國立交通大學

電子工程學系　電子研究所碩士班

碩　士　論　文

Gbps 高速渦輪碼之設計與實現

Design and Implementation of Gbps Turbo Decoders

學生 ： 賴名威

指導教授 ： 李鎮宜 教授

中華民國九十六年七月

Gbps 高速渦輪碼之設計與實現

Design and Implementation of Gbps Turbo Decoders

研 究 生：賴名威　　Student：Ming-Wei Lai

指導教授：李鎮宜　　Advisor：Chen-Yi Lee

國 立 交 通 大 學

電子工程學系 電子研究所 碩士班

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

Gbps 高速渦輪碼之設計與實現

學生：賴名威　　指導教授：李鎮宜 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘要

自九零年代初渦輪碼被發現以來，由於出色的錯誤更正能力一直以來廣泛的引起研究者的注意，在近期寬頻無線通訊以及第四代行動通訊等協定中，對於高速渦輪碼的資料流量的要求，也分別訂定了每秒 70Mb 以及每秒 20Mb 到 100Mb 的高速傳輸量，因此，對於高速渦輪碼的需求也與日俱增。而在渦輪碼的解碼中，由於尋找最大事後機率的解碼方式中，含有遞迴式的計算方式，也因此造成了在渦輪碼的解碼中，產生了很可觀的時間延遲。在這篇論文當中，針對高速的渦輪碼解碼，我們提出了一個完整的解決方案，其中，包含一個資料讀寫平順無誤的打散器設計，一個多級高階的渦輪解碼器，以及提出一種能夠運用在兩個維度的平行解碼器架構，由於這些因素，使得我們在本論文中所提出的設計，與現今科技之水準比較下，達到最高的能源效率以及每秒的資料解碼量，達到最高的水準。

Design and Implementation for Gbps Turbo Decoders

Student : Ming-Wei Lai Advisor : Dr. Chen-Yi Lee

Institute of Electronics Engineering

National Chiao Tung University

ABSTRACT

Turbo codes have received a lot of interest since 90's because of their excellent performance. To apply turbo codes in high-speed digital communications, such as in broadband wireless access based on the IEEE 802.16 standard supporting data rates of up to 70 Mb/s, and in fourth generation cellular systems, which are expected to provide a data rate from 20 to 100 Mb/s for high mobility, high throughput of turbo codes is a critical issue. The recursive computations in the MAP-based decoding of turbo codes usually introduce a significant amount of decoding delay. In this thesis, we present a total solution for a high throughput application, including a contention-free interleaver design, a high radix turbo decoder design, and the two-dimension parallel decoding architecture. The chip proposed in this thesis is the most power efficient and the fastest design in the state of the art.

誌　　謝

忙碌而充實的研究生生活，隨著口試的結束也悄悄地接近了尾聲。在這二年的研究生涯中，首先要感謝指導教授李鎮宜教授在這段時間對我的指導，並且提供一個良好的研環境讓我能夠專注於學業與研究。此外，還要感謝 Si2 實驗室以及 Ocean Group 所有的成員，在這段時間給了我非常多的協助與討論，使得我的研究得以順利的完成。其中，特別要感謝張錫嘉教授、林建青學長、陳志龍學長以及翁政吉學長給予我非常多的協助與指導，並且在相互的討論中，給予了我非常多的構想與啟發，使得這些研究能夠順利的完成。最後，感謝我的父母一路上對我的支持與協助，沒有你們也就沒有今天的我。

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

A communication system conveys a information source to a destination through a channel. Fig. 1.1 shows a fundamental block diagram of traditional digital communication system. Generally, the system can be divided into transmitter and receiver via a channel. The main task of transmitter, including 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. And 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. Although these redundant bits may lower data transmission rate, the channel coding eliminate the effects of noise disturbances and thus improve the performance, compared with an uncoded system.

With high coding gain provided by channel codes, the high performance channel codes are widely used in some circumstances, such as low power transmission, high order modulation, and complex channel conditions, in the recent decades. In channel codes, there are three codes that provide marvelously high performance: block turbo code, convolutional turbo code, briefly called turbo code, and low density parity check code. The block turbo code is hard to implement due to the irregular Trellis structure. Therefore, the candidates for the

high performance criterion remain turbo code and LDPC code.

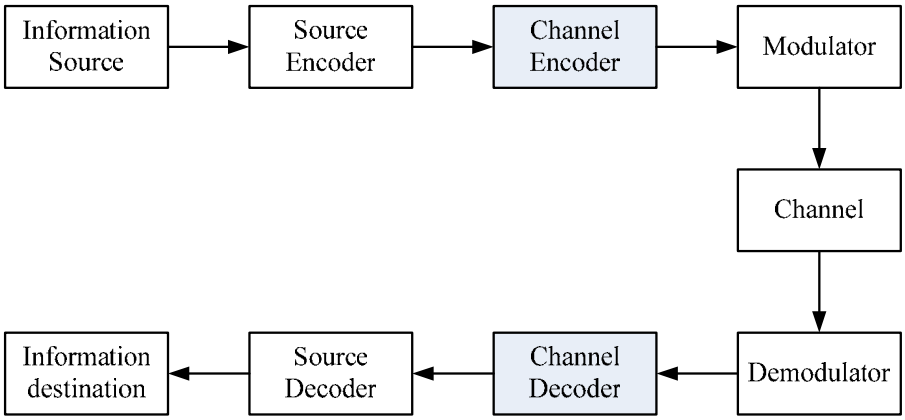


Fig. 1.1 The block diagram of digital communication system

The comparison of turbo code and LDPC are listed in Table 1.1. From the point of view with block length bigger than 10000, the performance of LDPC would be better than turbo code due to the property of component codes. With block length smaller than 10000, the performance of turbo code would be better due to the girth problem of LDPC. The Parallelism of LDPC is easier for implementation than turbo code. Most important of all, the routing problem of LDPC is getting serious as the throughput demand growing. Meanwhile, the advanced process for high speed implementation aggravates the routing congestion problem of LDPC. Apparently, for a high speed application, the turbo codes would be more suitable and area-efficient if we can increase the throughput of the turbo codes.

Table 1.1 Comparison of Turbo code and LDPC

		LDPC Code	Turbo Code
Performance (Block length)	**>10000**	**Better**	**Good**
	<10000	**Good**	**Better**
Throughput (Parallelism)		**Better**	**Medium**
Efficiency		**Medium**	**Medium**
Routing		**Difficult**	**Medium**

In this thesis, our work is motivated to design a high performance and high-throughput turbo decoder. We attempt to achieve the target from two aspects: First one is to speed up the decoding processing elements used in the whole turbo decoder by high radix structures and perfect utilization of hardware. Second, we employ a well-designed interleaver fit for parallel decoding architectures to reduce the latency caused by the interleaver and propose a practical hardware architecture for the whole turbo decoder. Finally, we will propose a new point of view of parallel decoding for MAP-based turbo decoder with the modest hardware cost.

1.2 Thesis Organization

This thesis consists of 7 chapters. In chapter 2, we'll focus on interpreting turbo coding and decoding algorithm and its relative techniques. Chapter 3 presents a total solution of a high speed turbo decoder with a parallel architecture, including the design of a contention-free interleaver, a high radix turbo decoder, and some techniques applied on our design. Chapter 4 explains how we improve the utilization of the previous chip. A Modified interleaver control for multiple block lengths support will be introduced. In chapter 5, we present the two architectures. A two-dimension parallel architecture will be proposed. Meanwhile, a simplified intra-codeword parallel architecture and the relative issues will be discussed. Finally, conclusion and future work are made in chapter 6.

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[1]. It has been proved to have a performance close to Shannon limit with simple constituent codes concatenated by an interleaver. This new technique is now adopted in 3GPP, 3GPP2 and WiMAX standards due to its excellent error correction ability. In this chapter, we'll describe the principle of both turbo encoding and turbo decoding methods. The sliding-window approach and the tail-biting coding structure will also be interpreted here.

2.1 Principle of Turbo code

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 a turbo interleaver. The two RSC encoders are also called constituent codes of the turbo code. The block diagram of the turbo encoder is illustrated in Fig. 2.1. Note that the same input data are encoded by each RSC encoder but in different order. In 3GPP2 standard, 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 encoded sequence should be $\{X, Y_0, Y_1, Y_0', Y_1'\}$, and the overall code rate is 1/5.

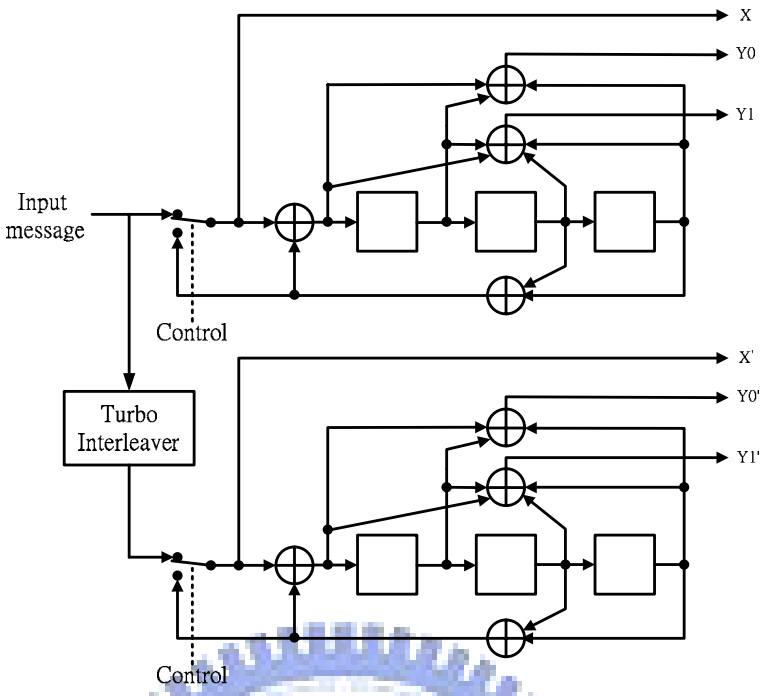


Fig. 2.1 Turbo encoder for 3GPP2 standard

After encoding all input messages, we have to generate several tail bits 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 simple solution is provided in Fig. 2.2. While encoding input messages, the switch is set to position "A". Once messages of whole block are encoded, the position of switch is changed to "B" for three additional cycles. This will force all registers to zeros and thus back to zero state.

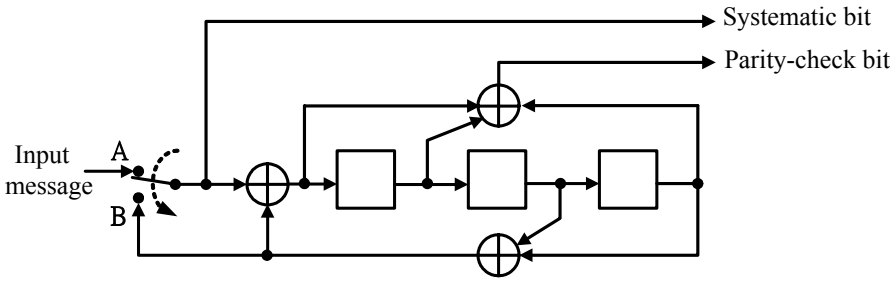


Fig. 2.2 Trellis Termination

2.1.2 Turbo Interleaver

The interleaver plays a very important role in turbo encoder. 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 has been evaluated in [9]. The result shows that the bit-error-probability upper bound of turbo code is approximately proportional to 1/N, where N is the block size of turbo interleaver. 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. 2.3, where r_s is the received systematic information, r_{p1} 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 decoders, which are soft-in/soft-out (SISO) decoders concatenated serially via one interleaver and one de-interleaver. An additional interleaver is used to interleave the input systematic information and then provides the interleaved data to the second SISO decoder. Two component decoders can be implemented based on either soft-output Viterbi algorithm (SOVA) [21] or maximum *a posteriori* probability (MAP) algorithm [2], which will be discussed particularly in the next section. During iterative decoding process, each constituent decoder delivers the extrinsic information $L_{ex}(u)$ which is taken as *a priori* information for the other constituent decoder. That is $L_{in1}(u_k) = \tilde{L}_{ex2}(u_k)$ and $L_{in2}(u_k) = \tilde{L}_{ex1}(u_k)$. As the number of iterations increases,

better coding gain is expected. However, the correlation between two SISO decoders is also raised up. Therefore, there is no significant performance improvement if the number of iterations reaches a threshold.

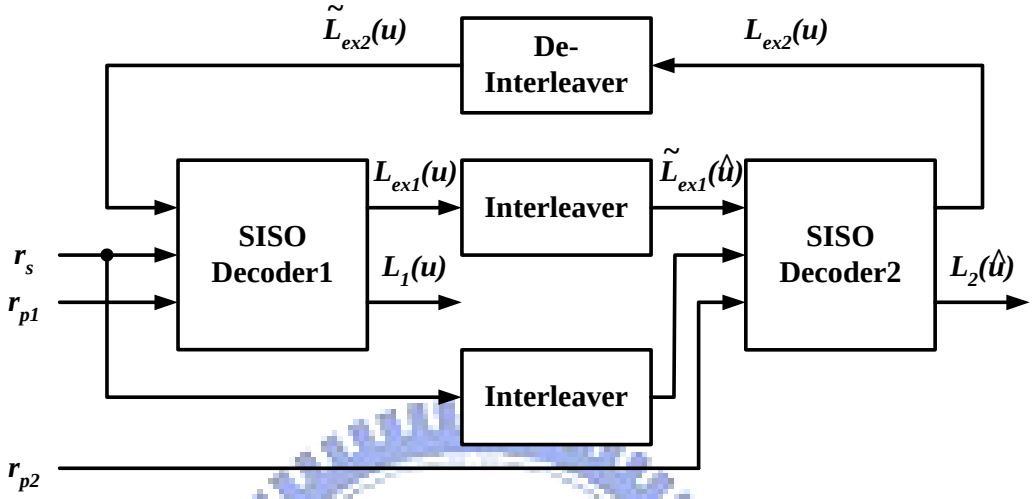


Fig. 2.3 Turbo decoding flowchart

2.1.4 Error floor effect

Although turbo coding provides an excellent performance, the bit-error-rate certainly starts to decrease quite slowly at high signal-to-noise ratio (SNR). This phenomenon can be observed in [19]. It is due to relative small free distance of turbo codes, and is called an "error floor" [22]. Consider the relation of the minimum free distance and the bit error probability in turbo coding, which can be expressed by

$$P_b \propto Q\left(\sqrt{2d_{free}R\frac{E_b}{N_0}}\right) \qquad (2.\ 1)$$

where d_{free} is the minimum free distance 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. Thus, the error correction ability

is limited on the weak constituent code only. To overcome this issue, we can increase the interleaver size to lower the position of the error floor or concatenate a block code, e.g. BCH code, as an outer code to remove the left error bits. For more details, please refer to [9] [23].

2.2 Decoding Algorithms for Turbo Code

It has been proved that the MAP algorithm is the optimal decoding method for turbo code while comparing with SOVA [10]. Unlike Viterbi algorithm which utilizes maximum likelihood (ML) algorithm to find the codewords with minimum error probability, the MAP algorithm minimizes the symbol (or bit) error probability. In this section, we'll focus on introducing the turbo decoding methods based on MAP algorithm [2][3]. Although SOVA is also one of the commonly used techniques for turbo decoding, we'll skip it since it's not adopted in our proposed design. To understand more detail about SOVA, please refer to [21]. And some comparisons of MAP algorithm and SOVA applied in turbo code system are shown in [10].

2.2.1 The MAP algorithm

The main idea of MAP algorithm is to compute the log-likelihood ratio (LLR) of the transmitted information bit u_k conditioned on the received information $\boldsymbol{r}_k$ for $1 \leqq k \leqq$ N, where N is the block length of encoded message.

$$L(\hat{u}_k) = L(u_k \mid \boldsymbol{r}) = \log \frac{P(u_k = +1 \mid \boldsymbol{r})}{P(u_k = -1 \mid \boldsymbol{r})} \tag{2. 2}$$

Here $\boldsymbol{r}$ is the vector of received soft values, and can be represented as $[r_1, r_2, \ldots, r_n]$ where n is the number of output bits for each encoded bit in the constituent code. Let's consider the trellis diagram of turbo code in 3GPP2 standard, which is shown in Fig. 2.4 as an example. Note that the solid lines represent the transitions corresponding to an information bit u_k of -1, while the dotted lines represent the transitions corresponding to an information bit u_k of +1.

Then, the equation can be further expressed as

$$L(\hat{u}_k) = \log \frac{P(u_k = +1 \mid \boldsymbol{r})}{P(u_k = -1 \mid \boldsymbol{r})} = \log \frac{\sum_{u_k=+1} P(s_{k-1}, s_k, \boldsymbol{r})}{\sum_{u_k=-1} P(s_{k-1}, s_k, \boldsymbol{r})}. \tag{2. 3}$$

where the numerator and denominator are the sum of joint probabilities for all existing transitions from state s_{k-1} to state s_k that corresponding to an information bit u_k of +1 and -1 respectively.

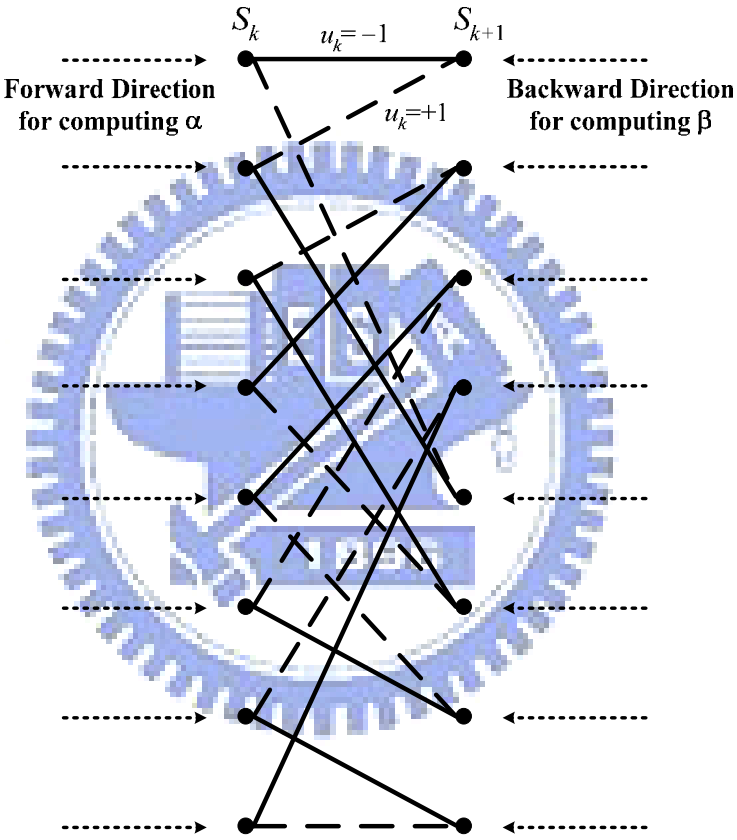


Fig. 2.4 Trellis diagram of turbo code in 3GPP2 standard

Assume the encoded data is transmitted through the discrete memoryless channel (DMC), and then the term $P(s_{k-1}, s_k, \boldsymbol{r})$ can be decomposed as three terms:

$$\begin{aligned} P(s_{k-1}, s_k, \boldsymbol{r}) &= \underbrace{P(s_{k-1}, r_{j<k})} \cdot \underbrace{P(s_k, \boldsymbol{r}_k \mid s_{k-1})} \cdot \underbrace{P(r_{j>k} \mid s_k)} \\ &= e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1}, s_k)} \cdot e^{\beta_k(s_k)} \end{aligned}. \tag{2. 4}$$

Here $e^{\alpha_{k-1}(s_{k-1})}$ is the joint probability of state s_{k-1} and received symbols $\boldsymbol{r}_j$ from the beginning

of the block up to time index "k-1". Similarly, $e^{\beta_k(s_k)}$ is that of state s_k and received symbols $\boldsymbol{r}_j$ from the end of block back to time index "k". By shifting the value "k", it can be perceived that α is the forward recursion of the MAP algorithm, and can be formulated as

$$e^{\alpha_k(s_k)} = \sum_{s_{k-1}} e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\alpha_{k-1}(s_{k-1})} . \tag{2. 5}$$

The same as above, the backward recursion β can be formulated as

$$e^{\beta_{k-1}(s_{k-1})} = \sum_{s_k} e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\beta_k(s_k)} . \tag{2. 6}$$

Note that since the trellis of turbo code diverges from state zero and converges to state zero, the initial condition of the forward recursion and backward recursion should be set as

$$\begin{cases} e^{\alpha_0(s_0)} = 1, & \text{for } s_0 = 0 \\ e^{\alpha_0(s_0)} = 0, & \text{otherwise} \end{cases} \tag{2. 7}$$

and

$$\begin{cases} e^{\beta_N(s_N)} = 1, & \text{for } s_N = 0 \\ e^{\beta_N(s_N)} = 0, & \text{otherwise} \end{cases} \tag{2. 8}$$

