AN ABSTRACT OF THE THESIS OF

Aldo Tjahjadi for the degree of Master of Science in Electrical and Computer Engineering presented on November 4, 2002.
Title: Turbo-Coded OCC-CDMA with Spatial Diversity for Wireless Mobile Communications.

Redacted for privacy

Abstract approved: _____
Mario E. Magaña

Turbo codes have been used successfully for error correction in digital communications, however, their application to wireless mobile communications is still a fresh research topic. The objective of this thesis is to present a new solution that involves the usage of turbo codes, spatial diversity, and orthogonal complementary codes in a CDMA spread spectrum system in order to deliver good performance in a fading environment at low signal-to-noise ratios.

The results of the research that was performed to write this thesis are presented in a modular and progressive fashion. That is, the design of a basic turbo-coded system in additive white Gaussian noise (AWGN) is developed first in order to get a good understanding of its potential in a wireless communication environment. Then the basic system design is expanded to include fading channel characteristics and multiple access spread spectrum capabilities using classical code division multiple access

(CDMA) techniques. Finally, orthogonal complementary codes are used in lieu of traditional spreading codes and spatial diversity is introduced to maximize system performance.

Turbo-Coded OCC-CDMA with Spatial Diversity for Wireless Mobile Communications

by
Aldo Tjahjadi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 4, 2002
Commencement June 2003

Master of Science thesis of Aldo Tjahjadi presented on November 4, 2002.

APPROVED:

Redacted for privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for privacy

Head of the Department of Electrical and Computer Engineering

Redacted for privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Aldo Tjahjadi, Author

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to the following for their contribution in this study:

- Dr. Mario E. Magaña for his valuable insight, knowledge and guidance.
- Dr. Huaping Liu and Dr. Luca Lucchese for their help.
- My parents, Ridwan Tjahjadi and So Kheng Hau for their spiritual and material support.

TABLE OF CONTENTS

Page

1. INTRODUCTION..... 1

2. TURBO CODES UNDER AWGN CHANNEL..... 5

2.1 PRINCIPLE OF TURBO CODES..... 5

2.2 TURBO CODES ENCODING..... 8

2.2.1 Encoding with Recursive Systematic Convolutional Codes..... 8
2.2.2 The Effect of Interleaver in Turbo Encoding..... 13
2.2.3 Trellis Termination..... 17
2.2.4 Turbo Codes Puncturing..... 19

2.3 AWGN CHANNEL MODEL..... 21

2.4 TURBO CODES DECODING..... 21

2.4.1 Turbo Decoder Top Level..... 21
2.4.2 MAP Decoder..... 24
2.4.2.1 Calculation of Branch Metric 25
2.4.2.2 Calculation of Forward Recursion 26
2.4.2.3 Calculation of Backward Recursion..... 28
2.4.3 Max-log-MAP and Log-MAP..... 29

2.5 PERFORMANCE ANALYSIS..... 31

2.6 CHAPTER SUMMARY..... 34

3. TURBO CODES UNDER FLAT RAYLEIGH FADING CHANNEL..... 35

TABLE OF CONTENTS (continued)

Page

3.1 FLAT RAYLEIGH FADING CHANNEL..... 35

3.2 CHANNEL ESTIMATION FOR TURBO CODES.....40

3.2.1 Fading Amplitude Estimation using Wiener Filtering..... 41
3.2.2 Noise Variance Estimation..... 43
3.2.3 Channel Estimation Effects on Turbo Code Performance...43
3.2.3.1 Slow Rayleigh Fading Channel Simulation Results.....45
3.2.3.2 Fast Rayleigh Fading Channel Simulation Results.....47

3.3 DIVERSITY TECHNIQUE FOR IMPROVING PERFORMANCE..... 52

3.4 CHAPTER SUMMARY.....56

4. TURBO CODES WITH ORTHOGONAL COMPLETE COMPLEMENTARY CDMA..... 58

4.1 OCC-CDMA..... 58

4.1.1 OCC-CDMA Codes Selection..... 58
4.1.2 OCC-CDMA Encoding and Decoding..... 61
4.1.3 OCC-CDMA Compared to Conventional CDMA.....66

4.2 TURBO-CODED OCC-CDMA IN AWGN..... 68

4.3 TURBO-CODED OCC-CDMA IN FLAT RAYLEIGH FADING CHANNEL.....71

4.4 CHAPTER SUMMARY.....79

5. CONCLUSION..... 81

BIBLIOGRAPHY.....84

APPENDICES.....88

LIST OF FIGURES

Figure		Page
1.1	Basic Communications Block Diagram	1
2.1	General Block Diagram of Turbo-Coded Systems	6
2.2	Convolutional Code Example	9
2.3	Recursive Systematic Convolutional Codes	11
2.4	State Transition Diagram for RSC	11
2.5	Trellis Structure for RSC	12
2.6	Turbo Encoder System Model	14
2.7	Performance Comparison of Interleavers	15
2.8	The Effect of Interleaver Size	16
2.9	RSC Encoder with Termination	18
2.10	Code Puncturing Mechanism	19
2.11	The Effect of Puncturing	20
2.12	Top Level Representation of Turbo Decoder	22
2.13	(a) Forward Recursion and (b) Backward Recursion	28
2.14	Turbo Code Simulations for Different Number of Iterations	31
2.15	Performance Bound Comparison	34
3.1	|c(t)| for Slow Fading Channel	38
3.2	|c(t)| for Fast Fading Channel	39
3.3	Communication System with Interleaving/Deingerleaving	40

LIST OF FIGURES (continued)

Figure		Page
3.4	Degradation due to Rayleigh Fading Channel	44
3.5	Effect of Estimating Channel Side Information	45
3.6	BER Comparison for Estimated and Perfect Channel Information in Slow Rayleigh Fading	46
3.7	BER Comparison for Slow Fading and Fast Fading	47
3.8	BER Comparison for Estimated and Perfect Channel Information in Fast Rayleigh Fading	49
3.9	Effect of Channel Estimation in Fast Rayleigh Fading	50
3.10	Proposed Receiver Diversity Block Diagram	52
3.11	The BER Comparison with and without Diversity for Slow Rayleigh Fading	54
3.12	The BER Comparison with and without Diversity for Fast Rayleigh Fading	55
4.1	Correlation process in the decoder for L = 4 where user1 is the intended message to be decode	60
4.2	Coding of Intermediate Output	61
4.3	Generation of Intermediate Output Example Case	62
4.4	The Output of Example Given in Figure 4.3	63
4.5	The Offset Stacked Modulation Scheme	63
4.6	OCC-CDMA Encoder	64
4.7	OCC-CDMA Decoder	65
4.8	Chip Generation for Conventional CDMA	66

LIST OF FIGURES (continued)

Figure		Page
4.9	The Overall Block Diagram for our OCC-CDMA System.....	68
4.10	BER for Turbo-Coded OCC-CDMA in AWGN.....	70
4.11	BER Comparison for Turbo-Coded OCC-CDMA in AWGN and Slow Flat Rayleigh Fading.....	72
4.12	BER Comparison for Turbo-Coded OCC-CDMA in Fast and Slow Fading	74
4.13	Proposed Diversity Technique for Turbo-Coded OCC-CDMA.....	75
4.14	Turbo-Coded OCC-CDMA in Slow Fading with and without Diversity.....	77
4.15	Turbo-Coded OCC-CDMA in Fast Fading with and without Diversity.....	77
4.16	Quadrature Multiplexing System.....	78

LIST OF TABLES

Table		Page
2.1	Trellis and State Computations	12
2.2	State Transition Table	27
3.1	Table of Results under Slow Rayleigh Fading Channel	51
3.2	Table of Results under Fast Rayleigh Fading Channel	51
3.3	Table of Diversity Simulation Results for Slow Fading	54
3.4	Table of Diversity Simulation Results for Fast Fading	55
4.1	Two Examples of Complete Complementary Codes with Element Code Lengths L = 4 and L = 16	59
4.2	Processing Gain Table for OCC-CDMA	69
4.3	Turbo-Coded OCC-CDMA in AWGN Simulation Results	71
4.4	Turbo-Coded OCC-CDMA in Flat Rayleigh Fading Simulation Results	72
4.5	Turbo-Coded OCC-CDMA in Flat Rayleigh Fading Simulation Results for Higher Eb/No (dB) Values	73
4.6	The Turbo-Coded OCC-CDMA Spatial Diversity Results	76

Turbo-Coded OCC-CDMA with Spatial Diversity for Wireless Mobile Communications

1. INTRODUCTION

The basic concept of communications is very simple: to send data from source to destination through some medium (channel) as efficiently as possible. Efficient means not only the data that is received contains acceptable error, but also has to arrive at the destination in a reasonable length of time. The main obstacles in achieving efficient transmission are noise and distortion. As a signal is transmitted through a medium and then received, it will be contaminated by noise and it will be distorted.

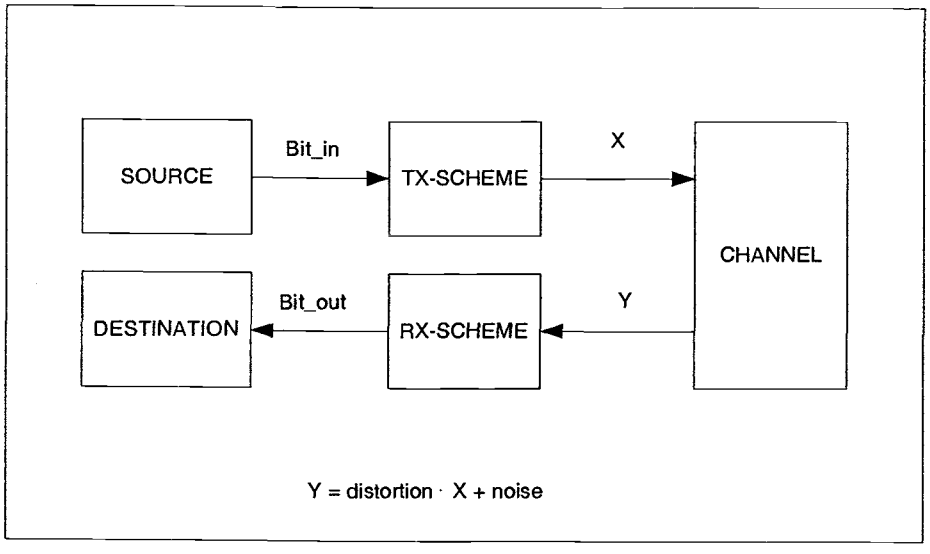

Figure 1.1: Basic Communications Block Diagram

As the need for better performance is higher and higher, especially in wireless communications, we need to apply more sophisticated signal processing to perform as demanded. Error correction coding is one of the ways to achieve better bit error rate by introducing redundancy in the transmitted information. The most powerful error correction coding so far is turbo coding.

In the late 1940's, Shannon formulated in his paper [1] the theoretical performance limit known as Shannon's theoretical capacity limit. Although he did not give the realization of such a system, he gave the idea of how good a system we can achieve. Turbo codes perform very close to Shannon's theoretical limit. Berrou, Glavieux, and Thitimajshima, in 1993 [2, 3], presented a scheme that achieves a bit error probability of 10^{-5}, using a rate ½ code over an additive white Gaussian noise channel and BPSK modulation at an E_b/N_o of 0.7 dB. These turbo codes by Berrou et al, considered by many as the fundamental turbo codes, use the parallel concatenation of codes generated by two encoders.

After the introduction of turbo codes by Berrou et al, there have been more researchers working on improvements of turbo codes, as well as on their implementations for several applications. In the mid 1990's, the turbo codes were popular in deep space communication systems [4, 5, 6]. Because of the unrestricted bandwidth and latency requirements, turbo codes are very suitable in this environment. The by-product of this research

is multiple turbo codes [7], where lower rate turbo codes are combined using three or more encoders. Later, other variants of turbo codes were introduced, such as serial turbo codes [8] and combination of serial and parallel turbo codes [9]. The more challenging application was then to use turbo codes in satellite communication systems. A satellite communication system, just like a deep space communication system, is a power-limited system. However, the bandwidth of a satellite communication system is limited and the latency is more of an issue compared to a deep space communication system. This issue has been studied and evaluated in [10, 11, 12].

The most hostile environment in the study of turbo codes is the terrestrial wireless channel. It is power-limited, very limited in bandwidth, has a very low latency requirement, and worst of all, it has distortion in the channel due to fading, shadowing, and multipath. In this environment, more design constraints are added and this changes the complexity of the problem. In most instances, the distortion in the terrestrial wireless channel is modeled as a Rayleigh fading channel. The performance of turbo codes in a Rayleigh fading channel is discussed in [13, 14, 15, 16, 17]. Recently, the application of turbo codes to multiple access communications environments has become a hot research topic. Papers such as [18] and [19] are about the application of such codes to direct-sequence code division multiple access (DS-CDMA) based systems.

The purpose of this thesis is to give an alternative way of implementing turbo codes in both additive white Gaussian noise (AWGN) channels and in Flat Rayleigh fading channels. It starts by reviewing in detail (chapter 2) the fundamentals of turbo codes under the AWGN channel assumption. The idea behind chapter 2 is to explain the system model of turbo codes that are used in this thesis. Chapter 2 also presents simulation results in terms of bit error rate for different configurations of turbo codes in an AWGN channel. In chapter 3, flat Rayleigh fading is introduced. The problems caused by Rayleigh fading are discussed and solutions are presented to mitigate the effect of this type of fading in turbo-coded systems. Chapter 4 begins with the introduction of code division multiple access based on orthogonal complete complementary codes (OCC-CDMA). This is followed by the implementation of turbo codes with OCC-CDMA in both AWGN channel and flat Rayleigh fading channel, and simulation results are presented.

All the results that are presented in this thesis are obtained using the MATLAB software package version 6.1 Release 12.1.

2. TURBO CODES UNDER AWGN CHANNEL

In this chapter, we discuss in detail the theory behind turbo codes in an additive white Gaussian noise (AWGN) environment. Under the AWGN channel assumption, the primary concern is additive noise since no distortion is introduced in the channel.

2.1 PRINCIPLE OF TURBO CODES

According to Shannon's Theorem, a random long code requires minimum signal-to-noise ratio (SNR). This means that when enough randomness in the code is used in the system, the probability of error will be arbitrarily low (assuming the rate of transmission is less than channel capacity). A random code has a large minimum distance, however, a truly random code is impossible to realize. Before turbo codes were introduced, highly structured codes were used. For example, block codes and convolutional codes. These codes suffer in the low SNR range due to the fact that they are so structured that the codes do not have the high degree of randomness. The result was an unacceptable performance with respect to Shannon's Limit. The idea behind turbo codes is to try to get a large degree of randomness by using combinations of multiple interleaved structured codes at the encoder, and using an iterative decoding algorithm at the decoder to get the

information back. By doing so, the turbo codes can perform very close to that of suggested by Shannon's Limit.

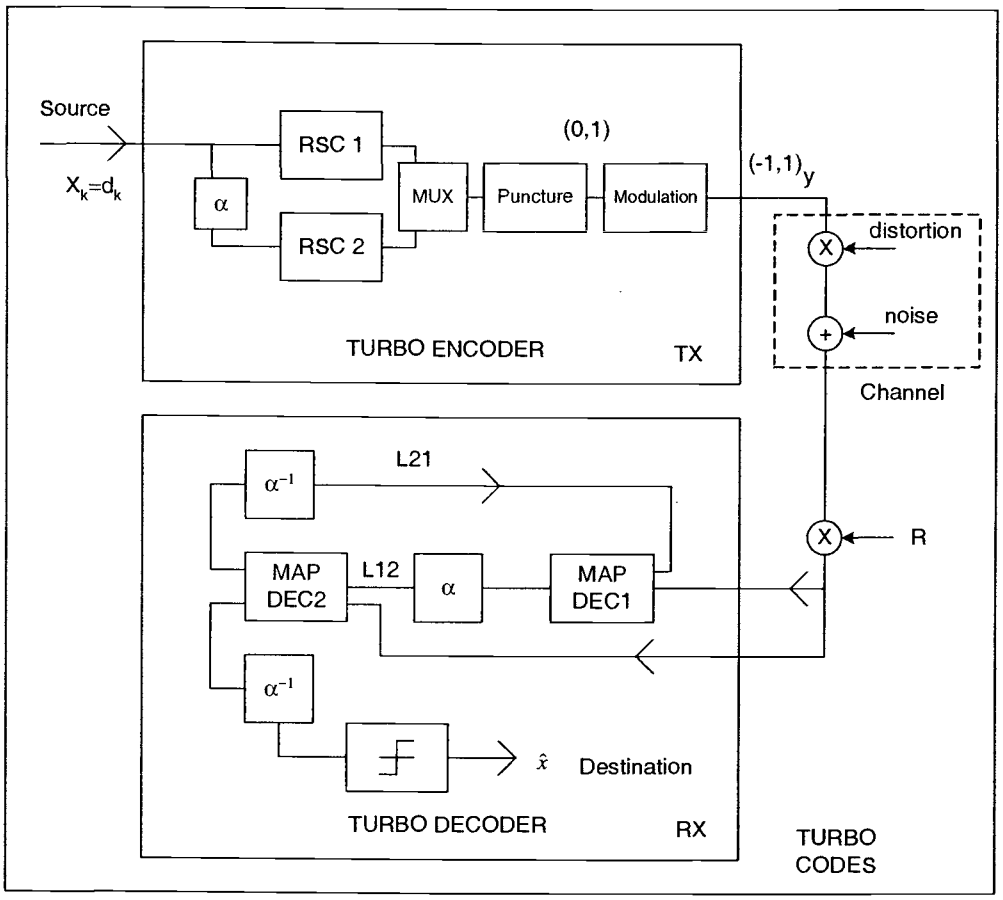

Figure 2.1: General Block Diagram of Turbo-Coded Systems

Figure 2.1 shows a block diagram of a typical turbo-code system in an AWGN environment. The system consists of two major blocks, which are the turbo encoder in the transmitter part and turbo decoder in the receiver part. Consider x as the binary data that we want to transmit. Note that x_k

represents the input going into turbo encoder at time k, which is the same as d_k. In the encoder, redundancy in introduced. This will make the rate lower, but, on the other hand, the system will not be as vulnerable to noise as before. The block with label α is the interleaver. In the code generation, we feed the first RSC encoder with x and the second RSC encoder with the interleaved version of x. If the interleaver is "good enough", it will appear that x and the interleaved version of x have no connection whatsoever. A good interleaver will produce a signal at the output of the MUX that is more random. After combining the data from RSC encoder 1 and RSC encoder 2 in the MUX, the code rate of the system becomes much smaller. Typically the code rate is 1/3, this means, for every data bit in we have 3 coded bits out. Of course, the ideal rate that we want is as close to 1 as possible. This can be achieved by puncturing. Puncturing codes means we remove unnecessary bits from the frame to make the rate of transmission faster. Typically a code rate of 1/3 can be improved to ½. The punctured coded bits are modulated before sending them through the channel. In this thesis, unless stated otherwise, the modulation scheme is BPSK, where we assume the output of the modulator is polar NRZ (1 or –1). If we look at the turbo decoder, we see 2 decoders. Because of the interleaver, with the same input x, the received values that we get contain two sets of codes, the one from x and the other one is the interleaved version of x. Decoder 1 will decode the message with codes generated from x and L21, that is the a

priori value to decoder 1 as a result of decoder 2. Decoder 2 will decode the message with codes generated from the interleaved version of x and L12 that is the a priori value to decoder 2 as a result of decoder 1. The exchange of information from decoder 1 to decoder 2 and vice versa makes the guess $\hat{x}$ more accurate in the decision device. A simple way to explain this is when decoder 1 makes a mistake because the noise introduced to code generated by x is too big, decoder 2 can help if it has a correct and confident output, so the final guess will not necessarily be wrong. This process of exchanging information from one decoder to another can be done in many iterations, and in each iteration the turbo decoder will give, in principle, a better guess in the final outcome, obeying the law of diminishing return. This is why we call the process iterative (turbo) decoding.

2.2 TURBO CODES ENCODING

2.2.1 Encoding with Recursive Systematic Covolutional Codes

Recursive Systematic Convolutional (RSC) codes are variations of convolutional codes, where systematic means that one of the outputs is the input itself and recursive means the previously encoded information bits are continually fed back to the encoder's input. In order to give a better illustration, it is better to look at an example. For a simple binary rate ½

encoder with constraint length K and memory K-1, the input to the encoder at time k is bit d_k. The output codes for convolutional codes (u_k, v_k) are

$$u_k = \sum_{i=0}^{K-1} g_{1i}\, d_{k-i} \quad \text{modulo–2, } g_{1i} = 0,1 \qquad (2.1)$$

and

$$v_k = \sum_{i=0}^{K-1} g_{2i}\, d_{k-i} \quad \text{modulo–2, } g_{1i} = 0,1 \qquad (2.2)$$

Where the generator matrix g is g = [g_{1i} ; g_{2i}]. For g = [1 1 1 ; 1 0 1] = [$(1+D+D^2)$; $(1+D^2)$] the block diagram is shown in the figure 2.2.

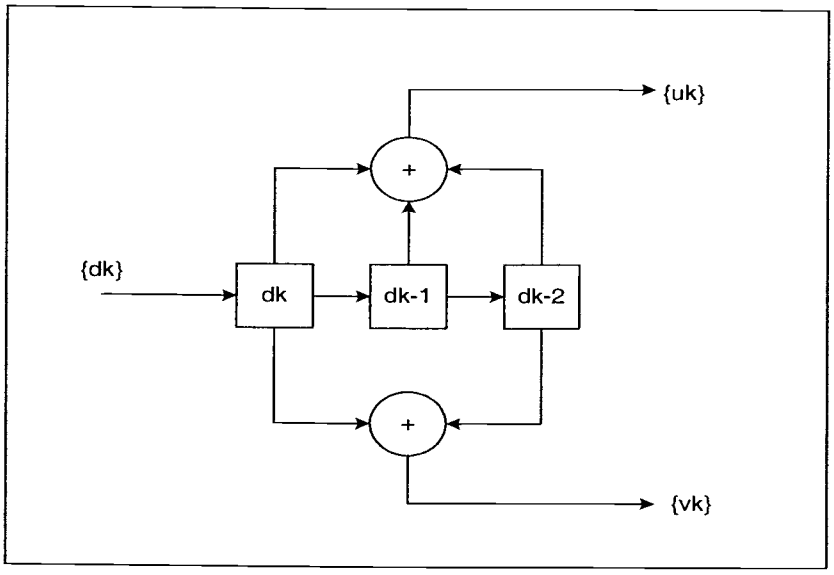

Figure 2.2: Convolutional Code Example

We can view the conventional or nonsystematic convolutional (NSC) code as a discrete-time finite impulse response (FIR) linear system. In a turbo

code, we use RSC, which is a class of infinite impulse response (IIR) convolutional code. By comparing figures 2.2 and 2.3 we can see the differences in each implementation. First of all, in RSC, one of the outputs must be the input itself. The next one is the feedback loop so that the next values that come into the register depend on the previous values in the register. For NSC construction the values in the shift register is dk-1 and dk-2, and the value dk is the input and not depends on dk-1 and dk-2. For RSC construction, instead of passing d_k right into the shift register there is a feedback loop and a_k is the value that passed into shift register.

$$a_k = d_k + \sum_{i=0}^{K-1} g_i' \, a_{k-i} \quad \text{modulo–2.} \qquad (2.3)$$

Where g'_I is equal to g_{1i} if $u_k = d_k$, and to g_{2i} if $v_k = d_k$. And for $u_k = d_k$, then

$$v_k = \sum_{i=0}^{K-1} g_{2i} \, a_{k-i} \quad \text{modulo–2.} \qquad (2.4)$$

Note that the generator matrix notation convention used up until now is for NSC. Since we can generate the equivalent model of RSC from NSC, the generator matrix notation from NSC to RSC is

$$G_{RSC} = [\, 1; \, g_{2i}(D)/g_{1i}(D)]. \quad (2.5)$$

Using the same generator matrix an equivalent example of RSC generator block diagram is shown in figure 2.3.

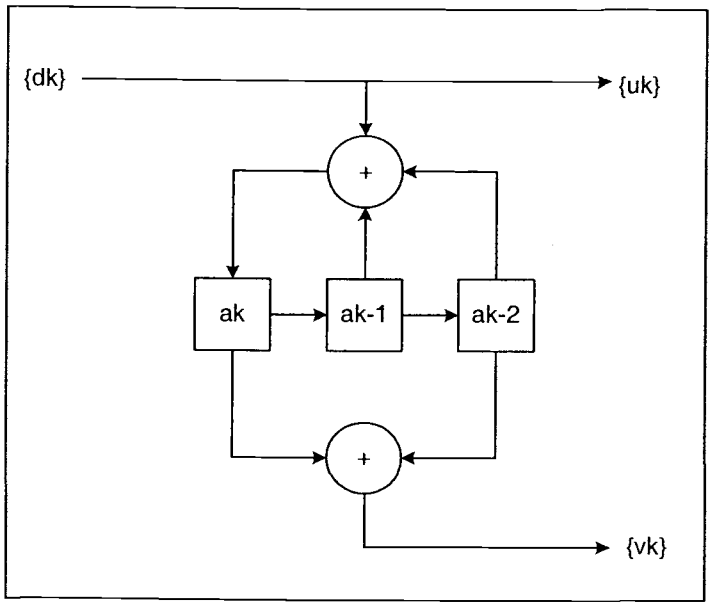

Figure 2.3: Recursive Systematic Convolutional Codes

In order to better understand the RSC encoder, figure 2.4 and 2.5 will be the state diagram and trellis structure for g = [1 1 1 ; 1 0 1] or G_{RSC} = [1 ; $(1+D^2)/(1+D+D^2)$].

