University of Alberta

Performance Evaluation of Low-Density Parity-Check Codes in the Presence of Inter-symbol Interference, Colored Noise, and $1/f$ Noise

by

Saeed Sharifi Tehrani ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2005

Your file Votre référence
ISBN: 0-494-09285-8
Our file Notre référence
ISBN: 0-494-09285-8

University of Alberta

Library Release Form

Name of Author: Saeed Sharifi Tehrani

Title of Thesis: Performance Evaluation of Low-Density Parity-Check Codes in the Presence of Inter-symbol Interference, Colored Noise, and $1/f$ Noise

Degree: Master of Science

Year this Degree Granted: 2005

____

Date: ________________

To my parents

Abstract

The class of Low-Density Parity-Check (LDPC) codes includes some of the most powerful capacity-approaching codes reported to date. As a result, LDPC codes have been considered for many new communication standards. However, a better understanding of the effects of the signal impairments that exist in such applications is required. In this thesis, the performance of various LDPC codes, including recent candidate LDPC codes for 10GBASE-T Ethernet, in the presence of channel impairments is evaluated and compared with the effects of conventional Additive White Gaussian Noise (AWGN). The channel impairments in this study include Inter-Symbol Interference (ISI), high-frequency and low-frequency Additive Colored Gaussian Noise (ACGN), and $1/f$ noise. The results show that LDPC codes appear to be more sensitive to AWGN than to ISI, but for the case of colored noise, they are more vulnerable to ACGN and $1/f$ noise than to AWGN.

Acknowledgements

I must thank my supervisors, Dr. Bruce Cockburn and Dr. Stephen Bates, for their guidance, assistance and valuable recommendations throughout my thesis research. I would also like to thank Sheryl Howard, presently a Ph.D. candidate in the High Capacity Digital Communication laboratory, for her helpful suggestions and assistance.

This research was partially supported by the University of Alberta, the Alberta Informatics Circle of Research Excellence (iCORE), CMC Microsystems, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Semiconductor Research Corporation (SRC).

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Significance**
ACGN	Additive Colored Gaussian Noise
ANEXT	Alien Near-End CrossTalk
ASK	Amplitude Shift Keying
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPA	Belief Propagation Algorithm
CIR	Channel Impulse Response
DDFSE	Delayed Decision Feedback Sequence Estimator
DFE	Decision Feedback Equalizer
FEXT	Far-End CrossTalk
FIR	Finite Impulse Response
FSM	Finite State Machine
IIR	Infinite Impulse Response
ISI	Inter-Symbol Interference
LAN	Local Area Network
LDPC	Low-Density Parity-Check
LDPC-CC	Low-Density Parity-Check Convolutional Code
LFSR	Linear Feedback Shift Register
LLR	Log-Likelihood Ratio
LMS	Least Mean Square
LTI	Linear Time Invariant
MAP	Maximum Aposteriori Probability
MLSE	Maximum Likelihood Sequence Estimation
MPA	Message Passing Algorithm

MSA	Min-Sum Algorithm
MSE	Mean Square Error
NEXT	Near-End CrossTalk
PAM	Pulse Amplitude Modulation
PAM-M	Pulse Amplitude Modulation with M levels of signal amplitude
PDF	Probability Density Function
SNIR	Signal-to-Noise-and-Interference Ratio
SNR	Signal-to-Noise Ratio
SPA	Sum-Product Algorithm
TCM	Trellis Coded Modulation
THP	Tomlinson-Harashima Precoding
UTP	Unshielded Twisted Pair
VA	Viterbi Algorithm

Chapter 1

Introduction

1.1 The Evolution of Ethernet

The ideas underlying Ethernet Local Area Network (LAN) technology were originally presented by Bob Metcalfe at the Xerox Palo Alto Research Center (PARC) in the early 1970's. Figure 1.1 shows an early drawing of an Ethernet network by Metcalfe [1]. He chose the word "Ether" to describe the physical medium that carries the information bits to all nodes in the network. In fact, the "Ether" prefix of Ethernet suggests that networks are not meant to be restricted for use on only one connection type, and that Ethernet could be used on many different systems and function the same way on all. Copper cables, fiber optic cables, and even wireless technologies have now all been used to implement Ethernet. This flexibility, combined with simple expandability, makes Ethernet an attractive networking solution in today's mixture of different physical layer technologies.

Industry standards based on Ethernet LAN were adopted in 1980 under the IEEE 802.3 series of specifications for data networks [6]. These specifications define low-level data transmission protocols and the technical details that are needed to build interoperable Ethernet LAN products like cards and cables. Under the Open System Interconnection (OSI) model, Ethernet technology operates at the physical and data link layers (see Figure 1.2).

The IEEE naming convention for Ethernet can be categorized as "XBASEY", where the prefix X indicates the data rate in Mbps, the second term stands

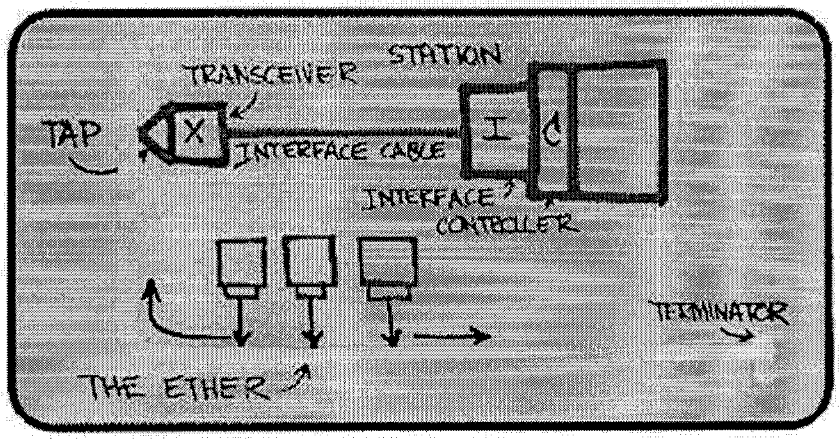

Figure 1.1: An early drawing of an Ethernet network by Metcalfe [1].

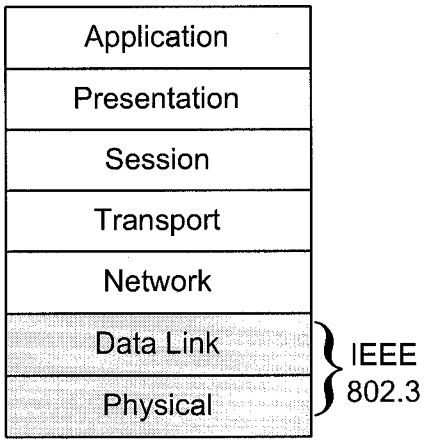

Figure 1.2: Ethernet in the OSI model.

for the "baseband" transmission type, and the suffix Y indicates the segment length (*e.g.* 500 m for 10BASE-5). In more recent standards, the suffix Y has been replaced by a letter indicating the type of medium, for instance, T for Unshielded Twisted Pair (UTP) copper cable (*e.g.* 1000BASE-T) and T4 for four such pairs.

Ethernet technology has evolved and matured over a relatively long period of time. The original Ethernet standard supports data transfers at a maximum rate of 10 Mbps. 10BASE5, often referred to as Thicknet, was the first incarnation of Ethernet technology. The industry used Thicknet in the early 1980's until 10BASE2, so-called Thinnet, appeared in 1986 [6]. Compared to Thicknet, Thinnet offered the advantage of thinner and more flexible cabling, making it easier to wire office buildings for Ethernet. In 1991, the IEEE 802.3 10BASE-T standard was approved. 10BASE-T soon became the most common form of Ethernet. It was even more convenient than Thicknet or Thinnet because 10BASE-T cables utilize cheap and flexible UTP cabling rather than bulky coaxial cabling. Over time, to meet the increasing performance needs of LANs, the industry created additional Ethernet specifications for Fast Ethernet, which extends Ethernet performance up to 100 Mbps [6]. By the mid-1990's, Fast Ethernet technology had matured and met its design goals of (i) increasing the performance of traditional Ethernet while (ii) avoiding the need to completely re-cable existing Ethernet networks. Fast Ethernet comes in the 100BASE-T (1995) and 100BASE-FX varieties, which operate on UTP and fiber optic cable, respectively. By far the most popular of these alternatives is 100BASE-T, a standard that includes 100BASE-TX over Category 5 (CAT-5) UTP, 100BASE-T2 (CAT-3 or better UTP), and 100BASE-T4 (a modified variation of 100BASE-T2 that includes two additional wire pairs).

Whereas Fast Ethernet sped up traditional Ethernet from 10 Mbps to 100 Mbps, Gigabit Ethernet boasts the same order-of-magnitude improvement over Fast Ethernet by offering speeds of 1000 Mbps. Gigabit Ethernet was first made to travel over optical and copper cabling, but the 1000BASE-T standard (1999) successfully supports UTP as well [7]. 1000BASE-T employs baseband transmission over four pairs of CAT-5 cabling. A throughput of 1 Gbps is achieved while transmitting 250 Mbps over each wire pair with a Bit Error Rate (BER) of less than 10^{-10}. Standard transceiver architectures for 1000BASE-T are commercially available, and can accommodate the target BER at the 1 Gbps data rate.

At present, the next generation of Gigabit Ethernet (10 Gigabit Ethernet or 10GBASE-T) is an active area of investigation in both companies and universities. In order to standardize 10GBASE-T, an IEEE Study Group was formed in November 2003. The Study Group became the 10GBASE-T Task

Force in January 2004 [8].

1.2 10GBASE-T Ethernet

10GBASE-T is aimed to provide a throughput of 10 Gbps over CAT-5 or CAT-6 cabling with a BER of less than 10^{-12} [8]. 10GBASE-T is an upgrade for the existing 1000BASE-T standard and a competitor to 10 Gigabit Ethernet over fiber [9]. Similar to 1000BASE-T, 10GBASE-T uses four wire pairs and supports full-duplex operation, however, with an extremely short clock period (1-2 ns).

There are several major engineering challenges and performance issues in the design of 10GBASE-T systems. As will be addressed in the following chapters, a 10GBASE-T transceiver must operate under the following severe conditions [2,10,11]:

- Significant levels of Inter-Symbol-Interference (ISI) caused by the band-limited UTP channel.
- Echo from the local transmitter on the same wire pair.
- Near-End CrossTalk (NEXT) from the local transmitters corresponding to other adjacent wire pairs.
- Far-End CrossTalk (FEXT) from the remote transmitters of the adjacent wire pairs.
- Alien Near-End CrossTalk (Alien NEXT) from transmitters in a separate multidimensional transmission.
- Insertion loss going onto the UTP Cable.
- Colored noise due to the use of equalization or a precoding scheme.
- Residual ISI due to imperfect equalization or a precoding scheme.
- Noise from sources other than those listed above (*e.g.* $1/f$ noise caused by the loop filter in a phase locked loop).

It should be noted that by increasing the bit rate and hence the signal edge speeds, the amount of interference introduced by channel properties will tend to increase. Therefore, a higher amount of interference will be present in a 10GBASE-T channel in comparison with a 1000BASE-T channel. In addition, a shorter clock period (1-2 ns) and a lower BER is required for 10GBASE-T.

1.3 Thesis Organization

Based on the experience from 1000BASE-T receiver design, this project reviews and investigates various equalization and coding alternatives for Gigabit Ethernet and the corresponding issues for band-limited channels in order to achieve sufficiently low BER. It then addresses the performance evaluation of LDPC codes, including the candidate LDPC codes for 10GBASE-T, in the presence of channel impairments such as ISI, residual ISI, Additive Gaussian Colored Noise (ACGN), and $1/f$ noise.

Following this introduction chapter, the rest of this thesis is organized as follows. In Chapter 2 background material and concepts are presented. Chapter 3 describes the coding scheme used in 1000BASE-T as well as the alternative coding schemes for 10GBASE-T Ethernet systems. Chapter 4 presents the results of performance evaluation experiments for LDPC codes in the presence of ISI. Chapter 5 presents and discusses the effects of low-frequency and high-frequency ACGN on the performance of LDPC codes. Chapter 6 presents the results of performance evaluation of LDPC codes in the presence of $1/f$ noise. Finally, conclusions and future work are given in Chapter 7 .

Chapter 2

Background Material and Concepts

This chapter is dedicated to background material and concepts related to Gigabit Ethernet. Section 2.1 introduces the multidimensional constellation used in Gigabit Ethernet. Section 2.2 discusses channel and impairments models. Finally, Section 2.3 reviews various equalization methods and provides some simulation results.

2.1 Symbol Constellations in Ethernet

Gigabit Ethernet uses a Pulse Amplitude Modulation (PAM) constellation for signal modulation during data transmission. PAM (also known as Amplitude Shift Keying or ASK) is a type of AM constellation in which the signal is sampled at regular intervals to obtain a pulse whose amplitude is proportional to the amplitude of the signal at the instant of sampling. A PAM-M signal (with M levels of signal amplitude) can be defined as:

$$A_m = \{2m - 1 - M\}d, \qquad m = 1, 2, ..., M \tag{2.1}$$

where A_m is the amplitude of the m-th symbol and $2d$ is the distance between adjacent symbol amplitudes.

PAM has the advantage of simplicity and controlled bandwidth but, like other AM constellations, PAM has relatively high susceptibility to noise and interference. The reason for this susceptibility to noise is that any interference in the transmission path will either add to or subtract from signal voltage. As well, channel impairments that cause signal distortion and/or echos can also change the signal voltage. Since the amplitude of the voltage encodes the transmitted information, any unwanted change to the signal contributes directly to bit errors at the receiver. This property makes equalization and, possibly, error control methods critical parts in Gigabit Ethernet receivers.

2.1.1 The 1000BASE-T Constellation

As introduced in the previous chapter, 1000BASE-T employs full duplex baseband transmission over four pairs of CAT-5 UTP cabling. The throughput of 1 Gbps is achieved by transmitting at 250 Mbps over each wire pair. Each transmitted symbol on each wire pair is modulated using 5-level PAM (PAM-5) having 2 information bits per symbol [10]. The total number of levels required is $2^2 = 4$. The extra level provided in PAM-5 is used for coding and control purposes. The amplitude levels in PAM-5 are labeled -2, -1, 0, +1, +2 (± 2 actually maps to ± 1 Volt, and ± 1 maps to ± 0.5 Volt). Figure 2.1 illustrates the transmission scheme used in 1000BASE-T.

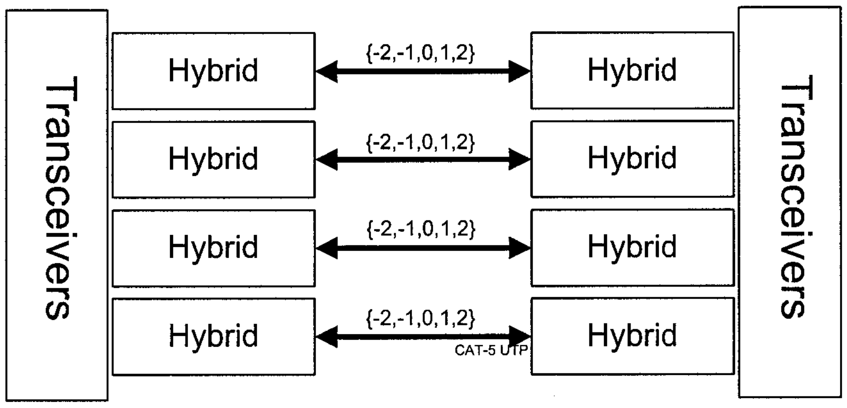

Figure 2.1: Transmission scheme in 1000BASE-T.

The combined output of all four transmitters on each wire pair forms a four-dimensional (4-D) symbol which carries eight information bits (*i.e.* 2 information bits per 1-D symbol on each wire pair). Therefore, the symbol rate is 125 MBaud/s. The 4-D constellation can be thought of as taking the possible outputs of each transmitter as an axis orthogonal (at right angles) to the other three axes. For every symbol period, one point in the constellation is sent. This joint constellation is referred to as 4-dimensional PAM-5 or 4-D PAM-5. By using 4-D PAM-5, $5^4 = 625$ distinct symbols can be generated, which provides enough symbol space to allow redundancy as well as special symbols for control purposes (in 1000BASE-T, 512 symbols are used for data and 113 is used for control signals) [12].

As mentioned before, PAM has a relatively high susceptibility to noise and interference. One way to alleviate this is to increase the distance between transmitted symbols, which makes them more distinguishable in the presence of noise and interference. For this reason, the PAM-5 constellation in 1000BASE-T is divided into two 1-D subsets $A = \{-1, 1\}$ and $B = \{-2, 0, 2\}$, leading to a minimum squared Euclidean distance, Δ^2, of 4 between any two points in these subsets[1]. In addition, in order to form the 4-D PAM-5 symbols, different combinations of the A and B subsets are grouped in a way that forms eight 4-D subsets $S0, ..., S7$. Each 4-D subset is sent over four wire pair and consists of both A-type and B-type 4-D symbols (see Table 2.1). 4-D subset partitioning guarantees the following properties [2]:

- The minimum square Euclidean distance of $\Delta^2 = 4$ between any two 4-D symbols in a same subset. For example, in the $S4$ subset, the squared distance between A-type symbol (1,-2,2,-1) and B-type symbol (0,-1,1,0) is $(1-0)^2+(-2-(-1))^2+(2-1)^2+(-1-0)^2 = 4$.

- The minimum squared Euclidean distance of $\Delta^2 = 2$ between any two 4-D symbols in either the even subsets $(S0, S2, S4, S6)$ or the odd subsets $(S1, S3, S5, S7)$. For instance, the square distance between 4-D symbol (0,0,0,1) from $S1$ and 4-D symbol (0,0,1,0) from $S3$ is 2.