For any existing transitions from s_{k-1} to s_k, the branch transition probability $e^{\gamma_k(s_{k-1},s_k)}$ can be further decomposed as

$$\begin{aligned} e^{\gamma_k(s_{k-1},s_k)} &= P(s_k, \boldsymbol{r}_k \mid s_{k-1}) \\ &= P(s_k \mid s_{k-1}) \cdot P(\boldsymbol{r}_k \mid s_{k-1}, s_k) . \\ &= P(u_k) \cdot P(\boldsymbol{r}_k \mid u_k) \end{aligned} \tag{2. 9}$$

Here, the term "$P(u_k)$" is well-known as *a priori* probability. According to the definition of LLR, which is

$$L(u_k) = \log \frac{P(u_k = +1)}{P(u_k = -1)}, \tag{2. 10}$$

$P(u_k)$ can be rewritten as

$$P(u_k = \pm 1) = \frac{e^{\pm L(u_k)}}{1 + e^{\pm L(u_k)}} = \frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}} \cdot e^{L(u_k) \cdot u_k / 2} = A_k \cdot e^{L(u_k) \cdot u_k / 2}. \quad (2.\ 11)$$

where the term A_k is equal for all transitions at the same time index, and thus will cancel out in (2. 3). On the other hand, the value of $P(\boldsymbol{r}_k|u_k)$ is dependent on channel characteristic. For an additive white Gaussian noise (AWGN) channel, the LLR of $\boldsymbol{r}_k$ conditioned on u_k can be expressed as

$$L(\boldsymbol{r}_k|u_k) = \log \frac{P(\boldsymbol{r}_k \mid u_k = +1)}{P(\boldsymbol{r}_k \mid u_k = -1)} = \log \frac{\prod_{v=1}^{n} \exp(-\frac{E_s}{N_0}(r_{k,v} - x_{k,v})^2)}{\prod_{v=1}^{n} \exp(-\frac{E_s}{N_0}(r_{k,v} - x_{k,v})^2)} = \sum_{v=1}^{n} L_c \cdot r_{k,v} \cdot x_{k,v} \quad (2.\ 12)$$

where $L_c = 4E_s/N_0$ and is called the channel reliability. Here, $x_{k,v}$ is the v-th transmitted symbol while encoding u_k. For systematic codes, $x_{k,1}$ is equal to u_k. Now we can obtain the value of $P(r_k|u_k)$ by using the technique in (2. 11) but substitute $L(u_k)$ with $L(r_k|u_k)$.

$$P(\boldsymbol{r}_k \mid u_k) = B_k \cdot \exp(\frac{1}{2} L_c r_{k,1} u_k + \frac{1}{2} \sum_{v=2}^{n} L_c r_{k,v} x_{k,v}) \quad (2.\ 13)$$

For the same reason in (2. 11), B_k will also cancel out in (2. 3). Combining (2. 11) and (2. 13), the γ_k in (2. 9) can be reduced to

$$e^{\gamma_k(s_{k-1}, s_k)} = A_k \cdot B_k \cdot \exp\left(\frac{1}{2}\left((L_c r_{k,1} + L(u_k)) \cdot u_k + \sum_{v=2}^{n} L_c r_{k,v} x_{k,v}\right)\right). \quad (2.\ 14)$$

Substituting (2. 5), (2. 6), (2. 14) into (2. 4), we can derive the *a posteriori* LLR in the form of

$$L(\hat{u}_k) = \log \frac{\sum_{\substack{(s_{k-1},s_k)\\ u_k=+1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\beta_k(s_k)}}{\sum_{\substack{(s_{k-1},s_k)\\ u_k=-1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\gamma_k(s_{k-1},s_k)} \cdot e^{\beta_k(s_k)}} \qquad (2.\ 15)$$
$$= L_c r_{k,1} + L(u_k) + L_{ex}(u_k)$$

where

$$L_{ex}(u_k) = \log \frac{\sum_{\substack{(s_{k-1},s_k)\\ u_k=+1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\frac{1}{2}\sum_{v=2}^{n} L_c r_{k,v} x_{k,v}} \cdot e^{\beta_k(s_k)}}{\sum_{\substack{(s_{k-1},s_k)\\ u_k=-1}} e^{\alpha_{k-1}(s_{k-1})} \cdot e^{\frac{1}{2}\sum_{v=2}^{n} L_c r_{k,v} x_{k,v}} \cdot e^{\beta_k(s_k)}}. \qquad (2.\ 16)$$

The term $L_{ex}(u_k)$ is called extrinsic information since it's a function of the redundant information that comes from the encoder. It removes the information about the systematic input and *a priori* information from $L(\hat{u}_k)$. Therefore, this term is useful to estimate *a priori* probability for the next component decoder, and great performance improvement in iterative MAP decoding can be achieved.

2.2.2 The Log-MAP algorithm

It can be figured out easily that Max-Log-MAP algorithm is a sub-optimal solution for turbo decoding since an approximation of (2. 21) is used to reduce the complexity of MAP algorithm. This problem can be solved by Log-MAP algorithm [24]. It employs the Jacobian algorithm

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|\delta_1 - \delta_2|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|), \end{aligned} \qquad (2.\ 17)$$

where $f_c(|\delta_1\text{-}\delta_2|)$ is a correction function, and thus the performance can be improved. It has been proved that (2. 21) can be computed exactly by a recursive operation of (2. 25) [10].

$$\begin{aligned}\log(e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_n}) &= \log(\Delta+e^{\delta_n}), \quad \Delta = e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_{n-1}} = e^{\delta} \\ &= \max(\log\Delta, \delta_n) + f_c(|\log\Delta - \delta_n|) \\ &= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|)\end{aligned} \tag{2. 18}$$

Substituting (2. 18) and (2. 19) into (2. 25), the forward and backward recursions can be represented as

$$\alpha_k(s_k) = \max_{s_{k-1},u_k}{}^{*}\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k)\right) \tag{2. 19}$$

and

$$\beta_{k-1}(s_{k-1}) = \max_{s_k,u_k}{}^{*}\left(\beta_k(s_k) + \gamma_k(s_{k-1}, s_k)\right), \tag{2. 20}$$

where the max*(.) operation is defined as

$$\max^{*}(\delta_1, \delta_2) = \max(\delta_1, \delta_2) + \log(1 + e^{-|\delta_1 - \delta_2|}). \tag{2. 21}$$

Finally, $L(\hat{u}_k)$ can be obtained by

$$\begin{aligned}L(\hat{u}_k) = &\max_{\substack{(s_{k-1},s_k)\\ u_k=+1}}{}^{*}\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right) \\ &- \max_{\substack{(s_{k-1},s_k)\\ u_k=-1}}{}^{*}\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right).\end{aligned} \tag{2. 22}$$

The performance of Log-MAP algorithm is identical to that of MAP algorithm. However, the complexity is also increased compared with Max-Log-MAP algorithm since computing f_c(.) still involves complicated exponentiations and multiplications. Thus, the values of f_c(.) are usually stored in a pre-computed table and Log-MAP algorithm can be implemented by table look-up. It has been found that excellent performance can be obtained with 8 stored values and $|\delta_1$-$\delta_2|$ ranging between 0 and 5, and no improvement is achieved with a finer representation [10].

2.2.3 The Max-Log-MAP algorithm

As we can see, the MAP algorithm involves too many exponentiations and

multiplications. These are quite complex for hardware realization. Thus, an approximation of MAP algorithm termed Max-Log-MAP algorithm [24] was derived for simple implementation of MAP decoders. Instead of calculating e^{γ_k}, e^{α_k}, and e^{β_k} directly, all computations are done in logarithm domain. Here we define γ_k, α_k, and β_k as transition metric, forward path metric and backward path metric respectively. γ_k can be formulated as

$$\gamma_k(s_{k-1}, s_k) = \log P(s_k, \boldsymbol{r}_k \mid s_{k-1}). \tag{2. 23}$$

Similarly, referring to (2. 4), α_k and β_k can be expressed as

$$\alpha_k(s_k) = \log P(s_k, \boldsymbol{r}_{j<k}) \tag{2. 24}$$

and

$$\beta_{k-1}(s_{k-1}) = \log P(\boldsymbol{r}_{j>k} \mid s_k) \tag{2. 25}$$

respectively. After substituting (2. 17), (2. 18), and (2. 19), $L(\hat{u}_k)$ in (2. 15) can be re-written as

$$L(\hat{u}_k) = \log \frac{\sum_{\substack{(s_{k-1}, s_k) \\ u_k=+1}} \exp\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right)}{\sum_{\substack{(s_{k-1}, s_k) \\ u_k=-1}} \exp\left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right)}. \tag{2. 26}$$

By utilizing the approximation of

$$\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) \approx \max(\delta_1, \delta_2, \cdots, \delta_n), \tag{2. 27}$$

$L(\hat{u}_k)$ can be further simplified to

$$\begin{aligned} L(\hat{u}_k) = & \max_{\substack{(s_{k-1}, s_k) \\ u_k=+1}} \left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right) \\ & - \max_{\substack{(s_{k-1}, s_k) \\ u_k=-1}} \left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)\right). \end{aligned} \tag{2. 28}$$

This computation consists of forward and backward recursions that repetitively compute the α_k and β_k, and can be expressed by

$$\alpha_k(s_k) = \max_{s_{k-1}, u_k} \left(\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k)\right) \tag{2. 29}$$

and

$$\beta_{k-1}(s_{k-1}) = \max_{s_k, u_k} \left(\beta_k(s_k) + \gamma_k(s_{k-1}, s_k) \right). \quad (2.30)$$

Both equations are add-compare-select (ACS) operations, which are similar to the path metric updating of Viterbi algorithm.

2.2.4 SNR sensitivity of Max-Log-MAP and Log-MAP algorithm

Referring to (2.13) and its followed deductions, it's evident that both MAP and log-MAP algorithm requires SNR estimation to obtain the value of channel reliability, i.e. L_c. Unfortunately, accurate estimation cannot be achieved easily. Several papers have discussed the effect of SNR mismatch in turbo decoding. In [25], the simulations show that about -3 to +6 dB SNR estimation offset is tolerable before significant performance degradation. However, Max-Log-MAP algorithm is able to provide a SNR independent scheme if *a priori* information is initialized with a reasonable value, such as all zero's for each state [26]. Due to the linearity of max(.) operations, the term L_c can be canceled out while computing $L(\hat{u}_k)$. The comparison of Max-Log-MAP and Log-MAP algorithm under different SNR estimation offsets was made in [26].

Although Log-MAP algorithm provides the performance better than that of Max-Log-MAP algorithm, it suffers the risk of serious SNR mismatch offset. Thus, channel characteristics play an important role in practical implementation. It has been concluded in [26] that if channel characteristics change over time, the Max-Log-MAP decoder is suitable to be the constituent decoder in turbo decoding. Otherwise, Log-MAP decoder should be preferable in the aspect of coding gain.

2.3 Sliding Window Approach

As what we described in the previous section, the MAP-based algorithm (including MAP algorithm, Max-Log-MAP algorithm, and Log-MAP algorithm) requires both forward and backward path metric to calculate the log-likelihood ratio. Since the forward and backward recursions start from different initial point, the entire block message has to be received and stored for computing forward and backward recursions. Furthermore, we have to store one of the path metrics of forward or backward recursion and wait for another. These restrictions enlarge the memory requirement for hardware implementation of turbo decoder. For example, the maximum block length of 3GPP standard is 5114, which means 5114 codewords and path metrics should be stored. Besides, long output latency is also introduced. It limits the speed and throughput of turbo decoder design.

The main problem is that long block length can not be divided into several shot sub-blocks immediately, since the lack of boundary path metric of sub-blocks in opposite direction of input sequences will degrade the performance. Thus, a sliding window approach was proposed in [27] and later on in [28] to overcome this drawback. This approach utilizes the fact that the backward path metrics can be highly reliable even without knowing the initial state if the backward recursion goes long enough. Fig. 2.5 and Fig. 2.6 shows the process of this approach in both directions and the detail operating flow is described as follows.

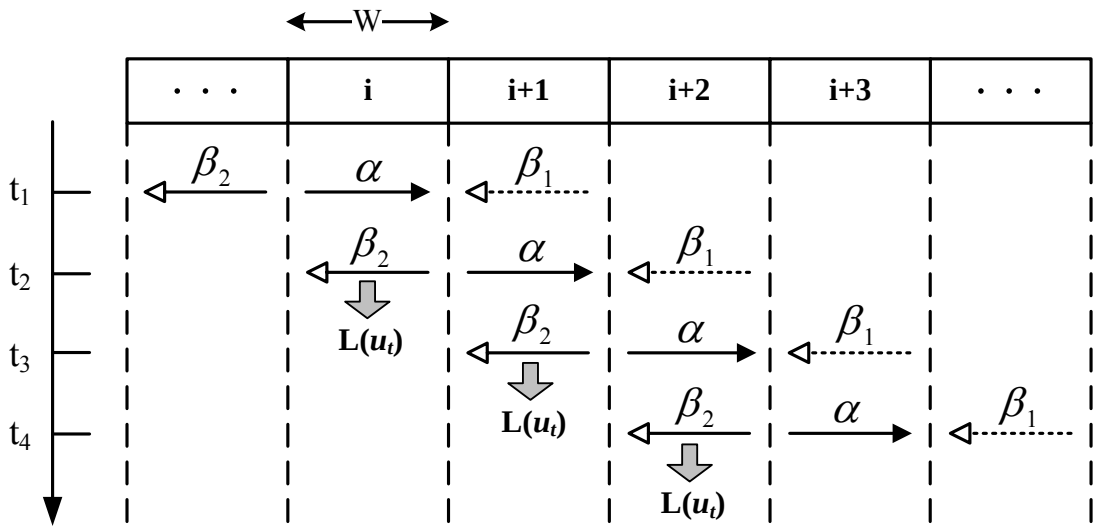


Fig. 2.5 The process diagram of sliding window approach in the forward direction

First, the received codeword is divided into many sub-blocks, with a sub-block length of *W*. *W* is called the convergence length with typically five times the constraint length of the encoder. For each sub-block *i*, the initial path metric values are inherited from the neighbor sub-blocks for both forward and backward recursion operations. Note that in Fig. 2.5 the dummy backward recursion β_1 is employed to obtain the initial path metric values for the true backward recursion β_2. Although the initial condition for β_1 is unknown except the last sub-block, we introduce the equal probability condition for β_1 values:

$$\beta_1(x_t^j) = \frac{1}{M}, \quad \text{for all } j = 0,1,...,M \tag{2. 31}$$

where x_t^j denotes the path metric of *j*-th state at time *t*, the last Trellis section of β_1 , 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 β_1 computation is finished, the initial metrics in the *i*+1 sub-block are available for β_2 metrics in computation, and the corresponding branches metrics in the *i*-th sub-block.

Fig. 2.6 shows the process diagram of sliding window approach in the backward direction. The operation flow is similar to the forward direction type except for two forward recursions α and one backward recursion β.

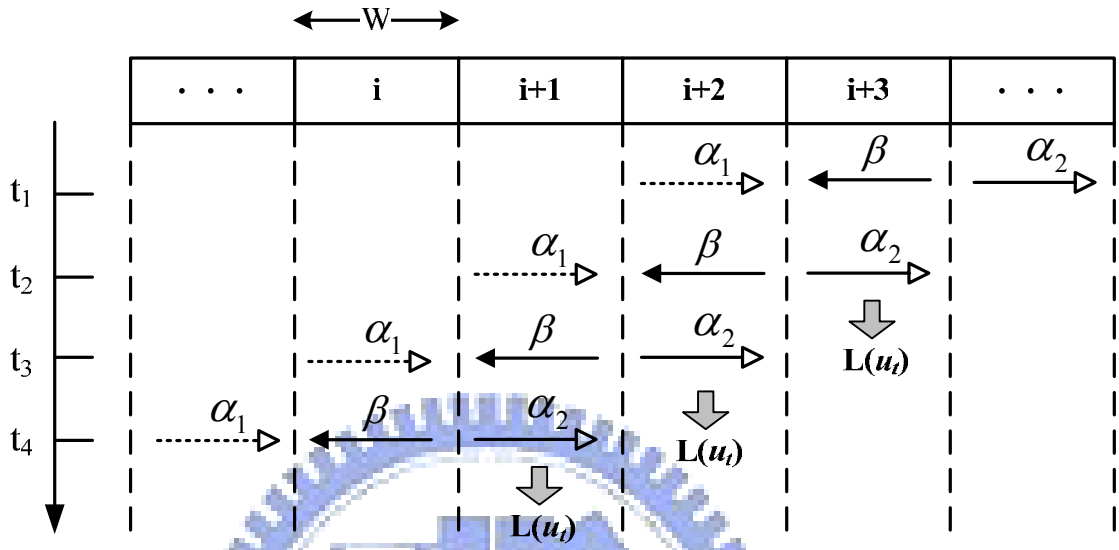


Fig. 2.6 The process diagram of sliding window approach in the backward direction

2.4 Tail-Biting Approach

Tail-biting convolutional codes are first developed by G. Solomon and H. C. A. van Tilborg[5] and recognized as equivalent to quasi-cyclic block codes.[6] From the strict definition of convolutional codes (CCs) it is clear that CCs can only be applied to semi-infinite sequences, i.e., encoding starts at time $t = 0$ in the all-zero state and goes on continuously. But almost any communication system is block-oriented, we must find methods to obtain finite length code blocks of CCs. The standard solution is to add same bits at the tail of information sequences to force the encoder back to the all-zero state. This method can avoid the weak error protection for the last codeword bits, however it causes same rate loss due to tail bits.

Tail-biting avoids the rate loss without suffering from degraded error protection at the end of the codeword. With tail biting technique, the starting state of encoder is not necessarily

the all-zero state. It can also be any one of the other states. The fundamental idea behind tail-biting is that the starting state should be the same as the ending state after encoding the information sequence, i.e., $x_0 = x_N$. In the Trellis representation of tail-biting codes only those paths that start and end at the state are valid codewords.

2.4.1 Encoding tail-biting codes using feedback encoders

Let us consider a feedforward encoder first. It is obvious that we only have to consider the last m input k_0-tuples of information sequences to fulfill the tail-biting boundary condition $x_0 = x_N$. But the situation is more complicated for feedback encoders. The last encoding state x_N depends on the entire information vector $\vec{u} = (u_0, \ldots, u_{N-1})$. Thus, we must calculate for a given information vector $\vec{u}$ the initial state x_0 that will lead to the same state after N cycle. To solve this problem, we consider the state representation:

$$x_{t+1} = \mathbf{A}x_t + \mathbf{B}u_t^T \qquad (2.\ 32)$$

To solve the iterated function by substitution, we can find that the complete solution of (2.32) equals to the superposition of the zero-input solution $x_t^{[zi]}$ and the zero-state solution $x_t^{[zs]}$.

$$x_t = \mathbf{A}^t x_0 + \sum_{\tau=0}^{t-1} \mathbf{A}^{(t-1)-\tau}\mathbf{B}u_\tau^T = x_t^{[zi]} + x_t^{[zs]} = \mathbf{A}^t x_0 + x_t^{[zs]} \qquad (2.\ 33)$$

If we demand that the state as time $t=N$ is equal to the initial state x_0, we obtain from (2.33):

$$x_N^{[zs]} = (\mathbf{A}^N + \mathbf{I}_m)x_0 \qquad (2.\ 34)$$

Where $\mathbf{I}_m$ denotes the m-by-m identity matrix. If a feedback encoder with certain information length N can provide an invertible matrix $(\mathbf{A}^N + \mathbf{I}_m)$, the correct initial state x_0 can be calculated by knowing the zero-state response $x_N^{[zs]}$.

The encoding process of tail-biting convolutional code shown in Fig. 2.9 is divided into

two steps:

First, the encoder starts from the all-zero state with given information sequences to determine the zero-state response $x_N^{[zs]}$. By knowing the zero-state response, we can calculate the corresponding initial state x_0 by (2.34). Second, the encoder starts from the correct initial state x_0 and a valid codeword results.

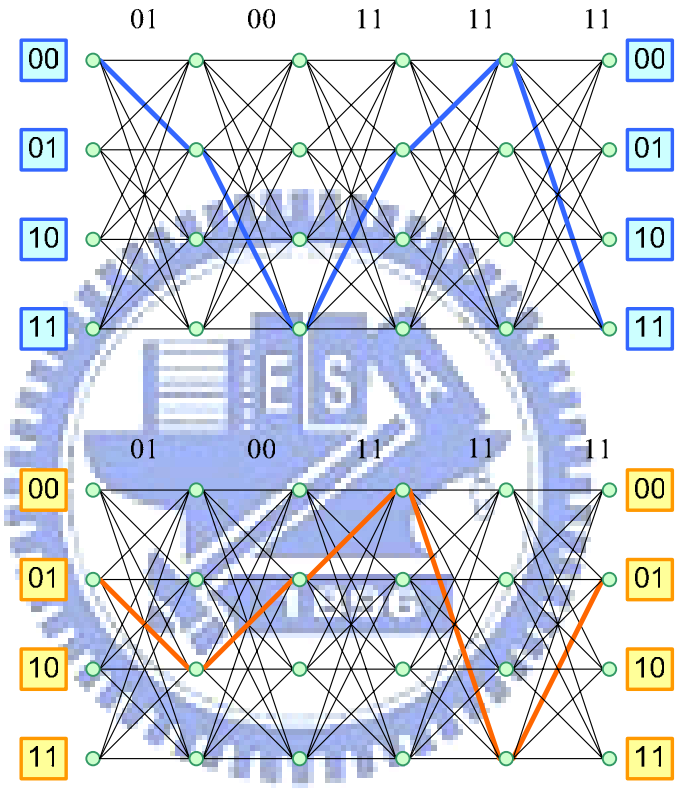


Fig. 2.7 The encoder process of tail-biting convolutional code

Since the matrix $(\mathbf{A}^N + \mathbf{I}_m)$ has to be invertible, not every code length is legal with a given feedback encoder. Moreover, some feedback encoder can not be tail-biting. Some detail discussion can be found in [7], [8], and[9].

