FACULTY OF GRADUATE STUDIES

On The Performance of MSOVA for UMTS and cdma2000 Turbo Codes

By

Hani Hashem Mis'ef

Supervisor

Dr. Wasel Ghanem

This Thesis was submitted in partial fulfillment of the requirements for the Masters Degree in Scientific Computing, from The Faculty of Graduate Studies at Birzeit University, Palestine.

January 2007

On The Performance of MSOVA for UMTS and cdma2000 Turbo Codes

By

Hani Hashem Mis'ef

This thesis was defended successfully on January 8, 2007 and approved by:

Committee Members	**Signature**
1. Dr. Wasel Ghanem	.....
2. Dr. Wasfi Al-Kafri	.....
3. Dr. Mohammad Saleh	.....

TO MY MOTHER, FATHER, AND FAMILY

مستخلص

تعد رموز تيربو (Turbo Codes) من أهم طرق الترميز الخاص بقنوات النقل، والتي تمَّ اعتمادها في كثير من أنظمة الاتصالات الرقمية الحديثة. سبب أهمية رموز تيربو يعود إلى أدائها الباهر في تصحيح الأخطاء الناتجة عن ضجيج القنوات بحيث يقترب كثيرًا من المستوى المثالي الذي تنبأ به شانون.

رموز تيربو تستخدم عدة طرق لفك الرموز أهمها طريقتي (SOVA) و (MAP). الطريقة الأولى سهلة التطبيق، ولكنَّ أداءها أسوأ من الثانية، غير أنَّ الطريقة الثانية أكثر تعقيدًا. هنالك العديد من التعديلات التي أجريت على الطريقتين إما للتحسين من أدائها أو للتقليل من تعقيدها. أحد التعديلات التي أجريت مؤخرًا على الطريقة الأولى هي طريقة (SOVA)–المعدلة والمسمّاة (MSOVA).

هذا البحث يدرس تطبيق طريقة (MSOVA) على رموز تيربو المستخدمة في نظامين معياريين من أنظمة الجيل الثالث للاتصالات الخلوية هما (UMTS) و (cdma2000)، بحيث يتم دراسة أدائها ومقارنته مع الطرق الأخرى، وإيجاد أفضل المعامِلات الخاصة بهذه الطريقة.

ABSTRACT

Turbo codes are one of the most important channel coding schemes because of its excellent performance that approaches shannon limit. So it is adopted by most of the modern digital communication systems. These codes are either a parallel or serial concatenation of two or more convolutional encoders separated by interleavers.

There are two main decoding algorithms for turbo codes; the MAP and SOVA algorithms. The MAP algorithm has a very good performance but with large complexity, while the SOVA algorithm is simpler but with worse performance. So many modifications appeared for both algorithms. Recently, a modification to the conventional SOVA algorithm called Modified SOVA or MSOVA was proposed, which suggests to add two scaling factors to the output of the conventional SOVA decoder.

In this research, the performance of the MSOVA decoding algorithm is studied for the turbo codes used by the two third generation cellular standards; UMTS and cdma2000, where the best values of the two scaling factors of the MSOVA algorithm suitable for these turbo codes are found.

ACKNOWLEDGMENTS

I would like to express my thanks to my advisor Dr. Wasel Ghanem for his support, patience and encouragement throughout this work. Also I would like to thank Dr. Wasfi Alkafri for his help and valuable comments on this thesis. I want also to send a special thank to Dr. Ali Grayyeb for his help in selecting the topic of my research.

Words can only fail to express my thanks and appreciation to my parents for their support and encouragement throughout my education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the importance of channel coding, and its role in modern communication systems are introduced briefly. Then, the idea of channel capacity is explained and the difference between the soft and hard decision is shown. The principle of turbo coding is then introduced. Finally, our work in this thesis is explained and the thesis is outlined.

1.1 Channel Coding

Shannon's second theorem [1] states that there is a theoretical maximum data rate (channel capacity) at which information can be transmitted through a channel with small probability of error if proper coding scheme is performed. So, errors caused by channel noise, interference or fading can be detected and/or corrected by means of what is called channel coding.

Channel coding plays a very important role in modern digital communication systems. Figure 1.1 shows a block diagram for a single digital communication link that explains the function of channel coding. In this diagram, the information is provided by a source which can be a telephone call or a computer file in its digital form. The information is passed through the *source encoder* which compresses the data by removing redundant information to increase the efficiency of the system. Then the data is passed through a *channel encoder* that adds a redundant information to it in a structured manner such that errors introduced by the channel can be detected or corrected. The *modulator* converts the information symbols into signals appropriate for transmission through the channel.

The channel in the diagram adds noise to the information, so the data becomes embedded in a very large number of errors. Then, the *demodulator* accepts the signals from the channel and converts it back to symbols that can be used by the channel

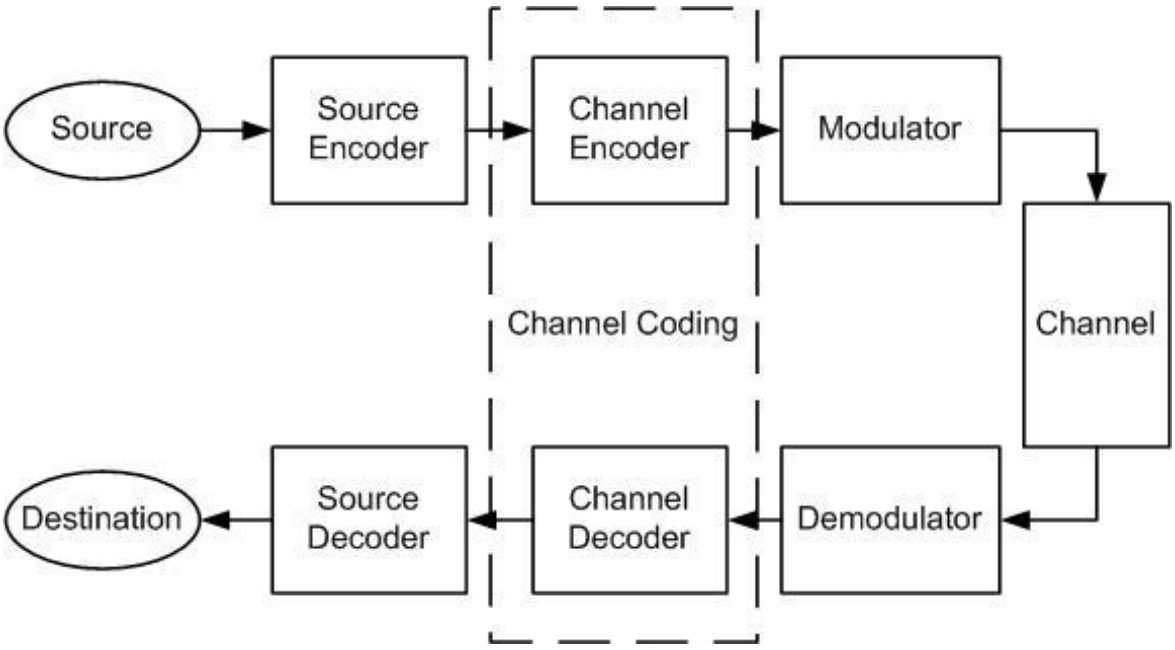

Figure 1.1: Block diagram of digital communication system.

decoder. Here, the *channel decoder* removes the redundant information added by the channel encoder and detects or corrects most of the errors introduced by the channel. Finally, the information is decompressed by the *source decoder* which adds the redundant information such that it becomes suitable for the destination.

A question may be asked, why to remove the redundant information by the source encoder and then adds redundant information by the channel encoder? The answer is simply, the redundant information on the data that comes from the source is unstructured, and removing it will increase the efficiency of the system without any degradation on the performance. Whereas the redundant information added by the channel encoder is structured to track errors and corrects them such that reliable transmission is achieved.

Channel codes can be classified into two types; Error Detection Codes (EDC) and Error Correction Codes (ECC). The first class can only detect errors introduced by the channel, but cannot repair them. The second class detects and then corrects the errors based on the received data. The system that combines error detection coding and error correction coding is called *error control coding*.

All Channel coding schemes accepts a message of k symbols and produces a codeword of n symbols where $k < n$. The code rate is defined as shown in Eq. (1.1) where

always $r < 1$.

$$r = \frac{k}{n} \tag{1.1}$$

1.2 Channel Capacity

Channel capacity which was introduced by Claude Shannon in [1], is the maximum data rate that the channel can support such that error free transmission is possible. For example, the channel capacity of the *Additive Wight Gaussian Noise* (AWGN) channel can be expressed as shown in Eq. (1.2).

$$C = \frac{1}{2}\log_2\left(1 + \frac{2E_s}{N_0}\right) \tag{1.2}$$

where E_s is the transmitted energy per symbol, and $\frac{N_0}{2}$ is the variance of the noise.

When using a coding scheme with code rate r, then the transmitted energy per symbol can be expressed as $E_s = rE_b$, where E_b is the transmitted energy per bit. So, for reliable transmission to take place, the code rate r must be less than the channel capacity C, i.e. $r < C$.

Now consider the signal to noise (SNR) ratio represented by $\frac{E_b}{N_0}$, then the minimum value of $\frac{E_b}{N_0}$ that required to acheive error free transmission (Shannon limit) can be expressed as shown in Eq. (1.3).

$$\frac{E_b}{N_0} \geq \frac{1}{2r}\left(2^{2r} - 1\right) \tag{1.3}$$

For practical issues, researchers take a *Bit-Error Rate* (BER) of (10^{-5}) as a reference to compare the performance of different coding schemes to the shannon limit.

1.3 Hard and Soft Decision Decoding

Based upon the received sequence y_k in any digital communication receiver, which is the output of the matched filter, the channel decoder makes a decision of which message was actually transmitted. So there are two types of decoding:

Hard Decision Decoding (HDD)

In HDD, the decoder uses a quantized values for the received message (the output of the matched filter) which are usually bits. These bits are used by the decoder to find errors introduced by the channel, and it is said that the decoder makes hard decision. This method of decoding is simple to implement, but its performance is bad because some informations are removed from the received sequence by quantization, these informations represent the reliability of the received sequence.

Soft Decision Decoding (SDD)

Unlike HDD, SDD decoder uses the output of the matched filter directly without quantization. So in this method of decoding, the decoder uses all the reliability values of the received sequence such that the performance of decoding process is improved of about ($3dB$) compared to that when hard decision decoding is used [2].

The soft decision decoding is more complex than hard decision decoding for two reasons [3]. The first reason is that the decoding process in SDD is done using real numbers compared to binary values in HDD. The second reason is that the SDD decoder needs to calculate the a posteriori statistics of the received message as well as the channel parameters.

Most of the traditional coding schemes use hard decision decoding such as linear block codes, cyclic codes, hamming codes, Reed-Solomon codes and convolutional codes. But the most recent codes which have the best performance use the soft decision decoding. Examples of such modern coding schemes are the turbo codes, and the *Low-Density Parity-Check* (LDPC) codes. Our work in this thesis is concentrated only on turbo codes.

1.4 Turbo Codes

Turbo codes which was first proposed in [4] is one of the most important known coding schemes. Because of their very good performance, turbo codes was adopted by many modern communication systems and standards such as the most famous third generation cellular standards; UMTS and cdma2000.

The first turbo code proposed in [4] was a parallel concatenation of two recursive systematic convolutional encoders (RSC) separated with an interleaver, so it is sometimes called *Parallel Concatenated Convolutional Codes* (PCCC). Computer simulations show that this type of codes has a very good performance at low signal-to-noise ratios (SNR), but it suffers from error floor and bad performance at higher signal-to-noise ratios. Some designs for PCCC schemes show that it is possible to increase the number of the RSC encoders to achieve lower code rates.

Another scheme of turbo codes was introduced in [5], which is called *Serial Concatenated Convolutional Codes* (SCCC). This type of codes concatenates two RSC encoders in serial manner, with the first encoder called the inner encoder and the other is called the outer encoder. SCCC turbo codes has lower performance than PCCC at low SNR, but it has a very good performance at higher SNR. So, sometimes to achieve good performance at low and high SNR, a hybrid concatenation (parallel and serial) of convolutional codes may be used.

The decoding of turbo codes is done in iterative fashion, where two decoders are used in the turbo decoder. The first decoder decodes the first encoder output, and the other decoder for the second encoder output . When the first decoder finishes decoding, it passes a reliability output to the second decoder. The second decoder uses this reliability values as input, and it in turns produces reliability output which is passed again to the first decoder. This process of decoding is called *iterative decoding*, and it is explained in details in the following chapters.

In this thesis, it is focused only on PCCC, since it is used in the two third generation cellular standards that are studied; UMTS and cdma2000.

1.5 Thesis Overview

In this thesis, the famous coding scheme known as *Turbo Codes* will be introduced, and a new decoding algorithm proposed in [6] and called *Modified SOVA* or *MSOVA* will be examined. This algorithm is based on applying two attenuators to the conventional SOVA decoder to improve the overall performance [6].

This decoding algorithm will be applied for turbo codes used by the two third generation cellular standards; *UMTS* and *cdma2000*, and then an extensive computer search will be done to find the best attenuators for these turbo codes schemes. Then, the performance of MSOVA decoding algorithm for these codes will be determined with several frame sizes and code rates, comparing all these results to the known MAP and SOVA algorithms.

Chapter two of this thesis will introduce the basics of turbo codes, including encoding, decoding, interleaving, puncturing and termination. In chapter three, the most known decoding algorithms will be investigated, including MAP algorithm and its approximation algorithms, also conventional SOVA and the MSOVA proposed in [6] will be explained.

A case study will be introduced in chapter four, showing the turbo codes used by the UMTS and the cdma2000 standards. Simulation results will be explained and showed in chapter five, and finally the conclusion and the future work will be included in chapter six.

Chapter 2

Principles of Turbo Codes

2.1 Introduction

In this chapter, the main principles for turbo codes are introduced. The structure of the turbo encoder is investigated in details explaining the reasons for the good performance of this coding scheme. Then the turbo decoding principles including the *Soft-input-Soft-Output* (SISO) and the iterative decoding principles are explained while investigating their effect on the code performance.

Some of the main components and design issues for turbo codes are introduced later in this chapter including interleaver purposes and its types, trellis termination methods, puncturing patterns and some of the main stopping rules for the iterative decoding process.

2.2 Preliminaries of Coding Theory

2.2.1 Types of Codes

Most of the known codes are categorized into two categories of codes; *Block Codes* and *Convolutional Codes*. All modern coding schemes are modifications and/or combinations of these codes.

Block Codes

Block codes accepts information messages block-by-block, where all blocks have the same length. These types of codes is memoryless, so each codeword depends only on the current message block, and is independent on the other codewords. Block codes work in the following manner; it accepts information in successive k-bit blocks, and adds for each block $n - k$ parity bits which are generated from the current k bits

of the message block with a particular function to produce the overall codeword as shown in Figure 2.1. Figure 2.1 shows a special type of codewords which is called systematic codeword where the information message appears explicitly as a block in the codeword.[1]

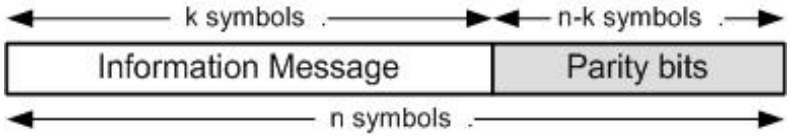

Figure 2.1: Codeword of a systematic block code

Convolutional Codes

Unlike block codes, convolutional codes treats incoming information in serial manner. The current output of a convolutional encoder not only depends on the current input, but it depends also on previous inputs and/or outputs. So, convolutional encoder has memory to store previous inputs.

Basically, convolutional encoder consists of a shift register (memory) and some modulo-2 arithmetic adders. So, it is similar to a Finite State Machine (FSM), where the state of the encoder is determined from the contents of the memory. The output of the encoder then can be calculated from its state and from the current input, so that no buffering is required for the input in convolutional codes unlike block codes. If the number of memory cells is m, then there are $m+1$ shifts needed to calculate the output (m states and the current input), this quantity is called the *constrained length* of the encoder K, where $K = m + 1$.

Figure 2.2 shows an example of a convolutional encoder with constrained length $K = 3$. In this example, since the input to the encoder does not appear explicitly in the output, the encoder is said to be non-systematic, and since the direct input to the encoder does not depend on previous states, it is called feedforward encoder, so this encoder acts as an FIR filter. This type of encoders is used in most known conventional convolutional encoders.

An example of a *Recursive Systematic Convolutional (RSC)* encoder is shown in Figure 2.3. This encoder is said to be recursive because its current state depends on its previous states in addition to the current input, so its response acts like an IIR filter. RSC encoders is widely used in concatenated codes, specially in the famous family of

[1] Block codes is beyond the scope of this thesis. For more details about block codes, see [2],[3].

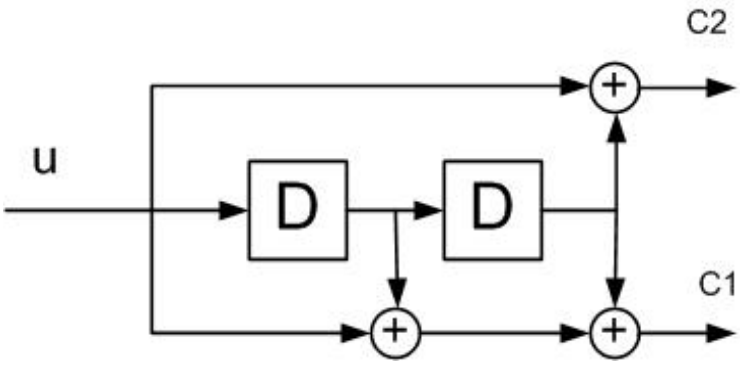

Figure 2.2: Non-systematic feedforward convolutional encoder with constraint length K=3.