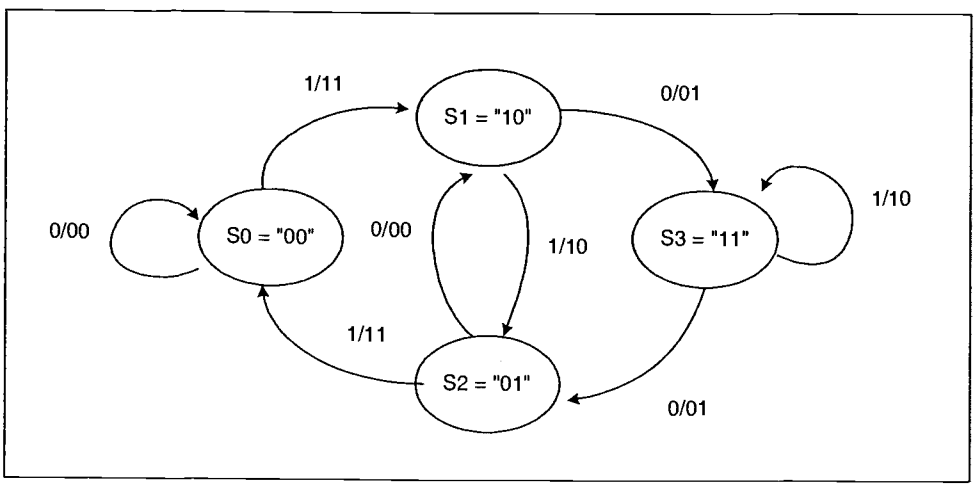

Figure 2.4: State Transition Diagram for RSC

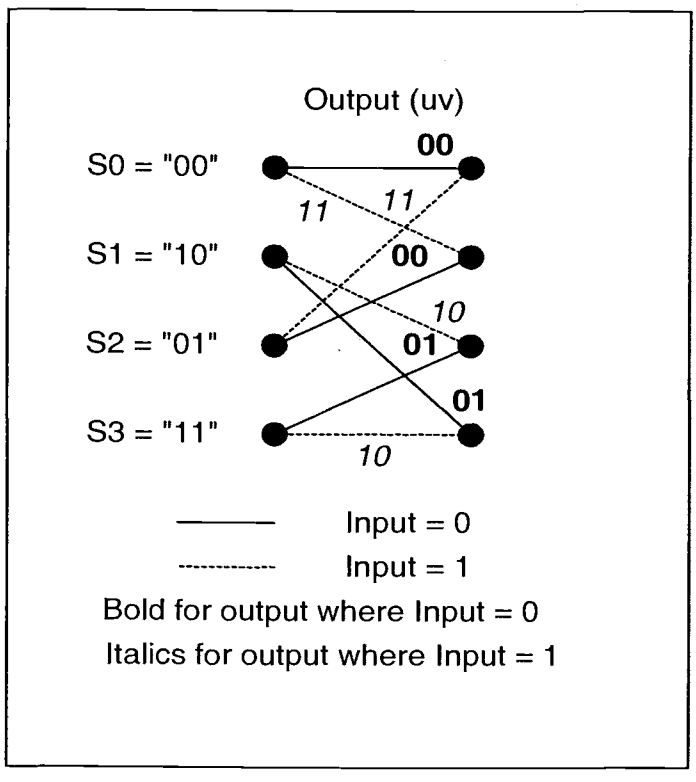

Figure 2.5: Trellis Structure for RSC

Trellis and state computations

Input bit	Present state			Output		Next State	
dk = xk = uk	Ak-1	ak-2	Ak	u	v	ak	ak-1
0	0	0	0	0	0	0	0
0	1	0	1	0	1	1	1
0	0	1	1	0	0	1	0
0	1	1	0	0	1	0	1
1	0	0	1	1	1	1	0
1	1	0	0	1	0	0	1
1	0	1	0	1	1	0	0
1	1	1	1	1	0	1	1

Table 2.1: Trellis and State Computations

2.2.2 The Effect of Interleaver in Turbo Encoding

The interleaver design is very important in turbo codes. By using the interleaver, we can have almost independent input data to the second RSC encoder since turbo decoding relies on the assumption that the noise samples affecting the transmitted symbols are mutually independent. Another function of an interleaver is to shape the weight distribution of the codes to improve the performance.

Typically, turbo codes have a fixed interleaver size which processes the data in a block-wise manner. Hence, a turbo code is a linear block code, which assumes the zero state to be the starting state for both RSC encoders. For a linear block code, an estimate of performance can be obtained by knowing the minimum distance, that is, the smallest non-zero Hamming weight. The idea of using an inteleaver in turbo encoding using RSC encoders is to achieve a high Hamming weight for most of the codes coming out of the turbo encoder. Note that the output of an RSC encoder normally has a high Hamming weight, however, for some input sequences it produces a low weight. Because of the interleaver, the encoders do not have the same input sequence. This really helps because it is very unlikely to have both encoders produce low weight outputs, although it happens sometimes. The goal here is to minimize the chance of having low weight codes at the output of both encoders.

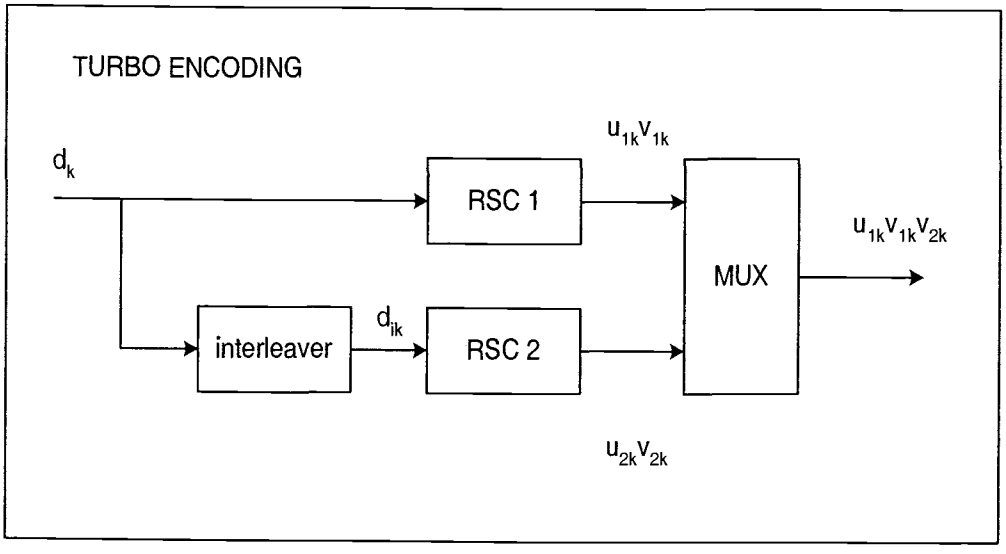

Figure 2.6: Turbo Encoder System Model

The system model of turbo encoding is explained in figure 2.6. Here we can see that RSC 2 has d_{ik} as its input, where d_{ik} is the interleaved version of d_k. The outputs of RSC 1 and RSC 2 are then multiplexed together to produce a rate 1/3 code. The u_{2k} is the interleaved version of u_{1k}, so if we know the interleaver structure used to encode, then we can de-interleave the received u_{1k} in the decoder to get u_{2k} back. Now, the design of the interleaver has to be realistic. For a random interleaver, this is not applicable because on the receiver end we do not know what the interleaver index is to de-interleave. There has been a lot of effort devoted to the design of turbo interleavers. Block or rectangular interleavers for example, use an m by n block, where we write in rows and read in columns. This is the easiest one to realize but the performance is not that great. For

the purpose of this thesis, many interleaver designs are considered, however, preference will be given to the golden interleaver design [20] because of its desired properties. In figure 2.7 we can see how the interleaver design affects the performance of the system. Where the x-axis is Eb/No, which is a measure of signal-to-noise ratio and y-axis is the bit error rates (BER), which is the ratio of number of error samples over total samples. For the simulation therein presented, the interleaver size is 400, rate 1/3, using a MAP decoder with 4 iterations in an AWGN channel and generator matrix g = [1 1 1 ; 1 0 1].

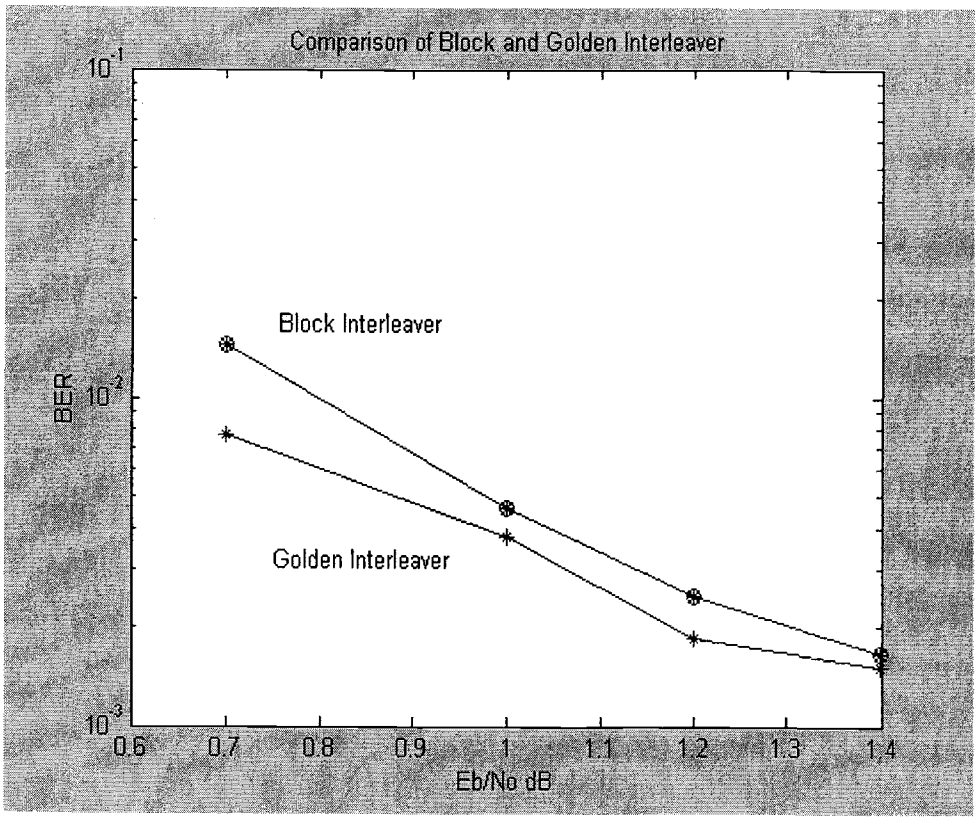

Figure 2.7: Performance Comparison of Interleavers

Another aspect of interleavers, which affects the performance of turbo codes, is the size. With a good interleaver design, the bigger the frame size the better the performance of the system. The figure 2.8 below illustrates how the performance of the system improves as we increase the interleaver or block size from 400 to 700.

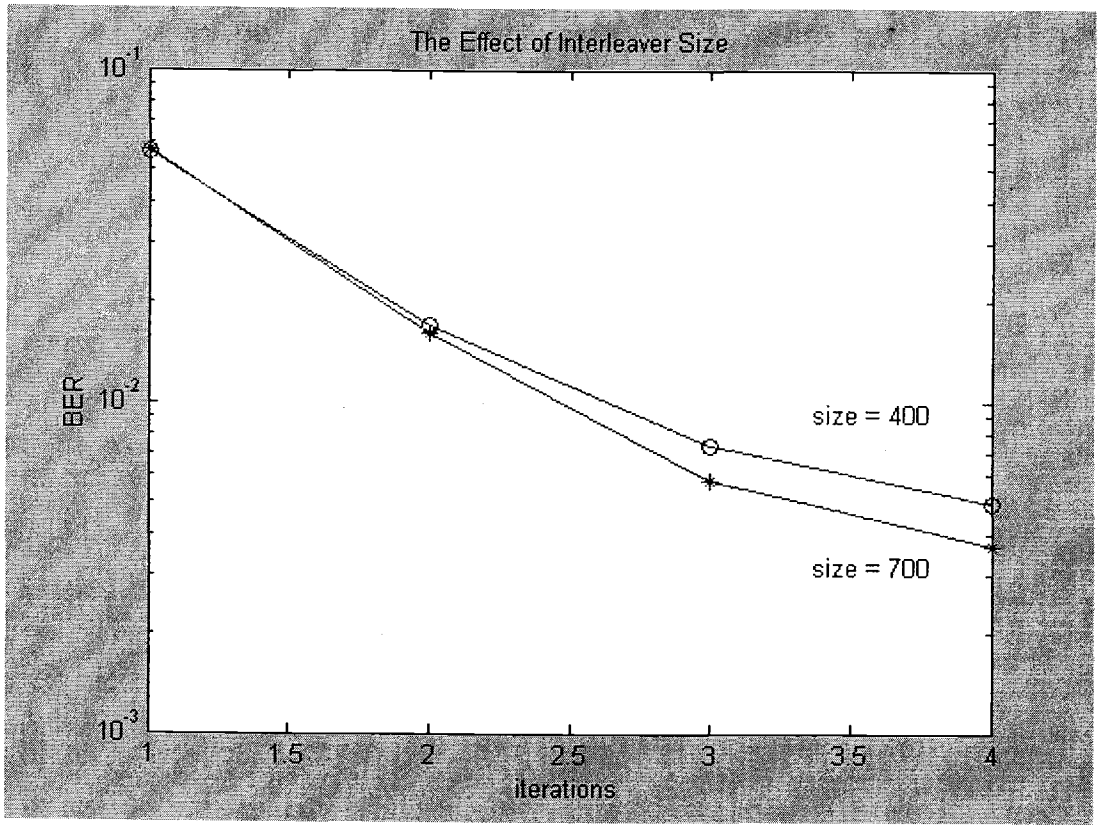

Figure 2.8: The Effect of Interleaver Size

The simulation in figure 2.8 uses 1 dB of Eb/No with a block interleaver, 4 iterations per frame and a rate 1/3 code. The generator matrix g is [1 1 1 ; 1 0 1].

2.2.3 Trellis Termination

As far as code generation is concerned, it is always preferred to start the RSC encoder from state zero. Knowledge of the starting state helps the decoding process, because now we know how to trace forward the trellis. Ideally, the best results can be obtained by perfect trellis state assignment at the start and at the end. The condition where we specified the end state of our trellis structure is trellis termination. Just like starting from state zero, we can always make the end state to be zero. In our RSC encoder, we can bring the state to the zero state using the scheme shown in figure 2.9. The structure does not change a lot, but now we incorporate a switch. When terminating the trellis, the switch will choose:

$$u_k = \sum_{i=0}^{K-1} g_i' \, a_{k-i} \quad \text{modulo–2} \quad (2.6)$$

instead of

$$u_k = d_k$$

and for v_k, instead of equation (2.4), we have

$$v_k = \sum_{i=1}^{K-1} g_{2i} \, a_{k-i} \quad \text{modulo–2.} \quad (2.7)$$

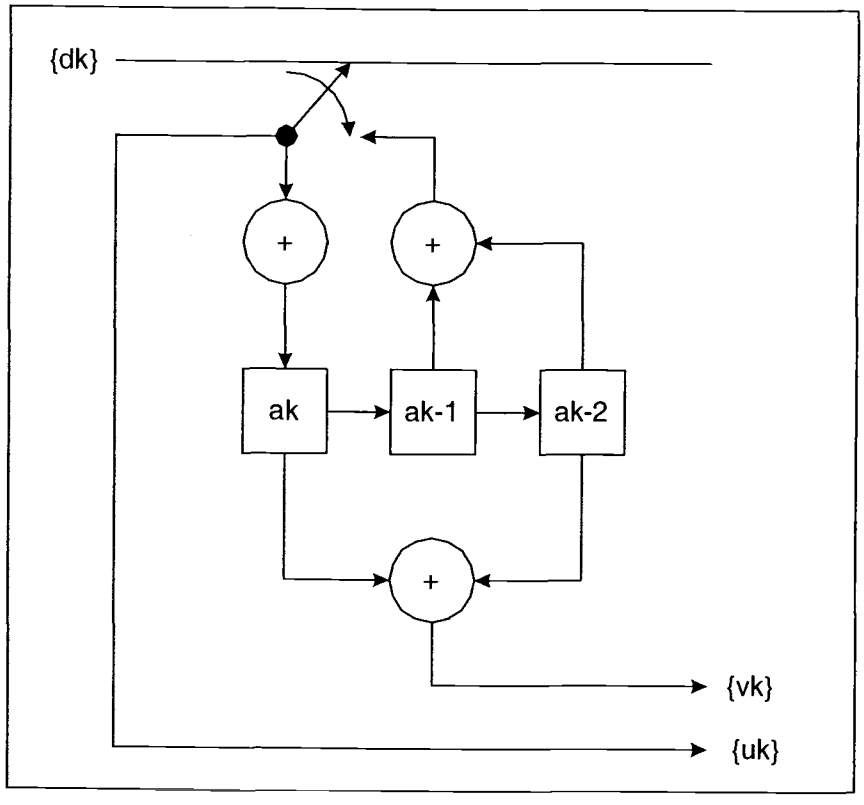

Figure 2.9: RSC Encoder with Termination

For the simulations in this thesis, single trellis termination turbo codes are used. The first RSC encoder is perfectly terminated, whereas the second RSC encoder is left un-terminated [21]. Using this method, the coded output u_k of RSC 1, which is also the input of the encoder plus terminating bits, is interleaved and then put in to un-terminated RSC encoder 2.

2.2.4 Turbo Codes Puncturing

One disadvantage of using turbo codes is that the code rate becomes slower, typically 1/3. To improve this, a puncturing mechanism can be implemented. The rate 1/3 can be improved to 1/2 using the method shown in figure 2.10.

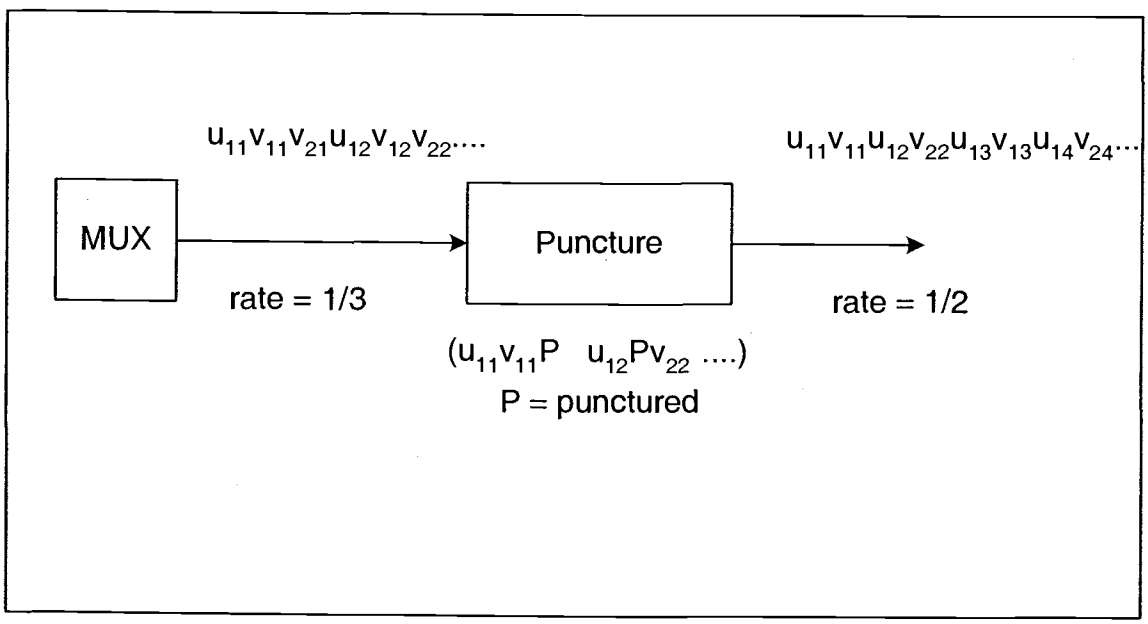

Figure 2.10: Code Puncturing Mechanism

The puncturing process has the disadvantage that it produces a higher bit error rate because of the lower redundance. Figure 2.11 shows the effect of puncturing in terms of BER. It is clear from such results that tradeoffs must be made. If the rate of transmission is the primary concern,

then puncturing the codes is a wise decision, however, for deep space communications, for example, it is not necessary to puncture because the bit error rate is the primary concern.

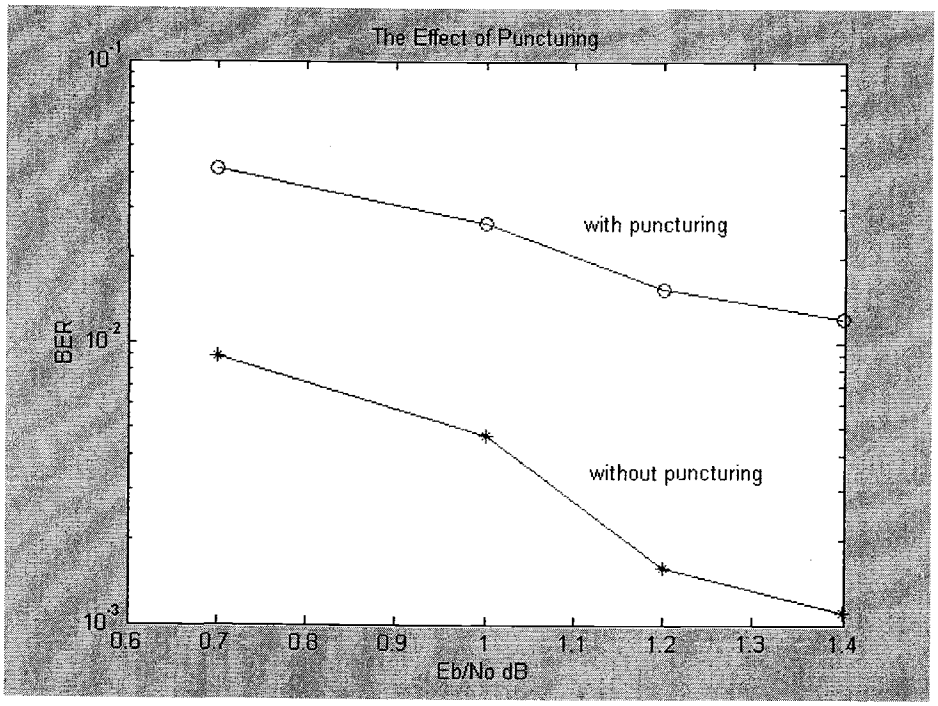

Figure 2.11: The Effect of Puncturing

2.3 AWGN CHANNEL MODEL

In calculating performance in terms of bit error rate, the input parameter is E_b/N_o. Where E_b is bit energy and N_o is the one-sided power spectral density of the zero-mean white Gaussian noise. Assuming that the signal has zero mean and variance of Es (symbol energy), the scaling of the noise is:

$$\sigma = \sqrt{\frac{1}{2\frac{Eb}{No}rate}}. \quad (2.8)$$

2.4 TURBO CODES DECODING

2.4.1 Turbo Decoder Top Level

The turbo decoding process generally uses soft-input and soft-output information to make a decision and usually based on the maximum a posteriori (MAP) decoding algorithm. The MAP algorithm was presented in 1974 by Bahl et al [22]. Originally, the algorithm was proposed as an alternative way for decoding convolutional codes. The MAP based algorithm has the advantage of performing well under low signal-to-noise ratios. However, its realization suffers from high complexity. Berrou et al [2] introduced iterative decoding utilizing MAP based decoders like the one shown in figure 2.12.

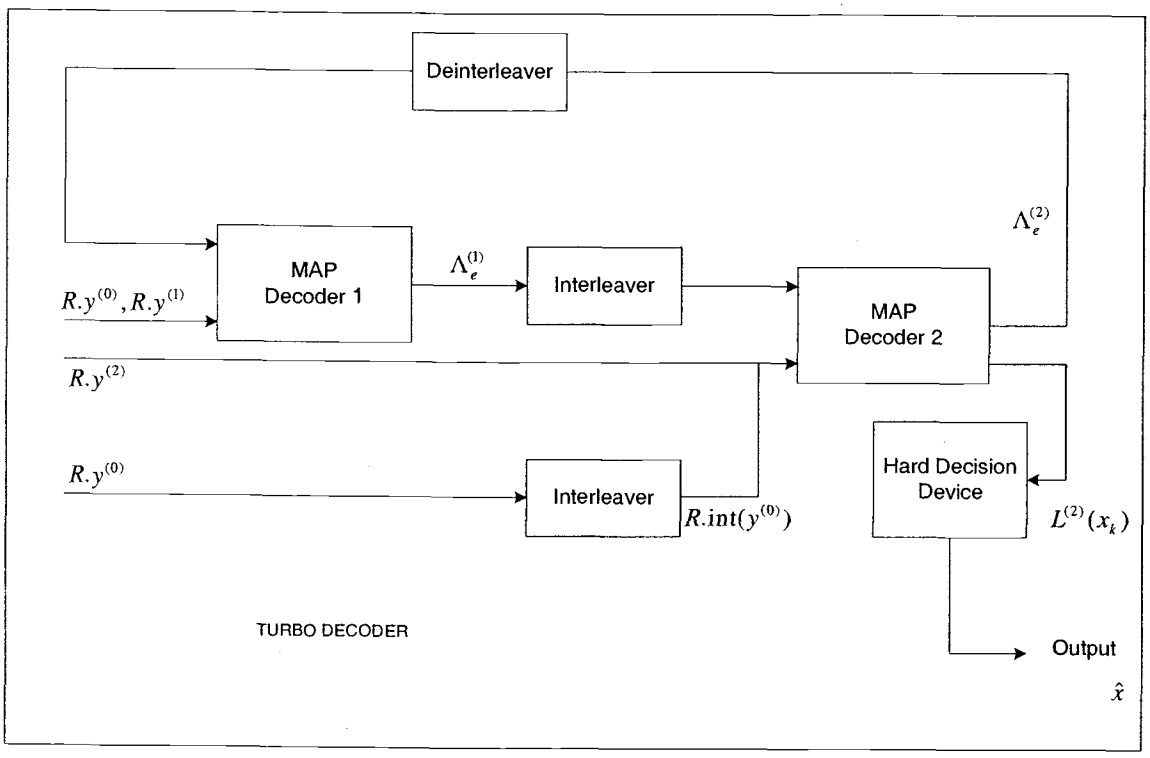

Figure 2.12: Top Level Representation of Turbo Decoder

where y is the noisy BPSK modulated code symbol vector with Es = 1, $y^{(0)}$ represents the noisy modulated coded signal u_k (refer to figure 2.9), and $y^{(i)}$ represents the noisy modulated coded signal v_k of RSC encoder i. Define R as

$$R = 4a_k \frac{Eb}{No}(rate). \qquad (2.9)$$

At time index k, the MAP decoder uses a log-likelihood ratio (LLR) of the form

$$L(x_k) = \ln \frac{P(x_k = 1 \mid y)}{P(x_k = 0 \mid y)}. \qquad (2.10)$$

The actual LLR then has a form

$$L(x_k) = \Lambda_{e,k}^{(m)} + \Lambda_s + \Lambda_k . \quad (2.11)$$

Where $\Lambda_{e,k}^{(m)}$ is called the extrinsic information from the m-th decoder, Λ_k is the a priori LLR due to the systematic bit x_k, and Λ_s is the LLR of the a posteriori probabilities of the systematic bit. It then follows that

$$\Lambda_{e,k}^{(m)} = L(x_k) - \Lambda_s - \Lambda_k , \quad (2.12)$$

where Λ_k can be obtained from the output of the other MAP decoder. For the first iteration, the value of Λ_k is set to zero, and that of Λ_s, from [23], is given by

$$\Lambda_s = 4a_k^{(0)} \frac{Eb}{No} rate \cdot y_k^{(0)} = R \cdot y_k^{(0)} . \quad (2.13)$$

Substituting (2.13) into (2.12), yields

$$\Lambda_{e,k}^{(m)} = L(x_k) - R \cdot y_k^{(0)} - \Lambda_k \quad (2.14)$$

This is part of the reason why we have to scale the received signal with R before we send it to the MAP decoders.