Chapter 3
The High Speed Turbo Decoder Design I

3.1 Introduction

Presented by Berrou et al. in 1993 [1], turbo codes have been recognized as a milestone in the channel coding theory. Due to their outstanding error-correcting capabilities, turbo codes have been highly appreciated in wireless communications, where signal-to-noise ratios (SNRs) are generally low. Two commonly used soft-input–soft-output (SISO) turbo decoding algorithms are maximum a posteriori probability (MAP) algorithm [2] and soft-output Viterbi algorithm (SOVA) [4]. MAP-based turbo decoders are known to have better performance than SOVA-based turbo decoders while having slightly larger complexity.

Many researches are proposed to improve the speed of turbo decoder. Bickerstaff proposed a high radix decoder [11]; Bougard introduced a full-duplex design [12]; Urard implemented a 5 iterations series turbo decoder [16]. Their works increase the throughput by refining the architectures of the SISO decoders. The highly parallel structure might be a solution to substantial improvement, but there are two difficulties that have to be overcome. One is the memory contention problem resulted from high-radix and multiple processing elements; the other is the critical path resided in the add-compare-select (ACS) circuit. We proposed a high speed solution that resolves these two problems by using a novel interleaving methods and modifying the MAP decoders. Some interleaving algorithms with contention-free properties have been published [9], and our design adopts the inter-block permutation (IBP) interleaver [13]. Then we exploit a high-radix MAP decoder with shorter

critical path to increase data rate [14]. The proposed turbo decoder provides both high throughput capability and outstanding energy efficiency while maintaining equivalent performance as 3GPP turbo code.

3.2 Decoder Structure

For high speed turbo decoder design, there are generally two types of architectures proposed in the state of the art. Fig 3.1 shows these architectures, the series architecture and the parallel architecture. The series architecture duplicates the same number of processing elements as iterations and each processing element decodes the codeword for only one iteration. After decoding, each processing element will pass the extrinsic value to the next element. This architecture is easy to implement but the hardware cost is very high. The parallel architecture decodes one codeword with multiple decoders. This architecture is more flexible since number of decoders varies from different specifications. The major problem of this architecture is that how to decode a block codeword with multiple decoders. The forward recursion and the backward recursion connect the whole codeword, so we should apply some techniques to separate them. In the following, we will introduce our proposed design using the parallel architecture to solve this problem.

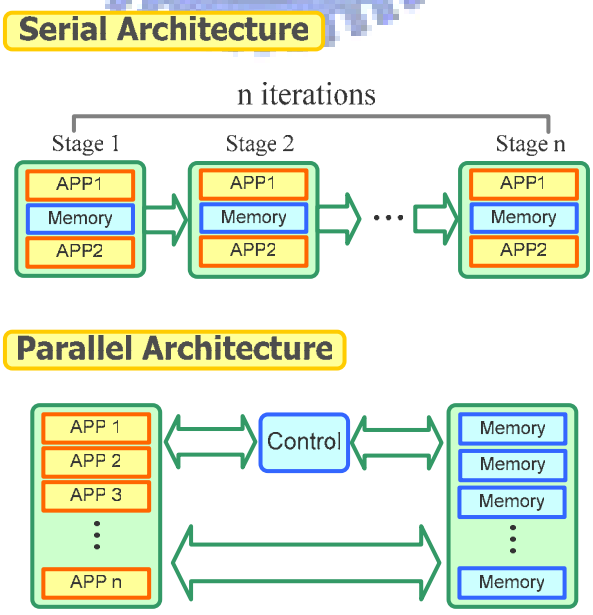


Fig. 3.1 Block diagram of proposed turbo decoder

Fig. 3.2 shows the block diagram of proposed decoder, which consists of 32 parallel MAP decoders and 32 parallel memory sets. We separate a codeword into 32 sub-codewords with length 128. Each sub-codeword is assigned to one decoder and decoded separately. These sub-codewords are connected by a well-designed inter-block permutation (IBP) interleaver. This method avoids the forward and backward recursion problem while using the parallel architecture. The decoding process is described as follows: first, each memory will collect a 128-bit sub-codeword from input buffer till the whole 4096-bit codeword is received. The memory stores the received symbols and extrinsic information, which is divided into two banks to support the radix-4 design. Second, the 32 memories will deliver the required data to the 32 MAP decoders through the IBP network, which is part of the interleaver. 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 is responsible for 128 bits. After 8 iterations, this design would output the decisions of current block and start to decode next block.

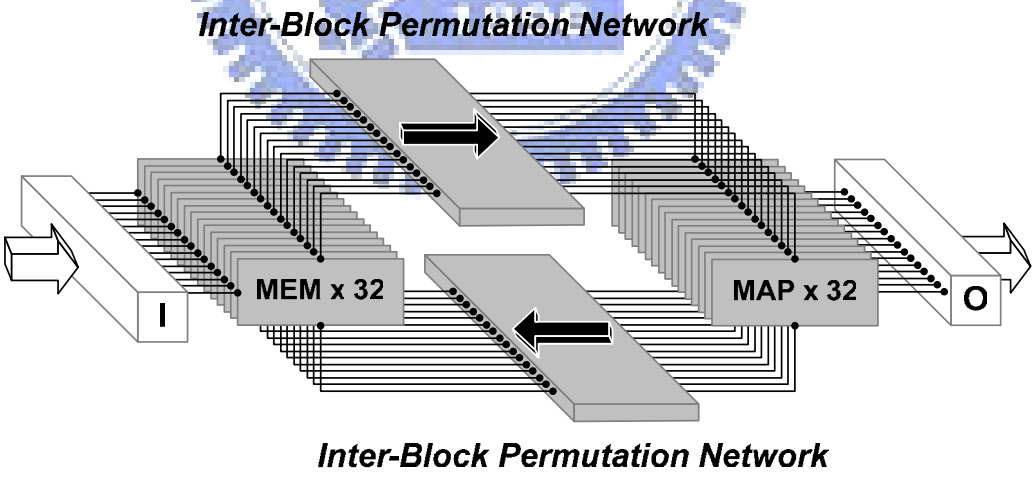


Fig. 3.2 Block diagram of proposed turbo decoder

3.3 *Interleaver Design for High Speed Turbo Code*

3.3.1 Contention-free Interleaver

To increase throughput, a log-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. Interleaving latency is eliminated by writing the M values generated each clock cycle directly to their interleaved positions. However, if the interleaver is not designed carefully, two or more MAP-based decoders may require access to the same memory bank on a given clock cycle, resulting in a memory contention. Moreover, a high radix decoding structure also suffers from the memory contention problem while accessing multiple codeword symbols from memories. Fig 3.3 shows an example of memory contention problem in a parallel decoding structure. We store a codeword sequence 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 different interleavers. 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 contention collisions at time t_0 and t_3.

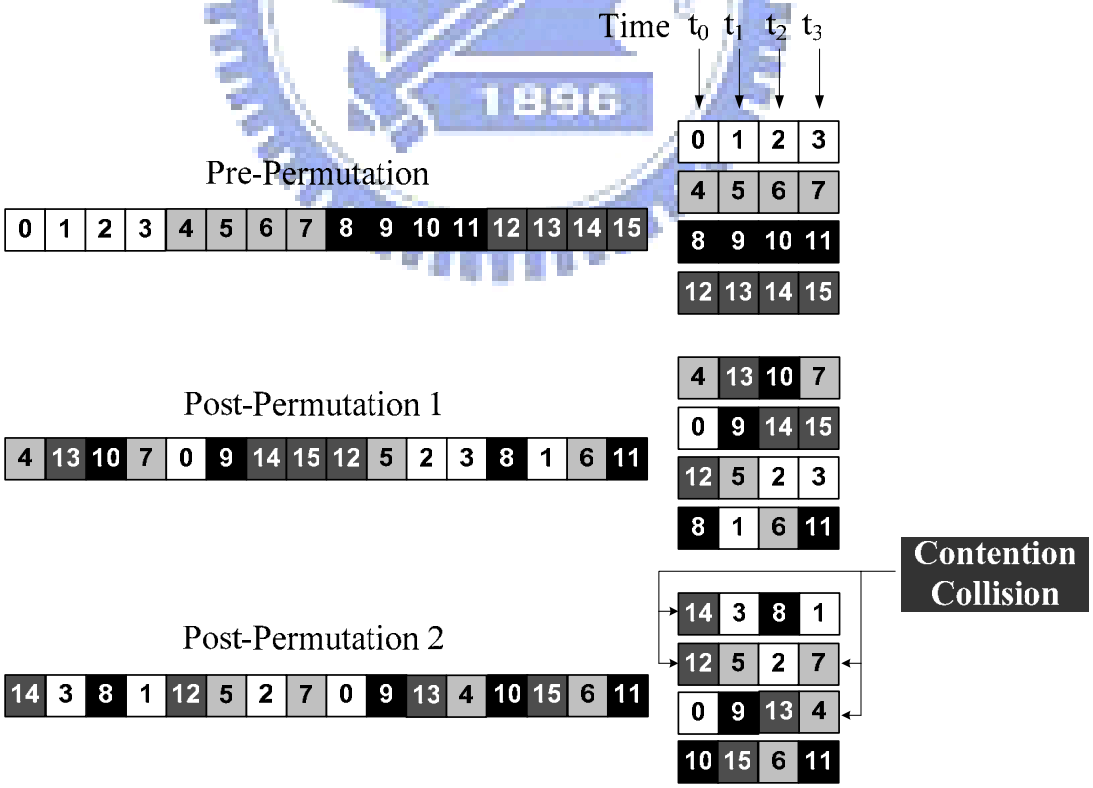


Fig. 3.3 Example of a contention-free permutation

3.3.2 IBP Interleaver

The IBP interleaver in [13] favors both performance and throughput of turbo decoder. Such method guarantees no hazards when multiple MAP decoders try to access multiple memories concurrently. The IBP interleaver consists of two steps of permutation: intra-block permutation and inter-block permutation. The first step rearranges the symbol sequences in each sub-block with the same rule. The second step swaps the sequences between blocks periodically. The destination can be derived by executing bit-wise exclusive-or between the original block index and the IBP parameter. Fig. 3.4 demonstrates an example of IBP interleaver with four sub-blocks. First, all sub-blocks are individually reordered by right rotate; Second, they exchange data among these permuted sequences.

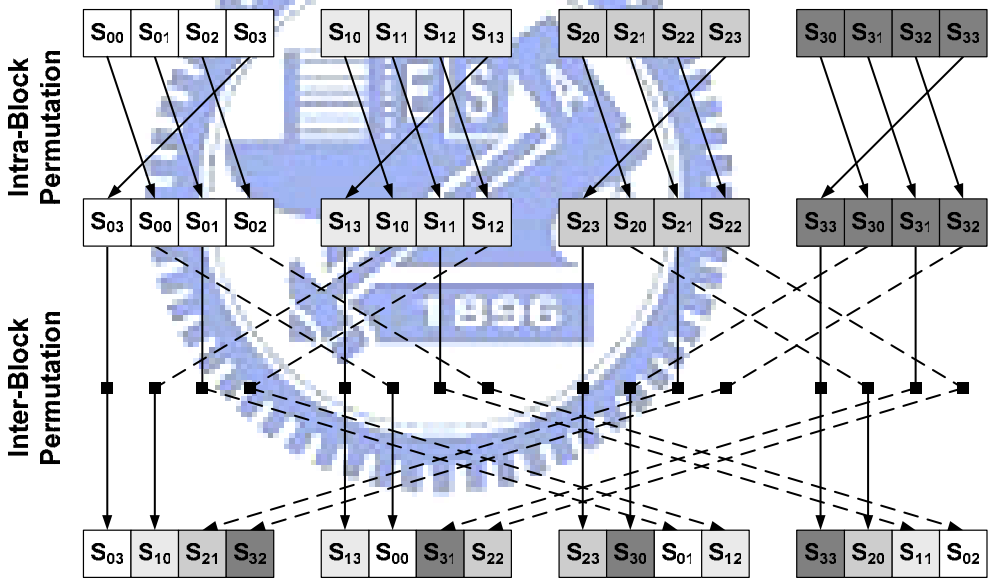


Fig. 3.4 An example of IBP interleaver with four sub-blocks

3.3.3 Butterfly network

The butterfly network is designed to perform the inter-block permutation in the IBP interleaver. This structure also avoids the memory contention problem between sub-blocks and reduces the circuit complexity. Fig. 4 shows the corresponding structure for above example illustrated in Fig. 3. The network is divided into two levels, and each level has one external signal to control the multiplexers. S0 and S1 will define four possible connections. In

general, the butterfly network links N memories to N MAP decoders by $\log_2 N$ levels of switches. Each level requires 1-bit control signal to manage its N multiplexers; the total $\log_2 N$ bits establish N possible connections.

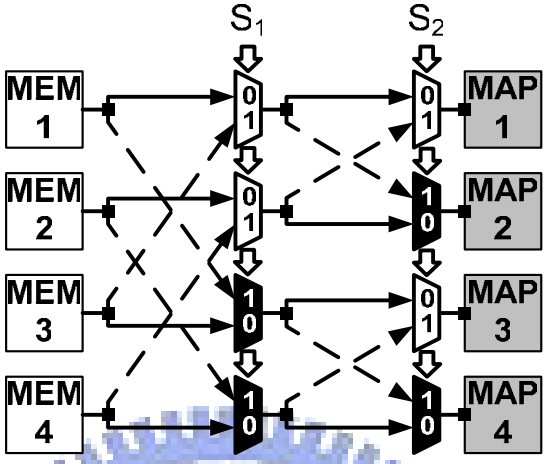


Fig. 3.5 A 4x4 butterfly network for IBP interleaver

3.3.4 Double prime interleaver

All the data inside each block will be divided into two groups and be stored in the two separate memory banks. When radix-4 MAP decoders request two symbols at each cycle, these two symbols must be derived from different memory banks. This is another contention problem that should be aware of. Our design uses the double prime interleaver to resolve this problem. The double prime interleaver is constructed by two prime interleavers whose function are expressed by

$$\pi(i) = \begin{cases} ((\lfloor i/2 \rfloor \times p) \bmod \frac{L}{2}) \times 2+1, & i \text{ is odd} \\ ((\lfloor i/2 \rfloor \times p+s) \bmod \frac{L}{2}) \times 2, & i \text{ is even} \end{cases} \quad (3.\ 1)$$

This L is the block length, and it must be an even number. Note that p must be relative prime to L/2 and s is a constant shift. Both the interleaver and de-interleaver could be expressed in (3.1) with different parameters. Double prime interleaver with well-searched parameters would outperform the interleaver in 3GPP turbo coding. Most important of all, an well-designed double prime interleaver is an fully contention-free interleaver for certain

sub-block length. For example, we can choose any factor of the sub-block length as the parallel access number and the memory bank number. It is guaranteed that a well-designed double prime interleaver is a contention-free interleaver.

3.4 High-Throughput MAP Decoders

3.4.1 Retimed radix-2x2 ACS unit

For trellis-based decoders, the branch number of conventional high-radix design increases exponentially however the branch number of the two-stage structure increases linearly. A two-stage ACS is introduced in [14] to ease the area overhead of high-radix ACS. The complexity of ACS unit depends on the branch number, so our design prefers radix- 2 × 2 ACS to radix-4 ACS. But the critical path of two-stage structure is longer than conventional structure. The recursive property of path metric would make the pipelining method inefficient here, however, the critical path can be reduced by our proposed retiming method.

It is obvious that the ACS unit could not execute compare-select operations until addition results are ready; such data dependency restricts the operating frequency. To eliminate the dependency, the data path of ACS unit must be modified. So the proposed decoder applies the retiming technique, and Fig. 3.6 demonstrates the procedure of a retimed radix-2× 2 ACS. The first step shown in Fig. 3.6(a) is retiming of registers. Move and duplicate the registers ahead of the compare circuits, then computation order is rearranged from add-compare-select to compare-select-add. The registers have to store the summation of path metric and branch metric rather than only path metric. The second step shown in Fig. 3.6(b) is relocation of adders. Move and duplicate the adders ahead of the multiplexers; now the compare-select and addition could execute concurrently. The modified ACS unit is shown in Fig. 3.7, where the critical path becomes two consecutive compare-select operations. It would cause extra area overhead because of double registers and double adders, and the improvement of the radix-2×2 architecture could compensate for this loss. The relocated method can accomplish

not only high-speed but area-efficient solution.

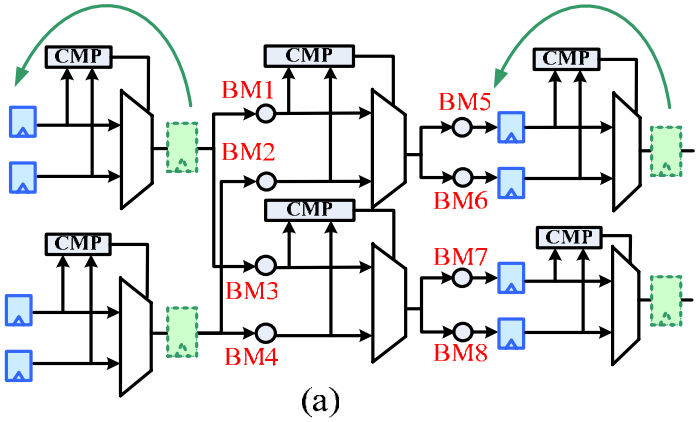


Fig. 3.6 Retiming procedure of a radix 2x2 ACS unit

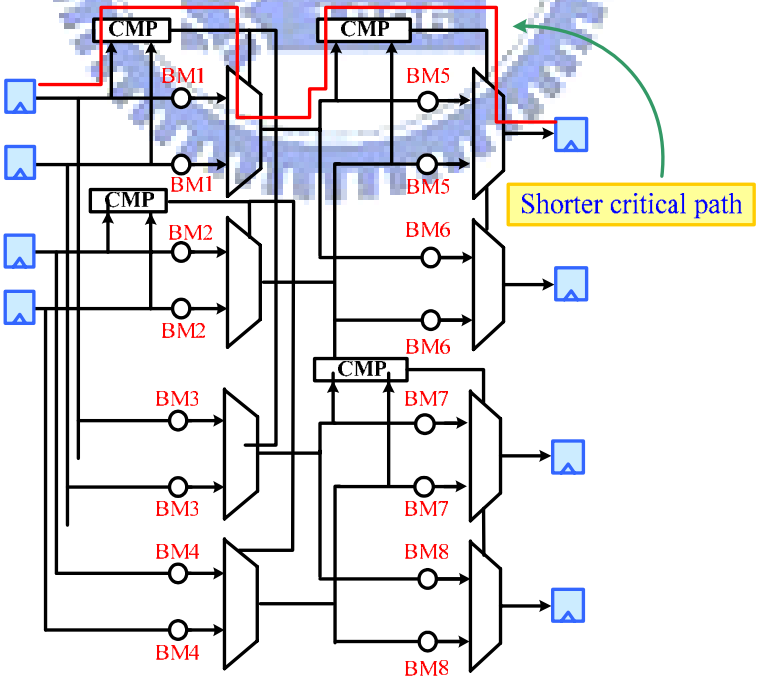


Fig. 3.7 A retimed radix 2x2 ACS unit

3.4.2 The circuit for log-likelihood ratio calculation

Our design adopts the modulo normalization to avoid over- flow of path metric [15]. This method requires only one more bit in the ACS unit and a simple modification inside the LLR unit; there are no specific circuits for normalization in ACS unit. Only the differences between forward path metrics and the differences between backward state metrics are significant in modulo normalization, so the LLR unit has to use these differences to calculate the log-likelihood value. Our design rearranges the computation order of log-likelihood value from circuit in Fig. 3.8(a) to circuit in Fig. 3.8(b). Although the two circuits have the same function, but 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.

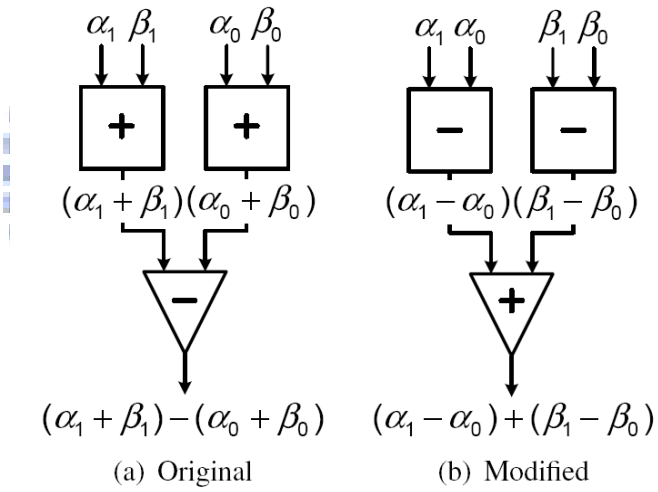


Fig. 3.8 The circuit for log-likelihood calculation

3.5 Simulation Result and Chip Implementation

The proposed turbo code with code rate 1/2 could decode 4096 bits after 8 iterations, and the implementation applies maximum log-MAP algorithm with a scaling factor 0.75. The other specifications are listed in Table. 3.1. Fig. 3.10 and Fig. 3.11 shows the performance comparison between the proposed code and 3GPP turbo code. The floating point and the fixed point simulation result are both competitive to the result of 3GPP standard. However, the proposed turbo design has better distance property due to the interleaver design than the 3GPP

standard. Obviously, the 3GPP standard suffers from the error floor phenomenon more than the proposed design.