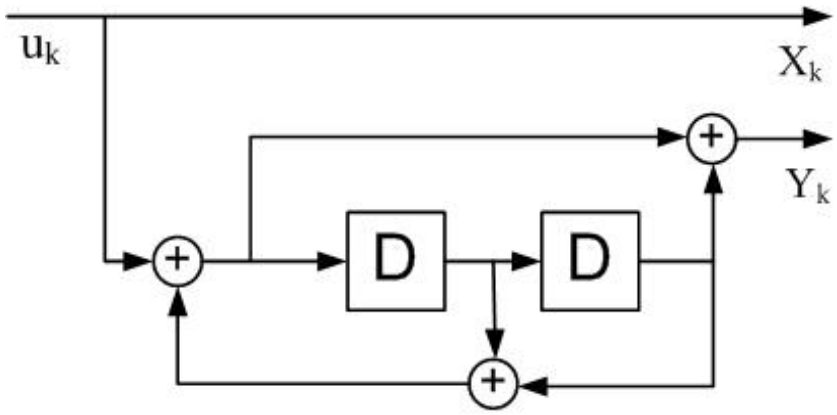

Figure 2.3: Recursive Systematic Convolutional encoder with constrained length K=3

codes, called Turbo Codes.[2]

The generator function of convolutional encoders is represented in the form $[g_0, g_1, ..., g_n]$ where $g_0, g_1, ..., g_n$ are octal numbers that represent the impulse response of each output of the encoder. for example, the encoder in Figure 2.2 has a generator polynomial of $[7, 5]$. If the encoder is RSC, then its generator is represented in the form $[1, \frac{g_1}{g_0}, \frac{g_2}{g_0}, ..., \frac{g_n}{g_0}]$, where g_0 represents the impulse response of the feedback branch, and $g_1, ..., g_n$ represents the impulse response of the parity outputs of the encoder. For the encoder shown in Figure 2.3, the generator polynomial is $[1, \frac{5}{7}]$.

Since Convolutional encoders act as a finite state machine, its function can be described using a state diagram. A state diagram of the encoder in Figure 2.3 is shown in Figure 2.4. If the state diagram is expanded in time, it forms the trellis structure of the code.

[2] All encoders used in this thesis are RSC encoders, since our research is focused on turbo codes.

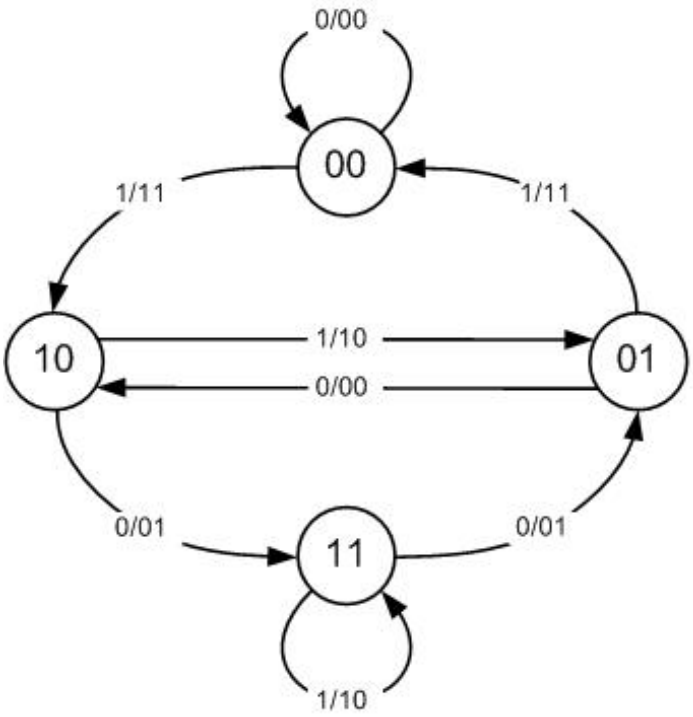

Figure 2.4: State diagram of [1,5/7] convolutional encoder.

Decoding Convolutional Codes The best known decoding algorithm for convolutional codes is the *Viterbi Algorithm*. It is first proposed in [7], then it is explained in [8]. The viterbi algorithm uses *Maximum Likelihood Decoding* principle by tracing the trellis of the code. The decoding procedure is done by selecting one surviving path from the trellis of the code which has the smallest distance from the received codeword.

To select the surviving path using viterbi algorithm, for each branch in the trellis, branch metrics which is the Hamming distance between the received code and the output code associated with the branch must be calculated. Then the path metric is calculated by adding the path metric of the previous state and the branch metric for the branch connecting between the previous state and the current state. The surviving path is selected such that the branch of the smallest path metric is chosen. Figure 2.5 shows portion of the trellis of the encoder shown in Figure 2.3, with the surviving path shown with bold lines. A description of the viterbi algorithm is shown in Algorithm 1.

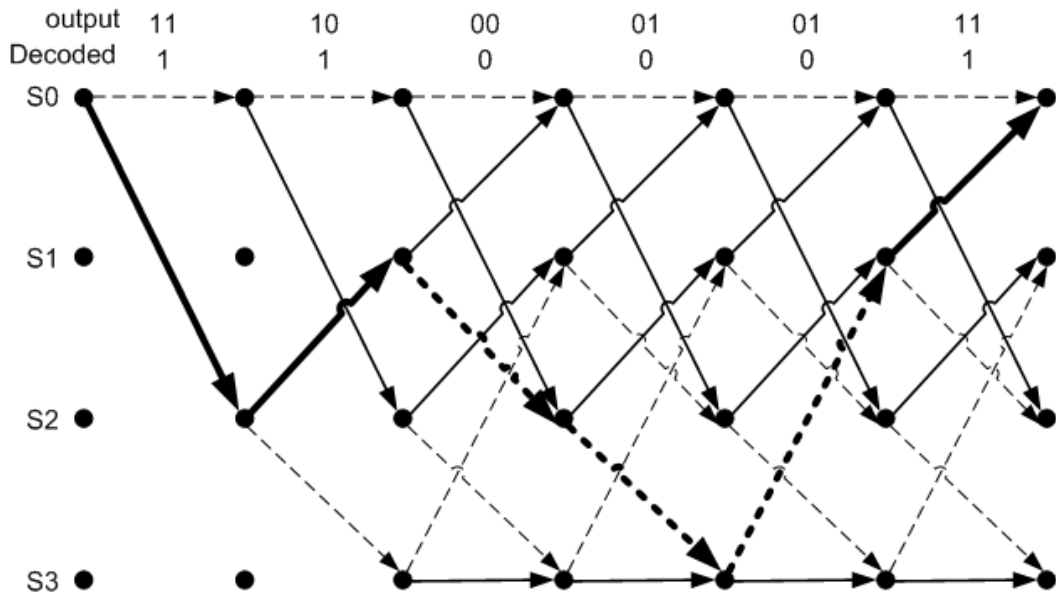

Figure 2.5: Trellis diagram of [1,5/7] convolutional encoder with 6 decoding steps

Algorithm 1 *The Viterbi Algorithm*

let t*: the decoding step time,* S_i^t*: state* i *at time* t*,*

$M(S_i^t)$*: path metric for state* i *at time* t*,* r^t*:the received vector at time* t*,*

$v_{j,i}$*: the output associated with the branch connecting states* j, i

$BM_{j,i}^t$*: the branch metric for the branch connecting states* j, i

$dH(c1, c2)$*: the Hamming distance between codewords* $c1, c2$

Initialization: set $t = 0$*,* $M(S_i^0) = 0$

for $t = 1, 2, 3, ...$

for $i = 0, 1, ..., 2^m - 1$

Calculate Branch metric: $BM_{j,i}^t = dH(r^t, v_{j,i})$*,* $BM_{k,i}^t = dH(r^t, v_{k,i})$

Add, compare, Select: $M(S_i^t) = \min\{M(S_j^{t-1}) + BM_{j,i}^t, M(S_k^{t-1}) + BM_{k,i}^t\}$

Update path

end

end

2.2.2 Properties of Codes

The question is now, how to distinguish good from bad codes? In fact, there are some properties of codes that can be investigated to say if any coding scheme is good or bad. In the following subsections, the most important properties of codes are introduced briefly.

Linearity

A given coding scheme is said to be linear if the result of the modulo-2 arithmetic addition (XOR operation) of any two codewords in this code is also a codeword in the code.

Since linear codes are easy to encode and decode, and because of its good performance, the most known coding schemes are linear.

Hamming distance

The *Hamming distance* between any two codewords in the code is the number of positions in which they differ. From this definition, the *Free Hamming distance* d_{free} of the code is defined, which is the minimum Hamming distance between any two codewords in the code. For convolutional codes, d_{free} can be found by tracing the state diagram of the given code.

Hamming Weight

The *Hamming Weight* of a codeword is the number of non-zero elements in the codeword. The *Minimum Weight* $w_{\min}$ of a code is the smallest Hamming weight of any nonzero codeword. If the code is Linear, then the free Hamming distance of the code is equal to the minimum weight of the code [2].

Sometimes, it is important to find the *Weight Distribution* of the code to examine its properties. For example, to design good interleavers for turbo codes, its weight distribution must be determined.

2.3 Turbo Encoder Structure

As stated in the previous section, turbo encoder consists of a parallel concatenation of two or more recursive systematic convolutional encoders RSC separated by interleavers as shown in Figure 2.6. The input for the first RSC encoder is the same as the input stream to the turbo encoder, while the inputs to the other RSC encoders are the interleaved versions of the turbo encoder's input stream. The function of the interleaver in the encoder is to permute the input stream to the turbo encoder such that the output of the interleaver appears random compared to its input.

The systematic output of the turbo encoder is taken from the systematic output of the first RSC encoder, while the systematic outputs from the other RSC encoders are truncated. Then the systematic output of the first RSC encoder and all the parity outputs of all the RSC encoders are multiplexed serially to form the overall codeword of the turbo code.

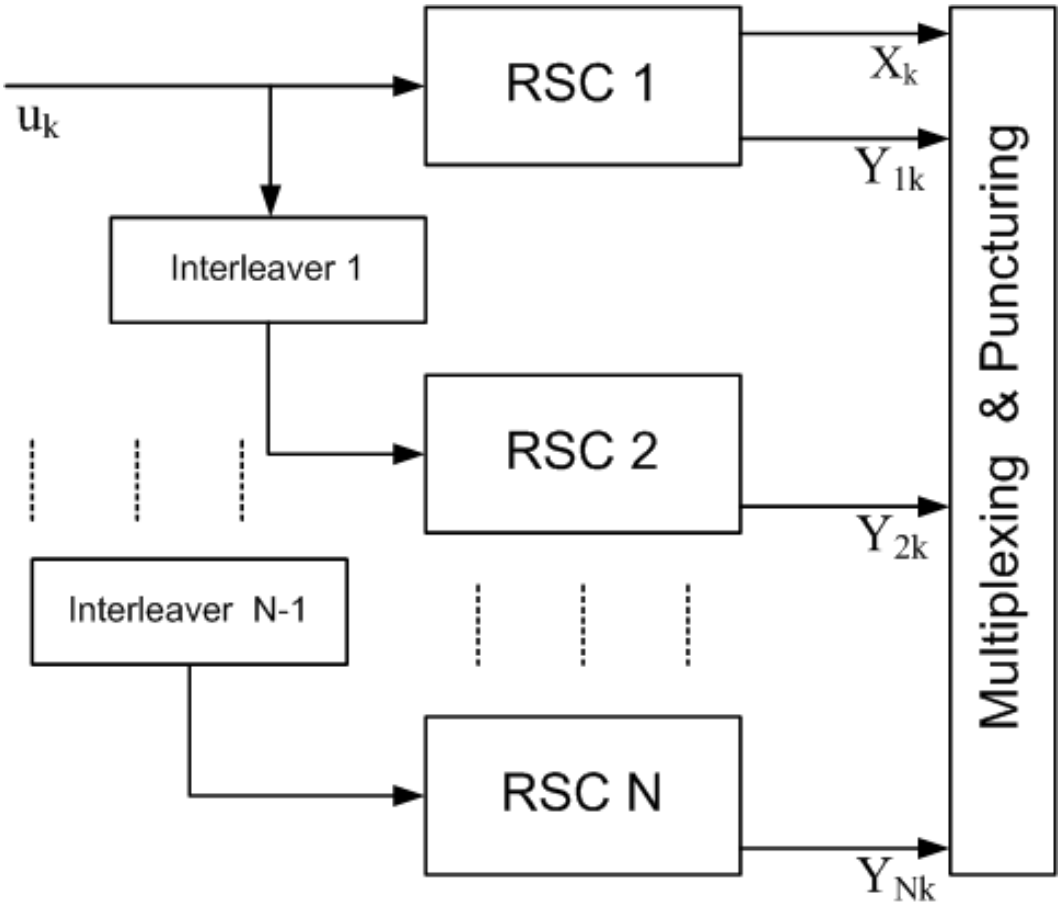

Figure 2.6: Block diagram of turbo encoder in its general structure.

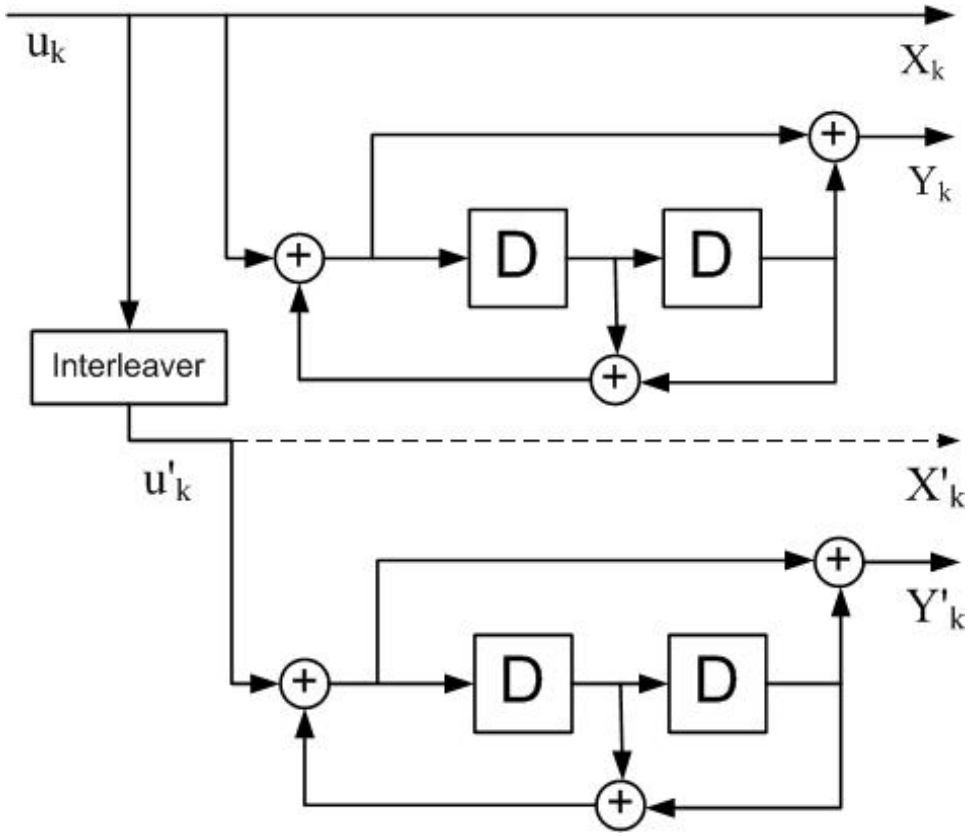

Figure 2.7: Turbo encoder example

To achieve a variable rate turbo code, puncturing may take place on the output of the encoder where some of the parity outputs of the turbo encoder may be omitted according to a special puncturing pattern. Turbo code designer must be careful in choosing puncturing pattern, since bad choice of the puncturing pattern will cause a large degradation in the performance.

An example for a turbo encoder is shown in Figure 2.7, where two RSC encoders are concatenated in parallel. The two RSC encoders are identical which is the case for the most known PCCCs, with generator polynomials $[1, \frac{5}{7}]$. These RSC encoders are the same to that shown in Figure 2.3, where its state diagram is shown in Figure 2.4.

Unlike conventional convolutional encoders which need to be implemented with large memory elements to have better performance, the RSC encoders used in turbo codes use only few memory elements to have a very good performance. So conventional convolutional encoders are much more complex than that used in turbo encoders.

The infinite impulse response of the RSC encoder makes its output code to have a large Hamming weight with only a few number of low weight codewords. The number of

these low weight codewords can be decreased in the turbo encoder by proper mapping of low weight codewords and high weight codewords from the first and the second RSC encoder by means of good design for the interleaver.

It was shown in [9] that the error floor on the performance that occurs at moderate SNR is due to the existence of low codewords in the turbo code. So to lower the error floor, the free distance of the code must be increased or the number of low weight codewords must be decreased. The author in [9] also showed that using RSC encoders with primitive feedback polynomials will give a higher free distance of the code.

The most important result shown in [9] was that by proper design of the interleaver, it is possible to reduce the low weight codewords by a process called *Spectral Thinning*. So by spectral thinning of turbo codes, it is possible to lower or remove the error floor on the performance.

2.4 Turbo Decoding

In this section, a framework for the turbo decoding process is introduced, where the basic structure of the decoder and the concept of iterative decoding are explained.

When the encoding of a codeword is finished, the encoded bits are modulated and transmitted through the channel. This thesis concentrates only on the *Additive Wight Gaussian Noise channel* (AWGN). At the receiver side, the signal is demodulated and the output of the matched filter is passed to the turbo decoder. Notice that the decoder input are soft values not hard values (bits).

At the front of the turbo decoder, the channel reliability values are calculated, and the soft inputs to the decoder are formulated in a proper manner for decoding. All inputs and outputs of the component decoders are implemented in *Log Likelihood Ratios* (LLRs) which is described in the following subsection.

2.4.1 Log Likelihood Ratio Representation

For complexity considerations, all the information in the turbo decoder is represented in LLRs. LLR representation of the information simplifies the decision of the received bit from absolute comparison of real values to only comparing the sign of the LLR value. Eq. (2.1) shows the LLR representation of the natural logarithm of the ratio of the conditional probability of that $u_k = +1$, given that the received sequence is y to the conditional probability of that $u_k = -1$, given that the received sequence is y, and is denoted by $L(u_k|y)$.

$$L(u_k|y) \triangleq \ln\left(\frac{P(u_k = +1|y)}{P(u_k = -1|y)}\right) \tag{2.1}$$

The LLR values $L(u_k|y)$ are called the *Log A Posteriori Probabilities*, and these values are what the component decoders in the turbo decoder search for.

The inputs to the turbo decoder are implemented in LLR based on the matched filter output. These LLRs are denoted by $L(y_k|x_k)$ which is the natural logarithm of the ratio of the conditional probability of that the matched filter output is y_k given that the transmitted bit is $x_k = +1$ to the conditional probability of that the matched filter output is y_k given that the transmitted bit is $x_k = -1$.

$$L(y_k|x_k) \triangleq \ln\left(\frac{P(y_k|x_k=+1)}{P(y_k|x_k=-1)}\right) \tag{2.2}$$

Assuming AWGN fading channel, the conditional probabilities shown in Eq. (2.2) can be found using Eqs. (2.3),(2.4).

$$P(y_k|x_k=+1) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{E_b}{2\sigma^2}(y_k-a)^2\right) \tag{2.3}$$

$$P(y_k|x_k=-1) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{E_b}{2\sigma^2}(y_k+a)^2\right) \tag{2.4}$$

Where σ is the standard deviation of the noise, E_b is the per bit transmitted energy and a is the fading coefficient of the channel.

