國 立 交 通 大 學

電 子 工 程 系

碩 士 論 文

使用事先資訊之三元渦輪編碼調變
Ternary Turbo Coded Modulation Using Embedded Prior Information

研 究 生：周毓堂

指導教授：張錫嘉　教授

中 華 民 國 九 十 四 年 十 月

使用事先資訊之三元渦輪編碼調變

Ternary Turbo Coded Modulation Using Embedded Prior Information

研　究　生：周毓堂　　　　Student : Yu-Tang Chou

指導教授：張錫嘉　　　　Advisor : Hsie-Chia Chang

國立交通大學 電子工程學系

電子研究所 碩士班

碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of

Master

In

Electronics Engineering

October 2005

Hsinchu, Taiwan, R.O.C.

中華民國九十四年十月

使用事先資訊之三元渦輪編碼調變

研究生：周毓堂　　　　　　　　　　指導教授：張錫嘉 博士

國立交通大學電子工程學系(研究所)碩士班

摘要

因為渦輪碼擁有非常好的錯誤更正能力，所以在近 10 年它已經成為無線通訊中相當重要的一種改錯碼技術。然而一個好的渦輪碼通常需要很長的交錯器，長的交錯器雖然能夠提升渦輪碼的性能但也造成冗長的解碼延遲以及需要大量的儲存記憶體。此外，因為渦輪碼有 error floor 的現象導致在高的 SNR 時它無法達到很低的錯誤率。

在這篇論文中我們提出了一種使用事先資訊之三元渦輪編碼調變系統，簡稱為鑲入已知資訊之三元渦輪編碼調變。相較於使用交錯器長度 20730 的一般渦輪碼，使用交錯器長度 800 的鑲入已知資訊之三元渦輪編碼調變不但可以節省 86%的記憶體並且降低 94%的解碼延遲。另外在錯誤率 10^{-5}和 10^{-6}之下，此系統比一般渦輪編碼器還要好 0.2dB 和 0.6dB。

Ternary Turbo Coded Modulation Using Embedded Known Digits Technique

student: *Yu-Tang Chou* Advisor: Dr. *Hsie-Chia Chang*

Department of Electronics Engineering

National Chiao Tung University

ABSTRACT

Recently, (binary) turbo code is one of most important coding techniques for wireless communication due to its excellent error correction ability. The main problem of turbo code is that using a long interleaver is able to support its remarkable performance, but a long decoding latency and huge memory size are introduced. Moreover, at high SNR, the error floor phenomenon causes that turbo code unable to achieve very low bit error rates.

In this thesis the ternary turbo coded modulation using embedded prior information, in short, called EPI TTC is presented. Comparison with binary turbo code with interleaver size of 20730, the proposed system with interleaver size of 800 can save not only memory size about 86% but also decoding latency about 94%. In addition, it has 0.2 dB and 0.6 dB performance gains than binary turbo code at BER of 10^{-5} and 10^{-6}, separately.

誌謝

在交大兩年的時光很快就結束了，這兩年我學到了許多做學問的方法，也交到許多很好的朋友，最後我得以順利完成我的論文首先要感謝我的指導教授：張錫嘉老師對我的指導，每當我遇到困難時老師總能夠給予我適當的幫助，另外我要感謝林建青學長，這兩年學長教我很多觀念也不厭其煩的跟我討論許多我不懂的問題，我還要感謝毓成、允率、弘安、維君、小科、景大和其它Oasis以及FEC Group的成員，有你們的陪伴讓我的生活多了很多樂趣，最後也感謝我的女朋友雯婷，感謝你能體諒這些日子我不能好好陪妳，最後我要再次向每個幫助過我的人說聲謝謝。

Content

LIST OF FIGURES:

LIST OF TABLES:

Chapter1 Introduction

1.1 Motivation

In the last decade, turbo code is one of most important coding techniques for wireless communication due to its excellent error correction ability. The main problem with turbo code is that a long interleaver is able to support its exceptional performance, but a long decoding latency and huge memory size are introduced. The second problem about turbo code is error floor effect that causes turbo code unable to satisfy the requirement of very low bit error rates. The occasion of main problem with turbo code originates form long interleaver, and second problem is caused by relative small of minimum code free distance. In general, to concatenate a RS code before turbo code can easily solve the error floor effect. However it decreases the data rate.

Our work is motivated by the need to achieve large coding gains where the required interleaver size is as small as possible and the data rate is not reduced. Firstly, the trellis coded modulation technique is considered since it can introduce additional coding by constellation expansion, but keeping constant data rate. Then, we are interested in building a coded modulation system for the transmission of a 3-ary signal constellation over an additive white Gaussian noise (AWGN) channel. Fig. 1.1 illustrates this idea where System 1 is binary turbo code with long interleaver and modulated by BPSK and the question mark in System 2 is the coding system that we want to design. The design challenge is how to design a coded modulation system which has not only excellence performance as binary turbo code with long interleaver size but also small latency. Finally the ternary turbo coded modulation is adopted since its property is similar to binary turbo code and its codeword sequences can be direct modulated by 3PSK.

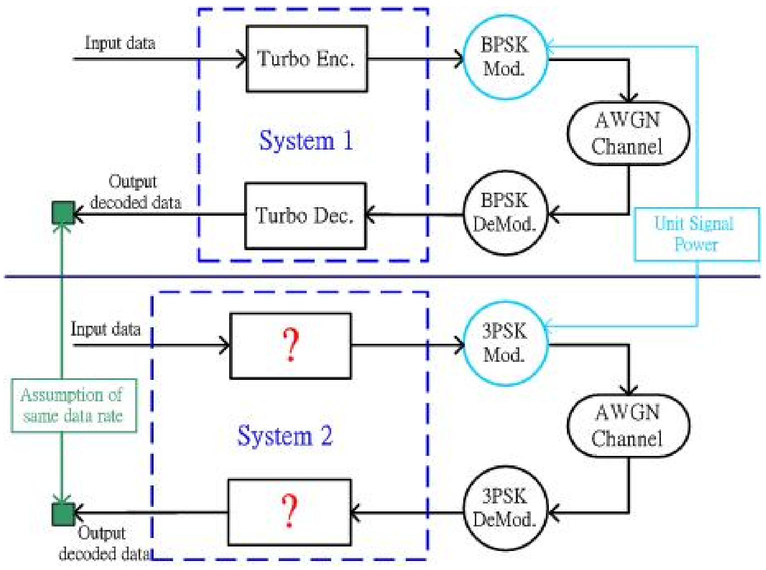

Fig. 1.1 Block diagram of the different systems

In this thesis, the ternary coded modulation using embedded prior information is presented. The proposed coding system with interleaver size of 800 has not only better performance but also lower error floor effect than binary turbo with interleaver size of 20730. In addition, it can save not only memory size about **86%** but also decoding latency about **94%**.

1.2 Thesis Organization

This thesis consists of 6 chapters. In chapter 2, the binary turbo code included structure of turbo coding, decoding algorithm and its relative techniques, is introduced. The concept of ternary turbo code and some property comparisons between binary turbo code and ternary turbo code is described in Chapter 3. In chapter 4, the proposed system is presented, and some performance effects introduced by known digits will be discussed. In chapter 5, simulation will be shown to verify the performance of proposed system has better performance gain than binary turbo code system. However, the hardware and computing complexity will discuss in this chapter. Finally, conclusion and future work are made in chapter 6.

Chapter2 Reviews of Turbo Code

The turbo code [1] over GF(2) was first proposed by C. Berrou, A. Glavieux, and P. Thitimajshima in 1993. The performance in terms of bit error rate (BER) has been proved to close to the Shannon limit. The basic elements in turbo coding include reliability-based, iterative decoding and random interleaver. Reliability-based means that turbo code employs soft-input soft-output (SISO) maximum-a-posteriori (MAP) decoders for the component codes. Iterative decoding signifies that turbo decoder uses extrinsic information, the symbol reliability, from a component decoder feedback to other one, and vice versa. Random interleaver is to ensure that, at each iteration the component MAP decoders get independent estimates on the information symbols.

The turbo code is now adopted by many standards listed in Table 2.1 due to its excellent error correction ability. In this chapter, we'll describe the principle of turbo coding.

Table 2.1 Standard specifications for turbo coding

Standard	Application	Iterative Code	Max. Throughput
DVB-RCS	Digital video broadcast	Parallel conc. of 8-state conv. codes	68 Mb/s (rate 7/8)
IEEE 802.16	Wireless networking (MAN)	Turbo product codes	25 Mb/s (rate 5/6)
3GPP UMTS	Wireless cellular	Parallel conc. of 8-state conv. codes	2 Mb/s (rate 1/3)
3GPP2 CDMA2000	Wireless cellular	Parallel conc. of 8-state conv. codes	3.09 Mb/s (rate 1/5)
CCSDS	Space telemetry	Parallel conc. of 16-state conv. codes	384 kb/s (rate 1/2)

2.1: The Structure of Turbo Code

2.1.1: Encoder of Turbo Code

The turbo encoder is built with a parallel concatenation of two recursive systematic convolutional (RSC) encoders and separating them by an interleaver. The two RSC encoders are also called component codes of the turbo code. The interleaver is a pseudo-random block scrambler. In a pseudo-random interleaver, a block of N input bits is read into the interleaver and read out pseudo-randomly. Note that the encoder structure is called parallel concatenation because each RSC encoder operates on the same set of input data but in different order, due to the presence of an interleaver. A block diagram of a rate 1/3 turbo encoder with constraint length 3 is shown in Fig 2.1. The generator matrix for a rate 1/2 component RSC code can be represented as polynomial matrix G(D) or octal matrix G.

$$G(D) = \begin{bmatrix} 1 & \frac{1+D}{1+D+D^2} \end{bmatrix} \text{ or } G=[1 \ \ \frac{5}{7}] \tag{2.1}$$

In a rate 1/2 RSC encoder, each input massage is encoded as one systematic bit and one parity check bits. In turbo encoder, two RSC encoders encode the same input message sequence, individually, but in the different order. In order to increase the code rate of turbo code, the systematic bits of RSC Encoder 2 are not transmitted. Therefore, the output sequence of turbo encoder should be $\{X_0, Y_{10}, Y_{20}, X_1, Y_{11}, Y_{21}, \ldots\}$, and the overall code rate is 1/3 as shown in Fig. 2.1.

After encoding a block included N input messages, transmission of some tail bits to drive the encoder to the all zero state is required. It makes sure that the initial state for next block is the all-zero state. The numbers of tail bit are equal to register numbers of RSC encoder. Since the component encoders are recursive, it is impossible to terminate the trellis to all zero state only by transmitting dummy zeros directly. A simple solution to solve this problem is provided in Fig 2.2. The switch of each component encoder is set to position “A” for inputting N messages, and then the position of switch is changed to “B” for 2 tail bits in an example of a rate 1/2 RSC encoder with memory order 2. This will force all registers of RSC encoder to zeros and thus the trellis return to all zero state.

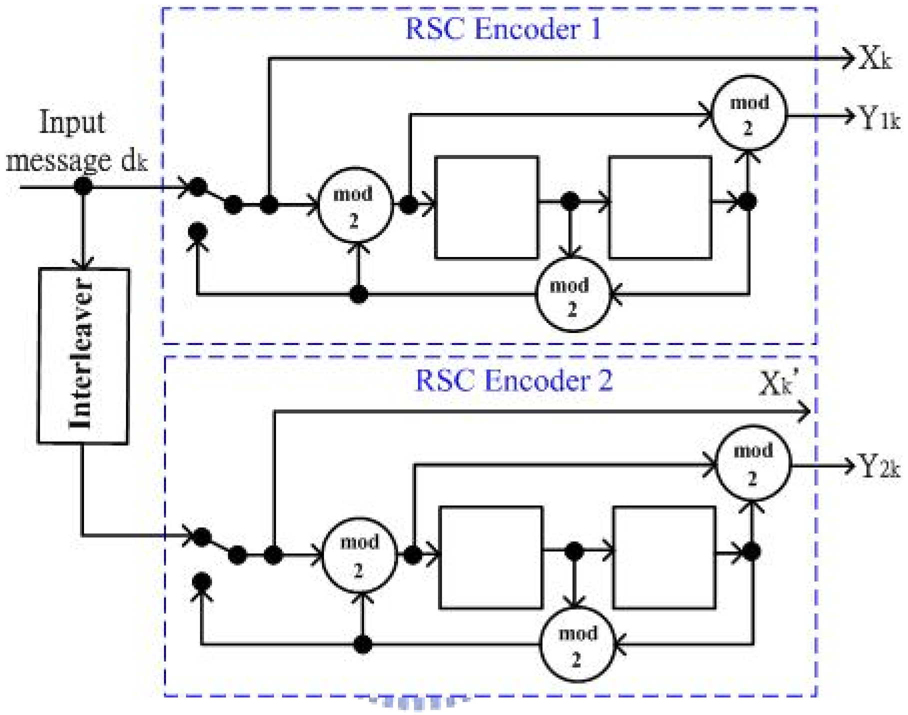

Fig. 2.1 The structure of turbo encoder

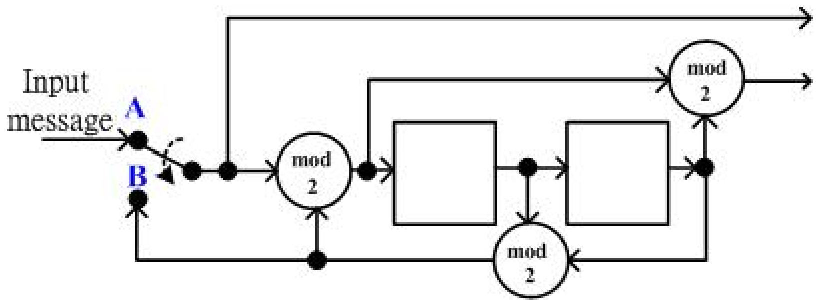

Fig. 2.2 Trellis termination for component RSC encoder

2.1.2: Interleaver

In the turbo code, the interleaver is to construct a long block code from small memory component convolutional codes. Secondly, it spreads out the burst error and de-correlates the input of two RSC encoders that iterative decoding algorithm can be applied between two component decoders. Lastly, the interleaver can reduce low weight codewords that reduce the coding gain. In [2][3], it uses analytical upper-bounding technique to show that a turbo code with RSC component codes can produce an interleaving gain. The performance upper-bound of turbo code is based on uniform random interlever. The result points out that the bit-error-probability upper bound of turbo code is approximately proportional to 1/N, where N is the block length of interleaver, and the factor "1/N" is also called the interleaver gain. The error performance of turbo code at low BER's is dominated by the interleaver size. Low weight codwords dominate the turbo code performance at high BER's, and are produced by low weight input sequence. However, the interleaver structure affects the mapping of low weight input sequences to the interleaver output and its size and structure play an important role in the asymptotic performance of the turbo code.

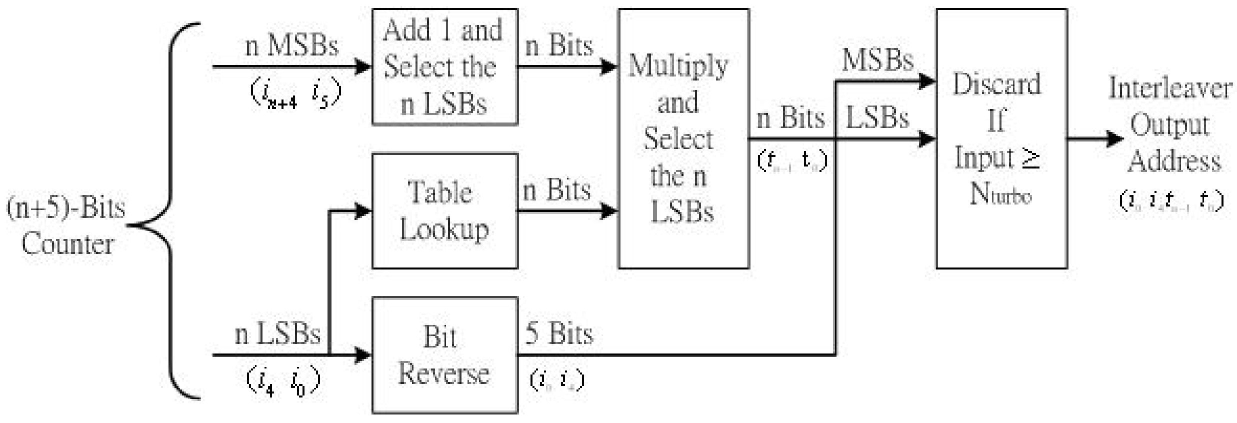

Fig. 2.3 The 3GPP2 inteleaver for 3GPP2 standard

Table 2.2 3GPP2 interleaver parameter

3GPP2 Interleaver Block Length N_{turbo}	3GPP2 Interleaver Parameter n
210	3
378	4
402	4
570	5
762	5
786	5
1,146	6
1,530	6
1,554	6
2,298	7
2,322	7
3,066	7
3,090	7
3,858	7
4,602	8
6,138	8
9,210	9
12,282	9
20,730	10

In this thesis, the performances of turbo code will be compared with a complex interleaver, S-Random interleaver[4] and a structured interleaver, 3GPP2 interleaver [5], so only both interleaver are introduced. An S-Random address generator, or an S-Random interleaver, is defined as follows. A new randomly selected integer is compared to the S_1 previously selected integers. If the absolute value of the difference between the current selected integer and any of the S_1 previous selected integers is smaller than S_2, then the current selected integer is rejected. The S_1 and S_2 are two integers smaller than N. This process is repeated until all N integers are selected. The 3GPP2 address generator provides the different block length N from maximum block length of 20,739 to minimum block length of 378. Fig 2.3 shows the procedure of 3GPP2 permutation in 3GPP2 standard. Detail supported block lengths and its corresponded parameter n are listed in Table 2.2. In next section, the performance comparison between S-Random interleaver and 3GPP2 interleaver will show.

2.1.3: Decoder of Turbo Code

Exchanging soft information among the receiver parts is the main idea for turbo iterative decoding. The principle is illustrated in Fig. 2.4, where $\boldsymbol{R_s}$ is the received systematic information sequence, $\boldsymbol{R_{p1}}$ and $\boldsymbol{R_{p2}}$ are the received parity information sequences generated by the first component RSC encoder and the second component RSC encoder, separately.