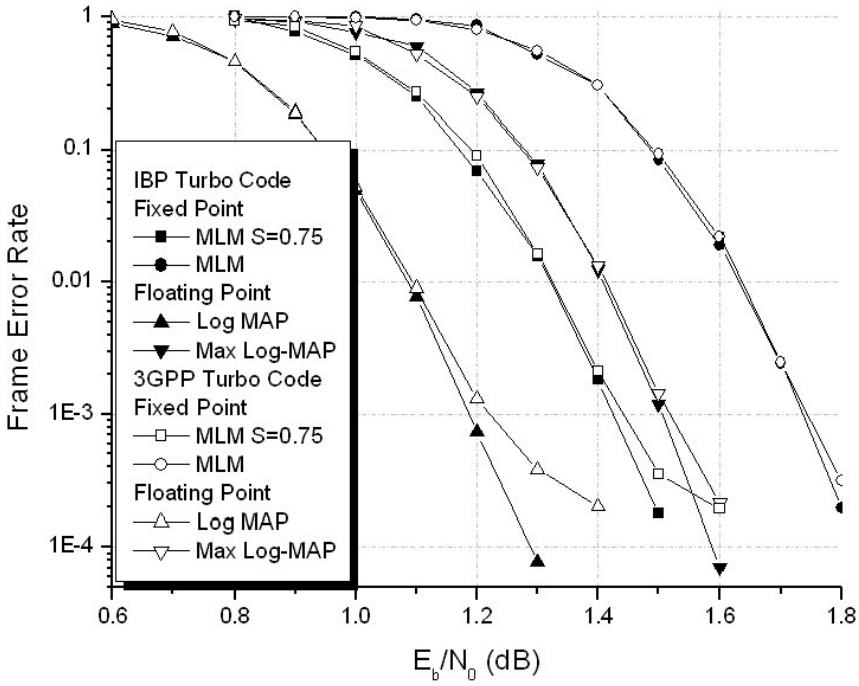


Fig. 3.9 FER performance compared with 3Gpp turbo code

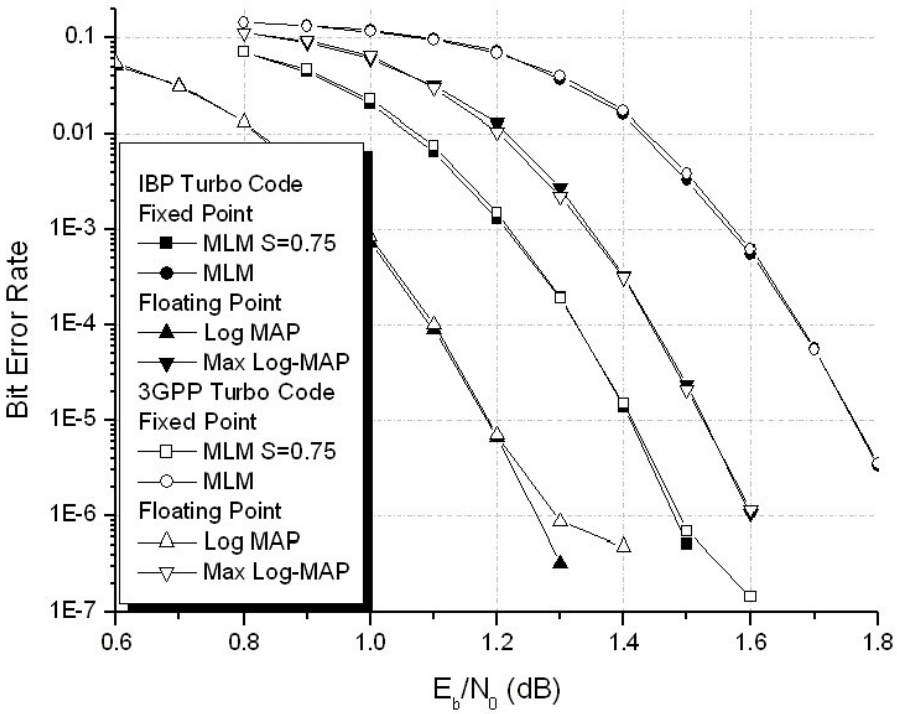


Fig. 3.10 BER performance compared with 3Gpp turbo code

Table 3.1 Turbo Decoder Specification

Algorithm	**Max-Log MAP**	
ACS unit	**Radix 2x2 (retimed)**	
Code polynomial	$\left[1 \quad \frac{1+D+D^3}{1+D^2+D^3}\right]$	
Interleaver	**IBP interleaver (p, s) = (15, 23)**	
Sliding Window	**32**	
Code Rate	**1/2 (punctured)**	
Block length	**4096(128 x 32)**	
Quantization	**6 bits (3.3)**	
iteration	**8**	
Scaling Factor	**0.75**	
Note	**Tail-Biting**	
Technology	**0.13um 1P8M**	
Clock rate	**250MHz /w DLL**	**80MHz/wo DLL ***
Throughput	**500Mbps**	**160Mbps ***
Gate count	**2.67M**	
Core Area	**17.8 mm2**	
power	**762mW**	**275mW ***
nJ/bit • iteration	**0.19**	**0.22 ***

The decoder chip is fabricated with a 0.13μm 1P8M CMOS technology, and the die photo is shown in Fig. 3.12. The core area is 17.8mm^2 with 2.67M gates count, including the 3.33mm^2 memory block. A delay lock loop (DLL) circuit is applied to generate internal clock

source as four times the external frequency. The design could operate at 250MHz with the help of DLL during post-layout simulation, due to the relocation technique. However, the DLL could not work as expected during measurement. The test chip could achieve 160Mb/s and 275mW power consumption with 1.32V supply. For the decoder with 8 iterations, the energy efficiency is 0.22nJ/b/iter. Table II lists the comparison of the proposed code with other published works, and the proposed design has the optimal energy efficiency [11][12][16].

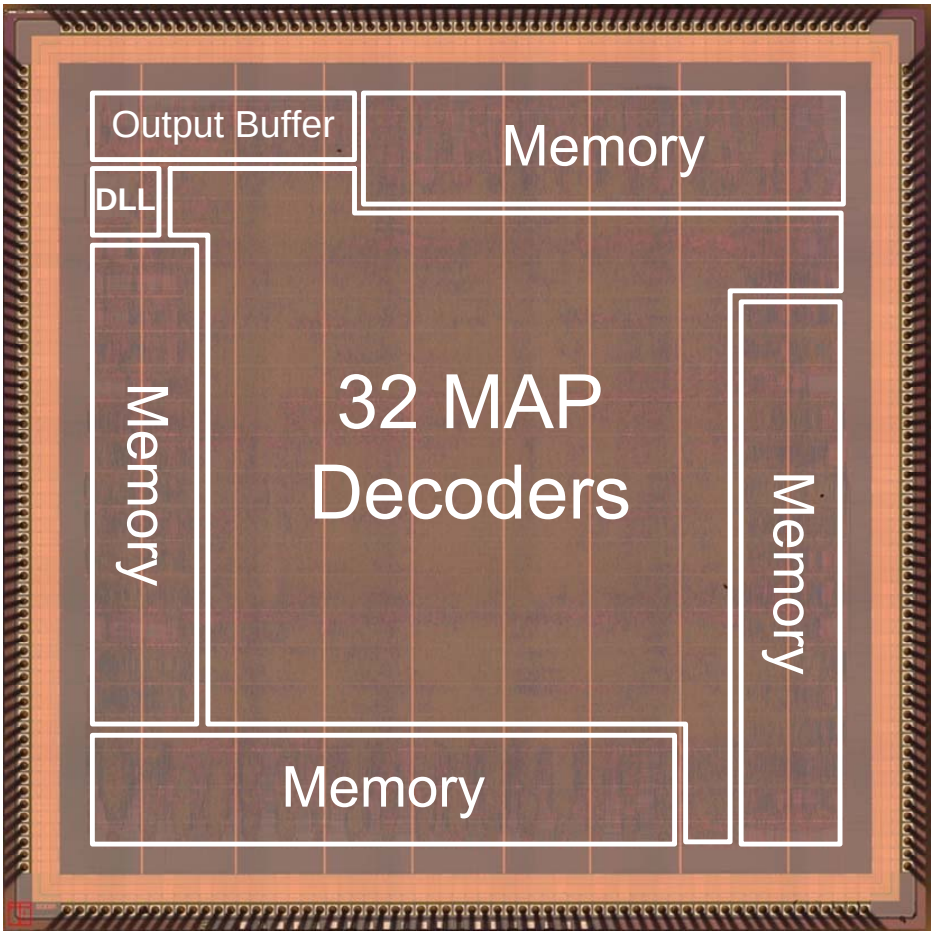


Fig. 3.11 Micro photo of proposed turbo decoder chip

Table 3.2 Comparison with Other Turbo Decoder

	proposed	[11]	[12]	[16]
Technology	0.13 μ m	0.18 μ m	0.18 μ m	0.13 μ m
Clock rate	80MHz	145 MHz	160 MHz	352 MHz
Throughput	160Mbps	24 Mbps	71.7 Mbps	352 Mbps
Block Size	4096	5114	384	2048
Core Area	17.8mm^2	14.5 mm^2	7.16 mm^2	10 mm^2
power	275mW	1450mW	N/A	2464mW
Energy Efficiency	0.22 nJ/bit・iter	10.0 nJ/bit・iter	9.7 nJ/bit・iter	1.4 nJ/bit・iter

3.6 Summary

The proposed turbo decoder with the parallel architecture enables multiple processing elements to decode one codeword concurrently. The proposed IBP interleaver connects all processing elements in the parallel architecture and avoids the limit of the forward and backward recursions. We also introduce a high speed methodology for high radix decoder structure. The combination of two stages ACS and the retiming technique efficiently speed up the decoding throughput with acceptable hardware cost. The energy efficiency of proposed turbo decoder is much smaller than that of the state of the art.

Chapter 4
The High Speed Turbo Decoder Design II

In chapter 3, we have introduced a power efficient turbo decoder design with 32 processing elements. The throughput of the proposed design is about 500Mbps in pre-layout simulation. The critical path of the proposed design is the ACS units. However, the throughput of a radix 2x2 ACS unit working under 250MHz is 500Mbps and the total throughput of the decoder should be 1Gbps with 32 processing elements under 8 iterations. The total throughput is reduced by the following two issues:

- One block is calculated twice due to the tail-biting. The calculation of α recursion of first block introduces a dummy sub-block and reduces the throughput.
- Due to the iterative decoding and the interleaver of turbo code, the decoder must stop and wait until the processed data stored in the memories. This data hazards happen twice per iteration between two different decoding rounds.

These issues will be discussed in detail in the following sections. We will propose methods to solve these problems and implement a 1Gbps high throughput and power efficient turbo decoder.

4.1 Introduction

4.1.1 Data Hazards

There is an iteration bound occurred in the MAP-based decoder structure, so the forward and backward recursion in a turbo decoder is always the critical path and occupy a large area in the implementation. This is a main reason that the hardware of the forward and backward

recursion is always reused. The cycle-based decoding procedure is shown in Fig. 4.1. This example shows a sub-block size 16 and a radix 4x4 decoder decodes 4 symbols each cycle. It shows that the forward and backward recursion modules and the LLR module are reused for four cycles. Furthermore, the pipelined method can be used while decoding different sub-blocks because there is no data dependency between different sub-blocks in the same decoding round. Fig. 4.2 shows the case we proposed in chapter 3 and a data hazard happens while decoding. The data dependency results from the interleaver between sub-block 4 in the pre-decoding round and sub-block 1 in the post-decoding round. The extrinsic information of sub-block 4 in pre-decoding round may be used in sub-block 1 in post-decoding round. This is the reason why the decoder should be idle until the extrinsic information stored in the memories.

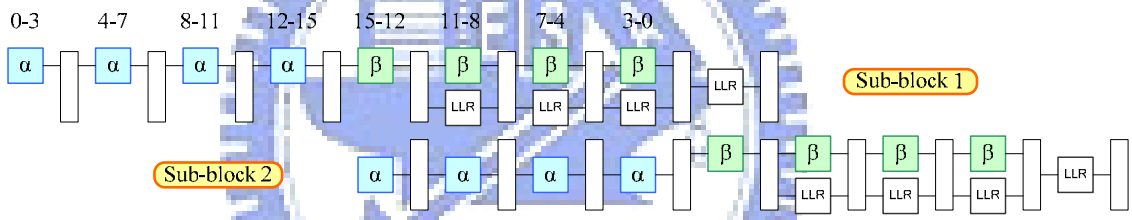


Fig. 4.1 A cycle-based decoding procedure

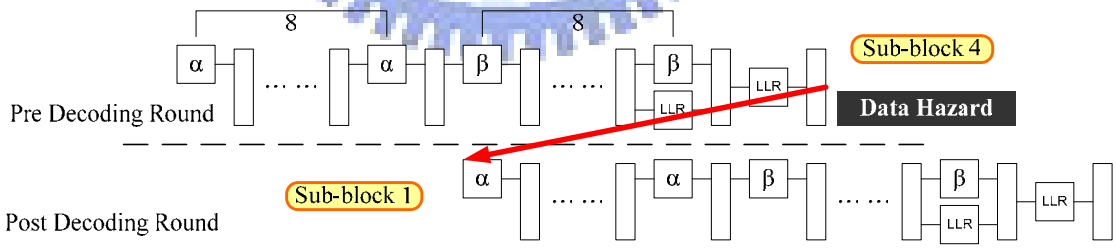


Fig. 4.2 A data hazard occurred while decoding

4.1.2 A dummy sub-block

A valid codeword in the tail-biting Trellis makes the encoder to start and end at the same state, instead of zero state only. Therefore, a dummy sub-block, as well as the last sub-block, will be calculated first to estimate the initial value of the forward recursion of the first sub-block. Decoding schedule of a sub-codeword is shown in Fig. 4.3 and Fig. 4.4. We can

easily find that the data hazard and the dummy block make the decoding procedure longer. It takes 128 cycles to decode a sub-codeword and the utilization of the hardware is 50% only. The decoder is idle for 12 cycles and some modules are idle while other modules are calculating. Therefore, the working duration of the forward and the backward recursion modules and the LLR module is 64 cycles.

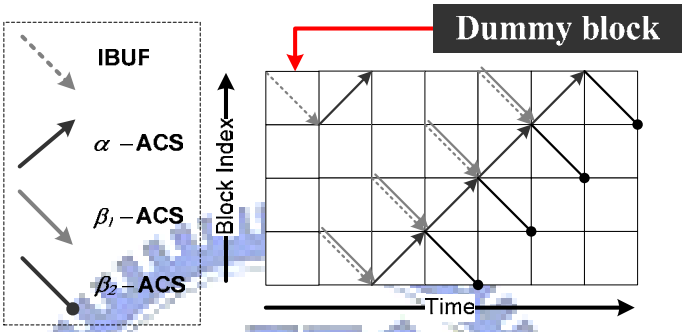


Fig. 4.3 Decoding schedule of a sub-codeword

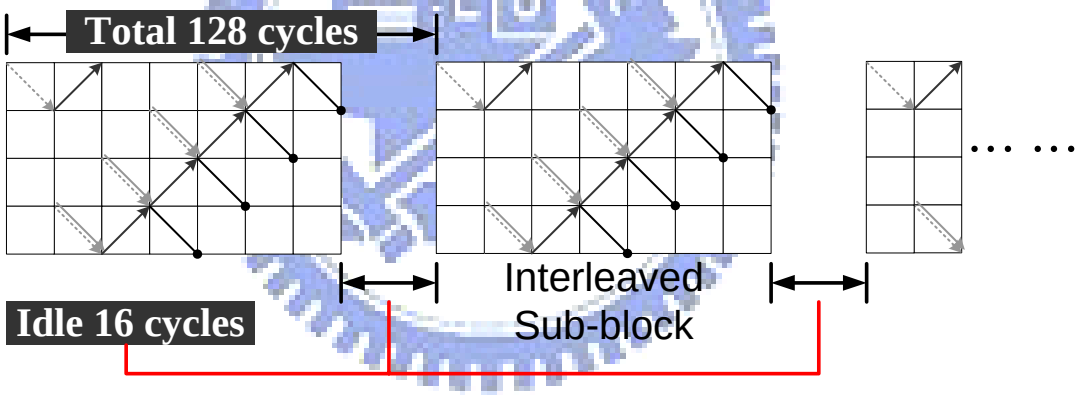


Fig. 4.4 Decoding schedule of previous design

4.2 Decoding Schedule

The data hazard and the dummy sub-block cause a 50% degradation of the throughput. In this section, we will propose a method to solve this problem and make the 100% utilization of the hardware.

4.2.1 Decoding with two codewords

Due to the data dependency of pre-decoding round and post-decoding round connected by the interleaver, a better way to break this relation is to decode two codewords alternately.

The proposed method achieves 100% hardware utilization without any extra logic cost. The only cost of this method is that we have to store two codewords in the memories. The detail procedure in Fig.4.5 is described as follows:

- Decode from the first sub-block and get the initial value of forward recursion of the first sub-block from the previous iteration. If it is the first iteration, then set an all zero initial value for beginning.
- Store the initial value needed by the next iteration, so it is not necessary to calculate the dummy sub-block.
- First, decode the pre-permutation sequences of sub-codeword A.
- Second, decode the pre-permutation sequences of sub-codeword B.
- Third, decode the post-permutation sequences of sub-codeword A.
- Decode the post-permutation sequences of sub-codeword B.
- Then decode alternately until the last iteration.

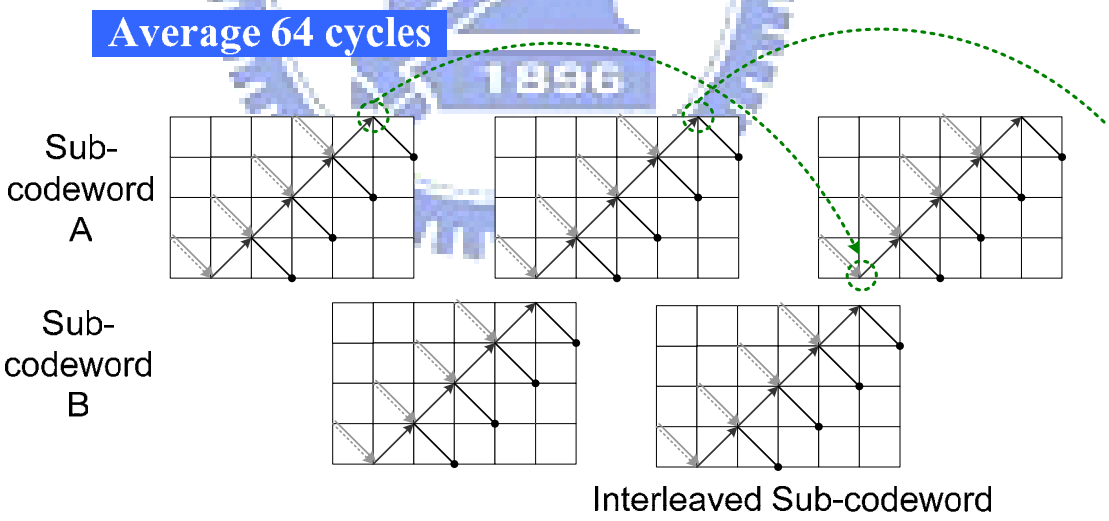


Fig. 4.5 Decoding schedule with two codewords

There are two more steps should be noticed about the dummy block:

- While decoding each sub-codeword, decode from the first sub-block and get the initial value of forward recursion of the first sub-block from the previous iteration. If it is the first iteration, then set an all zero initial value for beginning.

- Store the initial value needed by the next iteration, so it is not necessary to calculate the dummy sub-block. The dot line in Fig. 4.5 shows where we store the initial value and where we read the initial.

The fundamental idea of our proposed method is to keep the hardware calculating and avoid to calculate the same sub-block twice. Notice that at any timing frame all hardware modules are working, which means the hardware utilization reaches 100%. Applying the method, we can double the throughput by reducing the decoding cycles from 128 to 64 for each sub-codeword, but the extra storage of initial values is about 6144 bits in the case of our proposed design in chapter 3.

4.3 MAP Decoders

4.3.1 The structure of each processing element.

It is mentioned in section 3.4 that the number of the processing elements and the throughput of each element are two main factors of the total throughput. In addition to adding the number of processing elements, the throughput of each element should increase for a high throughput turbo decoder design. The method we used in the new proposed design is a higher radix Trellis structure.