The decoding algorithm of the turbo decoder is relatively simple. At the first iteration, MAP decoder 1 calculates $\Lambda_{e,k}^{(1)}$ using $\Lambda_k = 0$. This $\Lambda_{e,k}^{(1)}$ is passed to MAP decoder 2 as its Λ_k, MAP decoder 2 then calculates $\Lambda_{e,k}^{(2)}$, which is then used for the next iteration as Λ_k by decoder 1. This process of exchanging extrinsic information continues until a desired performance is

achieved. At the last iteration, $L^{(2)}(x_k)$ is compared with a threshold using a hard decision device. If $L^{(2)}(x_k)$ is greater than 0, then output $\hat{x}_k = 1$, if not, then output $\hat{x}_k = 0$.

Up to this point, only a general description of the turbo decoder has been given and the MAP decoding algorithm is still vague. In the next section, the MAP decoding algorithm and its implementation will be described in detail.

2.4.2 MAP Decoder

In essence, the MAP decoder computes the log-likelihood ratio shown in equation (2.10). Using a probabilistic approach, the trellis coding implementation of equation (2.10) can be solve by using

$$L(x_k) = \ln \frac{\sum_{x^+} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k(s', s)}{\sum_{x^-} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k(s', s)}. \quad (2.15)$$

The numerator adds all possible cases where the input x is 1, whereas the denominator adds all possible cases where the input x is 0.

Since the MAP algorithm is thoroughly described in the literatures [2, 22, 24], the goal of this thesis is to present a straightforward solution of the MAP decoder so it can be easily implemented in a simulation without sacrificing the basic idea of iterative decoding. The detail of the derivation leading to equation (2.15) can be found in [2, 22, 24].

2.4.2.1 Calculation of Branch Metric

The function $\gamma_k(s',s)$ is the branch metric associated with the transition from state s' to s. The notation s' corresponds to the state s_{k-1} whereas s corresponds to state s_k. For every state transition the probability given and the received noisy observation y is different. The function $\gamma_k(s',s)$ is a measure of how probable one state transition is compared to the others. Since the state transition occurs due to effect of $\hat{u}_k$, which represents the polar modulated u_k, $\gamma_k(s',s)$ may be written as

$$\gamma_k(s',s) = P(\hat{u}_k) \cdot P(y_k \mid \hat{u}_k). \quad (2.16)$$

The equation (2.16) is not yet practical for calculating $\gamma_k(s',s)$ in a simulations environment. So, this equation needs to be further expanded, i.e.,

$$\gamma_k(s',s) \cong \exp[\frac{1}{2}\hat{u}_k(\Lambda_{e,k}^{(m)} + R \cdot y_k^{(0)}) + \frac{1}{2} R \cdot y_k^{(m)}\ \hat{v}_k^{(m)}]$$

$$\gamma_k(s',s) \cong \exp[\frac{1}{2}\hat{u}_k(\Lambda_{e,k}^{(m)} + R \cdot y_k^{(0)})] \cdot \gamma_k^e(s',s), \quad (2.17)$$

$$\text{where } \gamma_k^e(s',s) = \exp[\frac{1}{2} \cdot R \cdot y_k^{(m)}\ \hat{v}_k^{(m)}]. \quad (2.18)$$

$\hat{v}_k^{(m)}$ in the equation above is the polar modulated version of the RSC output v_k. Equation (2.17) can be used to calculate the branch metric using all the parameters that we have described. Here we can see how scaling

the observation value R fits into the equation. Equation (2.18) can be later used to directly calculate $\Lambda_{e,k}^{(m)}$, i.e.,

$$\Lambda_{e,k}^{(m)} = \ln \frac{\sum_{x^+} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s',s)}{\sum_{x^-} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s',s)}. \quad (2.19)$$

Calculating $\gamma_k(s',s)$ is the first step of the MAP decoder, because we need this value to calculate $\tilde{\alpha}_{k-1}(s')$ and $\tilde{\beta}_k(s)$.

2.4.2.2 Calculation of Forward Recursion

$\tilde{\alpha}_k(s)$ is called the forward recursion of the MAP algorithm. This is a measure of probability that accounts for all probable forward transitions into state s at time k using branch metric $\gamma_k(s',s)$. The equation that we use to calculate $\tilde{\alpha}_k(s)$ is

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s',s)}{\sum_{s} \sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s',s)} \quad \begin{matrix} \tilde{\alpha}_0(0) = 1 \\ \tilde{\alpha}_0(s \neq 0) = 0 \end{matrix} \quad (2.20)$$

where the numerator calculates the sum of the products of branch metrics and the forward recursion for all possible transitions to state s at time k.

From the state diagram in figure 2.4 we can generate the table below.

State transition for

$G_{RSC} = [\ 1\ ;\ (1+D^2)/(1+D+D^2)]$

Case	Present State	Input bit(x)	Next State
1	0	0	0
2	1	0	3
3	2	0	1
4	3	0	2
5	0	1	1
6	1	1	2
7	2	1	0
8	3	1	3

Table 2.2: State Transition Table

For example, assume that at time k we want to calculate $\tilde{\alpha}_k(s)$. Let us suppose that state s is 3. The possible case for forward recursion is case 2 and case 8. In case 2, the input bit $x_k = 0$ causes a transition from state 3 to state 2. As for case 8, the input bit $x_k = 1$ causes a transition from state 3 onto itself.

So the numerator portion becomes

$$\sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s) = \gamma_k(1,3) \cdot \tilde{\alpha}_{k-1}(1) + \gamma_k(3,3) \cdot \tilde{\alpha}_{k-1}(3). \quad (2.21)$$

The denominator of equation (2.20) calculates the sum of the products of the branch metrics and the forward recursion for all possible transitions at time k. Again using the example above the denominator portion will be

$$\begin{aligned} &\tilde{\alpha}_{k-1}(0) \cdot (\gamma_k(0,0) + \gamma_k(0,1)) + \tilde{\alpha}_{k-1}(1) \cdot (\gamma_k(1,3) + \gamma_k(1,2)) \\ &+ \tilde{\alpha}_{k-1}(2) \cdot (\gamma_k(2,0) + \gamma_k(2,1)) + \tilde{\alpha}_{k-1}(3) \cdot (\gamma_k(3,2) + \gamma_k(3,3)). \end{aligned} \quad (2.22)$$

2.4.2.3 Calculation of Backward Recursion

The term $\tilde{\beta}_k(s)$ is called the backward recursion. Just like $\tilde{\alpha}_k(s)$, it is a measure of probability. This can be best described using figure 2.13, where we compare the backward and forward recursion paths using the same example from the previous subsection.

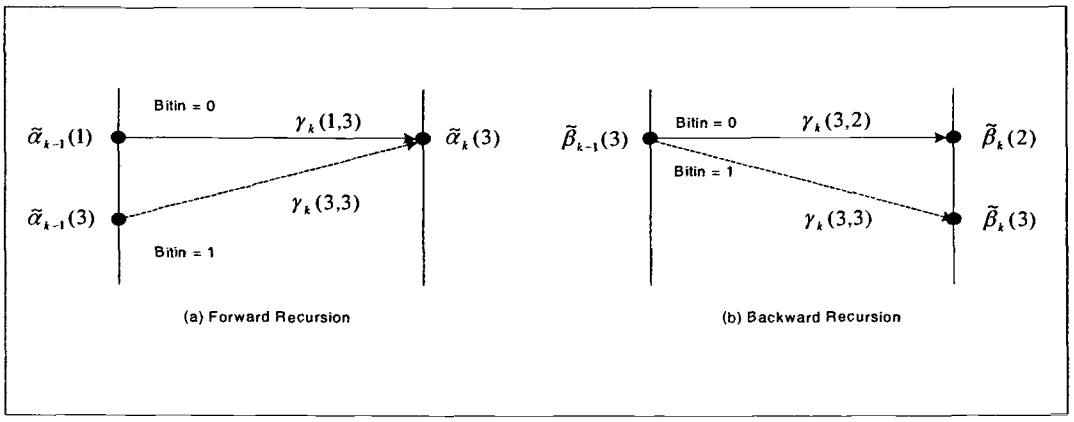

Figure 2.13: (a) Forward Recursion and (b) Backward Recursion

The equation that we use to calculate $\tilde{\beta}_k(s)$ is given by

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s'} \tilde{\beta}_k(s) \cdot \gamma_k(s', s)}{\sum_{s} \sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s)}. \quad (2.23)$$

The initial condition of the backward recursion at time k = N depends on the termination of the codes that was discussed in 2.2.3. If the trellis is not terminated, the initial condition for decoder 2 becomes $\tilde{\beta}_N(s) = 1$, s = any possible state. For dual termination decoders, and decoder 1 for single termination it will be $\tilde{\beta}_N(0) = 1$, and $\tilde{\beta}_N(s \neq 0) = 0$.

Now, using the same example of the previous section, we can rewrite the numerator of equation (2.23) into

$$\sum_{s'} \tilde{\beta}_k(s) \cdot \gamma_k(s', s) = \gamma_k(3,2) \cdot \tilde{\beta}_k(2) + \gamma_k(3,3) \cdot \tilde{\beta}_k(3). \quad (2.24)$$

After calculating all the branch metrics, forward recursions and backward recursions all we need to do is solve equation (2.15) for the LLR or if we want to get the extrinsic value right away we can directly calculate (2.19).

2.4.3 Max-log-MAP and Log-MAP

The disadvantage of the MAP algorithm lies in its computational complexity. To realize such a device is very expensive. Another disadvantage of the

MAP algorithm is its sensitivity to round off error, which is very critical because turbo codes deal with small values of signal-to-noise ratio. A slight round off error may cause saturation in performance. In order to avoid this, the actual realization of the MAP algorithm is done in the log domain [25, 26]. In the log domain, multiplications become additions of logs. The problem is when we have to perform additions. An alternative to this is the Max-log-MAP algorithm, which makes the approximation:

$$\ln(e^{\delta_1} + ... + e^{\delta_n}) = \max_{i \in \{1..n\}} \delta_i . \quad (2.25)$$

Since this is only an approximation of the MAP algorithm, it results in sub-optimal performance with inferior results compared to the actual MAP algorithm.

Using the Jacobian logarithm property [9-11], we can improve the approximation in (2.25) using

$$\ln(e^{\delta_1} + e^{\delta_2}) = \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|), \quad (2.26)$$

where $f_c(\cdot)$ is a correction function. The correction function is usually approximated using a pre-computed table. Since it only depends on $|\delta_1 - \delta_2|$, this table is one-dimensional and the realization usually only needs very few values stored. For more details about these alternative solutions, [9] explains in detail how to do this scheme.

2.5 PERFORMANCE ANALYSIS

Figure 2.14 shows the effect of the number of iterations in a rate 1/3 turbo code that uses a random interleaver with a size of 500 symbols per block in an AWGN channel. Here we can see that as the number of iterations continues, the improvements from the previous iterations get smaller and smaller. At some point the effect of increasing the number of iterations is almost negligible, this is where the saturation point occurs, because it is very close to the turbo code performance bound called minimum distance asymptote.

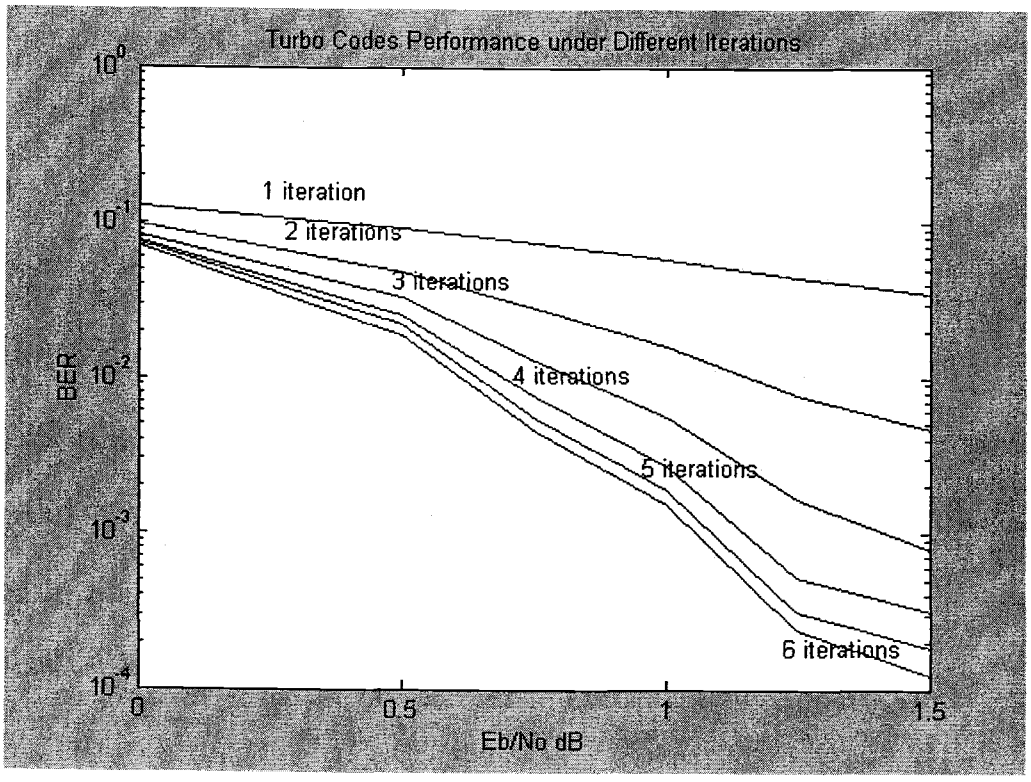

Figure 2.14: Turbo Code Simulations for Different Numbers of Iterations

We can approximate the performance bound of turbo codes using bit error probability of linear block codes under AWGN channel using maximum likelihood decoding. The bound for linear block codes of size k and rate r can be approximated by

$$P_b \le \sum_{i=1}^{2^k-1} \frac{w(m_i)}{k} Q\left(\sqrt{w(x_i)\cdot 2\cdot r\cdot \frac{Eb}{No}}\right), \quad (2.27)$$

where x_i is the code word output of the encoded message m_i. For turbo codes, equation (2.27) can be derived so that we can calculate the minimum distance asymptote using

$$P_b \approx \frac{\tilde{w}_{d\min} N_{d\min}}{k} Q\left(\sqrt{d_{\min}\cdot 2\cdot r\cdot \frac{Eb}{No}}\right). \quad (2.28)$$

In the above equation, $\tilde{w}_{d\min}$ is the average weight of the messages associated with the minimum distance code words. $N_{d\min}$ is the number of free distance code words on average and $d_{\min}$ is the minimum free distance. For example, the rate 1/2 turbo code with parameters (37, 21, 65536), i.e., interleaver size 65536 and code generator matrix G = [37;21], the minimum free distance is $d_{\min} = 6$, $\tilde{w}_{d\min} = 2$ and $N_{d\min} = 3$ [28]. For this code, the minimum distance asymptote is

$$P_b \approx \frac{(2).(3)}{65536} Q\left(\sqrt{(6)\cdot(2)\cdot(0.5)\cdot \frac{Eb}{No}}\right). \quad (2.29)$$

If we compare this result with the convolutional code bound [28] given by

$$P_b \approx \sum_{d=dfree}^{2(v+k)} W_d^0 \cdot Q\left(\sqrt{d \cdot 2 \cdot r \cdot \frac{Eb}{No}}\right), \quad (2.30)$$

where v is the number of tail bits used to force encoder to state 0, W_d^0 is the sum of the weights of all messages with $m_0 = 1$, which have code words whose weight is d. Using the example convolutional code with parameter (2,1,14), the bit error probability is evaluated in [29] and is given by

$$Pb \approx 137 \cdot Q\left(\sqrt{(18) \cdot (2) \cdot (0.5) \cdot \frac{Eb}{No}}\right). \quad (2.31)$$

Comparing equations (2.29) and (2.31) we can see that for a very low signal-to-noise ratio, the scaling factor of the Q-function is the dominant term of the performance measure. In general, it is easily noticed that for turbo codes, the bigger the interleaver size the smaller the scaling factor of the Q-function in the equation. This results in better performance or low probability of error. On the other hand, for convolutional codes, the scaling factor is large, hence, the lower performance in the low end of signal-to-noise ratio values. For higher signal-to-noise ratios, convolutional codes perform much better than turbo codes, because the value of Q-function decreases very rapidly as the argument of the Q-function increases. We can see in figure 2.15, the critical point where convolutional codes perform better than Turbo codes is about 3.56 dB of Eb/No.

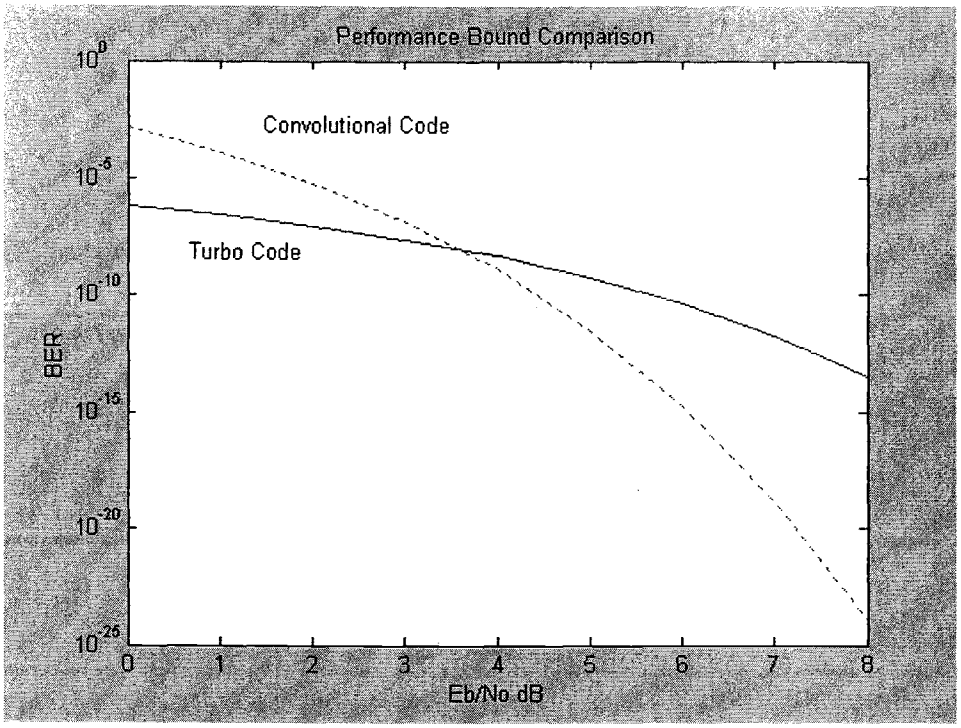

Figure 2.15: Performance Bound Comparison

2.6 CHAPTER SUMMARY

In this chapter, the analysis and performance of turbo codes in AWGN has been presented. We started with the basic idea or framework of turbo codes in general, and then as we went deeper into the chapter each block of the system was discussed in detail.

This was done to get a good understanding of how turbo codes work and to assess their potential in a wireless communication environment, in order to maximize performance.

3. TURBO CODES UNDER FLAT RAYLEIGH FADING CHANNEL

In the previous chapter, the channel did not experience any fading. In wireless communications, however, the signal that propagates through the channel will undergo distortion. This chapter will focus on the performance of turbo codes under flat Rayleigh fading conditions.

3.1 FLAT RAYLEIGH FADING CHANNEL

The output of a transmitted signal s(t) that travels through an equivalent baseband fading channel with additive white Gaussian noise is

$$y(t) = c(t)s(t) + n(t). \quad (3.1)$$

In equation (3.1), c(t) is the channel attenuation coefficient which is a complex-valued process, and n(t) is a zero-mean complex white Gaussian noise. The coefficient c(t) is usually represented as

$$c(t) = | c(t) | e^{(j\theta)} = (A_S + Y_C(t)) + jY_S(t), \quad (3.2)$$

where A_S represents the amplitude of the specular component . If the specular component is not present, $A_S = 0$, the channel is Rayleigh fading. In a Rayleigh fading channel, there is no direct line-of-sight component. However, this special case seldom occurs. If the specular component is present, which usually happens, the channel is called a Rician fading

channel. The construction of the fading channel in this thesis uses Jakes' model [30]. With $A_S = 0$, we can expand equation (3.2) into

$$c(t) = \frac{C_0}{\sqrt{2S_0 + 1}}[X_c(t) + jX_s(t)], \quad (3.3)$$

where

$$X_c(t) = 2\sum_{n=1}^{S_0}[\cos(\phi_n)\cos(\omega_n t)] + \sqrt{2}\cos(\phi_N)\cos(\omega_m t), \quad (3.4)$$

$$X_s(t) = 2\sum_{n=1}^{S_0}[\sin(\phi_n)\cos(\omega_n t)] + \sqrt{2}\sin(\phi_N)\cos(\omega_m t), \quad (3.5)$$

where S_0 is the total number sinusoids used. In [30], Jakes suggests that 8 sinusoids will give a good approximation. Other parameters used for simulating Jakes' Model are

$$\omega_m = 2\pi f_d, \quad (3.6)$$

$$\omega_n = \omega_m \cos(2\pi n / S), \quad (3,7)$$

$$S = 2(2S_0 + 1), \quad (3.8)$$

$$\phi_N = 0, \quad (3.9)$$

$$\phi_n = \pi n/(S_0 + 1), \quad (3.10)$$

and

$$C_0 = 1, \quad (3.11)$$

in order to satisfy the following statistics

$$E\{X_c^2(t)\} = S_0, \quad (3.12)$$

$$E\{X_s^2(t)\} = S_0 + 1, \quad (3.13)$$

$$E\{|c(t)|^2\} = C_0 = 1. \quad (3.14)$$

An important property of this fading model is

$$\Phi(\Delta t) = E\{c(t)c^*(t+\Delta t)\} = J_0(2\pi f_d \Delta t), \quad (3.15)$$

where $J_0()$ is the 0^{th}-order Bessel function of the first kind and f_d is the maximum Doppler shift frequency.

A flat fading channel is also known as a frequency non-selective fading. This means that the radio channel has a constant gain and a linear phase response over a bandwidth that is larger than that of the transmitted signal. Hence, there is no channel-induced ISI distortion because there is no significant overlap among neighboring received symbols.

Now, a flat Rayleigh fading channel performs worse than an AWGN channel. For example, if BPSK modulation is used, an error in the received symbol will occur if we send a positive s(t) and receive a negative y(t) or vice versa. If |c(t)| is less than 1, then the probability of receiving a symbol with error sign will be higher.

In this thesis, two different types of flat Rayleigh fading channels are considered, i.e., a slow fading channel and a fast fading channel. A slow fading channel is characterized by setting the value of $f_d \cdot T_s$ to be around

0.005 and for fast fading channel $f_d \cdot T_s$ = 0.02. The product $f_d \cdot T_s$ is usually called the normalized fade rate.

