quadratic permutation polynomials generate the contention-free interleavers. Therefore, we can access the required data from different memory banks without memory contention and decoding several blocks in parallel to reduce the large decoding latency. Each MAP decoder is based on the Max-Log-MAP algorithm and adopted the radix-2 ACS structure. The memory blocks are used for data storage, which store the input information and extrinsic values generated by the MAP decoder. Note that the number of MAP decoders and memory banks are the same, which is because that each decoder can access data from memory in a one-to-one mapping condition at the same time. Eventually, the network is a permutation control unit which controls the interchange of both the input symbols and extrinsic informations. In the following subsection, we will also introduce the characteristic of our design and the throughput calculation in detail.

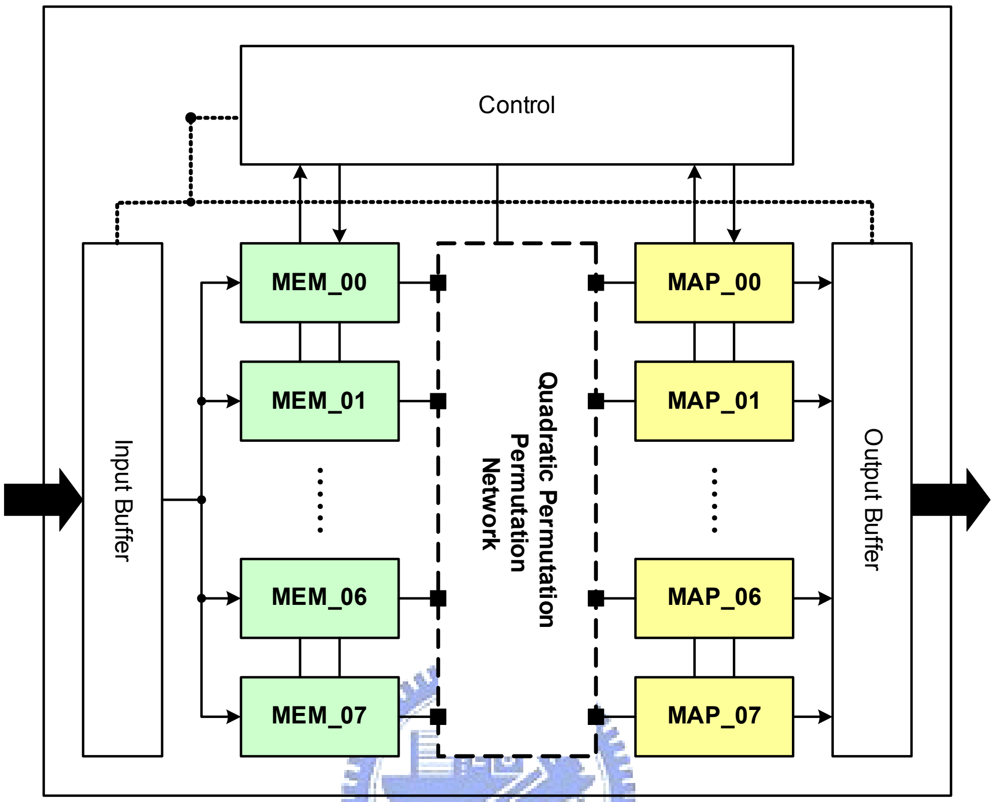


Figure 4.6: Proposed turbo decoder architecture.

4.3.1 Residue-only interleaver

From the QPP interleaver formula, $f(x) = f_1x + f_2x^2(modN)$, the algebraic construction makes a contention-free condition. A mathematical description of contention-free condition is now given. The exchange and processing of a sequence of $N = MW$ symbols between sub-blocks of the iterative decoder can be parallelized by M processors working on window sizes of length W in each sub-block provided that the following condition holds for the interleaver $f(x)$, $0 \leq x \leq N$:

$$\lfloor f(j + tW)/W \rfloor \neq \lfloor f(j + vW)/W \rfloor \,, \tag{4.1}$$

where $0 \leq j < W$, $0 \leq t < v < N/W$, and $\pi(\cdot)$ is $f(\cdot)$. First, it verifies the equation (4.1) for $f(x)$. Let

$$Q_t = \left\lfloor \frac{f(j+tW)}{W} \right\rfloor, Q_v = \left\lfloor \frac{f(j+vW)}{W} \right\rfloor, \tag{4.2}$$

then

$$\begin{aligned} f(j+tW) &= Q_t W + [f(j+tW)(\text{mod} W)] \\ f(j+vW) &= Q_v W + [f(j+vW)(\text{mod} W)]. \end{aligned} \tag{4.3}$$

It must show that $Q_t \neq Q_v$ for $t - v \neq 0(modM)$ and any $0 \leq j < W$. Assume $Q_t = Q_v$, then

$$Q_t - Q_v = \frac{f(j+tW) - [f(j+tW)(\text{mod} W)] - f(j+vW) - [f(j+vW)(\text{mod} W)]}{W} = 0. \tag{4.4}$$

Using the simple proposition below:

Proposition: Let M be an integer. Support that $M|N$ and that $x \equiv y$ (mod N). Then $x \equiv y$ (mod M).

Observing that

$$f(j+tW) \equiv f_1 j + f_2 j^2 (\text{mod} W), f(j+vW) \equiv f_1 j + f_2 j^2 (\text{mod} W). \tag{4.5}$$

It can conclude that

$$[f(j+tW)(\text{mod} W)] = [f(j+vW)(\text{mod} W)]. \tag{4.6}$$

Therefore, the absolute value of equation (4.4) can be simplified as

$$|Q_t - Q_v| = \frac{[f(j+tW)] - [f(j+vW)]}{W} = 0. \tag{4.7}$$

By noting that $(j+tW) \neq (j+vW)$ and that $f(x)$ is a permutation polynomial, concluding $f(j+tW) \neq f(j+vW)$ and having a contradiction in (4.7).

From the equation (4.5) and (4.6), we know that the residue part of the address calculation will be the same. That is, when using QPP interleaver in the parallel construction, the residue address of each memory bank is the same. It will be very helpful to our design. Because the longest block length is 6144 and multiple memory banks, the memory costs are too much. For this reason, we use the residue-only interleaver in our design, and the residue address can be repeatedly used in different memory banks. Only the residue address has to be saved instead of total address, leading to reduce the memory costs largely.

Fig. 4.7 shows a 4 MAP decoders and 4 memory banks example, we can see that each MAP decoder access data from the same address of each memory but different memory bank. The residue address will be the same after calculation in the right of Fig.4.7.

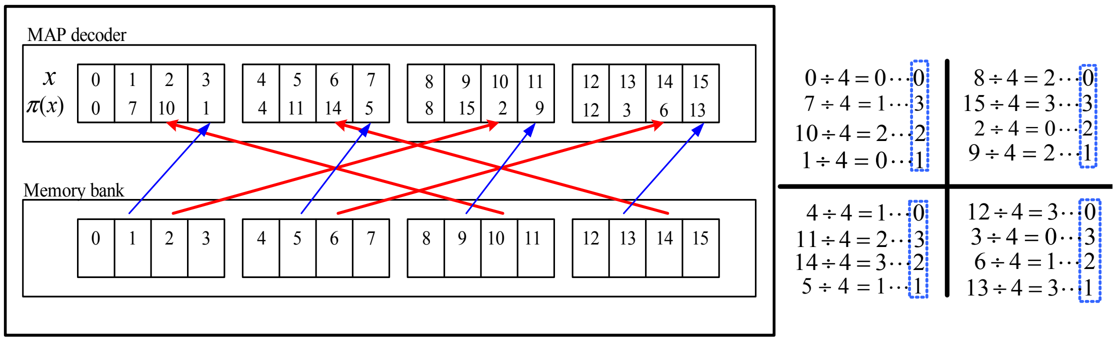


Figure 4.7: Example of residue-only interleaver.