For any Trellis-based decoder, two important factors should be considered carefully are the number of states and the branch number of each state, which affect the implementation complexity numerously. While applying a high radix design, another dimension should also be taken into account is the stage number of Trellis. In Fig. 4.6, both radix 16 and radix 4x4 Trellis diagram merge 4 stage Trellis diagram into one. The radix 16 Trellis has 16 branches for each state. The radix 4x4 Trellis has 4 branches for each state, but it is a two stage structure. However, the total branch number of the radix 4x4 is half of the radix 16. Therefore, the hardware of radix 16 is twice as that of radix 4x4.

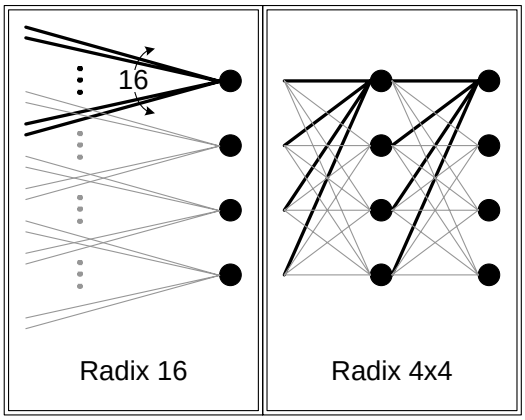


Fig. 4.6 Radix 16 and radix 4x4 Trellis diagram

Fig. 4.7 shows the hardware cost of two structures. We can find that the number of comparators and multiplexers of the radix 4x4 structure is twice as that of the radix 16 structure, but the complexity of a 4 to 1 comparator is much smaller than that of a 16 to 1 comparator. Besides, the branch number of the radix 4x4 is less than that of the radix 16. The new proposed design uses the radix 4x4 structure in each processing element.

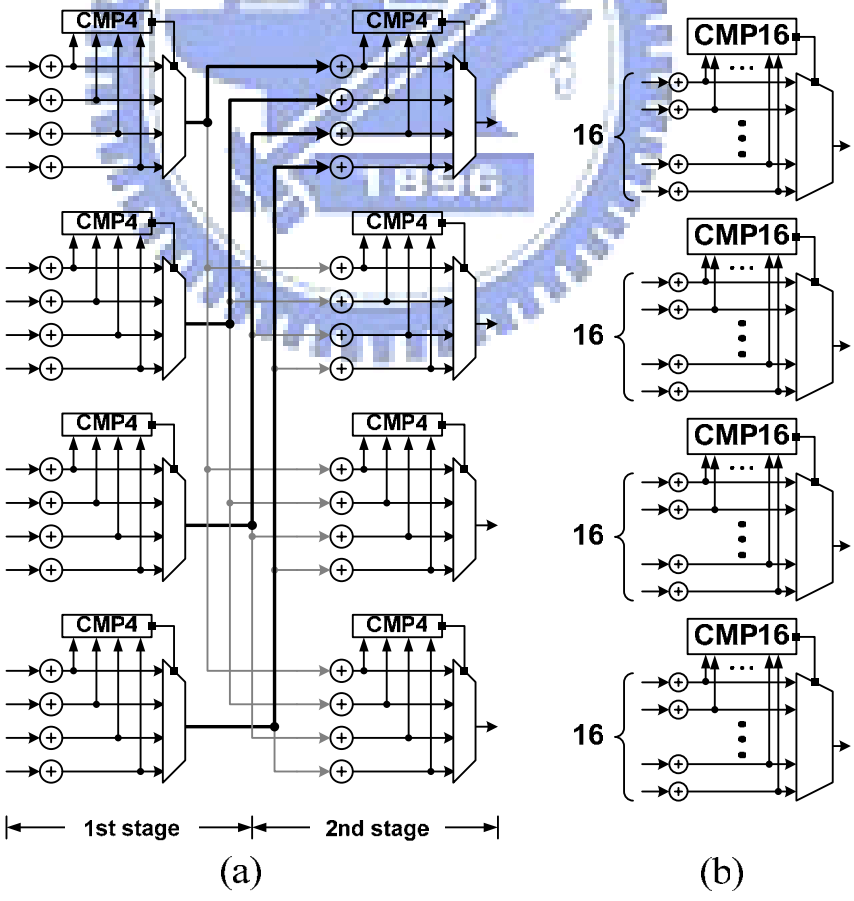


Fig. 4.7 Circuit diagrams of two structures

4.3.2 The memory units

Considering the radix 4x4 structure and the storage of two codewords, the memory units of the turbo decoder should be redesigned. First, the memory should be divided into four banks and each bank consists of five sub-banks. The division of the memory units is due to the bandwidth and contention. We have to access four input symbols for the processing element at each cycle and each symbol consists of information bits, parity bits and the extrinsic part. Fig. 4.8 shows one memory unit in detail. Notice that each bank in the memory unit is the same as that mentioned in chapter 3, but the number of banks is double and the bandwidth is also double. Furthermore, the total storage is double because of the additional codeword B.

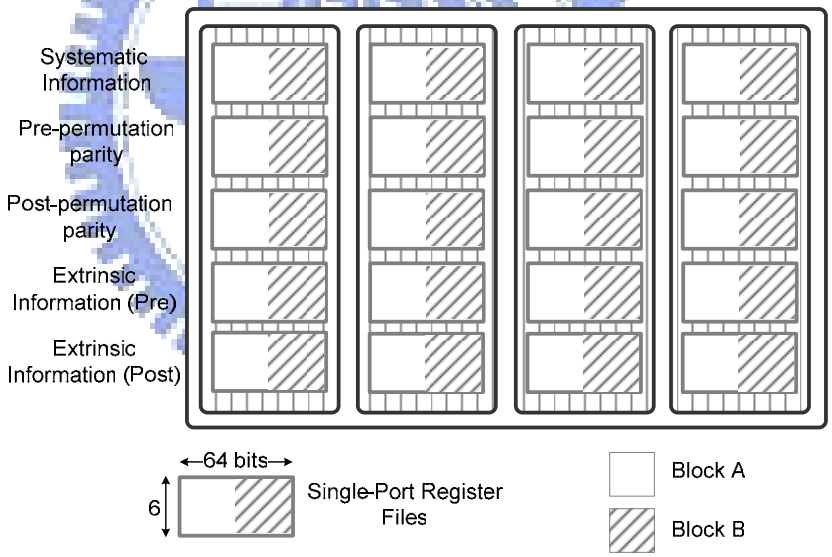


Fig. 4.8 The memory unit

4.3.3 Retime or not retime

In chapter 3, we have introduced a retiming technique to shorten the critical path for a two stage ACS structure. The retimed structure has more hardware costs than the no retiming version. The critical path comparison between these two versions is shown in Fig. 4.9, and the target technology of our implementation is the UMC 90 nanometers process. Obviously, the

retimed version has higher clock rate. But there are other important issues that should be considered in advance technology, such as wire delay. In advance technology, the wire delay dominates and the crosstalk phenomenon will be more critical. A popular solution for a large design is to reduce the routing complexity and the wire length. Table 4.1 shows the comparison between two versions. The area and the routing complexity of the retimed vesion are bigger than the no retiming one. Thus, the critical path of retimed version will grow faster than no retiming one due to the routing congestion problem. Therefore, our proposed 1Gbps turbo decoder chooses the no retiming radix 4x4 ACS structure for the processing elements. With the decoding schedule introduced in section 4.2, the utilization of all module in each processing element achieves 100%. The throughput of elements is 1Gbps each.

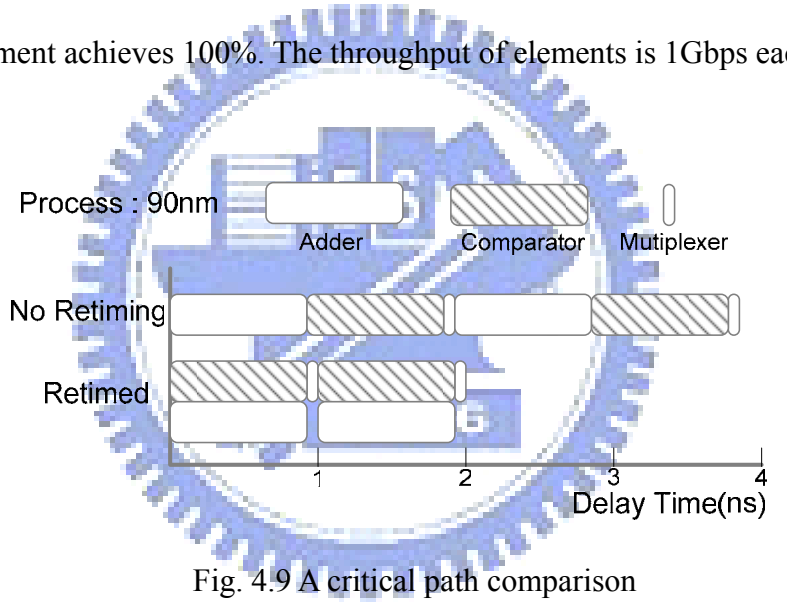


Fig. 4.9 A critical path comparison

Table 4.1 Comparison between two versions

90 nm Technology	**Radix 4x4(Retiming)**	**Radix 4x4**
Throughput	**1600Mbps**	**1000Mbps**
Area	**High**	**Medium**
Routing Complexity	**High**	**Medium**
Frequency	**400MHz**	**250MHz**
Critical Path	**2.5ns(pre-layout)**	**4ns(pre-layout)**

4.4 *Interleaver Design*

In the proposed 1Gbps turbo decoder, we reuse the IBP interleaver and remain the parameters. However, the number of processing elements reduces to 16 and the throughput of each processing element reaches to 1Gbps. The sub-codeword size remains 128. Fig. 4.10 shows the differences between the decoder described in chapter 3 and the proposed 1Gbps turbo decoder. Fig. 4.10(a) is the original architecture in chapter3, and Fig.4.10(b) reduces the decoder number by half. Thus we can combine the memory unit 1 and 2 with 3 and 4, shown in Fig.4.10(c). The original S_1 switch can be fixed to zero and the control signal S_1 will be passed to the address generators of memory units. The combination of memory units will make the total area of memory smaller and easier for placement. The flexibility of the IBP interleaver makes the variation of design possible without any hardware cost and any extra control overhead.

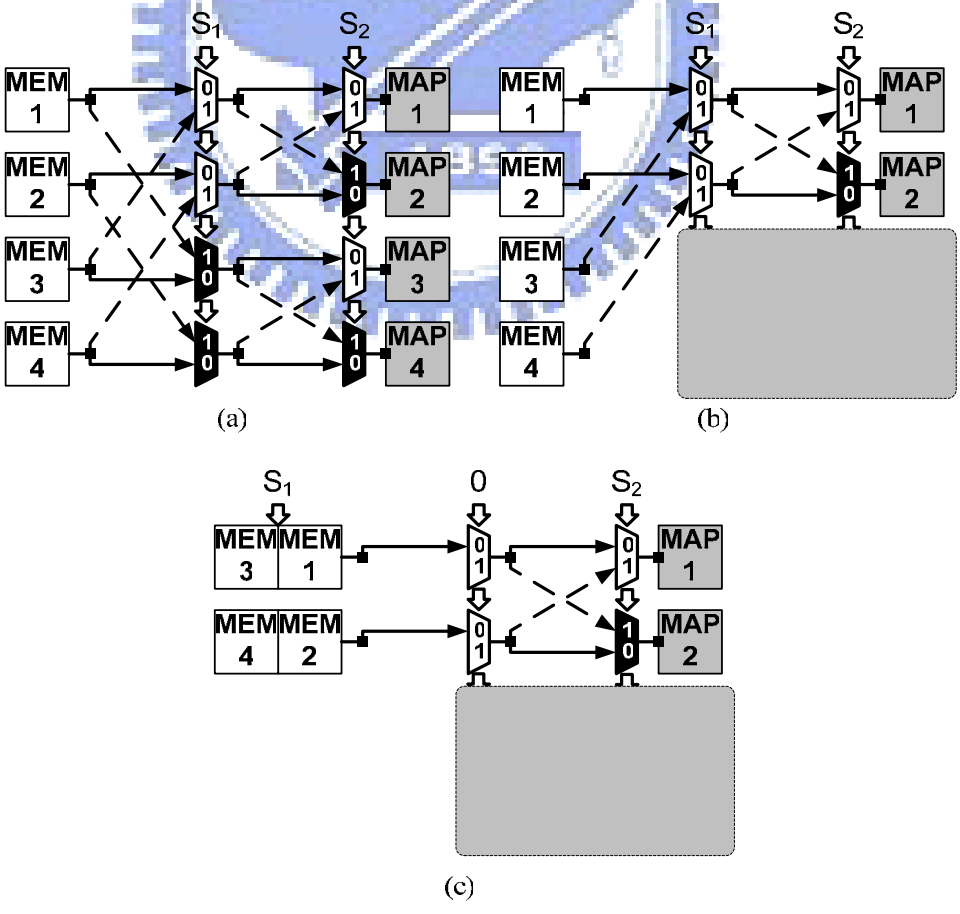


Fig. 4.10 The architecture transformation

4.4.1 Multiple block lengths support

Since the IBP interleaver is flexible, the proposed 1Gbps turbo decoder can be implemented to fit multiple block lengths. In Fig. 4.11(b), we fix the switch S_1 to zero so the decoder is working with only half number of processing elements and memory units. This means the block length of 2's power from 128 to 4096 can be supported in our design. While decoding different length codewords, the only difference in control turn the some switches of the butterfly network to zero.

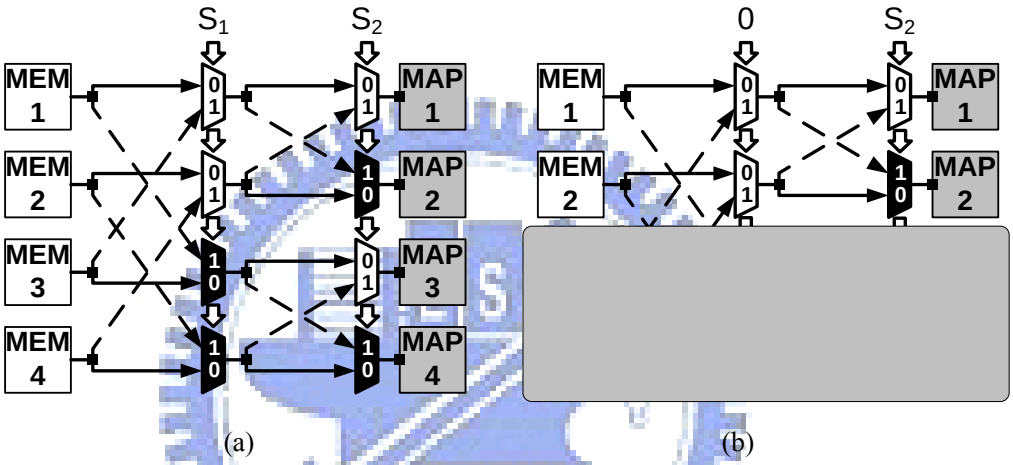


Fig. 4.11 Multiple block lengths support

4.5 Chip Implementation

The proposed 1Gbps turbo code reduces the code rate to 1/3 without any puncturing compared with our previous design, and the implementation applies maximum log-MAP algorithm with a scaling factor 0.75. The detail specifications are listed in Table. 4.1, and the post-layout view is shown in Fig. 4.13 with pin counts 208. The performance of this design is the same as that described in chapter 3. The power management of this design is more careful due to the failure of the delay lock loop (DLL) circuit. Fig. 4.12 shows that we have applied the power isolation technique on our design for the DLL module. The power supply of DLL is isolated from the other circuits, thus the power noise of whole chip will not affect the DLL. Furthermore, the internal clock would be more stable and the uncertainty of internal clock tree

will be smaller because the isolation provides a stable working environment for DLL.

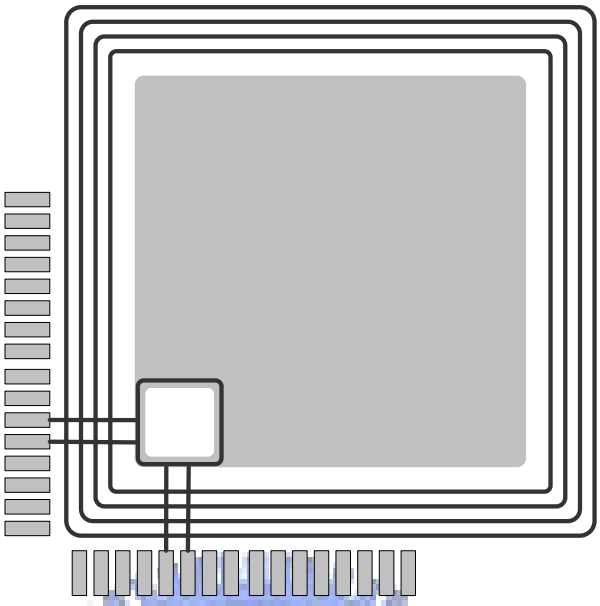


Fig. 4.12 Power isolation of DLL

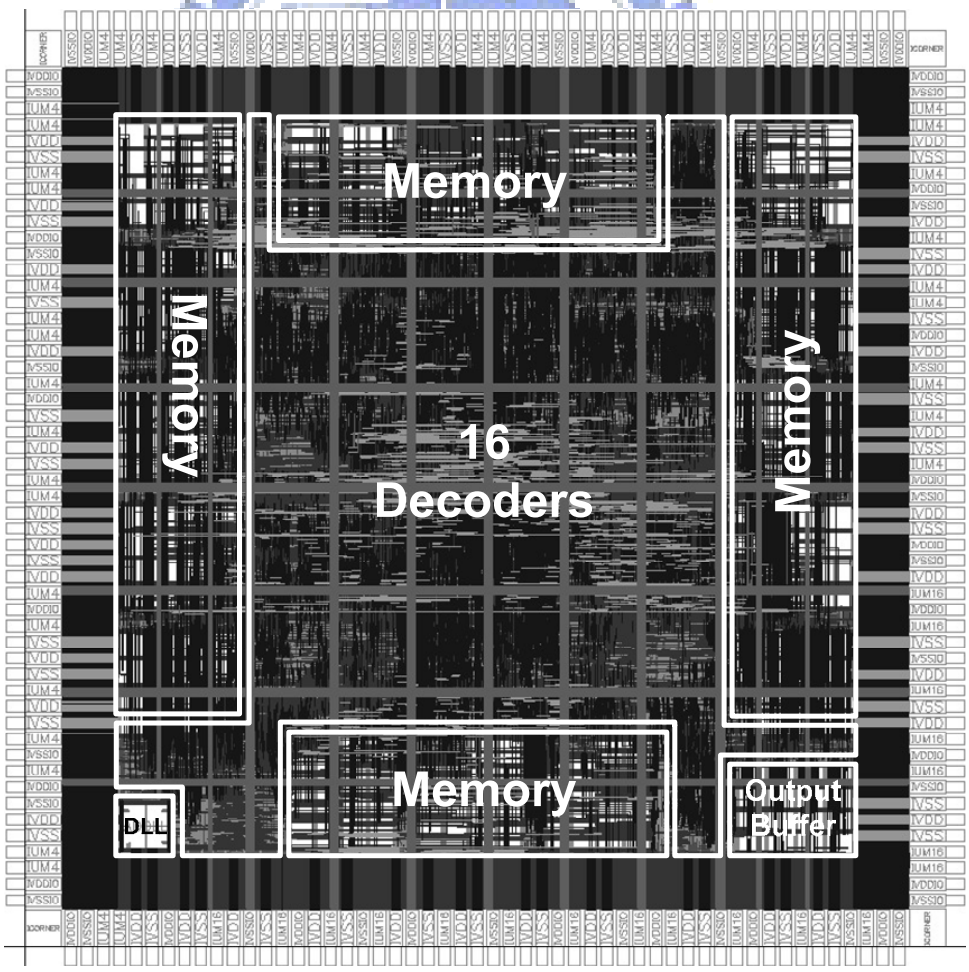


Fig. 4.13 Chip layout view

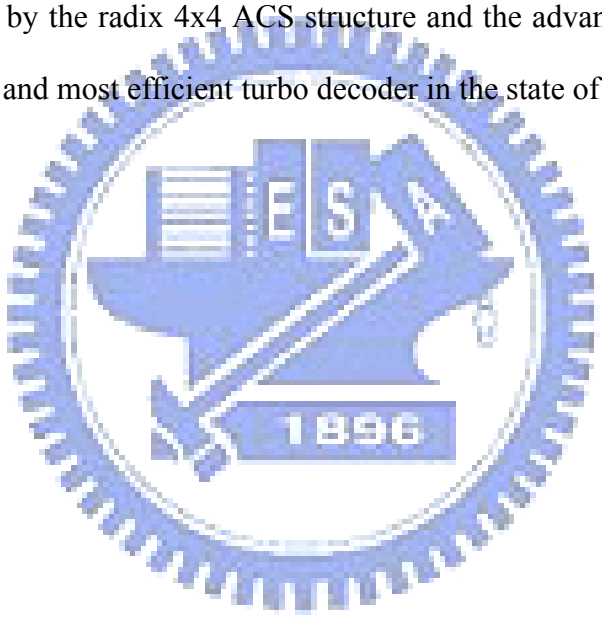
Table 4.2 Summary of the proposed 1Gbps turbo decoder

Algorithm	**Max-Log MAP**
ACS unit	**Radix 4x4**
Code polynomial	$\left[1 \quad \frac{1+D+D^3}{1+D^2+D^3}\right]$
Interleaver	**IBP interleaver (p, s)=(15,23)**
Sliding Window	**32**
Code Rate	**1/3**
Block length	**4096(128 x 32)**
Quantization	**6 bits (3.3)**
iteration	**8**
Scaling Factor	**0.75**
Note	**Tail-Biting**
Technology	**90nm 1P9M**
Clock rate	**250MHz ***
Throughput	**1Gbps ***
Gate count	**2.66M**
Core Area	**9.3 mm2**
power	**1158mW ***
nJ/bit・iteration	**0.144 ***

* post-layout simulation

4.6 Summary

The proposed 1Gbps turbo decoder is the first turbo decoder chip which achieves 1Gbps throughput. We modified the utilization of processing elements and made the decoding schedule more efficient. The improvement of throughput is marvelously 50%. The implementation of interleaver is more flexible than the previous design and the proposed 1Gbps turbo decoder can support multiple code lengths. The energy efficiency of this design improved from 0.22 to 0.144 nJ per bit per iteration compared with our previous design, which is accredited by the radix 4x4 ACS structure and the advanced process. The proposed design is the fastest and most efficient turbo decoder in the state of the art.