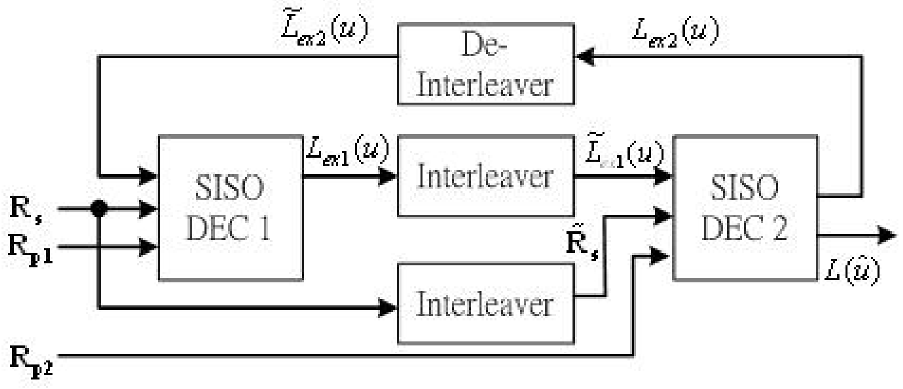

Fig. 2.4 The structure of turbo decoding

The general turbo decoding consists of two component decoders, which are soft-in/soft-out (SISO) decoders serially concatenated via an interleaver and a de-interleaver. The component decoder can be implemented based on either maximum a posteriori probability (MAP) algorithm [6], or soft-output Viterbi algorithm (SOVA) [7], which will be discussed particularly in the next section. In first iteration, the decoder, DEC 1, takes $\boldsymbol{R_s}$ and $\boldsymbol{R_{p1}}$ as input to produces a soft output $\boldsymbol{L_{ex1}}$ called

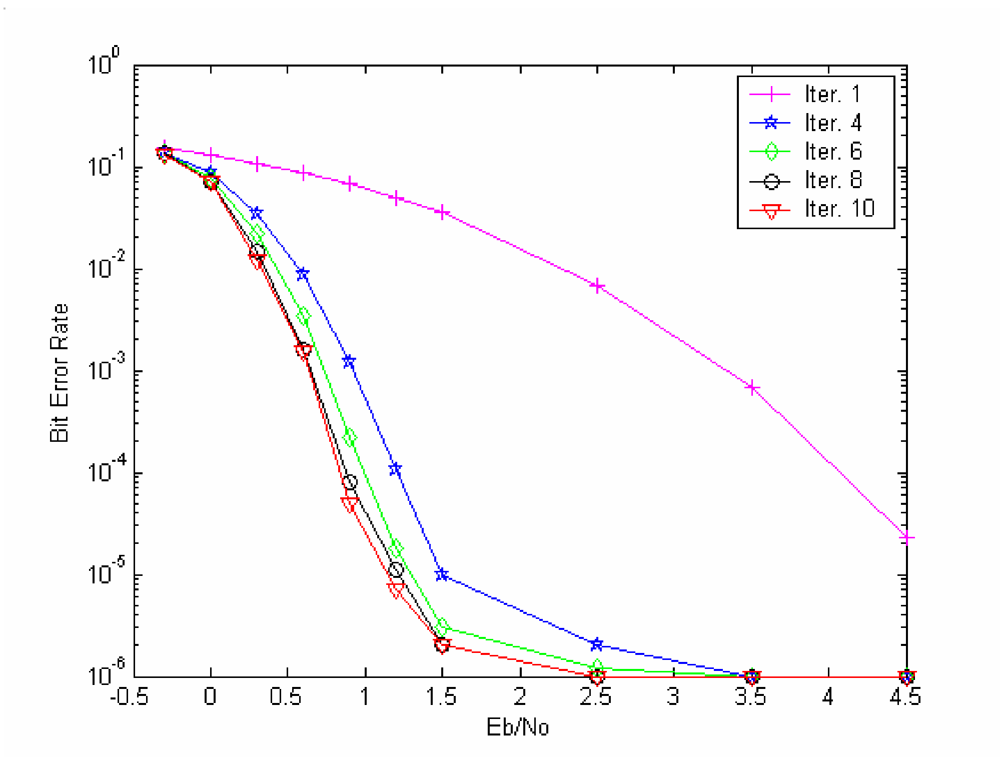

Fig. 2.5 Performance comparison under different iteration numbers in S-Random interleaver, S=10 (N=1530, code rate=1/3, state 4, MAP algorithm, BPSK)

extrinsic information, which is interleaved and used as the a priori probabilities of the information sequence for the second decoder, DEC 2. Then the DEC 2 takes $\boldsymbol{R_{p2}}$**,** interleaved extrinsic information and received systematic information sequences from DEC 1 as input to procedure a soft output $\boldsymbol{L_{ex2}}$, which is de-interleaved and used to improve the estimate of the a priori probabilities of the information sequence for first decoder, DEC 1. During the iteration decoding process, the decoder performance can be improved, as the number of iterations increases. However, the correlation between two component decoder is also raised up. There is no significant performance improvement after a certain number of iterations. Then the last stage decoding makes a hard decision after de-interleaver. A performance comparison under different iteration numbers and different iterleaver structures is shown in Fig. 2.5 and Fig. 2.6. Both simulation results are stopped when 35 error blocks have happened or 500000 blocks have run completely.

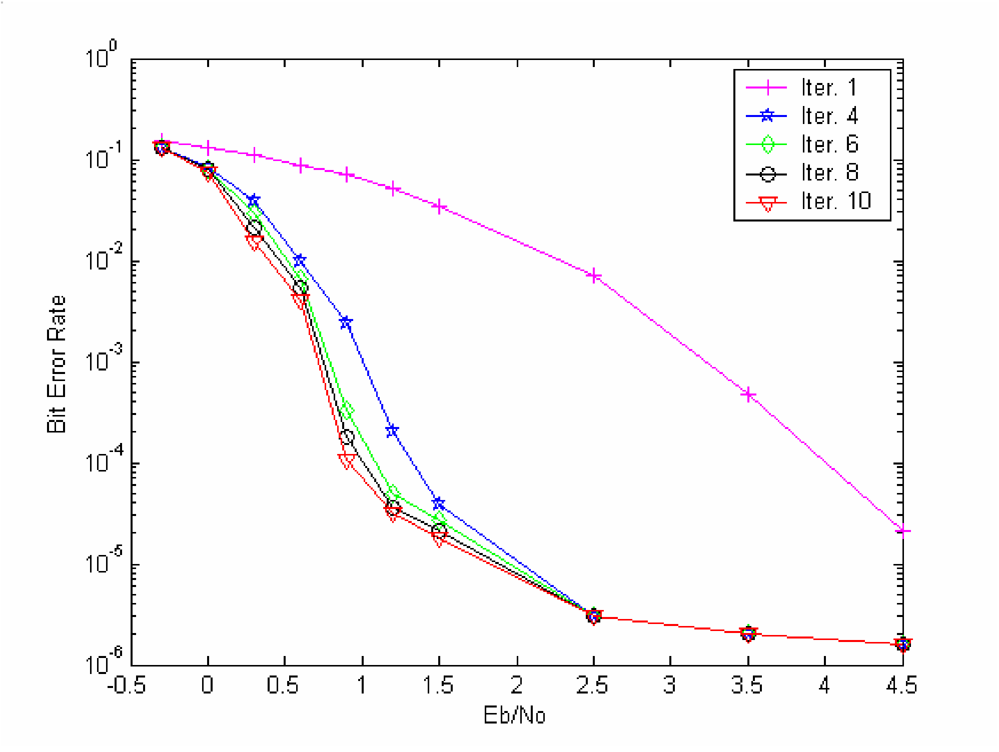

Fig. 2.6 Performance comparison under different iteration numbers in 3GPP2 interleaver (N=1530, code rate=1/3, state 4, MAP algorithm, BPSK)

2.1.4: Error Floor Effect

The asymptotic performance of turbo code can be separate into two regions shown in Fig. 2.5 and Fig. 2.6. One region about form 0dB to 1.5dB in Fig. 2.5 and Fig. 2.6 is called waterfall region. In this region, bit-error-rate (BER) has a sharply drop slop at low signal-to-noise ratio (SNR). The other about form 2.5dB to 4.5dB is called error floor region when the BER starts to decrease quite slowly at high SNR. The error floor phenomenon is due to relative small code free distance. Consider the relation between the minimum free distance and the bit error probability in turbo coding, which can be expressed by [4]

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

where B_{free} is the average number of ones on the minimum free distance path in the overall turbo code trellis, R is the code rate, E_b is the received bit energy, N_0 is the Gaussian noise one sided power spectral density, and d_{free} is the code minimum free distance. The value of d_{free} depends on the generator polynomials and the interleaver structure.

2.2: Decoding Algorithm for Turbo Code

Both the maximum a posteriori probability (MAP) algorithm [4] [6] [9] and soft-output Viterbi algorithm (SOVA) [7] are common used techniques for turbo decoding. The SOVA exploits maximum likelihood (ML) algorithm to find the codewords and to minimize the word error probability which is defined as the probability that the transmitted and estimated code sequences are unequal. The MAP algorithm minimizes the symbol (or bit) error probability which is defined as the probability that transmitted and estimated code symbols (or bits) are unequal. If the code sequences are equally likely, MAP and SOVA decoders are equivalent in terms of word error probability. In this section, the focus will be MAP algorithm, and the SOVA will be skipped, because it has been proved that the MAP algorithm is the optimal decoding method for turbo code while comparing with SOVA [8].

2.2.1: The MAP Algorithm

Due to minimizing the symbol (or bit) error probability, the MAP algorithm generates the soft output in the form of $P(u_k|\boldsymbol{R})$, a posteriori probability based on the received code sequence ***R,*** to estimate the hard value for each transmitted information bits u_k. It computes the logarithm of likelihood ratio (LLR), and the logarithm is the natural logarithm.

$$L(\hat{u}_k) = L(u_k \mid \mathbf{R}) = \log \frac{P(u_k = +1 \mid \mathbf{R})}{P(u_k = -1 \mid \mathbf{R})} \tag{2.3}$$

for $1 \le k \le N$, where N is the interleaver length, and compares this value to a zero threshold to determine the hard estimate as

$$\hat{u}_k = \begin{cases} 1 & \text{if } L(\hat{u}_k) \ge 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

a rate 1/2 state 4 (memory order 2) RSC encoder and its trellis diagram are shown in Fig 2.7 as an example, and its decoding trellis diagram is shown in Fig. 2.8. Note that the solid lines represent the transitions from S_{k-1} to S_k caused by the input information bit u_k of -1, and the dashed lines represent the transitions from S_{k-1} to S_k caused by the input information bit u_k of +1. Then, the equation.(2.3) can be further expressed as

$$L(\hat{u}_k) = \log \frac{P(u_k = +1 \mid \mathbf{R})}{P(u_k = -1 \mid \mathbf{R})} = \log \frac{\sum_{u_k=+1} P(S_{k-1}, S_k, \mathbf{R})}{\sum_{u_k=-1} P(S_{k-1}, S_k, \mathbf{R})} \tag{2.5}$$

Here, the numerator and denominator are the sum of joint probabilities for all existing transitions from state S_{k-1} to S_k that corresponding to the information bit u_k of +1 and -1, respectively. In order to compute equation (2.5), the joint probability $P(S_{k-1},S_k,\boldsymbol{R})$ was needed to calculate. Now we defined following metrics:

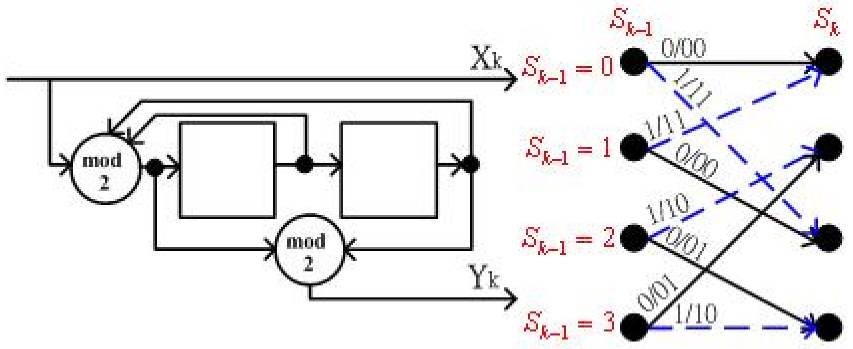

Fig. 2.7 A rate 1/2 state 4 RSC encoder and its trellis diagram

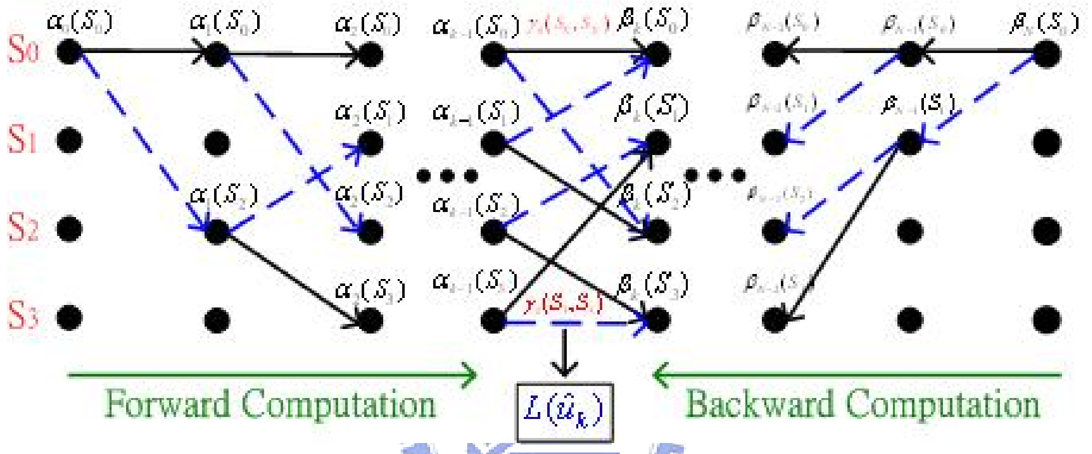

Fig. 2.8 The decoding trellis diagram of a rate 1/2 state 4 RSC encoder

- **The forward recursion metric α:**

$$\alpha_k(S_k) = \log P(S_k, \mathbf{R}_{j\le k}) \tag{2.6}$$

- **The backward recursion metric β:**

$$\beta_k(S_k) = \log\{P(\mathbf{R}_{j>k} \mid S_k)\} \tag{2.7}$$

- **The branch transition metric γ:**

$$\gamma_k(S_{k-1}, S_k) = \log\{P(S_k, \mathbf{R}_k \mid S_{k-1})\} \tag{2.8}$$

Assume the code sequence after encoding is transmitted through the discrete memoryless channel (DMC), and then the probability $P(S_{k-1},S_k,\mathbf{R})$ can be decomposed as three terms:

$$P(S_{k-1},S_k,\mathbf{R}) = \underbrace{P(S_{k-1},\mathbf{R}_{j<k})}_{e^{\alpha_{k-1}(S_{k-1})}} \bullet \underbrace{P(S_k,\mathbf{R}_k \mid S_{k-1})}_{e^{\gamma_k(S_{k-1},S_k)}} \bullet \underbrace{P(\mathbf{R}_{j>k}S_k)}_{e^{\beta_k(S_k)}} \tag{2.9}$$

Now we can further define the equations (2.6) (2.7) and (2.8) as follow:

- **The definition of forward recursion metric α:**

$$\begin{aligned}
\alpha_k(S_k) &= \log\{P(S_k,\mathbf{R}_{j\le k})\} \\
&= \log\sum_{S_{k-1}} P(S_{k-1},S_k,\mathbf{R}_{j\le k},\mathbf{R}_k) \\
&= \log\sum_{S_{k-1}} P(S_{k-1},\mathbf{R}_{j\le k})\bullet P(S_k,\mathbf{R}_k \mid S_{k-1},\mathbf{R}_{j\le k}) \\
&= \log\sum_{S_{k-1}} e^{\alpha_{k-1}(S_{k-1})}\bullet P(S_k,\mathbf{R}_k \mid S_{k-1}) \\
&= \log\sum_{S_{k-1}} e^{(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k))}
\end{aligned} \tag{2.10}$$

Note that since the trellis of encoding diverges from state zero, the initial condition of the forward recursion metric should be set as $\alpha_0(0) = 0$ and $\alpha_0(S_0) = -\infty$ for $S_0 \neq 0$.

- **The definition of backward recursion metric β:**

$$\begin{aligned}
\beta_k(S_k) &= \log\{P(\mathbf{R}_{j>k} \mid S_k)\} \\
&= \log\sum_{S_{k+1}} P(S_{k+1},\mathbf{R}_{j>k} \mid S_k) \\
&= \log\sum_{S_{k+1}} \frac{P(S_{k+1},\mathbf{R}_{k+1},\mathbf{R}_{j>k+1},S_k)}{P(S_k)} \\
&= \log\sum_{S_{k+1}} \frac{P(\mathbf{R}_{j>k+1} \mid S_{k+1},\mathbf{R}_{k+1},S_k)\bullet P(S_{k+1},\mathbf{R}_{k+1},S_k)}{P(S_k)} \\
&= \log\sum_{S_{k+1}} e^{\beta_{k+1}(S_{k+1})}\bullet\frac{P(S_{k+1},\mathbf{R}_{k+1},S_k)}{P(S_k)} \\
&= \log\sum_{S_{k+1}} e^{(\beta_{k+1}(S_{k+1})+\gamma_k(S_k,S_{k+1}))}
\end{aligned} \tag{2.11}$$

Note that since the trellis of encoding converges from state zero, the initial condition of the backward recursion metric should be set as $\beta_N(0) = 0$ and $\beta_N(S_N) = -\infty$ for $S_N \neq 0$.

- **The definition of branch transition metric γ:**

$$\begin{aligned}
\gamma_k(S_{k-1},S_k) &= \log\{P(S_k,\mathbf{R}_k \mid S_{k-1})\} \\
&= \log\{P(S_k \mid S_{k-1})\bullet P(\mathbf{R}_k \mid S_{k-1},S_k)\} \\
&= \log\{P_a(u_k)\bullet P(\mathbf{R}_k \mid u_k)\}
\end{aligned} \tag{2.12}$$

The term “$P_a(u_k)$” is well-known as a priori probability of u_k. According to the definition of log-likelihood ratio:

$$L_a(u_k) = \log\frac{P(u_k = +1)}{P(u_k = -1)} \tag{2.13}$$

$P(u_k)$ can be rewritten as

$$\begin{aligned} P(u_k = \pm 1) &= \frac{e^{\pm L_a(u_k)}}{1+e^{\pm L_a(u_k)}} \\ &= \frac{e^{-L_a(u_k)/2}}{1+e^{-L_a(u_k)}} \bullet e^{-L_a(u_k)\bullet u_k/2} \\ &= A_k \bullet e^{-L_a(u_k)\bullet u_k/2} \end{aligned} \tag{2.14}$$

And the term “$P(\mathbf{R}_k|u_k)$” is dependent on channel characteristic. For an additive white Gaussian noise (AWGN) channel, its log-likelihood ratio can be expressed as:

$$\begin{aligned} L(\mathbf{R}_k \,|\, u_k) &= \log\frac{P(\mathbf{R}_k \,|\, u_k = +1)}{P(\mathbf{R}_k \,|\, u_k = -1)} \\ &= \log\frac{\exp(-\frac{E_s}{N_0}(\mathbf{R}_{k,0} - X_k)^2)\,|_{u_k=+1}}{\exp(-\frac{E_s}{N_0}(\mathbf{R}_{k,0} - X_k)^2)\,|_{u_k=-1}} + \log\frac{\prod_{v=1}^{2}\exp(-\frac{E_s}{N_0}(\mathbf{R}_{k,v} - Y_{k,v})^2)\,|_{u_k=+1}}{\prod_{v=1}^{2}\exp(-\frac{E_s}{N_0}(\mathbf{R}_{k,v} - Y_{k,v})^2)\,|_{u_k=-1}} \\ &= 4\frac{E_s}{N_0}(\mathbf{R}_{k,0}\bullet X_k + \sum_{v=1}^{2}\mathbf{R}_{k,v}\bullet Y_{k,v}) = L_c(\mathbf{R}_{k,0}\bullet X_k + \sum_{v=1}^{2}\mathbf{R}_{k,v}\bullet Y_{k,v}) \end{aligned} \tag{2.15}$$

Here, $Lc = 4*E_s/N_0$, and it is the channel reliability. In a similar way liking (2.14), the conditioned probability $P(\mathbf{R}_k|u_k)$ for systematic convolutional codes can be written as

$$P(\mathbf{R}_k \,|\, u_k) = B_k \bullet \exp(\frac{1}{2}L_c\mathbf{R}_{k,0}u_k + \frac{1}{2}\sum_{v=1}^{2}L_c\mathbf{R}_{k,v}Y_{k,v}) \tag{2.16}$$

Noted that X_k is equal to u_k, for systematic codes. The terms A_k and B_k in (2.14) and (2.16) are equal for all transitions at the same time index, and hence will cancel out in the ratio of (2.5). Therefore, the branch transition probability can be reduced to the expression:

$$\begin{aligned} \gamma_k(S_{k-1}, S_k) &= \log\{P(u_k)\bullet P(\mathbf{R}_k \,|\, u_k)\} \\ &= \frac{1}{2}u_k(L_c\mathbf{R}_{k,0} + L_a(u_k)) + \frac{1}{2}\sum_{v=1}^{2}(L_c\mathbf{R}_{k,v}Y_{k,v}) \\ &= \frac{1}{2}u_k(L_c\mathbf{R}_{k,0} + L_a(u_k)) + \gamma_k^{(e)}(S_{k-1}, S_k) \end{aligned} \tag{2.17}$$

Substituting (2.10), (2.11) and (2.17) into (2.5), The LLR can be further expressed as

$$\begin{aligned}
L(\hat{u}_k) &= \log \frac{\sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=+1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}}{\sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=-1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}} \\
&= L_c \mathbf{R}_{k,0} + L_a(u_k) + \log \frac{\sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=+1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k^{(e)}(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}}{\sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=-1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k^{(e)}(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}} \qquad (2.18) \\
&= L_c \mathbf{R}_{k,0} + L_a(u_k) + L_{ex}(u_k)
\end{aligned}$$

The term $L_{ex}(u_k)$ is called extrinsic information. Since the extrinsic information is a function of the redundant information that introduces by the RSC encoder, it is independent on systematic input and a priori information, $L_a(u_k)$, from LLR. In general, extrinsic information has the same sign as u_k. Therefore, it is helpful to estimate the priori probabilities of code sequence for next component decoder.