As will be shown in the next chapter, this 4-D subset partitioning in combination with Trellis coding produces a significant Signal-to-Noise-Ratio (SNR)

[1]As an example, the minimum squared Euclidean distance for any two points in the subset A can be calculated as $\Delta^2 = (1-(-1))^2 = 4$.

improvement in 1000BASE-T.

Table 2.1: 4-D subsets in 1000BASE-T, $A = \{-1, 1\}$ and $B = \{-2, 0, 2\}$.

4-D Subset	A-Type	B-Type
$S0$	$AAAA$	$BBBB$
$S1$	$AAAB$	$BBBA$
$S2$	$AABB$	$BBAA$
$S3$	$AABA$	$BBAB$
$S4$	$ABBA$	$BAAB$
$S5$	$ABBB$	$BAAA$
$S6$	$ABAB$	$BABA$
$S7$	$ABAA$	$BABB$

2.1.2 The 10GBASE-T Constellation

10GBASE-T uses full duplex baseband transmission over four pairs of CAT-5 or CAT-6 UTP cabling. The data rate of 10 Gbps is achieved by transmitting at 2.5 Gbps over each wire pair. However, to achieve this target data rate, several dramatic improvements are required compared to the existing 1000BASE-T solutions. A straightforward extension of the techniques in the 1 Gbps specification to 10 Gbps is not realistic. For example, the use of a baud rate of at least 1.25 GBaud/s would be extremely difficult. As a result the constellation size has to increase beyond the PAM-5 used in 1000BASE-T [11].

At present, there are three proposals for the 10GBASE-T constellation, PAM-16, PAM-12 and PAM-8 [8]. The PAM-16 and PAM-12 proposals operate at 800 MBaud/s and 825 MBaud/s, respectively. In the PAM-12 proposal, eight signaling levels are used for data transmission and the extra four levels are used for coding and control purposes. PAM-16 and PAM-12 have the advantage of using lower baud rates than PAM-8, which operates at 1000 MBaud/s. However, PAM-8 might be able to tolerate more noise than them. Because PAM-8 maps sampled signal values into only eight distinct voltage levels, the difference between each level is greater than it would be for PAM-16 and PAM-12, which makes it easier for the receiver to determine the level of each sample.

2.2 Overview of Channel and Impairment Models

In general, a communications channel models the medium through which information is transmitted from a transmitter to a receiver. In many communications systems, the channel is often modeled by possibly frequency dependant attenuation of the transmitted signal, followed by additive noise. The attenuation captures the loss in signal power over the course of transmission. The noise in the model captures external interference and/or electronic noise in the transmission medium and in the receiver circuit. Electronic noise sources within the circuit can also be classified into two groups, namely, device noise and interference [13]. Thermal, shot, and flicker noise are examples of the former, while substrate and supply noise are in the latter group. Hence, depending on the application, the mathematical model for the communication system includes a model for the distortion introduced by the transmission medium (and the receiver circuit).

Figure 2.2 shows a typical communication channel model. The input sequence of the channel can be expressed as a power series in the delay operator D (assuming a D-transform) [14]:

$$x(D) = x_0 D + x_1 D^1 + x_2 D^2 + ... \tag{2.2}$$

Following the theory of Linear Time Invariant (LTI) systems, the received signal $y(D)$ can be expressed as:

$$y(D) = s(D) + n(D) = x(D)h(D) + n(D) \tag{2.3}$$

where $h(D)$ and $n(D)$ are the D-transforms of the channel impulse response and the channel noise signal, respectively, and $S(D)$ is the output signal from the channel before noise addition.

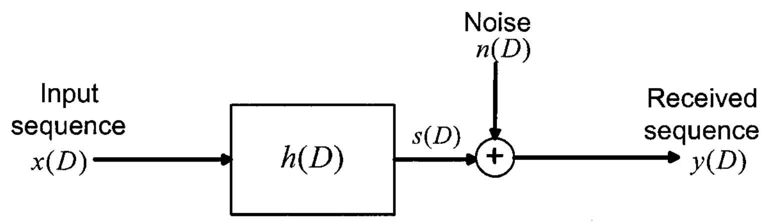

Figure 2.2: Channel model in the discrete-time domain.

2.2.1 Additive White Gaussian Noise

The Additive White Gaussian Noise (AWGN) channel is one of the simplest mathematical models for various physical communication channels, including wireline channels and some radio channels. An AWGN model is usually characterized by the mean and variance of the distribution. In this model the transmitted signal is assumed to corrupt by the addition of white noise with a Gaussian (normal) probability distribution of

$$P(x) = \frac{1}{\sigma_n\sqrt{2\pi}} e^{\frac{-(x-\mu)}{2\sigma_n^2}} \qquad (2.4)$$

where μ is the mean and σ_n^2 is the variance of the noise. Figure 2.3 shows the histogram of a typical AWGN sequence with $\mu = 0$ and $\sigma_n^2 = 1$ for 10^6 noise samples generated in Matlab. There are two major characteristics associated with AWGN:

- Statistical independence of any two noise samples.
- Constant power spectral density (*i.e.* the same distribution of power for all equal-sized frequency intervals).

Figure 2.4 illustrates the power spectrum of the same sequence. The spectrum gets flatter as the number of noise samples is increased.

Even if the AWGN channel is often used as a reference channel model in digital communication systems, it is not sufficient to describe real channels that, for example, suffer from Inter-Symbol Interference (ISI).

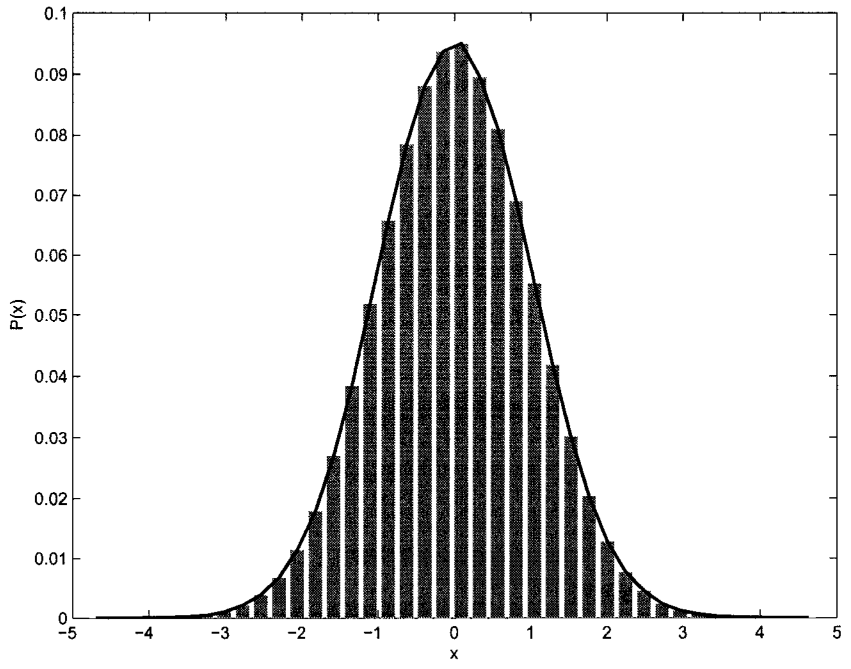

Figure 2.3: PDF of a typical AWGN sequence.

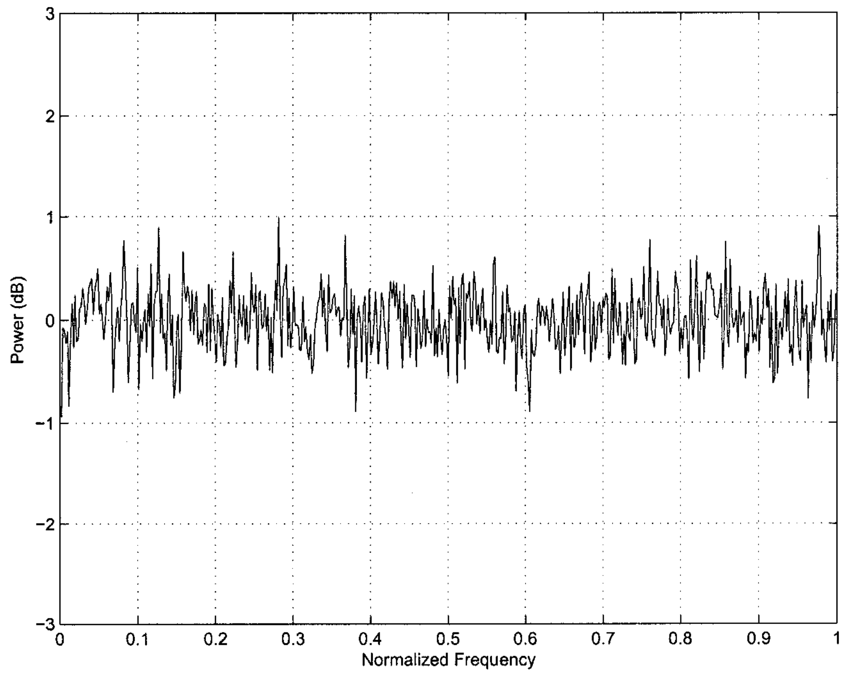

Figure 2.4: Power spectrum density of a typical AWGN sequence.

2.2.2 Inter-Symbol Interference

The process of transmitting an input sequence $x(D)$ through the channel $h(D)$ and obtaining a distorted signal $s(D)$ can be thought of as filtering the input signal. In practice, channels are band-limited. This is basically due to the frequency response characteristics of the communication medium. As an example, parasitic series and parallel capacitance in twisted pairs results in a limited frequency response channel model. A band-limited channel is usually modeled as a low-pass filter. In practical channels the inevitable filtering effects and channel distortion tends to cause a spreading or smearing out of individual data symbols passing through a channel (Figure 2.5). For consecutive symbols, this spreading causes part of the symbol energy to overlap with neighboring symbols causing a noise at the sampling time that is called ISI (Figure 2.6). In addition to band-limited channels, transmitter filtering (when channel spacing is crucial) and multi-path reflection in wireless channels can be additional sources of ISI. Figure 2.7 illustrates the postcursor ISI of a CAT-5 UTP channel referred to in the 1000BASE-T standard [2]. Figure 2.8 illustrates the discrete time impulse response of a typical CAT-6 UTP channel [3], which is an alternative medium for the 10GBASE-T standard. This channel has severe delay spread over many symbol intervals, resulting in severe ISI.

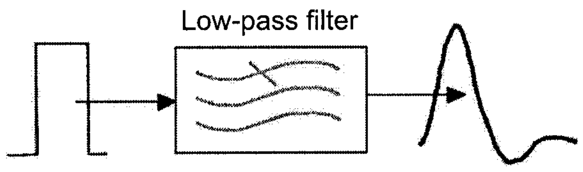

Figure 2.5: Channel as a low-pass filter.

When ISI exists in a channel, the channel behaves as if it has memory because each symbol value passed through the channel becomes correlated with previous symbols (postcursor symbols) and following symbols (precursor symbols). ISI can significantly degrade the ability of the data detector to differentiate the present symbol from the diffused energy of the adjacent inter-

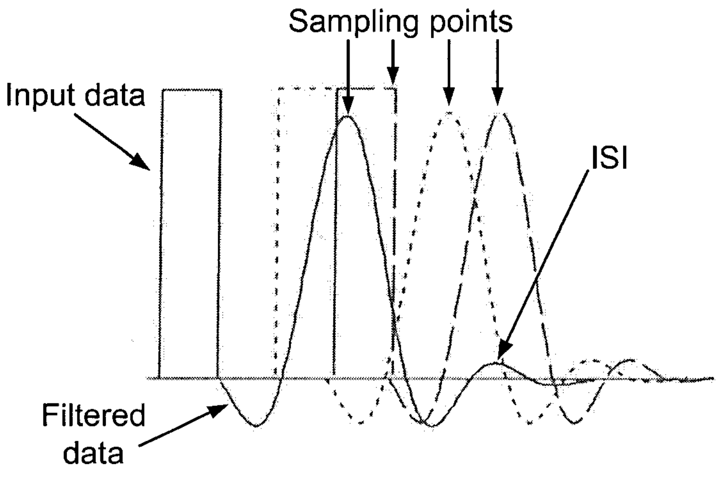

Figure 2.6: ISI in a non-ideal channel.

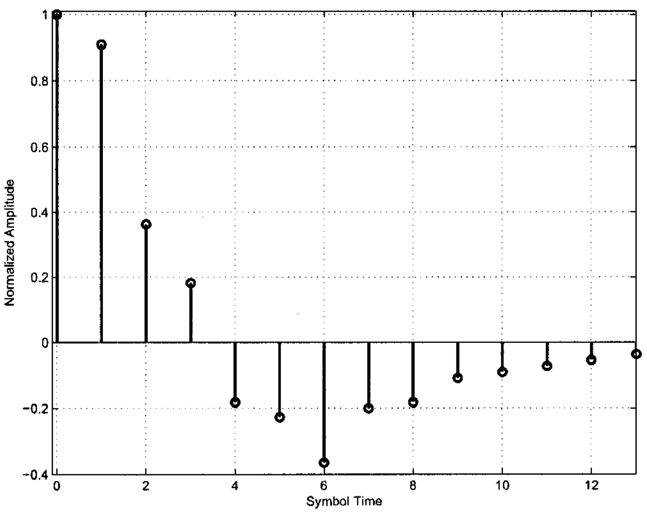

Figure 2.7: Postcursor ISI for CAT-5 UTP [2].

fering symbols. Even with no noise present in the channel, ISI alone leads to the occurrence of errors produced at the so-called irreducible error rate. These errors will degrade the bit and symbol error rate performance. To compensate for the effects of ISI, communication systems use fixed and/or adaptive channel equalizers. Equalization techniques will be discussed in Section 2.3.

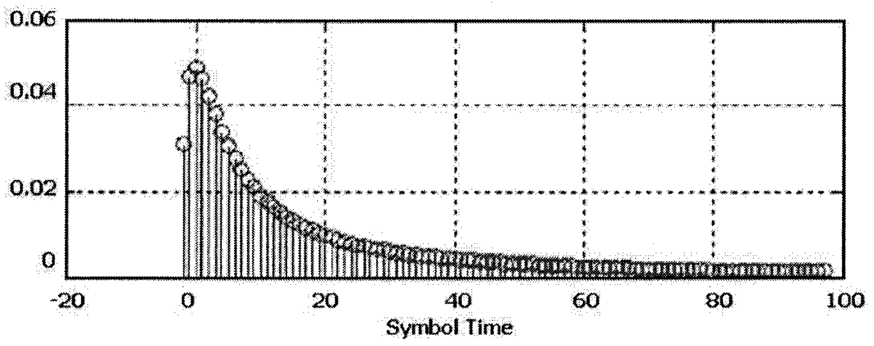

Figure 2.8: Channel impulse response for CAT-6 UTP [3].

2.2.3 Residual ISI

Residual ISI is mainly due to mismatch between the transmission channel and the equalizer or precoder [15, 16]. Such a mismatch can be a result of:

- Equalizers whose finite length limits their possible impulse response behaviour.
- Quantized (as opposed to infinitely precise) equalizer coefficients.
- Channel estimation error.

Residual ISI is among the important impairments for any high-speed wireline standard like 10GBASE-T Ethernet [17]. In practice, this phenomenon is unavoidable and there will always exist some level of residual ISI in any real wireline system.

2.2.4 Colored Noise

Most of the communication system models are usually analyzed under the assumption that the noise in the system is independent of the transmitted signals and that the noise components of any two samples are independent (*e.g.* AWGN). However, in many communication systems, the dominant noise sources are actually colored [17, 18], making colored noise an important impairment that exists in practical communication channels.

Colored noise is also a major impairment in the 10GBASE-T Ethernet. 10GBASE-T needs to implement an equalization scheme at the receiver to overcome the ISI caused by severe delay spreads in the CAT-6 UTP channel (see Figure 2.8). But such an equalization scheme results in correlated noise. Even when an equalizer is provided at the transmitter (*e.g.* in Tomlinson-Harashima precoding [19,20]), 10GBASE-T still needs to use various filtering schemes at the receiver to overcome the effects of residual ISI and other impairments, such as intra-cable crosstalk. In addition, the coding scheme in 10GBASE-T spans across four copper wire pairs which are in the same physical bundle and pass through the same connectors. All four wire pairs are latched into the same transceiver chip. Also transmitters on all four wire pairs are clocked by the same clock, which slightly correlates the noise sources in both directions.

Another important source of colored noise for 10GBASE-T is a new type of crosstalk that has been called Alien Near-End CrossTalk (ANEXT). ANEXT refers to signal energy that couples from one 4-pair UTP cable to another 4-pair UTP cable. This type of crosstalk is due to the fact that at high operating frequencies, a UTP cable starts to behave like an antenna, radiating and picking up energy from the surroundings [11]. This was not a problem in the earlier, lower data rate versions of Ethernet such as 1000BASE-T. However, ANEXT is predicted to be one of the major impairments for 10GBASE-T systems. According to [11], issues associated with ANEXT include the lack of synchronization between the symbol generator and the receiver, which implies a cyclo-stationary relationship. In addition, ANEXT characteristics may change abruptly and dramatically as cables at a switch box are plugged in and unplugged. These features make ANEXT cancelation impractical at present. Figure 2.9 illustrates ANEXT coupling in a multi-cable bundle and Figure 2.10 shows the measured spectrum of ANEXT for 10GBASE-T systems [4].