Chapter 5
Highly Parallel Decoding of Turbo code

In parallel decoding of turbo code, there are three issues should be particularly considered:

- The throughput of each decoder (processing element)
- The utilization of each decoder
- Parallelism (number of decoders)

In previous chapters, we have proposed some methods to improve the throughput of each decoder and made the decoding more efficient. In this chapter, we will put emphasis on the parallelism. In chapter 3 and chapter 4, the way we used to break the forward and backward recursions for parallelism is to partition a whole codeword into many sub-codewords. But the length of sub-codeword is limited by the distance property of short block length. The distance property of a component code with constraint length 4 is getting worse when the length is below 100. That's why we choose a sub-codeword length 128. Fig. 5.1 shows the architecture described in chapter 4, and the decoder with 16 processing elements achieves 1Gbps. The sub-codeword length is fixed to 128. If we would like to apply more processing elements in the design, we have to find some new approaches for parallelism. The problem returns to "How to decode one sub-codeword with multiple decoders?" In the following section, we will discuss the approach mentioned in [17] and [18] for parallel decoding and show the innovation of our new architectures.

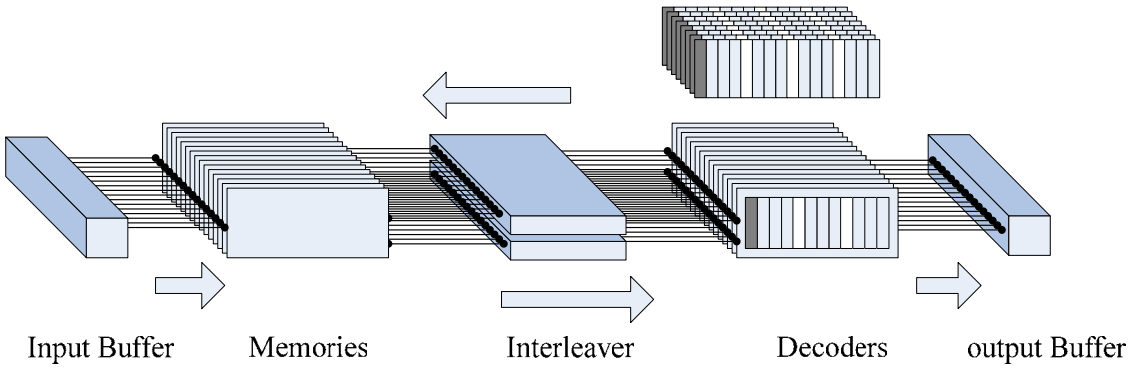


Fig. 5.1 The architecture of 1Gbps turbo decoder

5.1 A Sectionalized Method for Parallel Decoding

In Trellis-based turbo decoder, the forward and backward recursions connect the relation between symbols. The path metric values inherited from the previous Trellis stage make the parallel decoding of a codeword difficult. Even if applying the sliding window approach in fig.5.2, we still suffer from the connecting relation of the forward recursion.

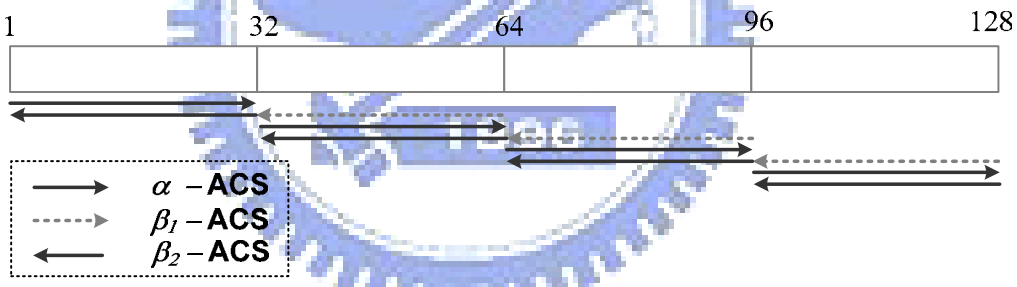


Fig. 5.2 Decoding procedure of sliding window approach

5.1.1 A sectionalized method

The solution of the inheritance of the initial value mentioned in [17] and [18] is to store the needed initial values in this iteration and to apply them in the next iteration. Fig. 5.3 shows the detail procedure of the sectionalized method. In the first iteration, the initial values needed in the next iteration are not available, thus the initial values will be set to zero. After the first iteration, the needed values will be calculated and stored in the memories. Therefore, the initial values will be accessed from the second iteration to the last one. Fig. 5.4 shows different sizes of sectionalized Trellis. The codeword of total block length N can be

sectionalized into different ‘fixed-length’ parts. 4T in Fig. 5.4 means the sectionalized part consists of four Trellis stage, as well as 4 symbols. The 8T and 16T cases are the same as 4T and so on. With the some sub-codeword length, the smaller section we partition, the higher parallelism we get. Note that if the N is getting bigger, the more storage we pay for. The storage of initial values consists of α and β, and the α and β initial values of different decoding round should be stored separately. For example, with a block length 64, state number 8, quantization 6 bits, and sectionalized to 4T case, total bits of the initial storage is 768.

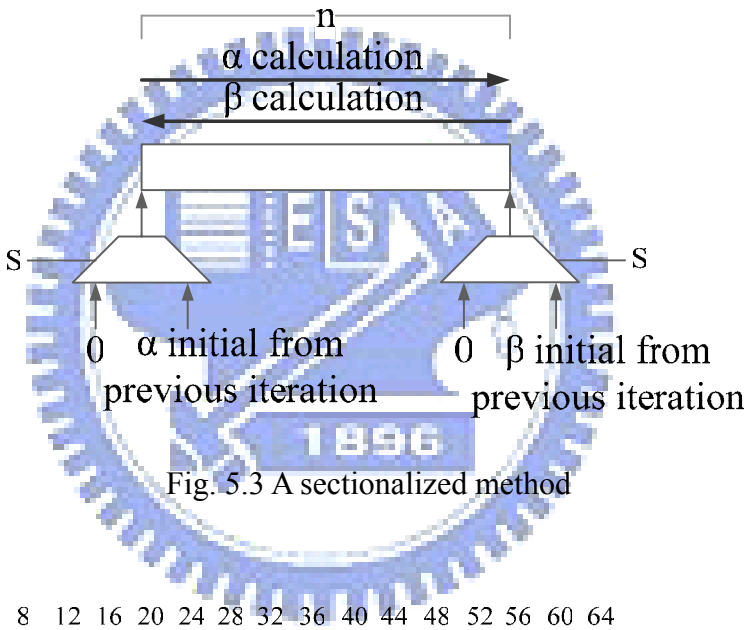


Fig. 5.3 A sectionalized method

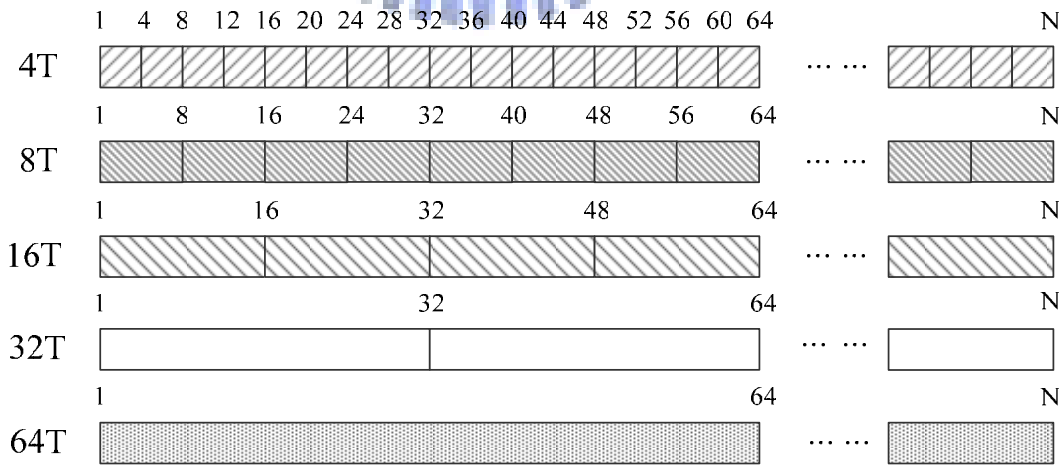


Fig. 5.4 Different sizes of the sectionalized method

5.1.2 Parallel decoding with the sectionalized method

When the recursion relation breaks by the initial storage method, it makes the parallel decoding of a codeword simple. Fig. 5.5 shows the comparison between the sliding window approach and the sectionalized method. The sliding window approach calculates the dummy β for the initial of the real β calculation. Besides, due to the forward recursion, this approach can't apply multiple decoders to decode concurrently. On the other hand, the sectionalized method decodes concurrently by accessing initial values for α and β initial, and saves the calculation time and hardware of the dummy β in the sliding window approach. The comparison of the trade-off will be discussed in the following sections.

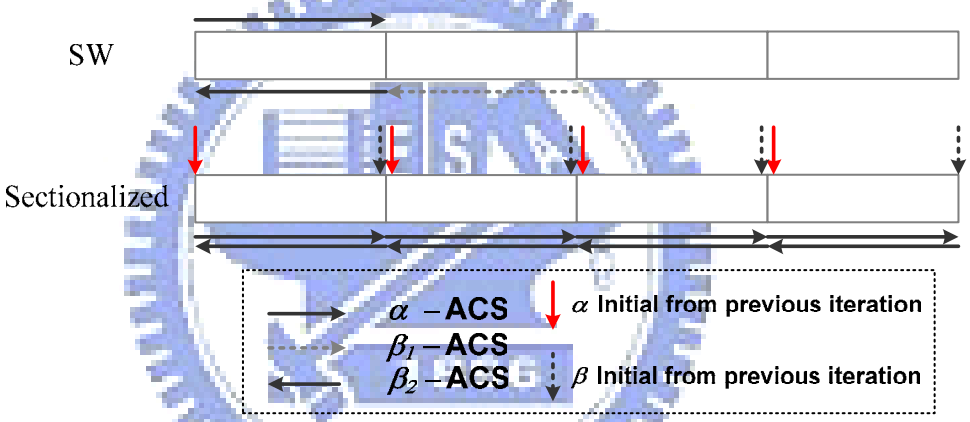


Fig. 5.5 Comparison of different structures

5.2 Proposed Architectures

The sectionalized method partitions a codeword into several sub-blocks and makes higher order parallelism possible. This method can be applied on our design to partition the sub-codewords into some fix-length sub-blocks. The combination of these two methods makes the higher order parallelism possible and can be accounted a 'two-dimension' parallel decoding. The first dimension of the parallelism is called 'inter-codeword' parallelism, which is used and introduced in chapter 3 and 4. The processing elements decode different sub-codewords at the same time. The second dimension of the parallelism is called

'intra-codeword' parallelism, which makes the processing elements decode the same sub-codeword concurrently. The combined method makes all kind of parallel structures possible under the contention-free constraint for memory-based design.

5.2.1 A two-dimension parallel architecture

Fig. 5.6 shows a two-dimension parallel method, which can be considered as a fully parallel type decoder. The architecture can be applied on a highly parallel situation. The contention-free constraint for the interleaver design in the case will be much more complicated. The two-dimension contention constraint should be considered and the interleaver which meets the constraint is few and hard to find. Thinking of the IBP interleaver mentioned in chapter 3, it consists of two-stage permutations and it is contention-free in both two dimensions. The IBP interleaver can be applied in the architecture.

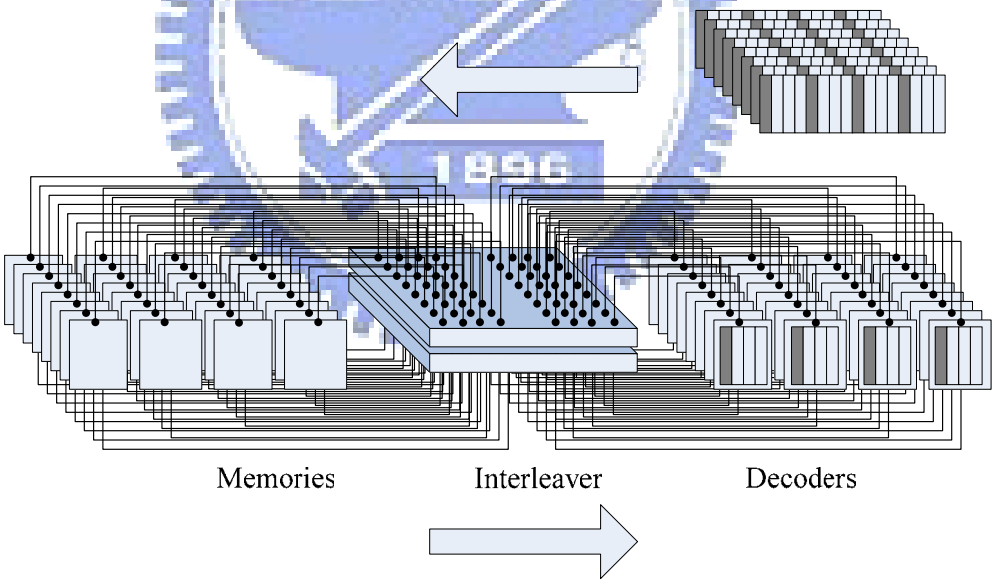


Fig. 5.6 A two-dimension parallel method

5.2.2 A intra-codeword parallel architecture

A downgrade architecture called 'intra-codeword parallel architecture' is the version only in one sub-codeword dimension. In this architecture, we only have to consider the contention

problem in one sub-codeword. It makes things easier. This architecture decode one sub-codeword each time with multiple processing elements. The sub-codewords will take turns to the decoder and go back to the memories.

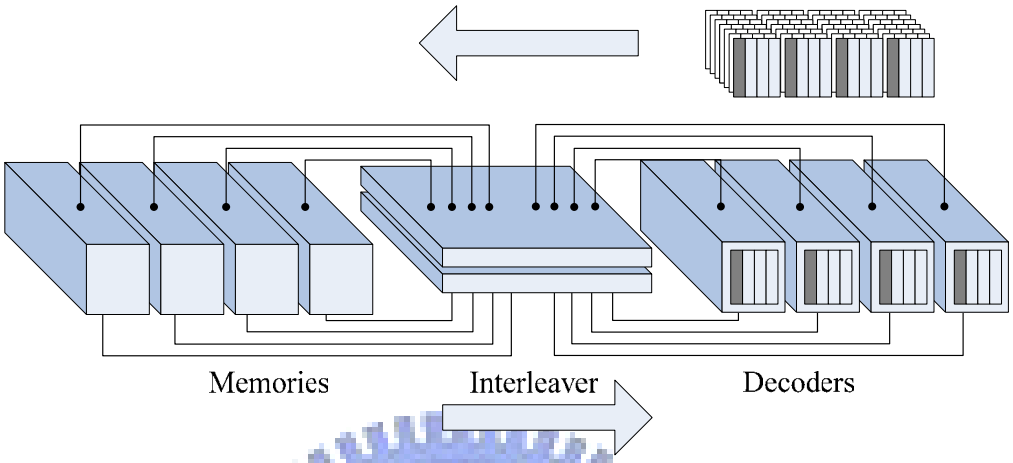


Fig. 5.7 A intra-codeword parallel architecture

5.2.3 Data hazards

The data hazards due to the iterative decoding mentioned in section 4.1 idle the decoding procedure and degrade the utilization. The intra-codeword parallel architecture provides simple and efficient way to remove data hazards. Fig. 5.8 shows that the last few sub-codewords of the pre-decoding and post-decoding rounds in every iteration cause data hazards. The way to solve it is to arrange a proper decoding order of sub-codewords. In Fig. 5.9, the first sub-codeword of the post-decoding round can be decoded at the time we decoding the last few sub-codewords, if we decode the first two sub-codeword of the pre-decoding round first. Because the extrinsic values needed by the first sub-codeword of the post-decoding round has been calculated and stored in the memories, we can decode it without and data hazards. Therefore, a proper arrangement of decoding order would avoid the data hazards and without any hardware overhead. However, the number of the sub-codewords must be large enough for arrangement if the decoding latency is long.

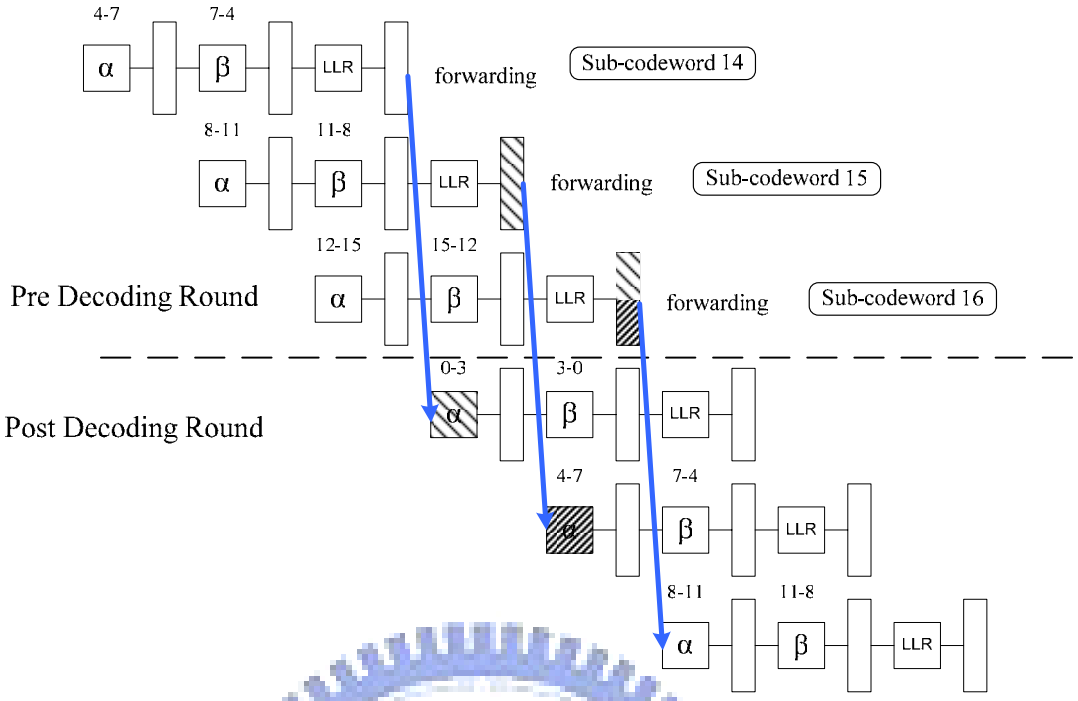


Fig. 5.8 Data hazards

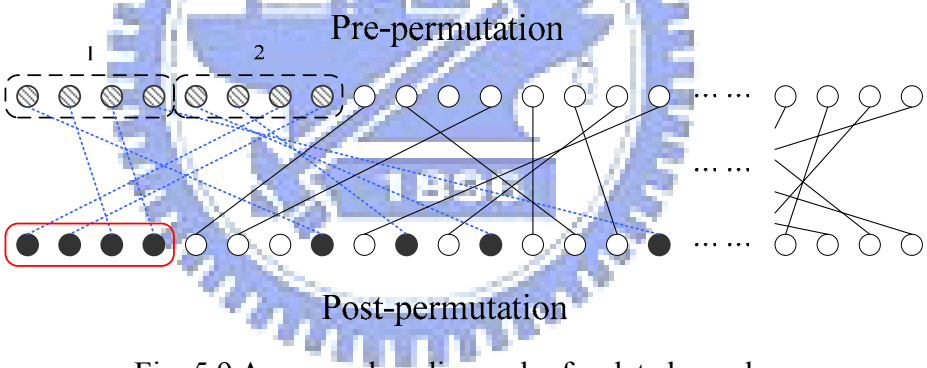


Fig. 5.9 A proper decoding order for data hazards

5.3 *Performance Analysis*

The sectionalized method partition a codeword by storing initial values. From Fig. 5.10 to Fig.5.14, they show the performance of different section sizes. Obviously, the 64T and 32T almost have no performance loss with the same iteration. The loss for each case is less than 0.01dB. However, from 16T to 4T, the performance is getting worse. The performance of BER convergence of the smaller section is worse than the bigger one.