2.2.2: The Max-Log-MAP Algorithm

In last section the MAP algorithm is derived. As we can see, this algorithm is too difficult in practice because of a high number of exponentiation and multiplication operations. For simplifying the implementation complexity of MAP decoders, an approximation of MAP algorithm termed Max-Log-MAP algorithm [10] [11] was derived. By using the approximation formula **max** function

$$\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) \approx \max_{i \in \{1,2,\cdots,n\}} \delta_i \qquad (2.19)$$

The equation (2.18) can be further simplified as:

$$L(\hat{u}_k) = \log \frac{\sum\limits_{\substack{(S_{k-1},S_k)\\u_k=+1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}}{\sum\limits_{\substack{(S_{k-1},S_k)\\u_k=-1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)}}$$

$$\approx \max_{\substack{(S_{k-1},S_k)\\u_k=+1}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k) + \beta_k(S_k)) - \max_{\substack{(S_{k-1},S_k)\\u_k=-1}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k) + \beta_k(S_k)) \quad (2.20)$$

Similarly, the forward recursive and backward recursive metrics (2.10) (2.11) can individually be expressed as

$$\alpha_k(S_k) = \log \sum_{S_{k-1}} e^{(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k))} \approx \max_{S_{k-1}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k)) \quad (2.21)$$

and

$$\beta_k(S_k) = \log \sum_{S_{k+1}} e^{(\beta_{k+1}(S_{k+1}) + \gamma_k(S_k,S_{k+1}))} \approx \max_{S_{k+1}}(\beta_{k+1}(S_{k+1}) + \gamma_k(S_k,S_{k+1})) \quad (2.22)$$

Both equations involved computations are add-compare-selection operations, which are similar to the path metric updating of Viterbi algorithm. Thus multiplications in the MAP algorithm are replaced by additions in the Max-Log-MAP algorithm, and the exponentiation operations can be avoided.

2.2.3: The Log-MAP Algorithm

The performance of Max-Log-MAP algorithm is suboptimal because of its use of the max function (2.19) to reduce the complexity of MAP algorithm. The Max-Log-MAP algorithm can be modified through the use of the Jacobian algorithm. The modified algorithm, called Log-MAP algorithm, is equivalent to the MAP algorithm, but without its major disadvantages. The Jacobian algorithm [10] is

$$\begin{aligned}\log(e^{\delta_1}+e^{\delta_2}) &= \max(\delta_1,\delta_2)+\log(1+e^{-|e^{\delta_2}-e^{\delta_1}|}) \\ &= \max(\delta_1,\delta_2)+f_c(|\delta_2-\delta_1|)\end{aligned} \tag{2.23}$$

where $f_c(\cdot)$ is a correction function, which can be implemented using a one-dimensional look-up table. By a recursive operation of (2.23), the expression (2.19) can be computed exactly.

$$\begin{aligned}\log(e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_n}) &= \log(\Delta+e^{\delta_n}),\quad \Delta=e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_{n-1}}=e^{\delta} \\ &= \max(\log\Delta,\delta_n)+f_c(|\log\Delta-\delta_n|) \\ &= \max(\delta,\delta_n)+f_c(|\delta-\delta_n|)\end{aligned} \tag{2.24}$$

Now we can substitute (2.21) and (2.22) into (2.23), the forward and backward recursive metrics can be represented as

$$\begin{aligned}\alpha_k(S_k) &= \log\sum_{S_{k-1}} e^{(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k))} \\ &= \max_{S_{k-1}}{}^*(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k))\end{aligned} \tag{2.25}$$

and

$$\begin{aligned}\beta_k(S_k) &= \log\sum_{S_{k+1}} e^{(\beta_{k+1}(S_{k+1})+\gamma_k(S_k,S_{k+1}))} \\ &= \max_{S_{k+1}}{}^*(\beta_{k+1}(S_{k+1})+\gamma_k(S_k,S_{k+1}))\end{aligned} \tag{2.26}$$

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

$$\max{}^*(\cdot)=\max(\delta_1,\delta_2)+f_c(|\delta_2-\delta_1|) \tag{2.27}$$

Then, the (2.20) can be expressed as

$$\begin{aligned}L(\hat{u}_k) = &\max_{\substack{(S_{k-1},S_k)\\ u_k=+1}}{}^*(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k)+\beta_k(S_k)) \\ &-\max_{\substack{(S_{k-1},S_k)\\ u_k=-1}}{}^*(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k)+\beta_k(S_k))\end{aligned} \tag{2.28}$$

The performance of Log-MAP algorithm is identical to the performance of MAP algorithm. However, by calculating $f_c(\cdot)$, its complexity is increased in comparison with Max-Log-MAP algorithm. For simplified the computation of correction function, it is usually stored in a pre-computed table. The table is only one dimensional, since the correction only depends on $|\delta_2-\delta_1|$. Then 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 $|\delta_2-\delta_1|$ ranging between 0 and 5, and no improvement can be achieved with using a finer representation.

2.2.4: SNR Sensitivity of Max-Log-MAP and Log-MAP Algorithm

From the deductions of MAP algorithm and Log-MAP, they need SNR estimation to obtain the channel reliability L_c. The accurate SNR estimation cannot be ease approached, because it depends on channel characteristic. There are many researches have discussed the effect of SNR sensitivity, or SNR mismatch, on the bit error rate performance of turbo code. In [12], the simulation results indicate that the larger interleaver length is more sensitive to SNR estimation errors than the smaller one, and the turbo code with MAP or Log-MAP algorithm is more sensitive to SNR estimation errors on AWGN channel than on the BSC. In [13], the simulation shows that the underestimation of SNR is more detrimental than overestimation for tolerating a SNR mismatch without significant degradation. Compared with MAP or Log-MAP algorithm, the paper [14] has proven that the Max-Log-MAP does not require knowledge of the SNR, if $L(u_k)$, a priori information, is initialized to zero for each bit. Due to the **max** function operation, the $L(u_k)$ can be assumed to be proportional to SNR, and the SNR can be factored out in (2.20), (2.21) and (2.22). Then, the soft outputs are scaled with the SNR, but the hard decisions become SNR independent.

Although the performance of Max-Log-MAP algorithm is poor than MAP algorithm, but it has not the risk of serious SNR mismatch offset. So, it has been suggested in [14] that if channel characteristics change over time, the Max-Log-MAP algorithm is more suitable than Log-MAP in turbo decoding. A simulation result form [14] is shown in Fig. 2.9.

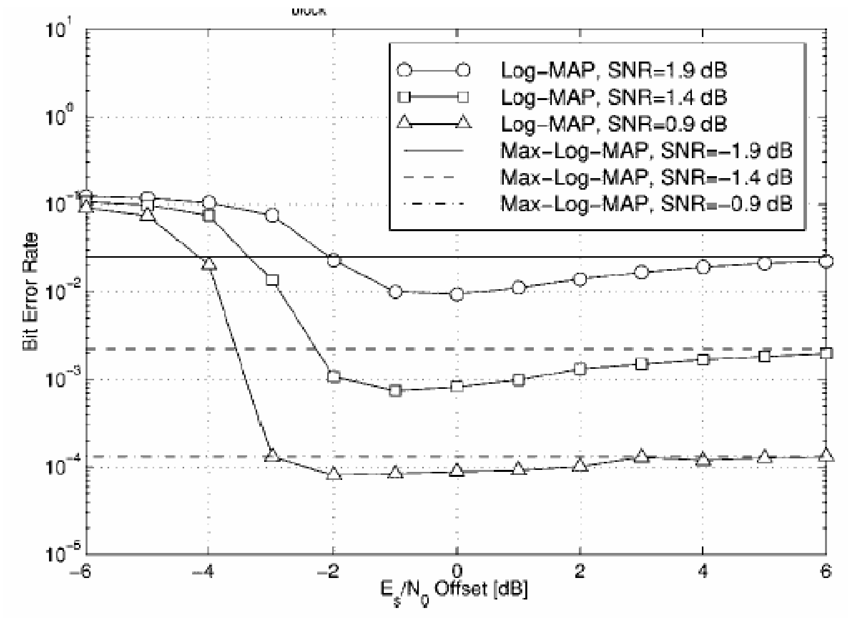

Fig. 2.9 SNR estimation offset, AWGN channel. R=1/2, m=2, L_{block}=600, random interleaver, 10 iterations.

2.3: Sliding Window Method for Turbo Decoding

In the standard turbo decoding algorithm included MAP algorithm, Log-MAP algorithm and Max-Log-MAP algorithm, the decisions are based on forward and backward operations, and the backward recursive operation initials from the end of decoding trellis shown in Fig. 2.8. So the decoding process starts after a received delay that is equal to received sequence length N. For large sequence lengths, the memory required for hardware implementation of turbo decoder is huge, and long output latency is also introduced. For example, the maximum block length of 3GPP2 standard is 20730, which means at least 2*20730 metrics for forward and back recursive need to be stored, and the received delay is 20730 time slots. These are main disadvantages of turbo code for real time applications.

These problems, in the main, cause by that long block length can not divide into several short sub-blocks, since the unknown initial state of backward recursive operation will damage the performance of turbo code. One method to overcome this problem is called sliding window algorithm [15]. The algorithm utilizes the fact that the backward recursive metrics can be highly reliable even without knowing the initial state if the backward recursion goes long enough. Fig. 2.10 illustrated the sliding window process diagram.

Initially, the received sequence is divided into several sub-blocks, with a sub-block length of L. L is normally about five times constraint length of component encoder in turbo code, and called the convergence length. For forward recursive operation, the end of each sub-block i is the initial of next sub-block i+1, and the initial metric values of each sub-block i are inherited from the last metric values of previous sub-block i-1. For backward recursive operation, a dummy backward recursion β_{pre} is employed to find the beginning metric values for β recursion. However, the forward recursion computes the forward path metrics α and storing these values into memory. In parallel, an additional dummy backward recursion β_{pre} is performed in the next sub-block. As soon as the β_{pre} operation is finished, the true backward recursion β operation starts and the decoder also stars making soft decision based on both forward and backward recursions.

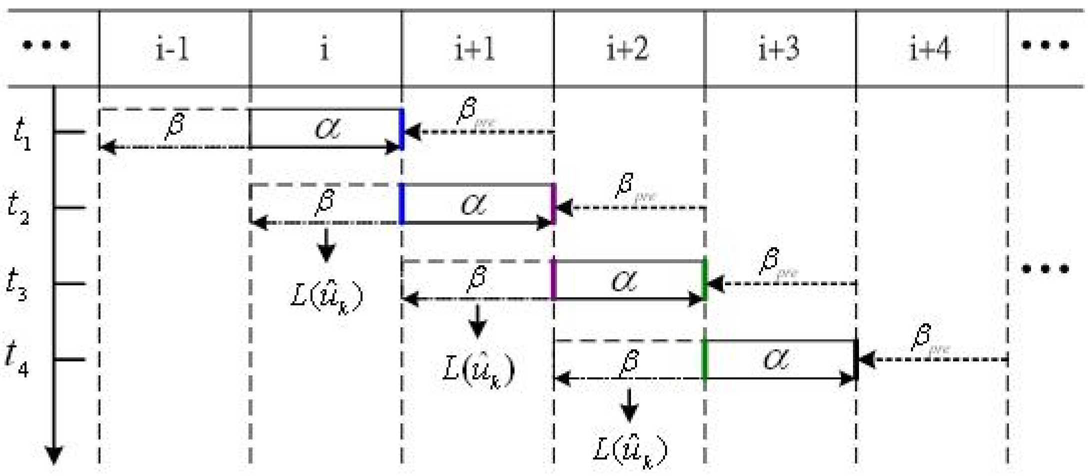

Fig. 2.10 The diagram of sliding window algorithm

Chapter3 Ternary Turbo Code

Compared with turbo code over GF(2), also called binary turbo code, the turbo code over GF(3) is called ternary turbo code. Its basic elements as reliability-based, iterative decoding and random interleaver are equivalent to binary turbo code. However, there are some diversity between ternary turbo code and binary turbo code, due to component convoluational codes over different fields. In this chapter, we'll describe the principle of ternary turbo coding.

3.1: The Structure of Ternary Turbo Code

The ternary turbo encoder is built with a parallel concatenation of two component RSC encoders and separating them by an interleaver. Since component RSC encoder is over GF(3), we'll call it component GF(3) RSC encoder. A block diagram of a rate 1/3 ternary turbo encoder with constraint length 3 is shown in Fig. 3.1. The generator matrix for a rate 1/2 component GF(3) RSC encoder can be represented as polynomial matrix G(D) [17].

$$G(D)=\begin{bmatrix}1 & \dfrac{1+D+2D^2}{2+D+D^2}\end{bmatrix} \tag{3.1}$$

From (3.1), it can be observed that the connections of ternary turbo encoder have connection weights, but, in binary turbo encoder, the connection is only "on" or "off". Since the operation principles of ternary turbo encoder and binary turbo encoder are similar, the design rules as to maximize the minimum output weight from low weight input of binary turbo encoder can be copied to design ternary turbo encoder. However, ternary turbo encoder also needs to implement trellis termination after encoded a block messages. The procedure of trellis termination is shown in Fig.3.2. The switch of each component encoder is set to position "A" for inputting N messages, and then the position of switch is changed to “B” for tail bits. Then trellis of ternary turbo encoder will return to all zero state. Note that the sun of W_1 and W_2 in component GF(3) RSC encoder must be equal to 3.

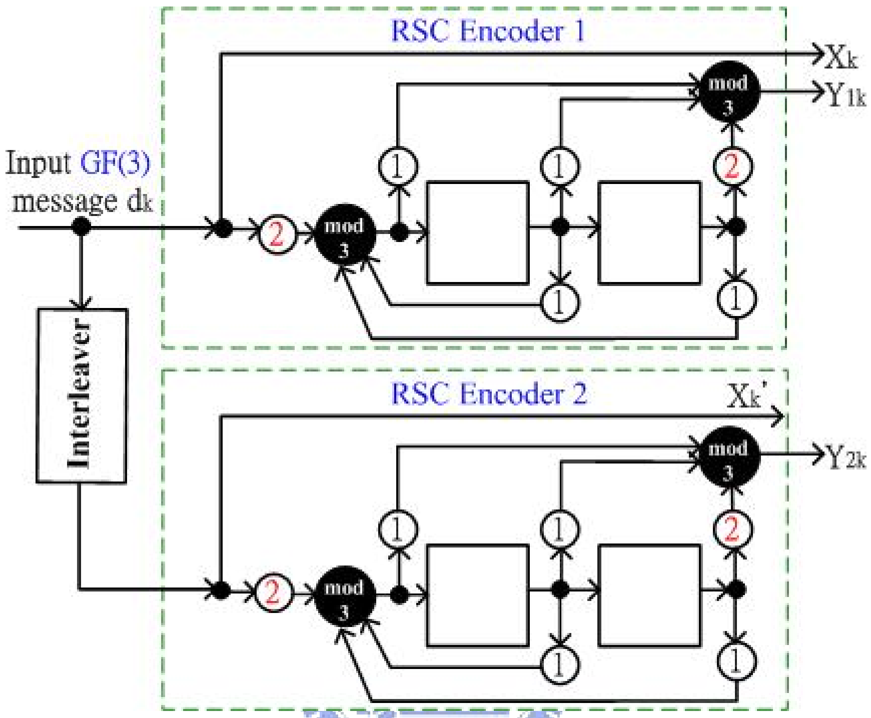

Fig. 3.1 The structure of ternary turbo encoder

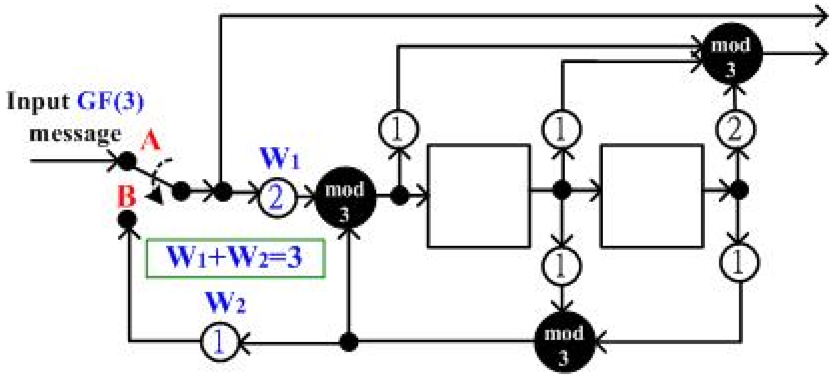

Fig. 3.2 Trellis termination for component GF(3) RSC encoder

In section 2.1.2, we have discussed the function and the property of interleaver. Due to same operation principles of ternary and binary turbo encoders unless weight distribution of their component encoders; the same capability for interleaver will be expected by intuition. In the performance simulation shown in Fig. 3.3 and Fig. 3.4, the S-Random interleaver has better BER performance than 3GPP2 interleaver's, and the result is as same as the performance of Fig. 2.5 and Fig. 2.6 in binary turbo code. So, it can be observed that interleaver be able to works well in binary turbo code as well as in ternary turbo code, for 'good' component encodes

Since the ternary turbo decoding which consists of two component decoders exchanged soft information among the receiver parts has same structure as binary turbo decoding shown in Fig. 2.4. And the component decoder can also be implemented based on either MAP algorithm, or SOVA, which will be discussed particularly in the next section. Hence, the more detail about the procedure of ternary turbo decoding, please refer to section 2.1.3. A performance comparison under different iteration numbers and different interleaver structures in ternary turbo code is shown in Fig. 3.3 and Fig. 3.4.