As indicated previously, if the channel is AWGN with BPSK modulation, then $L(y_k|x_k)$ can be simplified as followed:

$$\begin{aligned}
L(y_k|x_k) &\triangleq \ln\left(\frac{P(y_k|x_k=+1)}{P(y_k|x_k=-1)}\right) \\
&= \ln\left(\frac{\exp\left(-\frac{E_b}{2\sigma^2}(y_k-a)^2\right)}{\exp\left(-\frac{E_b}{2\sigma^2}(y_k+a)^2\right)}\right) \\
&= \left(-\frac{E_b}{2\sigma^2}(y_k-a)^2\right) - \left(-\frac{E_b}{2\sigma^2}(y_k+a)^2\right) \\
&= 4a\frac{E_b}{2\sigma^2}y_k \\
&= L_c y_k
\end{aligned} \tag{2.5}$$

Here L_c is constant through the decoding process and is called the Channel Reliability, since it only depends on the channel parameters. L_c can be expressed as shown in Eq. (2.6).

$$L_c = 4a\frac{E_b}{2\sigma^2} \tag{2.6}$$

The soft channel inputs to the turbo decoder are represented in the form as shown in Eq. (2.5).[10]

2.4.2 Turbo Decoder Structure

Turbo decoding process is done in two decoding phases, each with separate decoder as shown in Figure 2.8. The first decoder decodes information that belongs to the first RSC encoder, and the second decoder decodes information that belongs to the second RSC encoder.

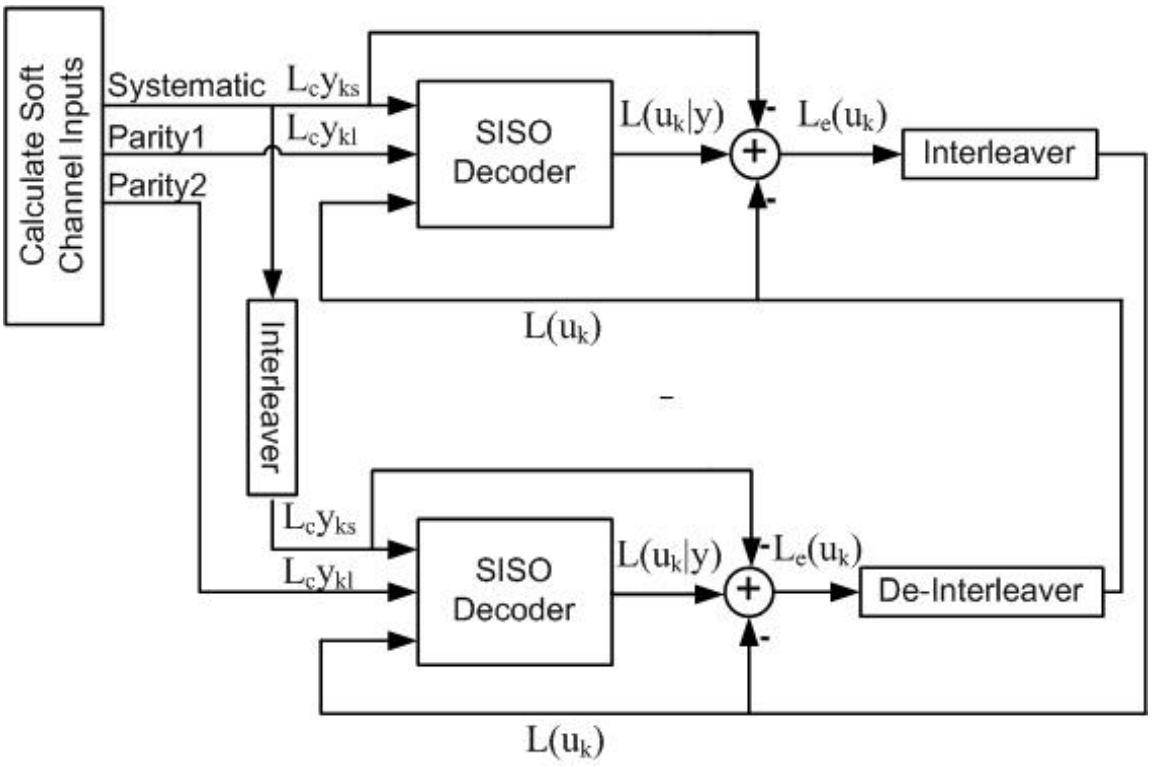

Figure 2.8: Turbo decoder block diagram

The two component decoders in the turbo decoder are *Soft-In-Soft-Out* (SISO) decoders, where the inputs are in the form of soft channel inputs and reliability values from the other decoder, and the output is the a posteriori probability. The reliability values that enter the decoder from the other decoder denoted by $L(u_k)$ is called the *Apriori* information. This information is sometimes called *Intrinsic Information* which is the information about the bits to be decoded provided by the other decoder before the decoding process begins. The soft channel inputs to the decoder are the matched filter outputs multiplied by the channel reliability L_c.

The direct output to the component decoder is the *a posteriori* information denoted by $L(u_k|y)$. Here, the purpose of the component decoder is to maximize the a posteriori information on the bits to be decoded taking into account the apriori infor-

mation and soft channel inputs. The soft channel inputs and the apriori information is excluded from the a posteriori information to produce a new information qauntity called the *Extrinsic Information* and is denoted by $L_e(u_k)$. This information is what the component decoder passes to the other component decoder. The relation between the extrinsic and the intrinsic information is shown in Eq. (2.7).

$$L_e(u_k) \triangleq L(u_k|y) - L(u_k) - L_c y_k \tag{2.7}$$

In Eq. (2.7), the extrinsic information is assumed to be independent from the intrinsic information. But in fact, this is not certain, since there is a dependency between the extrinsic and intrinsic information. This dependency will affect the iterative gain of the turbo decoder, and so the overall performance of the turbo code will degrade. So to have a better turbo code performance, the correlation coefficient between the extrinsic and intrinsic information must be reduced.

2.4.3 The SISO Decoder

The core of the decoding process is done in the SISO decoder. The SISO decoder uses the trellis structure of the RSC encoder to trace all possible states of the code and calculate LLR information by giving all the branches in the trellis a value called *Branch Metric*. These branch metrics are used through the decoding process to obtain an estimate for LLR of each bit.

There are two main algorithms for the SISO decoder; *Maximum A Posteriori* algorithm (MAP) and *Soft Output Viterbi Algorithm* (SOVA). The MAP algorithm was first proposed in [11], and because of its complexity, many approximations to this algorithm appeared such as the Log-MAP and the MAX-Log-MAP algorithms. The SOVA algorithm proposed in [12] is simpler than the MAP algorithm, but the performance of the MAP algorithm is better than SOVA. The SOVA algorithm keeps only one surviving path for each state in the trellis, and discards the other path.

Because of its bad performance, many modifications was proposed for the SOVA algorithm. One of these algorithms is the *Modified SOVA* or MSOVA proposed in [6], which suggested to add two scaling factors to the output of the conventional SOVA decoder. All these algorithms will be discussed in details in chapter three.

2.4.4 Iterative Decoding Process

The iterative decoding of turbo codes is one of the most important properties of turbo codes that improves its performance. The decoding process of the turbo decoder is described as follows; at the first iteration, the first SISO decoder takes the soft channel inputs corresponding to the systematic bits and the parity bits of the first

RSC encoder, while the intrinsic information is set to zero. The extrinsic information is then calculated by excluding the systematic soft channel inputs and the intrinsic information from the a posteriori information which is the output of the SISO decoder. Then the extrinsic information is interleaved and passed to the second decoder as intrinsic information.

The second SISO decoder takes its intrinsic information from the first decoder, and takes the interleaved version of the systematic soft channel input and the soft channel inputs corresponding to the parity bits of the second RSC encoder. The extrinsic information is then calculated and interleaved to be passed to the first encoder as intrinsic information.

In the second iteration, the first SISO decoder now has an LLR estimates for the decoded bits which are the intrinsic information provided by the other decoder. these intrinsic information improves the decoder's capability for decoding with better performance. When the number of iterations is increased, the performance gets better until some limit when increasing the number of iterations does not significantly improve the performance. When the number of iterations is increased, the decoding latency is increased, so a compromise between latency and performance must be considered. The number of iterations may be fixed to a value based upon experimental study, or stopping rule my be used in the algorithm to decide when to stop decoding.[3]

2.5 Interleavers

Interleaver is a major element in the turbo coding system. It appears in the encoder as well as the decoder as shown in Figure 2.6 and Figure 2.8. The following two subsections explain the effect of interleaving on turbo codes and introduce some types of interleavers.

The function of the interleaver in the turbo encoder and decoder is to permute its input stream to produce another stream that contains the same elements of the input but with a different order such that the input and the output seem uncorrelated.

2.5.1 Purpose of the Interleaver

There are two main purposes for the interleaver in turbo codes [13]. The first one is to reduce the correlation between the intrinsic and extrinsic information in the turbo decoder to increase the iterative decoding gain of the decoder which increases the overall performance of the decoder. The design of the interleaver must take this purpose into consideration. To do so, a measure for the performance of iterative

[3] For more details about decoding turbo codes, see [2],[3] and [10].

decoding called *Iterative Decoding Suitability* (IDS) [14] is defined to be used as a parameter in the interleaver design.

The second purpose for the inerleaver is to improve the weight distribution for the turbo code. It is known that turbo codes has a good weight distribution compared to other coding schemes, where this good weight distribution is a result of two main components in the turbo encoder, the RSC encoder and the interleaver.[4]

The weight distribution of a code that uses a recursive encoder is better than non-recursive encoder. The recursive encoder produces high weight codes even for low weight input messages, but there still be some input messages that cause the recursive encoder to produce low weight codewords.

The overall weight of the turbo encoder is the weight of the first RSC encoder added to the weight of the second RSC encoder. For a given input message, if the first RSC encoder produces low weight codeword, the other RSC encoder is likely to produce a high weight codeword because the input message is permuted by the interleaver before it enters the second RSC encoder, so the overall weight for the turbo code is high.

If the weight distribution of the turbo code is not taken into consideration in the interleaver design, there will be a considerable number of low weight codewords in the code, and so the free distance of the code will be relatively small. This is not good for the turbo code since there will be an error floor that affect the performance of the coding scheme. This error floor is affected by the free distance of the code as shown in Eq. (2.8). Here, the free distance is the slope of the error floor, so if the free distance is small, then the slope of the error floor is also small.[2]

$$P_b \approx \frac{N_{free}\widetilde{w}_{free}}{N} Q\left(\sqrt{\frac{d_{free} 2rE_b}{N_0}}\right) \tag{2.8}$$

where N_{free} is the number of codewords with distance d_{free} , and $\widetilde{w}_{free}$ is the average weight for the message that makes the encoder to produce codewords with distance d_{free}.[2]

2.5.2 Interleaver Types

The design of interleaver is a very important issue in turbo coding schemes. Researchers tried to find an interleaver structure that meats the requirements for the turbo code introduced in the previous subsection with the minimum complexity for implementation. So many types of interleavers was found ranging from simple to very complex. Here are some of the most known today interleavers.

[4] To learn how to find the weight distribution of the turbo codes, see [15].

Rectangular Interleaver

This type of interleavers is the simplest one for implementation, but its performance is very bad. It uses an $R \times C$ rectangular matrix with R rows and C columns, where the input stream is written row by row, while the output is read out column by column as shown in Figure 2.9.

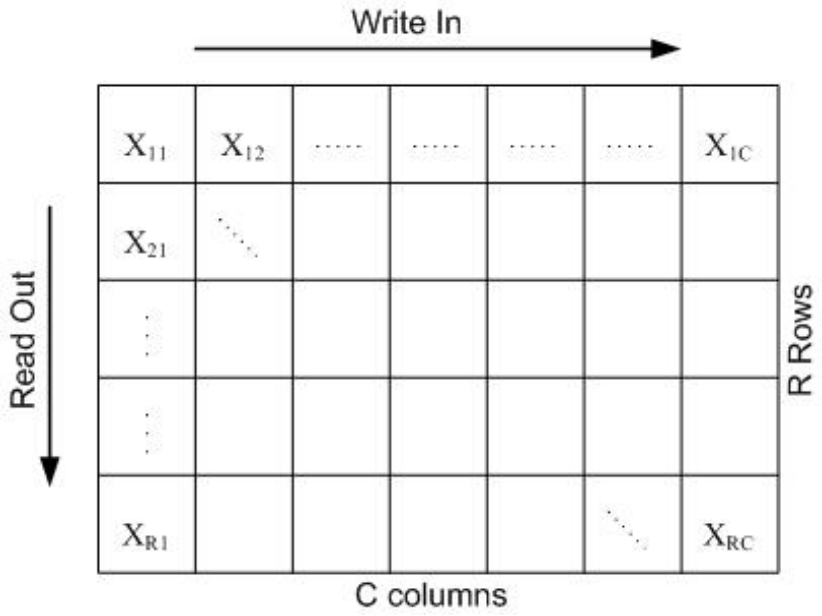

Figure 2.9: Rectangular Interleaver

Diagonal Interleaver

The structure of the diagonal interleaver is the same as rectangular interleaver. The input stream is written row by row but the output is read out in diagonals starting its output from the first element of the first row[16].

Helical Interleaver

As in diagonal interleaver, the input stream in helical interleaver is written row by row and read out in diagonals. The difference between them is in the starting position to read the output, where the helical interleaver starts to read out from the first element of the last row[16].

Berrou Interleaver

It is the first interleaver used in turbo codes proposed in [4]. This interleaver is a rectangular interleaver, where the input stream is written row by row, but the output is read out in a pseudo-random manner.

Pseudo-Random Interleaver

This type of interleavers maps the N-block input stream in a pseudo-random manner to an N-block output stream. This method of mapping helps in making the code seems random and also reduces the correlation between the extrinsic and intrinsic information in the turbo decoder such that an improvement in the performance is achieved.

Every interleaver to have good performance must have some random properties in its structure, so that random interleavers are considered as a reference for the comparison between the performance of all other interleavers.

S-Random Interleavers

The structure of S-random interleaver is the same as pseudo-random interleaver except that it has another restriction on the *Spread* S of the permuted sequence. This restriction states that if two inputs to the interleaver, say (i, j) are separated with a distance less than or equal to S, then they must be mapped to distance greater than S in the permuted sequence. Consider (i, j) as the two indices for the interleaver, then if

$$|i - j| \leq S$$

then, the interleaver design must guarantee that

$$|\pi(i) - \pi(j)| > S$$

where $\pi(.)$ is the permutation function for the interleaver [17].

The spread S must be chosen as large as possible to have good performance, but if it is so large, the interleaver design algorithm may fail to converge to the desired interleaver. The convergence of the interleaver design algorithm is guaranteed if $S < \sqrt{\frac{N}{2}}$, where N is the interleaver size [13].

Optimal interleaver

The design of optimal interleaver is very complex, and it is dependent on the structure of the turbo code. So every coding scheme will have a different interleaver design. The

design must take into consideration the weight distribution of the turbo code, where extensive search is done for the best permutation scheme that produces codewords with an optimum weight distribution. This can be achieved by first generating a random interleaver, and then generating all the possible input messages and encoding them while investigating the weights of all codewords to determine the minimum weight for the code. This process is extensively repeated to obtain the optimal interleaver. Two examples of such an optimal interleavers is that used for UMTS and cdma2000 standards which will be introduces later in this thesis.

2.6 Trellis Termination

Trellis termination is a very important issue in the design of turbo codes. If no trellis termination is used in the turbo coding scheme, poor decoding performance will take place near the end of the trellis, causing the overall performance of the turbo code to be degraded.

Since the constituent encoders used in the turbo encoder are recursive, the termination of each encoder not only depends on the input stream as in non-recursive encoders, but it also depends on the current state of the encoder. So each constituent encoder can be terminated using a tail bits that depends on its state which are added to the input message entering the encoder. But because of the presence of the interleaver, the two constituent encoders are terminated in different states, so they need two different tail bits to be terminated. This relation between trellis termination and the interleaver shows that the termination and the interleaver design are dependant to each other.

Five strategies for trellis termination in turbo codes can be considered as follows [18]:

- No termination of any RSC encoder: this choice degrades the performance, and makes the design of the interleaver very sensitive.
- Termination of the first RSC encoder: to terminate the first RSC encoder, the proper tail bits which depend on the final state of the encoder are attached to the input message. These tail bits are also passed to the second RSC encoder after interleaving, where care must be taken in the interleaver design to permute these tail bits far from the end of the trellis of the second RSC encoder.
- Termination of the second RSC encoder.
- Termination of both RSC encoders with separate tail bits: since the two RSC encoders is terminated in different states, each encoder requires different tail bits

to be appended to its input to be truly terminated. The interleaver is off while the tail bits being encoded by each encoder.

- Termination of both RSC encoders with single tail bits: the first RSC encoder is terminated with a tail bits, and then these bits are appended to the message and passed to the other encoder through a special designed interleaver. This interleaver makes the two encoders to be terminated in the same state, and it is called *Self-Terminating* interleaver.

2.7 Puncturing

As stated in the previous sections, when the two RSC encoders in the turbo code are concatenated in parallel, the systematic output of the first RSC encoder and the parity outputs of both encoders are multiplexed such that the overall code rate is $r = \frac{1}{n}$, where $n = 1 + 2 * z$, assuming z is the number of parity outputs of each RSC encoder. But to achieve variable rate turbo code, some of the parity bits of the RSC encoders must be omitted (punctured) from the output code. This process is called *Puncturing* of turbo codes, and the way in which the parity bits are punctured is called *Puncturing Pattern*.

So puncturing is done only on parity bits not on systematic bits, since the systematic bits are needed by the iterative decoder to obtain good results.

Two classes of puncturing patterns are found for turbo codes; the first class has a low rate code where its code rate takes the form of $r = \frac{1}{n}$, where $n = 2, 3, ..., 1 + 2 * z$. This class of puncturing is more conventional, and it is used by the cdma2000 third generation cellular standard.