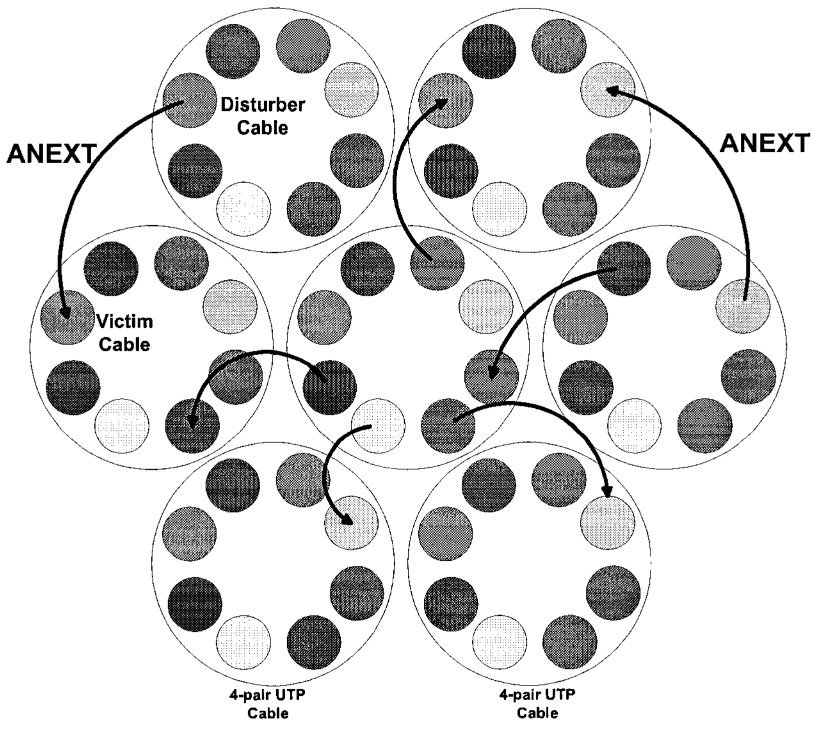

Figure 2.9: ANEXT in a multi-cable bundle.

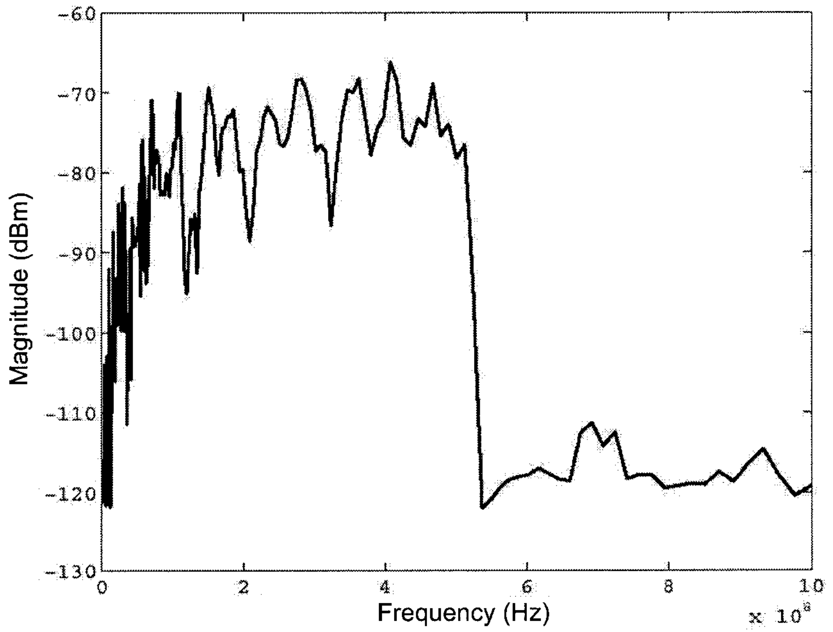

Figure 2.10: Spectrum of ANEXT in a 10GBASE-T system [4].

2.2.5 $1/f$ Noise

$1/f$ noise (also known as flicker or pink noise) is a type of low-frequency colored noise [21]. $1/f$ noise is ubiquitous in nature. It is observed in solid-sate circuits, fractals and music, *etc.* For analog solid-state circuits, in particular, $1/f$ noise becomes increasingly important because it can limit the channel minimum spacing in communication systems when it is upconverted to phase noise in oscillators and in mixers [13]. Even in the digital world, phase noise in the guise of timing jitter is important. Clock jitter directly affects the timing margins and hence limits system performance [13]. In addition, low frequency noise is very sensitive to the technological processes and parameters which are used in IC fabrication [22–25].

$1/f$ noise is characterized by a power spectrum that falls like $1/f$:

$$S(f) = \frac{a}{f} \tag{2.5}$$

where a is a gain factor and $S(f)$ is the power spectrum of the $1/f$ noise. This spectrum is characterized by a 3 dB per octave drop (*i.e.*, 10 dB per decade), which means that whenever the frequency doubles, the amplitude drops by a factor of 1/2:

$$S(2f) = \frac{a}{2f} = \frac{1}{2}S(f) \rightarrow 10\log_{10}\left[\frac{S(f)}{S(2f)}\right] = 10\log_{10}(2) = \ 3 \quad \text{dB} \tag{2.6}$$

This also implies that the amount of power contained in any octave interval $[f_1, f_2]$, where $f_2 = 2f_1$, is the same:

$$\int_{f_1}^{f_2} S(f)df = a\ln(\frac{f_2}{f_1}) = a\ln(\frac{2f_1}{f_1}) = a\ln(2) \tag{2.7}$$

Figure 2.11 shows the power spectrum of a typical $1/f$ noise sequence with a variance of 1, and mean of 0. The power spectrum data was calculated in Matlab by using 10^7 samples. Figure 2.12 shows the same spectrum in *log*-scale. The 10 dB per decade drop is apparent in this figure.

There exist various methods for generating $1/f$ noise. The method used in this thesis is discussed in Chapter 6. Discussion about other methods can be found in [26–28].

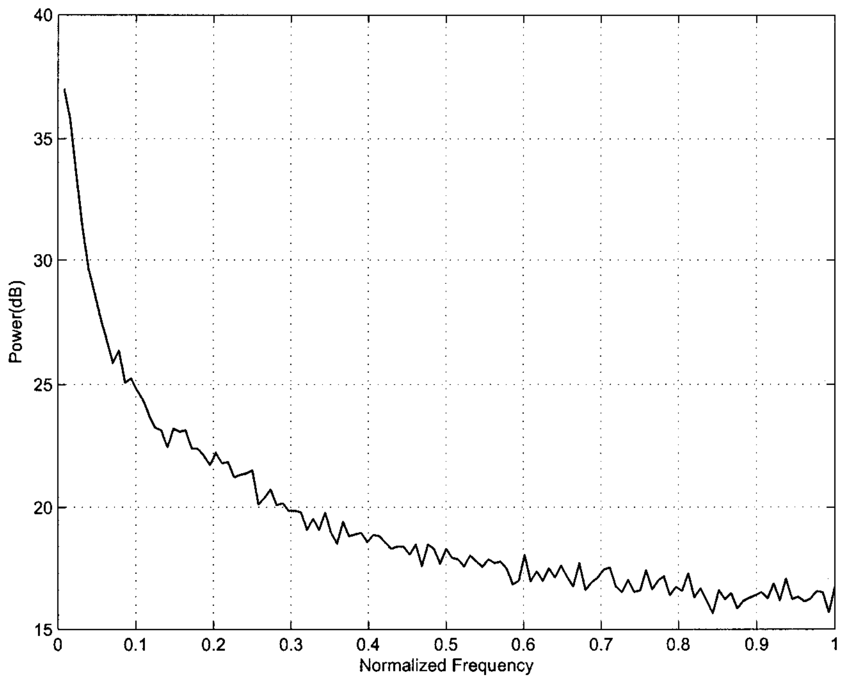

Figure 2.11: Power spectrum of $1/f$ noise.

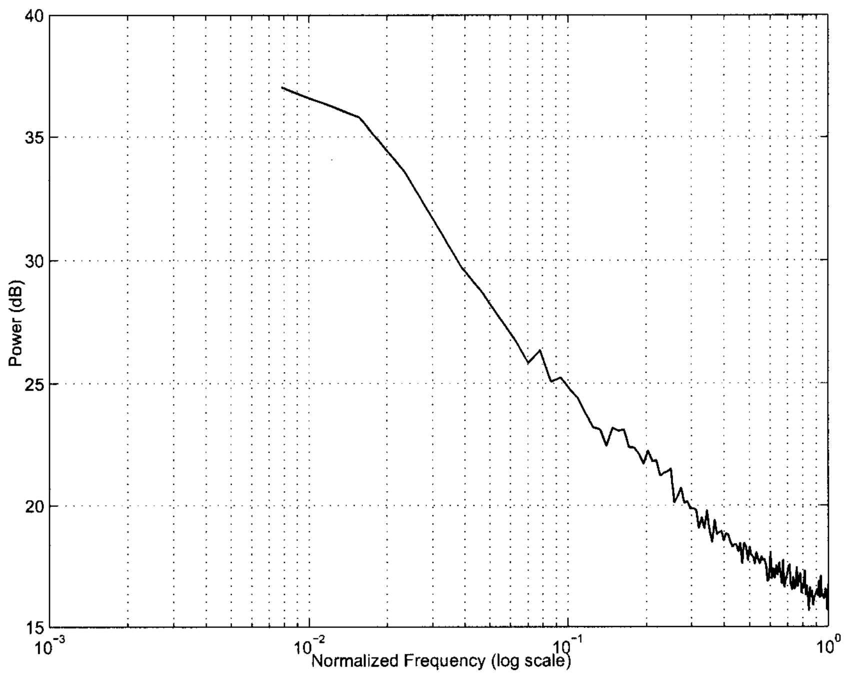

Figure 2.12: Power spectrum of $1/f$ noise (log scale).

2.2.6 Insertion Loss, Echo and Crosstalk

In addition to the previously discussed impairments, there are some other important impairments in Gigabit Ethernet systems, which are due to the multi-dimensional transmission scheme. Figure 2.13 illustrates the following impairments in a typical Gigabit Ethernet architecture.

Insertion Loss: This refers to the loss of signal power between two points along a cable. Insertion Loss is usually expressed as the reciprocal of the ratio of the signal power delivered to one point to the signal power delivered to the other point.

Echo: Echo is caused by an impedance mismatch between two points of medium. It is dominated in 10GBASE-T due to the mismatch introduced by the hybrids.

Crosstalk: This impairment is due by signal coupling from one pair of UTP cable to another pair. It can occur at the near end of the transmitter (NEXT), the farther end of the transmitter (FEXT), or as described in Section 2.2.4, from a remote transmitter (ANEXT).

The above mentioned impairments do not have the same effects in all XBASE-T Ethernet standards. However, the effect of all of them tends to be more significant in 10GBASE-T systems [11]. Figure 2.14 shows the frequency response of these impairments over CAT-5 UTP in a 10GBASE-T Ethernet system [5].

2.3 Equalization Methods

The process of compensating for the ISI and other channel impairments is called equalization. Equalization is usually performed by a filter, known as channel equalizer, which effectively flattens the channel transfer function and hence improves the recovery of the transmitted symbols. There are various types of equalizers which are different in structure, complexity, performance

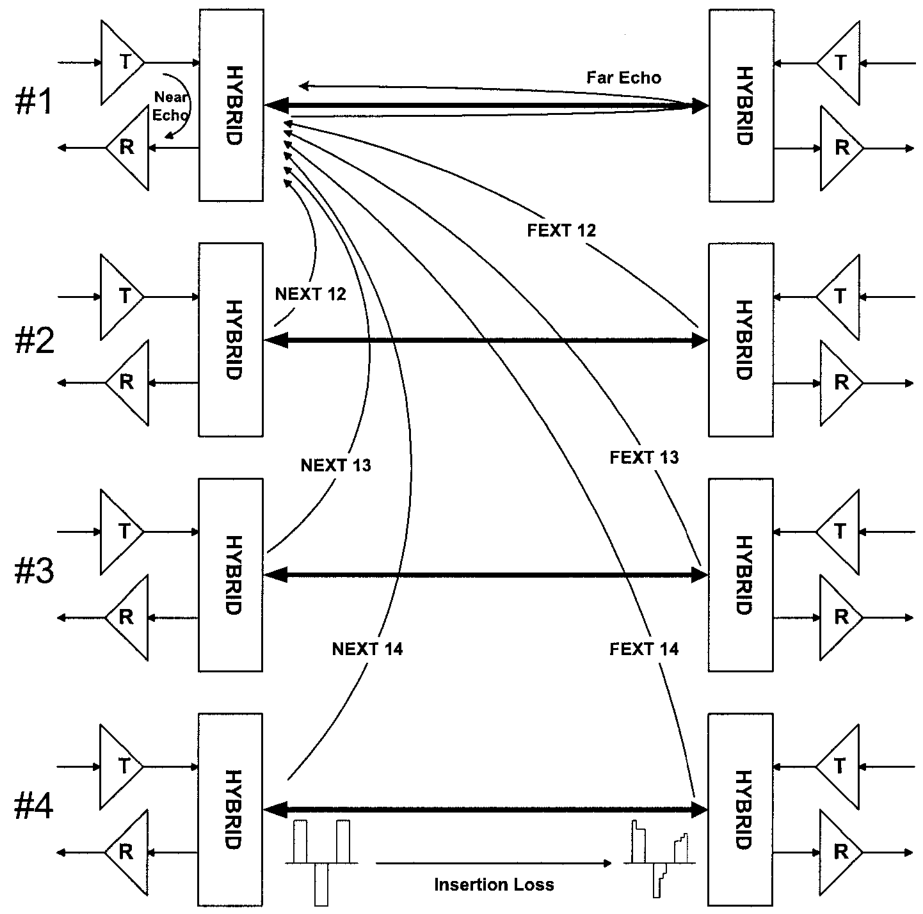

Figure 2.13: Insertion Loss, NEXT, Echo and FEXT in Gigabit Ethernet.

and the situations in which they can operate (*e.g.* channel conditions). In addition, equalizers can be categorized as being either preset (fixed) or adaptive equalizer. In preset equalizers, all the parameters are set and don't change during equalizer operation. Therefore, this type of equalizer is suitable for situations where the channel behavior is known a priori. Adaptive equalizers, on the other hand, are suitable in situations where the channel characteristics may vary over time since adaptive equalizers are capable of updating their parameters according to channel variations. This section gives an overview of classical equalization methods, their benefits and their drawbacks.

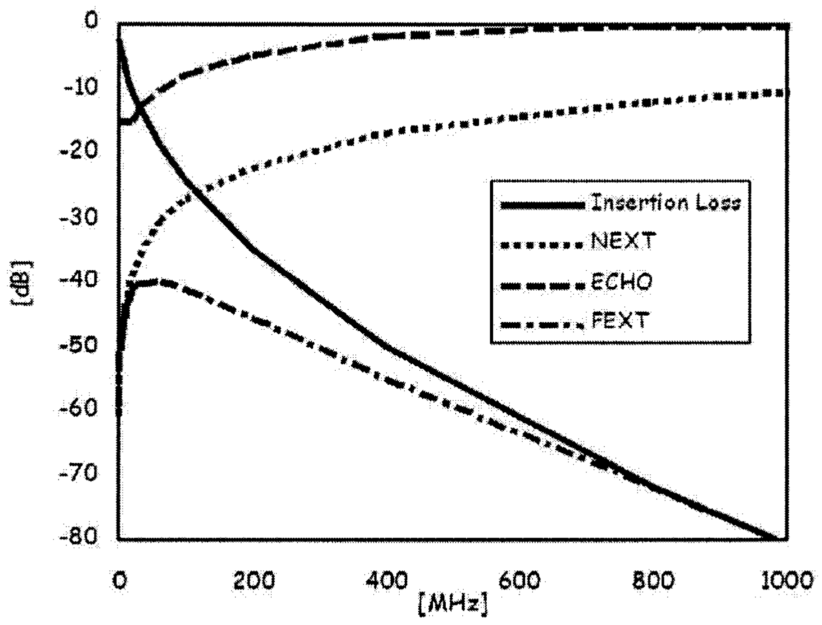

Figure 2.14: Frequency response for Insertion Loss, NEXT, Echo and FEXT for CAT-5 UTP cabling [5].

2.3.1 Linear Equalization

In the linear equalization method, the output of the equalizer is a linear combination of the present and post equalizer inputs. The equalizer is implemented using a Finite Impulse Response (FIR) filter. The coefficients of such a filter can be constant (preset scheme) or updated during equalization (adaptive scheme). Figure 2.15 shows the structure of a preset linear equalizer, where D is a delay element, N is the order of the filter, a_i is the i-th coefficient of the filter, $y(n)$ is the received signal and $z(n)$ is the equalized signal.

In the preset scheme the equalizer coefficients are calculated to estimate the inverse transfer function of the channel (*i.e.* $1/h(D)$). This method is also known as *zero-forcing* linear equalization as it can completely eliminate the ISI leaving only an AWGN component, $n'(D)$:

$$z(D) = y(D)/h(D) = (x(D)h(D) + n(D))/h(D) = x(D) + \underbrace{n(D)/h(D)}_{n'(D)} \quad (2.8)$$

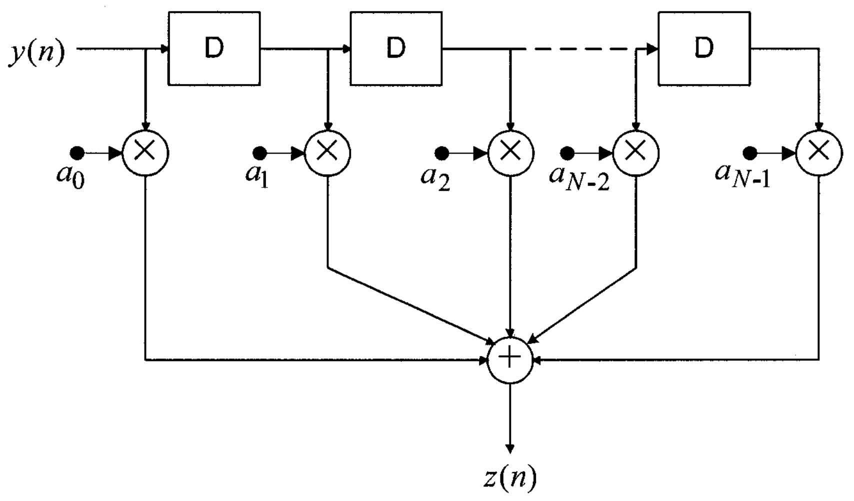

Figure 2.15: Linear equalizer.