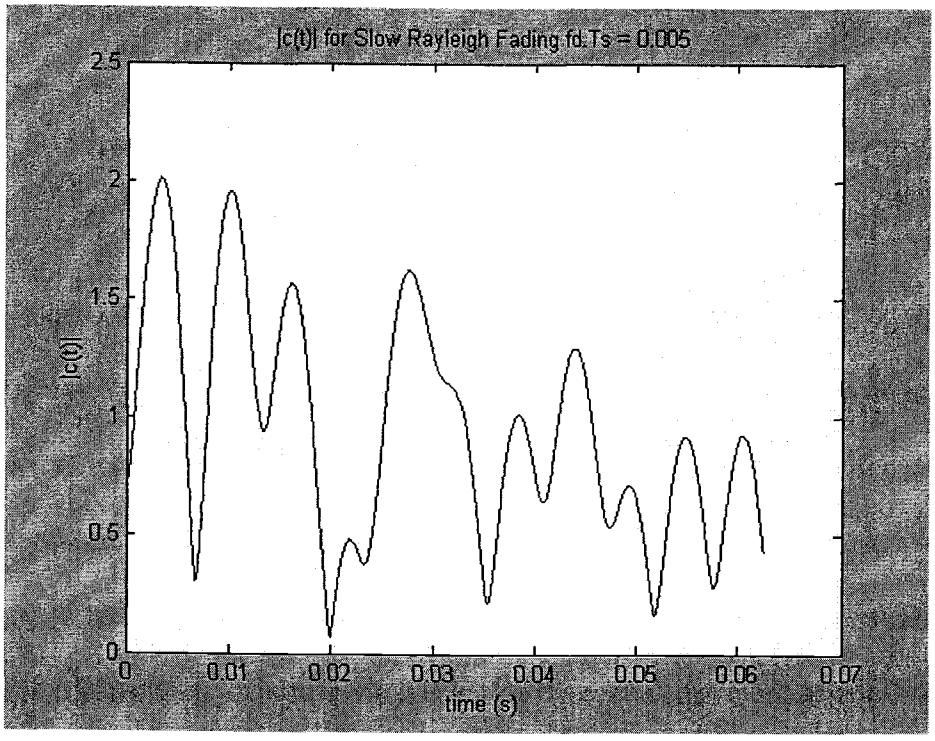

Figure 3.1: |c(t)| for Slow Fading Channel

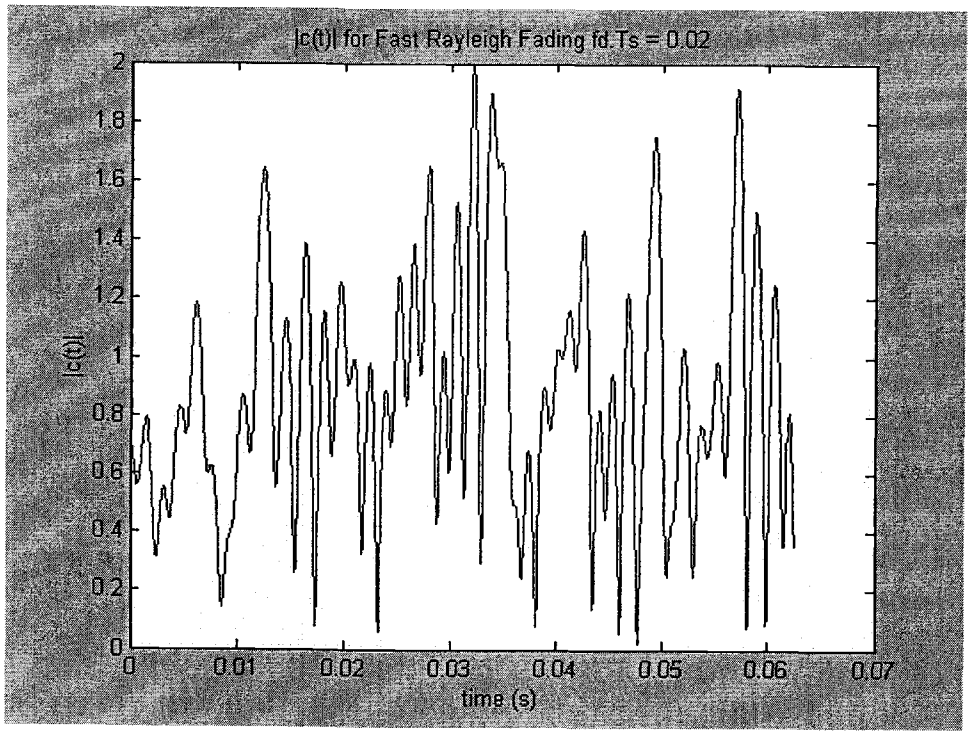

Figure 3.2: |c(t)| for Fast Fading Channel

Figures 3.1 and 3.2 are time domain representations of the fading channel coefficient |c(t)|. It can be shown that in a fast fading channel, the transition of a sinusoid-like wave is faster than the case of slow fading. As $f_d \cdot T_s$ becomes smaller, more burst errors, which are errors in consecutive symbols, occur. In order to reduce burst errors, a block interleaver is used, since it reduces the statistical dependence between consecutive symbols.

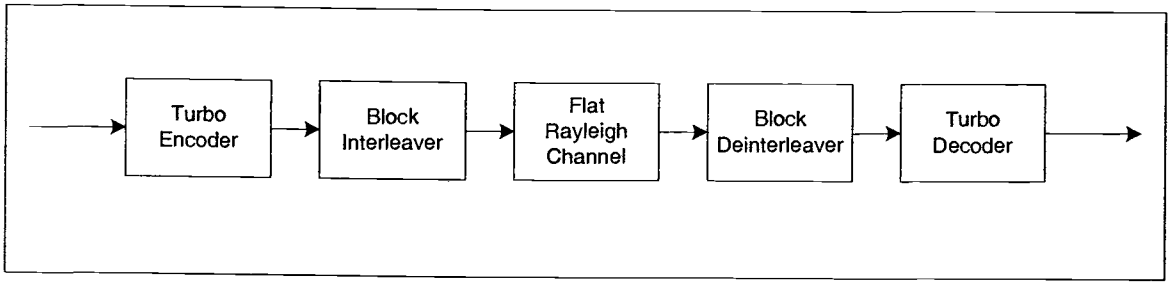

Figure 3.3: Communication System with Interleaving/Deinterleaving

3.2 CHANNEL ESTIMATION FOR TURBO CODES

In order for the MAP decoder to work properly, we need to know or estimate the channel parameters to solve equation (2.9). The noise variance σ^2, fading amplitude |c(n)|, and carrier phase θ_n, are called side information. For an AWGN channel only, it is easy to calculate these parameters because |c(n)|=1, and θ_n=0. However, in the fading channel scenario, we cannot get the perfect estimates of this side information because turbo codes typically operate at very low signal-to-noise ratios. Thus, the performance of a turbo-coded system in a fading channel environment will be degraded.

In this thesis, we assume that the values of the carrier phase can be accurately estimated using schemes like phase locked loops for sufficiently high signal-to-noise ratios and low normalized fade rates or using pilot signals as suggested by Valenti and Woerner [13] for low signal-to-noise ratios and higher normalized fade rate.

The effect of noise variance estimation in the performance of turbo codes is presented in [31, 32]. These results conclude that the performance is not sensitive to noise variance estimation errors. It is suggested that an error of less than 3 dB does not sharply degrade the performance.

3.2.1 Fading Amplitude Estimation using Wiener Filtering

Let v_n be the transmitted symbol after modulation and y_n be the received symbol at the input of the receiver. The best linear minimum mean-square (MMSE) estimate of the fading amplitude is

$$\hat{a}_n = |\hat{c}_n| = \sum_{i=-N_c}^{N_c} w_i v_{n-i} y_{n-i}, \quad (3.16)$$

where N_c is related to the order of the filter by $2N_c+1$ and w_i is the i-th Wiener filter coefficient. The Wiener filter coefficients can be obtained by solving the Wiener-Hopf equations

$$\overline{R}_x \overline{w} = \overline{r}_{dx}. \quad (3.17)$$

In order to solve this equation, we assume that the maximum Doppler shift frequency f_d is known. Equation (3.15) can be rewritten as

$$\overline{r}_{dx} = J_0(2\pi f_d T_s i), \quad (3.18)$$

for $-N_c \le i \le N_c$. Now, for $\overline{R}_x$, if we know the variance of the noise, we can generate the theoretical value for $\overline{R}_x$ using

$$\overline{R}_x = toeplitz\left(J_0(2\pi f_d T_s m) + [\sigma^2\ zeros(1,2*N_c)]\right), \quad (3.19)$$

for m = [0 1 2 ... $2N_c$], or we can just get the autocorrelation straight from the received symbol.

We note in equation (3.16) that v_n is not known in the receiver, but, a_n always has a positive sign. Therefore, we can simplify it, i.e.,

$$\hat{a}_n = |\hat{c}_n| = \sum_{i=-N_c}^{N_c} w_i \mid y_{n-i} \mid . \quad (3.20)$$

For a slow normalized fade rate and small filter order, the filter coefficients can be approximated by

$$w_i \approx \frac{1}{2N_c + 1} \quad \forall i. \quad (3.21)$$

In this case, the filter is just a moving average filter. An advantage of using this approximation is that we do not have to know the channel fade rate.

The value of N_c that we use depends on how fast the fade rate is. Since equation (3.20) evaluates the sum from $-N_c$ to N_c, this becomes a sliding window process that looks at the values before and after n and makes an intelligent guess of what $\hat{a}_n$ is. Hence, if the fade rate is slow, we can use a larger window size and larger value for N_c, in order to obtain a better average. For fast fading, the window size is forced to be smaller and for the same sampling rate, we use a fewer data points to predict $\hat{a}_n$ and therefore worse predictions usually occurs.

3.2.2 Noise Variance Estimation

Although noise variance does not affect the performance as much as the fading amplitude, we must have a way to approximate it at least within 3 dB of accuracy. From [14], it is proposed that

$$\hat{\sigma}^2 = \frac{c}{L-1}\sum_{k=1}^{L}((|\, y_k \,| - \hat{a}_k) - \mu)^2, \quad (3.22)$$

where L is the number of code bits per block and μ is the sample mean of $e_k = |\, y_k \,| - \hat{a}_k$, i.e.,

$$\mu = \frac{1}{L}\sum_{k=1}^{L}(|\, y_k \,| - \hat{a}_k). \quad (3.23)$$

The constant c in equation (3.22) is a scaling factor to correct the biased estimates due to the absolute value operation. The proposed value for c for all values of Eb/No is c = 1.5.

3.2.3 Channel Estimation Effects on Turbo Code Performance

We will always observe more performance loss in a fading channel than in a channel with only additive white Gaussian noise (AWGN). Figure 3.4 illustrates how bad a rate ½, 4 iterations turbo-coded system is in low signal-to-noise ratios when fading is included. Channel estimation provides estimates of channel information so that we can minimize the error at the receiver. Unfortunately, even with perfect estimates the performance loss is

still not acceptable. This is shown in figure 3.4 (note that it is better in higher signal-to-noise ratios).

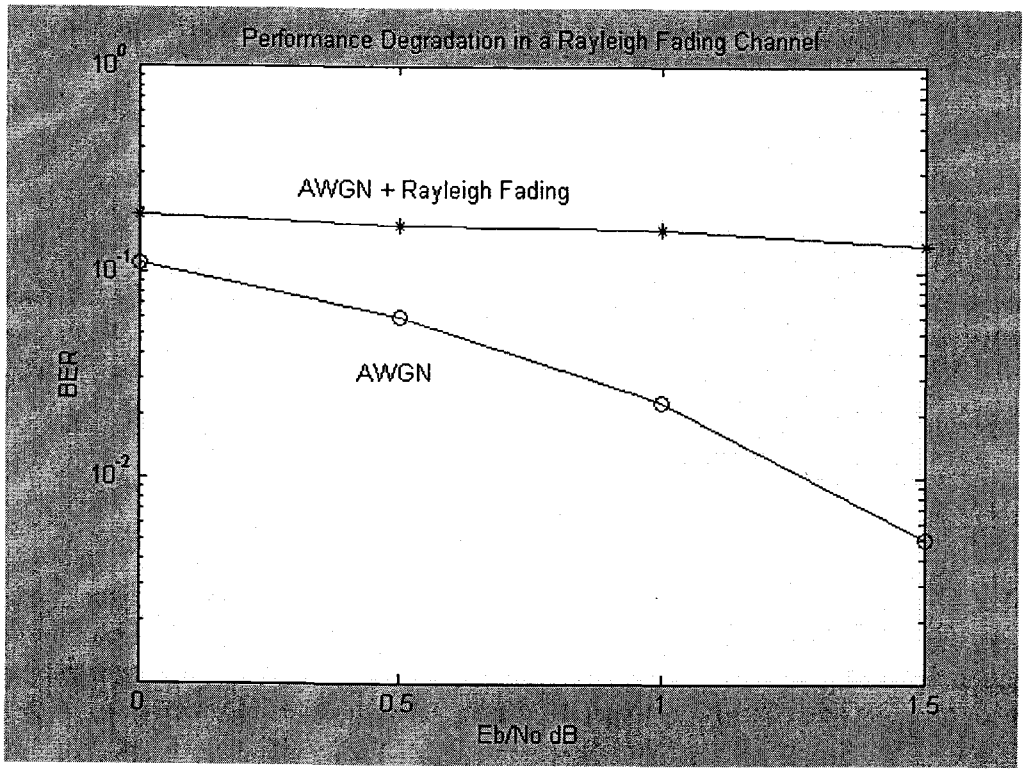

Figure 3.4: Degradation due to Rayleigh Fading Channel

As mentioned before, the simulations consider two scenarios, i.e., slow fading and fast fading. For both scenarios, we can compare the performance of the turbo codes under a flat Rayleigh fading using estimated channel side information and perfect channel side information. Although we cannot have perfect channel information in the realization, we can obtain a bound of system performance.

3.2.3.1 Slow Rayleigh Fading Channel Simulation Results

In slow Rayleigh fading, the transition between peaks of fading amplitude takes a longer period of time. Because of this, we can use a larger sliding window size for the filter coefficients for the channel fading amplitude estimation. For $f_d \cdot T_s$=0.005, the selected value for N_c is 25.

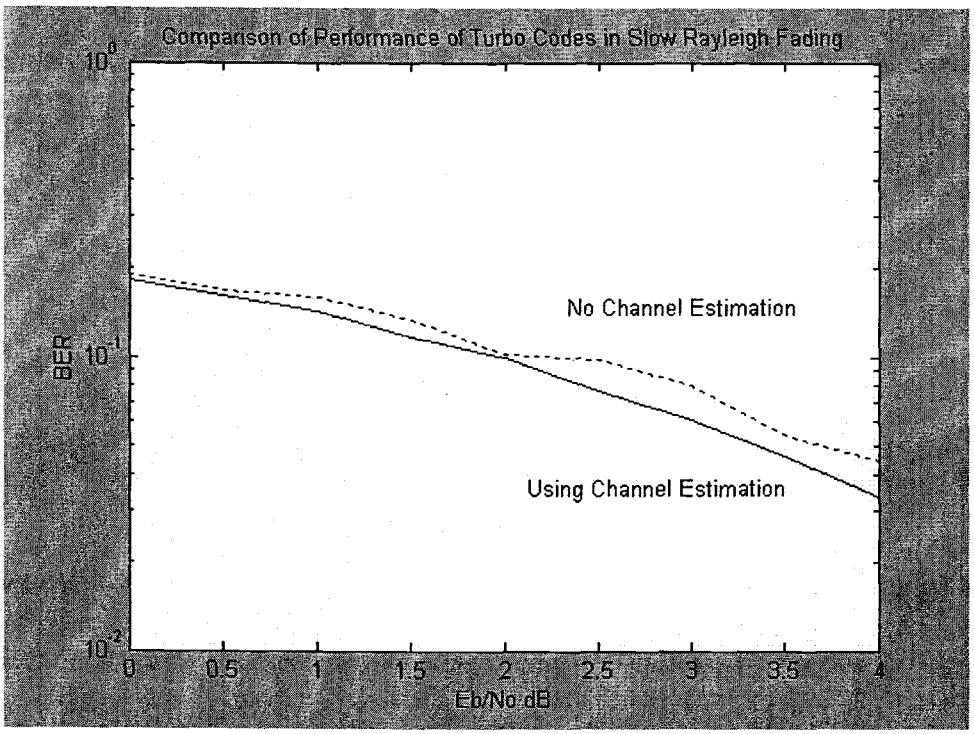

Figure 3.5: Effect of Estimating Channel Information

Figure 3.5 shows how we can improve the bit error rate by doing channel estimation. In this simulation, the turbo code used has rate ½ with constraint length 3, and uses a block channel interleaver of length 1200 with 4 iterations decoding. This code is used in all simulations for the

remainder of this chapter. Note that this is not necessarily the best configuration. We can always use bigger frame size, bigger constraint length or more iterations to get better results. Once again, for the purpose of this thesis, this setting is used due to the fact that we have to consider the bandwidth and latency requirement of a real system. A bigger frame size and more iterations means more latency. Also, this setting does not require a super fast computer to run the simulations and still get a good idea of the behavior of the system that being observed.

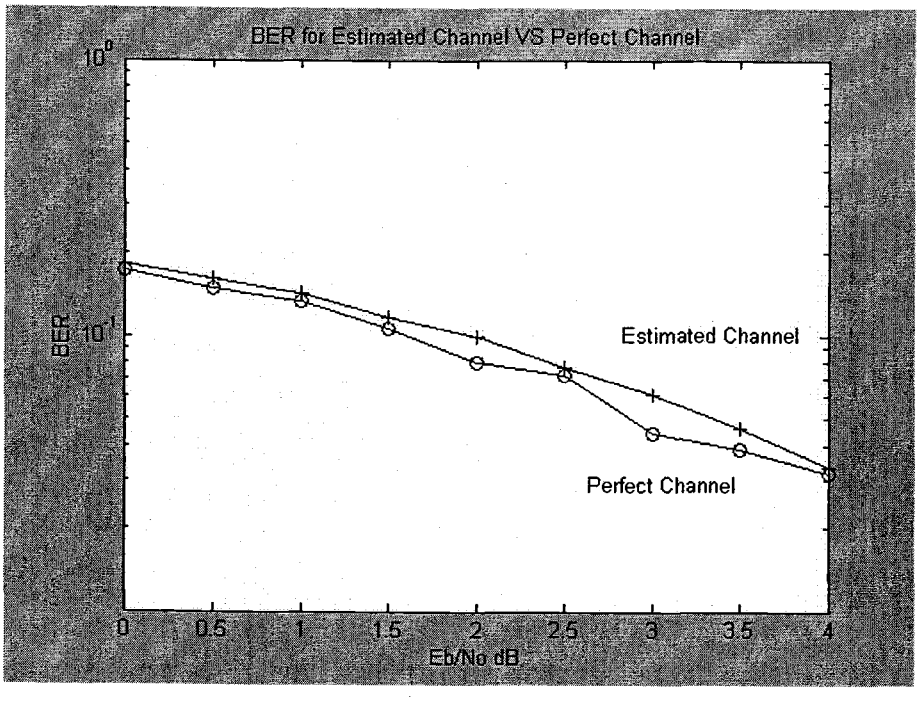

Figure 3.6: BER Comparison for Estimated and Perfect Channel Information in Slow Rayleigh Fading

In figure 3.6 we can see that by using the proposed estimation filter we can almost get the same bit error rate values as when we have perfect estimates of the channel information.

3.2.3.2 Fast Rayleigh Fading Channel Simulation Results

In fast Rayleigh fading, the transition between peaks of fading amplitude takes a shorter period of time (see figures 3.1 and 3.2). In this case, the N_c that we use is 10 and $f_d \cdot T_s$=0.02. Using the same code setting as in previous sub-section, figure 3.7 shows the turbo codes performance under fast fading in comparison with slow fading, assuming perfect knowledge of channel information.

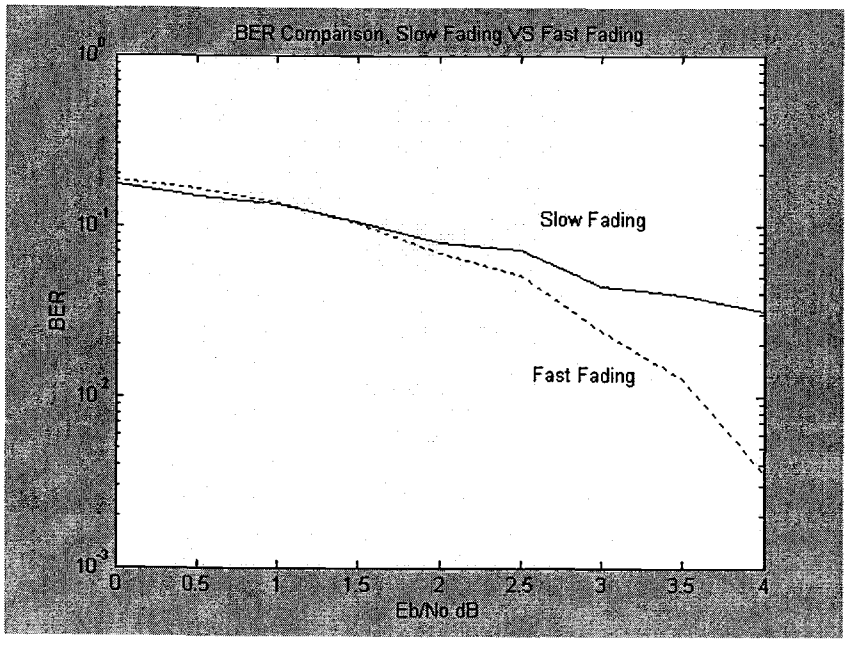

Figure 3.7: BER Comparison for Slow and Fast Fading

From this plot we can see that for low signal-to-noise ratios, turbo codes do not work well in both fading scenarios. But, as the signal-to-noise ratio increases, a fast fading channel environment allows turbo codes to work better. This can be explained as follows: in a slow fading scenario, the deep fades last a relatively long period of time. When a large percentage of transmitted frame bits are corrupted, it is less likely to get a good prediction in the output. For fast fading, on the other hand, the deep fade periods are relatively short, therefore, the deep fades are more spread and the turbo code has uncorrupted information to decode the output. So, even though the fading amplitude estimation is harder for a fast fading channel, the channel itself does not degrade the performance as severely as in a slow fading channel.

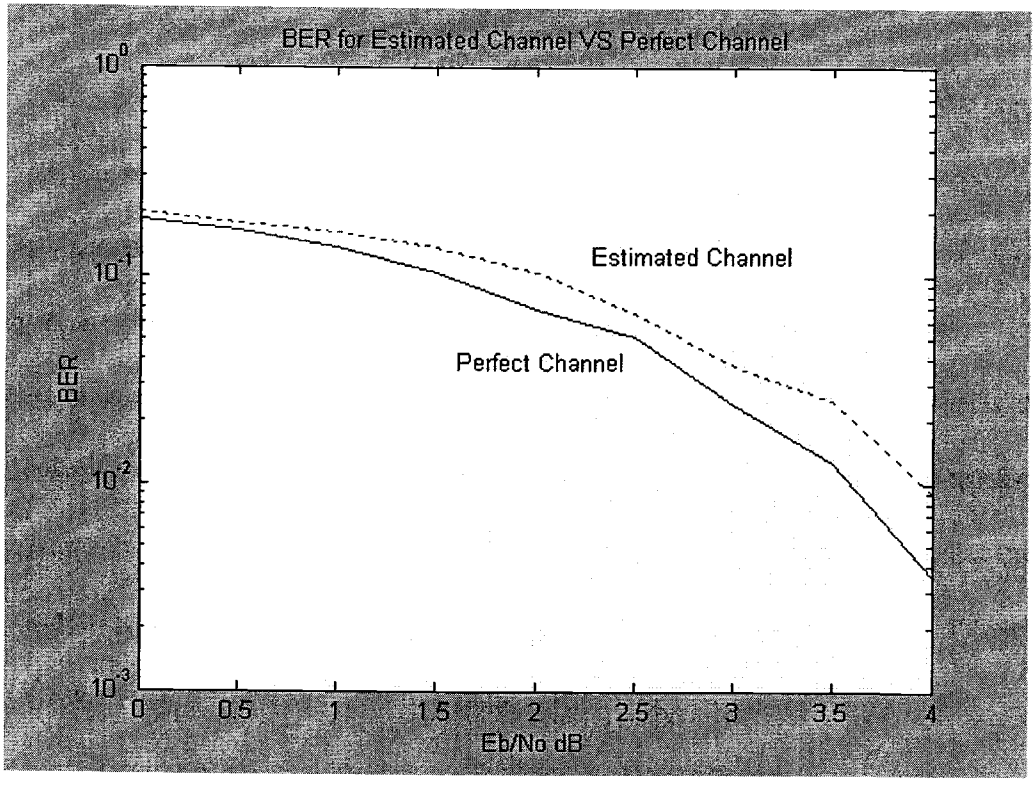

Figure 3.8: BER Comparison for Estimated and Perfect Channel Information in Fast Rayleigh Fading

Figure 3.8 shows the bit error rate comparison between the perfect knowledge of channel information and the realization using estimated channel information. Comparing figures 3.6 and 3.8, we can see that in a fast fading channel environment, estimating the channel information is harder than the case of slow fading channel. However, without channel estimation, turbo decoder performance becomes much worse. This can be seen in figure 3.9.

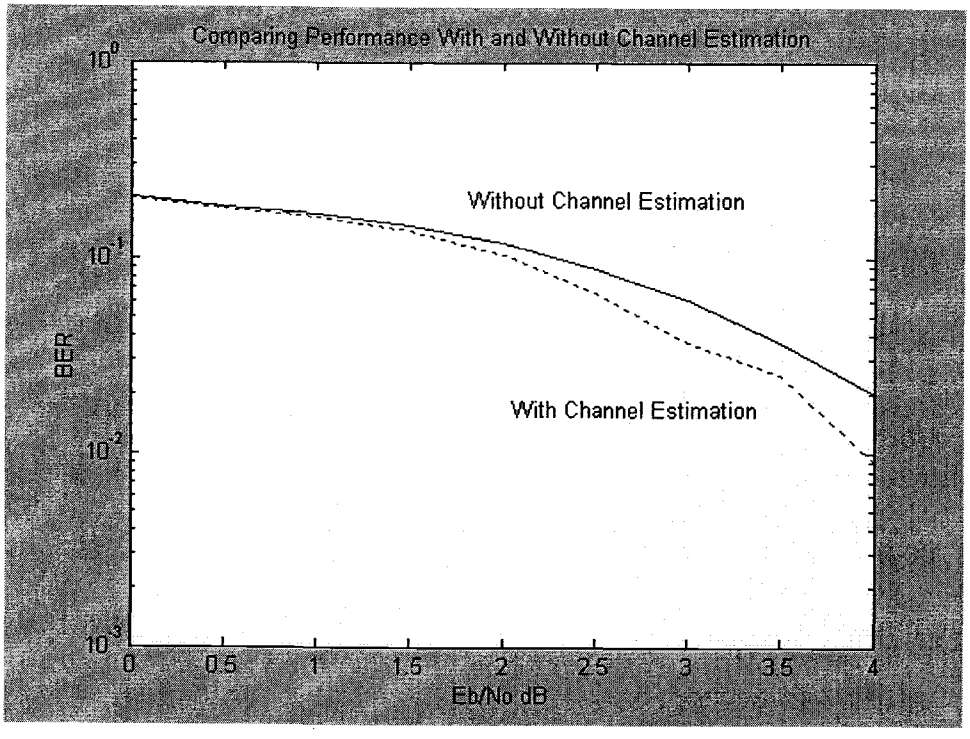

Figure 3.9: Effect of Channel Estimation in Fast Rayleigh Fading

Table of Results under Slow Flat Rayleigh Fading Channel with fd.Ts = 0.005

Eb/No (dB)	BER no Estimation	BER with Estimation	BER Perfect Estimation
0	0.1891	0.1815	0.1727
0.5	0.1684	0.1604	0.1486
1	0.1598	0.1426	0.1336
1.5	0.1329	0.1164	0.1057
2	0.1015	0.0999	0.0797
2.5	0.0974	0.0766	0.0716
3	0.0802	0.0615	0.0444
3.5	0.0541	0.0464	0.0387
4	0.044	0.0332	0.0314

Table 3.1: Table of Results under Slow Rayleigh Fading Channel

Table of Results under Fast Flat Rayleigh Fading Channel with fd.Ts = 0.02

Eb/No (dB)	BER no Estimation	BER with Estimation	BER Perfect Estimation
0	0.2041	0.2012	0.1862
0.5	0.1827	0.1777	0.1628
1	0.1677	0.1595	0.1369
1.5	0.1464	0.136	0.1032
2	0.1194	0.1031	0.0694
2.5	0.0884	0.0657	0.0512
3	0.0605	0.0366	0.0241
3.5	0.0366	0.0248	0.0127
4	0.0201	0.0089	0.0035

Table 3.2: Table of Results under Fast Rayleigh Fading Channel

3.3 DIVERSITY TECHNIQUE FOR IMPROVING PERFORMANCE

For low signal-to-noise ratios, multiple receiver antennas can, in principle, yield better performance. This kind of method is called spatial or space diversity. The proposed method uses two receiver antennas, though it can also be extended to more antennas. If the two receiver antennas are separated enough, then the received signals are described by

$$y_1(t) = a_1(t)s(t) + n_1(t), \quad (3.24)$$

$$y_2(t) = a_2(t)s(t) + n_2(t). \quad (3.25)$$

The idea is really simple, at time n, the less the fading amplitude $a_i(n),\ i = 1,2,$, the more likely errors will occur. So if we choose $a_1(n)$ when $a_1(n) > a_2(n)$ and choose $a_2(m)$ when $a_1(m) < a_2(m)$, a better bit error rate will be achieved.