4.3.2 Reconfigurable 1/2/4/8-MAP decoders

From the observation of the QPP parameter in Appendix A, the block length can be classified as following:

$$N = \begin{cases} 40 + 8a, 0 \leq a \leq 59 \\ 512 + 16b, 0 < b \leq 32 \\ 1024 + 32c, 0 < c \leq 32 \\ 2048 + 64d, 0 < d \leq 64 \end{cases} \tag{4.8}$$

There are all 188 modes, and each block size can be divided by 1, 2, 4, and 8 at most. In this opinion, we design a reconfigurable MAP decoder numbers in our architecture. Because of parallel decoding, a received codeword is partitioned into 8 memory banks. Therefore, we can decode the received codeword by utilizing 1-SISO, 2-SISO, 4-SISO or 8-SISO respectively, based on the block length, performance or throughput expected. It can be illustrated in Fig. 4.8.

In order to satisfy the 3GPP-LTE block length requirement, there are some compromises between performance and area cost for all block size. Fig. 4.9 shows the performance of the shortest size 40 with different MAP decoders. In Fig. 4.9, we can see that the performance curve of using 8-SISO decoders even 4-SISO or 2-SISO is worse than using 1-SISO. First, we would like to figure out how the performance degrades and then

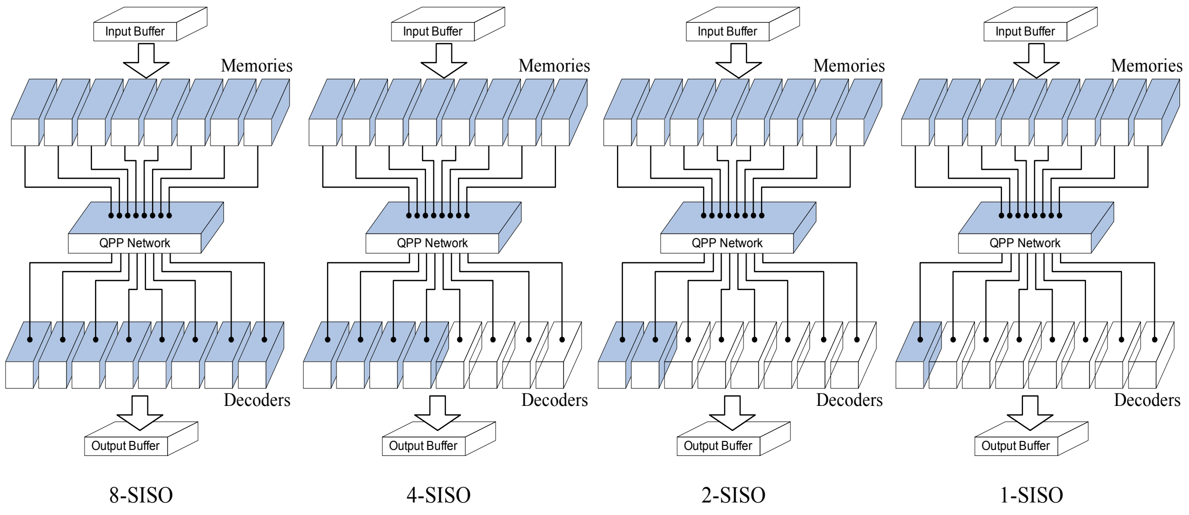


Figure 4.8: Decoding codeword with various processing elements.

we can find some approaches to improve it. The degradation of performance may be formed by two factor: the first one is that the small section size cause the shorter trellis structure. Therefore, the calculation of path metric α or β will be unreliable to make the final decision. The second is the boundary initial value. In general, we often set the initial value of each state to be zero in every boundary. Nevertheless, the performance loss can not be obviously found on decoding long block size with multiple SISO decoders. Because there are fewer such problems mentioned above for long block length.

The warm-up free method [21] is applied to improve performance degradation in our design. In this method, a warm-up free parallel architecture is proposed by using the α path metrics that were calculated in the previous iteration of the adjacent sub-block to initialize the path metrics for each sub-block in the next iteration. That is, the α path metrics of the last information bit in sub-block i were stored for initialization of the forward path metrics of the sub-block $(i + 1)$ in the next iteration. In the first iteration, the initial values of α path metrics needed in the next iteration are not available, thus the initial values will be set to zero.

After adoption of this method, the forward path metrics become more reliable. Therefore, the performance can be improve with multiple SISO decoders for short block length. Fig. 4.10 shows the final performance of our design of size 40. Comparing with Fig. 4.9, the performance of using multiple SISO decoders is near the performance of utilizing 1-SISO. Moreover, the performance of decoding long block size with different SISO decoder

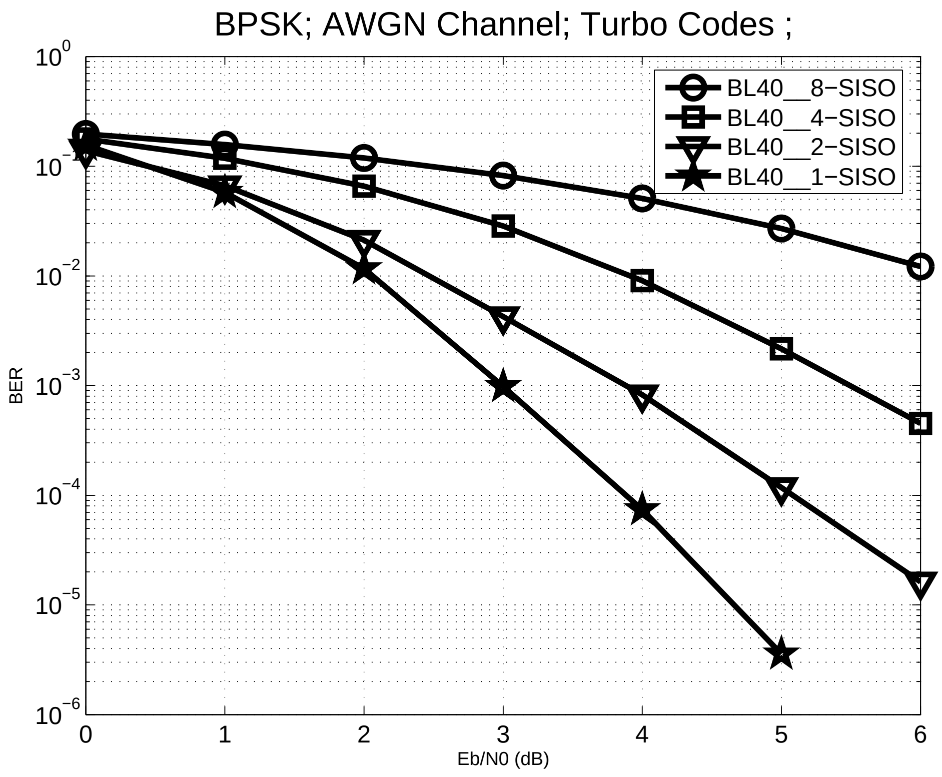


Figure 4.9: Short block length 40 of 1/2/4/8-SISO decoders without warm-up free method.

is almost close. It can be shown in Fig. 4.11 with a long size 6144.

In the following, we will also show some simulation results for the iteration consideration. Because we depend on two reasons: the first one is that the all block lengths are from 40 to 6144, and the needed iterations may be different at BER=10^{-5} under AWGN channel. The second is about throughput computation. Fewer iteration numbers we use, the higher throughput we achieve. For the iterative decoding process, increasing the iteration numbers may cause the performance curve to converge eventually and the required iterations can be obtained. The performance of different lengths and iterations are illustrated in Fig. 4.12 and 4.13. In Fig. 4.12 and 4.13, we can figure out that the shorter lengths need fewer iterations to converge, but the longer ones require more. Finally, we can derive the summary in Table 4.2.

Therefore, the numbers of iteration are so flexible for all block length in our design. That is to say, our turbo decoder can support the iteration numbers from $1 \sim 8$.