However, despite this advantage and its simple structure, this method has the following drawbacks:

- The noise energy experiences a gain of $||1/h||^2$.
- Filtering the AWGN, $n(D)$, makes it colored $(n'(D))$.
- Not all channel transfer functions have a well-defined inverse $1/h(D)$.

Due to these problems, another type of equalizer, known as a Decision Feedback Equalizer, is used in many high data rate communication systems.

2.3.2 Decision Feedback Equalization

The equalization scheme used in 1000BASE-T Ethernet is based on decision feedback equalization. A Decision Feedback Equalizer (DFE) is a nonlinear equalizer that employs previous decisions to eliminate the ISI caused by previously detected symbols on the current symbol to be detected [29]. A block diagram for a typical DFE is shown in Figure 2.16. The DFE structure can be thought of as a special type of Infinite Impulse Response (IIR) filter. The first filter in a DFE is called the feedforward filter. This filter is generally a Finite Impulse Response (FIR) filter with adjustable coefficients. The input for this filter is the received signal $y(D)$. The second filter is called the feedback filter and it is also an FIR filter with adjustable coefficients. Its input is the sequence of previously detected symbols. The output of the feedback filter is subtracted from the output of the feedforward filter to form the input to the detector. The detector determines which of the possible transmitted symbols is closest to the input signal (*i.e.* received symbol). Thus, it makes a decision and outputs the corresponding decision. What makes the DFE nonlinear is the nonlinear characteristic of the detector that provides the input to the feedback filter.

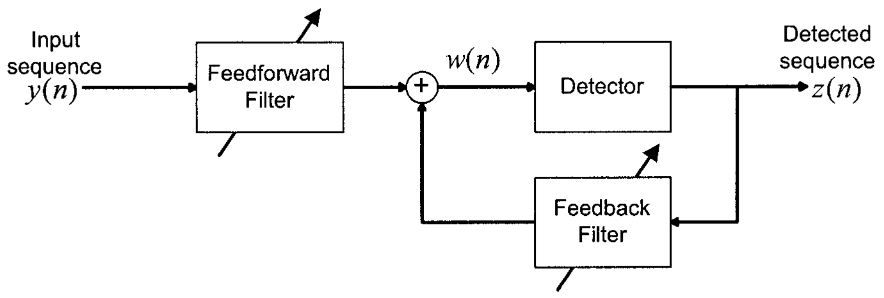

Figure 2.16: Decision feedback equalizer.

Let the N be the order of the feedforward filter, M the order of the feedback filter, and $\{a_i\}$ and $\{b_i\}$ the coefficients of the feedforward and feedback filters. The input of the detector can then be expressed as:

$$w(n) = \sum_{k=1}^{N} a_k y(n-k) - \sum_{l=1}^{M} b_l z(n-l) \tag{2.9}$$

2.3.2.1 Adaptive DFE

In Gigabit Ethernet the adaptive version of DFE is used. In the adaptive DFE scheme, the tap coefficients of the feedforward and feedback filters are selected to optimize some desired performance measure. For the sake of simplicity as well as lower hardware cost, the Least Mean Square (LMS) algorithm (also known as the stochastic gradient algorithm) [29,30] is usually applied for adaptation (see Figure 2.17). In each iteration of this adaptive scheme the following steps are applied:

i) The new input sequences for both the feedback and feedforward filters are updated.

ii) The difference between detector input and output is calculated as an error value.

iii) Based on the error, the filters' inputs and the step size parameter, the filter coefficients are updated.

To ensure convergence of this iterative procedure, the step size μ is chosen as a small positive number to scale down the error signal. The following matrix equations formulate the above steps [30]:

$$\begin{array}{ll} \text{i)} & \Psi(n) = U(n) + \sum_{l=1}^{M} b_l \Psi(n-l) \\ \text{ii)} & e(n) = z(n) - w(n) \\ \text{iii)} & C(n+1) = C(n) + \mu \times e(n)\Psi(n) \end{array} \tag{2.10}$$

where μ is the step size and $U(n)$ and $C(n)$ are two matrixes containing the filters' input sequences and coefficients, respectively, as follows:

$$\begin{array}{l} U(n) = [\ \{y(n-k)\}\ \ \{w(n-l)\}\ ] \\ C(n) = [\ \{a_k\}\ \ \{b_l\}\ ] \quad k = 1, ..., N \quad l = 1, ..., M \end{array} \tag{2.11}$$

When the DFE starts to operate, its coefficients are given initial values that are not the optimal coefficients. Hence, the DFE cannot initially make reliable decisions and the filter coefficients are updated by a nonzero error value. Therefore it is possible that the incorrect decisions will propagate through the DFE and hence the DFE will continue to make its next iterations based on previous wrong decisions. This phenomena known as *error propagation* [16, 31]. Error propagation phenomena can also happen during the normal mode operation of a DFE by making large number of consequent errors decisions. To prevent this practical problem at the beginning of the equalization, it is common to use a training sequence, which is a sequence of data symbols known in advance by both the receiver and transmitter. Thus, the equalizer has two operating modes. In training mode, the error $e(n)$ is calculated as the difference between the actual transmitted data (*i.e.* the predetermined training

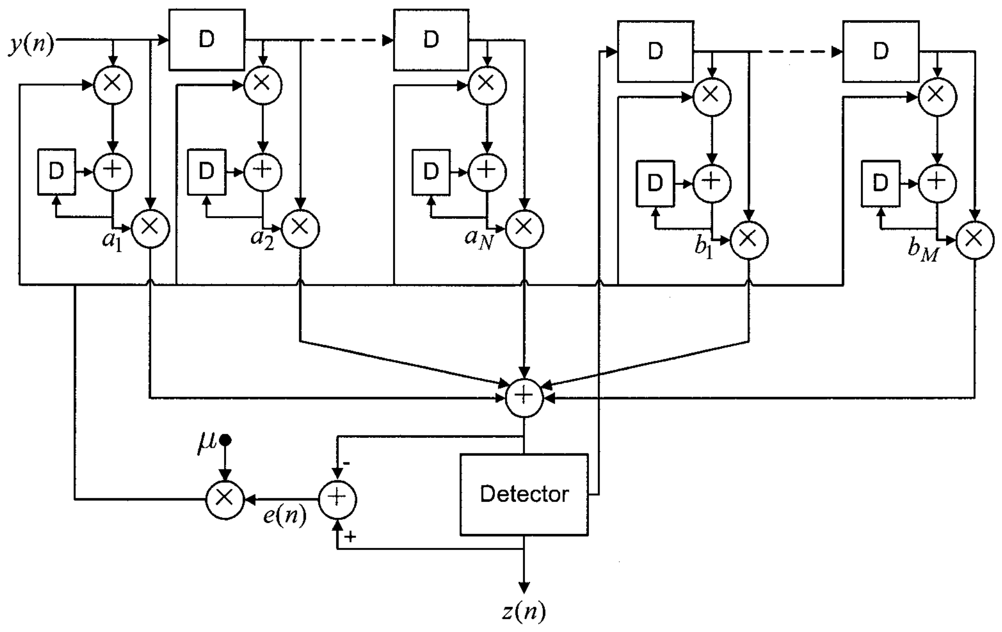

Figure 2.17: Adaptive decision feedback equalizer.

sequence) and the detector input. Training mode occurs when the detector is out of service. After an adequate number of iterations, the filters' coefficients converge to the optimum values. Then, the equalizer can switch to the normal (also called decision-directed) mode and use its detector to make decisions. The length of the training sequence (*i.e.* the number of iterations needed for convergence) as well as the maximum amount error after adaptation depends on the step size and channel characteristics. The smaller the step size the less the error but the more iterations that are required to converge. A rule of thumb for selecting the step size parameter to ensure convergence and good tracking capabilities in slowly-varying channels is [32]:

$$\mu = \frac{1}{5(M+N)E_y} \tag{2.12}$$

where E_y is the received signal-plus-noise power estimated from the received signal, y.

In general, the advantages of using an adaptive scheme, such as the LMS algorithm, for DFE can be summarized as follows:

- Simplicity of implementation.

- Accounts for noise, unlike the zero forcing equalizer.
- Avoids the need to calculate the inverse channel, $1/h(D)$.

From the optimization point of view, the LMS criterion in DFE tries to minimize the Mean Square Error (MSE). However, there is no guarantee that the DFE will indeed converge to the global minimum of the MSE function. It may happen that it converges to only a local minimum and gets stuck there because the step size is too small. In such cases, choosing a proper value for the step size is essential. Adding noise to the received input may help DFE to escape a local minimum and then converge to the global minimum, but this is not necessarily the case.

In comparison with linear equalizers, which are either adaptive or preset FIR filters, the DFE can achieve a better performance [14]. However, it is not the optimum equalizer from viewpoint of minimizing the probability of error in the detection of the information from the received signal samples [29]. From digital communications theory, optimum detection is done by using a Maximum Likelihood Sequence Estimator (MLSE). An MLSE outputs the most probable symbol sequence for the given received sampled sequence. This method will be discussed in the next chapter.

Due to its superior performance over linear equalizers [33], especially when the channel introduces severe signal attenuation within specific frequency regions, the DFE is preferred in many digital communication applications. However, there are some drawbacks for DFE. It suffers from error propagation effects. Error propagation generally does not severely affect DFE performance if the channel delay spread is on the order of a symbol duration or less [33,34]. However, to accommodate high data rates, the associated increase in relative delay spread may result in error propagation for a given channel [31]. Therefore, error propagation might result that causes bursts of decision errors and a corresponding increase in the average probability of bit and symbol errors in systems like 10GBASE-T. In addition to the error propagation problem, coding schemes such as LDPC coding cannot be applied in a straightforward manner in a DFE [33,35]. The difficulty arises because the DFE requires zero-delay decisions, which is incompatible with the idea of channel coding. These problems can be avoided by exploiting the feedback filter in the DFE at the transmitter and by introducing a nonlinearity for power limitation. This approach is in fact the basis for Tomlinson-Harashima Precoding (THP) [19,20]. THP is discussed in Section 2.3.3

2.3.2.2 Performance Evaluation of DFE in 1000BASE-T Ethernet

Figure 2.18 plots the simulation results that show the convergence of PAM-5 DFE for CAT-5 UTP channel in 1000BASE-T (see Figure 2.7). In the simulation study, the training sequence consists of 100K symbols and the step size, μ, was 0.0001. As shown in Figure 2.18(A), at the beginning the MSE is high but after about 50000 iterations, the DFE coefficients have largely converged to stable values and the MSE becomes much less. At the same time the PAM-5 equalized symbol becomes distinct. Figure 2.18(B) shows this behavior clearly. After transmission of the training sequence, the equalizer switches to its normal mode operation. At this time it is either possible to continue the adaptive scheme (decision directed) or turn off the adaptive scheme and work with constant filter coefficients. The former form is suitable for channels which might vary with time.

Figure 2.19 shows the result of a BER performance evaluation for a conventional DFE scheme using the 1000BASE-T 4-D PAM-5 constellation. It was assumed that the DFE cancels ISI due to 14 postcursor channel taps, *i.e.*, the channel memory is 14 (CAT-5 UTP channel in Figure 2.7). The symbol detector in the simulation study was a PAM-5 slicer which makes hard decisions on the equalized symbol. The SNR in this simulation is defined as $10log10(E_S/E_N)$, where E_S is the average symbol energy for a 4-D PAM-5 symbol and E_N is the AWGN variance. As illustrated, the DFE can achieve BER in order of 10^{-7} at SNR = 23 dB. Such performance is considerable, but it is actually far from the 10^{-10} target BER for 1000BASE-T. Therefore, as will be discussed in the next chapter, to satisfy the target BER, a more pow-

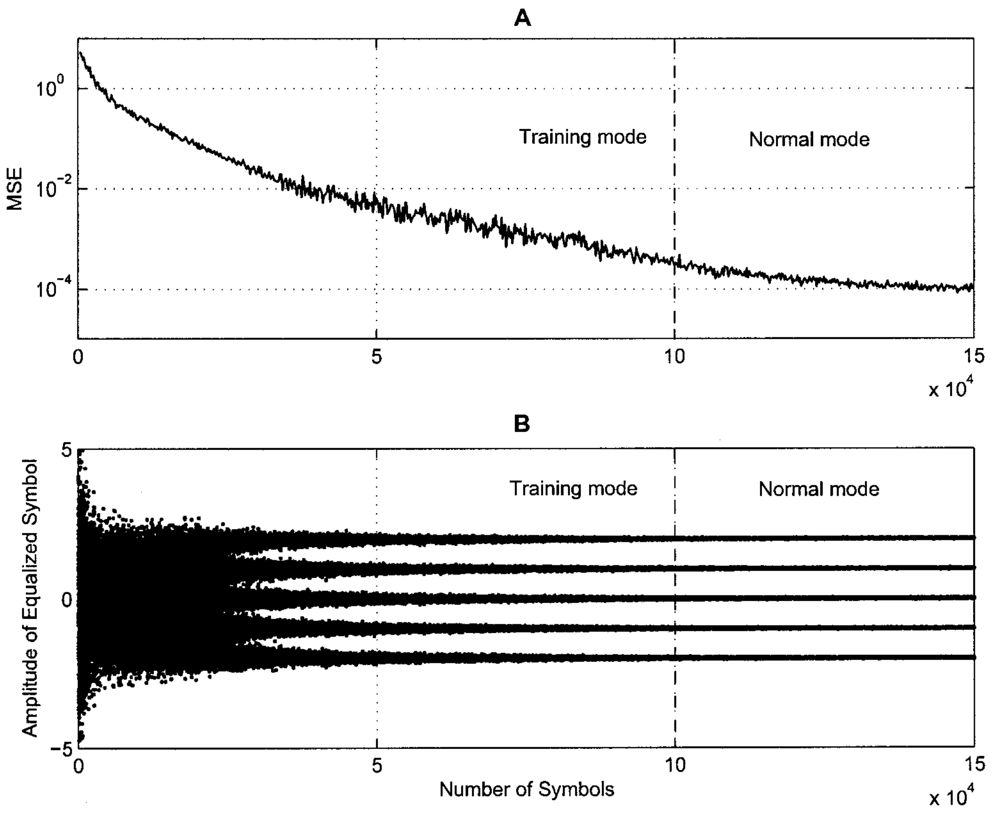

Figure 2.18: Convergence of PAM-5 DFE: A) MSE, B) Equalized PAM-5 symbols.

erfull joint equalization and coding scheme has been used in the 1000BASE-T standard.

2.3.3 Tomlinson-Harashima Precoding

The THP structure [19, 20] is illustrated in Figure 2.20. Unlike other equalization techniques in which equalization takes place in the receiver, THP is performed as a form of equalizing predistortion at the transmitter. In THP, it is assumed that the channel response $h(D)$ is known at the transmitter. The transmitter generates a data symbol, $d(n)$, using PAM-M. The transmitted symbol $x(n)$ is then formed by subtracting the ISI introduced by previously transmitted signal (*i.e.*, decision feedback in the transmitter) and then performing a modulo-$2M$ operation. The modulo-$2M$ operation limits $x(n)$ to

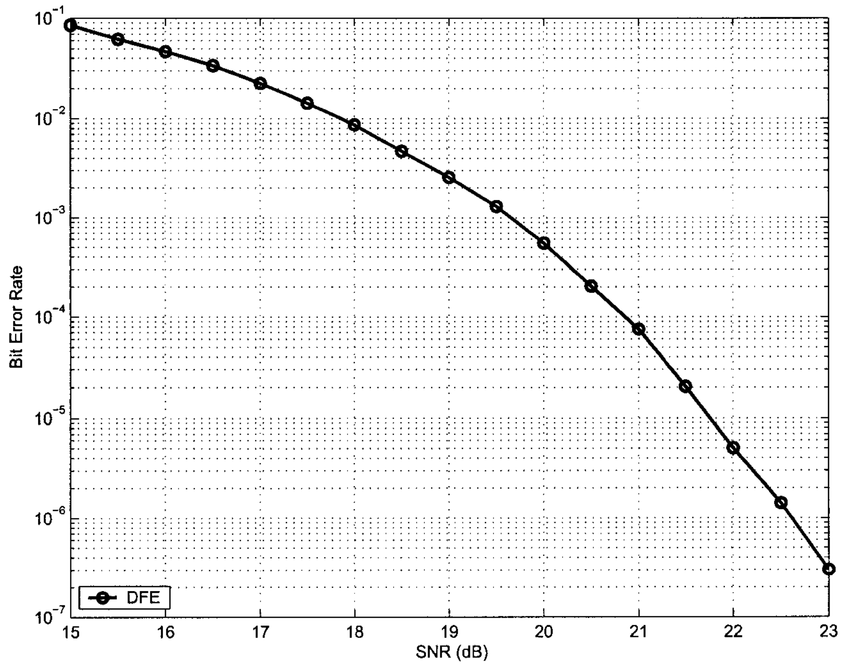

Figure 2.19: BER performance of a DFE in a 1000BASE-T Ethernet system.

the interval $(-M, M]$ and can be expressed as:

$$x(n) \overset{2M}{\equiv} d(n) - \sum_{i=1}^{N} h(i)x(n-i) \rightarrow \begin{cases} x(n) = d(n) + 2Mv(n) - \sum\limits_{i=1}^{N} h(i)x(n-i) \\ x(n) \in (-M, M] \end{cases} \tag{2.13}$$

where $v(n)$ is the unique integer that satisfies the above relationships. Equivalently, in D-transform notation the transmitted sequence, $x(D)$, can be expressed as:

$$x(D) = d(D) + 2Mv(D) - x(D)[h(D) - 1] \rightarrow x(D) = [d(D) + 2Mv(D)]/h(D) \tag{2.14}$$

Consequently, the received signal is an ISI-free signal:

$$y(D) = x(D)h(D) + n(D) = \underbrace{d(D) + 2Mv(D) + n(D)}_{\text{ISI-free}} \tag{2.15}$$

The original data symbols can be retrieved by reducing $y(n)$ to the interval $(-M, M]$ with a modulo-$2M$ operation.