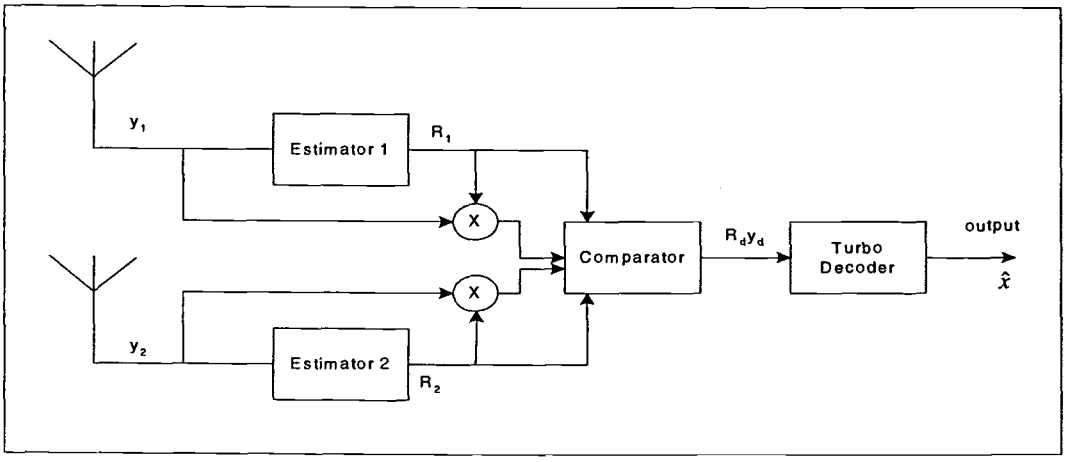

Figure 3.10: Proposed Receiver Diversity Block Diagram

The proposed process of space diversity is described in figure 3.10. As in the previous section, we still need to do some channel estimation. From the outputs of the channel estimators, we can generate the predicted channel scaling factors related to equation (2.9), i.e.,

$$R_1 = 4\hat{a}_1 \frac{Eb}{No}(rate), \quad (3.26)$$

and

$$R_2 = 4\hat{a}_2 \frac{Eb}{No}(rate). \quad (3.27)$$

At time n, the comparator compares the values of $R_1(n)$ and $R_2(n)$, and selects the one that has a larger value before passing it to the turbo decoder. So, the values coming into our turbo decoder are

$$R_d(n)y_d(n) = R_i(n)y_i(n), \quad (3.28)$$

where i = 1 if $R_1(n) > R_2(n)$ or i = 2 if $R_2(n) > R_1(n)$.

Let us now discuss the results of the simulation using the proposed space diversity technique. The following tables and plots are the simulation of turbo codes under Rayleigh fading using perfect knowledge of channel information compared with turbo codes implementing the diversity technique.

Table of Diversity simulation results for slow fading

Eb/No dB	BER without diversity	BER with diversity
0	0.1727	0.0738
0.5	0.1487	0.0503
1	0.1336	0.0316
1.5	0.1057	0.0169
2	0.0797	0.009
2.5	0.0716	0.0042

Table 3.3: Table of Diversity Simulation Results for Slow Fading

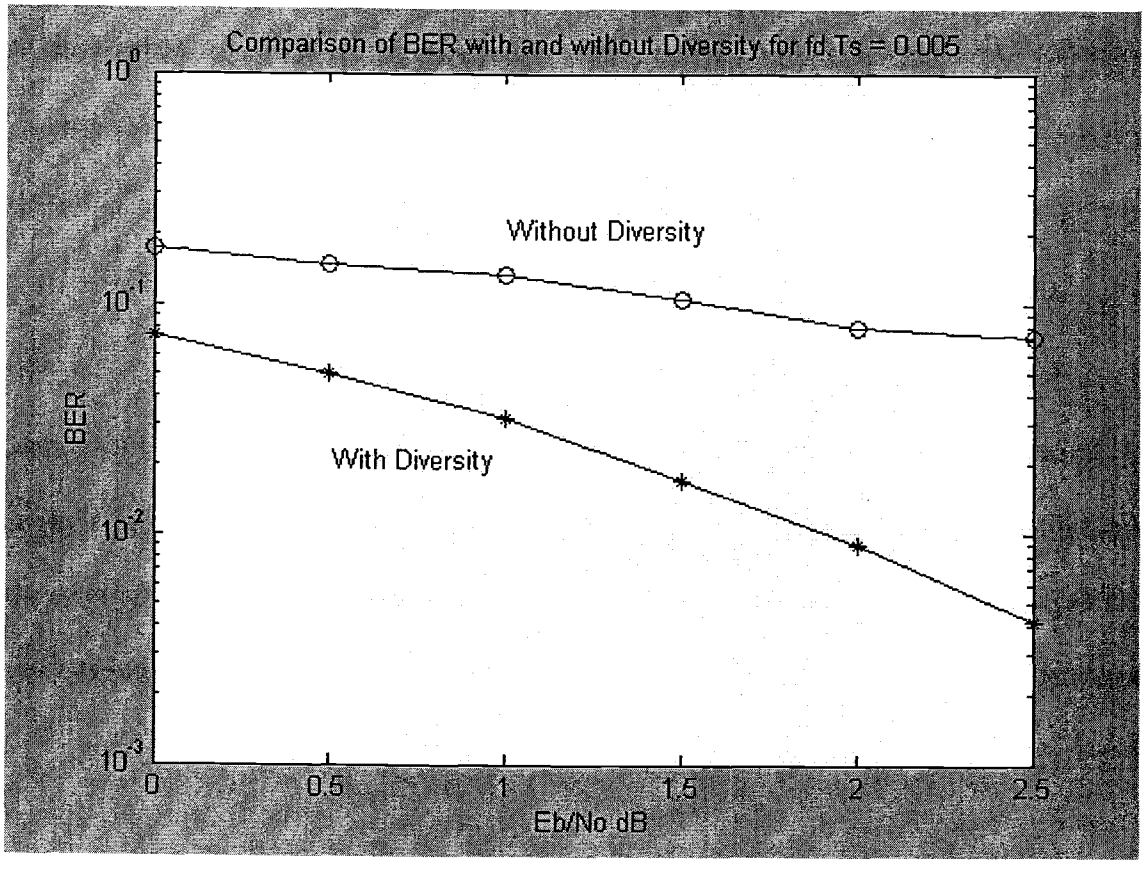

Figure 3.11: The BER Comparison with and without Diversity for Slow Rayleigh Fading

Table of Diversity simulation results for fast fading

Eb/No dB	BER without diversity	BER with diversity
0	0.1862	0.0954
0.5	0.1628	0.0571
1	0.1369	0.0246
1.5	0.1032	0.0121
2	0.0694	0.0034
2.5	0.0512	0.0006

Table 3.4: Table of Diversity Simulation Results for Fast Fading

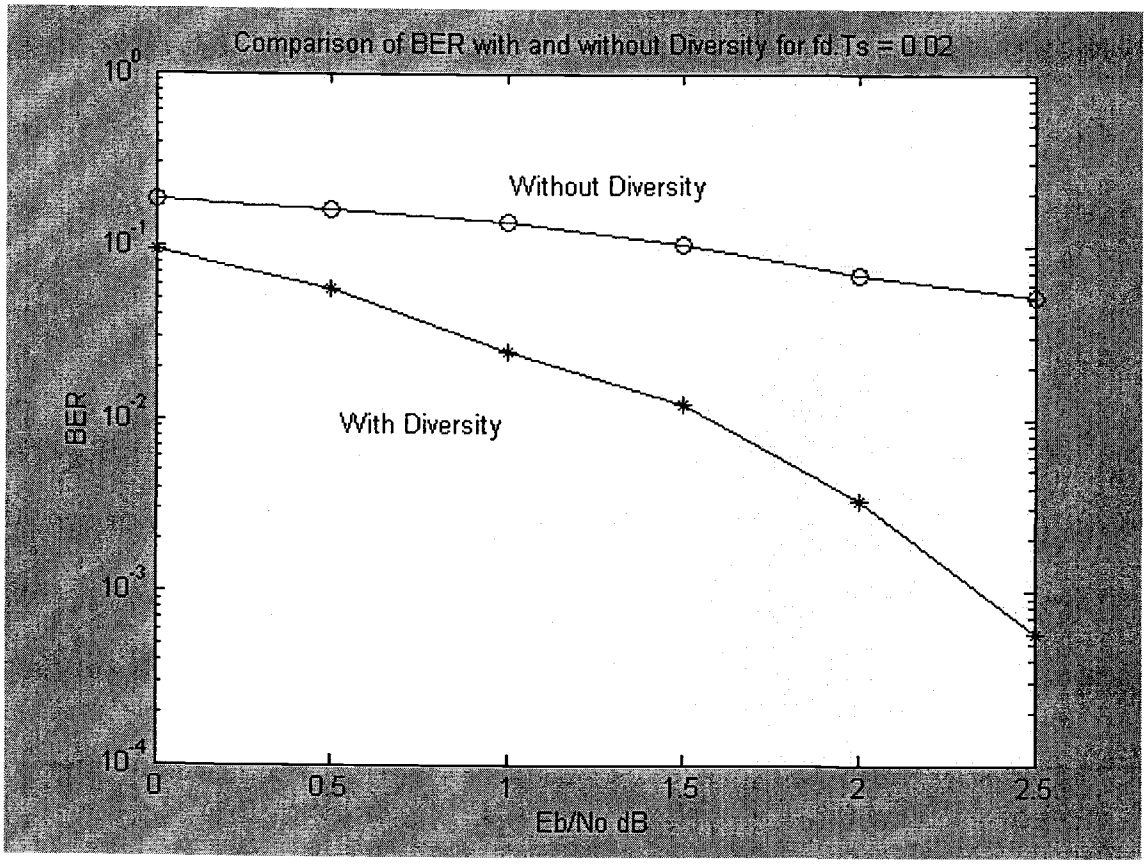

Figure 3.12: The BER Comparison with and without Diversity for Fast Rayleigh Fading

From these results, we can see that using the proposed diversity technique the performance of the turbo-coded system under flat Rayleigh fading channel can dramatically be improved. The diversity technique is not without its disadvantages. The first disadvantage is that we use more antennas in the receiver end, which is less cost effective. The second disadvantage is, that the antennas have to be separated far enough to make the fading amplitude uncorrelated to each other. Therefore, this scheme is not practical in portable devices.

3.4 CHAPTER SUMMARY

This chapter presents the performance of turbo codes in a flat Rayleigh fading channel in terms of bit error rate. Two types of flat fading environments are considered in this chapter, slow fading with $f_d \cdot T_s = 0.005$ and fast fading with $f_d \cdot T_s = 0.02$.

A channel estimation method is discussed, and the results are presented. For slower fading a larger window size is needed, and for faster fading the system restricts us to use a smaller one. Although a smaller record of observations means a worse estimate of the fading channel amplitude, the results show that overall the bit error rate is better for faster fading, so channel estimation is not as critical as in the case of slow fading.

The last part of this chapter presented an effective diversity technique to improve system performance. In signal-to-noise ratios less than 2 dB, the bit error rate is not that good. Of course, we can always improve the performance of the system by having more iterations, and a larger constraint length code. However, if these parameters are fixed, then the diversity technique can be implemented to yield much better results.

4. TURBO CODES WITH ORTHOGONAL COMPLETE COMPLEMENTARY CDMA

In this chapter, we design a Code Division Multiple Access (CDMA) system based on Orthogonal Complete Complementary (OCC) codes and augment it with turbo codes. The goal is to design and develop an alternative way of implementing a spread spectrum multi-user system that delivers improved data rate performance in a Rayleigh fading environment.

4.1 OCC-CDMA

The concept of OCC-CDMA was first introduced by Suehiro et al [33] in 2001. However, its origin can be traced back to the 60's, when Golay [34] and Turyn [35] studied pairs of binary complementary codes whose autocorrelation function is zero for all even shifts except the zero shift. Then, Suehiro in [36, 37] extended the research which resulted in Complete Complementary (CC) code families whose autocorrelation function is zero for all even and odd shifts except the zero shift and whose cross-correlation function for any pair is zero for all possible shifts.

4.1.1 OCC-CDMA Codes Selection

In OCC-CDMA, a "flock" of jointly orthogonal element codes are used instead of a single code as in traditional CDMA codes.

Element Code Length L = 4		Element Code Length L = 16	
Flock 1	A0: + + + -	Flock 1	A0: + + + + + - + - + + - - + - - +
			A1: + - + - + + + + + - - + + + - -
			A2: + + - - + - - + + + + + + - + -
			A3: + - - + + + - - + - + - + + + +
	A1: + - + +	Flock 2	B0: + + + + - + - + + + - - - + + -
			B1: + - + - - - - - + - - + - - + +
			B2: + + - - - + + - + + + + - + - +
			B3: + - - + - - + + + - + - - - - -
Flock 2	B0: + + - +	Flock 3	C0: + + + + + - + - - - + + - + + -
			C1: + - + - + + + + - + + - - - + +
			C2: + + - - + - - + - - - - - + - +
			C3: + - - + + + - - - + - + - - - -
	B1: + - - -	Flock 4	D0: + + + + - + - + - - + + + - - +
			D1: + - + - - - - - - + + - + + - -
			D2: + + - - - + + - - - - - + - + -
			D3: + - - + - - + + - + - + + + + +

Table 4.1: Two Examples of Complete Complementary Codes with Element Code Lengths L = 4 and L = 16

Table 4.1 [33] shows how element codes, for a specified code length, are grouped together to become a flock. For a code length L = 4, we can see that there are four element codes (A0, A1, B0, B1). A0 and A1 form the first flock, B0 and B1 become the second flock and each flock correspond to one user. So, in this case, we can only support 2 users.

The advantage of this special selection of codes is that it offers zero autocorrelation function for both even and odd shift except for zero, the

multiple access interference (MAI) is kept minimum even for an offset stacked scheme showed in figure 4.1 [33]. This is an illustration where we can see that except for no shift and the signature code match with the local code correlator, this codes selection will allow us to have autocorrelation peak and MAI free in a noiseless implementation.

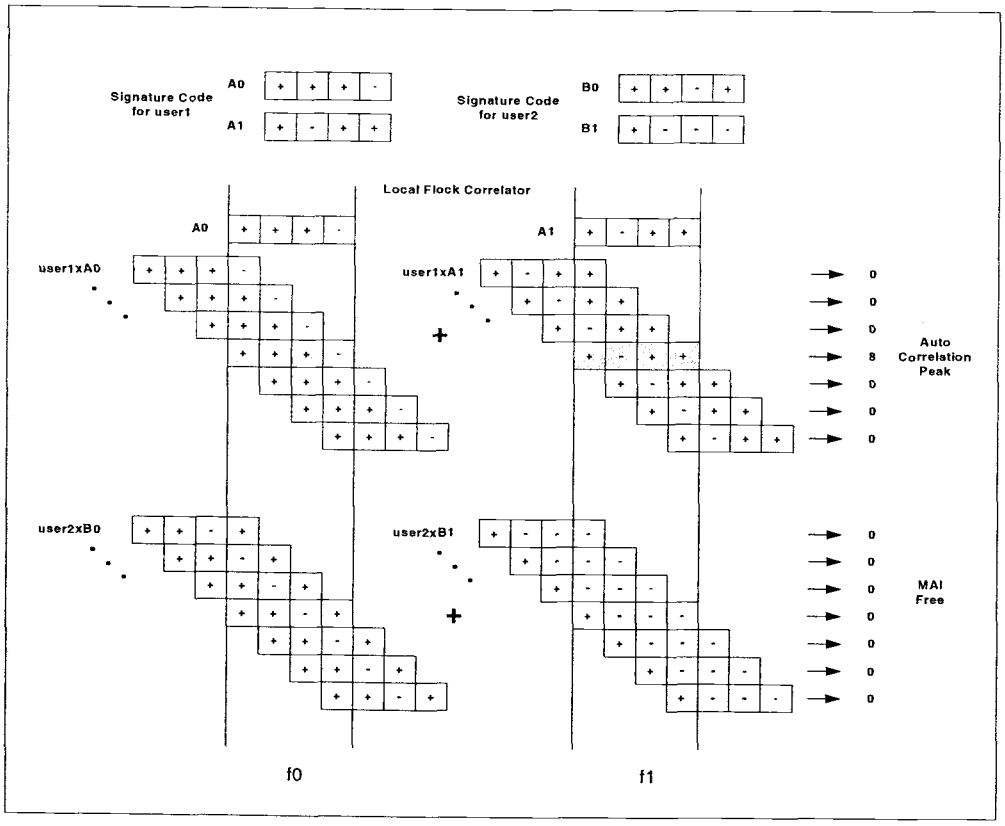

Figure 4.1: Correlation process in the decoder for L = 4 where user1 is the intended message to be decode.

4.1.2 OCC-CDMA Encoding and Decoding

In order to explain the architecture of OCC-CDMA thoroughly, an example with code length L = 4 is used for the remainder of this chapter. Let us begin with the generation of the OCC-CDMA outputs. The first step is the creation of the intermediate output. This is shown in figure 4.2.

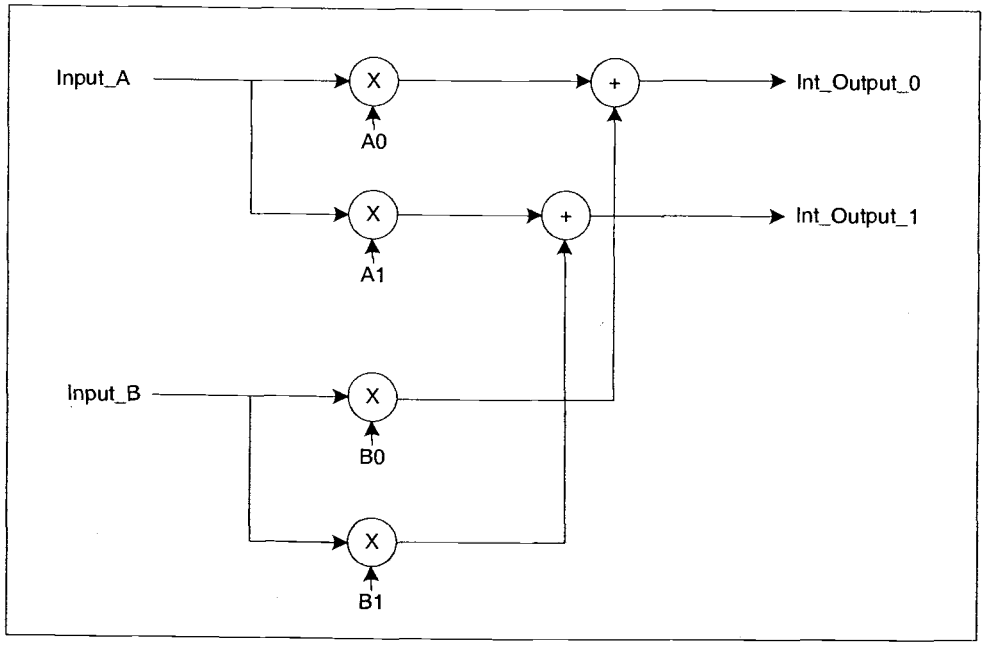

Figure 4.2: Coding of Intermediate Output

Input_A and Input_B in figure 4.2 correspond to the input symbols of user A and user B. If the symbols are implemented using polar NRZ (1 and –1) then the intermediate output can be described in figure 4.3.

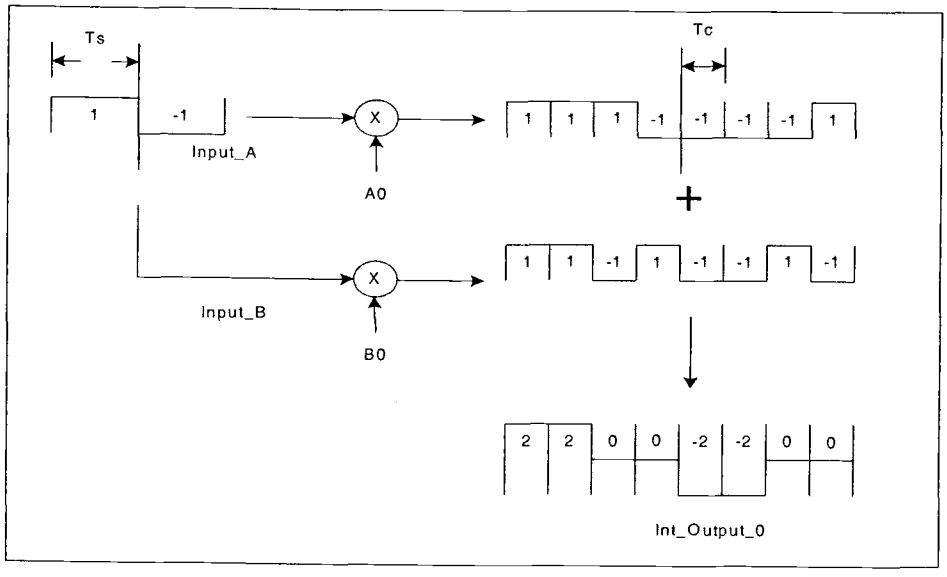

Figure 4.3: Generation of Intermediate Output Example Case

The second step of OCC-CDMA encoding is the offset stacked modulation scheme. The process is very similar to pipeline processing of computers. Unlike conventional CDMA which concatenates the produced chip every 4 Tc for code length L = 4, OCC-CDMA uses

$$chip\_out = c(m) = \sum_{k=1}^{L} I(k, m-k+1), \quad (4.1)$$

where $I(a,b)$ is the intermediate output produced by symbols at time b and a-th component of signature codes, and for b < 1, $I(a,b)$=0. Figure 4.4 will clarify this notation (and the output created in figure 4.3).

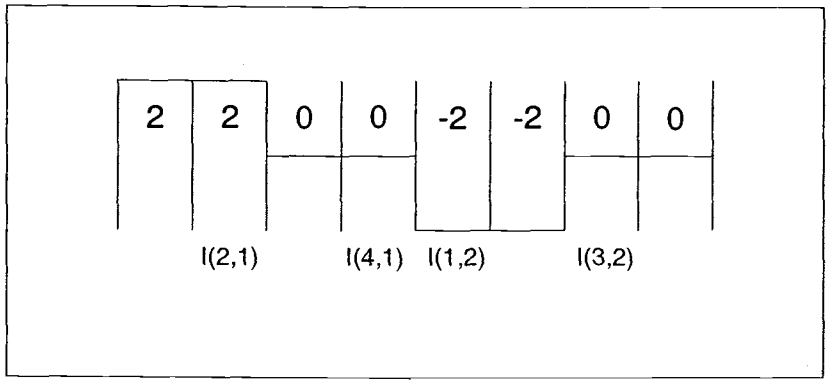

Figure 4.4: The Output of Example Given in Figure 4.3

In figure 4.5, the offset stacked modulation scheme is illustrated by adding up the values in the same column to produce c(m).

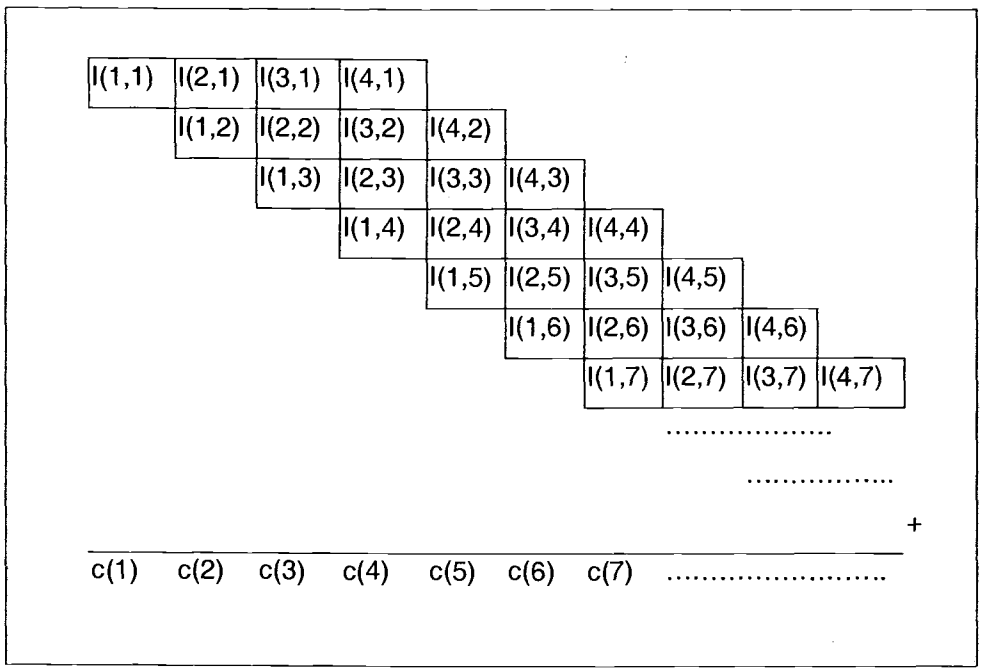

Figure 4.5: The Offset Stacked Modulation Scheme

After the offset stacked modulation scheme for both intermediate outputs is performed, we have c_0 and c_1 ready as the output of this OCC-CDMA. Notice that we have two parallel outputs that have to be sent and will arrive at the receiver at the same time. c_0 and c_1, in this case, are sent with two different carrier frequencies f0 and f1. Hence, OCC-CDMA is a multicarrier CDMA system.