The other class of puncturing has a high rate code [19], with code rate of the form $r = \frac{k}{k+1}$, where $2 \leq k \leq 16$. In this class of puncturing, only one parity bit from each RSC encoder appears in the output code for each $2k$ systematic bits.

The puncturing pattern must be chosen to achieve the best performance. The choice of the puncturing pattern should be based on the output weight distribution to achieve higher minimum distance of the code which is the key for good code performance[19].

2.8 Stopping Rules for the Iterative Decoder

As stated in the previous sections, when the number of iterations in the turbo decoder is increased, the performance gets better. But increasing the number of iterations will increase the latency of the decoder which limits the decoding speed of the decoder.

So a compromise between these factors must take place to choose the proper fixed number of iterations.

But since the input messages differ in its needs for the number of iterations, the worst case is chosen to find the fixed number for the iterations. To avoid this, a stopping rule may be used to decide when the iterative decoding can be halted with the same probability of bit error performance as in the fixed number of iterations.

Here are some of these stopping rules [2]:

The Cross Entropy Stopping Rule

This method uses the cross entropy between the extrinsic information output to the first component decoder in the turbo decoder, and the extrinsic information output to the second component decoder. The cross entropy is defined as a measure of similarity between these two quantities such that if the quantities are approximately equal, then the cross entropy between them is minimal.

The cross entropy $T(l)$ is defined as shown in Eq. (2.9)

$$T(l) \approx \sum_{k=0}^{N-1} \frac{\left(\Delta_e^{[l]}\right)^2}{\exp\left(L^{[l,1]}\left(u_k|y\right)\widehat{\widetilde{u}}_k^{[l]}\right)} \tag{2.9}$$

where l is the iteration step, $\widehat{\widetilde{u}}_k^{[l]}$ is the estimate for bit k at the l iteration step, $L^{[l,1]}\left(u_k|y\right)$ is the a posteriori probability output of the first decoder at the l iteration step, $\Delta_{e,t}^{[l]}$ is the difference between the extrinsic information output to the second and the first component decoders at the l iteration step, where $\Delta_e^{[l]} = L_e^{[l,2]}\left(u_k\right) - L_e^{[l,1]}\left(u_k\right)$.

When the cross entropy at any iteration is less than a certain threshold, say $T(l) < 10^{-3}T(1)$, then the decoding process can be stopped.

The Sign Change Ratio Rule

This rule depends on the number of changes of signs of the extrinsic information output to the first decoder compared to the extrinsic information output to the second decoder denoted by $C(l)$. When $C(l)$ is less than certain threshold, the decoding process is terminated.

The Hard Decision Aided Rule

This rule is similar to the sign change rule, except that the sign is taken for the a posteriori probabilities output to the first and the second decoders. If they have the same sign for all values in the frame, then the decoding process is stopped.

Chapter 3

Turbo Decoding Algorithms

3.1 Introduction

As indicated in the previous chapter, the SISO decoder is the most important element in the turbo decoder. The function of the SISO decoder is to find an LLR estimates in the form of a posteriori probabilities $L(u_k|y)$ for the decoded bits given the systematic and parity soft channel inputs and the apriori information as seen in Figure 3.1.

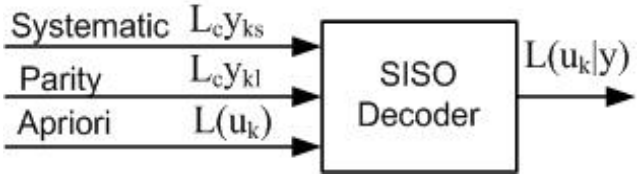

Figure 3.1: The SISO Decoder

The SISO decoder uses an algorithm to trace the trellis of the code, and calculates the proper LLRs. There are two main types of decoding algorithms for convolutional codes and hence for the SISO decoder used in turbo codes. The first algorithm is the *Maximum A Posteriori* (MAP) algorithm which was first proposed in [11], this algorithm is sometimes called BCJR algorithm named after its inventors, Bahl, Cocke, Jelinek, and Raviv. it will be seen in this chapter that this algorithm is very complex, so many algorithms based on an approximations for the MAP algorithm will be introduced in this chapter. These algorithms include The Log-MAP algorithm, the Max-Log-MAP algorithm, the Constant-Log-MAP algorithm, and the Linear-Log-MAP algorithm.

The second algorithm is the *Soft Output Viterbi Algorithm* (SOVA) proposed in [12]. This algorithm is simpler than the MAP algorithm, but it has a worse perfor-

mance. A modification to the SOVA algorithm was proposed in [6], and was called *Modified SOVA* or *MSOVA*. This Algorithm is the same as the conventional SOVA algorithm, except that it adds two scaling factors to the output of the conventional SOVA decoder. Simulation results show that MSOVA has a large improvement in the performance over conventional SOVA.

In this chapter, all these algorithms are investigated in details, starting with the MAP algorithm and its approximations, and then the SOVA and the MSOVA algorithms.

3.2 The MAP Algorithm

MAP algorithm computes the a posteriori probabilities of the input information bits by examining the received bits symbol by symbol. The a posteriori probability shown in Eq. (3.1) can be written using Bay's rule as shown in Eq. (3.2).

$$L(u_k|y) \triangleq \ln\left(\frac{P(u_k=+1|y)}{P(u_k=-1|y)}\right) \tag{3.1}$$

$$L(u_k|y) = \ln\left(\frac{P(u_k=+1,y)}{P(u_k=-1,y)}\right) \tag{3.2}$$

Suppose that the transition from state (p) at time $(k-1)$ to state (q) at time (k) in the trellis structure of the code is associated with $u_k=+1$ input bit[1], then the joint probability $P(u_k=+1,y)$ can be written as shown in Eq. (3.3).

$$P(u_k=+1,y) = \sum_{T^+} P\left(S_{k-1}=p, S_k=q, y\right) \tag{3.3}$$

where T^+ is the set of all transitions (p,q) that are associated with the input bit $u_k=+1$. In the same way, we can write

$$P(u_k=-1,y) = \sum_{T^-} P\left(S_{k-1}=p, S_k=q, y\right) \tag{3.4}$$

where T^- is the set of all transitions (p,q) that are associated with the input bit $u_k=-1$.

The received sequence y can be divided into three partitions; y_{prior}, $y_{current}$ and y_{future}, representing the prior observations, current observations and future observations respectively. So that the total received bits y is the union of the three partitions, and the probability of y is the joint probability of the three partitions such that the right side probability in Eq. (3.3) can be written as

[1] The calculation is the same if the input bit is $u_k=-1$.

$$P\left(S_{k-1}=p, S_k=q, y\right)=P\left(S_{k-1}=p, S_k=q, y_{prior}, y_{current}, y_{future}\right) \tag{3.5}$$

Using Bay's rule, the probability in Eq. (3.5) can be modified[2] as shown in Eq. (3.6).

$$\begin{aligned} P\left(S_{k-1}=p, S_k=q, y\right) &= P\left(S_{k-1}=p, y_{prior}\right) P\left(S_k=q, y_{current} | S_{k-1}=p\right) \\ &\quad \times P\left(y_{future} | S_k=q\right) \end{aligned} \tag{3.6}$$

Then Eq. (3.6) can be rewritten as

$$P\left(S_{k-1}=p, S_k=q, y\right)=\alpha_{k-1}(p)\, \gamma_k(p, q) \beta_k(q) \tag{3.7}$$

where

$$\alpha_{k-1}(p)=P\left(S_{k-1}=p, y_{prior}\right) \tag{3.8}$$

$$\beta_k(q)=P\left(y_{future} | S_k=q\right) \tag{3.9}$$

$$\gamma_k(p, q)=P\left(S_k=q, y_{current} | S_{k-1}=p\right) \tag{3.10}$$

Then the a posteriori probabilities can be calculated from the following equation.

$$L\left(u_k | y\right)=\ln\left(\frac{\sum_{T^+} \alpha_{k-1}(p)\, \gamma_k(p, q) \beta_k(q)}{\sum_{T^-} \alpha_{k-1}(p)\, \gamma_k(p, q) \beta_k(q)}\right) \tag{3.11}$$

To calculate the a posteriori probabilities from Eq. (3.11), the values of α_k and β_k for every state in the trellis of the code must be computed and stored. The calculation of α_k is done by *Forward Recursion* calculations of the trellis of the code, while β_k are calculated by *Backward Recursion* of the trellis.

Figure 3.2 shows a portion of the trellis diagram explaining how the forward recursion process is done to calculate $\alpha_k(q)$ for each state in the trellis. The calculation of the values of $\alpha_k(q)$ is done using the following equation

$$\alpha_k(q)=\sum_{p=0}^{Q-1} \alpha_{k-1}(p)\, \gamma_k(p, q) \tag{3.12}$$

where Q is the number of states in the trellis of the code.

The initial values of α_k are as the following: if the encoder begins encoding in the zero state as usual, then $\alpha_0(0)=1$, and $\alpha_0(p)=0$, for $p=1,2,...,Q-1$. Else, if the initial state of the encoder is undetermined, then $\alpha_0(p)=\frac{1}{Q}$, for $p=0,1,...,Q-1$.

[2] To find the proof, see [2] and[10].

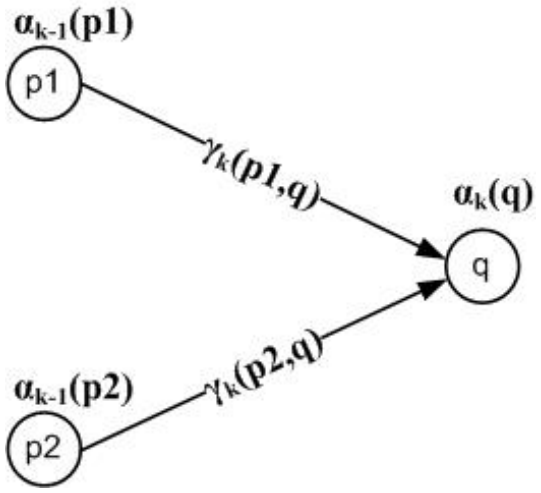

Figure 3.2: Calculation of α_k using the forward recursion process.

The backward recursion process is explained in Figure 3.3, where Eq. (3.13) is used to calculate the values of $\beta_k(p)$.

$$\beta_{k-1}(p) = \sum_{q=0}^{Q-1} \beta_k(q)\, \gamma_k(p,q) \tag{3.13}$$

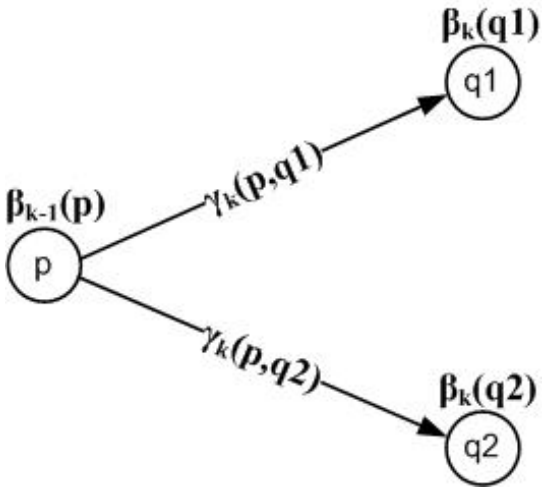

Figure 3.3: Calculation of β_k using the backward recursion process

The final values of β_k are chosen to be uniform, i.e. $\beta_N(q) = \frac{1}{Q}$, for $q = 0, 1, ..., Q-1$, if the encoder is not terminated in the zero state. But if the encoder is terminated in the zero state, then $\beta_N(0) = 1$, and $\beta_N(q) = 0$, for $q = 1, 2, ..., Q-1$.

The values of γ_k represent the *Branch Metrics* in the trellis of the code, where the value of $\gamma_k(p,q)$ is associated with the branch connecting state (p) at time $(k-1)$ to state (q) at time (k). These values of γ_k are calculated from the input sequence from the channel and from the apriori information depending on the model of the channel as shown in Eq. (3.14) assuming AWGN channel.

$$\gamma_k(p,q) = C.e^{(u_k L(u_k)/2)}.\exp\left(\frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl}\right) \tag{3.14}$$

where y_{kl} is the soft channel input, x_{kl} is the input bit associated with the branch (p,q), and n is the number of individual bits within the codeword of the code. C is a constant which is given by Eq. (3.15).

$$C = C_{L(u_k)}.C_{y_k}.C_{x_k} \tag{3.15}$$

where

$$C_{L(u_k)} = \left(\frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}}\right) \tag{3.16}$$

$$C_{y_k} = \frac{1}{\left(\sqrt{2\pi}\sigma\right)^n}\exp\left(-\frac{E_b}{2\sigma^2}\sum_{l=1}^{n} y_{kl}^2\right) \tag{3.17}$$

$$C_{x_k} = \exp\left(-\frac{E_b}{2\sigma^2}a^2 n\right) \tag{3.18}$$

where σ is the standard deviation of the noise, and a is the fading amplitude of the channel.[3]

3.3 Approximations to the MAP Algorithm

3.3.1 Representation of The MAP Algorithm in the Log Domain

The MAP algorithm is very complex for implementation in its original form proposed in [11], so many modifications take place on the MAP algorithm to decrease its complexity while maintaining nearly the same performance. Since the complexity of the MAP algorithm comes from the large number of multiplications needed to compute the values of α_k and β_k, this complexity can be reduced by implementing α_k and β_k in the log-domain where all the multiplications are transformed into additions.

Assume that

[3] All the equations and derivations in this section can be found in [2], [3], and [10].

$$\widetilde{\alpha}_k(p) = \ln\left(\alpha_k(p)\right)$$

$$\widetilde{\beta}_k(q) = \ln\left(\beta_k(q)\right)$$

$$\widetilde{\gamma}_k(p,q) = \ln\left(\gamma_k(p,q)\right)$$

From Eq. (3.12), the forward recursion process to calculate the values of $\widetilde{\alpha}_k$ can be written as

$$\widetilde{\alpha}_k(q) = \sum_{p=0}^{Q-1} e^{\left(\widetilde{\alpha}_{k-1}(p)+\widetilde{\gamma}_k(p,q)\right)} \tag{3.19}$$

In the same way, Eq. (3.13) can be written as

$$\widetilde{\beta}_{k-1}(p) = \sum_{q=0}^{Q-1} e^{\left(\widetilde{\beta}_k(q)+\widetilde{\gamma}_k(p,q)\right)} \tag{3.20}$$

The values of $\widetilde{\gamma}_k(p,q)$ can be computed as shown in Eq. (3.21).

$$\widetilde{\gamma}_k(p,q) = \ln(C) + \frac{1}{2}u_k L(u_k) + \frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl} \tag{3.21}$$

The a posteriori information in the log-domain can be written as shown in the following equation.

$$L(u_k|y) = \ln\left(\frac{\sum\limits_{T^+} e^{\left(\widetilde{\alpha}_{k-1}(p)+\widetilde{\gamma}_k(p,q)+\widetilde{\beta}_k(q)\right)}}{\sum\limits_{T^-} e^{\left(\widetilde{\alpha}_{k-1}(p)+\widetilde{\gamma}_k(p,q)+\widetilde{\beta}_k(q)\right)}}\right) \tag{3.22}$$

From mathematics, Eq. (3.23) shown below can be used to approximate Eqs. (3.19, 3.20, 3.22).

$$\ln\left(\sum_n e^{h_n}\right) = \overset{*}{\max_n}\left(h_n\right) \tag{3.23}$$

where $\overset{*}{\max}$ is an approximation function that will be explained in the following subsections. Some algorithms are introduced based on the MAP algorithm each with an approximation to the $\overset{*}{\max}$ function. Using the $\overset{*}{\max}$ function, Eqs. (3.19, 3.20, 3.22) can be written as the followings.

$$\widetilde{\alpha}_k(q) = \overset{*}{\max_{p\in[0,\ldots,Q-1]}}\left(\widetilde{\alpha}_{k-1}\left(p\right) + \widetilde{\gamma}_k(p,q)\right) \tag{3.24}$$

$$\widetilde{\beta}_{k-1}(p) = \underset{q\in[0,\ldots,Q-1]}{\overset{*}{\max}} \left(\widetilde{\beta}_k(q) + \widetilde{\gamma}_k(p,q)\right) \tag{3.25}$$

$$L(u_k|y) = \underset{T^+}{\overset{*}{\max}} \left(\widetilde{\alpha}_{k-1}(p) + \widetilde{\gamma}_k(p,q) + \widetilde{\beta}_k(q)\right) - \underset{T^-}{\overset{*}{\max}} \left(\widetilde{\alpha}_{k-1}(p) + \widetilde{\gamma}_k(p,q) + \widetilde{\beta}_k(q)\right) \tag{3.26}$$

3.3.2 The Max-Log-MAP Algorithm

This algorithm is the simplest approximation for the MAP algorithm, where the $\overset{*}{\max}$ is approximated as an ordinary max( ) function as shown in Eq. (3.27).

$$\overset{*}{\max}(h_1, h_2) \approx \max(h_1, h_2) \tag{3.27}$$

From Eq.(3.27), when the algorithm is doing the forward recursion calculations to find the values of $\widetilde{\alpha}_k(p)$, it selects only one path in the trellis which has the highest probability, and discards the other path. The same thing is done for the backward recursion to find the values of $\widetilde{\beta}_k(q)$.

3.3.3 The Log-MAP Algorithm

The $\overset{*}{\max}$ function can be approximated using the *Jacobian Logarithm* shown in Eq.(3.28)

$$\ln\left(e^{h_1} + e^{h_2}\right) = \max(h_1, h_2) + \ln\left(1 + e^{-|h_2 - h_1|}\right) \tag{3.28}$$

where the second term in Eq.(3.28) can be replaced with a correction term in the representation of the $\overset{*}{\max}$ function as shown in Eq. (3.29)

$$\overset{*}{\max}(h_1, h_2) = \max(h_1, h_2) + f_c\left(|h_2 - h_1|\right) \tag{3.29}$$

where $f_c\left(|h_2 - h_1|\right)$ is a nonlinear correction term that is only dependant on one variable which is the difference between the two arguments of the $\overset{*}{\max}$ function.

For practical implementation of the Log-MAP algorithm, the correction function can be implemented using a discrete points that are stored in a look-up table.

3.3.4 The Constant-Log-MAP Algorithm

The correction function $f_c\left(|h_2 - h_1|\right)$ can be implemented using only two constant values [20] as shown in Eq. (3.30).

$$f_c\left(|h_2 - h_1|\right) \approx \begin{cases} C & if\ |h_2 - h_1| \le T \\ 0 & if\ |h_2 - h_1| > T \end{cases} \tag{3.30}$$

For UMTS turbo codes, the best values for C and T are $C = 0.5$ and $T = 1.5$ [21].