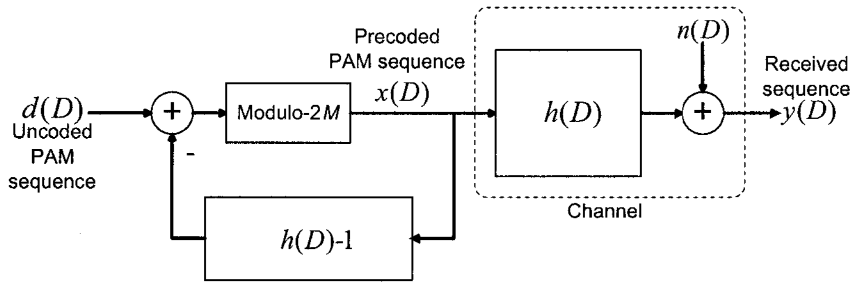

Figure 2.20: Tomlinson-Harashima precoding.

THP is a practical solution to overcome the DFE error propagation problem in a receiver equalizer. It has been widely used as an alternative to the DFE, *e.g.*, in digital subscriber loop (xDSL) systems, and it has been selected as an alternative equalization scheme for 10GBASE-T. The benefits of THP can be summarized as follows:

- Since channel equalization is performed at the transmitter, error propagation is circumvented.
- Coding techniques can be exploited in the same way as for channels without ISI.
- THP reduces the complexity of the receiver by bringing the equalization back to the transmitter. This is advantageous for applications like 10GBASE-T, in which many other computationally-expensive tasks are performed at the receiver.

There are also some drawbacks with THP. The precoded signal exhibits a huge dynamic range for a satisfactory degree of noise whitening, especially if the discrete time channel includes spectral nulls [35]. These dynamics cause increased sensitivity to equalization and symbol clock jitter. In addition, a transmitter with perfect THP requires knowledge of the actual channel characteristics for satisfactory precoder design [16, 35]. However, since a perfect match between the channel and precoder is not possible in practice, there will be always be some residual ISI.

2.3.4 Delayed Decision Feedback Sequence Estimation

Figure 2.21 shows the structure of a Delayed Decision Feedback Sequence Estimator (DDFSE). As shown in this figure, DDFSE [36] combines the structure of MLSE (*i.e.* the Viterbi algorithm) and DFE. Such a structure tends to have a superior performance than a DFE and trades off the exponentially increasing complexity of the Viterbi algorithm and MLSE performance. DDFSE has been used in 1000BASE-T Ethernet. The Viterbi algorithm and DDFSE are discussed in greater detail in the next chapter.

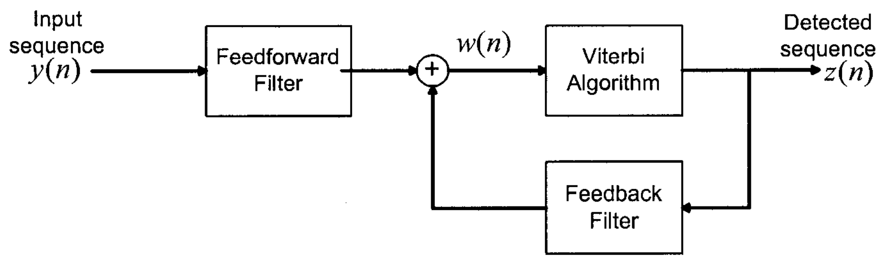

Figure 2.21: DDFSE structure.

Chapter 3

Coding Schemes in Gigabit Ethernet

The main purpose of coding is to add redundant structure to the transmitted data to allow the receiver to detect and correct errors introduced during passage through a noisy and distorting channel. In general, in order to decrease the effect of errors and achieve reliable communication, it is necessary to transmit sequences that are as different as possible, in a Hamming distance sense, so that the channel noise will be less likely to change one valid sequence into another. However, the introduction of redundancy results in the transmission of extra bits and thus a reduction of the information transmission rate. Thus there is a tradeoff between the transmission rate and the reliability of communication. Such a tradeoff is very important for communication systems such as 10GBASE-T, in which reliable high data rate transmission needs to be guaranteed over the band-limited UTP channel.

Channel coding schemes can be divided into two classes, convolutional codes and block codes [29]. These two schemes are reviewed in this chapter through the discussion of 1000BASE-T and 10GBASE-T Ethernet coding schemes, respectively.

3.1 The 1000BASE-T Ethernet Coding Scheme

As briefly mentioned in the previous chapter, 1000BASE-T uses a joint coding and equalization scheme known as DDFSE. In general DDFSE structure in 1000BASE-T can be considered as a combination of the *Viterbi Algorithm* (VA) and DFE. The VA is a famous algorithm for the optimal decoding of convolutional codes [37,38] as well as optimal detection of data sequences distorted by ISI [39–42]. This section firstly introduces convolutional codes, their encoding methods and the VA. It then discusses the usage of the VA in 1000BASE-T as an optimal detection algorithm for ISI-distorted data sequences.

3.1.1 Convolutional Coding

In convolutional coding, each block of k information bits is mapped into a block of n bits, which are not only determined by the present k information bits, but also by the previous information bits. The term "convolutional" is used because the encoded output sequence is generated by the convolution of the input sequence and a generator sequence. As will be discussed in the following sections, the dependence on the previous information bits introduces memory into the encoding scheme and indeed causes the encoder to behave as a Finite State Machine (FSM).

3.1.1.1 Encoding of Convolutional Codes

The encoder in a convolutional coding scheme is usually a sequence generator based on a Linear Feedback Shift Register (LFSR). Convolutional encoding is accomplished by multiplexing two or more different convolutions of the same source data onto a channel. This process is done in a continuous manner with the use of shift registers and modulo-2 adders. These modulo-2 adders are XOR gates whose inputs are various combinations of the shift register state bits and whose outputs are multiplexed together to form the output stream.

Figure 3.1 shows the block diagram of a typical convolutional encoder with Lk stages, where L is called the constraint length of the code and "+" implies the XOR operation. To input the next block of information, k information bits enter the LFSR and the contents of the last k stages of the LFSR are dropped. Then n linear combination of the content of the LFSR are calculated and used to generate the encoded sequence. From this coding scheme it is obvious that the n output bits outputs depends on the most recently recent k bits as well as the last $(L-1)k$ bits. The rate of this code is given by

$$R = \frac{k}{n} \tag{3.1}$$

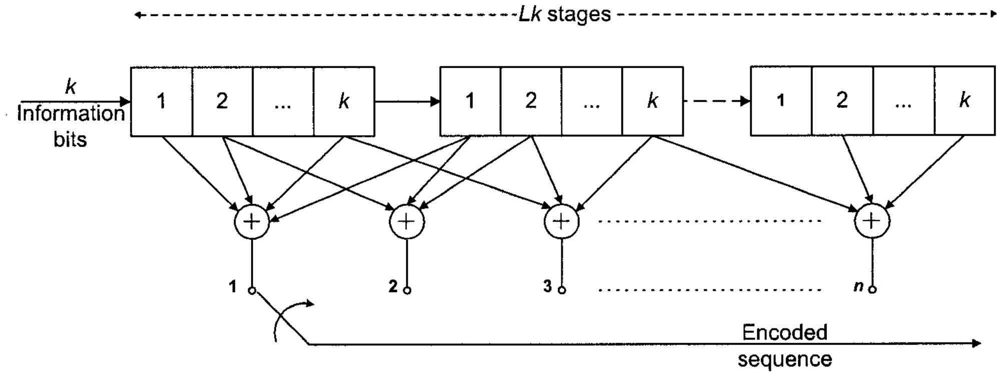

Figure 3.1: An example of a convolutional encoder.

The 1000BASE-T Ethernet coding scheme is based on Trellis Codes Modulation (TCM). TCM adds redundancy by combining convolutional coding and modulation into a single operation. As described previously, the more redundancy that is added to the data, the more error correction can usually be done at the receiver, thus the lower the BER. However in 1000BASE-T Ethernet, where the available bandwidth is limited and has to be used efficiently, the added redundancy can reduce the actual bit rate. Therefore, in this standard just one bit of redundancy is added to each word (8 bits) to preserve the bit rate at 1 Gbps. Figure 3.2 depicts the structure of the encoder for 1000BASE-T, where Sdn[0,...,7] denotes 8 bits of data for transmission. It can be seen in this figure that the rate of the TCM is $\frac{2}{3}$. In the encoder structure, the last two MSBs (Sdn[6] and Sdn[7]) of the transmitted word are used to generate one bit redundant information (*i.e.* Sdn[8]). After encoding, the 3-bit output of the TCM encoder (Sdn[6], Sdn[7] and Sdn[8]) is used to select among 8 possible 4-D symbol subsets (labeled $S0$ to $S7$ in Table 2.1) and the first 6-bit of Sdn (Sdn[0] to Sdn[5]) is used to select the symbol combination in the selected subset [7]. This is done by a mapper (Figure 3.3).

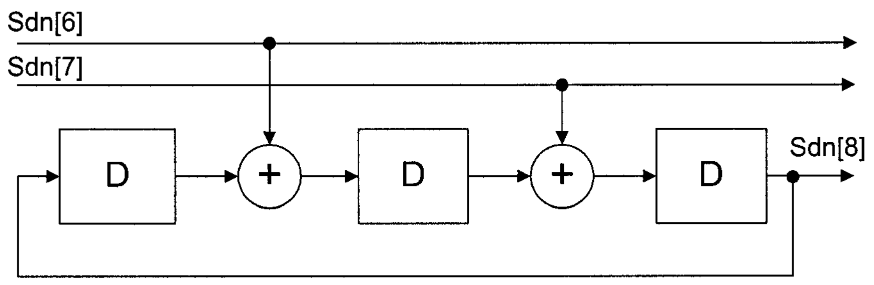

Figure 3.2: Convolutional encoder in the 1000BASE-T TCM.

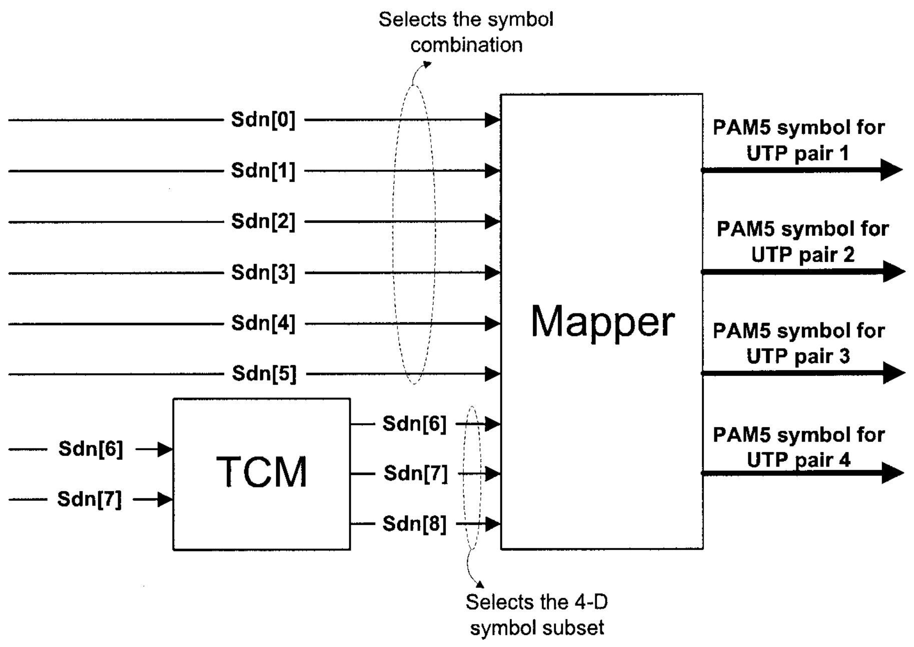
Selects the symbol combination
Sdn[0]
Sdn[1]
Sdn[2]
Sdn[3]
Sdn[4]
Sdn[5]
Sdn[6]
Sdn[7]
TCM
Sdn[6]
Sdn[7]
Sdn[8]
Mapper
PAM5 symbol for UTP pair 1
PAM5 symbol for UTP pair 2
PAM5 symbol for UTP pair 3
PAM5 symbol for UTP pair 4
Selects the 4-D symbol subset

Figure 3.3: The TCM encoder and mapper in the 1000BASE-T.

The preceding encoder can be considered as a FSM with 8 states (since it has 3 bits of memory) in which each state has two inputs (Sdn[6] and Sdn[7]) and 3 outputs (Sdn[6], Sdn[7] and Sdn[8]). A widely used method to represent such an FSM is to use a trellis diagram. A trellis diagram is basically a state transition diagram plotted versus time. In the 1000BASE-T literature, the trellis diagram for the encoder usually shows the combined function of the encoder and the mapper. Figure 3.4 shows the 1000BASE-T trellis diagram. In this diagram, each branch value (*i.e.* output of each state) represents a 4-D symbol. For example, when the encoder state is even (*e.g.* 000), only a 4-D symbol from the four even constellation subsets (*i.e.* $S0$, $S2$, $S4$ and $S6$) can be output by the TCM. Thus, the next state must be selected from $\{000, 001, 010, 011\}$, as determined by the input bits Sdn[6] and Sdn[7].

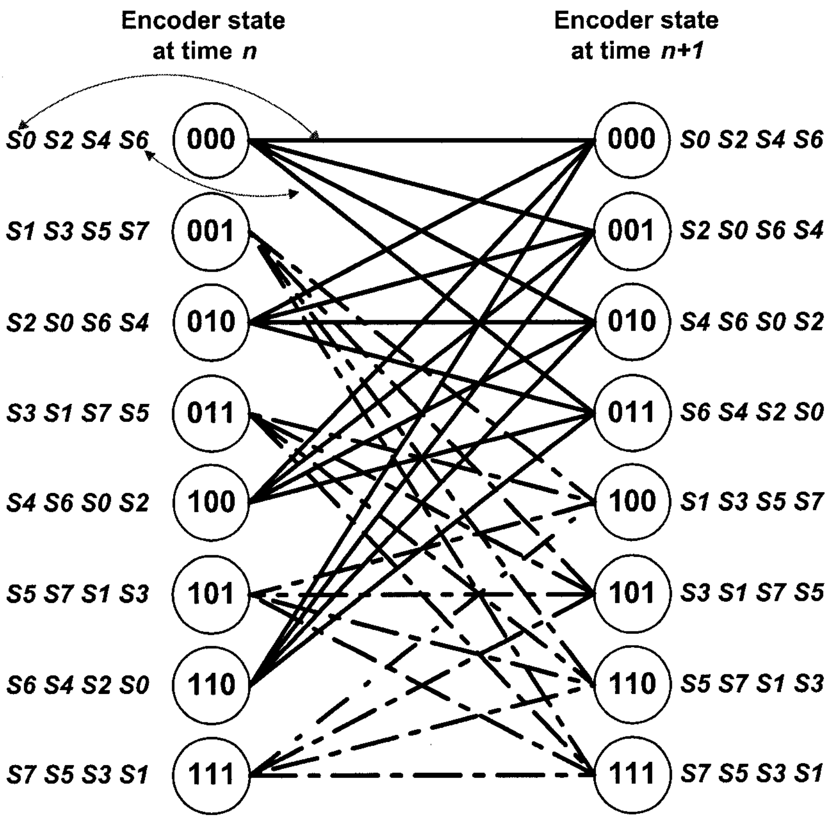

Figure 3.4: Trellis diagram of the 1000BASE-T TCM encoder.

The 1000BASE-T code trellis takes advantage of the subset partitioning, as discussed in Section 2.1.1. Due to subset partitioning and labeling of the transitions in this code trellis, only branches corresponding to even or odd

4-D subsets leave or enter each state. Therefore, the minimum square Euclidian distance between allowed sequence is $\Delta^2 = 4$, which corresponds to an asymptotic maximum coding gain of $10\log(4) = 6$ dB in SNR [2]. However, achieving this coding gain also depends on both decoding and equalization schemes, which are described in the following sections.

3.1.1.2 Decoding of Convolutional Codes

In general, the various decoding schemes can be classified as either soft decision decoding or hard decision decoding. In soft decision TCM decoding, the decoder input sequence is compared with all the possible signal points in the constellation of the coded modulation systems and the one with the closest Euclidian distance is chosen as the decoder output. In hard decision TCM decoding, on the other hand, the input sequence is first mapped to a binary sequence by making binary decisions on its individual components and then the codeword with the closest Hamming distance[1] is chosen [29].

There exist many algorithms for decoding convolutionally encoded data. The Viterbi algorithm (VA) [42] is probably the most widely used decoding method for convolutional codes. VA can be viewed as an efficient way of forming an optimal trellis searching algorithm. VA is an ML decoding algorithm which, upon receiving the channel output, searches through the trellis to find the path that is most likely to have generated the received sequence output. This path is called a "survivor path" and has the minimum distance from the received sequence. As previously mentioned, the distance can be computed as a Euclidian distance in soft decision decoding or as a Hamming distance in hard decision decoding. To do so, VA stores a metric for each state in the trellis. Each state metric represents the minimum distance of the paths leading to that state. VA steps can be described briefly as follows:

- Initialization: Set the metric of the leftmost state of the trellis to 0.