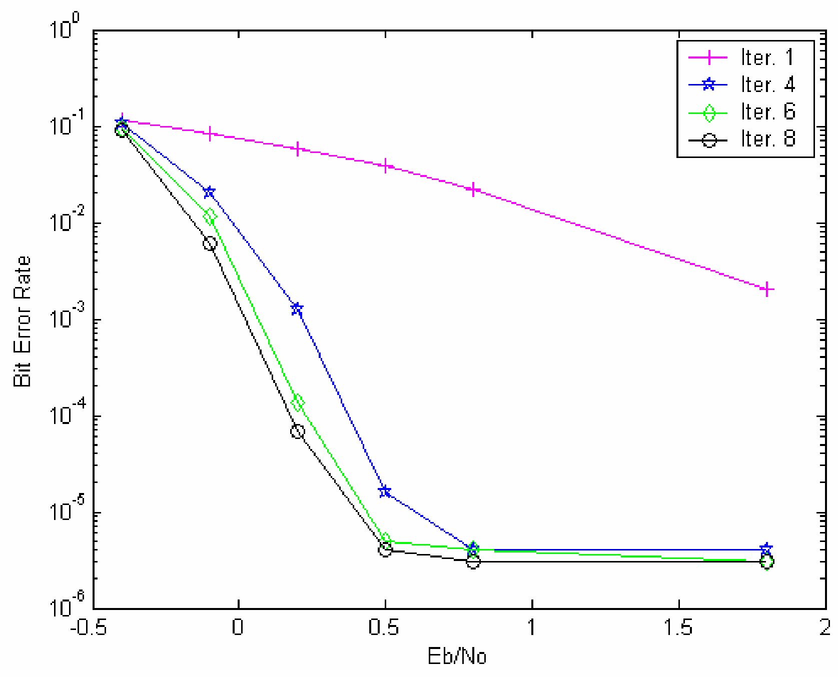

Fig. 3.3 Performance comparison under different iteration numbers in S-Random interleaver S=10 (N=1530, code rate=1/3, state 9, MAP algorithm, 3PSK)

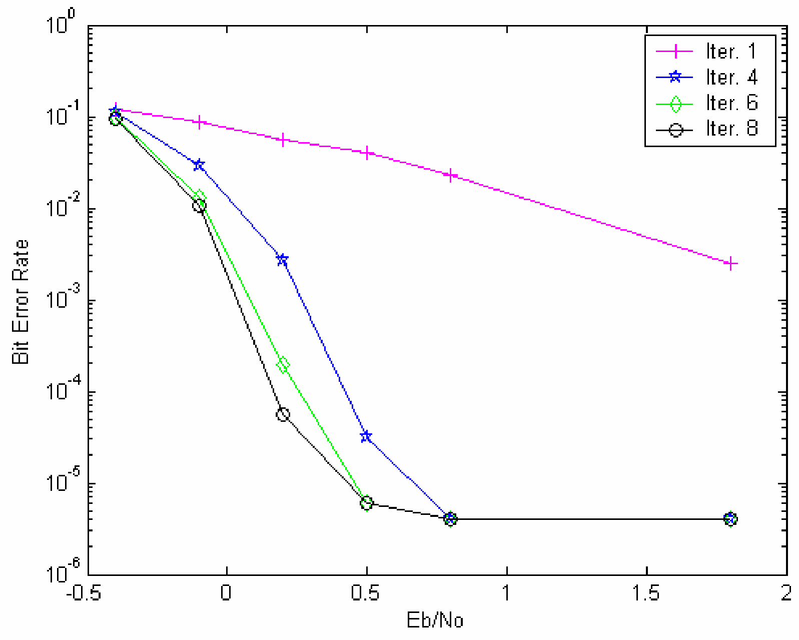

Fig. 3.4 Performance comparison under different iteration numbers in 3GPP2 interleaver (N=1530, code rate=1/3, state 9, MAP algorithm, 3PSK)

It has known that the error floor phenomenon is due to relative small code free distance. At high SNR, the bit error probability of ternary turbo coding with 3PSK modulation can refer to bit error probability of binary turbo code and expressed as:

$$P_b \propto B_{free} Q(\sqrt{2d_{free} R \frac{E_b}{N_0}}) \tag{3.2}$$

where B_{free} is the average number of 'non-zero values' on the minimum free distance path in the overall turbo code trellis, R is the code rate, and d_{free} is the code minimum free distance and depends on the generator polynomials and the interleaver structure. For an example illustrated in Fig. 3.1, the d_{free} is 6 which are relatively low. Through the simulation results shown in Fig. 3.3 and Fig. 3.4, it can be observed the BER has a very low slope at high SNR.

3.2: Decoding Algorithm for Ternary Turbo Decoding

3.2.1: The MAP Algorithm

Assume that input bit u_k takes value {0, 1, 2} with the same probability, the MAP algorithm in ternary turbo code evaluates the logarithm of a posteriori probability (LAPP), $P(u_k|\mathbf{R})$, to minimizing the symbol (or bit) error probability. It can be expressed as:

$$\begin{cases} \Lambda_0(\hat{u}_k) = \log P(u_k = 0 \mid \mathbf{R}) = \sum\limits_{u_k=0} P(S_{k-1}, S_k, \mathbf{R}) \\ \Lambda_1(\hat{u}_k) = \log P(u_k = 1 \mid \mathbf{R}) = \sum\limits_{u_k=1} P(S_{k-1}, S_k, \mathbf{R}) \\ \Lambda_2(\hat{u}_k) = \log P(u_k = 2 \mid \mathbf{R}) = \sum\limits_{u_k=2} P(S_{k-1}, S_k, \mathbf{R}) \end{cases} \tag{3.3}$$

The hard estimate for $1 \le k \le N$, where N is the interleaver length, can be determined by following conditions:

$$\hat{u}_k = \begin{cases} 0 & \text{if } (\Lambda_0(\hat{u}_k) > \Lambda_1(u_k)) \text{ } and \text{ } (\Lambda_0(\hat{u}_k) > \Lambda_2(u_k)) \\ 1 & \text{if } (\Lambda_1(u_k) > \Lambda_0(\hat{u}_k)) \text{ } and \text{ } (\Lambda_1(u_k) > \Lambda_2(u_k)) \\ 2 & \text{otherwise} \end{cases} \tag{3.4}$$

A rate 1/2 state 3 (memory order 1) GF(3) RSC encoder and its trellis diagram are shown in Fig.3.5 as an example. Here, the solid lines represent the transitions form S_{k-1} to S_k caused by the input information bit u_k of 0, and transitions caused by input information bit u_k of 1 and 2 are individually represented dashed lines and dotted lines.

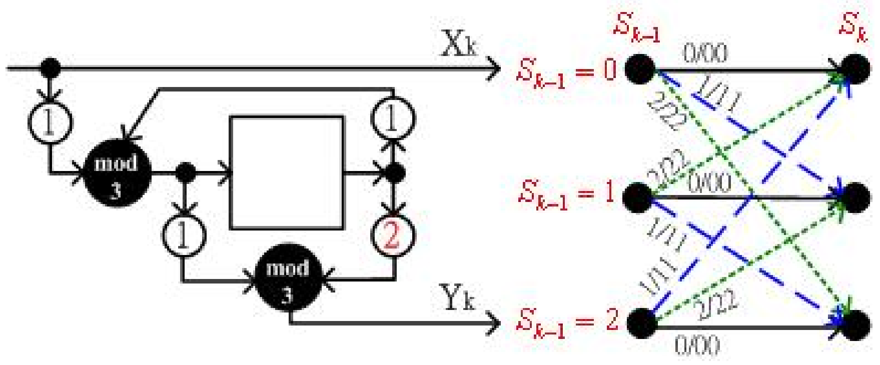

Fig. 3.5 A rate 1/3 state 3 GF(3) RSC encoder and its trellis diagram

In order to calculate equation (3.3), the joint probability $P(S_{k-1},S_k,\mathbf{R})$ was needed to computed. Based on the assumption of discrete memoryless channel, the joint probability can be decomposed as three terms.

$$\begin{aligned} P(S_{k-1},S_k,\mathbf{R}) &= \underbrace{P(S_{k-1},\mathbf{R}_{j<k})}\bullet\underbrace{P(S_k,\mathbf{R}_k \mid S_{k-1})}\bullet\underbrace{P(\mathbf{R}_{j>k}S_k)} \\ &= \quad e^{\alpha_{k-1}(S_{k-1})} \quad \bullet \quad e^{\gamma_k(S_{k-1},S_k)} \quad \bullet \quad e^{\beta_k(S_k)} \end{aligned} \tag{3.5}$$

Then, the following metrics can be defined: (The more details about the derivation of equations, please refer to section 2.2.1)

- **The forward recursion metric α:**

$$\begin{aligned} \alpha_k(S_k) &= \log\{P(S_k,\mathbf{R}_{j\le k})\} \\ &= \log\sum_{S_{k-1}} e^{(\alpha_{k-1}(S_{k-1})+\gamma_k(S_{k-1},S_k))} \end{aligned} \tag{3.6}$$

Note that since the trellis of encoding diverges from state zero, the initial condition of the forward recursion metric should be set as $\alpha_0(0)$ = 0 and $\alpha_0(S_0)$ = -∞ for $S_0 \neq 0$.

- **The backward recursion metric β:**

$$\begin{aligned} \beta_k(S_k) &= \log\{P(\mathbf{R}_{j>k} \mid S_k)\} \\ &= \log\sum_{S_{k+1}} e^{(\beta_{k+1}(S_{k+1})+\gamma_k(S_k,S_{k+1}))} \end{aligned} \tag{3.7}$$

Note that since the trellis of encoding converges from state zero, the initial condition of the backward recursion metric should be set as $\beta_N(0)$ = 0 and $\beta_N(S_N)$ = -∞ for $S_N \neq 0$.

- **The branch transition metric γ:**

$$\begin{aligned} \gamma_k(S_{k-1},S_k) &= \log\{P(S_k,\mathbf{R}_k \mid S_{k-1})\} \\ &= \log\{P(u_k)\bullet P(\mathbf{R}_k \mid u_k)\} \\ &= \log\{P(u_k)\} + \log\{(\sqrt{\frac{E_s}{\pi N_0}})^3 \bullet e^{-\frac{E_s}{N_0}(\mathbf{R}_{k,v}-\mathbf{X}_k)^2} \bullet \prod_{v=1}^{2} e^{-\frac{E_s}{N_0}(\mathbf{R}_{k,v}-\mathbf{Y}_{k,v})^2}\} \\ &= \log\{P(u_k)\} + A_k - \frac{E_s}{N_0}\bullet\{(\mathbf{R}_{k,v}-\mathbf{X}_k)^2 + \sum_{v=1}^{2}(\mathbf{R}_{k,v}-\mathbf{Y}_{k,v})^2\} \\ &\simeq \log\{P(u_k)\} - \frac{E_s}{N_0}\bullet(\mathbf{R}_{k,v}-\mathbf{X}_k)^2 - \frac{E_s}{N_0}\bullet\sum_{v=1}^{2}(\mathbf{R}_{k,v}-\mathbf{Y}_{k,v})^2 \\ &= \mathrm{L_a}(u_k) + L_c(u_k) + r_k^{(e)}(S_k,S_{k-1}) \end{aligned} \tag{3.8}$$

Noted that $\mathbf{X}_k$ is equal to u_k, for systematic codes. For 3PSK modulation, the $\mathbf{X}_k$ includes $\{X_{r,k}, X_{i,k}\}$, the real part and image part of x_k in signal constellation, and $\mathbf{Y}_{k,v}$ also includes $\{Y_{r,k,v}, Y_{I,k,v}\}$. The terms A_k in (2.16) is equal for all transitions at the same time index, through the show of simulation result, it can be neglected. Substituting (3.6), (3.7), and (3.8) into (3.3), and the LAPP can be further expressed as:

$$\begin{cases} \Lambda_0(\hat{u}_k) = \log \sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=0}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)} = L_{a0}(u_k) + L_{c0}(u_k) + L_{ex0}(u_k) \\ \Lambda_1(\hat{u}_k) = \log \sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=1}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)} = L_{a1}(u_k) + L_{c1}(u_k) + L_{ex1}(u_k) \\ \Lambda_2(\hat{u}_k) = \log \sum\limits_{\substack{(S_{k-1},S_k)\\ u_k=2}} e^{\alpha_{k-1}(S_{k-1})} \bullet e^{\gamma_k(S_{k-1},S_k)} \bullet e^{\beta_k(S_k)} = L_{a2}(u_k) + L_{c2}(u_k) + L_{ex2}(u_k) \end{cases} \quad (3.9)$$

The term $L_a(u_k)$ is called a priori information, and the term $L_{ex}(u_k)$ is called extrinsic information. Since the extrinsic information is a function of the redundant information that introduces by the RSC encoder, it is independent on systematic input and a priori information from LAPP. In general, extrinsic information has the same sign as u_k. Therefore, it is helpful to estimate the priori probabilities of code sequence for next component decoder.

3.2.2: The Max-Log-MAP Algorithm

For simplifying the implementation complexity of MAP decoders, an approximation of MAP algorithm termed Max-Log-MAP algorithm was derived by using equation (2.19), **max** function. The equation (3.9) can be further simplified as:

$$\begin{cases} \Lambda_0(\hat{u}_k) \approx \max\limits_{\substack{(S_{k-1},S_k)\\ u_k=0}} (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k) + \beta_k(S_k)) \\ \Lambda_1(\hat{u}_k) \approx \max\limits_{\substack{(S_{k-1},S_k)\\ u_k=1}} (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k) + \beta_k(S_k)) \\ \Lambda_2(\hat{u}_k) \approx \max\limits_{\substack{(S_{k-1},S_k)\\ u_k=2}} (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1},S_k) + \beta_k(S_k)) \end{cases} \quad (3.10)$$

Similarly, the forward recursive and backward recursive metrics (3.6) (3.7) can individually be expressed as

$$\alpha_k(S_k) \approx \max_{S_{k-1}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k)) \tag{3.11}$$

and

$$\beta_k(S_k) \approx \max_{S_{k+1}}(\beta_{k+1}(S_{k+1}) + \gamma_k(S_k, S_{k+1})) \tag{3.12}$$

3.2.3: The Log-MAP Algorithm

Through the use of the Jacobian algorithm shown in (2.23) and (2.24), the Max-Log-MAP algorithm can be modified and avoided the performance loss which caused by approximation **max** function. The modified algorithm, called Log-MAP algorithm, is equivalent to the MAP algorithm. Now we can substitute (3.11) and (3.12) into (2.23), the forward and backward recursive metrics can be represented as

$$\alpha_k(S_k) = \max_{S_{k-1}}{}^*(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k)) \tag{3.13}$$

and

$$\beta_k(S_k) = \max_{S_{k+1}}{}^*(\beta_{k+1}(S_{k+1}) + \gamma_k(S_k, S_{k+1})) \tag{3.14}$$

where the $\max^*(\cdot)$ operation is defined by (2.27). Then, the (3.10) can be expressed as

$$\begin{cases} \Lambda_0(\hat{u}_k) = \mathbf{max}^*_{\substack{(S_{k-1},S_k)\\ u_k=0}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k) + \beta_k(S_k)) \\ \Lambda_1(\hat{u}_k) = \mathbf{max}^*_{\substack{(S_{k-1},S_k)\\ u_k=1}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k) + \beta_k(S_k)) \\ \Lambda_2(\hat{u}_k) = \mathbf{max}^*_{\substack{(S_{k-1},S_k)\\ u_k=2}}(\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k) + \beta_k(S_k)) \end{cases} \tag{3.15}$$

3.2.4: SNR Sensitivity of Max-Log-MAP and Log-MAP Algorithm

In section 2.2.3, it has been discussed that, for binary turbo code, both MAP and log-MAP algorithm requires SNR estimation and Max-Log-MAP algorithm is SNR independent if a priori information is initialized with a reasonable value. For ternary turbo code, the MAP and Log-MAP algorithms also need SNR estimation from their deductions. The effect of SNR sensitivity on BER performance of ternary turbo code can is observed through simulations shown in Fig.3.6 and Fig.3.7. Both Figs. show the BER versus SNR estimation offset based on ternary turbo code, MAP algorithm, 10 iterations, and AWGN channel. The simulation result points out that the underestimation of SNR is more harmful than overestimation, and the sensitivity of ternary turbo code with 3GPP2 interleaver is lightly less than ternary turbo with S-Random interleaver. Note that in both Figs., the minimum BER is not achieved for 0dB offset-SNR, and this effect occurs only for the short block sizes under consideration, please refer to [14] for more detail discuss.

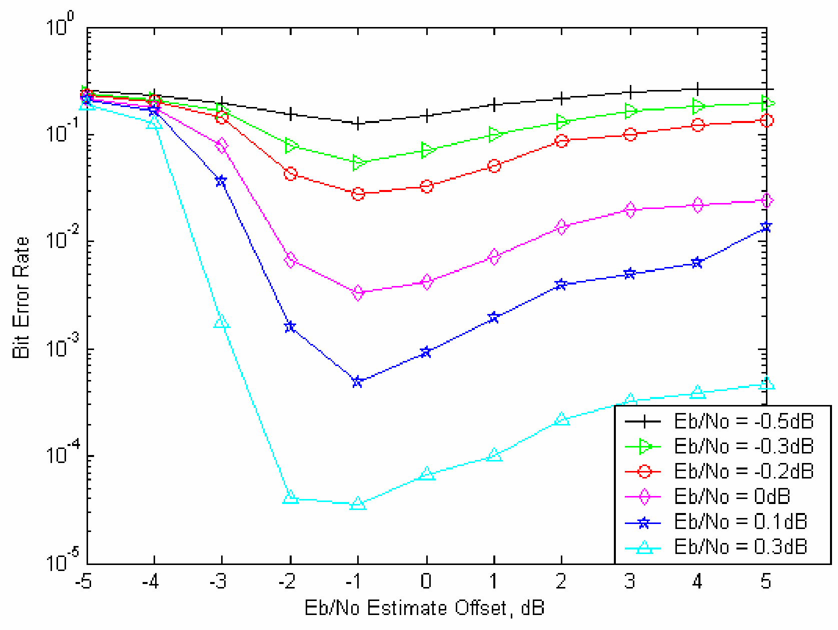

Fig. 3.6 Sensitivity of BER performance of ternary turbo code with S-Random interleaver (length 1200)

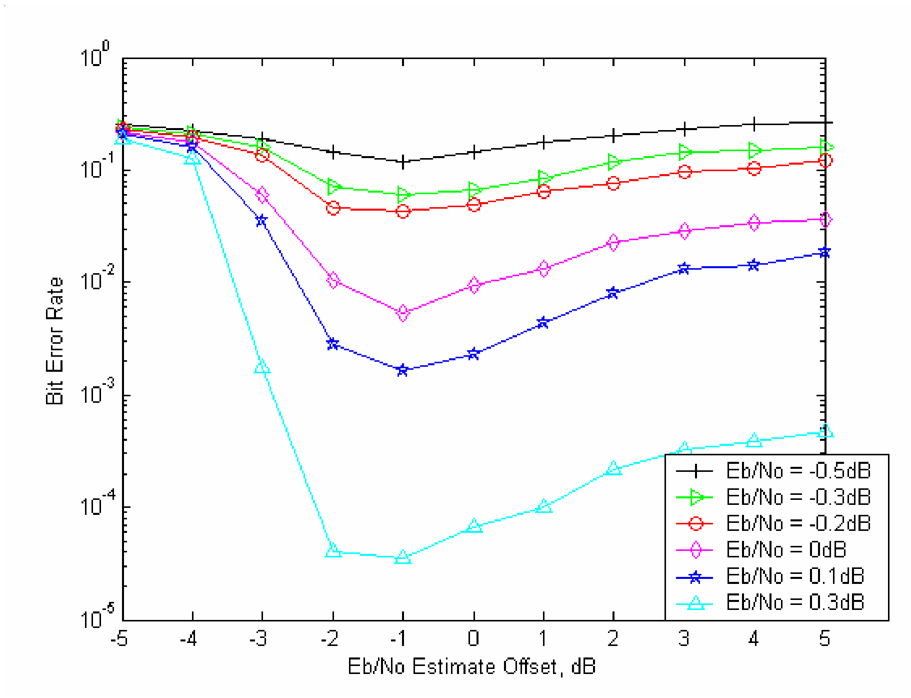

Fig. 3.7 Sensitivity of BER performance of ternary turbo code with 3GPP2 interleaver (length 1146)

For Max-Log-MAP algorithm in ternary turbo code, the method of paper [14] can be extend to show the fact that it does not require knowledge of the SNR, if $L_a(u_k)$, a priori information, is initialized to zero for each bit.

3.3: Sliding Window Method for Ternary Turbo Decoding

Since the decoding algorithm of ternary turbo code is very similar to the algorithm of binary turbo code, the sliding window decoding method can directly be employed to ternary turbo decoding. The more detail discuss, please refer to section 2.3.

Chapter4 Embedded Prior Information Ternary Turbo-Coded Modulation

Coded modulation is to combine coding and modulation technique. It achieves significant coding gain without bandwidth expansion the trellis coded modulation (TCM) is a form of coded modulation based on convolutional codes, and block coded modulation (BCM) is based on block codes. In this chapter, we'll focus on turbo code which is one of trellis codes, so only TCM is introduced.

4.1: Trellis-Coded Modulation

How to construct the bits-to-symbols mapping is main fundamental questions on combining coding and modulation. Now, consider that if soft ML-decoding is applied, the error-event probability will approach asymptotically at high SNR, the lower bound [16] is

$$\begin{aligned} P_r(e) &\approx N(d_{free}) \bullet Q\left(\sqrt{\frac{d_{free}^2}{2N_0}}\right) \\ &\approx N(d_{free}) e^{-\frac{d_{free}^2}{4N_0}} \end{aligned} \tag{4.1}$$

here N(d_{free}) denote the number of nearest neighbors of transmitted coded sequence, and d_{free}^2 is the minimum Euclidean free distance between transmitted coded sequence and any other coded sequence. Thus, it is very obvious that the term of d_{free}^2 dominates the performance at high SNR, so the goal of bits-to-symbols mapping is to maximum the minimum Euclidean free distance, but how to achieve this goal?