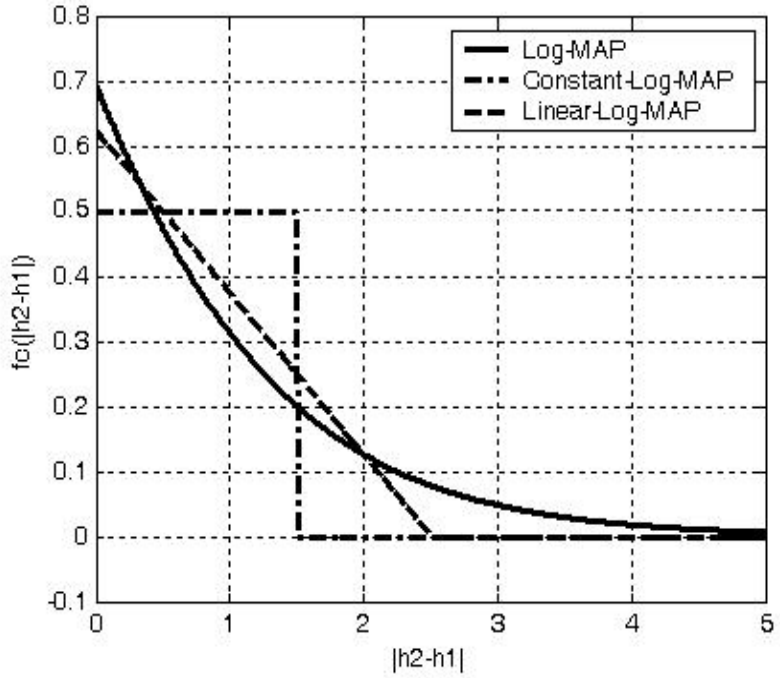

Figure 3.4: Correction function for the $\overset{*}{\max}$ function in Log-MAP, Constant-Log-MAP and Linear-Log-MAP algorithms[21].

3.3.5 The Linear-Log-MAP Algorithm

The correction function in the Jacobian logarithm can be approximated by a straight line as shown in Figure 3.4, where it can be written as shown in Eq. (3.31).

$$f_c\left(|h_2 - h_1|\right) \approx \begin{cases} a\left(|h_2 - h_1| - T\right) & if\ |h_2 - h_1| \leq T \\ 0 & if\ |h_2 - h_1| > T \end{cases} \tag{3.31}$$

where a and T are constants that can be found by minimizing the total squared error between the exact correction function and the linear approximation [21]. The author in [21] proved that the best values for the parameters in the linear approximation are $a = -0.24904$ and $T = 2.5068$.

3.4 The Soft Output Viterbi Algorithm (SOVA)

The Viterbi algorithm proposed in [7] cannot be used directly to decode turbo codes. A modification to the Viterbi algorithm was proposed in [12] to make it suitable for iterative decoding used in turbo codes. This new algorithm was called the *Soft-Output*

Viterbi Algorithm (SOVA). To make the SOVA algorithm suitable for iterative decoding, two modifications to the conventional Viterbi algorithms must be done. First, the path metrics used by the SOVA must be modified to use the apriori information, then the algorithm must provide soft output in the form of LLR, which is a measure for the reliability of the decisions.

The modified path metric for the SOVA algorithm is shown in Eq. (3.32), where the second term represents the apriori information provided by the other component decoder of the turbo decoder.

$$M\left(S_k^i\right) = M\left(S_{k-1}^j\right) + \frac{1}{2}u_k L\left(u_k\right) + \frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl} \tag{3.32}$$

where S_k^i is state i at time k, and S_{k-1}^j is the previous state of S_k^i.

Let $M^1\left(S_k^i\right)$ and $M^2\left(S_k^i\right)$ denote the two path metrics calculated for the state S_k^i which are resulted from the transitions from states S_{k-1}^m and S_{k-1}^n to state S_k^i respectively, and assume that $M^1\left(S_k^i\right) > M^2\left(S_k^i\right)$, then $M^1\left(S_k^i\right)$ is chosen as the path metric for the state S_k^i, and the path connecting between the states S_{k-1}^m and S_k^i is chosen to be the surviving path, while the other path is discarded.

Let's define the path metric difference at state S_k^i as

$$\Delta_k^i = M^1\left(S_k^i\right) - M^2\left(S_k^i\right) \tag{3.33}$$

where always $\Delta_k^i \geq 0$ since $M^1\left(S_k^i\right) > M^2\left(S_k^i\right)$.

The probability that the choice of the path metric $M^1\left(S_k^i\right)$ is correct is

$$P(correct) = \frac{e^{M^1\left(S_k^i\right)}}{e^{M^1\left(S_k^i\right)} + e^{M^2\left(S_k^i\right)}} = \frac{e^{\Delta_k^i}}{1 + e^{\Delta_k^i}} \tag{3.34}$$

Then the Log Likelihood Ratio is given as

$$L(correct) = \ln\left(\frac{P(correct)}{1 - P(correct)}\right) = \Delta_k^i \tag{3.35}$$

To explain the process of decoding used by the SOVA algorithm, portion of the trellis of an $[1, \frac{5}{7}]$ RSC encoder is shown in Figure 3.5, where the surviving path is shown in bold line. Also the discarded paths are shown where the dashed lines represent the transitions associated with -1 input bits, while the continuous lines represent the transitions associated with +1 input bits.

To calculate the LLR estimates which are the outputs of the SOVA decoder, a window of length δ is taken through the trellis, where δ is chosen to be greater than five times of the constraint length. Then the LLRs can be estimated as shown in Eq. (3.36).

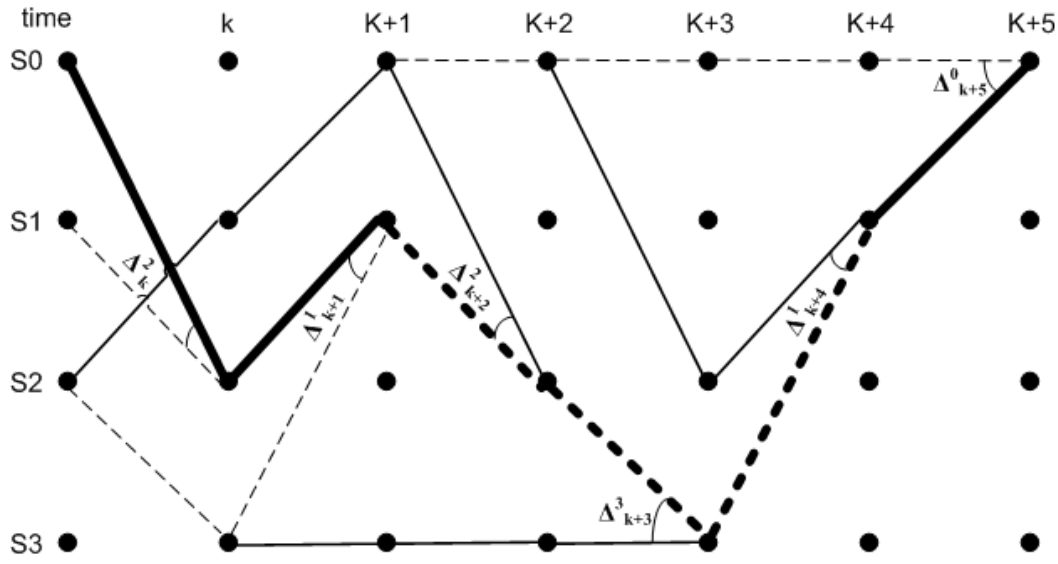

Figure 3.5: Portion of the trellis explainning the decoding process used by SOVA

$$L(u_k|y) \approx u_k \min_{\substack{l=0\ldots\delta-1 \\ u_k \neq u_k^l}} \Delta_{k+l}^i \tag{3.36}$$

where u_k^l is the input bit at time k associated with discarded path at time time $k+l$.[4]

3.5 The Modified SOVA Algorithm (MSOVA)

The conventional SOVA algorithm suffers from bad performance when compared to the MAP algorithm. The reason for the bad performance of the conventional SOVA algorithm is the existence of a bias in the extrinsic information produced by the decoder. This bias is a result of the high correlation between the intrinsic information at the input of the SOVA decoder and the extrinsic information at the output of the SOVA decoder [22], [23], [6].

The authors in [22] and [23] proposed to add a scaling factor to reduce the bias at the output of the SOVA decoder. The addition of this scaling factor improves the performance of the overall turbo decoder with a considerable amount.

Another modification to the SOVA was proposed in [6], which is called the *Modified SOVA* or *MSOVA*. The author in [6] modified the conventional SOVA algorithm such that the correlation between the extrinsic information and the intrinsic information is minimized.

[4] To find more details about SOVA, see [2], [3], and [10].

Consider the equation for the extrinsic information shown in Eq. (2.7), and let's define a new term $L_i(u_k)$ as the sum of the intrinsic information $L(u_k)$ and the soft channel output $L_c y_k$.

$$L_i(u_k) = L(u_k) + L_c y_k \tag{3.37}$$

Then, Eq.(2.7) can be written as shown in Eq. (3.38).

$$L_e(u_k) = L(u_k|y) - L_i(u_k) \tag{3.38}$$

In Eq. (3.38), it is assumed that there is no correlation between $L_e(u_k)$ and $L_i(u_k)$ which is not true, where the correlation between them is strong [6]. So this causes the poor performance of the SOVA decoder as indicated in [6].

Assuming that the channel is AWGN channel, the extrinsic information $L_e(u_k)$ follows a gaussian distribution [12] as well as $L_i(u_k)$. To modify the conventional SOVA algorithm, let's define $V(u_k|y)$ as the direct output of the conventional SOVA decoder which produces $V_e(u_k)$ as extrinsic information, such that Eq. (3.38) can be written as

$$V_e(u_k) \triangleq V(u_k|y) - L_i(u_k) \tag{3.39}$$

The a posteriori probability $L(u_k|y)$ can be modified as shown below

$$\begin{aligned} L(u_k|y) &= \ln\left(\frac{P\left(u_k=+1|V_e(u_k),L_i(u_k)\right)}{P\left(u_k=-1|V_e(u_k),L_i(u_k)\right)}\right) \\ &= \left(\frac{\frac{2m_e}{\sigma_e^2}-\rho\frac{2m_i}{\sigma_e\sigma_i}}{1-\rho^2}\right)V_e(u_k)+\left(\frac{\frac{2m_i}{\sigma_i^2}-\rho\frac{2m_e}{\sigma_e\sigma_i}}{1-\rho^2}\right)L_i(u_k) \\ &= aV_e(u_k)+bL_i(u_k) \end{aligned} \tag{3.40}$$

where ρ is the correlation coefficient between $V_e(u_k)$ and $L_i(u_k)$, m_e and σ_e^2 are the mean and variance of $V_e(u_k)$, m_i and σ_i^2 are the mean and variance of $L_i(u_k)$ respectively.

From the definition of $V_e(u_k)$ shown in Eq. (3.39), the LLR a posteriori probability can be written as

$$\begin{aligned} L(u_k|y) &= a\left(V(u_k|y)-L_i(u_k)\right)+bL_i(u_k) \\ &= aV(u_k|y)+(b-a)\,L_i(u_k) \end{aligned} \tag{3.41}$$

Then, the extrinsic information $L_e(u_k)$ can be obtained by substituting Eq. (3.41) in Eq. (3.38) as follows

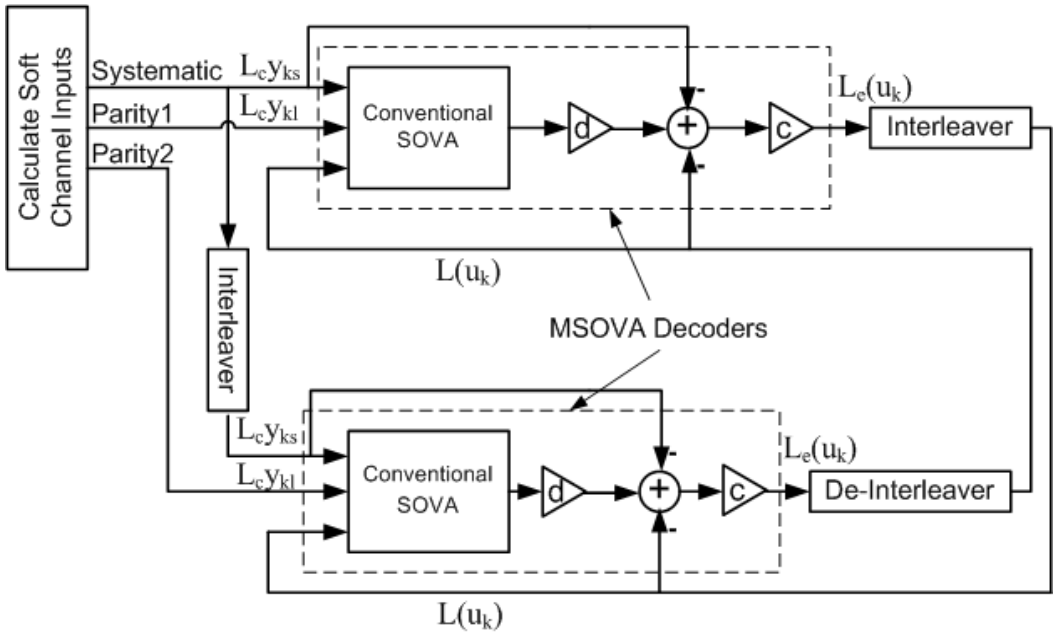

Figure 3.6: Modified turbo decoder with MSOVA component decoders

$$\begin{aligned} L_e(u_k) &= (a+1-b)\left(\frac{a}{a+1-b}V(u_k|y) - L_i(u_k)\right) \\ &= c \times (d \times V(u_k|y) - L_i(u_k)) \\ &= c \times (d \times V(u_k|y) - L(u_k) - L_c y_k) \end{aligned} \tag{3.42}$$

To calculate the values of the pair (c, d), the means and variances of $V_e(u_k)$ and $L_i(u_k)$, and also the correlation coefficient between $V_e(u_k)$ and $L_i(u_k)$ must be computed. This calculations will make the new MSOVA algorithm very complex. Fortunately, it is claimed by the author in [6] that for a fixed encoder structure and puncturing pattern, there is a fixed pair (c, d) than can be computed using computer simulations to have the best performance.

In this thesis, an extensive computer search is donen to find the (c, d) pair suitable for the MSOVA algorithm for the turbo codes used by the UMTS and cdma2000 standards. The results of these simulations is explained in details in chapter five.

3.6 Comparison of the Turbo Decoding Algorithms

From the previous sections, it can be seen that the MAP algorithm is the most complex algorithm among all known turbo decoding algorithms. The Max-Log-MAP algorithm

is the simplest version of the MAP algorithm. It is shown in [24] that the total number of operations per bit of the Max-Log-MAP algorithm is 30% less than the MAP algorithm. Also, it is shown in [24] that the total number of operations per bit of the Log-MAP algorithm is 3.6% less than the MAP algorithm.

The Max-Log-MAP algorithm is much less complex than the MAP algorithm since all the multiplications in the MAP are transformed to additions and selections using the max function. The complexity added to the Log-MAP algorithm over the Max-Log-MAP is that of the correction function used. The correction function used in the Log-MAP algorithm needs to be stored as separate values in a lookup table, so an additional memory for the lookup table is needed by the Log-MAP algorithm. The same thing can be said for the Constant-Log-MAP and the Linear-Log-MAP algorithms, where the complexity of the implementation of the correction function is added to the complexity of the Max-Log-MAP algorithm.

The Max-Log-MAP algorithm can be viewed as two SOVA decoders, one operated in the forward recursion, and the other in the backward recursion. So the complexity of the Max-Log-MAP algorithm is twice the complexity of the conventional SOVA algorithm [21].

Since the MSOVA algorithm differs from the conventional SOVA only by adding the two multipliers, its complexity is the same as that for the conventional SOVA except that there is a small additional complexity caused by the two scaling factors (c, d).

The memory requirements of the MAP algorithm and all its approximation versions are larger than that for the SOVA and the MSOVA algorithms. For example, the values of α_k in the forward recursion, and the values of β_k in the backward recursion must be stored for each node in the trellis. In the SOVA algorithm, The values of the path metric difference Δ_k^i for each node in the trellis must be stored.

One way to reduce the memory requirements used by the turbo decoding algorithm is by using a *sliding window* to decode the entire codeword. Another way is to decrease the number of bits used to represent each path metric. The last way may decrease the performance of the turbo decoder if the proper precision is not used.

The MAP algorithm has the best performance among all other algorithms. The Log-MAP algorithm performance is nearly the same as that for the MAP algorithm. For example, it is shown in [21] that for the UMTS turbo code with frame =5114 bits, and 14 iterations, the Max-Log-MAP algorithm needs about 0.4 dB for the E_b/N_0 higher than that required for the Log-MAP algorithm at a *Bit-Error-Rate* (BER) of 10^{-5} . The BER for the Linear-Log-MAP and the Constant-log-MAP is also shown in [21] to be near the BER of the Log-MAP algorithm with a better performance of the Linear-Log-MAP.

The performance comparison between the MAP, SOVA and MSOVA algorithms is left to be discussed in details in chapter five, where our simulation results in this thesis is used for the comparison.

Chapter 4

Case Study: UMTS and cdma2000 Turbo Codes

4.1 Introduction

In this chapter, the turbo codes used in the two third generation cellular standards, UMTS and cdma2000 are introduced. The encoder structure for the turbo codes in each standard is explained in details describing the trellis termination used and how the output codeword is multiplexed. Also the puncturing patterns used in the cdma2000 turbo codes are showed for all code rates. Finally, the interleavers structures for both standards are described.

4.2 UMTS Turbo Codes

4.2.1 Encoder Structure

The turbo encoder used in the UMTS standard is a parallel concatenation of two recursive systematic convolutional encoders with constraint length $K = 4$, and number of states $Q = 8$ as shown in Figure 4.1. The generator function of each RSC encoder is

$$G = [1, \frac{15}{13}]$$

where 15 is the feedforward polynomial in octal, and 13 is the feedback polynomial[1].

The state diagram of this RSC encoder is shown in Figure 4.2, where the dashed transition lines correspond to 0 input bits, and the continuous lines correspond to 1

[1] The generator polynomials can be obtained from the tap connections of the RSC encoder, or using the impulse response of the encoder.