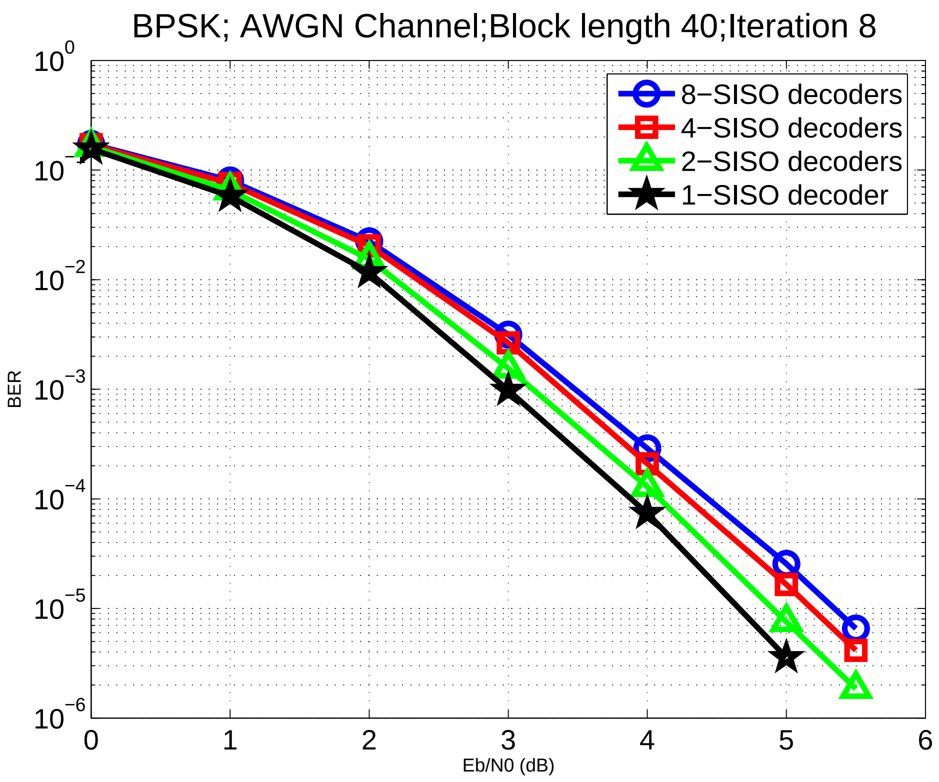


Figure 4.10: Performance comparison of size 40 of 1/2/4/8 decoders with warm-up free method.

4.3.3 Throughput consideration

In this section, we will introduce how to computer the throughput. There are some critical factors affecting the result. The throughput is defined as decoding how many bits per second. In the parallel architecture, the function can be translated in equation (4.9). Each factor will interpret as following: f is the operating frequency, which means the inverse of the critical path of our design. Moreover, the critical path in general resides in the ACS unit. For trellis-based 3GPP-LTE decoder, each state receives two branch metrics and also sends two messages to other states. As a result, radix-2 ACS unit is

Table 4.2: Needed iterations of different size

Block length	**Needed iteration**
40-512	3 or 4
512-2048	5 or 6
2048-6144	7 or 8

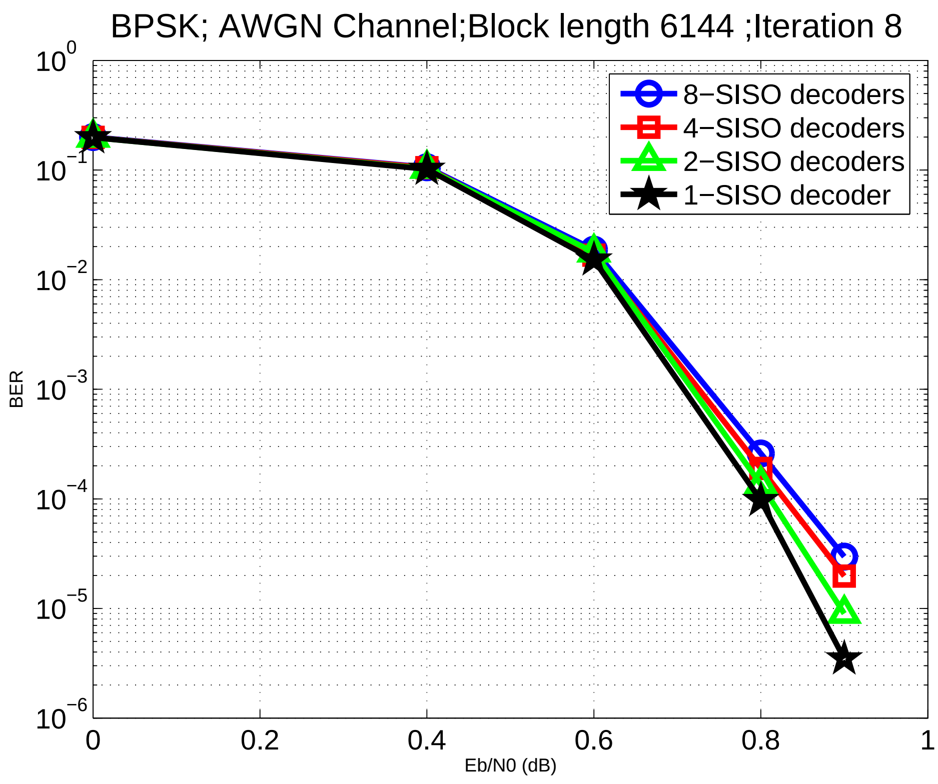


Figure 4.11: Performance comparison of size 6144 of 1/2/4/8 decoders with warm-up free method.

required to decode the trellis structure. In order to diminish the critical path ,only radix-2 structure is considered in our design. It also means that the decoder can decode one bit each time. That is, the $Radixfactor$ in equation (4.9) is 1. $Parallel$ is the number of decoders which are used to decode. As we know, the more decoders, the higher throughput can be derived. But the cost may be increased oppositely. The maximum decoders in our architecture are 8. The significant factor effecting the throughput is the $efficiency$ of each MAP decoder. From the decoding schedule in Fig. 4.4, we realize that there are about three sliding window size latency in each decoding round. The decoder actually starts to decode in the third time interval. Thus the efficiency is formed in equation (4.10). Decoding with 1-SISO, 2-SISO, 4-SISO, and 8-SISO , the efficiency of each SISO decoder may be different. The efficiency in all case is shown in Table 5.1. The final factor is the number of iterations. Based on MAP decoding algorithm, the decoding process through pre-decoding round and post-decoding round is called one iteration. That is why the denominator of equation (4.9) are two times of iterations. Based on all factors mentioned

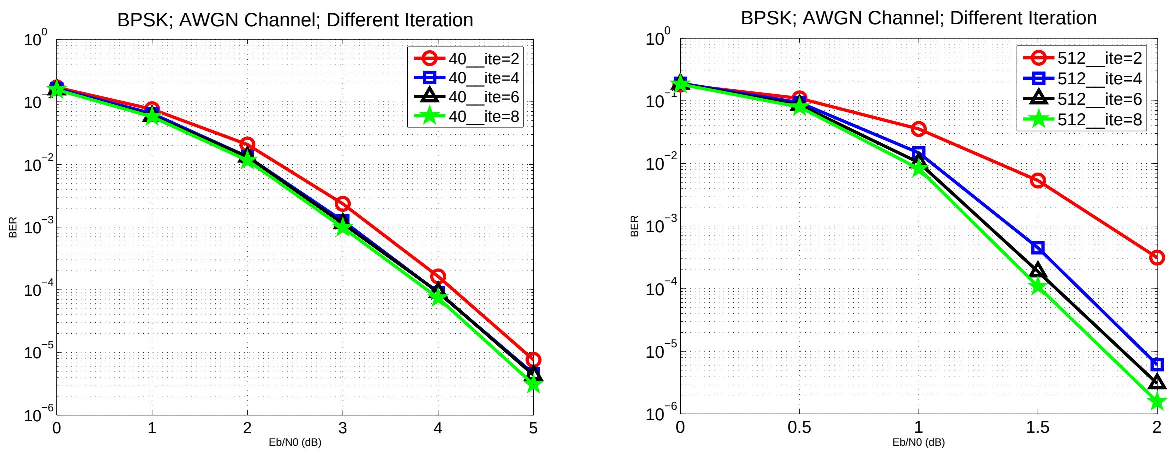


Figure 4.12: Performance of size 40 and 512 under various iterations.