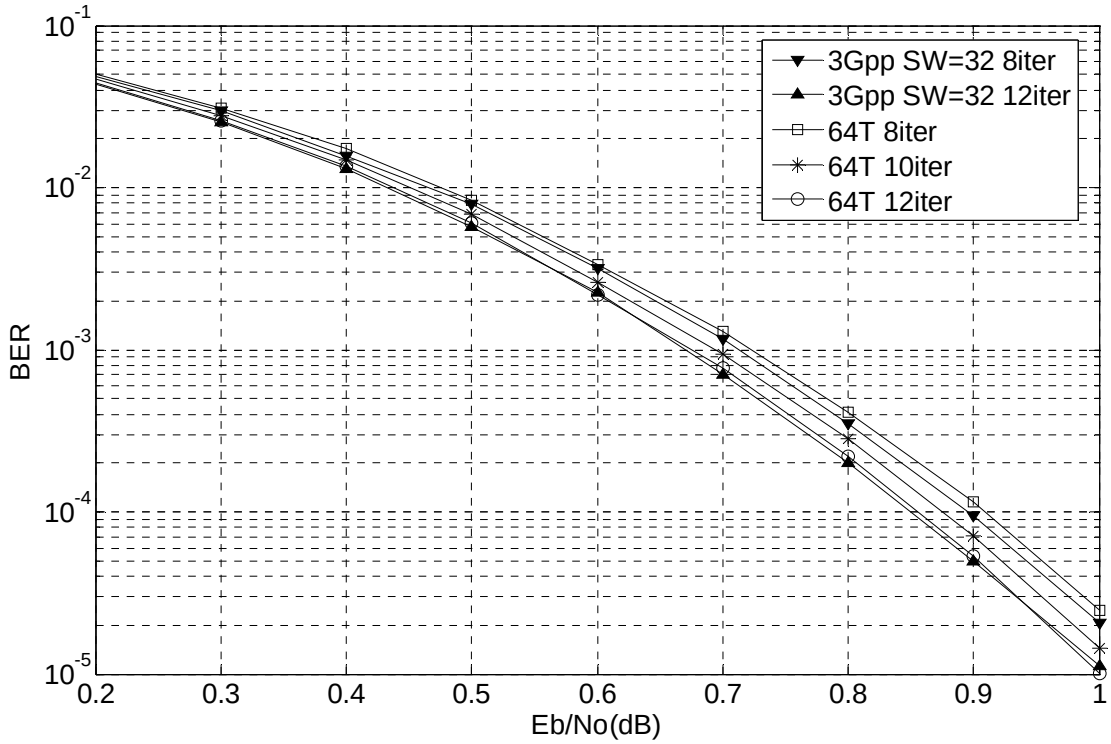


Fig. 5.10 64T performance

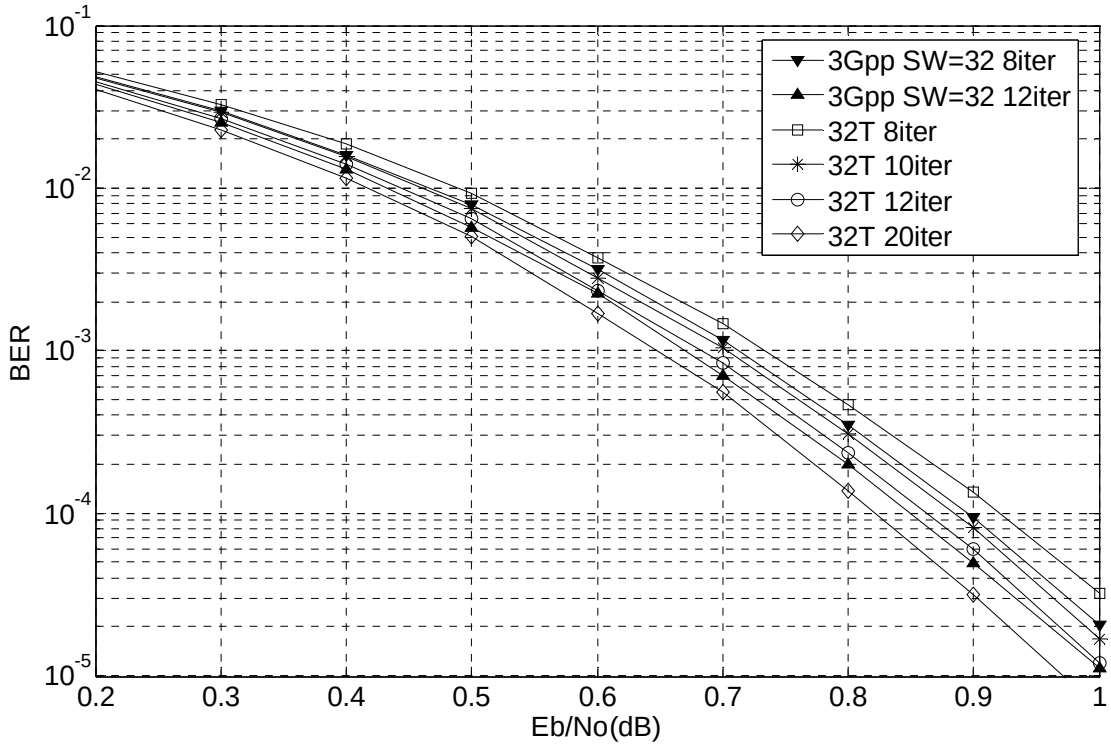


Fig. 5.11 32T performance

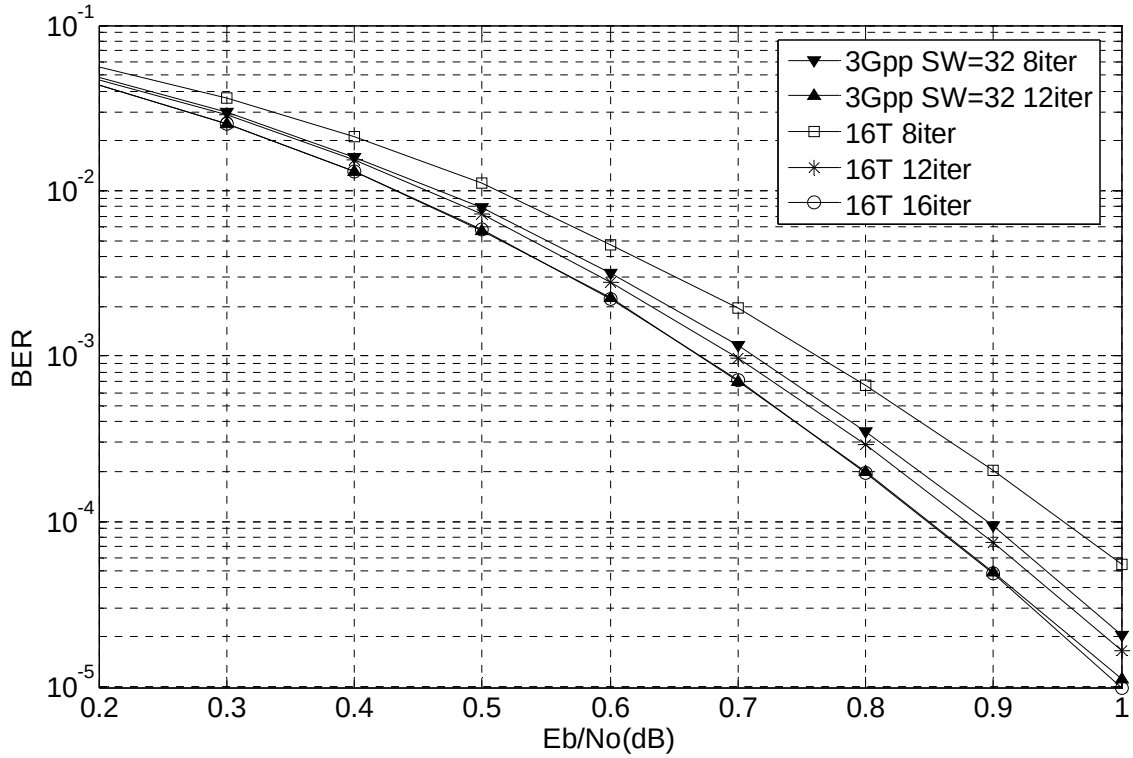


Fig. 5.12 16T performance

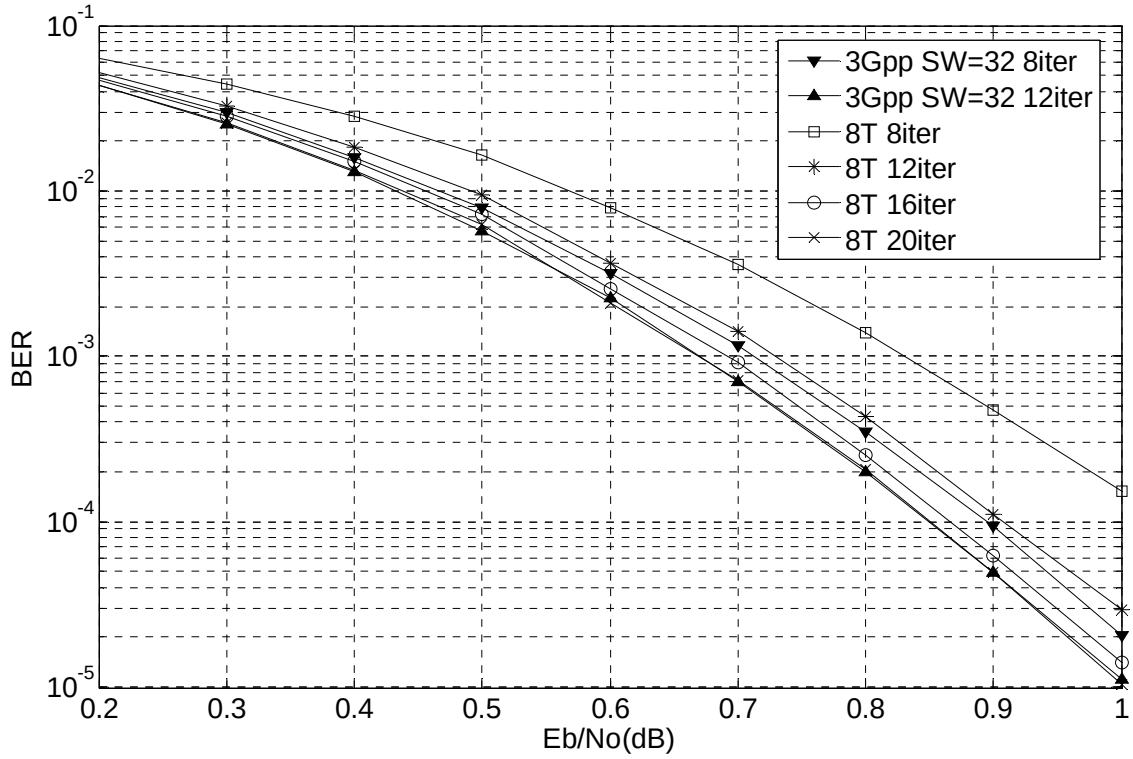


Fig. 5.13 8T performance

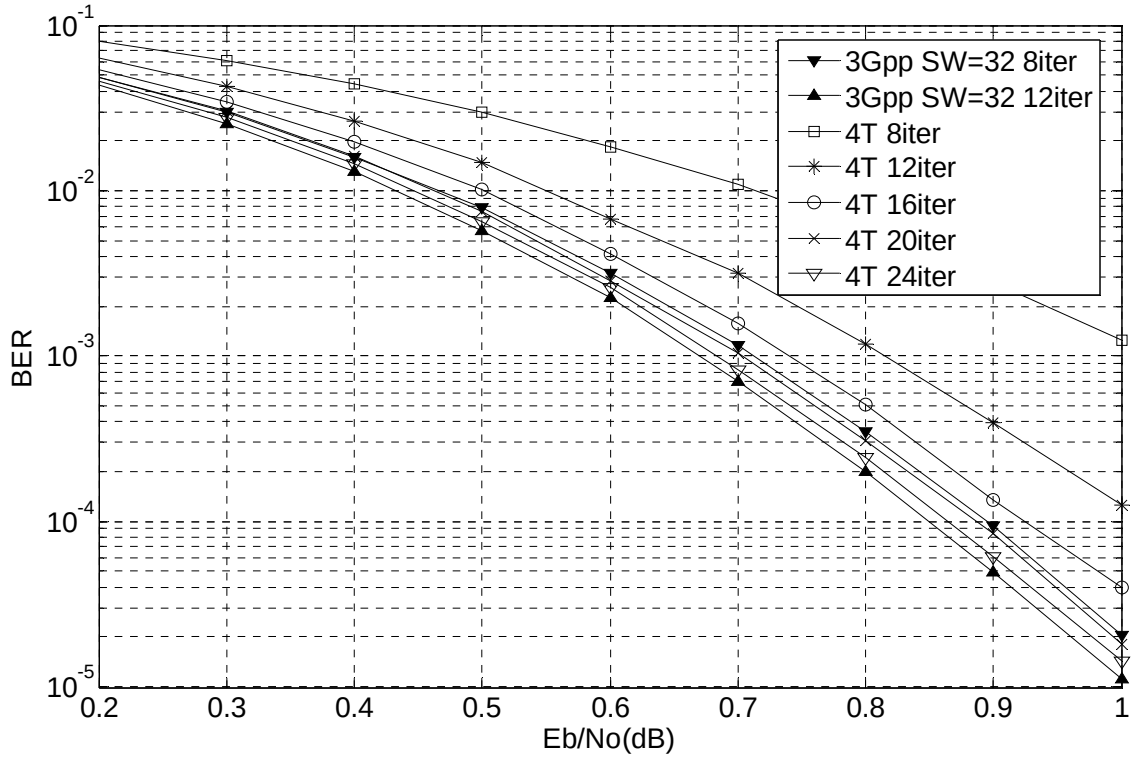
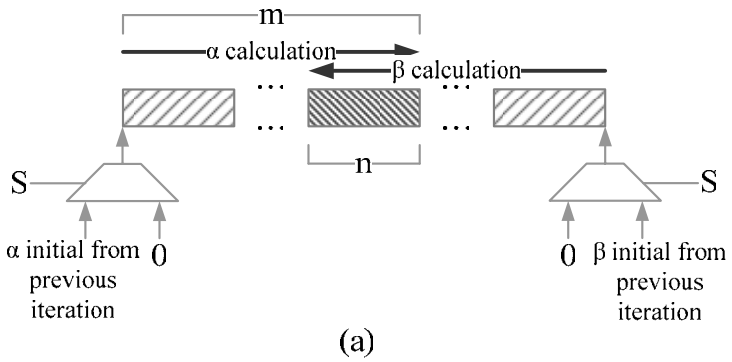


Fig. 5.14 4T performance

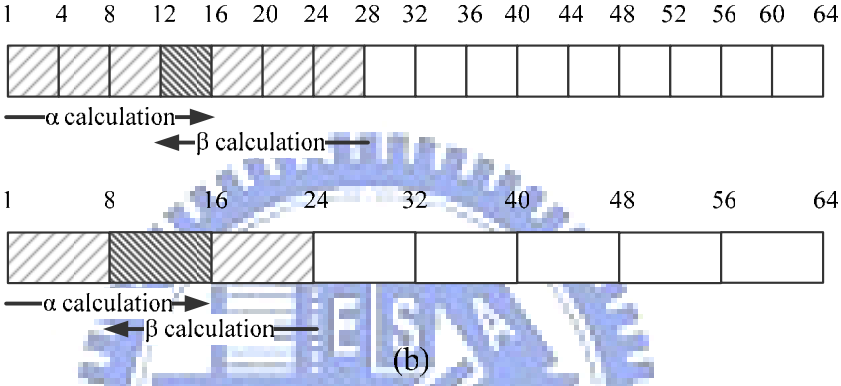
5.4 *Proposed Method to Improve Performance*

Since the performance is getting worse and diverge with small section size, we propose a method to improve the performance. First, we would like to figure out how the performance degrades and then we can find approaches to improve it. The degradation of performance may be formed by two factors: the fist one is the initial zero in the first iteration. The second is the initial values from the previous iteration. The proposed method extend the path metric recursion to more Trellis stage, Which means we would like to accumulate more Trellis stage in this iteration. Fig. 5.15(a) shows that if we access the initial values from the earlier sections and accumulate more correct path metric in this iteration, the performance increase as long as we calculating a path metric long enough. The longer we accumulate, the better the performance is. Fig. 5.16 and Fig. 5.17 show the case of 8T extending to 16T and 4T extending to 8T, 12T, and 16T. The effect of the initial zero can also be found in Fig. 5.16 and Fig. 5.17. It is unapparent to claim that the initial zero is the major factor, but it is obvious that

extension improve the performance greatly.



(a)



(b)