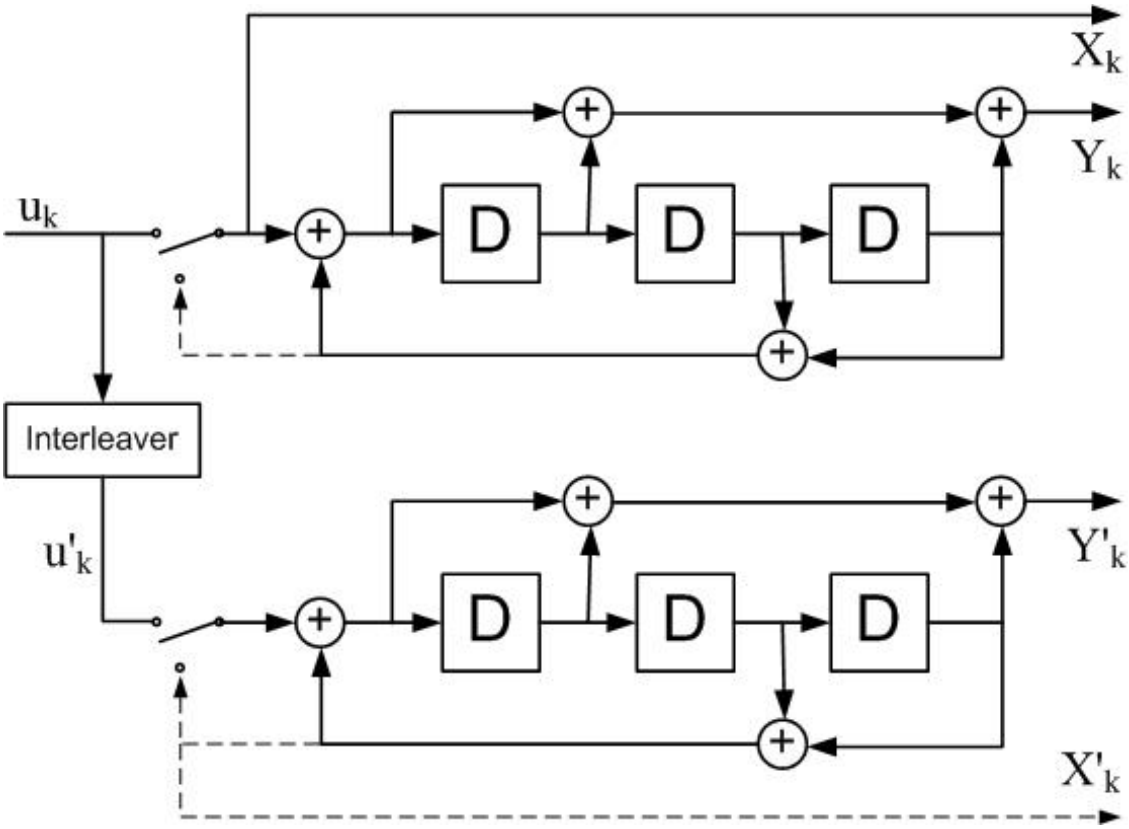

Figure 4.1: UMTS turbo encoder [25]

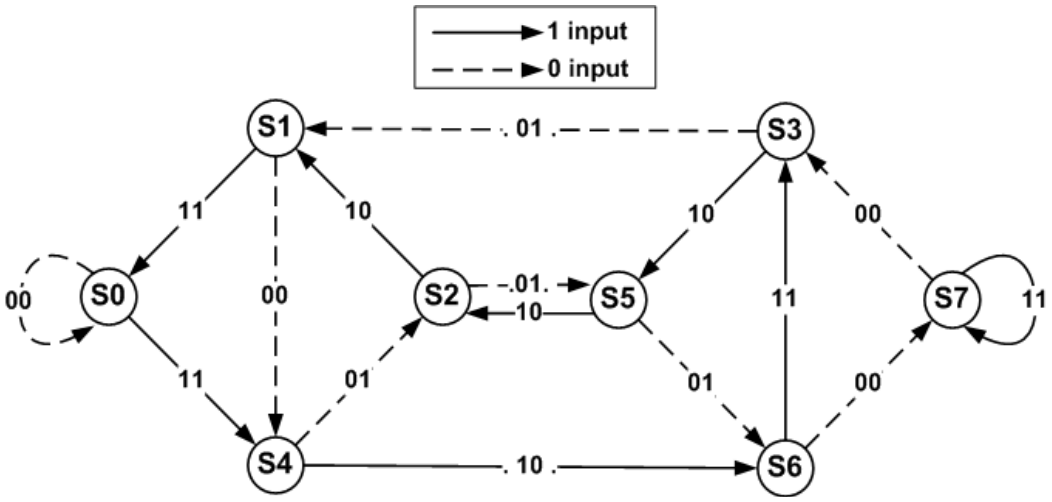

Figure 4.2: State diagram for the RSC encoder used in the UMTS turbo encoder

input bits. The outputs of the encoder are also shown attached to each transition line.

The input bits u_k for the encoder is arranged in blocks of frame sizes ranging from 40 to 5114 bits ($40 \leq N \leq 5114$) [25]. The outputs of the encoder are multiplexed serially to form the encoded codeword, where the systematic output of the first RSC encoder is put first in the coderword followed by the parity outputs of the first and the second RSC encoders respectively such that the codeword of the UMTS turbo code becomes as

$$..., X_{k-1}, Y_{k-1}, Y_{k-1}^{'}, X_k, Y_k, Y_k^{'}, X_{k+1}, Y_{k+1}, Y_{k+1}^{'}, ...$$

Because The output of the UMTS turbo encoder consists of three bits for every input bit, and there is no puncturing at the output of the encoder, then the code rate will be $r = \frac{1}{3}$.

4.2.2 Trellis Termination

The first and the second RSC encoders of the UMTS turbo encoder are both terminated with two separate tail bits[25]. The switch at the front of each encoder as shown in Figure 4.1 is connected to the input of the encoder when encoding the entire frame with size N. When the encoder finishes encoding the entire frame, the second RSC encoder is disabled while the switch at the front of the first encoder changes its state to connect to the feedback tap such that the encoder inputs will be zeros[2], and the encoder is terminated after 3 clocks. The outputs of the turbo encoder during this process are only the systematic and the parity bits of the first encoder.

When the first RSC encoder is terminated, it is disabled, and the same process takes place for the second RSC encoder. During this process, the outputs of the turbo encoder are the systematic and the parity outputs of only the second encoder. So, the overall turbo encoder outputs during the termination process are as shown below.

$$X_{N+1}, Y_{N+1}, X_{N+2}, Y_{N+2}, X_{N+3}, Y_{N+3}, X_{N+1}^{'}, Y_{N+1}^{'}, X_{N+2}^{'}, Y_{N+2}^{'}, X_{N+3}^{'}, Y_{N+3}^{'}$$

Since the previous 12-bits tail is added to the overall output of the turbo encoder, the exact code rate is modified as seen in Eq. (4.1).

$$r = \frac{N}{3N + 12} \tag{4.1}$$

[2] Because the XOR operation of the same input will yeild a zero output.

Frame Size N	No. of rows R	Inter-row permutation patterns
$(40 \leq N \leq 159)$	5	4,3,2,1,0
$(160 \leq N \leq 200)$ or $(481 \leq N \leq 530)$	10	9,8,7,6,5,4,3,2,1,0
$(2281 \leq N \leq 2480)$ or $(3161 \leq N \leq 3210)$	20	19,9,14,4,0,2,5,7,12,18,16,13,17,15,3,1,6,11,8,10
N = any other value	20	19,9,14,4,0,2,5,7,12,18,10,8,13,17,3,1,16,6,15,11

Table 4.1: Inter-row permutation patterns for UMTS interleaver

4.2.3 UMTS Turbo Interleaver

The UMTS turbo code interleaver uses an $R \times C$ rectangular matrix, where the input bits are written row by row, and read column by column after performing inter-row and intra-row permutations on the rectangular matrix. Since the size of the input bits may be less than the size of the matrix, a dummy bits is padded to the empty positions in the matrix. When the output needs to be read from the interleaver, the dummy bits added with padding to the matrix must be pruned from this output.

The number of rows of the rectangular matrix can be 5, 10 or 20 according to the value of the frame size N. The inter-row permutation patterns used in the UMTS interleaver is shown in Table 4.1 [25].

The intra-row permutations is more complex, where a prime number p and a corresponding primitive root v are chosen from a special table. The selected value of p is used to select the number of columns C, and the values of p and v are used to perform the intra-row permutations using a special algorithm which is explained in details in [25].

To read out the interleaved sequence from the permuted rectangular matrix, it must begin from first element, and read column by column while pruning the dummy bits padded in the matrix at the write process.

4.3 The cdma2000 Turbo Codes

4.3.1 Encoder Structure

The turbo encoder used by the cdma2000 standard is shown in Figure 4.3. The RSC constituent encoders used in this turbo encoder is the same as that used in the UMTS turbo encoder, except that it has a second parity output. Since there are two parity outputs in each RSC encoder, the basic code rate for the cdma2000 turbo code is $r = \frac{1}{5}$.

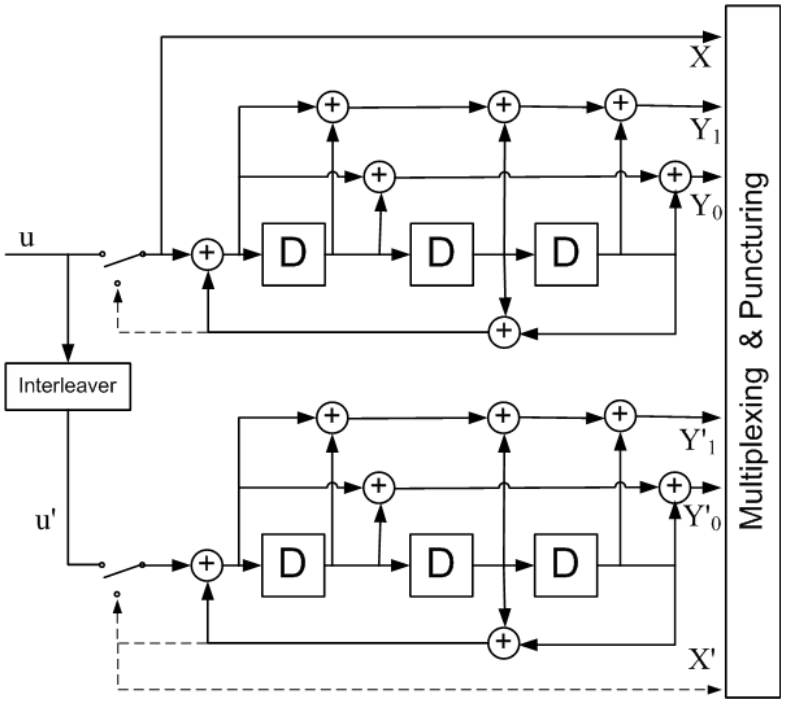

Figure 4.3: cdma2000 turbo encoder [26]

The generator function for the RSC encoders is

$$G = [1, \frac{15}{13}, \frac{17}{13}]$$

where 13 is the feedback polynomial generator in octal, 15 is the generator polynomial for the first parity output, and 17 is the generator polynomial for the second parity output.

Code rates of $\frac{1}{4}$, $\frac{1}{3}$, and $\frac{1}{2}$ are also used in cdma2000 turbo codes as well as the $\frac{1}{5}$ code rate. These code rates can be achieved by puncturing the parity outputs of the first and the second RSC encoders. Puncturing patterns for the cdma2000 turbo codes are discussed in details in the following subsection.

The trellis termination of the cdma2000 turbo encoder is the same as that for the UMTS turbo encoder, where two separate tail bits are used to terminate each encoder. The only difference between these two trellis termination methods is in the puncturing of the tail bits if code rates lower than $\frac{1}{5}$ are used[26].

The Frame size of the input bits to the cdma2000 turbo encoder can take only one of the following sizes; 186, 378, 402, 570, 762, 786, 1146, 1530, 1554, 2298, 3066, 3090, 4602, 4626, 6138, 6162, 9210, 9234, 12282, 12306, 15378, 18450, 20730.

Output	$r = \frac{1}{2}$	$r = \frac{1}{3}$	$r = \frac{1}{4}$	$r = \frac{1}{5}$
X	11	11	11	11
Y_0	10	11	11	11
Y_1	00	00	10	11
$X^{'}$	00	00	00	00
$Y_0^{'}$	01	11	01	11
$Y_1^{'}$	00	00	11	11

Table 4.2: Puncturing Patterns for the cdma2000 turbo codes

4.3.2 Puncturing Patterns

As stated in the previous subsection, code rates of $\frac{1}{4}$, $\frac{1}{3}$, and $\frac{1}{2}$ in addition to the basic code rate of $\frac{1}{5}$ can be achieved in the cdma2000 turbo codes using puncturing process. Puncturing is done by deleting some of the parity outputs of the turbo encoder in a special manner, such that variable code rate is possible.

The puncturing patterns used in the cdma2000 turbo codes for every code rate are shown in Table 4.2 [26], where this table is read first from top to bottom and then from left to right.

In Table 4.2, the 0 entry means that the corresponding output bit must be removed from the overall turbo code output, while the 1 entry means that the corresponding output bit must remain in the turbo code output.

4.3.3 cdma2000 Turbo Interleaver

The cdma2000 interleaver consists of an array with size of the same as the input bits frame size. The input bits are written in the array starting from the first position of the array. Then the output bits are read out from a sequence of addresses in the array defined by the algorithm described in the block diagram shown in Figure 4.4 [26].

The calculations of the output addresses are done using an $(n + 5)$ bit counter, where n are chosen to be the minimum value such that $2^{(n+5)} \geq N$. The algorithm shown in Figure 4.3 is equivalent to writing the values of the counter to an $2^5 \times 2^n$ matrix, where the inter-rows permutations follow the bit reversal rule, while the intra-row permutations are done according to the following rule

$$x(i+1) = (x(i) + c) \bmod 2^n$$

and

$$x(0) = c$$

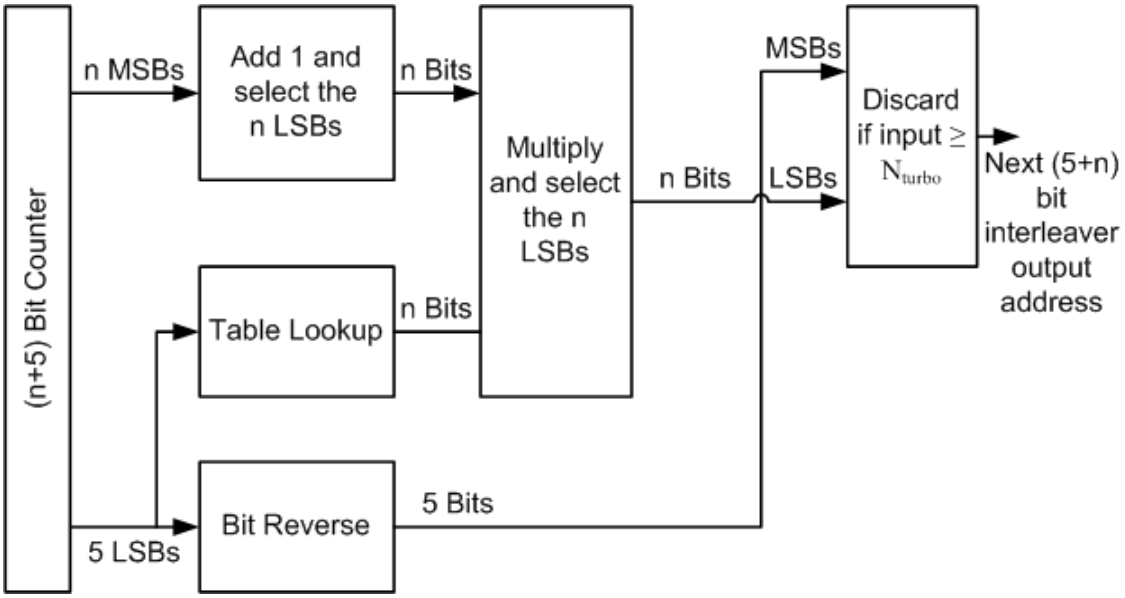

Figure 4.4: Calculation of the output addresses for the cdma2000 turbo interleaver [26]

where c can be found from a lookup table that can be found in [26].

Chapter 5

Simulation Results

5.1 Introduction

In this chapter, the simulation results for the UMTS and the cdma2000 turbo codes are shown in details for different decoding algorithms, concentrating on the MSOVA algorithm. The results are shown for various inerleavers, frame sizes, code rates, number of iterations and frame sizes. These results are analyzed while comparing the performance for each case.

All the results are shown in the form of the Bit-Error Rate (BER) against the signal to noise ratio represented in the form of $\frac{E_b}{N_0}$, where E_b is the per bit transmitted energy, and N_0 is the one sided power spectral density of the code. The simulations in this thesis assume AWGN channel with QPSK modulation.

5.2 Simulation Setup

5.2.1 Basic Processing

All the simulations used in this thesis are done using ANSI C language under the Linux environment. The simulation setup consists basically of an encoder, decoders and interleavers as described in Figures 2.8, 3.6, 4.1 and 4.3. The input messages are streams of binary bits that are generated randomly with the built-in Linux pseudo-random generator. The input messages are encoded with the turbo encoder which consists of two identical RSC encoders. The structures of the RSC encoders used for both, the UMTS and the cdma2000 turbo codes are described using tables that contain a detailed description of the state diagram of the RSC encoder.

The interleaver and de-interleaver are saved in two arrays that contain the indices corresponding to the permutation pattern of the interleaver.

When the input massage is encoded by the turbo encoder, it is mapped to the corresponding $\{-1, +1\}$ symbols which represent the QPSK modulation used in our simulation, and then, it is summed with an AWGN input that corresponds to the given N_0. The AWGN noise is generated using a model that is described in the following section.

The output of the channel then is passed to the turbo decoder which is described in Figures 2.8 and 3.6. The decoding process is done iteratively for a fixed number of iterations, then decoded output bits are taken by investigating the sign of L_{out} which is defined in Eq. (5.1).

$$L_{out} = L_{12} + L_{21} + L_c y_k \tag{5.1}$$

where L_{12} is the extrinsic information passed from the first decoder to the second decoder, and L_{21} is the extrinsic information passed from the second decoder to the first decoder after de-interleaving.

The number of errors then is calculated by comparing the decoded output to the input message. Then the procedure is repeated for a large number of frames until a considerable number of errors is reached.