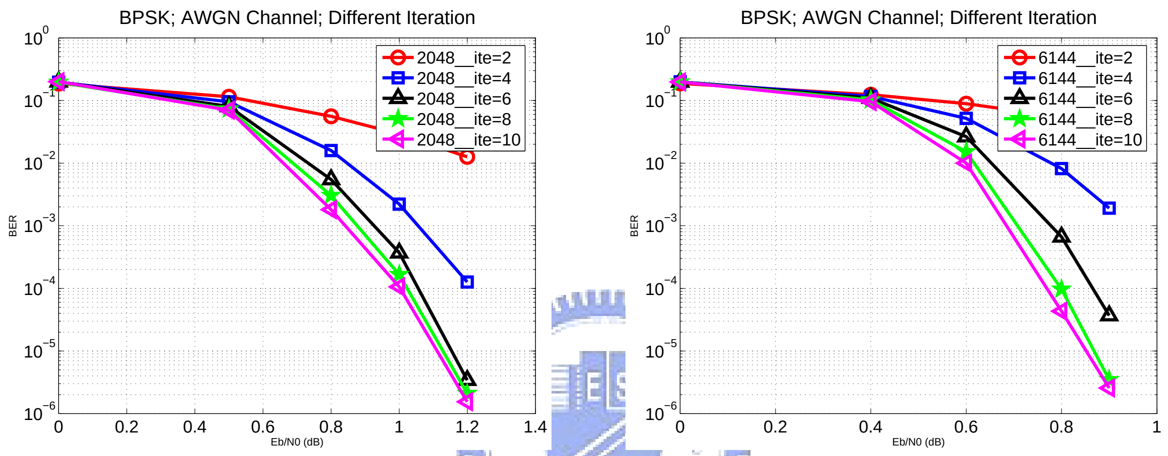


Figure 4.13: Performance of size 2048 and 6144 under various iterations.

above, the throughput can be evaluated eventually. The throughput of our design will show in the next chapter in Table 5.2.

$$\begin{aligned} Throughput(^{bits}/_{s}) &= \frac{Block\ Length}{Needed\ cycles} \times clock\ rate \\ &= \frac{Block\ Length}{decoding\ round\# \times \frac{cycle}{decoding\ round\#} \times \frac{time}{cycle}} \\ &= \frac{f \times \frac{Block\ Length}{cycle\ per\ decoding\ round}}{decoding\ round\#} \\ &= \frac{f \times \frac{Radix\ factor \times Parallel \times real\ decoding\ cycle}{cycle\ per\ decoding\ round}}{decoding\ round\#} \\ &= \frac{f \times Radix\ factor \times Parallel \times efficiency}{2 \times iteration\#}. \end{aligned} \tag{4.9}$$

- **f** : $Operation\ frequency$
- **Radix factor** : $Radix\ structure\ of\ ACS\ unit$
- **Parallel** : $Parallel\ decoder\ numbers$
- **efficiency** : $decoding\ efficiency\ of\ each\ MAP\ decoder$
- **iteration** : $number\ of\ iterations$

$$efficiency = \frac{real\ decoding\ cycle}{cycle\ per\ decoding\ round}. \tag{4.10}$$

Chapter 5

Implementation Results

5.1 Chip characteristic

As described in section 4.3.3, the efficiency of each SISO decoder with 1/2/4/8-decoders is different. It can be calculated from equation (4.10). Table 5.1 shows the efficiency results in all case. In Table 5.1, we know that when block length becomes longer, the efficiency also becomes larger. Moreover, the efficiency of decoding the long block size with 1-SISO and 8-SISO decoders becomes close.

Based on the MAP efficiency results, we can use equation (4.9) to compute the throughput. In our design, the operation frequency can reach 277MHz. The needed iteration numbers can estimate from the performance simulation. Therefore, the throughput summary is shown in Table 5.2.

Based on the architectures described in chapter 4, we proposed a reconfigurable turbo

Table 5.1: Efficiency of each SISO decoder with 1/2/4/8-decoders

	1/2/4/8-SISO decoders			
Block length	**each decoder of 1-SISO**	**each decoder of 2-SISO**	**each decoder of 4-SISO**	**each decoder of 8-SISO**
40	0.465	0.303	0.227	0.172
512	0.917	0.847	0.735	0.581
2048	0.978	0.957	0.917	0.847
6144	0.992	0.985	0.971	0.943

Table 5.2: Throughput summary

Block length	**Needed iteration**	**Parallel decoder numbers**			
		1-SISO	**2-SISO**	**4-SISO**	**8-SISO**
40	3	21.5	28.0	41.9	63.5
512	4	31.8	58.7	101.8	160.9
2048	6	22.5	44.2	84.7	156.4
6144	8	17.2	34.1	67.2	130.6

* unit: Mb/s.

decoder with the quadratic permutation polynomial interleaver applied to 3GPP-LTE system. The proposed decoder supports 188 different block length (40 to 6144). The smaller sliding window size can reduce both the storage size of input buffer and α buffer. The smaller iterations can increasing the decoding throughput. Therefore, we use sliding window size 16, flexible iteration numbers, and also utilize scaling factor 0.75 for performance compensation.

The primary chip specification of the reconfigurable turbo decoder is given in Table 5.1, which is implemented through the cell-based design flow. A test chip has been implemented in a 0.9V, 90nm 1P9M CMOS technology, and the layout view is shown in Fig. 5.1. The chip size is 4.5㎟ (2.5mm*1.8mm) while the core occupies 2.10㎟ (1.84mm*1.14mm) in 90-nm process. The total gate count is 602K including memory units and the chip core density is nearly 81%. After static timing analysis and post layout simulation, the operating frequency can achieve up to 277MHz and the throughput is 130Mb/s with block length 6144. Furthermore, the power consumption operated at 277MHz is 149.03mW with 0.9V supply voltage.

Besides, the power consumption estimated by the post-layout simulation is 149.03mW and can be converted to energy efficiency. The energy efficiency is defined as the average energy consumed per bit within each decoding iteration (nJ/bit/iter). For our decoder with block length 6144 under 8 iterations, the energy efficiency can be calculated as

$$\frac{149.03\text{mW}}{8 \times 130.6\text{Mb/s}} = 0.142\text{nJ/bit/iter}.$$

Table 5.3: 3GPP-LTE turbo decoder chip summary

Technology	UMC 90-nm 1P9M CMOS
Block length	40 to 6144
Iteration	1 to 8
Sliding window	16
Scaling factor	0.75
Quantization	6 bits(3.3)
SISO decoder algorithm	Max-Log MAP algorithm
Code polynomial	$[\ 1\ \frac{1+D+D^3}{1+D^2+D^3}\ ]$
code rate	1/3
MAP	radix-2 MAP
Core area	2.10mm^2
Memory area	0.634mm^2
Gate count	602K
Throughput	130.6Mb/s
Supply voltage	0.9V
Clock rate	277MHz
Power consumption	149.03mW (@0.9V and 277MHz)
Utilization	81%
nJ/bit/iter	0.142

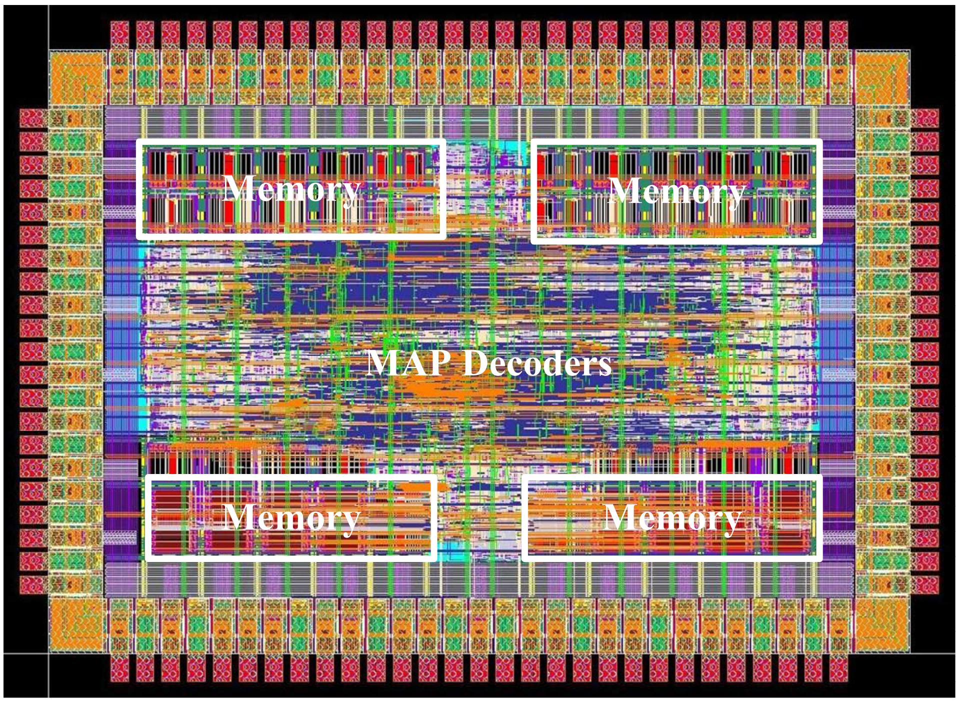


Figure 5.1: Layout photo of 3GPP-LTE turbo decoder.