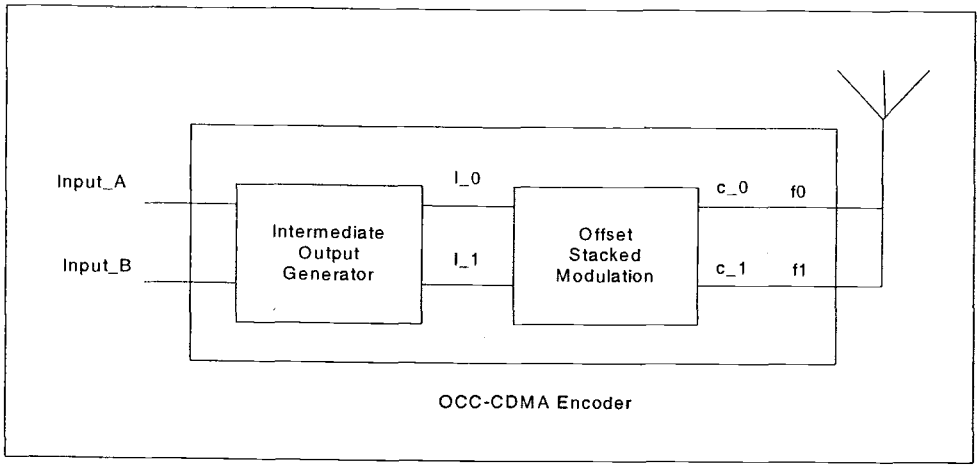

Figure 4.6: OCC-CDMA Encoder

The decoder implementation of OCC-CDMA is also relatively simple. The easiest model of the OCC-CDMA decoder can be explained using figure 4.7. This decoder is a simplified model of an OCC-CDMA decoder presented in [33], which later on is used in our simulation.

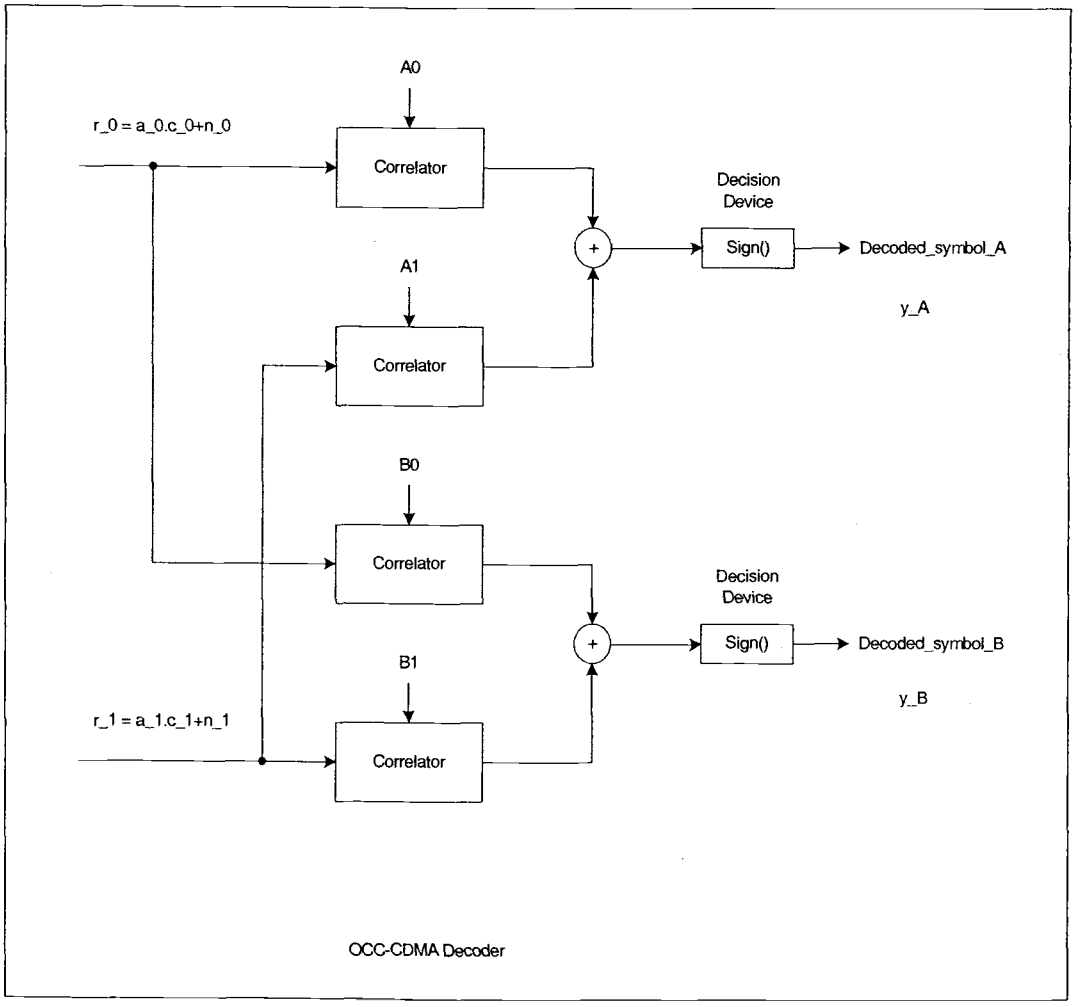

Figure 4.7: OCC-CDMA Decoder

Note that for a soft decision output, which is used as the turbo decoder input, we simply just remove the decision device in the above model.

4.1.3 OCC-CDMA Compared to Conventional CDMA

Next, let us look at what the difference between OCC-CDMA and conventional CDMA is, and what the advantages and disadvantages of this system are. In conventional CDMA, the construction of the spreading chips is very different from that of OCC-CDMA. Specifically, in conventional CDMA the codes are not orthogonal for every shift, therefore, it is impossible to implement the offset stacked modulation like the one in OCC-CDMA because of multiple access interference. So, for a code length L = 4, the next 4 chips generated from the next symbol are concatenated after 4 Tc, or after the fourth chip generated from the present symbol. Also, as far as the number of users is concerned, conventional CDMA can support up to 4 users.

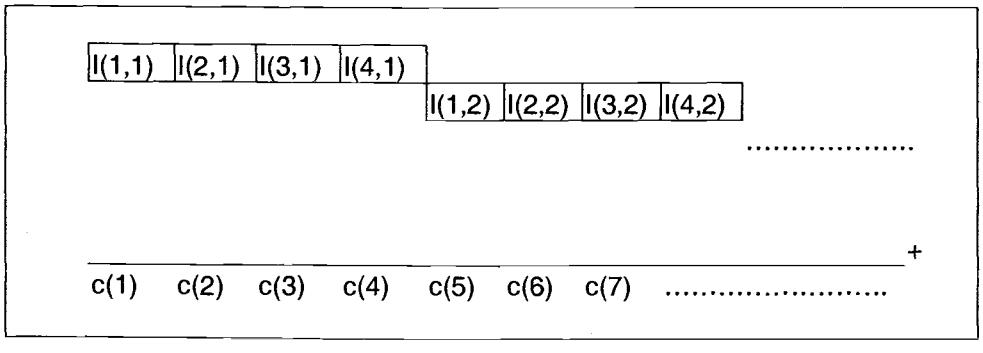

Figure 4.8: Chip Generation for Conventional CDMA

Let us assume that for this particular conventional CDMA system we have 4 users. For every 4 chips received, we can decode back 4 transmitted symbols. If we compared this to the OCC-CDMA, because of the offset stack modulation scheme, after the first 4Tc, the next received chip allows us to decode 2 users. This means that if the transmission frame size is large enough, the data rate of the OCC-CDMA will be twice as fast as the conventional CDMA system. As we know, one of the drawbacks of a turbo-coded system is its low data rate. So, by implementing this OCC-CDMA in our turbo-coded system, we can have a faster data rate. Because of this property, an OCC-CDMA system can improve the data rate without even shortening the chip period Tc.

Another advantage of OCC-CDMA, as stated in [33], is this system is very suitable for a multi-rate signal transmission. Although this is beyond the scope of this thesis, it is a very important topic, since most of the time the communication between the base and the mobile stations does not have the same maximum data rate. So, being able to change the offset in the offset stacked modulation scheme from Tc to 2Tc, for example, will change the data rate of transmission without having a complicated rate-matching algorithm like in a conventional CDMA system.

For the remainder of this chapter, we concentrate on improving the data rate of a turbo-coded OCC-CDMA system. By using OCC-CDMA, which we already know has the property of improving the data rate, we can

double it. In figure 4.9, we treat the users to be the outputs of serial to parallel converter.

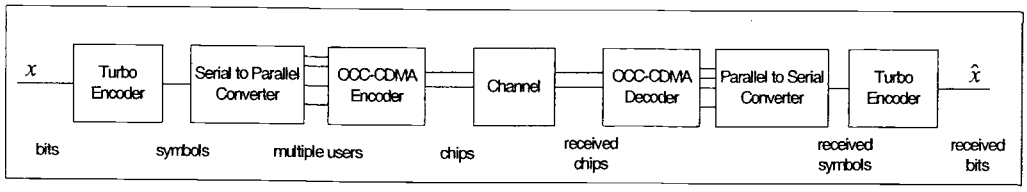

Figure 4.9: The Overall Block Diagram for our OCC-CDMA System

Simple analysis will show that this is not a perfect solution. In fact, such an implementation means that a trade-off has been made, since in this proposed system we send information in parallel using two different carrier frequencies. This means higher cost in terms of frequency space in RF. However, the usage of two different carrier frequencies may help us if we have frequency selective interference in the channel.

4.2 TURBO-CODED OCC-CDMA IN AWGN

Before we continue with the evaluation of the proposed system performance, we need to look at the turbo codes noise level comparison with and without OCC-CDMA. This is crucial because failing to account for the noise correctly will make the results interpretation totally wrong.

From equation (2.8), we have

$$\sigma = \sqrt{\frac{1}{2\frac{Eb}{No}rate}}.$$

Here, σ^2, the variance of the noise, relates to the average power of the noise. Because in a CDMA system we have a processing gain which is related to the change in average power of the chips compared to the average power of the symbols, and, if Tc<Ts, then an adjustment in σ has to be made. So, the modified version of σ is

$$\sigma_{CDMA} = \sqrt{\frac{PG}{2\frac{Eb}{No}rate}}\sqrt{\frac{Ts}{Tc}}, \quad (4.2)$$

The processing gain (PG) for OCC-CDMA is presented in the table 4.2 given below.

Element code length ($L=4^n$)		4	16	64	256	1024
PG	$L(\sqrt{L})$	8	64	512	4096	32768
Flock size	$\sqrt{L}$	2	4	8	16	32

Table 4.2: Processing Gain Table for OCC-CDMA

For the simulations in this chapter, the element code length L=4, and Tc=Ts. If less Tc is used, faster data rate can be achieved. However, more error will be introduced because $\sqrt{\frac{Ts}{Tc}}$ in equation (4.2) will be larger, and

more bandwidth will be used. The parameters of the turbo codes used here are very similar to those used in chapter 3. The turbo codes are punctured with rate = ½. The constraint length is 3. The size of turbo interleaver is 400 and 4 decoding iterations are implemented.

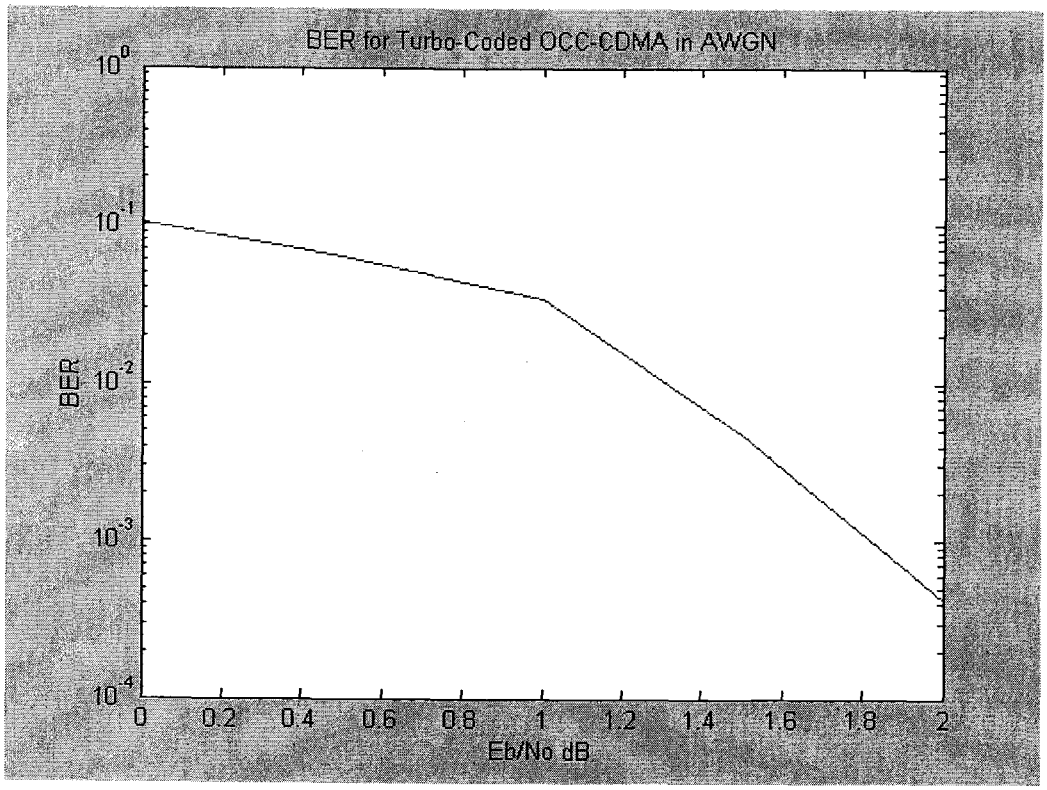

Figure 4.10: BER for Turbo-Coded OCC-CDMA in AWGN

Eb/No dB	BER without OCC-CDMA	BER with OCC-CDMA
0	0.113	0.1032
0.5	0.0545	0.0632
1	0.0278	0.0337
1.5	0.0053	0.0047
2	0.0008	0.0004

Table 4.3: Turbo-Coded OCC-CDMA in AWGN Simulation Results

Looking at the results presented in table 4.3, we can see that the bit error rate (BER) of the turbo codes with OCC-CDMA does not degrade. So by trading off the frequency usage, we gain a faster data rate without sacrificing the bit error rate in an AWGN channel.

4.3 TURBO-CODED OCC-CDMA IN FLAT RAYLEIGH FADING CHANNEL

In dealing with a flat Rayleigh fading channel, we expect to get a reduction in performance in terms of bit error rate. In this section we will discuss the proposed turbo-coded OCC-CDMA in the presence of flat Rayleigh fading. The simulations for flat Rayleigh fading use the same settings as in chapter 3, i.e., $f_d \cdot T_s = 0.005$ for slow fading and $f_d \cdot T_s = 0.02$ for fast fading, and as far as the turbo-coded OCC-CDMA system is concerned, the same settings as in the previous section are used.

Eb/No dB	AWGN BER with OCC-CDMA	Rayleigh Slow Fading BER with OCC-CDMA	Rayleigh Fast Fading BER with OCC-CDMA
0	0.1032	0.1805	0.1974
0.5	0.0632	0.164	0.1821
1	0.0337	0.1457	0.1522
1.5	0.0047	0.135	0.1358
2	0.0004	0.1154	0.1149

Table 4.4: Turbo-Coded OCC-CDMA in Flat Rayleigh Fading Simulation Results

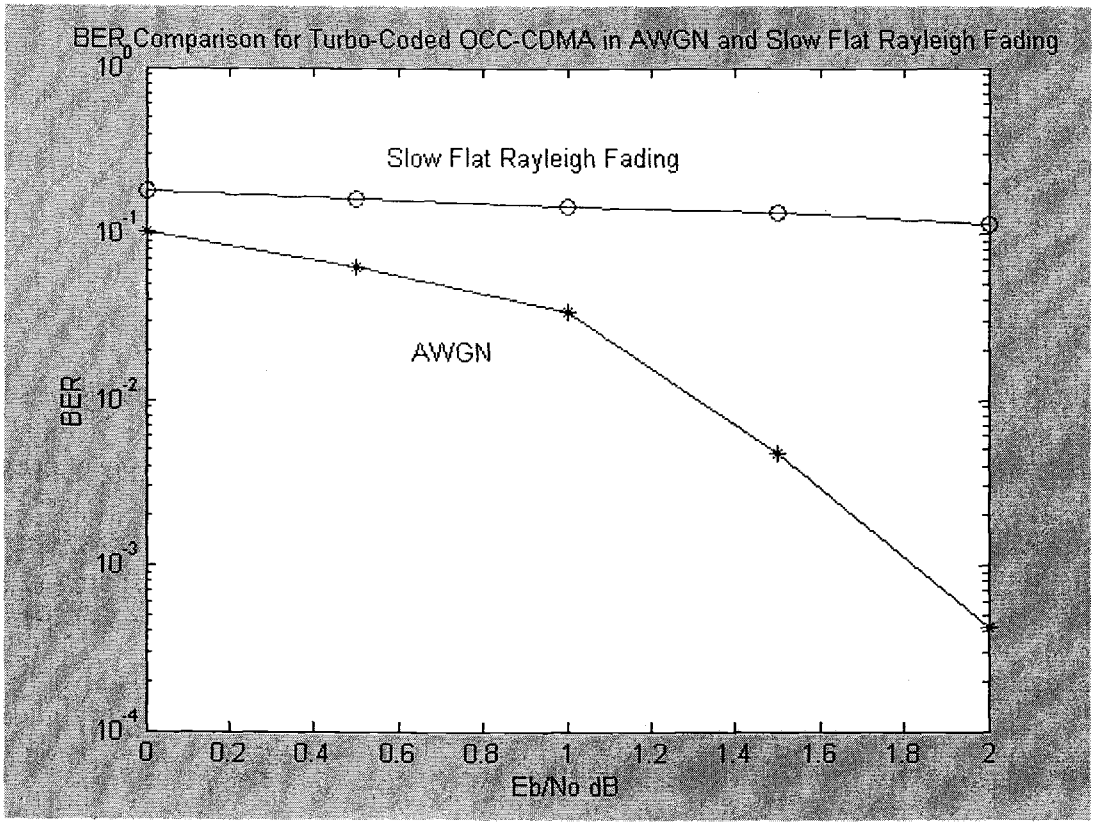

Figure 4.11: BER Comparison for Turbo-Coded OCC-CDMA in AWGN and Slow Flat Rayleigh Fading

From the table 4.4 and figure 4.11 above, we can see that without any modification in our system design, it will not work well in flat Rayleigh fading environments, especially in low signal-to-noise ratios.

In Eb/No less than 3 dB, the resulting BER for the slow fading channel is better BER than fast fading channel, but as the signal-to-noise ratio improves, we can see that we have a better BER in a fast fading channel. This can be seen in the results of the simulation shown in table 4.5.

Eb/No dB	Rayleigh Slow Fading BER with OCC-CDMA	Rayleigh Fast Fading BER with OCC-CDMA
2.5	0.0829	0.0844
3	0.0776	0.0595
3.5	0.0536	0.0432
4	0.0462	0.0217
4.5	0.0333	0.0113

Table 4.5: Turbo-Coded OCC-CDMA in Flat Rayleigh Fading Simulation Results for Higher Eb/No (dB) Values

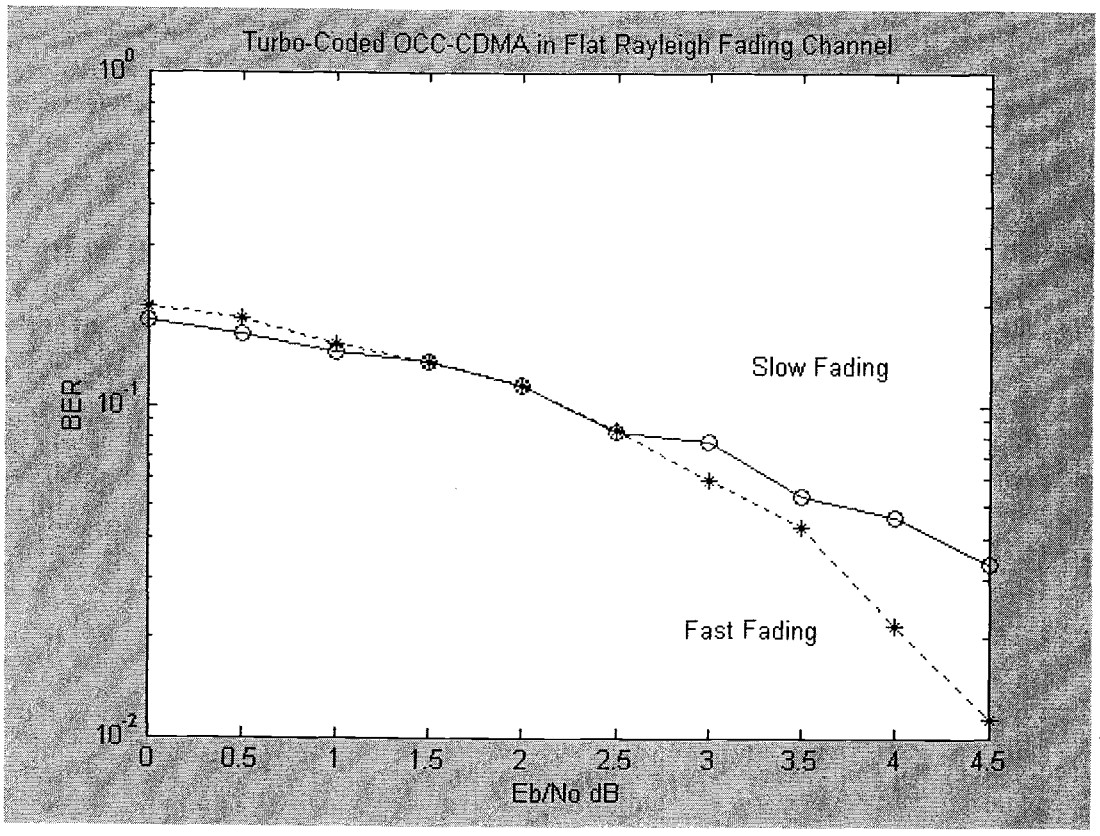

Figure 4.12: BER Comparison for Turbo-Coded OCC-CDMA in Fast and Slow Fading

Obviously, in fading channel environments, we need to do something to improve the bit error rate for the system. The proposed solution to this problem is a spatial diversity technique. The implementation of diversity in a turbo-coded OCC-CDMA is a little bit different compared to conventional turbo codes. Figure 4.13 is the block diagram of the proposed diversity technique for our turbo-coded OCC-CDMA system.

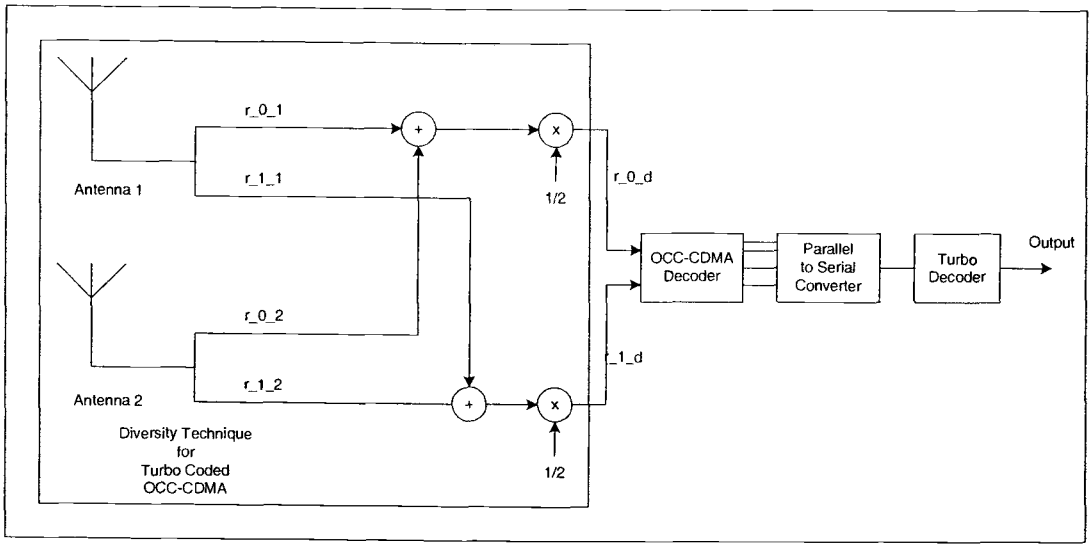

Figure 4.13: Proposed Diversity Technique for Turbo-Coded OCC-CDMA

Here, there is no channel estimation implementation. This is due to the fact that for an OCC-CDMA coded system, once the chips are decoded in OCC-CDMA decoder, there is no direct one-to-one relation between symbols and fading channel amplitudes.

What we do in this diversity technique is just averaging the received signals from the multiple receiver antennas. So, the inputs into the OCC-CDMA receiver are

$$r\_i\_d = \frac{1}{N}\sum_{k=0}^{N-1} r\_i\_k, \quad (4.3)$$

where N is the number of receiver antenna used and i is the index of the chip out (c_i). The basic idea of averaging these received signals is somewhat simple. By averaging, we are making the variance of the received signal smaller, which translates to average fading amplitude closer to one, and also a smaller noise variance. The results of the simulation are given in the table 4.6, figures 4.14 and 4.15.