Fig. 5.15 A proposed method to improve performance

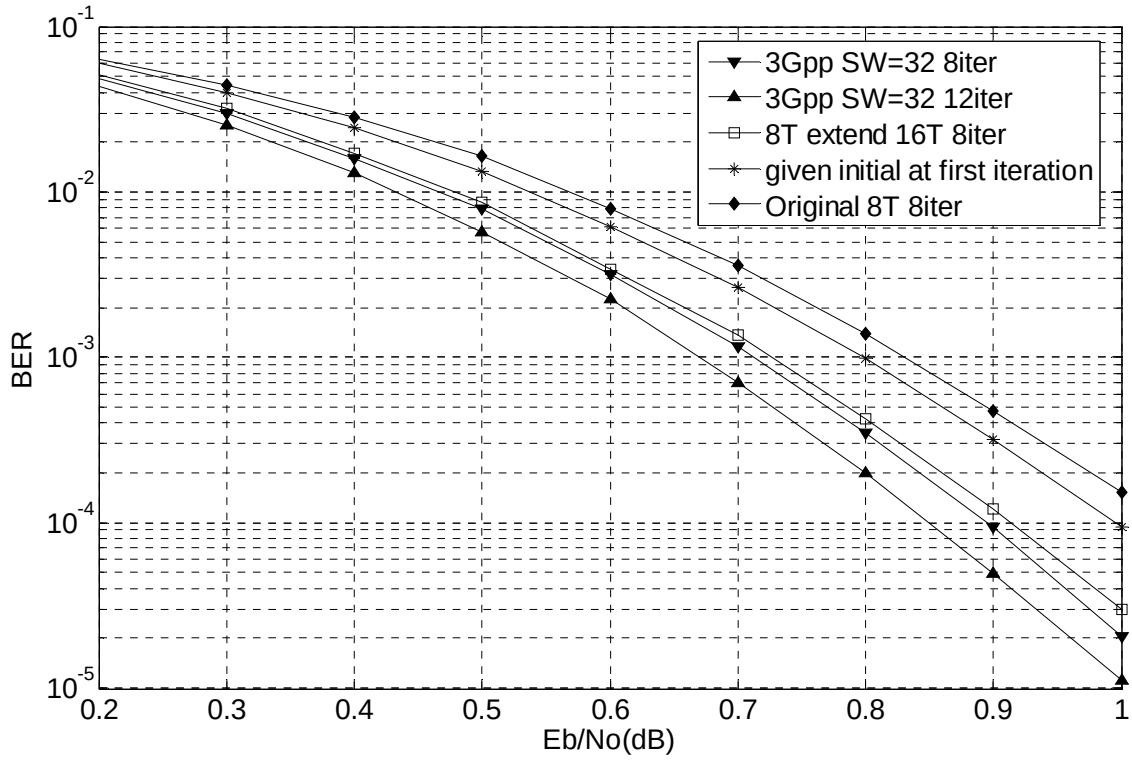


Fig. 5.16 8T extend to 16T

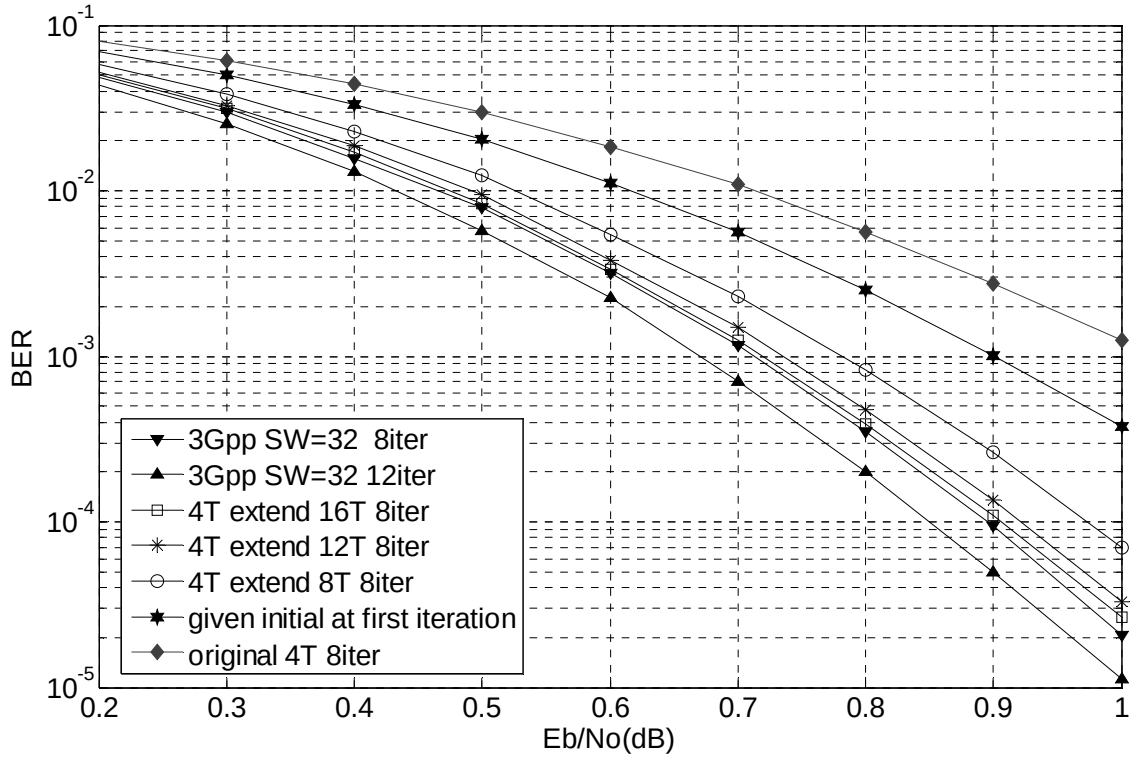


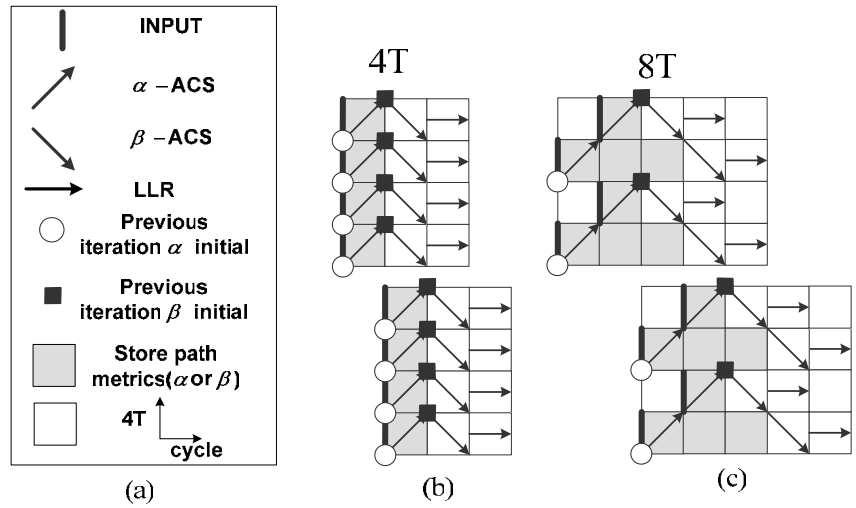
Fig. 5.17 4T extend to 8T, 12T, and 16T

I

5.5 *Decoding Schedule*

As we mentioned in section 5.1, with a fixed block length, the smaller we partition, the more decoders we have. In this section, we would like to apply more decoders for a higher throughput. Fig. 5.18(a) shows the notation of decoding schedule. The circles and the squares denote the initial storage of α and β. The decoding schedule in Fig.5.18(b), (c) and (d) are the original 4T, 8T, and 16T cases individually. If we can make a proper decoding schedule, the number of decoders can be doubled for higher throughput. Comparing with Fig. 5.18(c) and Fig. 5.19(b), the number of decoders is doubled without any overhead, and the decoding latency is shortened. Moreover, the number of decoders in Fig. 5.19(b) is equal to the 4T case, which means that we achieve the 4T case throughput with 8T case overhead. The decoder number of 4T case can not be doubled because the design is based on radix 4x4 design. Fig. 5.18(a) shows that one step of vertical axe means reading 4 symbols per cycle. The Fig. 5.20

shows the decoding schedule of the extension verision.



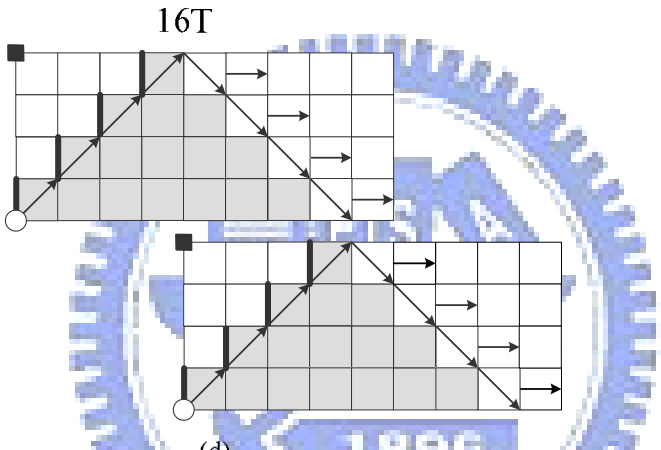


Fig. 5.18 The original decoding schedule

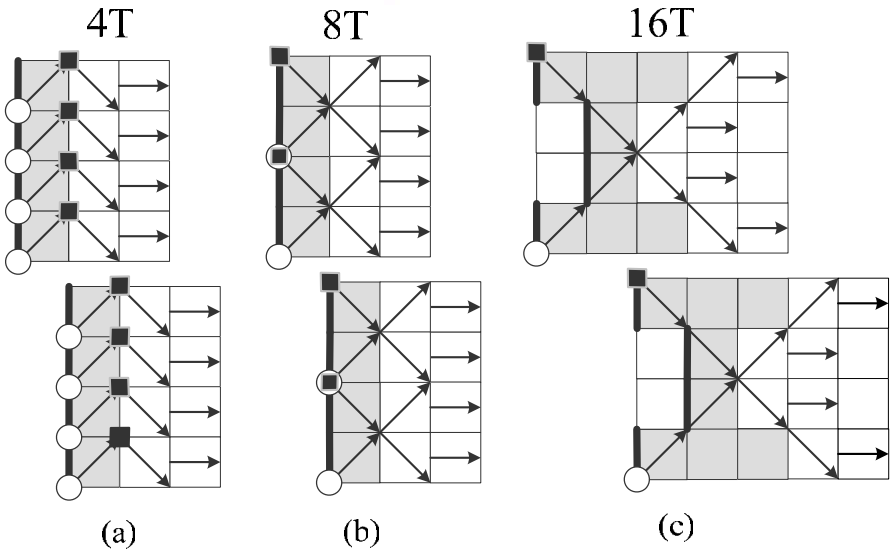


Fig. 5.19 example of the new 8T and 16T decoding schedule

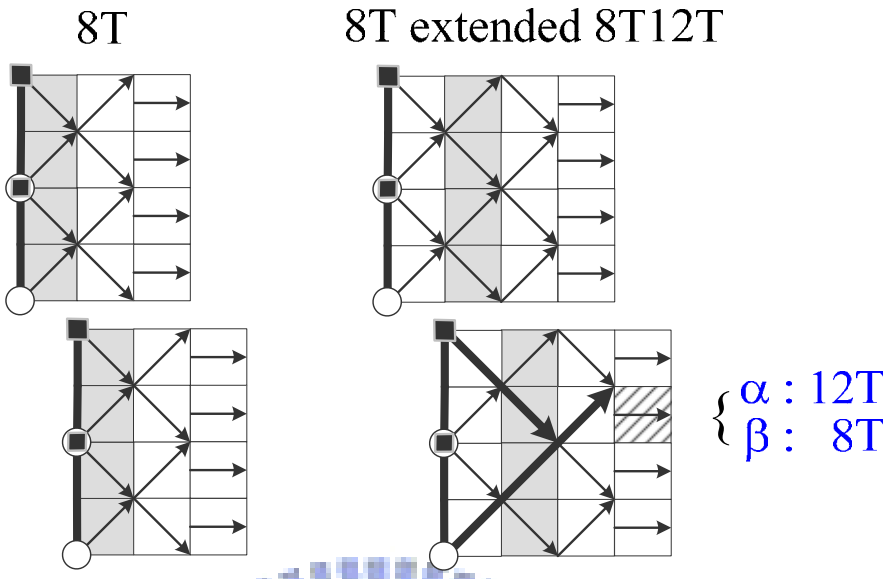


Fig. 5.20 Decoding schedule of extension

5.6 *Hardware Comparison*

Table 5.1 and Table 5.2 list the comparison between original design and the proposed two architectures. It is obvious that the storage different will be affected by three parameters, N, n, n_0, and M. the most important parameters is M, because the greater M makes the storage reduction larger. In other word, if we apply more processing elements in our design, the total storage compared with the original SW case may be reduced. If the case is a medium M, there will have some storage overhead. However, the reduction of ACS will save the area and gate count apparently.

Table 5.1 Hardware comparison of two-dimension parallel architecture

	ACS No.	Path metric Storage(bits)	Total Reduction compare with SW	Cost compare with SW (initial value storage)
Original SW Approach	3 x M	n_0 x S x b x M		
Two-dimension Parallel Architecture (n x M = N)	2 x M (+M if extending)	n x S x b x M	ACS :reduced M (0 if extending) Storage: reduced (n0 – n) x S x b x M	N/n x S x b x 2 (x 2 if no scheduling)

Example	Total Reduction		Cost	
	No Scheduling	Scheduling	No Scheduling	Scheduling
N = 4096 n = 16 n_0 = 32 b = 6 S = 6	M = 256 ACS : reduced 256 Storage: reduced 196608 bits	M = 512 ACS : reduced 512 Storage: reduced 393216 bits	M = 256 Storage: increased 49152 bits	M = 512 Storage: increased 24576 bits

N: Block length
M: No. of decoders
n: sectionalized-window size
n_0: sliding-window size
b: quantization bit
No. of path metric
S: state No.

Table 5.2 Hardware comparison of intra-codeword parallel architecture

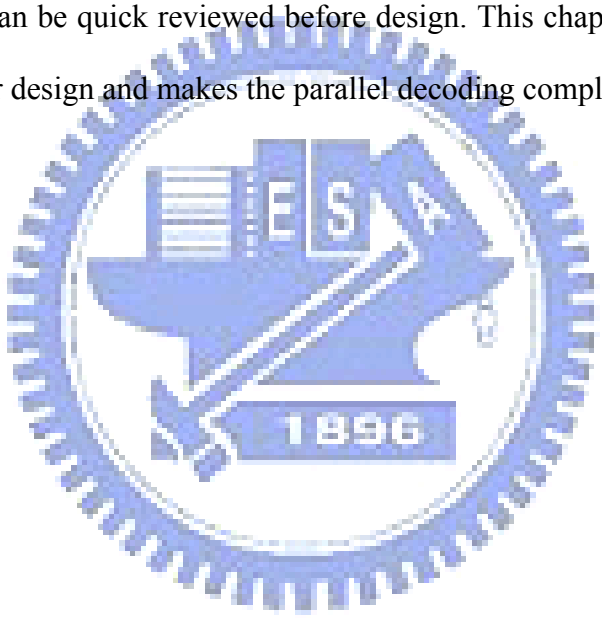
	ACS No.	Path metric Storage(bits)	Total Reduction compare with SW	Cost compare with SW (initial value storage)
Original SW Approach	3 x M	n_0 x S x b x M		
Intra-codeword Parallel Architecture (n x M = p)	2 x M (+M if extending)	n x S x b x M	ACS :reduced M (0 if extending) Storage: reduced (n0 – n) x S x b x M	N/n x S x b x 2 (x 2 if no scheduling)

Example	Total Reduction		Cost	
	No Scheduling	Scheduling	No Scheduling	Scheduling
N = 4096 n = 8 p = 128 n_0 = 32 q = 32 b = 6 S = 6	M = 16 ACS : reduced 16 Storage: reduced 18432 bits	M = 32 ACS : reduced 32 Storage: reduced 36864 bits	M = 16 Storage: increased 98304 bits	M = 32 Storage: increased 49152 bits

N: Block length
M: No. of decoders
N = p x q
p: sub-block length
q: sub-block No.
n: sectionalized-window size
n_0: sliding-window size
b: quantization bit
No. of path metric
S: state No.

5.7 Summary

In this chapter, we modified and combined the concept in [18] with our original design to innovate a new two-dimension parallel structure. The performances with different section sizes have been analyzed for different applications. A method to improve the performance convergence is proposed with reasonable hardware cost. Two parallel architectures are proposed for different design constraints and modified hazard-free method is discussed in section 5.2.3. a double throughput scheduling method is proposed for highly parallelism. Meanwhile, the parametric hardware comparisons are list in Table 5.1 and Table 5.2 with example and they can be quick reviewed before design. This chapter facilitates the ultra high speed turbo decoder design and makes the parallel decoding complete.



Chapter 6
Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed two turbo decoders with the parallel architecture which enables multiple processing elements to decode one codeword concurrently. The proposed IBP interleaver connects all processing elements with a easily implemented structure and avoids the limit of the forward and backward recursions.

In the first design, we also introduce a high speed methodology for high radix decoder structure with a matching contention-free IBP interleaver. The combination of two stages ACS and the retiming technique efficiently speed up the decoding throughput with acceptable hardware cost. The energy efficiency of proposed turbo decoder is much smaller than that of the state of the art.

In the second 1Gbps design, we modified the utilization of processing elements and made the decoding schedule more efficient with doubled throughput. The implementation of interleaver is more flexible than the previous design and the proposed 1Gbps turbo decoder can support multiple code lengths. The proposed 1Gbps turbo decoder is the most power efficient and the fastest turbo decoder chip which achieves 1Gbps throughput in the state of the art.

In chapter 5, we proposed a combined method to make the parallelism work in two dimensions. The performance and the hardware cost with different condition have been analyzed and a new extension method and a new scheduling method are proposed to improve the performance and the throughput.

6.2 *Future Work*

Up to now, the early termination scheme is regarded as the most efficient way to reduce the power consumption in turbo decoders. It uses several characteristics in turbo decoding to judge if decoding sequence is nearly correct before maximum iteration number is achieved. Once iterative decoding can be stopped earlier, then the power can be saved. In [37], an iteration stopping criterion has been modified based on the cross entropy between the *a posteriori* probabilities of two SISO decoders for each iteration. Some other simplified criteria was proposed in [38] and [39]. Most of these criteria make the decoder idle for saving power. The idea of utilization mentioned in chapter 4 will be useful for thinking of a new stopping criterion, which should be more precisely called "skipping criterion." If we set a "skipping criterion" for all sub-codewords in our proposed intra-codeword parallel architecture, the decoder will skip decoding certain sub-codewords which is meet the "skipping criterion." Therefore, the decoding procedure would be more efficient and the throughput will increase as the channel condition going better.

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error- correcting coding and decoding: Turbo-codes (1)," in *Proc. IEEE Int. Conf. on Commun*., Geneva, Switzerland, May 1993, pp. 1064–1070.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol," *IEEE Trans. Inform. Theory*, no. IT-20, pp. 284–287, Mar. 1974.

[3] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429-445, Mar. 1996.

[4] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-decision Outputs and its Applications," in *IEEE GLOBE-COM*, Dallas, TX, pp. 47.1.1-47.1.7, Nov. 1989.

[5] G. Solomon and H. C. A. van Tilborg, "A connection between block and convolutional codes," *SIAM J. Appl. Math*., vol. 37, pp. 358–369, Oct. 1979.

[6] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun*., vol. COM-34, pp. 104–111, Feb. 1986.

[7] C. Weiss, C. Bettstetter, S. Riedel, and D. J. Costello, "Turbo decoding with tailbiting trellises," *in Proc. URSI Int. Symp. Signals, Systems, Electronics*, 1998, pp. 343–348.

[8] C. Weiss, C. Bettstetter, and S. Riedel, "Code construction and decoding of parallel concatenated tail-biting codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 366-386, Jan. 2001.

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

[10] P. Robertson, E.Villebrun and P. Hoeher, “A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms operating in the Log Domain,” *Proc. ICC’95*, Seattle, June 1995.

[11] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, “A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless,” in *ISSCC Dig. Tech. Papers*, 2003, pp. 151–484.

[12] B. Bougard, A. Giulietti, V. Derudder, J. Willem, S. Dupont, L. Hollevoet, F. Catthoor, L. V. der Perre, H. D. Man, and R. Lauwereins, “A scalable 8.7nj/bit 75.6Mb/s parallel concatenated convolutional (turbo-)codec,” in *ISSCC Dig. Tech. Papers*, 2003, pp. 152–484.

[13] Y. Zheng, “Network for permutation or de-permutation utilized by channel coding algorithm,” U.S. Patent Pending.

[14] 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-4×4 ACS architecture,” in *IEEE A-SSCC*, 2006, pp. 79–82.

[15] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, “VLSI architectures for metric normalization in the Viterbi algorithm,” in *Int. Conf. Communications*, vol. 4, Atlanta, CA, Apr. 1990, pp. 1723–1728.

[16] P. Urard, L. Paumier, M. Viollet, E. Lantreibecq, H. Michel, S. Muroor, and B. Gupta, “A generic 350Mb/s turbo-codec based on a 16-states SISO decoder,” in *ISSCC Dig. Tech. Papers*, 2004, pp. 424–536.

[17] Z. He, P. Fortier, and S. Roy, “Highly parallel decoding architectures for convolutional turbo codes,” *IEEE Trans. VLSI Syst.*, vol. 14, no. 10, Oct. 2006.

[18] S. Yoon, and Y. Bar-Ness, “A Parallel MAP Algorithm for Low Latency Turbo Decoding,” *IEEE Commun. Lett.*, vol. 6, no. 7, pp.288-290, Jul. 2002.

[19] J. H. Andersen, “‘Turbo’ Coding for Deep Space Application,” in *IEEE International Symposium on Inform. Theory*, 17-22, pp.36, Sep. 1995.

[20] D.Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, “Performance Analysis of Turbo Codes,” in *IEEE Military Communication conf.*, vol. 1, 5-8, pp. 91-96, Nov. 1995.

[21] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-decision Outputs and its Applications," in *IEEE GLOBE-COM*, Dallas, TX, pp. 47.1.1-47.1.7, Nov. 1989.

[22] J. H. Andersen, "'Turbo' Coding for Deep Space Application," in *IEEE International Symposium on Inform. Theory*, 17-22, pp.36, Sep. 1995.

[23] J. H. Andersen, "Turbo codes extended with outer BCH code," in *Electronics Letters*, vol. 32, no. 22, 24, pp.2059-2060, Oct. 1996.

[24] 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.

[25] T. A. Summers and S. G. Wilson, "SNR Mismatch and Online Estimation in Turbo Decoding," *IEEE Trans. Commun.*, vol. 46, pp.421-423, Apr. 1998.

[26] A. Worm, P. Hoeher, N. Wehn, "Turbo-Decoding Without SNR Estimation," *IEEE Commun. Letters*, vol. 4, no. 6, pp.193-195, June 2000.

[27] S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," University of South Australia, *PhD Dissertation*, Aug. 1995.

[28] S. A. Barbulescu, "On Sliding Window and Interleaver Design," *Electronics Letters*, vol. 37, no. 21, pp.1299-1300, Oct. 2001.

[29] A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp.260-269, Mar. 1973.

[30] Y. Wu and B. D. Woerner, "Internal data width in SISO decoding module with modular renormalization," in *IEEE Vehic. Tech. Conf.*, vol. 1, pp. 675-679, May 2000.

[31] Y. Wu, B. D. Woerner, and T. K. Blankenship, "Data Width Requirements in SISO Decoding With Module Normalization," in *IEEE Trans. On Commun.*, vol. 49, no. 11, pp. 1861-1868, Nov. 2001.

[32] T. K. Blankenship and B. Classon, "Fixed-Point Performance of Low-Complexity Turbo Decoding Algorithms," in *IEEE Vehic. Tech. Conf.*, vol. 2 pp. 1483-1487, May 2001.

[33] C. B. Shung, P. H. Siegel, G. Ungerboeck and H. K. Thapar, “VLSI architectures for metric normalization in the Viterbi algorithm,” *IEEE International Conference on Communications*, vol. 4, pp.1723-1728, Apr. 1990.

[34] A. P. Hekstra, “An Alternative to Metric Rescaling in Viterbi Decoders,” *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220-1222, Nov. 1989.

[35] G. Feygin and P. G. Gulak, “Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoder,” *IEEE Trans. On Commun.*, vol. 41, no. 3, pp. 425-429, Mar. 1993.

[36] M. A. Bickerstaff, D. Garrate, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, R. H. Yan, “ A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS”, in *IEEE Journal of Solid-State Circuits*, vol.37, no.11, Nov. 2002

[37] M. Moher, “Decoding via Cross Entropy Minimization,” in *Proc. IEEE Globecom Conf.*, Houston, TX, Dec. 1993, pp.809-813.

[38] R. Y. Shao, S. Lin, and M. P. C. Fossorier, “Two Simple Stopping Criteria for Turbo Decoding,” *IEEE Trans. On Commun.*, vol. 47, no. 8, pp.1117-1120, Aug. 1999.

[39] Y Wu, D. Woerner, and J. Ebel, "A simple stopping criteria for turbo decoding," *IEEE Commun. Letters*, vol. 4, pp. 258-260, Aug. 2000.