5.2 Comparison

Table 5.2 lists the comparison of the proposed design with other published works. We both find some architecture for using in 3G wireless communication standards [22] [23] [24] and with parallel structure [22] [25] [24]. In Table 5.2, we can realize that higher throughput can be derived by increasing parallelism. But the penalty is largely area expense. In our proposed design, we utilize 8 parallel SISO decoder and get the modest area cost. In addition, our throughput is about 2 times than [22] under approximate parallelism and our energy efficiency is the smallest of the others.

Table 5.4: Comparison of different turbo decoders

	Proposed	Ref. [22]	Ref. [25]	Ref. [23]	Ref. [24]
Technology	90nm	0.18μm	0.13μm	0.13μm	65nm
Parallelism	8 SISO	6 SISO	64 SISO	1 SISO	32 SISO
Block length	6144	5114	5120	5114	6144
Word bit-width	6	N/A	7	5	6
Performance (dB)	0.73 @BER=10^{-4}	N/A	N/A	0.83 @BER=10^{-4}	N/A
Iteration	8	6	6	6	6
Operating frequency(MHz)	277	166	256	246	200
Core area (mm^2)	2.10	N/A	13	1.2	N/A
Gate count	602K	1299K	N/A	44.1K	4.93M
Power consumption(mW)	149.03	525.67	573	57.8	N/A
Energy efficiency (nJ/bit/iter)	0.14	1.47	0.76	0.7	N/A
Throughput(Mb/s)	130.6	59.6	758	18.6	711
Status	APR	Synthesis	APR	CHIP	Synthesis

Chapter 6

Conclusion and Discussion

6.1 Conclusion

In this thesis, we propose a reconfigurable turbo decoder using contention-free QPP interleaver for the 3GPP-LTE applications. The proposed decoder support full 3GPP-LTE block size range, that is, 188 different block sizes in the range 40-6144. The contention-free interleaver makes the parallel architecture feasible. The modified Max-Log MAP algorithm is used to reduce the hardware complexity and keep the performance close to Log-MAP algorithm. In MAP decoder design, reversing the input sequence order is utilized to eliminate the β_d input buffer requirement. The decoder chip consists of 8 MAP decoders with its corresponding memory sets. The residue-only interleaver is also adopted to reduce the memory storage. Besides, the reconfigurable 1/2/4/8-MAP decoders is flexible and can be utilized in various conditions. Finally, the post-layout simulation result shows that the decoder chip with 602K gate counts can achieve the throughput 130Mb/s with length 6144 and the energy efficiency of this decoder is 0.142 nJ/b/iter.

6.2 Discussion

As described in previous chapters, our design consists of 8 parallel MAP decoders and 8 parallel memory sets. Our parallel architecture decodes one codeword by using multiple processing elements. That is, we separate a codeword into 8 sub-codewords and each sub-codeword is decoded by one MAP decoder separately. For example, when decoding with length 1024, each memory will collect a 128-bits sub-codeword from input buffer till the whole 1024-bits codeword is received. This parallel decoding method is more suitable for long block length to increase throughput efficiently.

In order to support all block lengths, from 40 to 6144, the hardware cost is depended on the longest one. Thus the decoding process utilizing the parallel method mentioned above becomes inefficient for short block size. It is because that each memory still has some unused addresses. For improving this condition, we may decode multiple codewords concurrently in parallel architecture. Fig. 6.1 shows an example of 8, 4, and 2 codewords store in memories at length 40. Due to the limited memory storage, decoding multiple codewords have the maximum block length restriction. As decoding 8 codewords, the maximum block length support up to 768 and each received codeword stores without partition. The maximum size can reach 1536 when decoding 4 codewords and each codeword is separated into two part "Code0-A" and "Code0-B". In the same way, the maximum length of decoding two codewords is 3072 and per codeword must divide into 4 sub-codewords as shown in Fig. 6.1.

Fig. 6.2 shows the schedule of decoding 8 codewords in our architecture. We assume the time of codeword stored in memory is within one decoding round and a whole codeword is stored in one memory bank instead of dividing into several sub-codewords. In Fig. 6.2, when the codeword "Code0" is ready in memory "MEM0", then decoder "MAP0" starts to decode. The incoming "Code1" still enters in "MEM1" to be decoded at the second time interval. The same process still runs until decoding the codeword "Code8" at the eighth interval. At this time, all 8 MAP decoders operate concurrently and it can improve the original throughput about 8 times.

Fig. 6.3 and 6.4 are the time schedule of decoding 4 codewords and 2 codewords respectively. In these two cases, the codewords stored in memory distinguish from decoding 8 codewords. In Fig. 6.3, one received codeword is divided into two sub-codewords, "Code0-A" and "Code0-B", which are stored in "MEM0" and "MEM1" and then are decoded by "MAP0" and "MAP1" concurrently. At the same time, the "MEM2" and "MEM3" still collect coming codeword which also separates into two parts. The similar procedure keeps on until decoding the fourth codeword. At the eighth time interval, the 8

MAP decoders operate simultaneously and we can derive approximately 4 times throughput than original one. Similarly, the received codeword is divided into 4 sub-codewords and we can use at most 4 MAP decoders to decode per codeword as decoding 2 codewords. It can be shown in Fig. 6.4.

Decoding multiple codewords in parallel architecture can improve the throughput. But there are still two issues to be considered. Firstly, due to the limited memory area in our design, decoding multiple codewords have the maximum block length restriction. Secondly, the iteration numbers are at least 4 times in this decoding process. In Fig. 6.2, 6.3 and 6.4, we can realize that decoding multiple codewords need at least 8 decoding rounds (4 iterations) to achieve the all MAP decoders working simultaneously. The summary of decoding multiple codewords and some estimations are summarized in Table 6.1. We can derive the throughput of decoding multiple codewords within different block lengths. When the block length is under 768, the estimated maximum throughput can achieve 208Mb/s. Moreover, the size under 1536 and 3072 also can derive the maximum throughput 174Mb/s and 208Mb/s, respectively.

Table 6.1: Summary of decoding multiple codewords

Multiple codewords	**Max. SISO no. per codeword**	**Max. block length**	**Needed iteration**	**Max. throughput**
1	8	6144	8	130 Mb/s
2	4	6144/2=3072	7	149 Mb/s
4	2	6144/4=1536	6	174 Mb/s
8	1	6144/8=768	5	208 Mb/s