Through set partition applied a natural mapping of bits to signals, the author, Ungerboeck, of [16] describes the concrete method to maximum the minimum Euclidean free distance of a coded modulation system combined binary convolutional codes and modulation. This coded modulation is well-known trellis coded modulation (TCM). Comparison with un-coded system, TCM combined ordinary rate $R_c = k/(k+1)$ binary convolutional codes with an M-ary signal constellation ($M = 2^{k+1} > 2$) achieves coding gain without increasing transmitted symbol rate. Note that due to TCM scheme uses signal set expansion rather than additional transmitted symbols to accommodate the redundant bits, performance comparisons must be made with un-coded modulation systems that have same spectral efficiency. The design rule of TCM can be summarized as follows:

I. All subsets should occur in the trellis with equal frequency.

II. State transitions that begin and end in the same state should be assigned subsets separated by the largest Euclidean distance.

III. Parallel transitions are assigned signal points separated by the largest Euclidean distance.

4.2: Embedded Prior Information Technique

The main idea of TCM is similar to extend signal constellation to join redundant parity bits. For convenience, parity bits mentioned in this chapter express redundant bits which is introduced by coding and obtained from constellation extension. This concept can apply to a coded system, such as a rate 1/2 binary convolutional code with binary phase shift keying (BPSK) modulation, an extra coding gain can be obtained in such a way that by expanding the signal constellation which from 2-ary extends to m-ary to add extra parity bits into original coded system, without increasing the required bandwidth. The idea is experimented on a concatenated coding system with ternary phase shift keying (3PSK) modulation. The simulation result shows the concatenated coding built with an inner rate 1/2 ternary convolutional code and an outer (128, 64, t=6) BCH code has better performance gain than original binary convolutional code with BPSK modulation. More detail discuss about these, please refer to Appendix A. However, the additional coding gain based on expanding constellation to concatenate an outer code to original coded system is not always able to achieve, and an opposite instance is described in Appendix B.

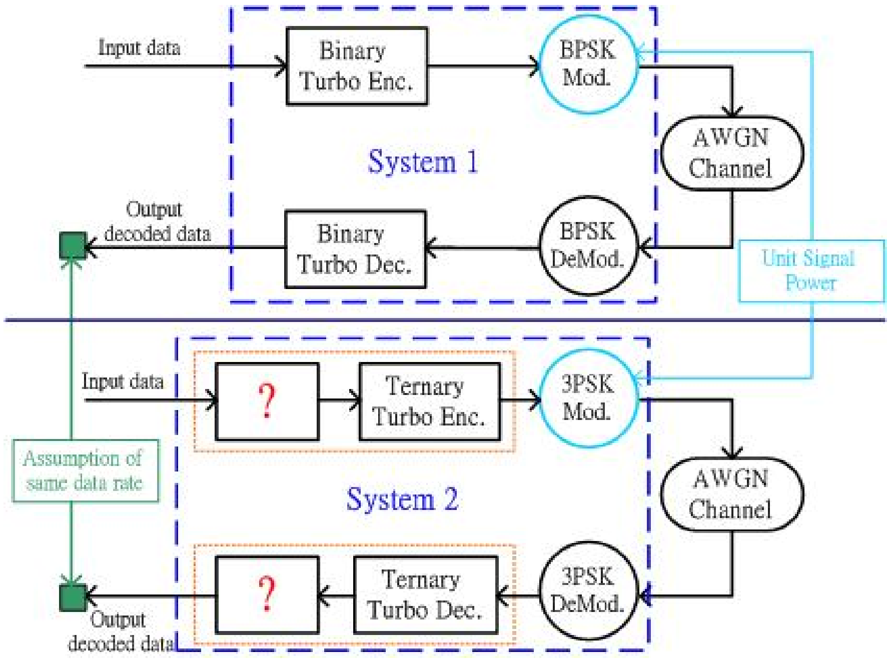

Fig. 4.1 Structure diagram of two different coding systems

Now, consider two coding systems show in Fig. 4.1. Based on following conditions: 1. Constant data rate, 2. Unit transmitted signal power, 3. Constant spectral efficiency (transmitted symbols/unit time slot), and 4. Constant block size. System 1 is a rate 1/3 state 8 binary turbo code with BPSK modulation. System 2 extends BPSK modulation to 3PSK and utilizes a rate 1/3 state 9 ternary turbo code for coding. Note that the question marks in System 2 represent how to utilize redundant information which can be used since constellation expansion. Whereas joining additional parity bits introduced by coding into a complex coding system, such as turbo code, increase not only hardware complexity but also decoding latency, how to “efficiently” use redundant information to achieve remarkable coding gain is the key design. In this section, we propose a method which embedded known digits into ternary turbo code to improve the performance. This method was kind of like [20].

4.2.1: Proposed Structure

In general, two GF(3) symbols can carry about 3-bit information, that is, 1200 bits can be represented only 800 GF(3) symbols. Supposed that the System 1 want to transmit 1200 binary symbols, based on same channel efficiency condition, System 2 also transmits 1200 GF(3) symbols. As former mention, and based on same data rate condition, System 2 has additional 400 GF(3) symbols can arbitrary use. How to maximum minimum free distance of System 2 is first considered. Due to the minimum free distance depends on minimum low weight coded sequences which are generated from low weight input sequences, therefore, how to maximize low weight input sequences is design key point. A simple method to maximize low weight input sequences is to insert known GF(3) digits, into ternary turbo code. Here, two questions will be considered as follow:

I. Are known digits constant or variable?

II. Which positions of information sequence are known digits inserted into?

For question one, the constant known digits are selected and by simulation that inserting constant known GF(3) digit of 2 into ternary turbo coding has the best performance, comparison to constant known GF(3) digit of 0 and 1. For question two, inserting known digits uniformly into information sequence is considered. The idea of uniform distribution known digits originates in S-Random interleaver which is able to break up burst error.

The structure of System 2 inserted known GF(3) digits is illustrated in Fig. 4.2. Here, "π" represents the interleaver. This system, in short, is called Embedded Prior

Information-I Ternary Turbo Code (EPI-I TTC) and uses constant and known GF(3) digits rather than parity bits. The inserting locations of known GF(3) digits are fixed and one known GF(3) digit is inserted after two GF(3) information digits. Note that known digit is not to transmit, since it is constant and has already known by both encoder and decoder.

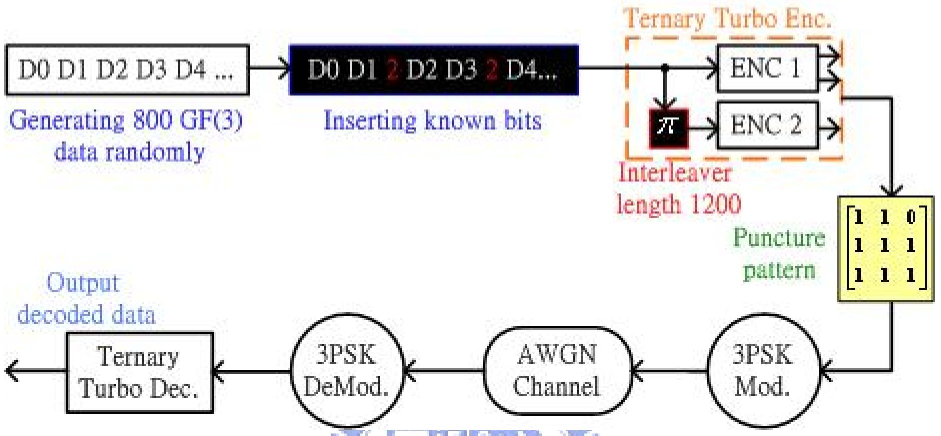

Fig. 4.2 Structure of EPI-I TTC

In Fig. 4.2, it can be observed that positions of known digits that input into ENC 1 and ENC 2 are different, since interleaver before ENC 2 reorders the input sequence. However, the non-uniform location distribution of known digits is not one as expected. Fig. 4.3 shows a modified system, and it called Embedded Prior Information-II Ternary Turbo Code (EPI-II TTC). The EPI-II TTC has not only uniform known digit location distribution but also smaller interleaver size. Note the interleaver size in EPI-II TTC is 800.

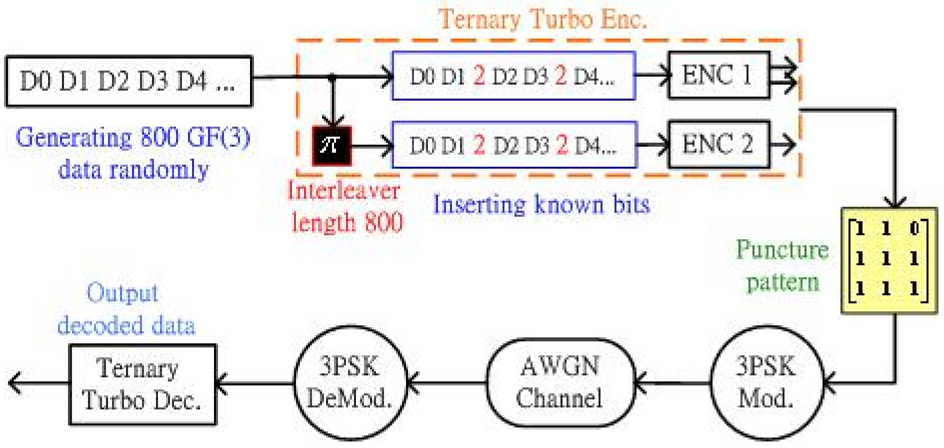

Fig. 4.3 Structure of modified EPI-II TTC

If the block size of binary turbo code in System 1 is 1200, based on constant block size condition, the block lengths of EPI-I TTC and EPI-II TTC are also 1200 included 800 GF(3) information digits and 400 known Gf(3) digits. In general, the block size of turbo code is as same as interleaver size, but in Fig. 4.3 they are different, since known GF(3) digits is inserted after intleaver. As mention before, the ternary turbo code have only 800 GF(3) information digits and, thus, its interleaver size is the same. The advantage of using embedded known digits to fill in parity bits is to remarkable enlarge low weight coded sequence, that is not only improving BER performance but also increasing no additional complexity of hardware and computation, comparison to original ternary turbo code.

4.2.2: Performance Effects

In this section we discuss the effect of embedded known digits into ternary turbo code and binary turbo code. The influence of embedded known digits can be discussed from two viewpoints.

I. Breaking up burst errors.

II. Maximizing minimum low weight coded sequences.

Consider an extreme case shown in Fig. 4.4, there are 15 burst errors which is not able to decode correctly by ternary turbo code. But in EPI-II TTC, since known digits break a long burst-error sequence into some small parts, and the strong reliable extrinsic values of known digits revise error path close to correct one, through iteration by iteration, the burst error sequence lastly can be correct decoded.

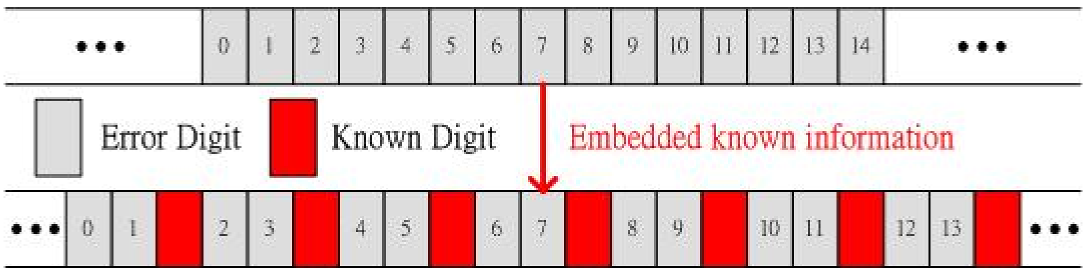

Fig. 4.4 Sketch diagram of breaking off burst error by inserting known digits

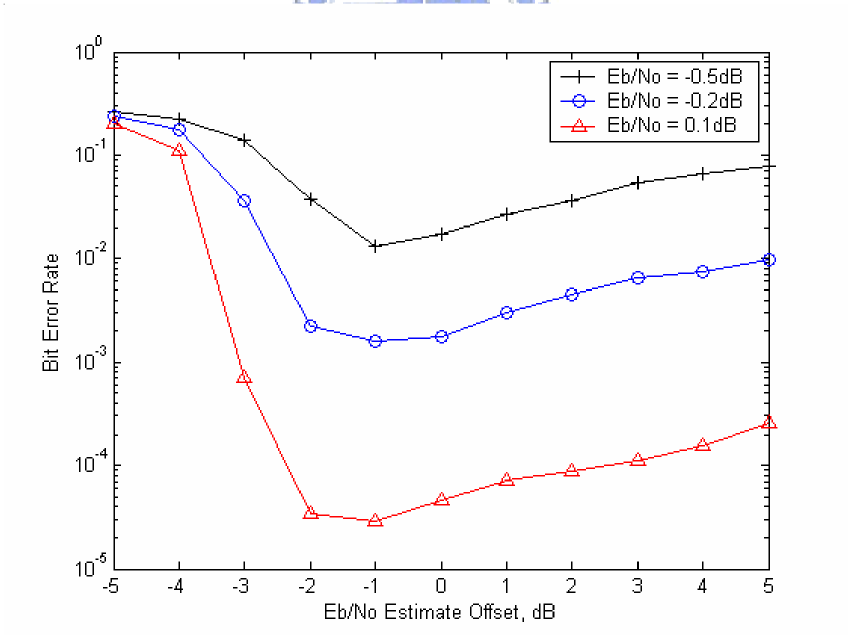

Fig. 4.5 Sensitivity of BER performance of EPI-II TTC with S-Random interleaver (length 800)

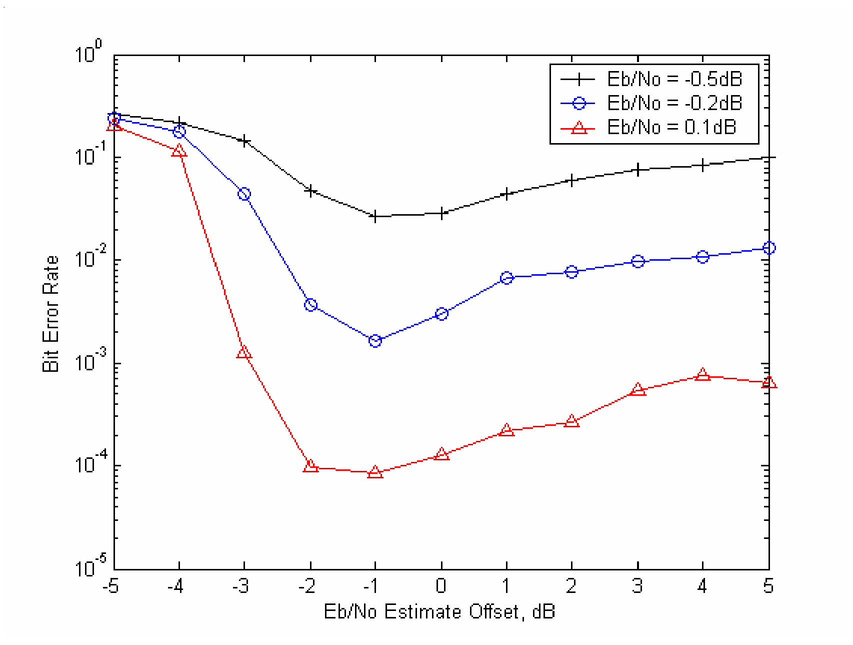

Fig. 4.6 Sensitivity of BER performance of ternary turbo code with 3GPP2 interleaver (length 1146)

Let **v** = (v_0, v_1, … , v_{n-1}) be a Gf(3) n-tuple. The Hamming weight of **v**, denoted by *w*(**v**), defined as the number of nonzero components of **v**. For example, the Hamming weight of **v** = (1 2 0 1 0 2) is 4, Let **v** and **w** be two n-tuples. The Hamming distance between **v** and **w**, denoted d(**v**, **w**), is defined as the number of places where they differ. For example, the Hamming distance between v = (1 2 1 0 0 2) and w = (2 1 1 0 0 0) is 3; they differ in the zeroth, first, and, sixth places. Now, consider a rate 1/2 ternary convolutional code with a generator matrix, $[1 \ \frac{1+D+2D^2}{2+D+D^2}]$, its minimum weight input is (2,1,1,0,…) and Hamming weight is 3. This value is very small. If inserted constant known digits into input sequence in such way that two input data insert a known digit of 2. The effect of inserting known digits for ternary convolutaional code is equal to multiply a factor, $\frac{2+D^2+2D^4}{1+D^2+2D^4+2D^6}$, into generator matrix. Then, $[\frac{2+D^2+2D^4}{1+D^2+2D^4+2D^6} \quad \frac{2+2D+2D^2+D^3+D^4+2D^5+D^6}{2+D+D^3+D^4+2D^5+2D^7}]$ is the new generator matrix, so the low weight input can be enlarged well.

Due to the decoded algorithm of EPI-II TTC is MAP algorithm, its SNR sensitivity showed in Fig. 4.5 and Fig. 4.6 needs to be considered. Fig. 4.5 and Fig. 4.6 show the BER versus SNR estimation offset based on EPI-II TTC, MAP algorithm, 10 iterations, and AWGN channel, but different interleavers. The simulation result points out that the underestimation of SNR is more harmful than overestimation, and the sensitivity in 3GPP2 interleaver is lightly less than S-Random interleaver.

However, the SNR sensitivity of EPI-II TTC and ternary turbo code seems to be the same for a glance. But through the charts shown in Fig. 4.7 and Fig. 4.8, their diversity can be noted. In charts, the BER of real E_b/N_0 is normalized to 1, thus, the level of SNR sensitivity corresponded to SNR offset can be observed. The chart points out that the SNR sensitivity of EPI-II TTC is less than ternary turbo code. Fig. 4.7 shows SNR sensitivity comparisons between EPI-II TTC and ternary turbo code with S-Random interleavers at E_b/N_0 of 0.1dB and Fig. 4.8 shows SNR sensitivity comparisons with 3GPP2 interleaver.

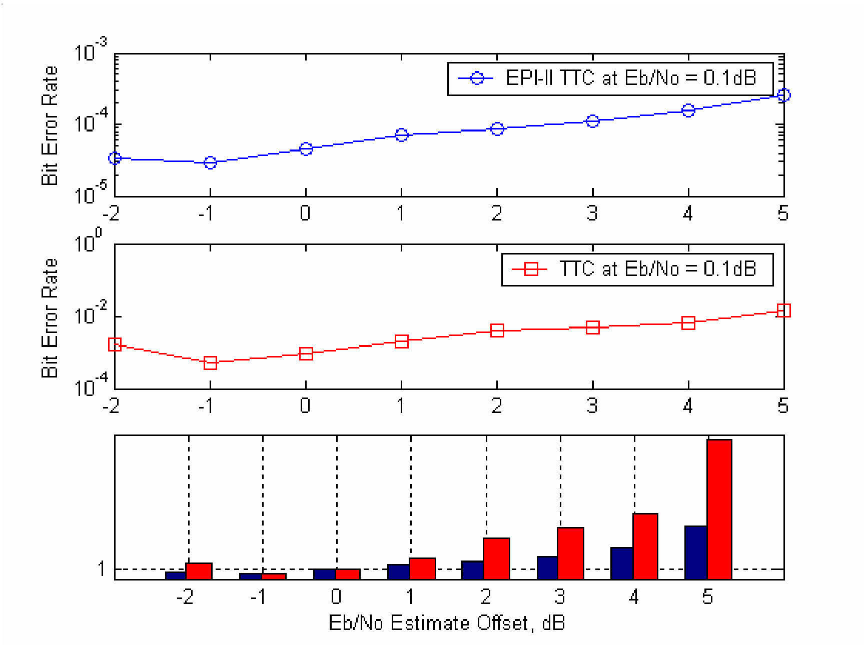

Fig. 4.7 SNR sensitivity comparisons between EPI-II TTC with S-Random interleavers.