[1]The Hamming distance is defined as the number of positions in two equal-length sequences for which the corresponding elements are different. In another words, it measures the number of bit inversions required to change one into the other. For example, the Hamming distance between 1011101 and 1001001 is 2.

- Computation step $n+1$: Assume that at the previous step (time n) all survivor paths to each state are identified and each state's survivor path is stored. For each state at level $n+1$, the metric of all of the incoming paths is computed as the addition of the metric of the incoming branch and the metric of the survivor path at the time n. Then the path with the smallest metric for each state is chosen as a survivor path for that state. For the trellis in Figure 3.4, four incoming paths for each of the eight possible next encoder states would need to be considered. Only one of these paths would survive for each state going to the next iteration.

- Final step: The computation is iterated until the algorithm reaches the termination node (*i.e.* the last node in the trellis), at which the VA makes a decision on the maximum-likelihood path which is the final survivor path. The decoded sequence is the sequence of bits corresponding to this path's branches.

Figure 3.5(B) illustrates the VA steps on the 4-state trellis of the simple one input TCM encoder shown in Figure 3.5(A). In the trellis diagram, a solid line indicates that the received data is 0 and a dashed line indicates that the received data is 1. It is assumed that the encoded sequence "00 00 00 00 00" is sent over the channel and received as "01 00 01 00 00" (*i.e.* 2 bit errors). As shown in Figure 3.5, the VA is able to properly decode the received sequence as the best survivor path indicates the "00 00 00 00 00" sequence. In general, a Viterbi decoder provides both error detection and error correction. Thus, the overall system performance, expressed in terms of the SNR, is effectively increased by several dB when a TCM and Viterbi decoder is employed in transceivers [10].

In the context of convolutional decoding, the VA has been shown to be the optimal detection scheme for detection data signals distorted by ISI. In 1000BASE-T, VA is an optimal trellis searching algorithm, that simultaneously provides equalization and detection. The major drawback of the VA is

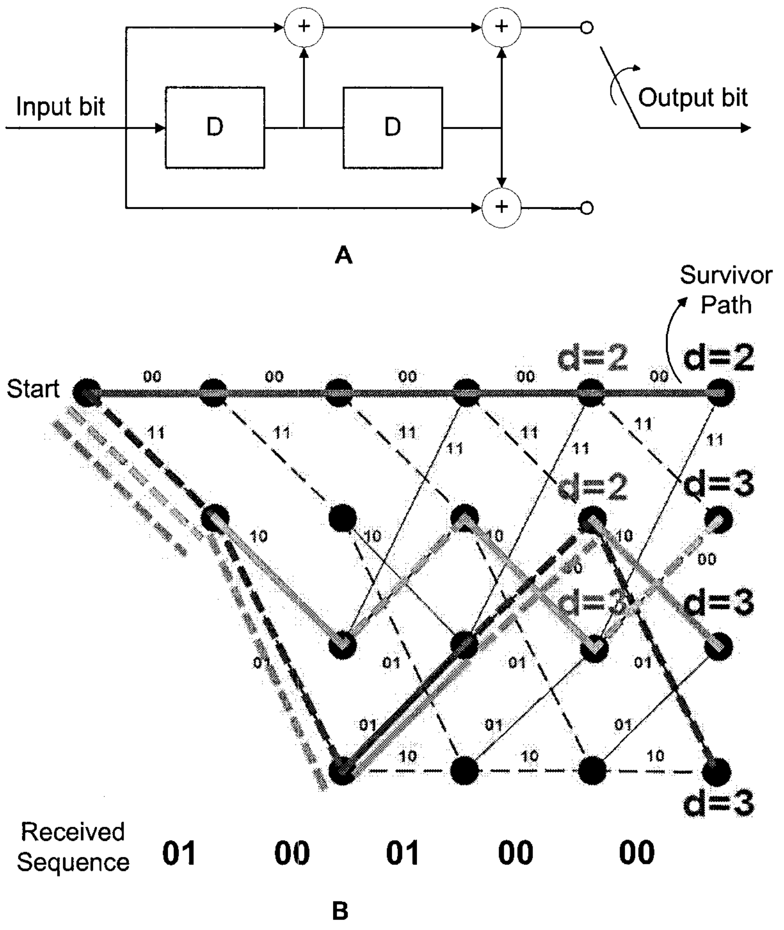

Figure 3.5: A) A simple TCM encoder, B) VA steps on the trellis of the TCM encoder.

its exponential behavior in computational complexity. An optimal implementation of the VA for this purpose would require a large number of states. The total number of states in a trellis for joint optimal decoding and equalization of 4-D PAM-5 is given by

$$S \times 2^{mL} \tag{3.2}$$

where S is number of coding states, m is the number of information bits contained in a 4-D code symbol and L is the length of trellis. In 1000BASE-T, $S = 8$, $m = 8$ and L is 14, which is equal to the expected 1000BASE-T channel length (see Figure 2.7). Therefore, the total number of states for an optimal detection scheme would be 10^{34} [2], which would make the MLSE scheme a prohibitively computationally expensive solution for 1000BASE-T. This problem motivated the search for alternative near-optimal solutions with reduced complexity for 1000BASE-T.

3.1.2 Separate Equalization and Decoding

One possible near-optimal way to reduce the MLSE complexity for 1000BASE-T systems is to use an 8-state Viterbi decoder (Figure 3.4). Here it is assumed that the ISI was separately canceled by a DFE equalizer. Figure 3.6 shows this structure. This scheme is based on pre-equalization performed by four parallel DFE (one for each wire pair) to remove ISI and a Viterbi decoder that runs the VA on the code trellis to decode convolutional coded symbols.

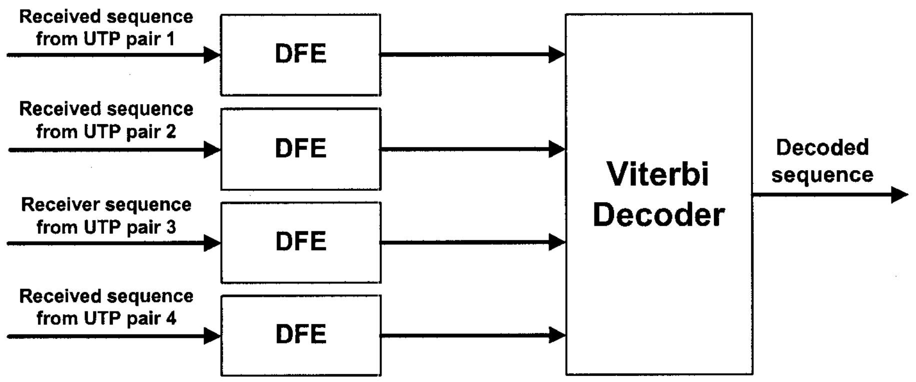
Received sequence from UTP pair 1
Received sequence from UTP pair 2
Receiver sequence from UTP pair 3
Received sequence from UTP pair 4
DFE
DFE
DFE
DFE
Viterbi Decoder
Decoded sequence

Figure 3.6: Separate equalization and decoding for 1000BASE-T.

The advantage of this structure is that it is simple, however, the SNR improvement by exploiting this scheme in 1000BASE-T is not sufficient to meet the target BER of 10^{-10}. According to [12], the SNR improvement is only about 1 dB in comparison with uncoded DFE detection (Figure 2.19).

3.1.3 Delayed Decision Feedback Sequence Estimation

DDFSE [36] is another method to reduce the number of states in VA. The DDFSE algorithm recursively finds an approximation to the MLSE problem. In fact, DDFSE is a detection algorithm that trades complexity off against performance. The complexity of the algorithm is controlled by a parameter,

l, which can be varied from zero up to the length of channel (*i.e.* $0 \leq l \leq L$). When $l = 0$, the DDFSE structure is the same as a DFE and when $l = L$, the complexity of DDFSE is the same as VA. For intermediate values of l, the structure of DDFSE can be described as the combination of a reduced state VA and DFE. In this case, the VA cancels the ISI for the most recent l received samples and DFE cancels ISI from the past inputs greater than l samples (*i.e.* samples $l + 1$ to L) [36].

The steps of the DDFSE algorithm are similar to the steps in the VA described in Section 3.1.1.2. DDFSE is also based on a symbol-to-symbol trellis but with a reduced number of states. As in the VA, it recursively estimates the survivor path in the trellis. However, since each state in the DDFSE trellis provides only partial information about the full state of the channel (*i.e.* when $l \leq L$), the algorithm also stores the best path leading to each state and extracts the feedback information, provided by DFE, from each of these paths to compute the state metric. This implies that the state metric calculation, hence the survivor path selection, also depends on the feedback information provided by DFE.

Figure 3.7 depicts the DDFSE structure in 1000BASE-T system. In the DDFSE approach for 1000BASE-T, an independent feedback signal is computed for each path in the Viterbi decoder as the convolution of the sequence of symbols associated with that path and the coefficients of the feedback filter of the DFE. For an 8-state trellis decoder (see Figure 3.4), there are 8 paths associated with each state (one path per state). This requires the computation of 8 independent feedback signals (the feedforward filter is common). Since the feedback filter coefficients are the same for all states, the only difference is in symbols associated with the 8 different paths. The use of DDFSE implies that instead of one decision in the VA, there are 8 of them (one per path) that must be tracked independently. As it will be shown in the simulation results, by using DDFSE, it is possible to take advantage of coding gain up to 6 dB without using a full-blown Viterbi decoder.

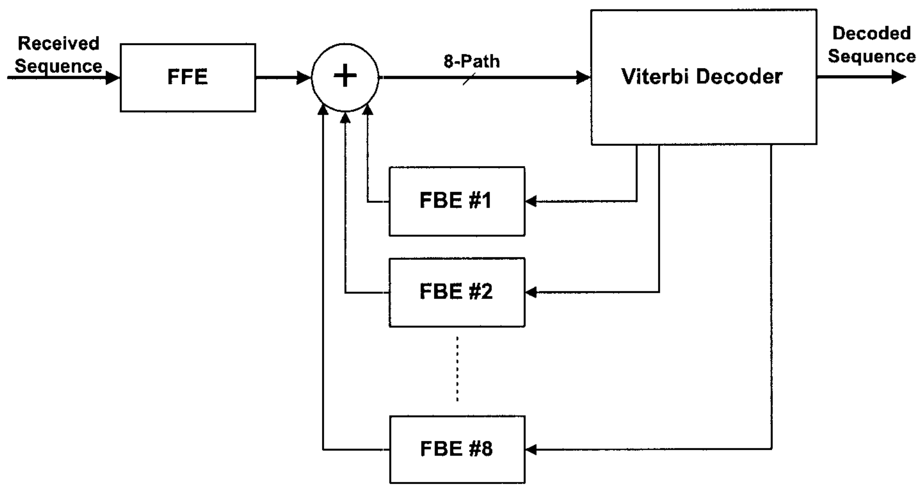

Figure 3.7: DDFSE structure in 1000BASE-T.

3.1.4 BER Performance of DDFSE in 1000BASE-T Systems

Figure 3.8 shows a block diagram of the 1000BASE-T encoding/decoding scheme. This scheme performs the following tasks:

- Generates a random byte stream.
- Performs TCM on each byte, which adds 1 bit of redundancy to each byte of data (a total of 9 bits).
- Maps each 9-bit data word to one of the 4-D PAM-5 symbols.
- The output stream of the transmitter is distorted by channel ISI and then corrupted by AWGN.
- The received sequence is decoded by the DDFSE.

Figure 3.9 shows the results of a BER performance evaluation for a conventional DDFSE scheme using the 1000BASE-T 4-D PAM-5 constellation. For comparison, simulation results of uncoded PAM-5 transmission and detection with DFEs (from Figure 2.19) is also included in this figure. In the simulation study, it was assumed that the channel memory was 14 (CAT-5 UTP channel in Figure 2.7). The SNR in the figure is $10\log_{10}(E_S/E_N)$, where E_S is

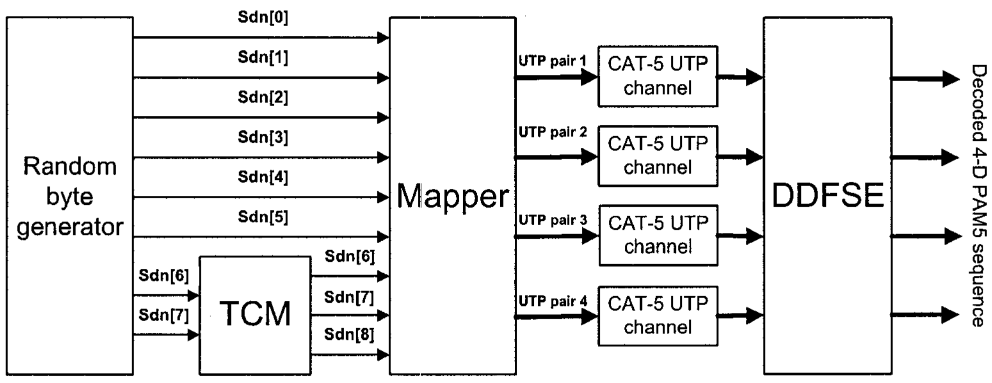

Figure 3.8: Block diagram of the 1000BASE-T encoding/decoding scheme used in the simulation study.

the average symbol energy for a 4-D PAM-5 symbol and E_N is the AWGN variance (noise power). As illustrated, in comparison with the DFE scheme, the DDFSE scheme can achieve a coding gain of about 5.3 dB. Such a coding gain enables the 1000BASE-T system to satisfy the target BER of 10^{-10} at a reasonable SNR[2].

3.2 Alternative Coding Schemes for 10GBASE-T Ethernet

The choice of coding scheme is crucial for 10GBASE-T, in a sense that such a scheme should guarantee a BER performance of 10^{-12} with a reasonable decoding complexity to achieve the required 10 Gbps data transmission rate. Compared to the 1000BASE-T coding scheme, the coding scheme in 10GBASE-T should also have an improved noise performance as well as coding gain in order to be able to operate despite severe impairments that exist in the 10GBASE-T channel. In addition, the coding scheme also needs to be compatible with Ethernet practice. For example, it has to have a reasonable encoding and decoding latency to accommodate the maximum allowable latency of an Ethernet system. Also the coding scheme needs to be compatible with the variable-length frame formats in Ethernet.

[2]Approximately, at 22 dB SNR.

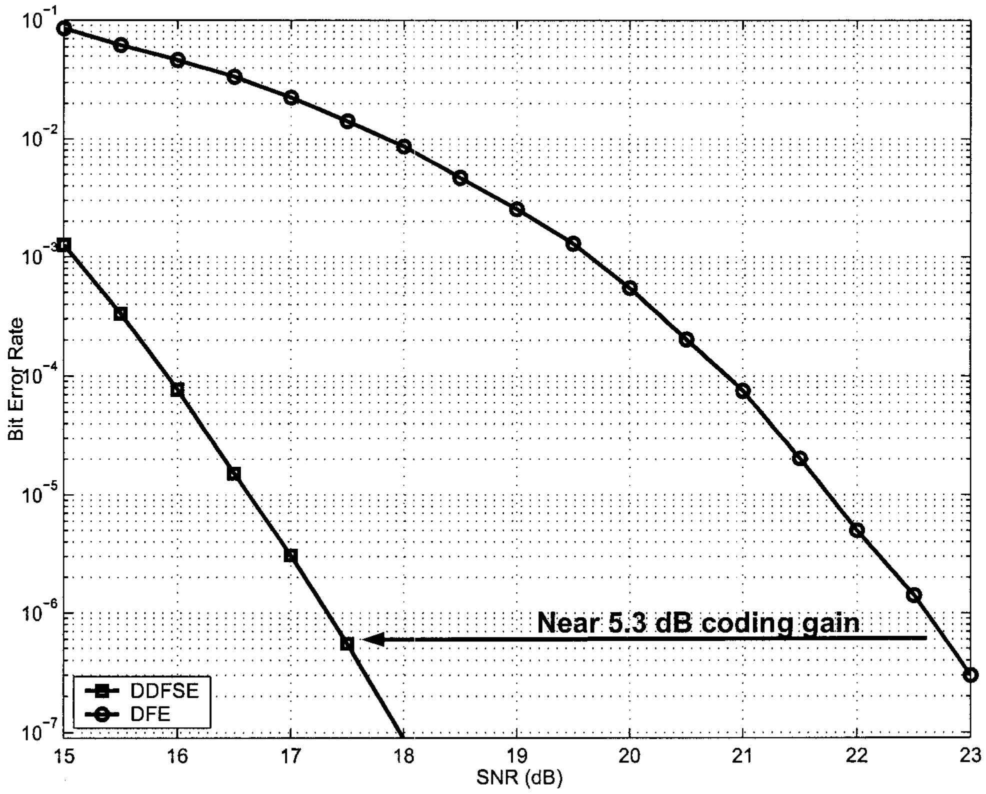

Figure 3.9: BER performance of DDFSE versus DFE.

There have been several coding proposals for 10GBASE-T, including conventional coding schemes such as TCM as well as novel coding schemes such as Low-Density Parity-Check (LDPC) codes. This section reviews and describes the alternative coding schemes for 10GBASE-T.

3.2.1 LDPC Codes

LDPC codes were discovered by Gallager in the early 1960s [43]. This class of codes was recently shown to be capable of error correcting performance close to the Shannon limit [44–47]. For example, it has been shown for a rate 0.5 LDPC code that reliable communication is possible within 0.0045 dB of the Shannon limit for the binary input AWGN channel at BER of 10^{-6} [48]. LDPC codes are decoded with iterative decoding algorithms, such as the sum-product algorithm, with linear decoding complexity. These essential features, along with recent improvements in LDPC code design, have produced coding systems that match or outperform many conventional and modern coding schemes, and, hence, have made LDPC codes as a candidate coding scheme for 10GBASE-T.