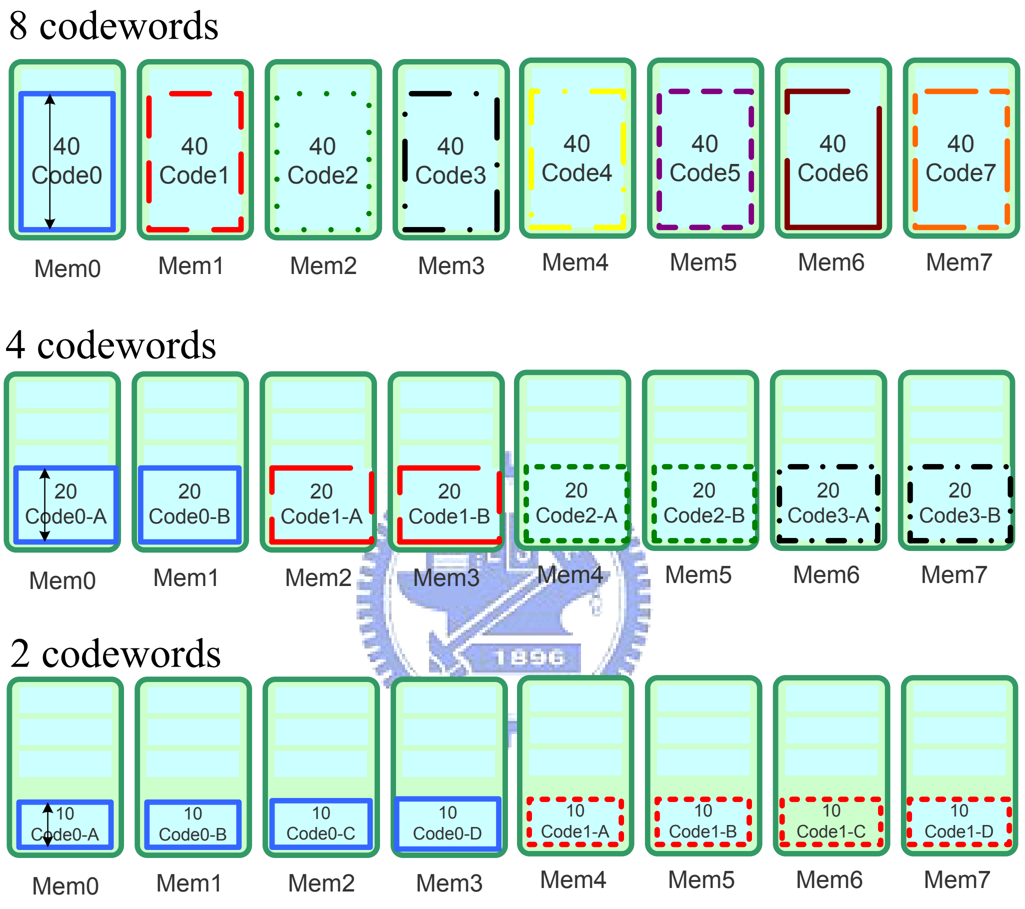


Figure 6.1: Example of multiple codewords store in memories (Length=40).

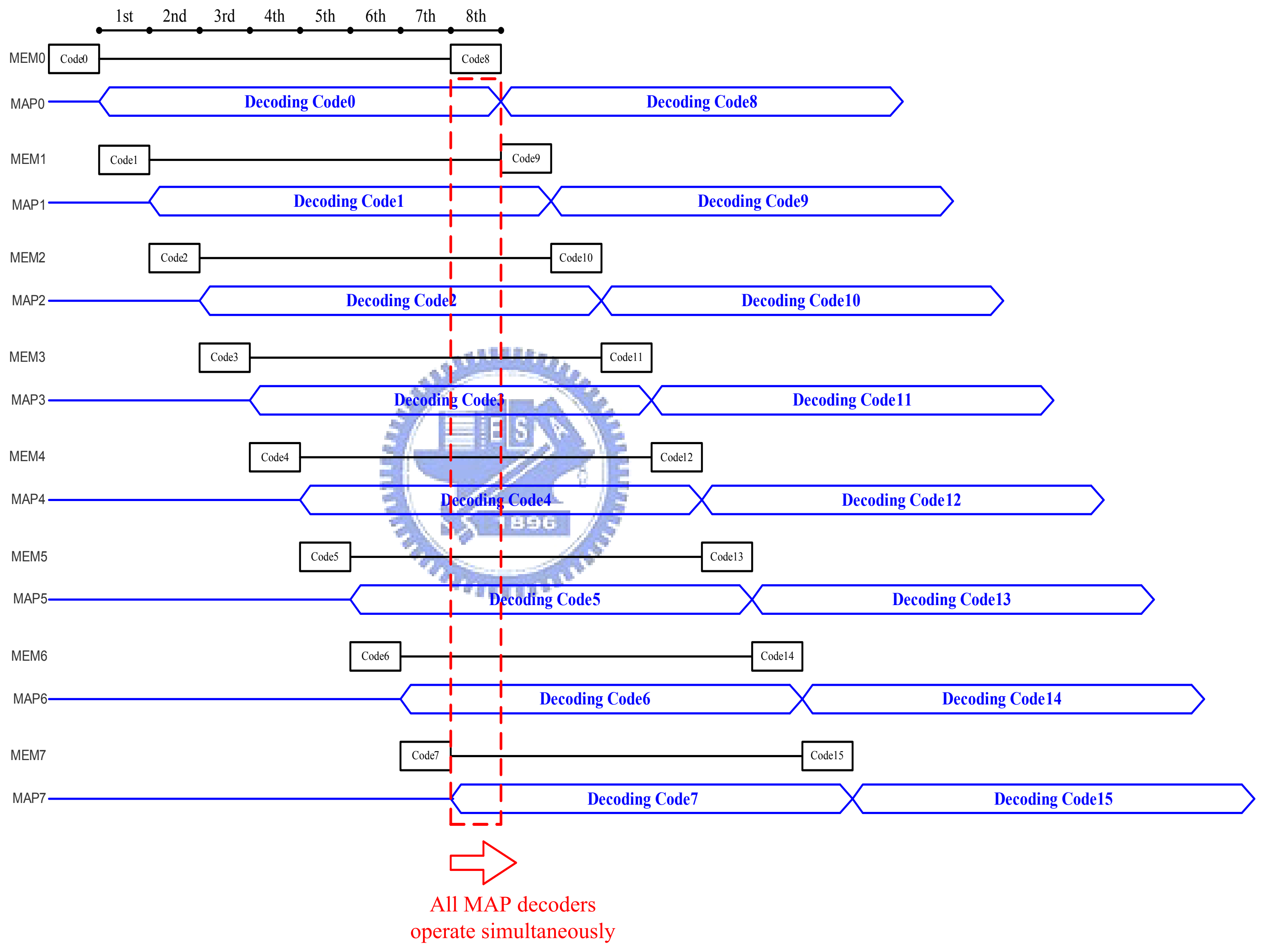


Figure 6.2: Schedule of decoding 8 codewords.