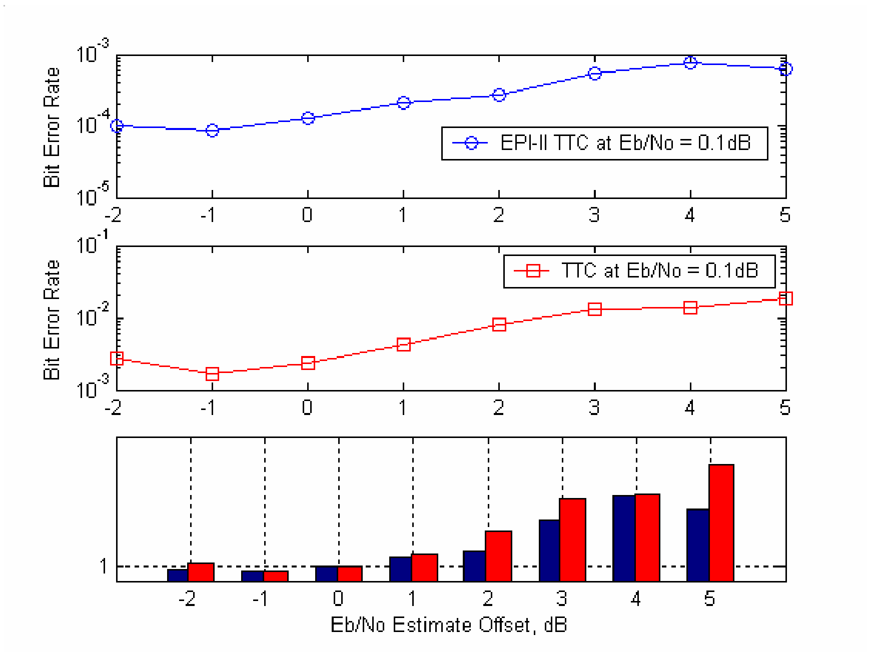

Fig. 4.8 SNR sensitivity comparisons between EPI-II TTC and normal ternary turbo code in 3GPP2 interleaver.

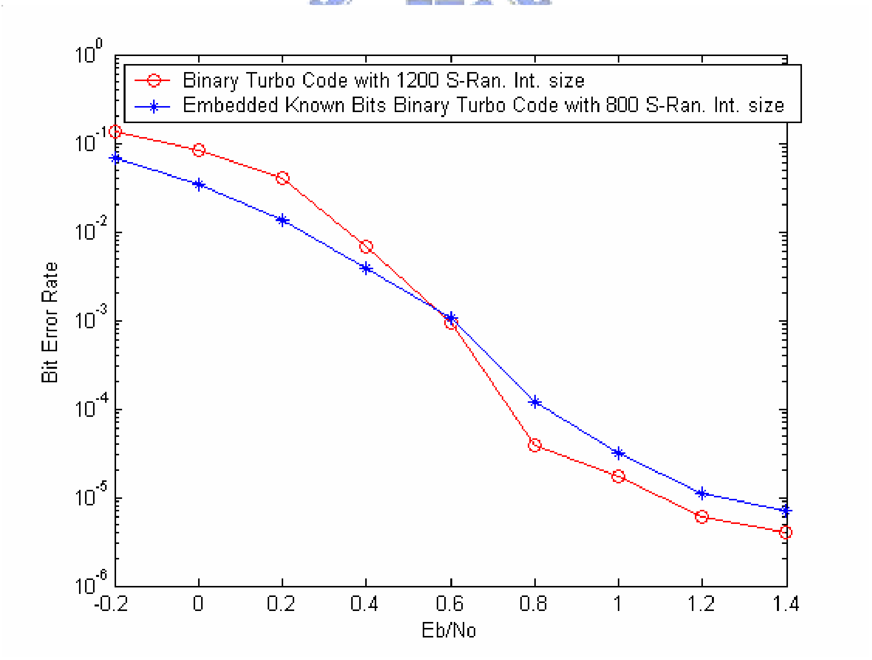

Fig. 4.9 Performance comparison between embedded known digits binary turbo code and binary turbo code with S-Random, S=10 (10 iterations, state 8)

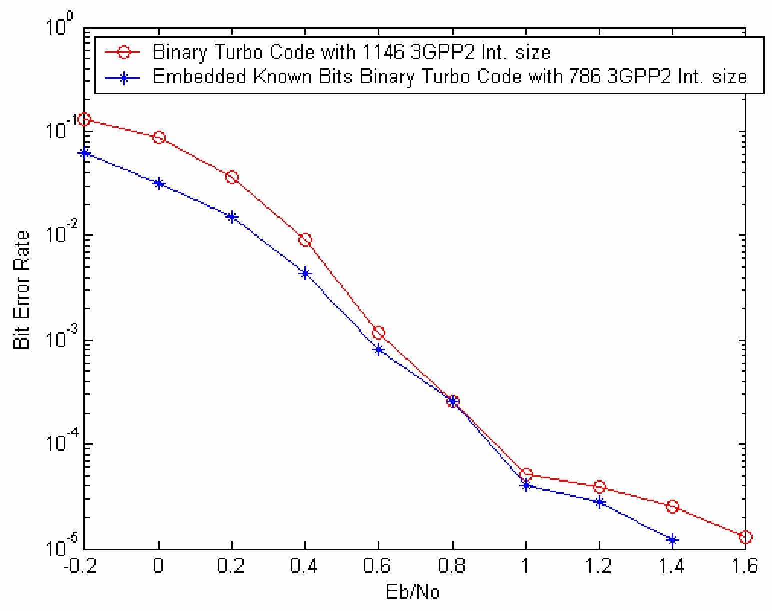

Fig. 4.10 Performance comparison between embedded known digits binary turbo code and binary turbo code with 3GPP2 interleaver (10 iterations, state 8)

If the constant data rate condition does not considered, can binary turbo code improve its performance in terms E_b/N_0 by using embedded known bits technique? The solution is answered by simulations. Fig. 4.9 and Fig. 4.10 display BER performances of a rate 1/3 state 8 embedded known bits binary turbo code and a rate 1/3 state 8 binary turbo code. Both Figs. are based on MAP algorithm, 10 iterations, BPSK modulation, AWGN channel, and two information data insert a known bit of 1 for embedded known bits binary turbo code. In Fig. 4.9, with S-Random interleaver, binary turbo code has better BER performance than embedded known bits one after E_b/N_0 of 0.6dB. In Fig. 4.10, with 3GPP2 interleaver, the embedded known bits binary turbo code has better BER performance than binary turbo code from top to bottom. In Fig. 4.11, the different embedded known bit numbers is examined, the simulation result points out that the binary turbo code is always superior after E_b/N_0 of 0.001, than others. However, the less known bits are embedded into binary turbo code; the better performance is achieved. Due to limitation of computing power, the simulation in Fig. 4.11 is done with only 4 iterations. It can be conclusion that the method of embedded known bit is helpful for structured interleaver, but in random interleaver, it has better performance at lower SNR.

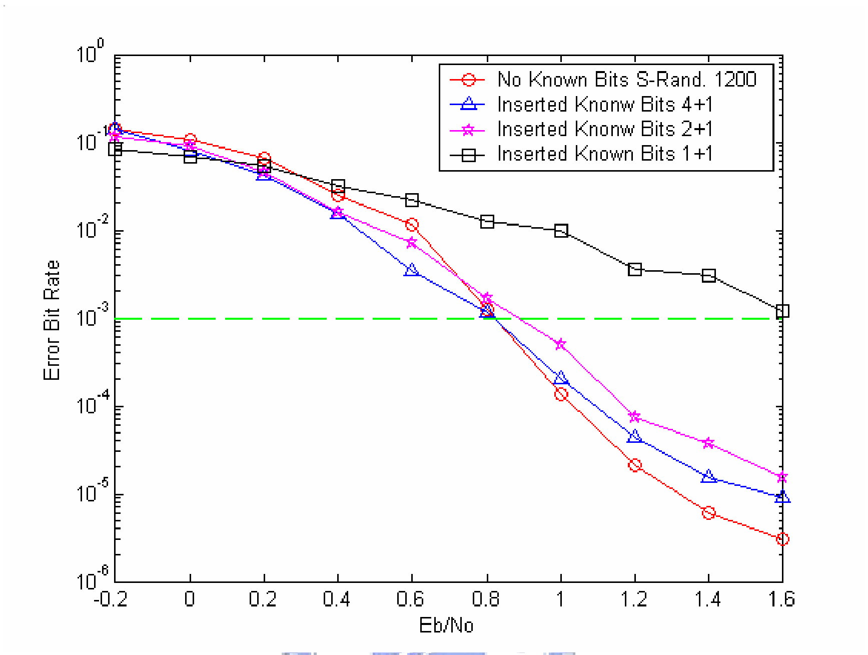

Fig 4.11 Performance comparison under different embedded known bit numbers in binary turbo code (4 iterations, state 8, S-Random interleaver)

Chapter5 Simulation and Implement Results

Look back on Fig. 4.1 it has been expected that System 2 has better performance than System 1, by using embedded known digits technique. And both systems need to satisfy following conditions as: 1. unit transmitted power, 2. constant data rate, 3. constant channel efficiency. In this chapter, performance simulations in terms of E_b/N_0 and hardware complexities of two systems will be compared. Section 5.1 discusses simulation results between Embedded Prior Information-II Ternary Turbo Code (EPI-II TTC), Embedded Prior Information-I Ternary Turbo Code (EPI-I TTC), ternary turbo code (TTC), and binary turbo code (BTC). And hardware complexity is describes in section 5.2.

5.1: The Comparison of Simulation Performance

Before entering this section, some environment specifications in our simulation have been defined.

Table 5.1 Environment specification:

	Binary Turbo Code	**Ternary Turbo Code**
Modulation	BPSK	3PSK
Transmitted Channel	AWGN	AWGN
Numbers of Iteration	10	10
Decoding Algorithm	Log-MAP	Log-MAP
Generator Matrix	$[1\ \frac{1+D^2+D^3}{1+D+D^2+D^3}]$	$[1\ \frac{1+D+2D^2}{2+D+D^2}]$
Numbers of State	8	9

Here, ternary turbo code includes EPI-I TTC, EPI-II TTC, and TTC. And the factor, S, of S-Random is set to 10. However, the coding gain on BER of 10^{-5} will be focus, since the BER of 10^{-5} is the basic demand in most systems.

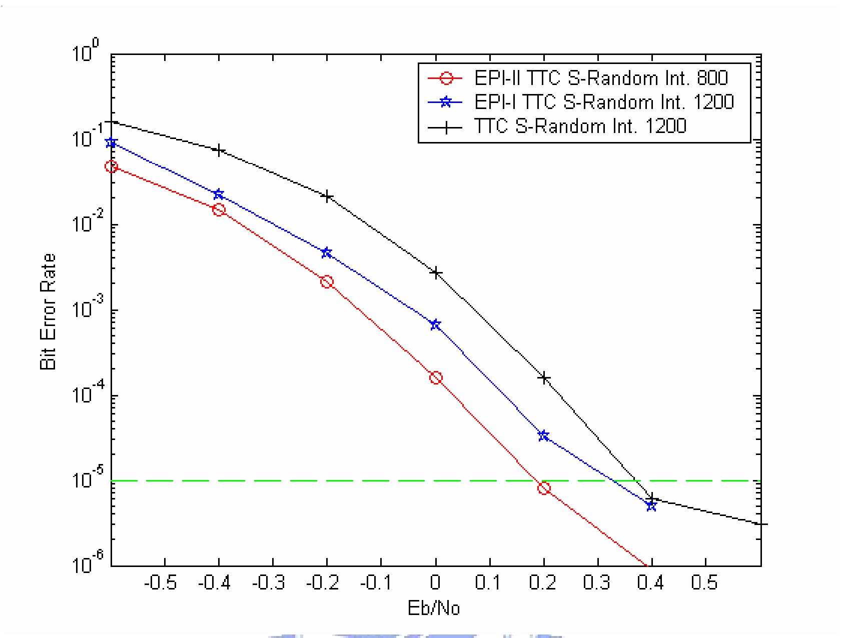

Fig. 5.1 Performance comparisons of ternary turbo codes with S-Random Interleaver

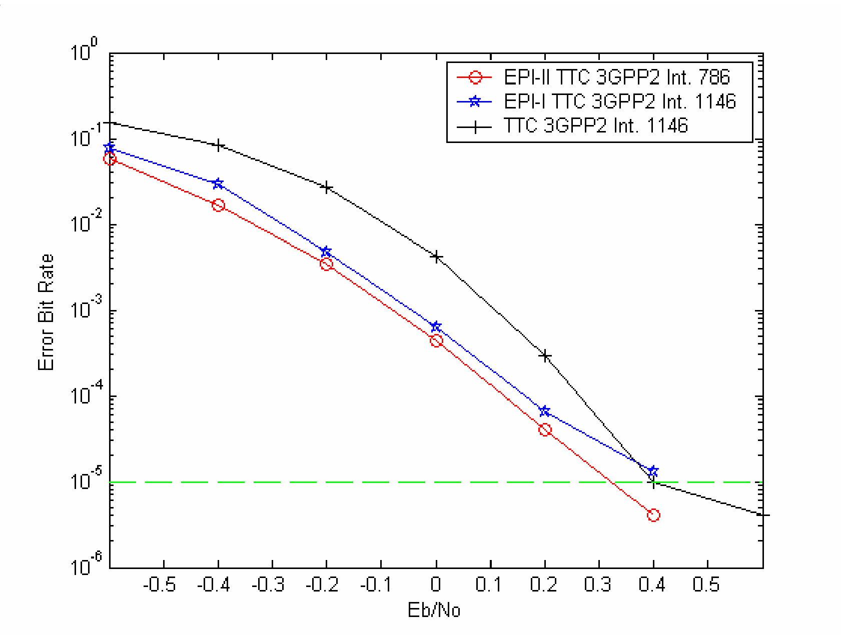

Fig. 5.2 Performance comparisons of ternary turbo codes with 3GPP2 interleaver

Fig. 5.1 and Fig. 5.2 compare performances of ternary turbo code included EPI-II TTC, EPI-I TTC, and TTC on S-Random interleaver and 3GPP2 interleaver, separately. From both simulations, it can be observed that known digit is able to improve BER performance, and EPI-II TTC is better than EPI-I TTC. Note that the data rate of TTC is larger than EPI-II TTC and EPI-I TTC about 1.5 times, but the interleaver size of EPI-II TTC is smaller than EPI-I TTC and TTC about 1/3 since the interleaver size in EPI-II TTC is 800 and others are 1200 if the block size is 1200. For constant block size of 1200, Fig. 5.3 displays the performance comparison of turbo codes included EPI-II TTC, TTC, and BTC with S-Random interleaver. The simulation result points out that the EPI-II TTC has performance gain about 0.2dB than TTC and about 1dB than BTC. Fig. 5.4, with 3GPP2 interleaver, shows that the EPI-II TTC has performance gain about 0.5dB than TTC and about 1.2dB than BTC.

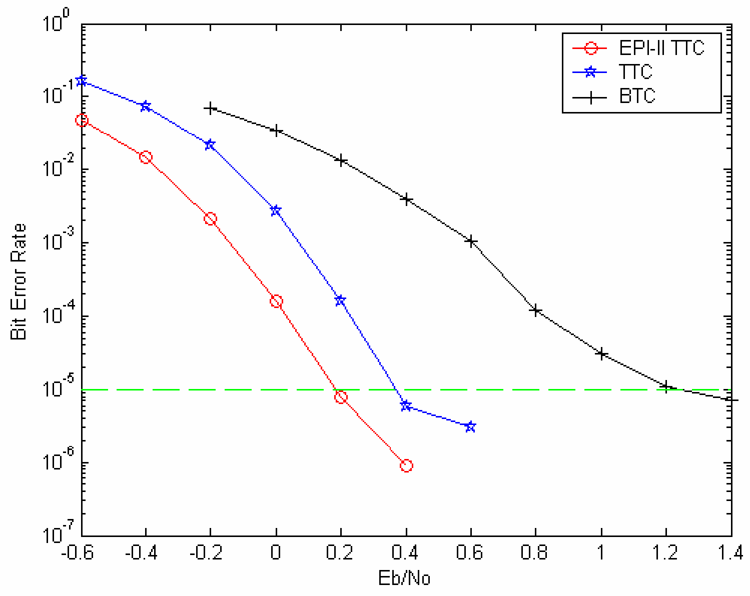

Fig. 5.3 Performance comparisons of turbo codes with S-Random Interleaver (block length=1200)

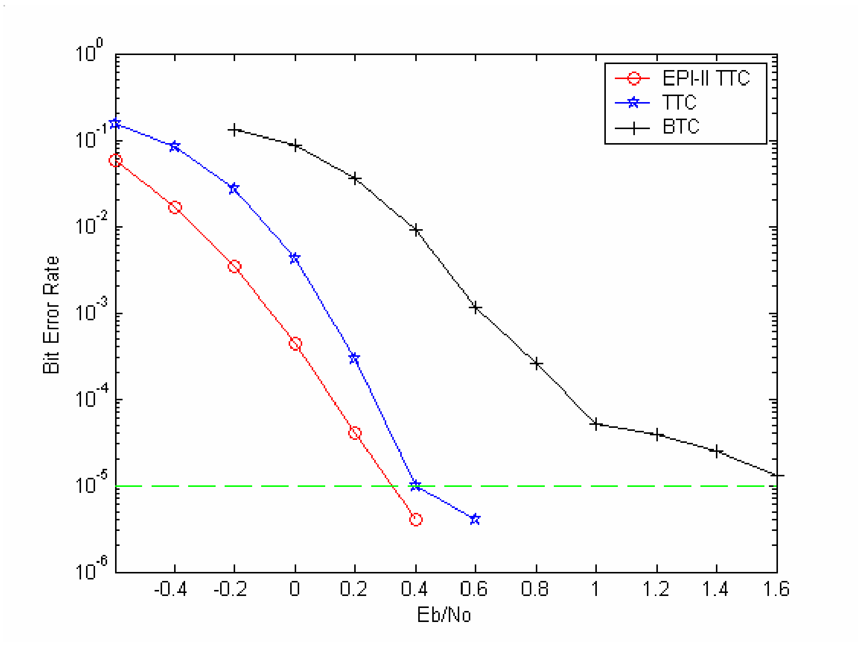

Fig. 5.4 Performance comparisons of turbo codes with 3GPP2 Interleaver (block length=1146)

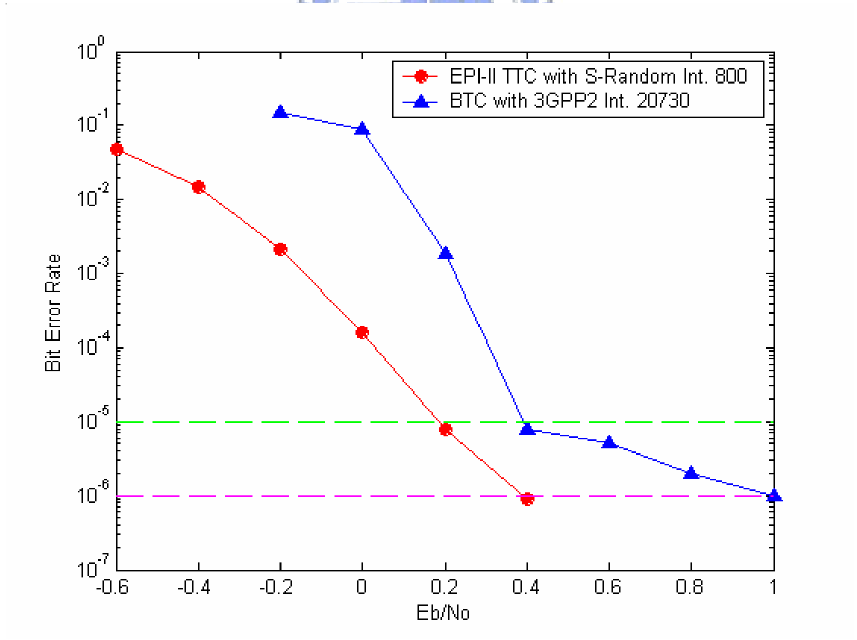

Fig. 5.5 Performance Comparison between EPI-II TTC and Binary turbo code with 3GPP2 interleaver length of 20730

Former discuss focus on that in same block length, EPI-II TTC is able to improve how many performance gains if comparison with TTC and BTC. But the key point, which we really care about, is that the performance comparison between EPI-II TTC with short interleaver and BTC with long interlaver. Fig. 5.5 shows their performance simulation, which EPI-II TTC using length-800 S-Random interleaver and BTC using length-20730 3GPP2 interleaver. The simulation result points out that EPI-II TTC has performance gain than BTC about 0.2dB at BER of 0.00001 and about 0.6dB at BER of 0.000001. However, it is very obvious that the error floor region of BTC occurs after BER of 0.00001, but the error floor region of EPI-II TTC does not appear yet. Finally a performance constellation shown in Fig. 5.6 displays that based on BER of 0.00001, the performance comparison between different turbo codes which employ different interleaver structures and different interleaver lengths. The horizontal axis represents E_b/N_o and the vertical axis represents interleaver length in log-scale.