LDPC codes are a special class of linear block codes. In block codes, each block of k information bits is mapped into a length n block of output bits by a rule defined by the code. The coding rule ignores all input bits prior to the k most recent information bits. An (n, k) *block code* is a collection of 2^k binary sequences, each of length n, called *codewords.* Such a block code is called *linear*, if the modulo-2 sum of any two codewords is also a codeword.

A rate $\frac{k}{n}$ *binary* LDPC code can be defined as an (n, k) linear code with an $(n-k) \times n$ parity-check matrix H. Let d_v and d_c be the maximum number of 1's in each row and column of H, respectively. Then, the parity-check matrix H usually has the following properties [32]:

i) No two columns have more than two rows with 1's in those two row locations.

ii) Both d_c and d_v are small compared to the dimensions of H.

An example of the parity-check matrix for a (12,3) LDPC code with $d_v = 4$ and $d_c = 3$ is as follows [32]:

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{3.3}$$

Let $x = (x_1, x_2, ..., x_n)$ be a codeword associated with an (n, k) LDPC code with parity-check matrix H. Then for all codewords x it is true that

$$xH^T = 0 \tag{3.4}$$

where the operator T denotes the matrix transposition operation. This relationship introduces $n-k$ linear equations, each containing d_v of x_i's. For example, for the parity-check matrix in Equation 3.3, we have the following set of parity-check equations (E_1 to E_9):

$$\begin{cases} E_1 : x_3 \oplus x_6 \oplus x_7 \oplus x_8 = 0 \\ E_2 : x_1 \oplus x_2 \oplus x_5 \oplus x_{12} = 0 \\ E_3 : x_4 \oplus x_9 \oplus x_{10} \oplus x_{11} = 0 \\ E_4 : x_2 \oplus x_6 \oplus x_7 \oplus x_{10} = 0 \\ E_5 : x_1 \oplus x_3 \oplus x_8 \oplus x_{11} = 0 \\ E_6 : x_4 \oplus x_5 \oplus x_9 \oplus x_{12} = 0 \\ E_7 : x_1 \oplus x_4 \oplus x_5 \oplus x_7 = 0 \\ E_8 : x_6 \oplus x_8 \oplus x_{11} \oplus x_{12} = 0 \\ E_9 : x_2 \oplus x_3 \oplus x_9 \oplus x_{10} = 0 \end{cases} \tag{3.5}$$

where $\oplus$ denotes the XOR operation.

LDPC codes can be represented effectively by a bipartite graph called a *Tanner graph* [49]. Tanner graphs are an effective graphical representation for LDPC codes. They not only provide a complete representation of the code, but they also help to describe the decoding algorithm as explained later on in this chapter. As with all bipartite graphs, the nodes of a Tanner graph are separated into two distinctive sets and each edge connects nodes from the two sets. The two types of nodes in a Tanner graph are called variable nodes and check nodes. Figure 3.10 shows the Tanner graph of the party check matrix in Equation 3.3. It consists of $m = n - k$ check nodes (the number of parity bits) and n variable nodes (the number of bits in a codeword). Check node c_i is connected to variable node v_j if and only if element h_{ij} of H is one. In another words, check node c_i performs the party check equation E_i in Equation 3.5.

A LDPC code is called *regular* if d_c is constant for every column and $d_v = d_c(n/m)$ is also constant for every row. For example, the LDPC code of Equation 3.3 is a regular code with $d_c = 3$ and $d_v = 4$ for all columns and rows. The regularity of this code can also be seen in its graphical representation in Figure 3.10. Note that the variable nodes all have the same degree, that is, the same number of connecting edges. In addition the check nodes all have the same degree. If H is low density but the number of 1's in each row or column

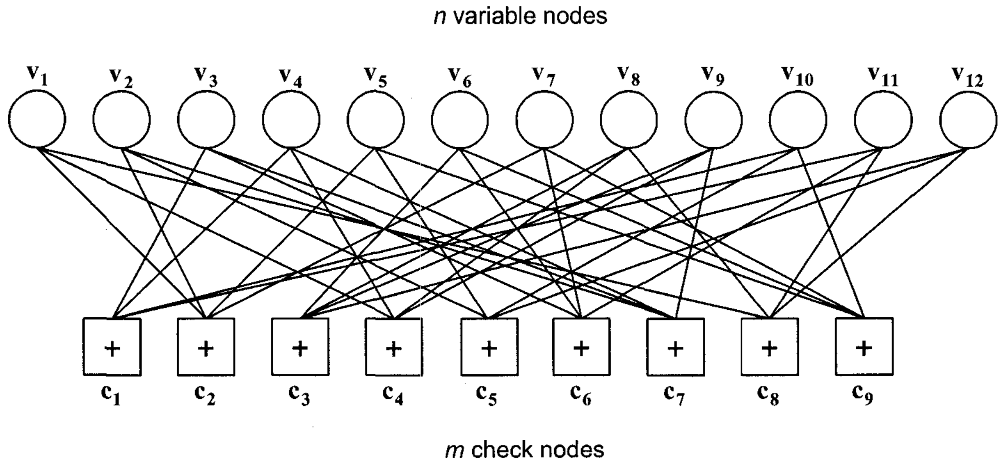

Figure 3.10: Tanner graph of the parity-check matrix in Equation 3.3 .

is not constant, then the code is called an *irregular* LDPC code. Another category for LDPC codes is *full rank* LDPC codes. In a full rank LDPC code, $m = n - k$. When an LDPC code is not full rank, it has redundant rows in its H matrix, thus $m \neq n - k$.

3.2.1.1 LDPC Encoding

Given a parity-check matrix H, we can define a corresponding $k \times n$ generator matrix, G, such that

$$GH^T = 0 \tag{3.6}$$

The generator matrix can be used as an encoder according to

$$x^T = u^T G \tag{3.7}$$

where u is the input sequence of the encoder and x is the encoded sequence.

Equation 3.6 implies a quadric complexity for LDPC encoding with respect to the code length [50], *i.e.* $O(n^2)$. It is worth mentioning that there exist encoding algorithms for LDPC codes with less complexity. Some of these techniques exploit the sparseness of the parity-check matrix for efficient encoding or that impose some structure on the Tanner graph so that encoding is simpler. Repeat-Accumulate [50] codes are an example of codes based on structured graphs. It has been shown that transforming the generator matrix to upper triangular form leads to reduced complexity encoding [50].

3.2.1.2 LDPC Decoding

As mentioned before, a principal advantages of LDPC codes is that they can be decoded by an iterative algorithm with complexity that is linear in the code length, *i.e.* $O(n)$. This important property, in addition to the promising performance of LDPC codes, make them a strong candidate coding scheme for 10GBASE-T. This section briefly reviews two main algorithms for LDPC decoding.

3.2.1.3 Sum Product Algorithm

The *Sum Product Algorithm* (SPA), also known as *Message Passing Algorithm* (MPA), is an iterative algorithm for LDPC decoding. SPA can be done using either hard decision or soft decision decoding. In *hard decision decoding*, each received symbol is thresholded to yield a single received bit as input to the decoding algorithm, and the messages passed between the variable and check nodes each consist of single bits only. In *soft decision decoding*, multiple bits are used to represent each received symbol and the messages passed between the variable and check nodes. Soft decision decoding can achieve substantially better coding performance because the confidence with which each decoder decision is encoded and is forwarded to subsequent decoder iterations [51].

The soft decision form of SPA is called the *Belief Propagation Algorithm* (BPA). It has been shown that BPA can closely approximate the optimal decoder algorithm for AWGN channels, *i.e.* the *Maximum Aposteriori Probability* (MAP) algorithm [50]. The MAP algorithm computes the log-likelihood ratio (LLR) of the received sequence and makes a decision by comparing this LLR to the threshold value. In a PAM-2 modulation[3], the LLR value of the

[3]PAM-2 is equivalent to the Binary Phase Shift Keying (BPSK) modulation.

i-th received signal defined as:

$$\lambda(x(i)) = \log(\frac{P(x(i) = 1|y(i))}{P(x(i) = 0|y(i))}) \tag{3.8}$$

where $x(i)$ and $y(i)$ denote the i-th sample of the transmitted and received sequence, respectively. A positive LLR value (*i.e.* $\lambda(x(i)) > 0$) implies that $P(x(i) = 1|y(i)) > P(x(i) = 0|y(i))$ and therefore, it is more likely that $x(i) = 1$ would have been transmitted. On the other hand, when the LLR value is negative, $P(x(i) = 1|y(i)) < P(x(i) = 0|y(i))$ and it is more likely that $x(i) = 0$ would have been transmitted. A zero LLR value means that $P(x(i) = 1|y(i)) = P(x(i) = 0|y(i))$.

Let PAM-2 be the modulation scheme, $y_i \in R$ the received symbol at variable node i, and $\lambda_i \in R$ the decision at variable node i. A message from variable node i to check node j is represented by $\alpha_{i\to j} \in R$, and a message from check node j to variable node i is represented by $\beta_{i\to j} \in R$. Let $V_{j\backslash i}$ denote the set of variable nodes which connect to check node j, excluding variable node i. Similarly, let $C_{i\backslash j}$ denote the set of check nodes which connect to variable node i, excluding check node j. The decoding algorithm is then as follows [50]:

- Step 1: Initialize $\lambda_i = 2y_i/\sigma^2$ for each variable node.
- Step 2: Variable nodes send $\alpha_{i\to j} = \lambda_i$ to each check nodes $j \in C_i$.
- Step 3: Check nodes connected to variable node i compute and send (see Figure 3.11(A))

$$\beta_{j\to i} = 2\tanh^{-1}(\prod_{l\in V_{i\backslash j}} \tanh(\frac{\lambda_l}{2})) \tag{3.9}$$

- Step 4: Variable nodes connected to check nodes j compute and send (see Figure 3.11(B))

$$\alpha_{i\to j} = \sum_{l\in C_{j\backslash i}} \beta_{l\to i} \tag{3.10}$$

- Step 5: Stop decoding process once a fixed number of iterations has been completed or the estimated codeword, $\hat{x}$, satisfies the $H\hat{x} = 0$ criterion. Otherwise return to Step 3.

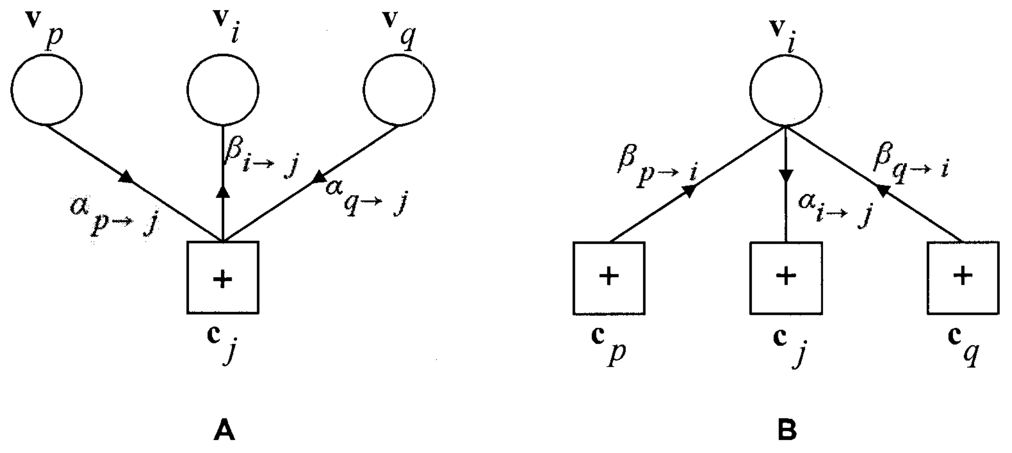

Figure 3.11: A) extrinsic and B) intrinsic messages in SPA.

The above mentioned SPA algorithm process log-likelihood ratios in the probability domain. As Equation 3.9 involves transcendental functions, a hardware implementation of the SPA will often use look-up tables to more rapidly calculate tanh(.) and $\tanh^{-1}(.)$ functions. The SPA can also be done in logarithmic domain. In the this approach, the computation and the subsequent calculation of the extrinsic check message is greatly simplified by operating logarithms of probabilities because multiplications become then additions and divisions become subtractions. For example, this approach has been used for implementation of LDPC decoder in [51].

3.2.1.4 Min-Sum Algorithm

The *Min-Sum Algorithm* (MSA) is a simple approximation to the SPA [52,53]. In MSA, the extrinsic message in Equation 3.9 is approximated as:

$$\beta_{i\to j} \approx |\lambda_{\min}| \, (\prod_{l\in V_{i\backslash j}} \mathrm{sign}(\lambda_l)) \tag{3.11}$$

where λ_{min} is the minimum magnitude input LLR.

The main advantage of min-sum approximation is that it is simple to implement, which makes it more suitable for high data rate applications like 10GBASE-T. However, MSA causes a loss of about 0.5 to 1 dB [54,55] in performance compared to SPA. If all input LLR messages except one are large, the min-sum output is quite accurate as the tanh product would then be dominated by the smallest LLR. However, if all of the input LLRs are relatively small in magnitude, the min-sum approximation overestimates the output LLR, as compared to exact sum-product decoding [56]. This problem has motivated the development of the modified min-sum approximations [57, 58] to recoup some of the performance loss of min-sum approximation.

3.2.2 LDPC Convolutional Codes

Low-Density Parity-Check Convolutional Codes (LDPC-CCs) were first proposed in [59]. LDPC-CCs can be considered as convolutional codes defined by low-density parity-check matrices and decoded in an iterative way [59, 60]. LDPC-CCs have been shown to have a comparable performance in comparison with LDPC block codes [61]. They are similar to LDPC block codes in the way that they generate code words based on parity-check operations. However, LDPC-CCs are similar to convolutional codes since any codeword is generated using both previous information bits and previously generated code-bits.

In comparison with LDPC block codes, there are some advantages associated with LDPC-CCs which make them suited for certain applications like 10GBASE-T. One advantage is that LDPC-CCs are able to encode and decode arbitrary lengths of data without the need to fragment them into fixed-sized blocks [62]. This feature is advantageous for streaming applications, such as streaming video, and also for packet switching applications such as Ethernet. The frame length in Ethernet systems can vary and this makes the integration of LDPC block codes awkward and difficult. However, by using LDPC-CCs, there is no need to fragment randomly-sized packets into fixed-sized packets. Another important advantage of LDPC-CCs is that the parity-check matrix for LDPC-CCs is lower diagonal and this simplifies the encoding process and reduces the encoding latency [62]. Therefore, the encoder structure is simpler for LDPC-CCs than for LDPC block codes. More details about hardware implementation of LDPC-CCs can be found in [62, 63].

3.2.3 Other Coding Schemes

In addition to the above mentioned schemes, there have been other coding proposals for 10GBASE-T. Some of these proposals are based on using convolutional codes for 10GBASE-T. As an example, in [64] two TCM schemes are proposed, that can relax the decoding speed requirement. Also in [3] a joint multiple-input, multiple-output equalization and decoding based on a 4-D TCM and DFE is proposed. Some other proposals are based on other coding system such as Turbo equalization (such as [65]). A discussion about these coding schemes can be found in [50].

Chapter 4

Performance Evaluation of LDPC Codes in the Presence of ISI

As discussed in Chapter 2, ISI and residual ISI are among the most important impairments in 10GBASE-T systems. Although the effect of ISI on the performance of conventional coding systems such as TCM has been well studied [66–69], for LDPC codes, the effects require more investigation.

This chapter presents and discusses the simulation result of BER performance evaluation of LDPC codes in the presence of ISI. The simulation study considers various LDPC codes, including two recent candidate codes for 10GBASE-T Ethernet.

4.1 Codes Used

Table 4.1 lists the six LDPC codes that were considered in the study. In this table, n is the block length of the code, k is the number of information bits, d_v is the maximum variable node degree and d_c is the maximum check node degree. The reason for selecting these particular codes was to isolate, and hopefully more clearly, observe the effects of ISI on BER performance. Also, to study different classes of LDPC codes as well as 10GBASE-T candidate codes. *Code A* and *Code B* in the table are two recent candidate codes for 10GBASE-T Ethernet [8]. These codes are not full rank. They have 129 and

59 redundant check nodes, respectively. The special structure of these codes are a result of the methods in [70]. *Code C* and *Code D* are two regular codes with the same variable degrees and the same check node degrees. They both have relatively high error-floors. The major difference between these two codes is their shortest cycle. The reason for choosing these codes was to see if there is any similarity between the effects of correlation between received symbols caused by ISI, and the effects of cycles on the performance and error-floor of LDPC codes. *Code E* is a relatively low rate regular code and, finally, *Code F* is an irregular code with a relatively high error-floor.

Table 4.1: LDPC codes used in the simulation study.

Code	n	k	R	d_v	d_c	Structure	Characteristics
A	1024	833	0.81	10	32	Regular	4-cycle-free, not full rank
B	2048	1723	0.84	6	32	Regular	4-cycle-free, not full rank
C	1024	512	0.50	3	6	Regular	6-cycle-free
D	1024	512	0.50	3	6	Regular	2-cycle-free
E	816	544	0.33	4	6	Regular	4-cycle-free
F	4000	2000	0.50	7	7	Irregular	4-cycle-free

4.2 System Model

Figure 4.1 illustrates the system model used in the studies. This model is based on an AWGN channel with ISI. It is assumed that the encoded sequence, $x(D)$, is transmitted using PAM-M. The transmitted sequence $x(D)$ is then passed through the channel and corrupted by AWGN to form the received sequence $y(D)$:

$$y(D) = x(D)h(D) + n(D) \tag{4.1}$$

The Channel Impulse Response (CIR) $h(D)$ is assumed to have the following D-transform:

$$h(D) = h_0 + h_1 D + h_2 D^2 + ... + h_{L-1} D^{L-1} \tag{4.2}$$

where h_i is the *i-th* tap of the CIR and L is the channel length. For the sake of simplicity, we refer to the CIR by its ordered sequence of taps, h, as follows:

$$h = \{h_0, h_1, h_2, ..., h_{L-1}\} \tag{4.3}$$

The CIR is used to model post-cursor ISI in the system. Therefore, h_0 was first set to 1 and the remaining taps were considered to be much less than h_0. Then h was also normalized to have a unity gain as follows:

$$h \to h/\left\|h\right\| \tag{4.4}$$

where $\|h\| = \sqrt{\sum_{i=0}^{L-1} h_i^2}$ is the Euclidean norm of h. The LDPC decoder in this model performs standard soft-decision message-passing decoding and uses full *tanh* processing in the parity-check nodes.