The BER is then calculated as

$$BER = \frac{NE_{total}}{N_f \times size}$$

where NE_{total} is the total number of errors, N_f is the number of frames, and $size$ is the frame size.

To find the values of the pair (c, d) for the MSOVA algorithm in the UMTS and cdma2000 turbo codes, an extensive search for these parameters was done, assuming that their values are within the period $[0.7, 0.9]$ which is shown by the author in [6].

5.2.2 Channel Model

The channel model used in these simulations uses the Box-Muller method to calculate the noise. In Box-Muller method, suppose that two pseudo-random numbers are generated, say u_1 and u_2, then the noise is calculated using the following equations[27].

$$f = \sqrt{-2 \ln (u_1)}$$

$$g_1 = \sin (2\pi u_1)$$

$$g_2 = \cos (2\pi u_1)$$

Algorithm	$\frac{E_b}{N_0}$ at BER=10^{-5}	$\frac{E_b}{N_0}$ at BER=10^{-6}
MAP	0.875	1.12
MSOVA	1.07	1.26
SOVA	1.37	1.67

Table 5.1: Minimum value of SNR required for the desired BER in UMTS TC using different decoding algorithms with 1280-bits frame.

$$x_1 = f \times g_1 \tag{5.2}$$

$$x_2 = f \times g_2 \tag{5.3}$$

The generated AWGN noise can then be calculated by multiplying the desired standard deviation σ of the noise by either x_1 or x_2 found in Eqs. (5.2), (5.3).

5.3 Results for the UMTS turbo codes

Simulation results show that the best values of the (c, d) pair of the MSOVA algorithm for the UMTS turbo codes are $(0.74, 0.82)$. The results obtained for the UMTS turbo codes are shown in the following subsections, explaining the performance of the MSOVA algorithm compared to the MAP and conventional SOVA algorithms.

5.3.1 Effect of the Decoding Algorithm

The simulation results for the performance of the BER of the UMTS turbo code with different decoding algorithms are shown in Figure 5.1. From Figure 5.1, it can be seen that there is a large improvement in the BER using the MSOVA algorithm over that of the conventional SOVA algorithm.

Table 5.1 shows a comparison between the minimum required $\frac{E_b}{N_0}$ to achieve a desired BER using the MAP, MSOVA and SOVA decoding algorithms with 1280-bits frame and 10 decoding iterations. Using the MSOVA algorithm only requires about (0.2 dB) for the $\frac{E_b}{N_0}$ larger than that when using the MAP to achieve a BER of $\left(10^{-5}\right)$ compared to about (0.5 dB) required if the SOVA is used. If the desired BER is $\left(10^{-6}\right)$, then the MSOVA algorithm requires only about (0.14 dB) larger than the MAP, compared to about (0.55 dB) required by the SOVA.

It can be seen from Figure 5.1 that the BER performance using the MAP and the MSOVA algorithms become very close at SNR larger than (1.4 dB), which is very important result if the complexity of both algorithms is taken into consideration.

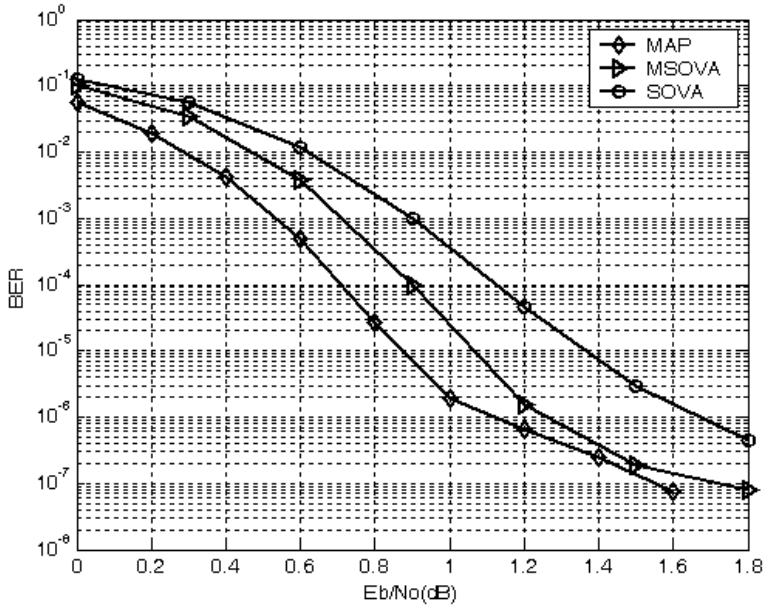

Figure 5.1: BER performance of the UMTS TC using different decoding algorithms with frame size =1280 bits and 10 iterations.

5.3.2 Effect of Frame Size

A comparison between the BER of the UMTS turbo codes using the MSOVA decoding algorithm with different frame sizes based upon our simulations is shown in Figure 5.2. From this figure, it is clear that a large improvement in the BER performance is achieved if the frame size is increased. for example, using a frame size of (40 bits) requires about (3.2 dB) for the $\frac{E_b}{N_0}$ larger than that when using a frame size of (5114 bits) to achieve a BER of $\left(10^{-5}\right)$. A comparison between the minimum required $\frac{E_b}{N_0}$ to achieve a desired BER using the MSOVA decoding algorithm with different frame sizes are shown in Table 5.2.

So to have a better performance, a larger frame size must be used. But using larger frame size will increase the memory requirements for the decoder, and increase the decoding latency. So a compromise must take place between the performance from one side, and the memory requirements and latency from the other side.

Frame Size	$\frac{E_b}{N_0}$ at BER=10^{-5}	$\frac{E_b}{N_0}$ at BER=10^{-6}
40	3.8	5.15
256	1.95	2.23
512	1.49	1.8
1280	1.07	1.26
5114	0.61	0.85

Table 5.2: Minimum value of SNR required for the desired BER in UMTS TC using MSOVA with different frame sizes.

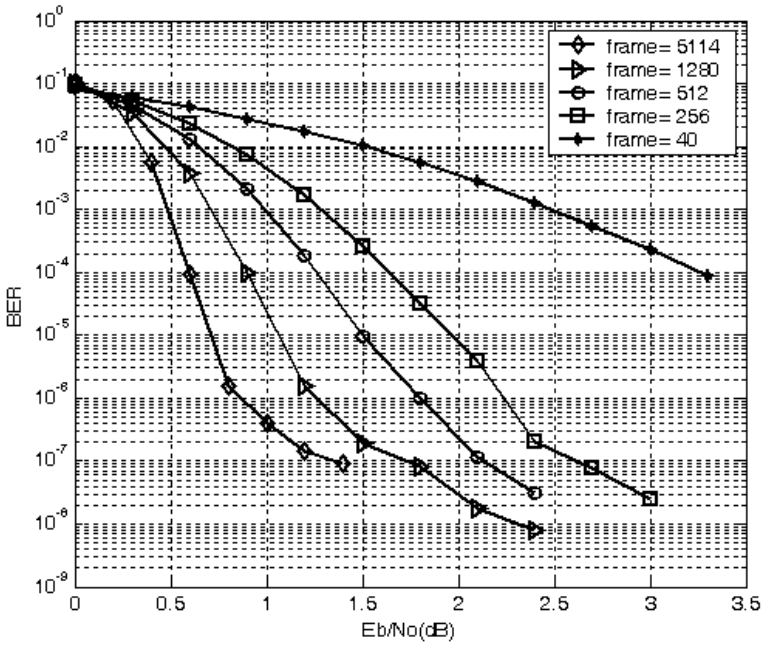

Figure 5.2: BER performance of the UMTS TC using MSOVA decoding algorithm with different frame sizes and with 10 iterations.

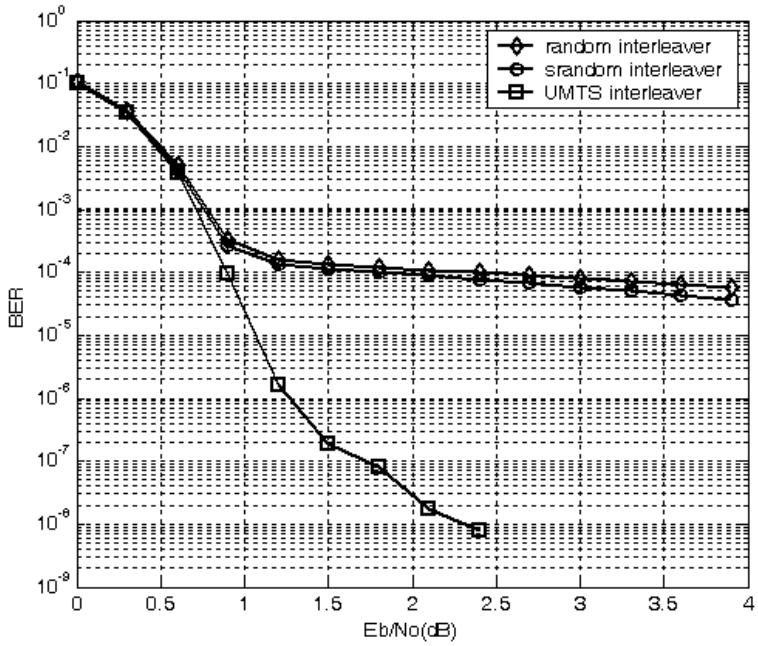

Figure 5.3: BER performance of the UMTS TC using MSOVA decoding algorithm with different interleaver types at 1280-bits frame and 10 iterations.

5.3.3 Effect of the Interleaver Type

As indicated in chapter two, the interleaver has a great effect on the turbo codes performance. In this research, the effects of three types of interleavers are investigated on the UMTS turbo codes using the MSOVA decoding algorithm. Figure 5.3 shows the BER performance of the UMTS turbo codes with a random, s-random and the UMTS interleavers with frame size of 1280 bits. From Figure 5.3, it is seen that the performance using the random and s-random interleavers suffers from an error floor after (1 dB) of $\frac{E_b}{N_0}$ with a better performance of the s-random interleaver.

The performance using the UMTS interleaver is shown to have no error floor with a very good BER. The reason for that good performance is because of the good weight distribution of the UMTS turbo encoder when the UMTS interleaver is used.

5.3.4 Effect of the Number of Decoding Iterations

The number of decoding iterations has a large effect on the decoding performance. Figure 5.4 shows the BER performance of the UMTS turbo codes using MSOVA algorithm with 1280-bits frame.

It can be seen from Figure 5.4 that for the first few iterations, there is a significant

No. of Iterations	$\frac{E_b}{N_0}$ at BER=10^{-5}
3	1.76
6	1.18
10	1.07
14	1.03

Table 5.3: Minimum value of SNR required for the desired BER in UMTS TC using MSOVA with different number of iterations with 1280-bits frame

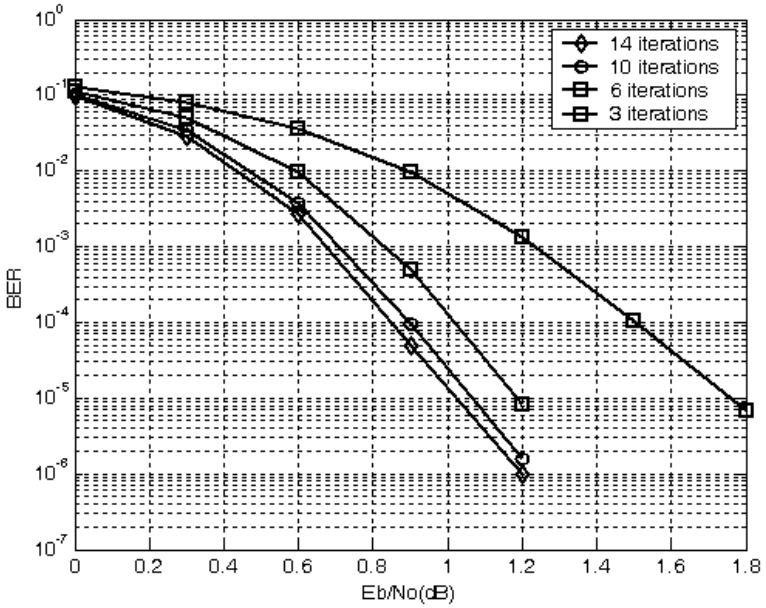

Figure 5.4: BER performance of the UMTS TC using MSOVA decoding algorithm with different number of iterations.

improvement in the BER performance, while when the iteration number is increased after certain value, the improvements are very small. From Table 5.3, the gain in the value of $\frac{E_b}{N_0}$ at BER of $\left(10^{-5}\right)$ obtained from 14 iteration over 10 iteration is only about (0.04 dB), while the gain of using 10 iterations over 3 iterations is about (0.7 dB).

Since increasing the number of iterations over 10 iterations didn't give a considerable gain in the BER performance, and because increasing the number of iterations will increase the latency of the decoding process, the number of iterations is chosen to be 10 iterations in all the simulations included in this research.

Code Rate	(c, d) pair
$\frac{1}{2}$	$(0.73, 0.82)$
$\frac{1}{3}$	$(0.74, 0.82)$
$\frac{1}{4}$	$(0.73, 0.83)$
$\frac{1}{5}$	$(0.72, 0.81)$

Table 5.4: The best values for the MSOVA pair (c,d) used in the cdma2000 TC for all code rates

	Decoding Algorithm		
Code Rate	MAP	MSOVA	SOVA
$\frac{1}{2}$	1.48	1.63	2.24
$\frac{1}{3}$	0.8	0.96	1.7
$\frac{1}{4}$	0.55	0.71	1.54
$\frac{1}{5}$	0.37	0.52	1.36

Table 5.5: Minimum value of SNR required for the desired BER in cdma2000 TC using different decoding algorithms with 1530-bits frame.

5.4 Results for the cdma2000 turbo codes

All the simulations included in this section use the encoder structure shown in Figure 4.3, with the cdma2000 inerleaver described in Figure 4.4, and use the puncturing patterns specified in Table 4.2 with $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$ and $\frac{1}{5}$ code rates. Fixed number of decoding iterations is assumed with 10 iterations.

Through extensive computer search, it is found that the values of the c scaling factor for the MSOVA decoding algorithm that give the best BER performance are in the range $[0.7 - 0.75]$ for all code rates, and the values of the d scaling factor are in the range $[0.8 - 0.85]$. The exact values of (c, d) are shown in Table 5.4.

5.4.1 Effect of the Decoding Algorithm

BER performance plots of the cdma2000 turbo code with MAP, MSOVA and SOVA decoding algorithms for $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$ and $\frac{1}{5}$ code rates and with 1530-bits frame are shown in Figures 5.5, 5.6, 5.7 and 5.8 respectively. Table 5.5 shows a comparison between the minimum required $\frac{E_b}{N_0}$ to achieve a BER of $\left(10^{-5}\right)$ using the MAP, MSOVA and SOVA decoding algorithms with different code rates and with 1530-bits frame.

From Table 5.5, it can be seen that using MSOVA algorithm for cdma2000 turbo code has a large improvement on the BER performance over the SOVA algorithm.

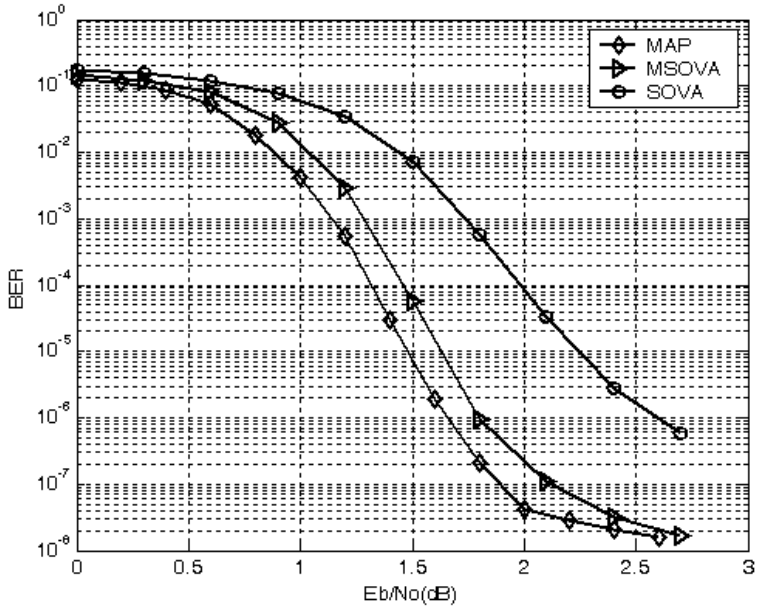

Figure 5.5: BER performance of the cdma2000 TC using different decoding algorithms with $\frac{1}{2}$ code rate and 1530-bits frame size.

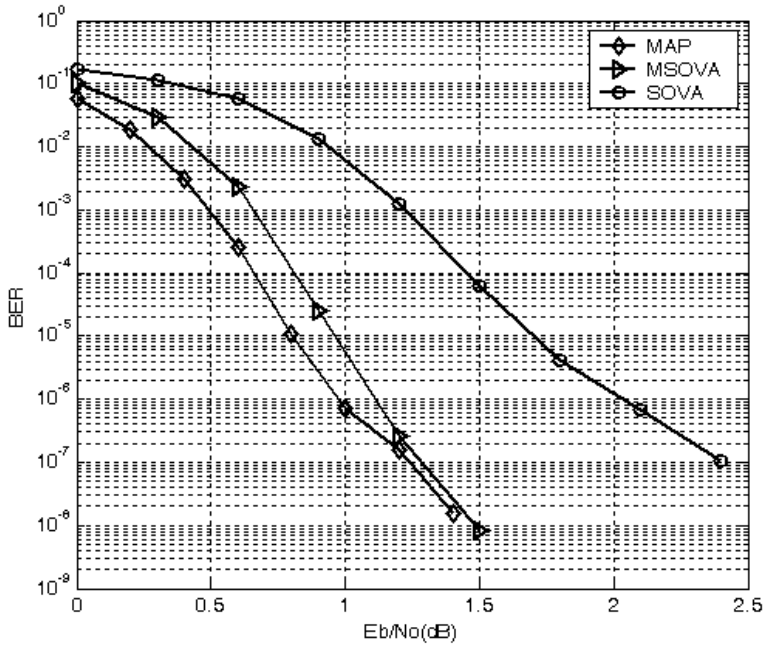

Figure 5.6: BER performance of the cdma2000 TC using different decoding algorithms with $\frac{1}{3}$ code rate and 1530-bits frame size.