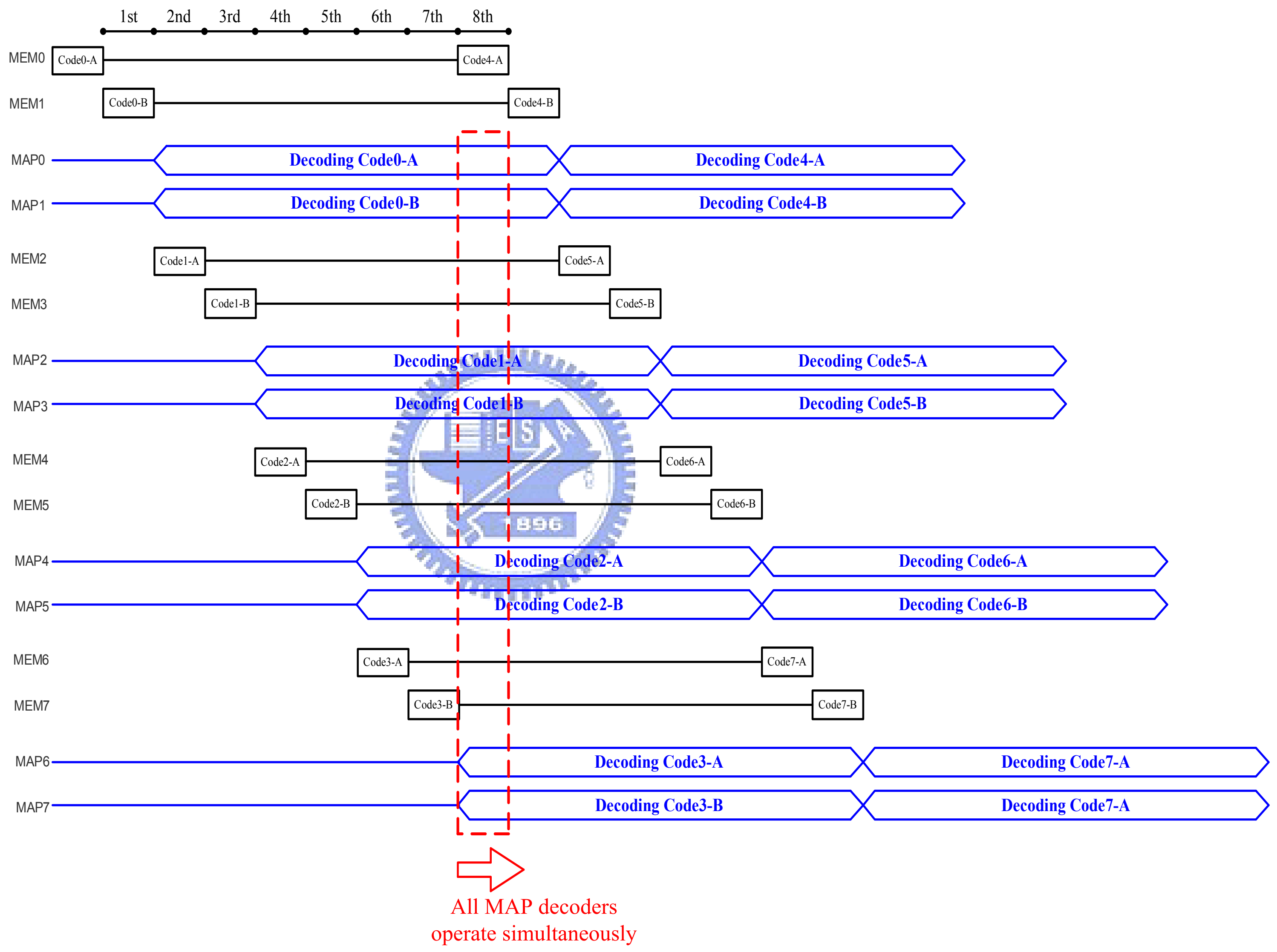


Figure 6.3: Schedule of decoding 4 codewords.

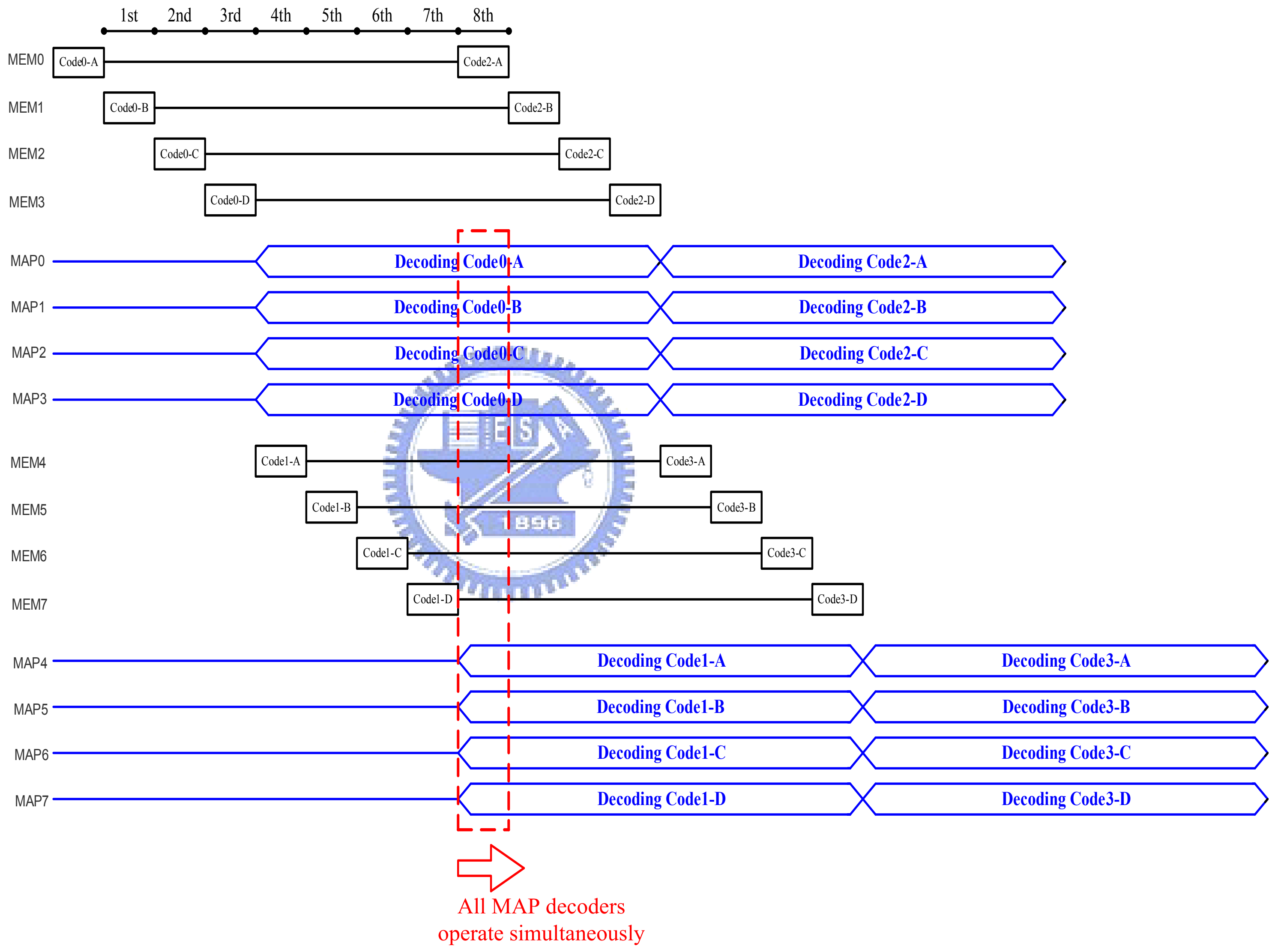
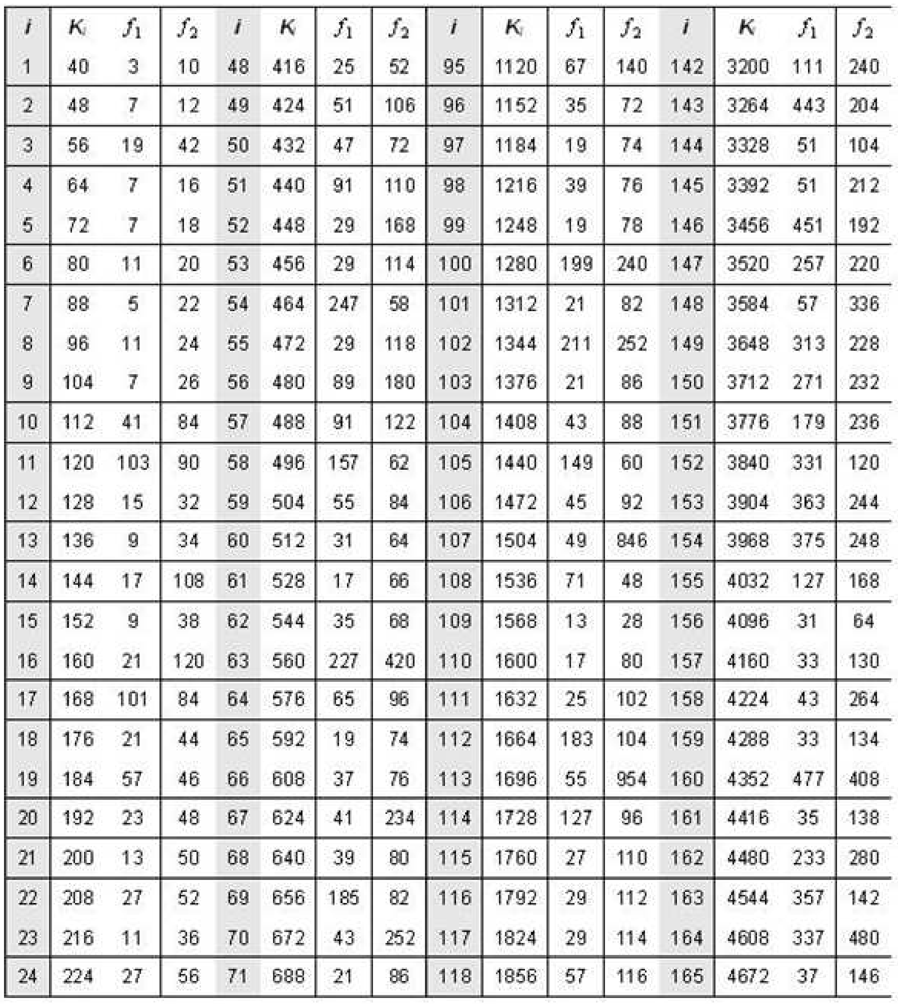


Figure 6.4: Schedule of decoding 2 codewords.