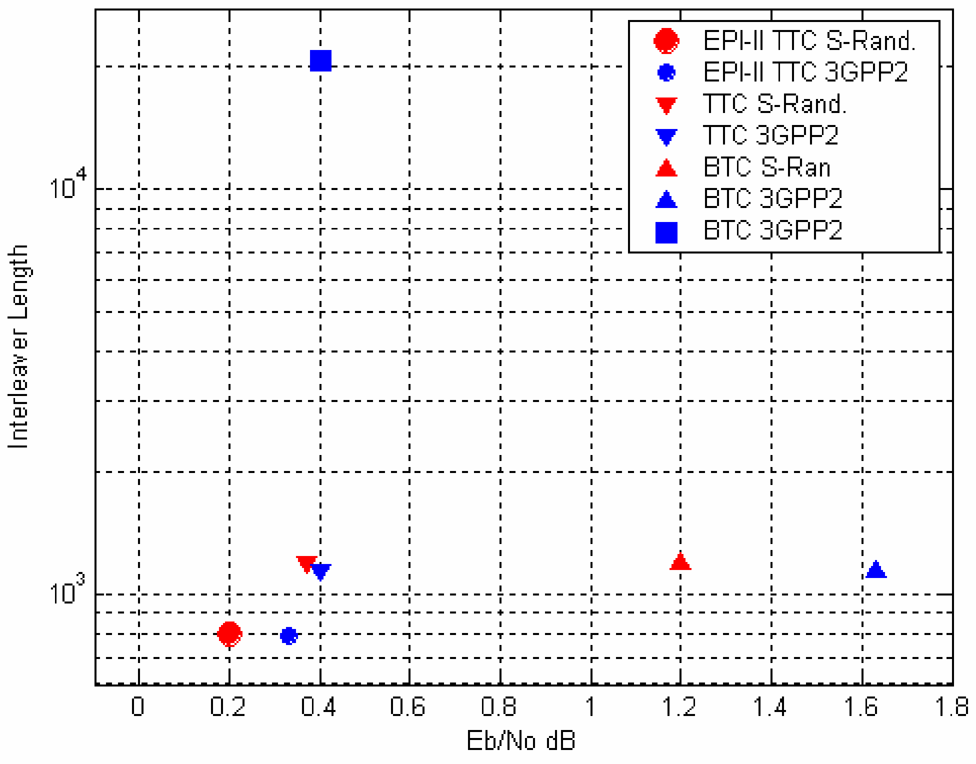

Fig. 5.6 Performance constellation of turbo codes at $E_b/N_0=10^{-5}$

5.2: The Estimation of Hardware Complexity

In previous chapter, all discussions about turbo codes are based on importing floating point information as required soft-input. Nevertheless, the floating point value should be bounded since infinite precision is impossible to be achieved for practical implementation. A trade-off between hardware cost and the performance must be concerned since coding performance may suffer quantization loss due to internal bit-width limitation. In general, hardware complexity of turbo code can be estimation in computing complexity and memory size which is proportional to bit-with. In this section, based on acceptable performance loss, the fixed point analysis and computing complexity is discussed. Note that in this section, only hardware complexities of EPI-II TTC with length-800 S-Random interleaver and BTC with length-20730 3GPP2 interleaver are compared, since there have similar performance gain. And the sliding window method for turbo decoder is assumed. The length of sliding window is set as 5 times of constrain length of component convolutional codes.

5.2.1 Memory Size Analysis

The most area in turbo code implementation is dominated by memories which store receiver information, LLR value, extrinsic value, forward recursive metric, backward recursive metric, and branch transition metric. And overall required size of memory is determined by fixed point analysis. The bit-width of receiver information is determined firstly, since this value influences numbers of bit-width of internal metric and LLR. Fig. 5.7 plots the quantization loss of the bounded receiver information with 3PSK modulation and EPI-II TTC. Fig. 5.8 shows the same simulation but with BPSK and BTC. Due to the BER performance is centered below 0.00001dB, the 3.3 scheme for EPI-II TTC and the 3.2 scheme for BTC are chose.

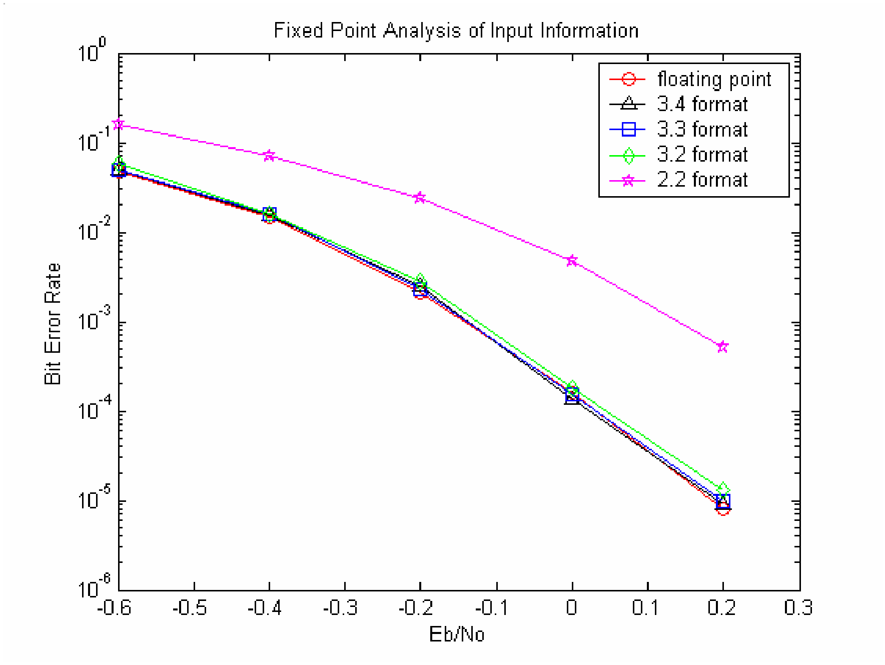

Fig. 5.7 Fixed point simulation result of the input symbols with 3PSK modulation and a rate 1/3 state 9 EPI-II TTC with S-Random interleaver size of 800

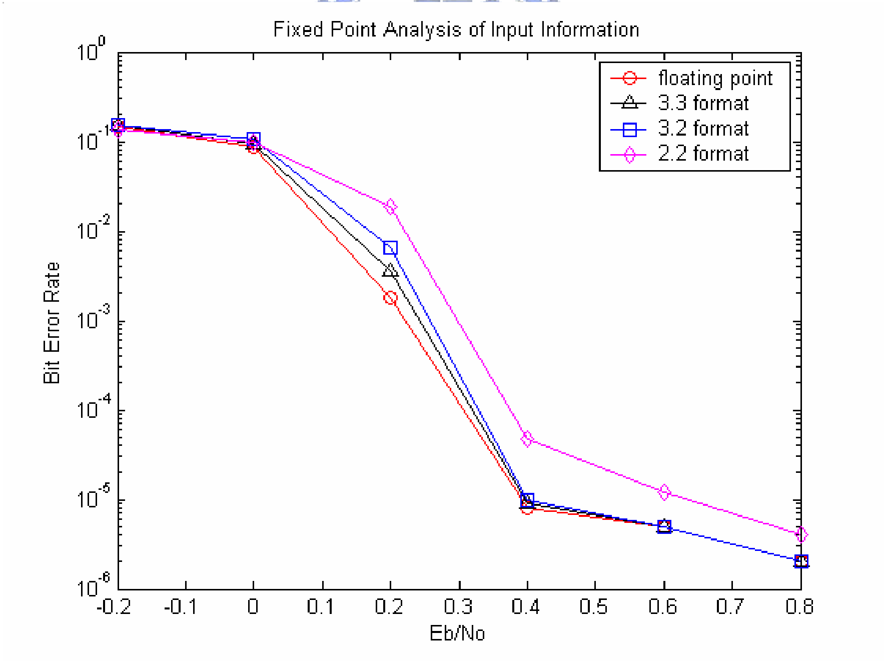

Fig. 5.8 Fixed point simulation result of the input symbols with BPSK modulation and a rate 1/3 state 8 binary turbo code with 3GPP2 interleaver size of 20730

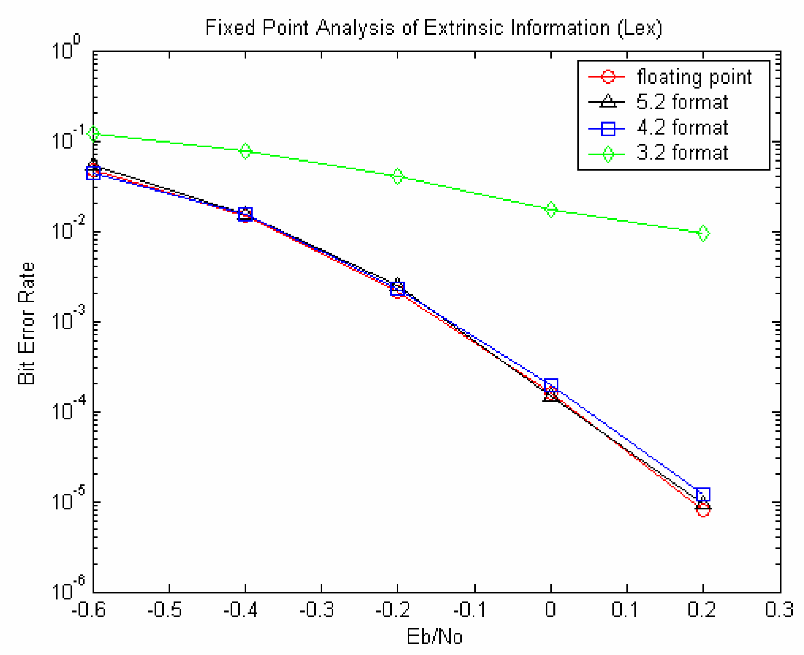

Fig. 5.10 Fixed point simulation result of the extrinsic information (Lex) in a rate 1/3 state 9 EPI-II TTC with S-Random interleaver size of 800

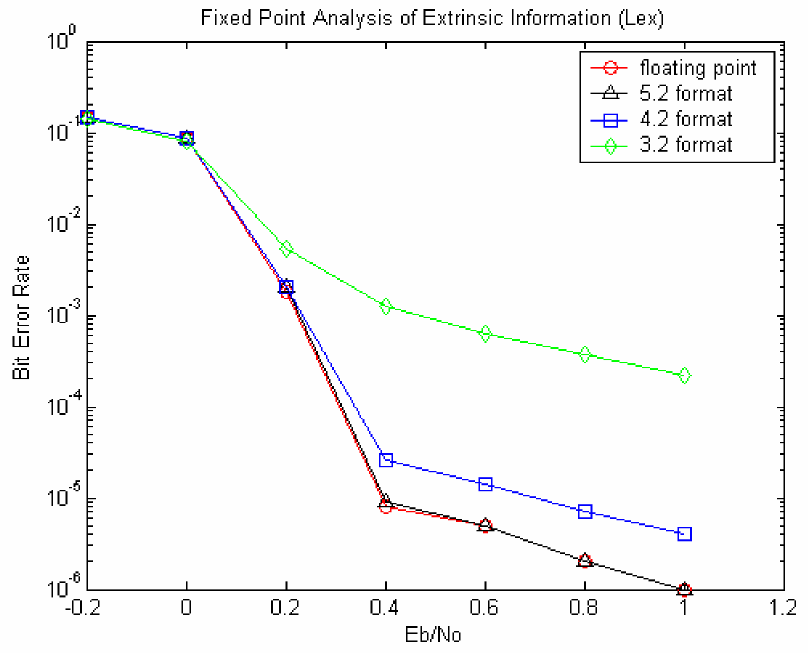

Fig. 5.11 Fixed point simulation result of the extrinsic information (Lex) in a rate 1/3 state 8 binary turbo code with 3GPP2 interleaver size of 20730

Extrinsic data provides reliable information to be a priori information for turbo code to perform the iteration decoding. So quantization of extrinsic information should be done carefully or the effect of iterative decoding will be diminished. A fixed point simulation of the extrinsic information is performed, as shown in Fig. 5.10 and Fig. 5.11. Then, the 4.2 scheme for EPI-II TTC and the 5.2 scheme for BTC are selected. Now, ranges of received information and a priori information that comes from the extrinsic information are determined. According the bounded method of [18] and [19], the bit-width of internal variables, α, β, γ, and LLR value, for Log-MAP algorithm can be derived. The final bit-width of all information that is required by Log-MAP decoding algorithm is listed in Table 5.2. Note that any information exceeding the range that can be expressed is saturated to the nearest value instead to truncating it directly.

Table 5.2 Summary of bit-width decision for EPI-II TTC and binary turbo code

Bit-width	**Rec. Infor.**	$L_a(u_k)$	α	β	γ	$L_{ex}(u_k)$	***LLR***
EPI-II TTC	6 (3.3)	6 (4.2)	8 (6.2)	8 (6.2)	8 (6.2)	6 (4.2)	9 (7.2)
BTC	5 (3.2)	7 (5.2)	8 (6.2)	8 (6.2)	8 (6.2)	7 (5.2)	9 (7.2)

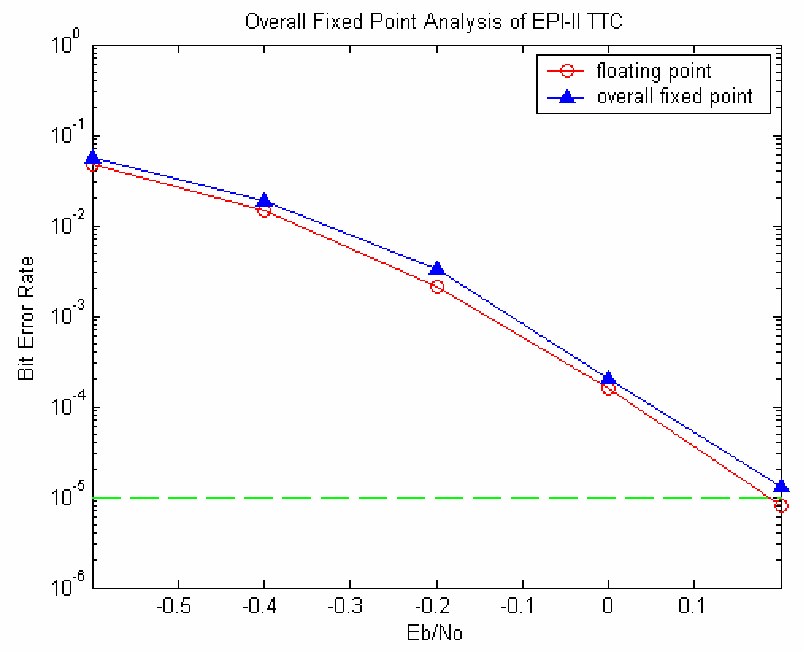

Fig. 5.12 Performance loss of EPI-II TTC comparison between floating point and fixed point scheme

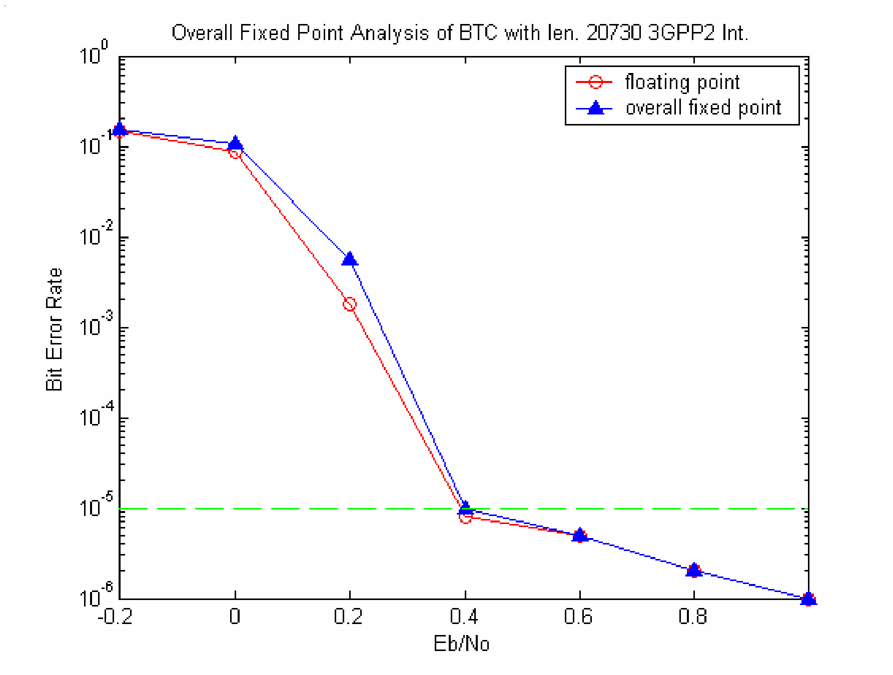

Fig. 5.13 Performance loss of binary turbo code comparison between floating point and fixed point scheme

Table 5.3 Memory sizes comparison between BTC and EPI-II TTC

	BTC with 3GPP2 Int. Len.20730	**EPI-II TTC with S-Rand. Int. Len. 800**
LLR	186570 (20730*9)	21600 (3*800*9)
L_{ex}	145110 (20730*7)	14400 (3*800*6)
X_1	103650 (20730*5)	9600 (2*800*6)
Y_1 & Y_2	207300 (2*20730*5)	28800(2*2*1200*6)
α	1280 (8*20*8)	1080 (9*15*8)
β	1280 (8*20*8)	1080 (9*15*8)
γ	2560(2*8*20*8)	3240 (3*9*15*8)
Table	0	8000 (800*10)
Sum	**647750**	**87800**
Normalize	**1**	**0.14**

Finally, both performances of EPI-II TTC and BTC under complete fixed point condition are simulated, as shown in Fig. 5.12 and Fig. 5.13. For simulation results, their performance losses of fixed point are all less than 0.05dB, comparison with floating point simulations. Now, sizes of memory which required by decoders of EPI-II TTC and BTC can be compared, as listed in Table 5.3. In Table 5.3, the total memory sizes required for EPI-II TTC are 87800 bits and total memory sizes for BTC are 647750, that is, the EPI-II TTC can save about **86%** of memory size than BTC.

5.2.2 Computing Complexity Analysis

Based on Log-MAP decoding algorithm and Sliding method for turbo decoding, the following procedures are executed, for each decoded bit. Note that the trellis diagram of BTC has 8 states and 16 branches and the trellis diagram of EPI-II TTC has 9 states and 27 branches.

- Update forward recursive metrics:

Update forward node metrics based on equation (2.25), and the operation of $\max{}^*(\cdot) = \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|)$ consists of one adder, one 2-input comparator, called comp-2, and one table. Each node for BTC needs 2 adders, 1 comp-2, and 1 table. For EPI-II TTC, each node needs 3 adders, 1 comparator, and 1 table. Assume one 3-input comparator is composed by two comp-2.

- Update backward recursive metrics:

The computation is the same as that in the forward recursive metric update. Since sliding window is able to estimate β_{pre} in advance, 2-suit **max** function operations are needed.