Eb/No (dB)	WITH DIVERSITY (2 ANTENNAS)	
	Rayleigh Slow Fading BER with OCC-CDMA	Rayleigh Fast Fading BER with OCC-CDMA
0	0.0359	0.0306
0.5	0.0234	0.009
1	0.0149	0.0031
1.5	0.0067	0.0009
2	0.0026	0.0002

Table 4.6: The Turbo-Coded OCC-CDMA Spatial Diversity Results

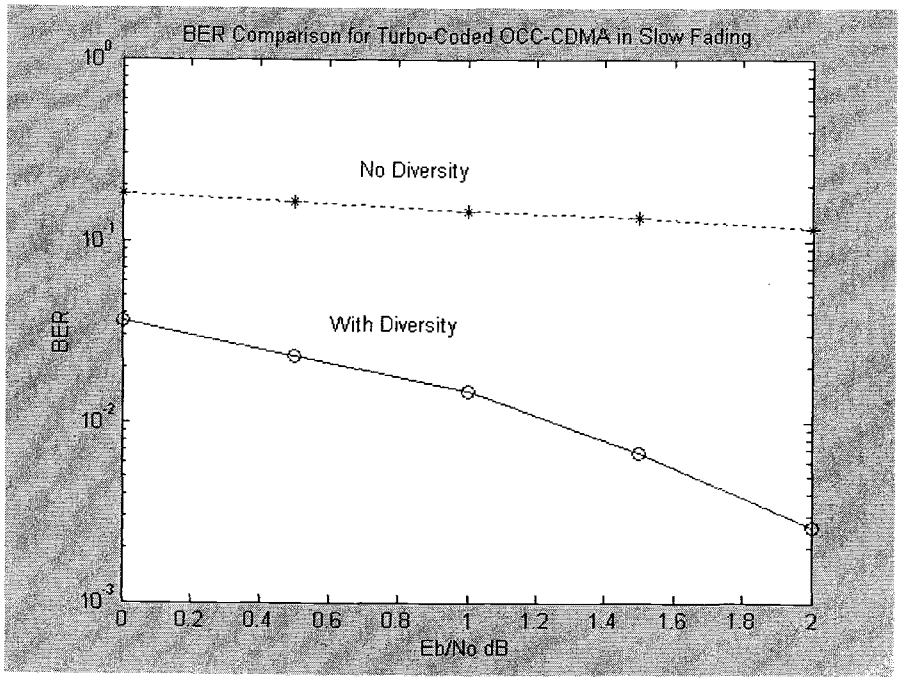

Figure 4.14: Turbo-Coded OCC-CDMA in Slow Fading with and without Diversity

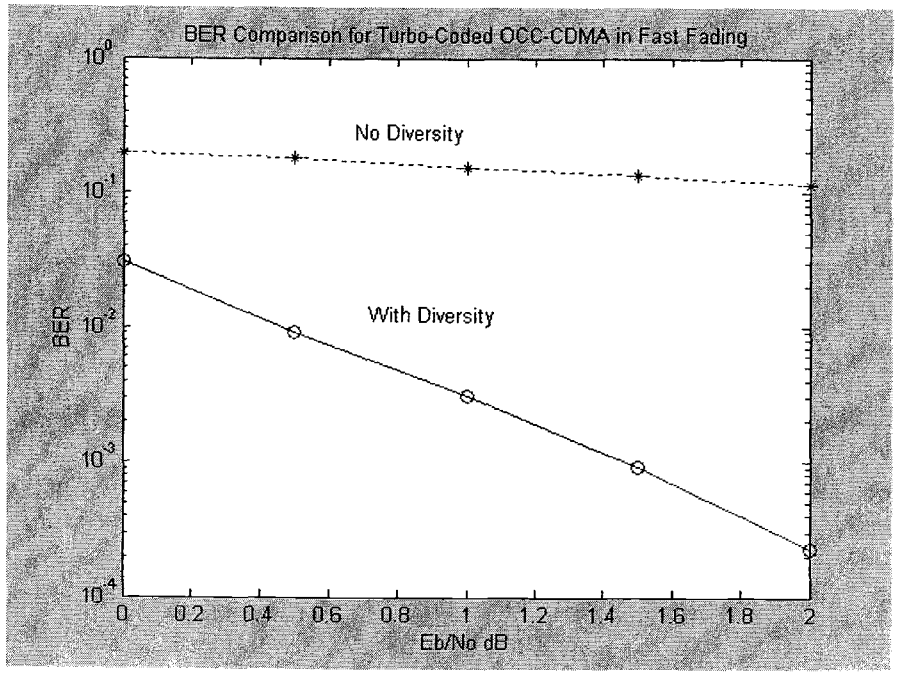

Figure 4.15: Turbo-Coded OCC-CDMA in Fast Fading with and without Diversity

From the simulation results, we can conclude that even in a very hostile environment, we can still use the proposed turbo-coded OCC-CDMA with spatial diversity and get a very good result.

Lastly, the inventors of OCC-CDMA in their paper [33] suggest that the chip signals be sent using two different carrier frequencies. But, for code length L = 4, it is convenient to use quadrature multiplexing system to transmit these signals, without sacrificing a significant loss in performance. This way, we do not have to use multi-carrier system. Figure 4.16 is the time domain representation of the suggested quadrature multiplexing system.

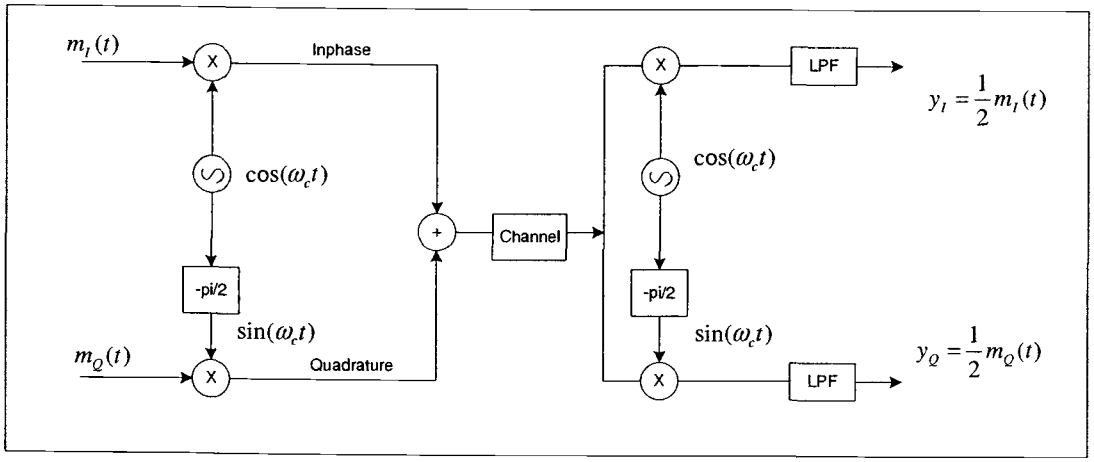

Figure 4.16: Quadrature Multiplexing System

4.4 CHAPTER SUMMARY

In this chapter, we introduced a new type of CDMA spread spectrum system called OCC-CDMA implementation using turbo codes. By using orthogonal sets of specially selected codes, this type of CDMA introduces very low MAI and faster achievable data rate with the same chip period Tc.

The second part of this chapter discussed the performance of a turbo-coded OCC-CDMA system in an AWGN channel. From the results of the simulation, we can see that the system performance does not deteriorate in an AWGN channel.

The last part of this chapter was devoted to the evaluation of the performance of a turbo-coded OCC-CDMA system in a flat Rayleigh fading channel. Without any modification of our system, the resulting bit error rates were found to be unacceptable, especially in very low signal-to-noise ratios. A spatial diversity scheme was proposed to deal with this problem and simulation results were presented.

An alternative way of implementing the turbo-coded OCC-CDMA system is using Quadrature Multiplexing System instead of multi-carrier transmission. By doing so, we just use the inphase and quadrature properties for transmitting signals of turbo-coded OCC-CDMA with code length L = 4, and theoretically will not degrade the performance in terms of bit error rates.

Because a faster data rate is allowed using this proposed OCC-CDMA system due to its bandwidth efficiency, in some hostile environments we can even use 1/3 rate turbo codes to lower bit error rate and still have an acceptable system data rate.

5. CONCLUSION

In this thesis, the concepts of parallel concatenated coding and iterative turbo decoding have been applied to a new CDMA spread spectrum transceiver architecture called OCC-CDMA in both AWGN and fading mobile wireless channels. The performance of this transceiver was tested using computer simulations of the system in slow and fast fading environments. Performance of the transceiver was further improved by introducing spatial diversity.

As shown in chapter 2, for an AWGN channel, turbo codes can perform effectively even in very low signal-to-noise ratios. In a wireless communication channel, as presented in chapter 3, the simulation results showed that without any modification to our turbo-coded system, the performance in terms of bit error rate degrades significantly. The solutions to the problems faced by turbo codes in fading channels are channel estimation and diversity. Under a moderate signal-to-noise ratio, channel estimation for the fading channel amplitude should be enough to mitigate the high bit error rates. For the systems considered here, to get low bit error rates in low signal-to-noise ratios, space diversity techniques in the receiver side can improve the bit error rate performance considerably. In chapter 3 we presented the simulation results for this proposed solution. The disadvantages of using this diversity technique are high cost and difficult

realization in portable devices. Having two receiver antennas compared to just one, will increase the production cost, and it is more complicated to design the architecture of multiple antennas in a portable device because we need to have uncorrelated fading amplitudes between the received signals. Generally, the antennas have to be separated enough to get uncorrelated received fading amplitudes.

Another issue in turbo coding is the slower data rate due to the introduced coding redundancy. Although we can introduce code puncturing, which will decrease the code rate, other modifications may be implemented to improve the speed of the system. In chapter 4, the proposed solution to this problem was the introduction of OCC-CDMA to our turbo codes. OCC spreading codes yield minimum MAI, have a unique modulation architecture, and permit the turbo-coded OCC-CDMA systems to generate faster data transmission rates and have better bandwidth efficiency. The simulation results for this proposed system in an AWGN channel environment was shown to be very close to that of turbo codes used in conventional CDMA. This leads us to conclude that turbo-coded OCC-CDMA systems do not suffer much bit error rate degradation in an AWGN channel. In a flat Rayleigh fading channel, further diversity techniques in the receiver side have to be implemented to get acceptable results.

In future research work, the latency of the turbo-coded system may be improved. Depending on the environment, we may want to reduce or

increase the number or iterations. If we could reduce the number of iterations by tracking the characteristic of the channel, then we would reduce the decoding time. Also, in this thesis we assumed that we can track the phase shift caused by the fading channel, however, in a very low signal-to-noise ratio situation, we might not be able to track the phase of the transmitted signal using conventional techniques such as phase locked loops. This could also be a very interesting research topic. Finally, the antennas architecture for the proposed spatial diversity technique is not investigated in this thesis. Therefore, this could be another research topic worth studying.

BIBLIOGRAPHY

[1] C. E. Shannon, "A mathematical theory of communication," Bell Sys. Tech. J., vol 27, pp. 379-423 and 623-656, 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, " Near Shannon Limit Error-Correcting coding and Decoding: Turbo Codes," in *IEEE Proc. of the Int. Conf. On Communications*, Geneva, Switzerland, May 1993 (ICC '93), pp. 1064-1070.

[3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. on Commununications*, vol. 44, pp. 1261-1271, Oct. 1996.

[4] D. Divsalar and F. Pollara, "Low-rate turbo codes for deep-space communications," in *Proc., IEEE Int. Symp. On Information Theory*, 1995, p. 35.

[5] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communications," JPL TDA Progress Report, vol. 42, May 1995.

[6] J. D. Andersen, "Turbo coding for deep space applications," in *Proc. IEEE Int. Symp. on Information Theory*, (Whistler, Canada), Sept. 1995, p.36.

[7] D. Divsalar and F. Pollara, "Multiple turbo codes," in *Proc. IEEE MILCOM*, Nov. 1995, pp. 279-285.

[8] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," JPL TDA Progress Report, vol. 42, Aug. 1996.

[9] S. Benedetto and G. Montorsi, "Generalized concatenated codes with interleavers," in *Proc., Int. Symp. On Turbo Codes and Related Topics*, (Brest, France), Sept. 1997, pp. 32-39.

[10] M. Penicaud, A. Yongacoglu, and J. Y. Choinard, "Iterative decoding of rate adaptive codes for mobile satellite fading channels," in *Proc. Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), Sept. 1997, pp. 231-234.

[11] H. Koorapaty, S. Chennakeshu, Y.P. Wang, and R. Ramesh, "MAP decoding for satellite channels," in *Proc., IEEE Veh. Tech. Conf.*, 1996, pp. 477-481.

[12] J. C. Morakis and W. H. Miller, "Coding techniques under study at NASA," in *Proc. IEEE Aerospace Conf.*, 1997, pp. 559-565.

[13] M. C. Valenti and B. D. Woerner, "Iterative Channel Estimation and Decoding of Pilot Symbol Assisted Turbo Codes over Flat-fading Channels," *IEEE Journal on Selected Areas in Comm.*, vol. 19, pp. 1697-1705, Sept. 2001.

[14] M. C. Valenti and B. D. Woerner, "Performance of turbo codes in interleaved flat fading channels with estimated channel state information," in *Proc. IEEE Veh. Tech. Conf., (Ottawa, Canada)*, May 1998, pp. 66-70.

[15] E. K. Hall and S. G. Wilson, "Design and Analysis of Turbo Codes on Rayleigh Fading Channel," *IEEE Journal on Selected Areas Comm.*, vol. 16, pp. 160-174, Feb. 1998.

[16] E. K. Hall and S. G. Wilson, "Design and performance analysis of turbo codes on Rayleigh fading channels," in *Proc., CISS*, (Princeton, NJ), Mar. 1996.

[17] H. Koorapaty, Y. P. E. Wang, and K. Balanchandran, "Performance of turbo codes with short frame sizes," in *Proc. IEEE Veh. Tech. Conf.*, 1997, pp. 329-333.

[18] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. IEEE Int. Conf. On Comm.*, May 1995, pp. 54-59.

[19] B. Melis and G. Romano, "Application of turbo codes in the up-link of a DS-CDMA system," in *Proc. Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), Sept. 1997, pp. 243-246.

[20] S. Crozier, J. Lodge, P.Guinand, and A. Hunt, "Performance of Turbo codes with relative prime and golden interleaving strategies," in *Proc. Sixth International Mobile Satellite Conference (IMSC '99)*, Ottawa, Canada, June 1999, pp. 269-275.

[21] M. C. Reed and S. S. Pietrobon, "Turbo-code termination schemes and a novel alternative for short frames," in *Proc. IEEE PIMRC*, 1996, pp. 354-358.

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

[23] J. Haganauer, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429-445, Mar. 1996.

[24] P. Robertson, "Illuminating the structure of code and decoder for parallel concatenated recursive systematic (turbo) codes," in *Proc. GLOBECOM '94*, Dec. 1994, pp. 1298-1303.

[25] J. A. Erfanian, S. Pasupathy, and G. Gulack, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Comm.*, vol. 42, pp. 1661-1671, Feb./Mar./Apr. 1994.

[26] W. Koch and A. Baier," Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference," in *Proc. GLOBECOM '90*, Dec. 1990, pp. 1679-1684.

[27] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", in *Proc. IEEE International Conference on Communications*, 1995, pp. 1009-1013.

[28] L. Perez and C. Schegel, *Trellis Coding.* New York: John Wiley and Sons, 1997.

[29] M. Cedervall and R. Johannesson, " A fast algorithm for computing distance spectrum of convolutional codes," *IEEE Trans. Inform. Theory*, IT-35, pp. 1146-1159, 1989.

[30] W. C. Jakes, *Microwave Mobile Communications.* New York: John Wiley and Sons, 1974.

[31] M. A. Jordan and R. A. Nicols, "The effect of channel characteristics on turbo code performance," in *Proc. IEEE MILCOM*, 1996, pp. 17-21.

[32] T. A. Summers and S. G. Wilson, "SNR mismatch and online estimation in turbo decoding," *IEEE Trans. Comm.*, vol. 46, pp. 421-423, Apr. 1998.

[33] H. Chen, J. Yeh, N. Suehiro, "A multicarrier CDMA architecture based on orthogonal complementary codes for new generations of wideband wireless communications," *IEEE Communications Magazine*, Volume: 39 Issue: 10, Oct. 2001, pp. 126-135.

[34] M. J. E. Golay," Complementary series," *IRE Trans. Info. Theory*, vol. IT-7, pp. 82-87, 1961.

[35] R. Turyn, "Ambiguity function of complementary sequences," *IEEE Trans. Info. Theory*, vol. IT-9, pp. 46-47, Jan. 1963.

[36] N. Suehiro, "Complete complementary code composed of N-multiple-shift orthogonal sequences," *Trans. IEICE*, vol. J65-A, pp. 1247-53, Dec. 1982.

[37] N. Suehiro and M. Hatori, "N-shift cross-orthogonal sequences," *IEEE Trans. Info. Theory*, vol. IT-34, no. 1, pp. 143-46, Jan. 1988.

[38] B. Sklar, *Digital Communications*. New Jersey: Prentice Hall, 2001.

[39] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. Toronto: John Wiley and Sons, 1996.

[40] T. S. Rappaport, *Wireless Communications Principles and Practice*. New Jersey: Prentice Hall, 1996.

[41] M. K. Simon and M,-S, Alouini, *Digital Communication over Fading Channels*. New York: John Wiley and Sons, 2000.

[42] L. W. Couch, *Digital and Analog Communication Systems*. New Jersey: Prentice Hall, 1997.

APPENDICES

Matlab Simulation M-Files

The matlab files used for the simulations are:

- turbothesis.m > this is the top level simulation file.
- encode.m > turbo encoding function.
- rscgenerator.m > RSC encoder with perfect termination.
- rscgenerator2.m > RSC encoder with no termination.
- trellis2.m > trellis matrix generator.
- intg2state.m > function to convert integer to state representation.
- state2int.m > function to convert state to integer representation.
- turbodec2.m > turbo decoder top level function.
- mapdec1.m > MAP decoder for perfect termination.
- mapdec2.m > MAP decoder for no termination.
- intv.m > rectangular interleaver function.
- goldenint.m > golden interleaver function.
- rfadingslow.m > Jakes' Fading Coefficient for slow fading function.
- rfadingfast.m > Jakes' Fading Coefficient for slow fading function.
- occ4.m > OCC-CDMA encoder for L = 4.
- docc.m > OCC-CDMA decoder for L = 4.

Important parameters in turbothesis.m to change the setting and environment:

- msg_info
  This corresponds to the number of information bits in each of transmission frame. The value that is used for most simulations is 398.

- ebnodb
  This specifies the Eb/No in dB. We can use more than one value in this setting to simulate back-to-back simulations in different Eb/No values. Example: ebnodb = [0 2 4].

- g
  This represents the generator matrix for the turbo code. In the simulation in this thesis, g = [1 1 1 ; 1 0 1].

- niter
  This specifies the number of iterations per frame in this simulation. niter = 4 is used in the simulations.

- stop_num
  This specifies the number or frame error to stop the simulation for each specified ebnodb.

- pp
  For a punctured turbo code simulation, pp = 1.
  For an un-punctured turbo code simulation pp = 0.

- method
  This is to select the type of interleaver used in turbo encoding.
  method = 0 > golden interleaver
  method = 1 > golden dithered interleaver
  method = 2 > random interleaver
  method = 3 > rectangular interleaver

- mm
  This is the row dimension for the rectangular interleaver. The column dimension for the rectangular interleaver will be calculated automatically with respect to corresponding frame size.

- channel_int
  By setting the value of channel_int = 1 instead of 0, we are implementing the channel interleaver to our system.

- channel
  This selects the type of channel and scheme used in the simulation.
  channel = 1 > AWGN Turbo Codes Simulation
  channel = 2 > Flat Rayleigh Fading Turbo Codes
  channel = 3 > Flat Rayleigh Fading Turbo Codes with Diversity
  channel = 4 > Turbo-Coded OCC-CDMA in AWGN
  channel = 5 > Turbo-Coded OCC-CDMA in Rayleigh Fading
  channel = 6 > Turbo-Coded OCC-CDMA Rayleigh Fading Channel with Diversity

- fading_speed
  This is to select slow fading for fading_speed = 0, or fast fading for fading_speed = 1.

- perfect_est
  For the flat Raleigh Turbo Codes simulation:
  perfect_est = 0 > using channel estimation to predict the fading channel amplitude.
  perfect_est = 1 > using perfect knowledge of fading channel amplitude.

- Nc
  Nc is the width of the wiener filter window, or moving average window, refer to 3.2.1.
  For a slow fading acceptable Nc = 25, and for a fast fading Nc = 10.