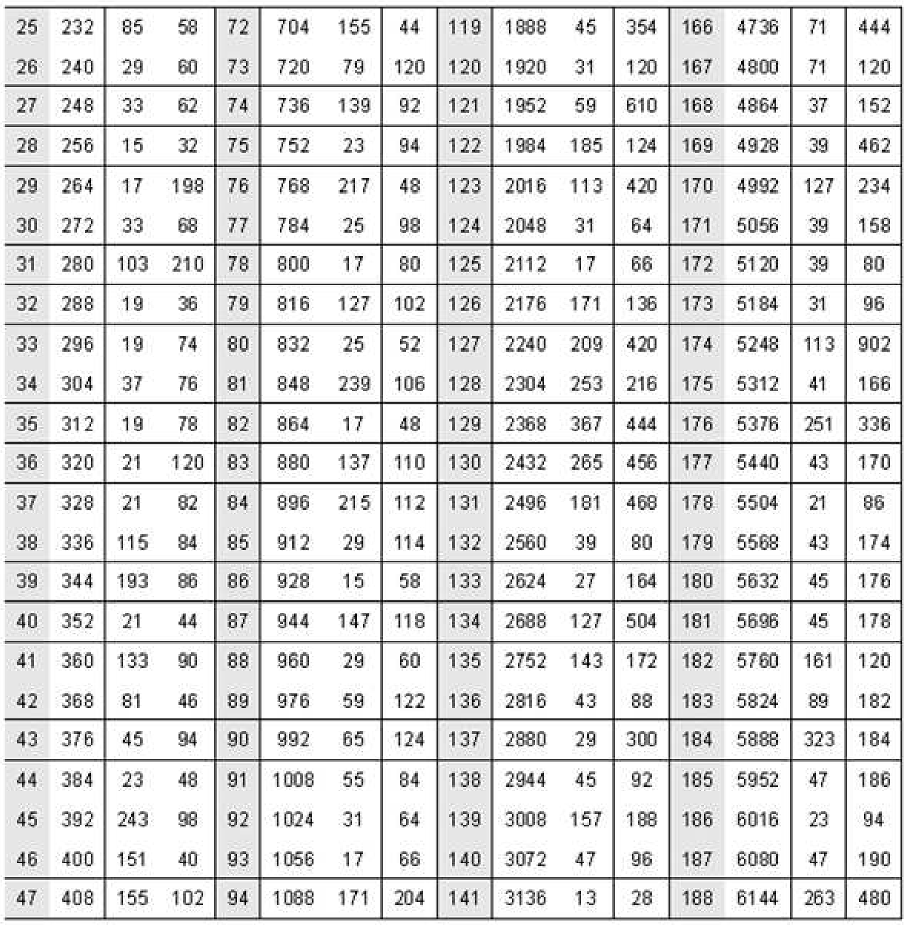
Appendix A

QPP Interleaver Parameter



i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2	i	K_i	f_1	f_2
1	40	3	10	48	416	25	52	95	1120	67	140	142	3200	111	240
2	48	7	12	49	424	51	106	96	1152	35	72	143	3264	443	204
3	56	19	42	50	432	47	72	97	1184	19	74	144	3328	51	104
4	64	7	16	51	440	91	110	98	1216	39	76	145	3392	51	212
5	72	7	18	52	448	29	168	99	1248	19	78	146	3456	451	192
6	80	11	20	53	456	29	114	100	1280	199	240	147	3520	257	220
7	88	5	22	54	464	247	58	101	1312	21	82	148	3584	57	336
8	96	11	24	55	472	29	118	102	1344	211	252	149	3648	313	228
9	104	7	26	56	480	89	180	103	1376	21	86	150	3712	271	232
10	112	41	84	57	488	91	122	104	1408	43	88	151	3776	179	236
11	120	103	90	58	496	157	62	105	1440	149	60	152	3840	331	120
12	128	15	32	59	504	55	84	106	1472	45	92	153	3904	363	244
13	136	9	34	60	512	31	64	107	1504	49	846	154	3968	375	248
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146

Figure A.1: QPP parameters



25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1056	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1088	171	204	141	3136	13	28	188	6144	263	480

Figure A.2: QPP parameters(Conti.)

Bibliography

[1] *Physical Layer Standard for cdma2000 Spread Spectrum Systems*, 3GPP2 Std. C.S0002-C, 2002.

[2] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over interger rings," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 101–119, Jan. 2005.

[3] J. H. Andersen, "'turbo' coding for deep space applications," in *IEEE International Symposium on Inform. Theory*, no. 17-22, Sep. 1995, p. 36.

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[5] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision output and its applications," in *IEEE CLOBE-COM*, Dallas, Nov. 1989, pp. 47.1.1–47.1.7.

[6] Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal decoding algorithm," in *Proc. IEEE Int. Conf. Commun.*, 1995, pp. 1009–1013.

[7] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1261–1271, Feb./Mar./Apr. 1994.

[8] S. A. Barbulescu, "Sliding window and interleaver design," *IEEE Electronics letters*, vol. 37, no. 21, pp. 1299–1300, Oct. 2001.

[9] *Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)*, 3GPP TS 36.212 Std. V8.2.0, 2008.

[10] E. Rosnes and O. Y. Takeshita, "Optimum distance quadratic permutation polynomial based interleavers for turbo codes," in *Proc. International Symposium on Inform. Theory*, Seattle, USA, July 2006, pp. 1988–1992.

[11] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," in *TDA Progress Report 42-122*, JPL, Pasadena, CA, Aug. 1995.

[12] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[13] A. Nimbalker, T. K. Blankenship, B. Classon, T. E. Fuja, and J. D. J. Costello, "Contention-free interleavers," in *Proc. IEEE International Symposium on Inform. Theory*, Chicago, IL, June-July 2004, p. 54.

[14] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.

[15] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. Int. Conf. Commun.*, Seattle, WA, Jun. 1995.

[16] J. Vogt and A. Finger, "Improving the max-log-map turbo decoder," *Electronics Letters*, vol. 36, pp. 1937–1939, Nov. 2000.

[17] C.-C. Wong, C.-H. Tang, M.-W. Lai, Y.-X. Zheng, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y.-T. Su, "A 0.22nJ/b/iter 0.13um turbo decoder chip using inter-block permutation interleaver," in *IEEE CICC*, 2007, pp. 273–276.

[18] C. H. Tang, C. C. Wong, C. L. Chen, C. C. Lin, and H. C. Chang, "A 952mb/s max-log map decoder chip using radix-4x4 acs architecture," in *IEEE A-SSCC*, 2006, pp. 79–82.

[19] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Trans. on VLSI Systems*, vol. 10, no. 3, pp. 279–285, June 2002.

[20] C. Shung, P. Siegel, G. Ungerboeck, and H. Thapar, "VLSI architertures for metric normalization in the Viterbi algorithm," in *Int. Conf. Communications*, vol. 4, Atlanta, CA, Apr. 1990, pp. 1723–1728.

[21] Z. He, P. Fortier, and S. Roy, "Highly-parallel decoding architectures for convolutional turbo codes," in *IEEE Trans. VLSI Systems*, vol. 14, no. 10, Oct. 2006.

[22] M. J. T. et al., "A scalable system architecture for hight hroughput turbo-decoders," *The Journal of VLSI Signal Processing*, pp. 63–77, 2005.

[23] C. Benkeser, A. Burg, T. Cupaiuolo, and H. Qiuting, "A 58mw 1.2mm2 hsdpa turbo decoder asic in 0.13um cmos," in *ISSCC Dig. Tech. Papers*, 2008, pp. 264–612.

[24] Y. Sun, Y. Zhu, M. Goel, and J. R. Cavallaro, "Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards," in *Application-Specific Systems, Architectures and Processors (ASAP)*, 2008, pp. 209–214.

[25] G. Prescher, T. Gemmeke, and T. Noll., "A parametrizable low-power high-throughput turbo-decoder," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, Mar. 2005, pp. 25–28.