- Update branch metrics

For equation (2.17), each branch for BTC needs 3 adder and 3 multipliers. For equation (3.8), each branch for EPI-II TTC needs 11 adders and 5 multipliers. Here, subtraction operation is instead of adder.

- Calculate LLR information:

For equation (2.28), each LLR for BTC needs 33 adders, 2 tab, and two 8-input comparators. For equation (3.15), each LAPP for EPI-II TTC needs 54 adders, 2 tab, and three 9-input comparators, but one GF(3) digit is represented to 3 LAPP values. Assume one 8-input comparators are consisted of 7 comp-2s and one 9-input comparators are consisted of 9 comp-2s.

The comparison of computing complexity between BTC and EPI-II TTC are listed in Table 5.4.

Table 5.4 Comparisons of computing complexity between BTC and EPI-II TTC

		BTC with 3GPP2 Int. Len.20730	**EPI-II TTC with S-Rand. Int. Len. 800**
LLR		33 add. + 2 tab. + 14 com-2	162 add. + 6 tab. + 72 com-2
α		16 add. + 8 tab. + 8 comp-2	27 add. + 9 tab. + 18 comp-2
β		32 add. + 16 tab. + 16 com-2	54 add. + 18 tab. + 36 com-2
γ		48 add. + 48 mult.	297 add. + 135 mult.
Sum	**Add.**	**129**	**540**
	Mult.	**48**	**135**
	Tab.	**26**	**33**
	Comp-2	**38**	**126**
Normal.	**Add.**	**1**	**4.186**
	Mult.	**1**	**2.8125**
	Tab.	**1**	**1.2692**
	Comp-2	**1**	**3.3158**

Conclusion

In this thesis, the ternary turbo coded modulation using embedded prior information, and in short, called embedded prior information ternary turbo code (EPI-II TTC) is presented. The proposed method not only overcomes the main disadvantage of binary turbo code with long interleaver but also lowers error floor effect.

Comparison with binary turbo code (BTC) with BPSK modulation which employs a length-20730 3GPP2 interleaver to achieve BER of 10^{-5} at E_b/N_0 of 0.4dB, the EPI-II TTC with 3PSK modulation which employs a length 800 S-Random interleaver to achieve the same BER at E_b/N_0 of 0.2dB has 0.2dB performance gain, but at BER of 10^{-6} EPI-II TTC can obtain 0.6dB performance gain than BTC. Since embedded known digits is able to widely enlarge the minimum low weigh coded sequences, the error floor of EPI-II TTC can be reduced. And further, comparison with BTC, the proposed system can save not only memory size about **89%** but also decoding latency about **94%** for the same data rate condition.

Comparison to general ternary turbo code, the SNR sensitivity of MAP decoded algorithm as well as interleaver size can be reduced when embedded known digits technique is employed. However, comparison with binary turbo code, advantages of EPI-II TTC can be concluded as following:

I. Decreasing required memory (saving about 86%)

II. Reducing computing latency (saving about 94%).

III. To lower down the error floor (0.6 dB performance gain at E_b/N_0 of 10^{-6}).

In addition, the overhead which the EPI-II TTC needs to pay is the more computing complexity for Log-MPA algorithm than BTC. And due to ternary turbo code works on GF(3), the additional hardware need to implement that two tables, a table for the transformation of binary bit to ternary digit and another table for ternary turbo encoder, are required to built, but both tables are small. Moreover, comparison to BPSK modulation technique, the 3PSK modulation is more complex.

Appendix A: Scheme I (BCH + Ternary Convolutional Code)

Since comparison with un-coded system, TCM which extended signal constellation to join redundant parity bits have great performance gain. So we want to observe that how many performance gains can be achieved when the idea of TCM is extended into coded system. In Fig. A.1, System 2 is a concatenated coding built with an outer (126, 84, t=6) BCH code and rate-1/2 state-9 ternary convolutional code. System 1 is a rate 1/2 state-128 convolutional code. The generator matrices of binary and ternary convolutional code are $G=[1+D^2+D^5+D^6+D^7,\ 1+D+D^2+D^3+D^4+D^7]$ and $G=[1+D+2D^2,\ 2+D+D^2]$, separately. Both systems have the same conditions, such as: constant data rate, constant channel efficiency, and unit transmitting power. Then the simulation result shown in Fig. A.2 points out that System 2 has about 0.5dB performance gains than System 1. Note that the decoding algorithm of binary or ternary convolutional code is Viterbi algorithm.

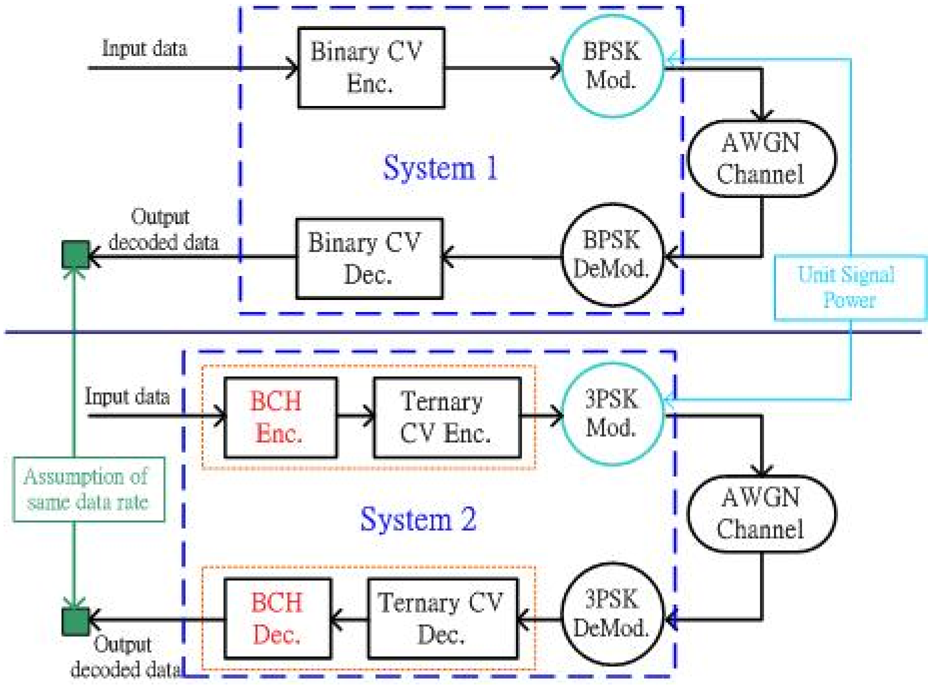

Fig. A.1 Structure diagram of two different coding systems

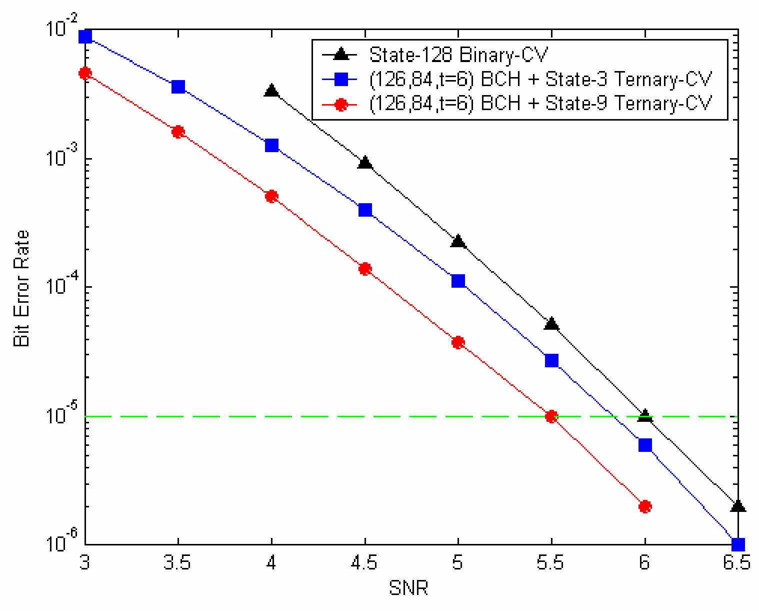

Fig. A.2 Performance comparisons between binary convolutional code and ternary convolutional code with BCH code

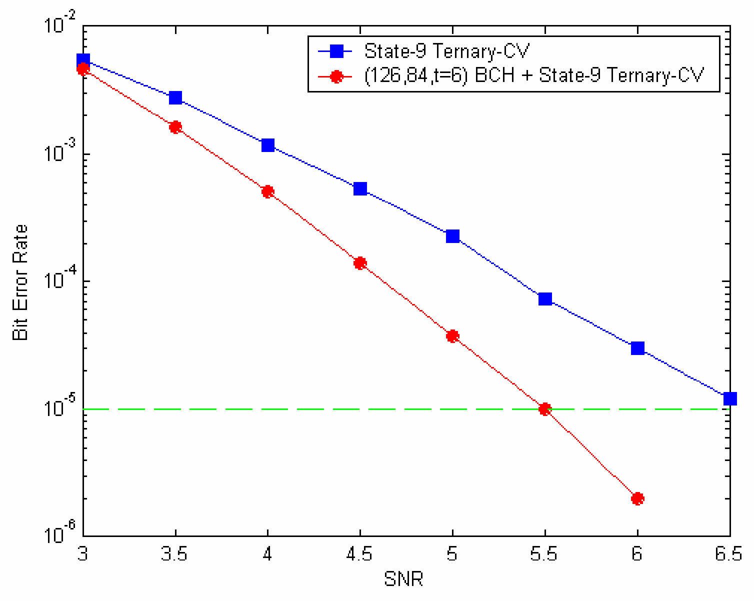

Fig. A.3 Performance comparison between ternary convolutional code and ternary convolutional code with BCH code

In addition, comparison with ternary convolutional code with 3PSK modulation, the concatenated coding in System2 has about 1dB performance gain as shown in Fig.

A.3. Now, we have a conclusion that by extended constellation to join redundant parity bits into convolutional codes the performance can be remarkably proved. Furthermore, in chapter 4, we know that inserting known digits into ternary turbo code in such way that one GF(3) known digit is inserted after two GF(3) information datum, the extra performance gain can be achieved. So, an experiment for inserting known digits into ternary convolutional code with the same method as former description is implemented. In Fig. A.4, the simulation result displays that the concatenated system has better performance than inserting known digit into ternary convolutional code, that is, inserting known digit into ternary convolutional code has almost no performance gain.

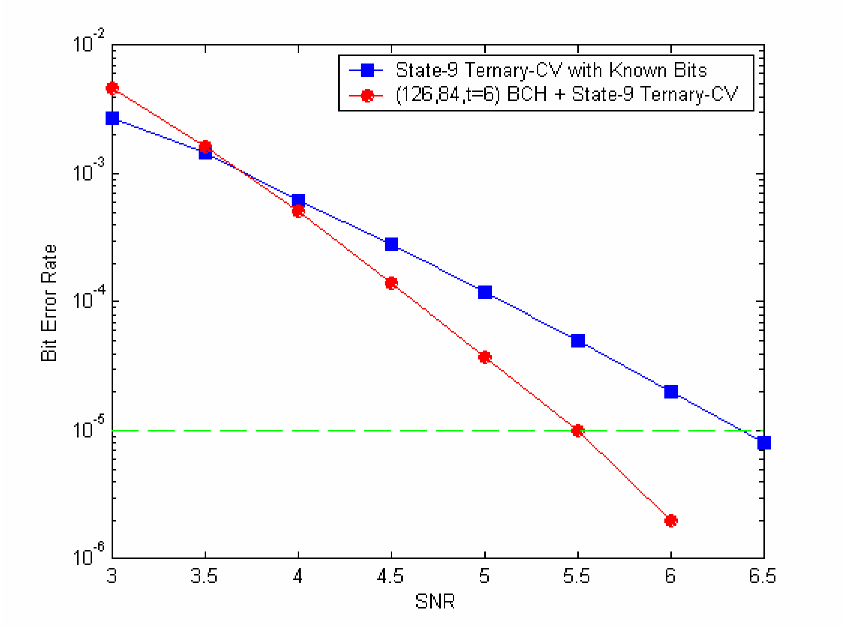

Fig. A.4 Performance comparison between ternary convolutional code with BCH and ternary convolutional code with inserting known digits

Appendix B: Scheme II (RS + Ternary Turbo Code)

Compared with a concatenated coding system with 3PSK modulation, a rate 1/3 state-8 binary turbo code with BPSK modulation has better performance in terms of BER as shown in Fig. B.2. The concatenated coding system formed from a rate 1/3 state-9 inner ternary turbo code and an outer (63, 42) RS code. For both turbo codes, the S-Random interleavers are employed. Simulation points out that System 1 has about 0.2 dB performance gain than System 2 at BER of 10^{-4}, but at high SNR the System 2 will has better performance than System 1 since error floor phenomenon. However, in turbo code system, using an outer code instead redundant information maybe is not an efficiency method due to computing analysis in section 5.2.2 shows that the computing complexity of ternary turbo code is more complex than binary turbo code. So, if System 2 has not significant performance gain, the cost of concatenated coding system is too large.

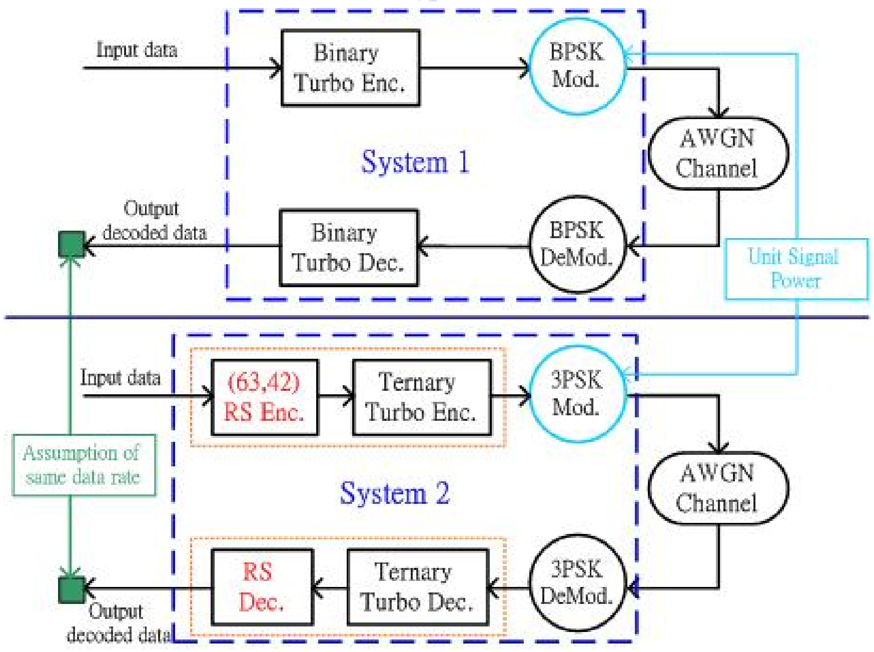

Fig. B.1 Structure diagram of binary turbo code with BPSK modulation and a concatenated coding system with 3PSK modulation.

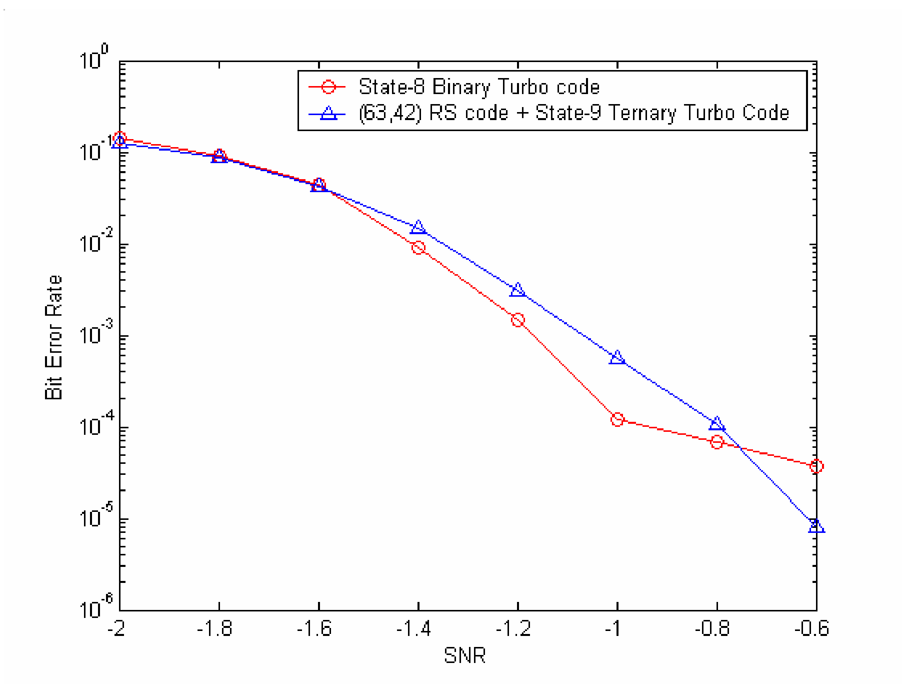

Fig. B.2Performance comparison between binary turbo code and concatenated coding formed form a ternary turbo code and a RS code.

BIBLIOGRAPHY

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: turbo-codes (1),” in *IEEE Int. conf. Communications (ICC)*, pp.1064-1070, May 1993.

[2] S. Benedetto and G. Montorsi, “Unveiling turbo codes: some results on parallel concatenated convolutional codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409-428, May 1996.

[3] D. Divsalar, S. Dolinar, R.J. McEliece and F. Pollara, “Performance analysis of turbo codes”, in *IEEE MILCOM’95*, vol. 1, pp. 91-96, Nov. 1995.

[4] V. Branda and J. Yuang, “Turbo Codes: Principles and Applications”.

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

[6] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate,” *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284-287, Mar. 1974.

[7] 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.

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

[9] J. Hagenauer, E. Offer and L. Papke, “Iterative Decoding of Binary Block and Convolutional Codes,” *IEEE Trans. Inform. Theory*, vol. 42, No. 2, Mar. 1996.

[10] J. A. Erfanian, S. Pasupathy, and G. Gulak, “Reduced Complexity Symbol Detectors with Parallel Structures,” in *IEEE GLOBECOM’90*, vol. 2, pp. 704-708,

Dec. 1990.

[11] P. Robertson, E. Villebrun, and P. Hoeher, “A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain,” in *Comm. ICC 95 Seattle*, vol. 2, pp. 1009-1013, June 1995.

[12] M. Jordan and R. Nichols, “The Effects of Channel Characteristics on Turbo Code Performance,” in *IEEE MILCOM’96*, vol. 1, pp. 17-21, Oct. 1996.

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

[14] A. Worm, P. Hoeher, and N. When, “Turbo-Decoding Without SNR Estimation,” *IEEE Comm. Letters*, vol. 4, no. 6, pp. 193-195, June 2000.

[15] S. A. Barbulescu, “Sliding Window and Interleaver Design,” *IEEE Electronics Letters*, vol. 37, no. 21, pp. 1299-1300, Oct. 2001.

[16] G. Ungerboeck, “Channel Coding with Multilevel/Phase Singals,” *IEEE Trans. Inform. Theory*, IT-28: 55-67, Jan. 1982.

[17] A. C. Reid, D. P. Taylor, and T. A. Gulliver, “Non-Binary Turbo Codes,” in *Proc. IEEE Int. Symp. Information Theory*, pp. 57, 2002.

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

[19] Yen-Hsu Shih, “A Dual Mode Channel Decoder for 3GPP2 Mobile Wireless Communications,” Master Thesis, NCTU, 2004.

[20] O.M. Collins, M. Hizlan, “Determinate State Convolutional codes,” IEEE Trans. Commun., vol. 41, issue 12, pp. 1785-1794, Dec. 1993.

作者簡介

姓名: 周毓堂

出生: 台北市

學歷 : 虎尾國小　揚子國中　大成商工

88.9 ~ 92.6 國立虎尾科技大學飛機工程學系(電子組)

92.9 ~ 94.10 國立交通大學 電子研究所 (oasis lab)