- To get a good data statistically, it is preferred to use stop_num not less than 36.

```
% Turbo Code matlab Simulation
% Aldo Tjahjadi
% Oregon State University
% turbothesis.m
% This is the top level of matlab turbo codes simulation

clear all
close all

% Display title
disp(' ');
disp(' Turbo Code Demonstration ');
disp(' ');

% Input Parameters
msg_info = 398;          % Message size per frame
ebnodb = [1];            % Eb/No in dB
g = [1 1 1 ; 1 0 1];     % Generator Matrix
channel=1;               % Channel Type
niter = 4;               % Number of iterations
stop_num = [ 50 ];       % Error frame to stop simulation
channel_int = 1;         % Use or not use channel interleaver
pp = 0;                  % Rate select 1/2 or 1/3
method = 2;              % Method use for turbo interleaver
mm = 40;                 % Block interleaver dimension
fading_speed = 0;        % The flat Rayleigh Fading speed
perfect_est = 0;         % Perfect Channel Estimate or Not
Nc = 25;                 % Width of wiener filter window

% If pp = 1, then it is rate 1/2 or else rate = 1/3
if pp == 1,
    p = 2;
else
    p = 3;
end

% Calculating the required info the generate the RSC trellis coding
[n,K] = size(g);
m = K - 1;               % Memory in RSC encoder
max_state = 2^m;         % Total number of states
L_frame=msg_info+m;      % Need 2*m bits tail bit to reset state to
                         % initial state for 2 RSC encoder

% Resolving the dimension for block interleaver
nn = ceil(L_frame/mm);
```

```
% Generating the turbo interleaver index with the method chosen
if method == 3
    disp(' Interleaving using block interleaver');
    index = intv([1:msg_info+m], mm, nn);
elseif method == 2
    disp(' Interleaving using Random Interleaver');
    index = randperm(msg_info+m);
elseif method == 1
    disp(' Interleaving using Golden Dithered Interleaver');
    index = goldenint(msg_info+m, 1);
else
    disp(' Interleaving using Golden Interleaver');
    index = goldenint(msg_info+m, 0);
end

% The interleaver index for channel interleaver
iindex = intv([1:L_frame*p], 64, ceil(L_frame*p/64));

% Initializing the error frame and error bit matrix
% This will record the frame errors
errorframe = zeros(length(ebnodb), niter);
% This will record the bit errors
errorbit = zeros(length(ebnodb), niter);

% Generating random variables for starting time in fading channel
tbs = round(rand(1,1)*1000000);
tbs1 = round(rand(1,1)*1000000);
tbs2 = round(rand(1,1)*1000000);
tbs11 = round(rand(1,1)*1000000);
tbs22 = round(rand(1,1)*1000000);


% Simulating every ebnodb given
for ebnow = 1:length(ebnodb),

   % Changing ebnodb to ebno in linear scale
   ebnolin = 10^(ebnodb(ebnow)/10);
   % Rate of transmition
   rate = 1/p;
   % AWGN channel standard deviation parameter
   sigma = sqrt(1/(2*ebnolin*rate));
   % Frame Number being decoded
   frameno(ebnow) = 0;
   % Reliability value of the channel
   L_c =4*ebnolin*rate;
```

```
% Keep on simulating to the next frame if the frame error on the
% last iteration is less than the specified stop frame error
while errorframe(ebnow, niter)<stop_num(ebnow),

   % Increment the frame number
   frameno(ebnow) = frameno(ebnow) + 1;
   % Randomly generating the message bit information
   x=round(rand(1,msg_info));
   % Turbo encoding the information bit
   en_output =encode(x,g,p,index);
   % Generating AWGN noise vectors
   noise=randn(size(en_output))*sigma;

   % Channel interleaving
   if channel_int ==1
       en_output = en_output (iindex);
   end


   % AWGN Turbo Codes only
   if channel==1
       % Adding AWGN noise
       r1 =(2*en_output-1)+noise;


   % Flat Rayleigh Fading Turbo Codes
   elseif channel==2

       % Length calculation for en_output
       en_length = length(en_output);
       % Generating Flat Rayleigh Fading using Jake's Model
       if fading_speed==0
           a=rfadingslow(en_length,tbs);
       else
           [a, fdts] =rfadingfast(en_length,tbs);
       end
       % Incrementing the time for fading channel
       tbs = tbs + en_length;
       % Adding noise and Rayleigh Flat Fading
       r0 =a.*(2*en_output-1)+noise;
       if perfect_est == 0
           m_out = move_ave(abs(r0),Nc);
           r1 = (r0.*m_out);
       else
           r1 = (r0.*a);
       end
```

```
% Flat Rayleigh Fading Turbo Codes with Diversity
elseif channel ==3

    % Length calculation for en_output
    en_length = length(en_output);
    % Generating Flat Rayleigh Fading using Jake's Model
    if fading_speed==0
        a1=rfadingslow(en_length,tbs1);
        a2=rfadingslow(en_length,tbs2);
    else
        [a1, fdts] =rfadingfast(en_length,tbs1);
        [a2, fdts] =rfadingfast(en_length,tbs2);
    end
    % Incrementing the time for fading channel
    tbs1 = tbs1 + en_length;
    tbs2 = tbs2 + en_length;
    % Adding noise and Rayleigh Flat Fading
    r0 = a1.*(2*en_output-1)+noise;
    noise2 = randn(size(en_output))*sigma;
    r02 =  a2.*(2*en_output-1)+noise2;
    % Channel Estimation
    m_out1 = move_ave(abs(r0),Nc);
    m_out2 = move_ave(abs(r02),Nc);
    % Combining diversity
    r11 = (r0.*m_out1);
    r12 = (r02.*m_out2);
    [temp, abta2] = find(m_out1>m_out2);
    r1 = r12;
    r1(abta2) = r11(abta2);


%OCC-CDMA turbo code AWGN only
elseif channel == 4 %OCC-CDMA turbo code AWGN only

    % Generating NRZ Output
    polarout =(2*en_output-1);
    % Generating OCC-CDMA outputs
    [occout,codes]= occ4(polarout);
    % Generate noise
    noise2(1,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    noise2(2,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    % AWGN  channel
    occout2(1,:) = occout(1,:)+ noise2(1,:);
    occout2(2,:) = occout(2,:)+ noise2(2,:);
    % Decoding the output
    doccout = docc(occout2, codes);
    r1 = doccout;
```

```
%Turbo-Coded OCC-CDMA in Rayleigh Fading
elseif channel ==5

    % Generating NRZ Output
    polarout =(2*en_output-1);
    % Generating OCC-CDMA outputs
    [occout,codes]= occ4(polarout);
    % Generate noise
    noise2(1,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    noise2(2,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    % Length of vector calculation
    en_length = length(noise2(1,:));
    % Fading channel coefficient generation
    if fading_speed==0
        a1 =rfadingslow(en_length,tbs1);
        a2 =rfadingslow(en_length,tbs2);
    else
        [a1, fdts] =rfadingfast(en_length,tbs1);
        [a2, fdts] =rfadingfast(en_length,tbs2);
    end
    % Increment time
    tbs1 = tbs1 + en_length;
    tbs2 = tbs2 + en_length;

    % AWGN and fading channel
    occout2(1,:) = a1.*occout(1,:)+ noise2(1,:);
    occout2(2,:) = a2.*occout(2,:)+ noise2(2,:);

    % Decoding the output
    doccout = docc(occout2, codes);
    r1 = doccout;
```

```
% Turbo-coded OCC-CDMA Rayleigh Fading Channel with Diversity
elseif channel==6

    % Generating NRZ Output
    polarout =(2*en_output-1);
    % Generating OCC-CDMA outputs
    [occout,codes]= occ4(polarout);
    % Generate noise
    noise2(1,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    noise2(2,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    noise2d(1,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    noise2d(2,:)=randn(1,length(occout(1,:)))*sigma*sqrt(8);
    % Length of vector calculation
    en_length = length(noise2(1,:));
    % Fading channel coefficient generation
    if fading_speed==0
        a1 = rfadingslow(en_length,tbs1);
        a2 = rfadingslow(en_length,tbs2);
        a11 = rfadingslow(en_length,tbs11);
        a22 = rfadingslow(en_length,tbs22);
    else
        [a1, fdts] =rfadingfast(en_length,tbs1);
        [a2, fdts] =rfadingfast(en_length,tbs2);
        [a11, fdts] =rfadingfast(en_length,tbs11);
        [a22, fdts] =rfadingfast(en_length,tbs22);
    end
    % Increment time
    tbs1 = tbs1 + en_length;
    tbs2 = tbs2 + en_length;
    tbs11 = tbs11 + en_length;
    tbs22 = tbs22 + en_length;
    % AWGN and fading channel
    occout2(1,:) = occout(1,:).*a1+ noise2(1,:);
    occout2(2,:) = occout(2,:).*a2+ noise2(2,:);
    occout2d(1,:) = occout(1,:).*a11 + noise2d(1,:);
    occout2d(2,:) = occout(2,:).*a22 + noise2d(2,:);
    % Diversity combining
    occ2d1= (occout2+occout2d)/2;
    % Decoding the output
    doccout = docc(occ2d1, codes);
    r1 = doccout;


else
    r1 =(2*en_output-1)+noise;
end
```

```
        % Channel Deinterleaving
        if channel_int ==1
            r1(iindex)= r1;
        end

        % Scaled Signal
        r=L_c*r1;

        % Calculating the length of the received signal
        r_length = length(r);

        % DEPUNCTURING
        if p == 3,
          dp_out=reshape(r,3,length(index));
        else
          dp_out = zeros(3, length(index));
          dp_out(1,:) = r(1:2:r_length);
          dp_out(2,1:2:length(index)) = r(2:4:r_length);
          dp_out(3,2:2:length(index)) = r(4:4:r_length);
        end


        % Rearranging the received signal after depuncturing for
        % better format in iterative decoding

        in_dec1=[dp_out(1:2,:)];
        in_dec2(1,:)= dp_out(1,index);
        in_dec2(2,:)= [dp_out(3,:)];

        % Turbo decoding
        [err,f_error]  = turbodec2(x,g,niter,index,in_dec1,in_dec2);
        err

        % Incrementing frame error
        errorframe(ebnow,:) = errorframe(ebnow,:)+f_error;

        % Increment the error bit
        errorbit(ebnow,1:niter) = errorbit(ebnow,1:niter)
        + err(1:niter);

        % Display out the BER
        ber(ebnow,:) = errorbit(ebnow,1:niter)
        /(frameno(ebnow)*msg_info)

    end

    % Final BER and FER calculation
    ber(ebnow,:) = errorbit(ebnow,1:niter)
    /(frameno(ebnow)*msg_info) ;
    fer(ebnow,:) = errorframe(ebnow,:)/frameno(ebnow);

end
```

```
% encode.m
% This function is for turbo encoding
% Input is the binary message
% output =encode(input,g,p,index)
% g is the generator matrix
% p gives the information whether it is punctured or not
% index is the turbo interleaver index

function output =encode(input,g,p,index)

[n,K] = size(g);

% Msg_info calculation
msg_info = length(input);

% Output from RSC generator 1 (terminated)
en_output1 =rscgenerator(g,input);

% Interleaving the sequence from input plus tail bits
int_input = en_output1(1,(index));

% Output from RSC generator 2 (unterminated)
en_output2 =rscgenerator2(g,int_input);

%PUNCTURING

if p == 3
    % Generating rate 1/3 puncturing scheme output
    en_out_temp = [en_output1(1:2,:); en_output2(2,:)];
    output = reshape(en_out_temp, 1, length(int_input)*3);
else
    % Puncturing rate 1/2 scheme output
    tempoutput = zeros(1,length(index));
    tempoutput(1:2:length(index))
    = en_output1(2,1:2:length(index));
    tempoutput(2:2:length(index))
    = en_output2(2,2:2:length(index));
    temp3 = [en_output1(1,:); tempoutput];
    output = reshape(temp3, 1, length(index)*2);
end
```

```
% rscgenerator.m
% This function is to generate RSC outputs
% with perfect termination output
% [output] = rscgenerator(g,dk)
% g is the generator matrix
% dk is the input

function [output] = rscgenerator(g,dk)

% Calculating the parameter needed for computation
[n,K] = size(g);
m = K - 1;
L_input= length(dk);


% Assign the present state inital as ' 0 0 '
present_state = intgs2state( 0 , m );

% RSC encoding scheme form message signal
for k=1:L_input
   ak = mod( g(1,:)*[dk(k) present_state]', 2 );
   for v=2:n,
      output(v,k) = mod( g(v,:)*[ak present_state]', 2 );
   end
   next_state=[ak present_state(1:(m-1))];
   present_state=next_state;
   output(1,k)=dk(k);
end

% Terminating the trellis so it goes back to original state
for k=1:m
   t_output(1,k)=mod( g(1,:)*[0 present_state]', 2);
   for v = 2:n,
      t_output(v,k) = mod( g(v,:)*[0 present_state]', 2 );
   end
   next_state=[0 present_state(1:(m-1))];
   present_state=next_state;
end

output = [output t_output];
```

```
% rscgenerator2.m
% This function is to generate RSC outputs
% with unterminated output
% [output] = rscgenerator2(g,dk)
% g is the generator matrix
% dk is the input

function [output] = rscgenerator2(g,dk)

% Calculating the parameter used in the computation
[n,K] = size(g);
m = K - 1;
L_input= length(dk);

% Inital state at '0 0'
present_state = intgs2state( 0 , m );

% For the second RSC generator there is no termination
% So the last state leaves open
for k=1:L_input
   % Calculating ak
   ak = mod( g(1,:)*[dk(k) present_state]', 2 );
   for v=2:n,
      output(v,k) = mod( g(v,:)*[ak present_state]', 2 );
   end
   next_state=[ak present_state(1:(m-1))];
   present_state=next_state;
   output(1,k)=dk(k);
end
```

```
% trellis2.m
% This function will provides the trellis structure
% for the RSC code with generator matrix g.
% The format is trellisout(state,:)
%  = [ present_state next_state0 next_state1 output0 output1 ]
% The outputs are in polar form

function [trellisout] = trellis2(g)

% Calculating the parameters of state and g
[n,K] = size(g);
m = K - 1;
total_state = 2^m;


% Calculating the trellis structure for input bit dk = 0
dk = 0;
for state=1:total_state,
    % Present state conversion from integer (ak-1 ak-2 ... ak-m)
    present_state = intgs2state( state-1, m );
    % ak is the most recent bank
    ak = mod( g(1,:)*[dk present_state]', 2 );
    % Next state (ak ak-1 .. ak-m)
    next_state=[ak present_state(1:(m-1))];
    % Integer representation of next state
    nstate=state2int(next_state)+1;
    % Calculating the outputs v
    v(1)=0;
    for xx=2:n
        v(xx) = mod( g(xx,:)*[ak present_state]', 2 );
    end
    % Recombining the structure information
    next_state0(state,:) = [state nstate];
    out0(state,:) = [round(v-0.5)];
end
```

```
% Calculating the trellis structure for input bit dk = 1
dk = 1;
for state=1:total_state,
     % Present state conversion from integer (ak-1 ak-2 ... ak-m)
    present_state = intgs2state( state-1, m );
     % ak is the most recent bank
    ak = mod( g(1,:)*[dk present_state]', 2 );
    % Next state (ak ak-1 .. ak-m)
    next_state=[ak present_state(1:(m-1))];
    % Integer representation of next state
    nstate=state2int(next_state)+1;
    % Calculating the outputs v
    v(1)=1;
    for xx=2:n
        v(xx) = mod( g(xx,:)*[ak present_state]', 2 );
    end

    % Recombining the structure information
    next_state1(state,:) = [nstate];
    out1(state,:) = [round(v-0.5)];
end

trellisout = [next_state0 next_state1 out0 out1];
```

```
% intg2state.m
% This will change the integer values to vectors
% of binary state representations of the input in
% integers
% bin_state = intgs2state( integs, m )
% integs is the integer input
% m is the state size

function bin_state = intgs2state( integs, m )

for j = 1:length( integs )
   for i = m:-1:1
       state(j,m-i+1) = fix( integs(j)/ (2^(i-1)) );
       integs(j) = integs(j) - state(j,m-i+1)*2^(i-1);
   end
end

bin_state = state;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% state2int.m
% This function will convert the vectors of binary
% state representation to integer value

function intgs = state2int( state )

m = length( state );

for i = 1:m,
   vect(i) = 2^(m-i);
end

intgs = state*vect';
```

```
% turbodec2.m
% This function is the turbo decoding process for 1 iteration
% Where the output of decoder 1 is feed to decoder 2 and then
% at the beginning of the next iteration the output of decoder 2
% is the input of decoder 1.
% [err,errorframe]=turbodec2(x,g,niter,index,in_dec1, in_dec2)
% x is the input message to calculate the error
% g is the generator matrix
% niter is the number of iteration
% index is the turbo interleaver index
% in_dec1 is the input to decoder1
% in_dec2 is the input to decoder2


function [err,errorframe]=turbodec2(x,g,niter,index,in_dec1,
in_dec2)

iter=1;
err = [];
[n,K] = size(g);
m = K - 1;
msg_info = length(in_dec1)-m;
L_dec1 = zeros(1,msg_info+m);
errorbit = zeros(1, niter);
errorframe = zeros(1, niter);

while iter<=niter,

    % Decoder I
    L_dec1out  = mapdec1(in_dec1,L_dec1,g);
    L_dec2=[L_dec1out(index)];
    % Decoder II
    L_dec2out = mapdec2(in_dec2,L_dec2,g);
    L_dec1(index) = L_dec2out;

    xhat1(index) = (sign(L_dec2out+L_dec2+in_dec2(1,:))+1)/2;

    % Cut the tail bits
    xhat=xhat1(1:msg_info);

    % Error Calculation
    err(iter) = length(find(xhat~=x));
    % Increment the frame error
    if err(iter)>0
        errorframe(1,iter) = errorframe(1,iter)+1;
    end

    % Increment the interation counter
    iter=iter+1;

end
```

```
% mapdec1.m
% This is MAP decoder for RSC 1
% Perfectly terminated trellis
% L_out = mapdec1(input, Le,g)
% g is the generator matrix
% input is the input we want to decode
% Le is the extrinsic information

function L_out = mapdec1(input, Le,g)

trels = trellis2(g);

% Scaling of the input to fit into other parameter
% to calculating gamma
input=input/2;
Le=Le/2;

% Calculating the dimension of the input matrix
[input_w,input_l]=size(input);

% Calculating the generator matrix states property
[n,K] = size(g);
m = K - 1;
total_state = 2^m;

% Extracting the output given input is 0 and 1
[cc,dd] = size(trels);
v = (dd-3)/2;
output0= trels(:,4:4+v-1);
output1= trels(:,4+v:4+v+v-1);
[aa,bb] = size(output0);

% Trellis state when the input is 0 and 1
next_state0 = trels(:,2);
next_state1 = trels(:,3);

% Initialization of alfa_km1 here we know that
% It will always start from state 1 (0 0 ..)
alfa_km1(1,1) = 1;
alfa_km1(2:total_state,1) = zeros(total_state-1,1);

% Initialization of beta since this is single termination
% turbo scheme, we set the final states to have equal opportunity
beta(1,input_l) = 1;
beta(2:total_state,input_l) = zeros(total_state-1,1);
```

```
% Calculation of branch Metrics gamma
for k=1:input_l,
   gamma0(:,k)=exp(output0*input(:,k)+(-1).*Le(:,k));
   % -1 is the polar representation of 0
   gamma1(:,k)=exp(output1*input(:,k)+(+1).*Le(:,k));
   % +1 is the polar representation of 1

   gamma_e_0(:,k)=exp(output0(:,2:bb)*input(2:input_w,k));
   gamma_e_1(:,k)=exp(output1(:,2:bb)*input(2:input_w,k));

end

% Computation of alfa_km1
for k =1:input_l,
   % gamma0 multiplied by alfa_km1 foward transition
   % that is given input is 0
   mult_ag0(next_state0,k+1)=alfa_km1(:,k).*gamma0(:,k);
   % gamma1 multiplied by alfa_km1 foward transition
   % that is given input is 1
   mult_ag1(next_state1,k+1)=alfa_km1(:,k).*gamma1(:,k);
   % sum of all gamma multiplied by alfa_km1 coming to
   % all state in time k+1
   alfa_km1_temp(:,k+1)=(mult_ag0(:,k+1)+mult_ag1(:,k+1));
   % sum of all possible state of alfa_km1_temp
   alfa_km1_sum(k+1)=sum(alfa_km1_temp(:,k+1));
   % Computation of alfa(k-1) note that
   % alfa_km1(k=2) = alfa(1)
   alfa_km1(:,k+1)=alfa_km1_temp(:,k+1)/alfa_km1_sum(k+1);
end

% Tracing backward for beta
for k = input_l-1:-1:1,
   % Just like computing alfa this is sum of all gamma multiplied
   % by beta(k+1)
   beta_temp(:,k)=beta(next_state0,k+1).*gamma0(:,k+1)+
   beta(next_state1,k+1).*gamma1(:,k+1);
   beta(:,k)=beta_temp(:,k)/alfa_km1_sum(k+1);
end

% Compute the soft output, log-likelihood ratio of symbols in the
% frame
for k=1:input_l,
   % Computing lambda
   lambda0(next_state0,k)=alfa_km1(:,k).*gamma_e_0(:,k)....
      .*beta(next_state0,k);
   lambda1(next_state1,k)=alfa_km1(:,k).*gamma_e_1(:,k)....
      .*beta(next_state1,k);
end

% Computing L_out
L_out=log(sum(lambda1))-log(sum(lambda0));
```

```
% mapdec2.m
% This is MAP decoder for RSC 2
% Unterminated trellis
% L_out = mapdec2(input, Le,g)
% g is the generator matrix
% input is the input we want to decode
% Le is the extrinsic information

function L_out = mapdec2(input, Le,g)

trels = trellis2(g);

% Scaling of the input to fit into other parameter
% to calculating gamma
input=input/2;
Le=Le/2;

% Calculating the dimension of the input matrix
[input_w,input_l]=size(input);

% Calculating the generator matrix states property
[n,K] = size(g);
m = K - 1;
total_state = 2^m;

% Extracting the output given input is 0 and 1
[cc,dd] = size(trels);
v = (dd-3)/2;
output0= trels(:,4:4+v-1);
output1= trels(:,4+v:4+v+v-1);
[aa,bb] = size(output0);

% Trellis state when the input is 0 and 1
next_state0 = trels(:,2);
next_state1 = trels(:,3);

% Initialization of alfa_km1 here we know that
% It will always start from state 1 (0 0 ..)
alfa_km1(1,1) = 1;
alfa_km1(2:total_state,1) = zeros(total_state-1,1);

% Initialization of beta since this is single termination
% turbo scheme, we set the final states to have equal opportunity
beta(1,input_l) = 1;
beta(2:total_state,input_l) = ones(total_state-1,1);
```

```
% Calculation of branch Metrics gamma
for k=1:input_l,
   gamma0(:,k)=exp(output0*input(:,k)+(-1).*Le(:,k));
   % -1 is the polar representation of 0
   gamma1(:,k)=exp(output1*input(:,k)+(+1).*Le(:,k));
   % +1 is the polar representation of 1

   gamma_e_0(:,k)=exp(output0(:,2:bb)*input(2:input_w,k));
   gamma_e_1(:,k)=exp(output1(:,2:bb)*input(2:input_w,k));

end

% Computation of alfa_km1
for k =1:input_l,
   % gamma0 multiplied by alfa_km1 foward transition
   % that is given input is 0
   mult_ag0(next_state0,k+1)=alfa_km1(:,k).*gamma0(:,k);
   % gamma1 multiplied by alfa_km1 foward transition
   % that is given input is 1
   mult_ag1(next_state1,k+1)=alfa_km1(:,k).*gamma1(:,k);
   % sum of all gamma multiplied by alfa_km1 coming to
   % all state in time k+1
   alfa_km1_temp(:,k+1)=(mult_ag0(:,k+1)+mult_ag1(:,k+1));
   % sum of all possible state of alfa_km1_temp
   alfa_km1_sum(k+1)=sum(alfa_km1_temp(:,k+1));
   % Computation of alfa(k-1) note that
   % alfa_km1(k=2) = alfa(1)
   alfa_km1(:,k+1)=alfa_km1_temp(:,k+1)/alfa_km1_sum(k+1);

end

% Tracing backward for beta
for k = input_l-1:-1:1,
   % Just like computing alfa this is sum of all gamma multiplied
   % by beta(k+1)
   beta_temp(:,k)=beta(next_state0,k+1).*gamma0(:,k+1)+
   beta(next_state1,k+1).*gamma1(:,k+1);
   beta(:,k)=beta_temp(:,k)/alfa_km1_sum(k+1);
end

% Compute the soft output, log-likelihood ratio of symbols in the
% frame
for k=1:input_l,
   % Computing lambda
   lambda0(next_state0,k)=alfa_km1(:,k).*gamma_e_0(:,k)....
      .*beta(next_state0,k);
   lambda1(next_state1,k)=alfa_km1(:,k).*gamma_e_1(:,k)....
      .*beta(next_state1,k);
end

% Computing L_out
L_out=log(sum(lambda1))-log(sum(lambda0));
```

```
% intv.m
% This function is the rectangular interleaver
% [sout] = intv(sin,m,n)
% sin is the input before interleaver
% m,n is the dimension of the rectangular interleaver

function [sout] = intv(sin,m,n)

% This is to count how long the vector of sout should be
kk = ceil(length(sin)/(m*n));
% If the sin is not a factor of m*n then, pad it with zero
spad = [sin zeros(1,(m*n)*(kk)-length(sin))];
yy = length(spad)/(m*n);
zz = m*n;

% This is the actual interleaving algorithm
for i = 1:yy,
    x = reshape(spad((i-1)*zz+1:i*zz), m, n);
    x = x';
    sout((i-1)*zz+1:i*zz) = reshape(x,1,m*n);
end

[ uu , hh ] = find(sout ==0);
sout(hh) = [];
```

```
% goldenint.m
% This is Golden interleaver index generator
% i = goldenint(N, method)
% N > input length of index generated
% Method > Select if using interleaver of dithered golden
interleaver
% s = starting point for index generator
% m = any integer greater than 0 (1 or 2 typical)
% r = the index spacing between nearby elements to be
% max spread (1)
% D = dithered parameter typically for turbocode 0.01
% In typical max spreading adjacent elements j = 0 and r = 1
% For turbocode, greater values of jand r can be used to obtain
% the best spreading for elements spaced r apart
% For turbo Code with 16 state j = 9 and r = 15 with rsc repetition
% period of r = 15

function i = goldenint(N, method)
m = 2;
j = 0;
r = 1;
if method == 0
    D = 0;
else
    D = 0.01;
end
s = 1;

% Used for dithered golden interleaver
ND = N*D;
d = rand(1,N)*ND;
% Golden Section calculation
g = (sqrt(5)-1)/2;
% Compute real increment
C = N*(g.^m+j)/r;
% Calculation of real Vector v
for n = 0:N-1,
    v(n+1) = mod(s + n*C+d(n+1), N);
end
% Find sort vector
[a,b] = sort(v);
n = 1:length(b);
% index vector
i(b) = n;
```

```
% rfadingslow.m
% This is Jakes model for Rayleigh fading Channel
% for slow fading
% [coef]= rfadingslow(blk_length,tbs)
% blk_length is the length of the outputs
% tbs is the time to start

function [coef]= rfadingslow(blk_length,tbs)

Rb = 19200;
tb = 1/Rb;
v = 70*1609/3600;
f = 900e6;
nn = tbs + blk_length;
t = tbs*tb:tb:(nn-1)*tb;
So = 8;
c = 3e8;
fd = f*v/c;
S = 2*(2*So +1);
wm = 2 * pi * f * v/c;
pN = 0;
pn = pi.*[1:So]/(So+1);
S = 2*(2*So +1);
wn = wm * cos(2*pi.*[1:So]/S);



%calculating xc and xs
for x = 1:length(t),
    xct = (cos(pn).*cos(wn.*t(x)) );
    xst = (sin(pn).*cos(wn.*t(x)) );
%Calculating xc and xs
xc(x) = 2*sum(xct)+ sqrt(2)*cos(pN)*cos(wm*t(x));
xs(x) = 2*sum(xst)+ sqrt(2)*sin(pN)*cos(wm*t(x));
end

%calculating ct
ct = (xc+(i*xs))/sqrt(2*So+1);
coef = abs(ct);
```

```
% rfadingfast.m
% This is Jakes model for Rayleigh fading Channel
% for fast fading
% [coef,fdts]= rfadingfast(blk_length,tbs)
% blk_length is the length of the outputs
% tbs is the time to start

function [coef,fdts]= rfadingfast(blk_length,tbs)

Rb = 19200;
tb = 1/Rb;
v = 70*1609/3600;
f = 3.6821e9;
nn = tbs + blk_length;
t = tbs*tb:tb:(nn-1)*tb;
So = 8;
c = 3e8;
fd = f*v/c;
fdts = fd*tb;
S = 2*(2*So +1);
wm = 2 * pi * f * v/c;
pN = 0;
pn = pi.*[1:So]/(So+1);
S = 2*(2*So +1);
wn = wm * cos(2*pi.*[1:So]/S);

%calculating xc and xs
for x = 1:length(t),
    xct = (cos(pn).*cos(wn.*t(x)) );
    xst = (sin(pn).*cos(wn.*t(x)) );
%Calculating xc and xs
xc(x) = 2*sum(xct)+ sqrt(2)*cos(pN)*cos(wm*t(x));
xs(x) = 2*sum(xst)+ sqrt(2)*sin(pN)*cos(wm*t(x));
end

%calculating ct
ct = (xc+(i*xs))/sqrt(2*So+1);
coef = abs(ct);
```

```
% occ4.m
% This function is the generation of OCC-CDMA output
% [occout, codes] = occ4(data)

function [occout, codes] = occ4(data)

% OCC encoder
size = 4;
% Reshape the input data
datar = reshape(data, 2, length(data)/2);
userA = datar(1,:);
userB = datar(2,:);

% The codes used
codeA0 = [ 1 1 1 -1];
codeA1 = [ 1 -1 1 1 ];
codeB0 = [ 1 1 -1 1 ];
codeB1 = [ 1 -1 -1 -1];

% Initiation of output variables
outC = zeros(1, (length(userA)-1)+(size));
outD = zeros(1, (length(userA)-1)+(size));

% OCC-CDMA generation
for k = 1:length(userA),
symbolA0 = userA(k)*codeA0;
symbolA1 = userA(k)*codeA1;
symbolB0 = userB(k)*codeB0;
symbolB1 = userB(k)*codeB1;
II0 = symbolA0+symbolB0;
II1 = symbolA1+symbolB1;
outC(k:k+3) = outC(k:k+3) +II0;
outD(k:k+3) = outD(k:k+3) +II1;
end

codes = [ codeA0; codeA1; codeB0; codeB1];
input = [ outC; outD];
occout = input;
```

```
% docc.m
% This is to decode OCC-CDMA with length L = 4
% [dcodeout] = docc(input, codes)
% Input is the undecoded
% Codes are the OCC-CDMA codes

function [dcodeout] = docc(input, codes)

size = 4;

% Arrange the input and codes
outC = input(1,:);
outD = input(2,:);
codeA0 = codes(1,:);
codeA1 = codes(2,:);
codeB0 = codes(3,:);
codeB1 = codes(4,:);

% Calculating message size
msgsize = length(outC)-size+1;

% Correlation and sum
for w = 1:msgsize,

    duserA(w) = (outC(w:w+3)*codeA0')+(outD(w:w+3)*codeA1');
    duserB(w) = (outC(w:w+3)*codeB0')+(outD(w:w+3)*codeB1');

end


duser = [duserA; duserB]/8;
dcodeout = reshape(duser, 1, length(duserA)*2);
```