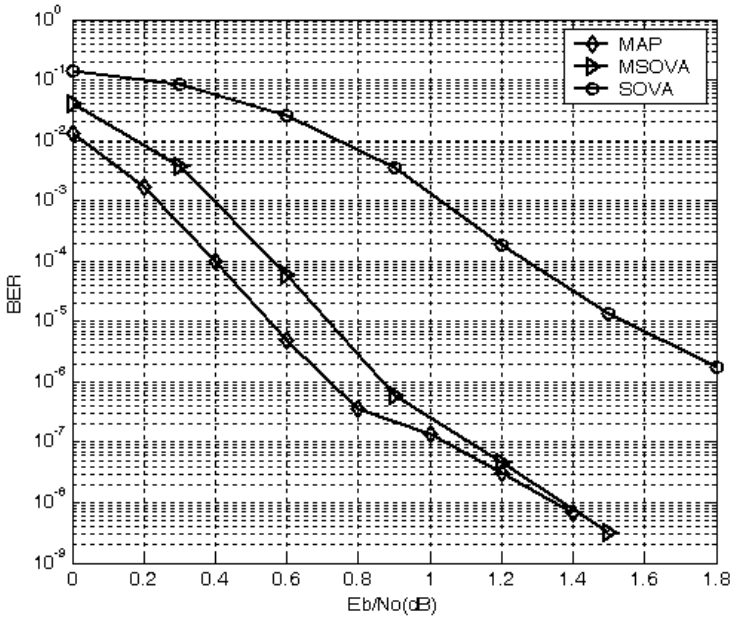

Figure 5.7: BER performance of the cdma2000 TC using different decoding algorithms with $\frac{1}{4}$ code rate and 1530-bits frame size.

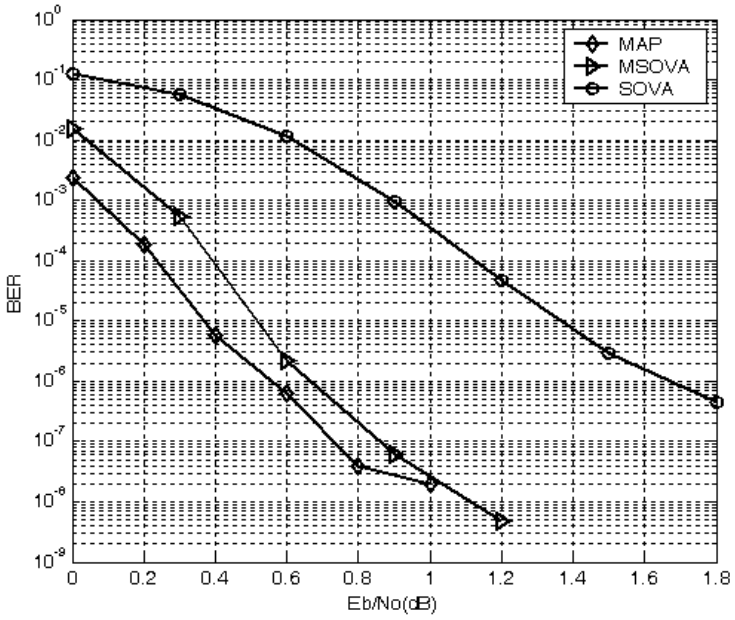

Figure 5.8: BER performance of the cdma2000 TC using different decoding algorithms with $\frac{1}{5}$ code rate and 1530-bits frame size.

Frame Size	$\frac{E_b}{N_0}$ at BER=10^{-5}	$\frac{E_b}{N_0}$ at BER=10^{-6}
186	1.8	2.4
570	0.975	1.18
1530	0.52	0.67
4602	0.24	0.36
9210	0.17	0.24

Table 5.6: Minimum value of SNR required for the desired BER in cdma2000 TC using MSOVA with different frame sizes and with rate 1/5.

For example, the BER performance of the cdma2000 turbo code using the MSOVA decoding algorithm requires only about (0.15 dB) over the MAP algorithm at $\left(10^{-5}\right)$, while the SOVA algorithm requires a bout (1 dB) over the MAP algorithm to have the same BER.

It can be seen also that the BER performance for the MSOVA approaches that of the MAP for values larger than (2.5 dB) with code rate of $\frac{1}{2}$, and for values larger than (1.2 dB) with code rate of $\frac{1}{3}$, and for values larger than (1 dB) with codes rates of $\frac{1}{4}$ and $\frac{1}{5}$.

5.4.2 Effect of Frame Size

Simulation results for various frame sizes of the cdma2000 turbo code using MSOVA algorithm with $\frac{1}{5}$ code rate are plotted in Figure 5.9, and these results are analyzed using Table 5.6.

From Table 5.6, it is clear that a large improvement on the performance is achieved when increasing the frame size. For example, when using a frame size of 4602 bits, the algorithm requires about (1.56 dB) to have BER of $\left(10^{-5}\right)$ less than that required when using a frame size of 40 bits, while it requires about (1.28 dB) when using 1530 bits frame less than that when using 40 bits frame.

So it is desired from simulation results that to have a good performance, the frame size must be as large as possible. But since increasing the frame size will increase the memory requirements and the decoding latency as indicated in the previous section, a compromise between the frame size and the memory size must be considered. This result is found in most books talking about turbo codes.

5.4.3 Effect of Code Rate

The simulation results for various code rates using MSOVA algorithm with frame sizes of 186 and 1530 bits are plotted in Figures 5.10 and 5.11 respectively.

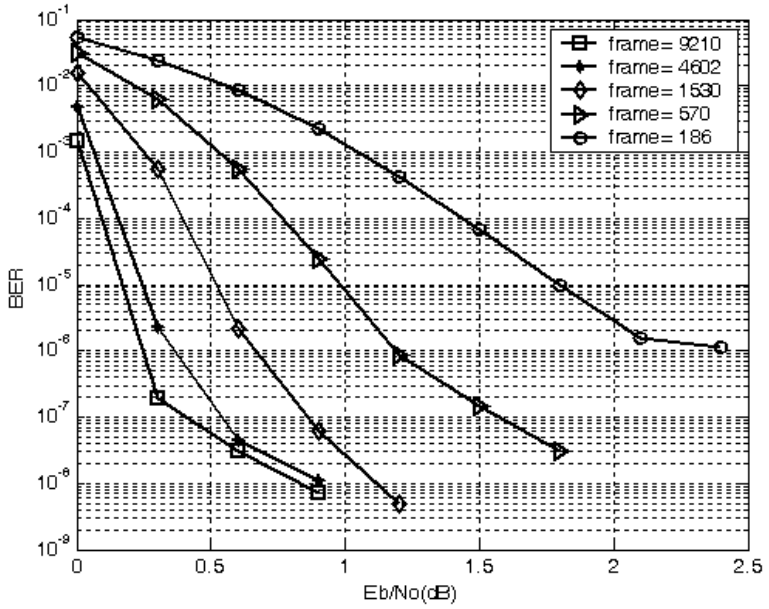

Figure 5.9: BER performance of the cdma2000 TC using MSOVA decoding algorithm with different frame sizes and with $\frac{1}{5}$ code rate.

Table 5.7 shows the minimum required $\frac{E_b}{N_0}$ to have a BER of $\left(10^{-5}\right)$ with different code rates for both; 186 and 1530 bits frame sizes. Using Figures 5.10 and 5.11, and Table 5.7, it can be concluded that increasing the code rate will give a great improvement in the BER performance. This result coincides with the literature of coding theory. For example, using code rate of $\frac{1}{5}$ gives a gain of about (1.1 dB) at BER of $\left(10^{-5}\right)$ over the use of $\frac{1}{2}$ code rate at 1530 bits frame size, while code rate of $\frac{1}{3}$ gives a gain of about (0.67 dB) over that of $\frac{1}{2}$ code rate at 1530 bits frame size.

	Frame size	
Code Rate	**186 bits**	**1530 bits**
$\frac{1}{2}$	2.86	1.63
$\frac{1}{3}$	2.2	0.96
$\frac{1}{4}$	1.96	0.71
$\frac{1}{5}$	1.8	0.52

Table 5.7: Minimum value of SNR required for the desired BER in cdma2000 TC using MSOVA with different code rates.

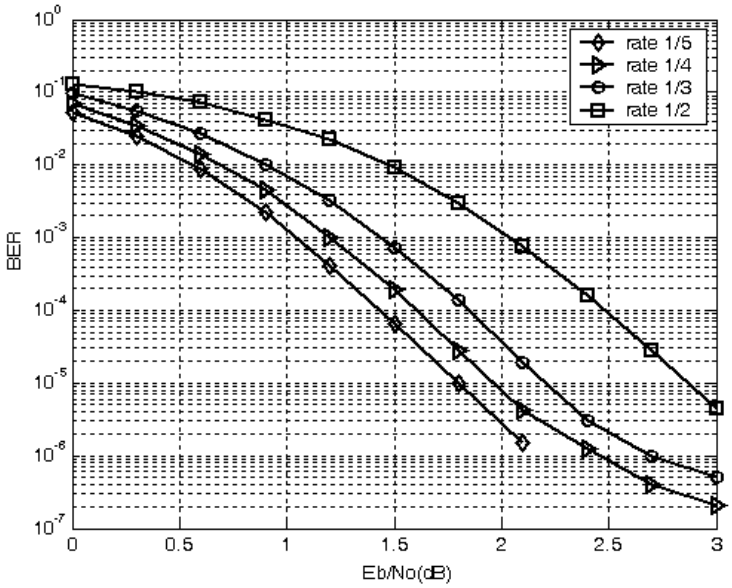

Figure 5.10: BER performance of the cdma2000 TC using MSOVA decoding algorithm with different code rates and with 186-bits frame.

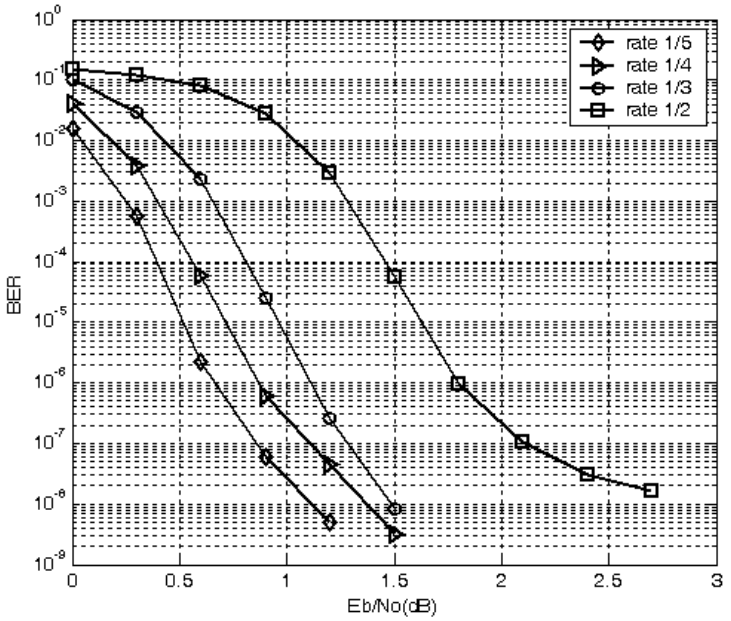

Figure 5.11: BER performance of the cdma2000 TC using MSOVA decoding algorithm with different code rates and with 1530-bits frame.

Chapter 6

Conclusions and Future Work

6.1 Summary of Work

The Basics of turbo coding were introduced in this thesis, including encoding, decoding, interleaving, termination and puncturing. Decoding algorithms were investigated in details, while explaining the most known turbo decoding algorithms. Based upon literature, comparisons of the complexity, performance and memory requirements were also introduced.

The decoding algorithms introduced in this thesis were the MAP, Log-MAP, Max-Log-MAP, Constant-Log-MAP, Linear-Log-MAP, SOVA and MSOVA. The MSOVA algorithm proposed in [6] was considered to be the core of our research.

UMTS and cdma2000 turbo codes were studied using the MSOVA, MAP and SOVA decoding algorithms. Comparisons between the performance of the three decoding algorithms based upon computer simulations were introduced. An extensive computer search was done to find the best scaling factors (c, d) used in the MSOVA algorithm suitable for the UMTS and cdma2000 turbo codes.

6.2 Conclusions

The performance of MSOVA algorithm used in the UMTS and cdma2000 turbo codes is excellent compared to the performance of the SOVA algorithm, taking into consideration that the complexity of the MSOVA is similar to the complexity of the SOVA algorithm, except that it has a small additional complexity caused by the two scaling factors (c, d).

Although the MAP algorithm has a little bit better performance than the MSOVA algorithm, it has a very complex structure such that the hardware and software requirements for its implementation are very large, whereas the requirements for the

MSOVA algorithm are relatively small.

Since the complexity of the simplest version of the MAP algorithm (The Max-Log-MAP) is about twice the complexity of the MSOVA, The decoding speed of the MSOVA algorithm is faster than that of all the versions of the MAP algorithm. So the throughput of the MSOVA decoder is larger than the throughput of the MAP decoder.

The best values of the c scaling factor for the MSOVA algorithm suitable for the UMTS and cdma2000 turbo codes were found to be in the range $[0.7 - 0.75]$, and the values for the d scaling factor were in the range $[0.8 - 0.85]$.

Because of the previous results, it is recommended to adopt the MSOVA decoding algorithm for the UMTS and cdma2000 turbo codes.

6.3 Future Work

Since all the simulations in this thesis were done for only the AWGN channel model, other channel models such as flat fading channels can be used for further investigation of the performance of the MSOVA algorithm used for UMTS and cdma2000 turbo codes.

The hardware implementation of the MSOVA algorithm will give a better view on its performance. So MSOVA can be implemented on FPGA to test its performance in practical systems.

Bibliography

[1] C. E. Shannon, “A mathematical theory of communication,” *Bell Systems Technical Journal*, Vol. 27, pp. 379–423, 623–656, July, October 1948.

[2] T. K. Moon, "Error correction coding, mathematical methods and algorithms," *John Wiley and Sons, Inc.*, 2005.

[3] R. H. Morelos-Zaragoza, "The art of error correcting coding," *John Wiley and Sons, Ltd.*, 2002.

[4] C. Berrou, A. Glavieux, and P. Thitimasjshima, “Near shannon limit error-correcting coding and decoding: turbo-codes,” *Proceedings of the IEEE International Conference on Communications* , pp. 1064–70, May 1993.

[5] S. Benedetto, D. Divsalar, D. Montorsi, and F. Pollara, “Serial concatenation of interleaved codes, performance analysis, design, and iterative decoding,” *IEEE Trans. on Information Theory*, Vol. 44, no. 3, pp. 909–26, May 1998 .

[6] C. Xiu Huang, and A. Ghrayeb, "A simple remedy for the exaggerated extrinsic information produced by the SOVA algorithm," *IEEE Trans. on Wireless Communications*, Vol. 5, no. 5, pp. 996-1002, May 2006.

[7] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Information Theory*, Vol. IT-13, no. 2, pp. 260-269, April 1967.

[8] G. D. Forney, Jr., "The Viterbi algorithm," *IEEE Trans. on Information Theory*, vol. IT-61, no. 3, pp. 268-278, March 1973.

[9] L. C. Perez, J. Seghers, and D. J. Costello Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. on Information Theory*, Vol. 42, no. 6, pp. 1698–1709, November 1996.

[10] L. Hanzo, T.H. Liew, and B.L. Yeap, "Turbo coding, turbo equalization and space-time coding for transmission over fading channels ," *John Wiley and Sons Ltd.*, 2002.

[11] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Information Theory*, vol. 1T-20, pp. 284-287, March 1974.

[12] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proceedings of IEEE GLOBECOM,* pp. 1680-1686, November 1989.

[13] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and Gabriele Nebe,"Interleaver design for turbo codes," *IEEE on Selected Areas in Comm.*, Vol. 19, no. 5, pp. 831-837, May 2001.

[14] J. Hokfelt, O. Edfors, and T. Maseng, "Assessing interleaver suitability for turbo codes," *Nordic Radio Symposium*, Saltsjöbaden, Sweden, October 1998.

[15] Shu Lin, D. J. Costello, Jr., "Error control coding," *2nd edition, Pearson Education, Inc.*, 2004.

[16] C.C. Wang, "On the performance of turbo codes," *Military Comm. Conf. Proc., MILCOM*, Vol. 3, pp. 987-992, October 1998.

[17] C. Fragouli, and R. D. Wesel, "Semi-random interleaver design criteria", Proc. Comm. Theory Symposium at Globecom, , vol. 5, pp. 2352–2356, December 1999.

[18] J. Hokfelt, O. Edfors, and T. Maseng, "A Survey on trellis termination alternatives for turbo codes," *IEEE VTC'99, Houston, Texas*, May 1999.

[19] Fan Mo, S.C. Kwatra, and J. Kim, "Analysis of puncturing pattern for high rate turbo codes," *Military Comm. Conf. Proc., MILCOM*, Vol. 1, pp. 547-550, November 1999.

[20] W. J. Gross, and P. G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders," *Electronics Letters*, Vol. 34, no.16, pp. 1577-1578, August 1998.

[21] M. C. Valenti, and J. Sun, "The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios," *International Journal of Wireless Information Networks*, Vol. 8, no. 4, pp. 203-215, October 2001.

[22] D. W. Kim, T. W. Kwon, J. R. Choi, and J. J. Kong, "A modified two-step SOVA-based turbo decoder with a fixed scaling factor," *IEEE ISCAS*, Geneva, Switzerland, Vol. 4, pp. 37-40, May 2000.

[23] G. Colavolpe, G. Ferrari, and R. Raheli, "Extrinsic information in iterative decoding: a unified view," *IEEE Trans. on Comm.*, Vol. 49, no. 12, pp. 2088-2094, December 2001.

[24] H.R. Sadjadpour, "Maximum a posteriori decoding algorithms for turbo codes," *Proc. of SPIE Conf.*, Orlando, FL, Vol. 4045, pp. 73-83, April 2000.

[25] European Telecommunications Standards Institute (ETSI), "Universal Mobile Telecommunications System (UMTS); Multiplexing and Channel Coding (FDD)," *3GPP TS 125.212 version 7.1.0 Release 7*, pp. 16-21, June 2006.

[26] Third Generation Partnership Project 2 (3GPP2), "Physical layer standard for cdma2000 spread spectrum systems, Release C," *3GPP2 C.S0002-D, Version 2.0*, pp. 97-104, September 2005.

[27] D. U. Lee, J. D. Villasenor, W. Luk, and P. H.W. Leong, " A hardware gaussian noise generator using the Box-Muller method and its error analysis," *IEEE Trans. on Computers*, Vol. 55, no. 6, pp. 659-671, June 2006.