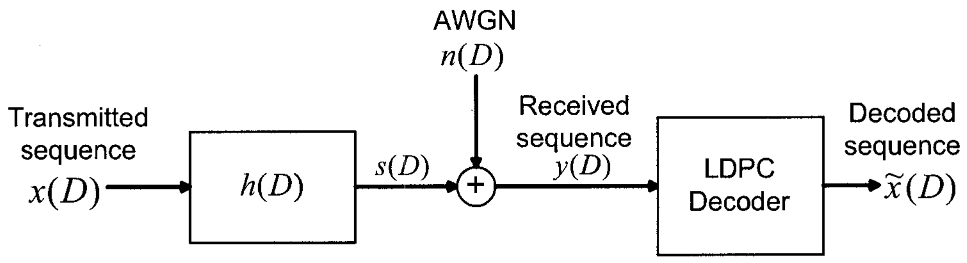

Figure 4.1: System model used for the performance evaluation in the presence of ISI.

4.3 SNIR Scenarios

The standard approach in the literature for evaluating the performance of error correcting codes is through BER measurements for data transmission over AWGN channels. A common way to do this is to measure the code BER performance at different SNRs. Hence, the performance of the systems considered in this paper was also compared to the memoryless (*i.e.* no ISI) AWGN channel. However, as the model in Figure 4.1 shows, the simulated system includes both noise and interference. Therefore, in the study both noise and

ISI power were taken into account, and instead of SNR, the *Signal-to-Noise-and-Interference Ratio* (SNIR) is used. Equation (4.5) gives the definition of the SNIR for the received sequence $y(D)$:

$$\text{SNIR} = \frac{\sigma_y^2}{\sigma_{n_{total}}^2} = \frac{\sigma_x^2 h_0^2}{\sigma_n^2 + \sigma_x^2 \sum_{i=1}^{L-1} h_i^2} \tag{4.5}$$

where σ_x^2 and σ_y^2 are the average power of the transmitted and received signals, respectively. $\sigma_{n_{total}}^2$ is the total variance of the AWGN (per dimension) and interference, while σ_n^2 is the variance of AWGN noise component alone.

In order to have a fair comparison between a memoryless AWGN channel and the channel model in Figure 4.1, the SNIR of the received sequence $y(D)$ was kept the same for both cases:

$$\text{SNIR} = \frac{\sigma_x^2}{\sigma_{n_1}^2} = \frac{\sigma_x^2 h_0^2}{\sigma_{n_2}^2 + \sigma_x^2 \sum_{i=1}^{L-1} h_i^2} \tag{4.6}$$

where $\sigma_{n_1}^2$ is the variance of the AWGN (per dimension) in a memoryless AWGN channel and $\sigma_{n_2}^2$ is the variance of the AWGN in our system model. The right hand side of Equation 4.6 is equivalent to the SNIR for the AWGN channel with ISI, and the left hand side is the equivalent to the SNIR for the AWGN memoryless channel[1]. The $\sigma_{n_1}^2$ in this equation is known according to the average bit energy in the transmitted sequence and the operating SNR in the AWGN memoryless channel (E_b and SNR in Equation 4.7 and Equation 5.4, respectively):

$$E_b = \frac{E_s}{B \times R} \tag{4.7}$$

$$\text{SNR} = 10\log_{10}(\tfrac{E_b}{N_0}) \rightarrow \sigma_{n_1}^2 = \tfrac{N_0}{2} = \tfrac{1}{2}(10^{(-\text{SNR}/10)}E_b) \tag{4.8}$$

where E_s is the average energy per symbol ($E_s = \sigma_x^2$), B is the number of bits per symbol, R is the code rate and N_0 represents the AWGN variance for two dimensions. Therefore, based on Equation 4.6, the $\sigma_{n_2}^2$ can be computed as:

[1]Since there is no interference in the AWGN memoryless channel, the SNIR for an AWGN channel is the same as its SNR.

$$\sigma_{n_2}^2 = \sigma_{n_1}^2 h_0^2 - \sigma_x^2 \sum_{i=1}^{L-1} h_i^2 \tag{4.9}$$

It is important to mention here that since $\sigma_{n_2}^2$ is a positive number, Equation 4.9 imposes a constraint on the maximum amount of ISI. This constraint varies according to the SNR and can be calculated by setting $\sigma_{n_2}^2$ equal to zero in this equation. For instance, given the values of $E_s = 1$, $R = 0.5$, $B = 1$ (*i.e.*, PAM-2 Modulation) and SNR $= 3$ dB in Equation 5.4, the $\sigma_{n_1}^2$ would be equal to 0.5012. By assuming $h = \{1, h_1\}$ as the CIR model, the maximum value for h_1 would be 0.7079.

4.3.1 Error Probability of the Received Signal

In order to have a fair comparison between the proposed ISI model and an AWGN channel, it is useful to consider the error probability of the received sequence $y(D)$ for both cases. The error probability of an AWGN channel can be obtained by using Q-function, which expresses the right-tail cummulative probability for a Gaussian random variable, r, with zero mean and unit variance [14]. The Q-function is defined as the probability that r exceeds a given r_0 value:

$$Q(r_0) = P(r > r_0) = \frac{1}{\sqrt{\pi}} \int_{\frac{r_0}{\sqrt{2}}}^{\infty} e^{-t^2} dt, \qquad r \sim N(0,1) \tag{4.10}$$

When r has a mean value of μ and a variance of σ^2, this cummulative probability can be calculated by translating and scaling the normalized Q-function as follows:

$$P(r > r_0) = Q(\frac{r_0 - \mu}{\sigma}), \qquad r \sim N(\mu, \sigma^2) \tag{4.11}$$

Figure 4.2 shows the Probability Density Function (PDF) of the received sequence in an AWGN memoryless channel with PAM-2 modulation. As illustrated, the received sequence in this figure has Gaussian distributions centered over the two nominal symbol values of 1 and -1. Therefore, by assuming the parameters in Equation 4.6, the error probability of the received sequence in the AWGN channel, EP_1, can be expressed as:

$$EP_1 = P(y > 0|x = -1) + P(y < 0|x = 1) = \frac{1}{2}Q(\frac{0-(-1)}{\sigma_{n_1}}) + \frac{1}{2}(1 - Q(\frac{0-1}{\sigma_{n_1}})) \tag{4.12}$$

where x and y are the transmitted and received sequences, respectively.

Equation 4.12 can be extended for ISI channels. Figure 4.3 shows the resulting PDF of the received sequence for an ISI channel with a CIR model of $\{h_0, h_1\}$ using PAM-2 modulation and at the same fixed SNIR as in Figure 4.2. The ISI in this CIR model causes the received sequence to form Gaussian distributions over four possible values:

$$\mu_1 = -h_0 - h_1, \quad \mu_2 = -h_0 + h_1, \quad \mu_3 = h_0 - h_1, \quad \mu_4 = h_0 + h_1 \tag{4.13}$$

The error probability of the received sequence for this case, EP_2, is[2]:

$$EP_2 = \frac{1}{4}(Q(\frac{0-\mu_1}{\sigma_{n_2}}) + Q(\frac{0-\mu_2}{\sigma_{n_2}}) + (1 - Q(\frac{0-\mu_3}{\sigma_{n_2}})) + (1 - Q(\frac{0-\mu_4}{\sigma_{n_2}}))) \tag{4.14}$$

Equations 4.12 and 4.14 can be easily extended for other PAM and ISI schemes using similar criteria. As an example, for PAM-4 modulation the received sequence forms Gaussian distributions over four values in AWGN memoryless channel and over 16 values in the CIR model with $\{h_0, h_1\}$.

Tables 4.2, 4.3 and 4.4 give the EP_1 and EP_2 error probabilities for $R = 0.33$, $R = 0.5$ and $R = 0.81$, respectively[3], for PAM-2 modulation and $h = \{1, h_1\}$. The minus sign,"-", in the tables indicates that h_1 has reached its maximum possible value in association with Equation 4.9. From these tables, it can be seen that at higher SNRs and at high values of h_1, $EP_2 < EP_1$. This means that at these values, the ISI model will introduce fewer errors in the receiver than a purely AWGN channel. Therefore, to have a fair judgement about the BER performance of LDPC codes in the proposed ISI model, the BER evaluation should be made at the SNRs and h_1 values in which $EP_2 \geq EP_1$.

[2] σ_{n_2} in this equation is calculated according to Equation 4.9.
[3] See Equations 4.7 to 4.9.

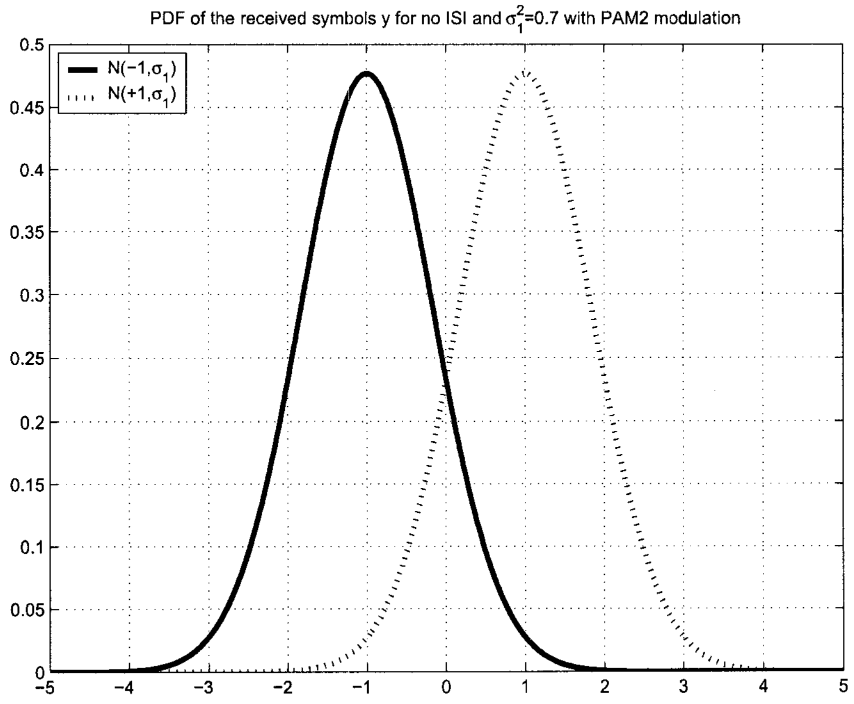

Figure 4.2: PDF for an AWGN in a memoryless channel.

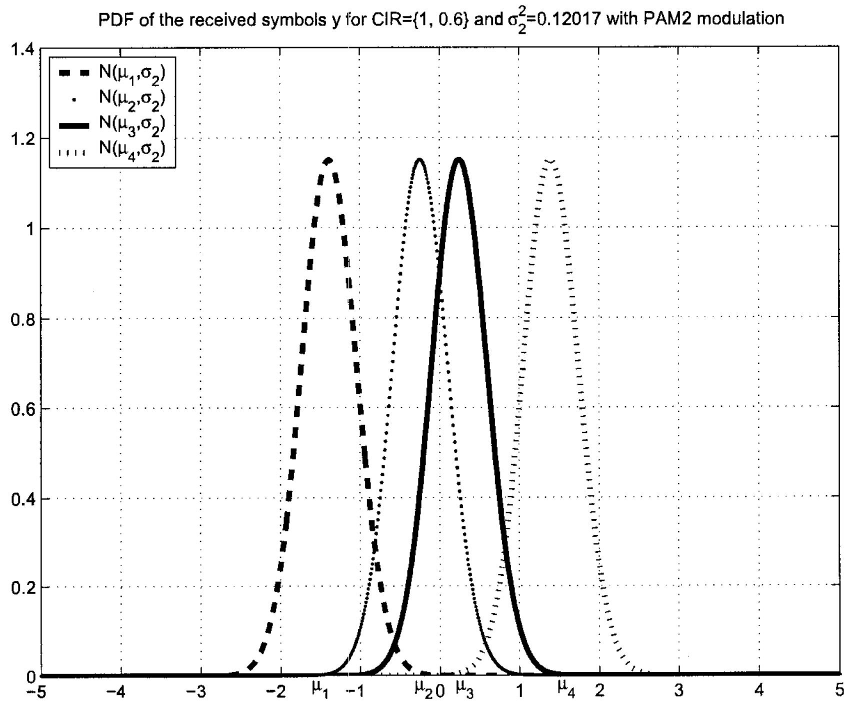

Figure 4.3: PDF for an ISI channel with $h = \{1, 0.6\}$.

Table 4.2: Error probability for AWGN and ISI channels, $R = 0.33$.

SNIR	EP_1	EP_2							
(dB)	$h_1 = 0$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1	0.1798	0.1798	0.1799	0.1801	0.1807	0.1821	0.1851	0.1906	0.2008
2	0.1520	0.1520	0.1521	0.1524	0.1533	0.1553	0.1595	0.1672	0.1798
3	0.1244	0.1244	0.1245	0.1249	0.1260	0.1286	0.1333	0.1396	0.1374
4	0.0978	0.0978	0.0979	0.0984	0.0996	0.1018	0.1031	0.0899	-
5	0.0733	0.0733	0.0734	0.0737	0.0743	0.0732	0.0592	-	-
6	0.0516	0.0516	0.0517	0.0516	0.0500	0.0401	0.0005	-	-
7	0.0338	0.0338	0.0337	0.0326	0.0270	0.0061	-	-	-

Table 4.3: Error probability for AWGN and ISI channels, $R = 0.5$.

SNIR	EP_1	EP_2							
(dB)	$h_1 = 0$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1	0.1309	0.1309	0.1310	0.1314	0.1325	0.1350	0.1398	0.1472	0.1527
2	0.1040	0.1040	0.1041	0.1046	0.1058	0.1082	0.1111	0.1061	-
3	0.0789	0.0789	0.0790	0.0794	0.0802	0.0803	0.0718	0.0011	-
4	0.0565	0.0565	0.0566	0.0566	0.0557	0.0485	0.0101	-	-
5	0.0377	0.0377	0.0376	0.0368	0.0324	0.0130	-	-	-
6	0.0230	0.0230	0.0227	0.0206	0.0117	-	-	-	-
7	0.0126	0.0126	0.0120	0.0086	0.0006	-	-	-	-

Table 4.4: Error probability for AWGN and ISI channels, $R = 0.81$.

SNIR	EP_1	EP_2						
(dB)	$h_1 = 0$	0.1	0.2	0.3	0.4	0.5	0.6	0.7
1	0.0766	0.0766	0.0767	0.0771	0.0778	0.0775	0.0670	-
2	0.0545	0.0545	0.0546	0.0546	0.0535	0.0452	0.0050	-
3	0.0361	0.0361	0.0360	0.0351	0.0302	0.0100	-	-
4	0.0218	0.0218	0.0215	0.0193	0.0101	-	-	-
5	0.0118	0.0118	0.0112	0.0077	0.0003	-	-	-
6	0.0055	0.0055	0.0047	0.0015	-	-	-	-
7	0.0022	0.0021	0.0014	0.0000	-	-	-	-

4.4 Simulation Results

Figures 4.4, 4.5 and 4.6 show the effects of ISI on the BER of *Code A* and *Code C*. Figure 4.4 shows the performance of *Code A* at an SNIR of 3 dB and Figures 4.5 and 4.6 show the performance of *Code C* for PAM-2 and PAM-4 at SNIRs of 2 dB and 5.75 dB, respectively. The BER in each simulation was calculated based on at least 100 frame errors by the 50-*th* decoding iteration. The dashed line in each figure refers to the BER for the AWGN memoryless channel with the same SNIR. The CIR model in this experiment was $\{1, h_1\}$ and the x-axis in these figures represents the amount of h_1 before gain normalization.

The simulation results indicate improved performance for LDPC codes in the presence of ISI and AWGN, compared to purely AWGN in a memoryless channel at the same fixed SNIR. In another words, by decreasing the amount of AWGN and introducing ISI to the system to preserve the same fixed SNIR, the LDPC decoder will perform better. Therefore, in this scheme the LDPC decoder appears to be more sensitive to AWGN than to the modeled ISI. As shown in the figures, for a small amount of ISI the BER performance is very close to the performance of an AWGN channel. However, by introducing more ISI (and decreasing the AWGN in the system) the performance gap becomes more significant. As h_1 reaches its maximum possible value, the performance gap between these two cases becomes very significant. This arises because it is at this point that $\sigma_{n_2}^2$ becomes very small and EP_2 becomes a lot smaller than EP_1. This can be seen in Table 4.2 to 4.4 for large values of h_1.

Figures 4.7 to 4.12 show the effects of ISI for *Code A* to *Code F*, respectively, in a so-called waterfall plot over a wide range of SNIRs for $h = \{1, h_1\}$. As shown in these figures, the BER performance of all codes for an ISI channel is upper-bounded by their BER performance in the AWGN memoryless channel. As mentioned in Section 4.3.1, a fair comparison between the given ISI model and AWGN channel should be made in the SNIRs and h_1 values in which $EP_2 \geq EP_1$. As an example, in Table 4.2 at SNIR=3 dB, EP_1 is 0.1244 for AWGN channel and EP_2 is 0.1249 for a $\{1, h_1 = 0.3\}$ ISI channel

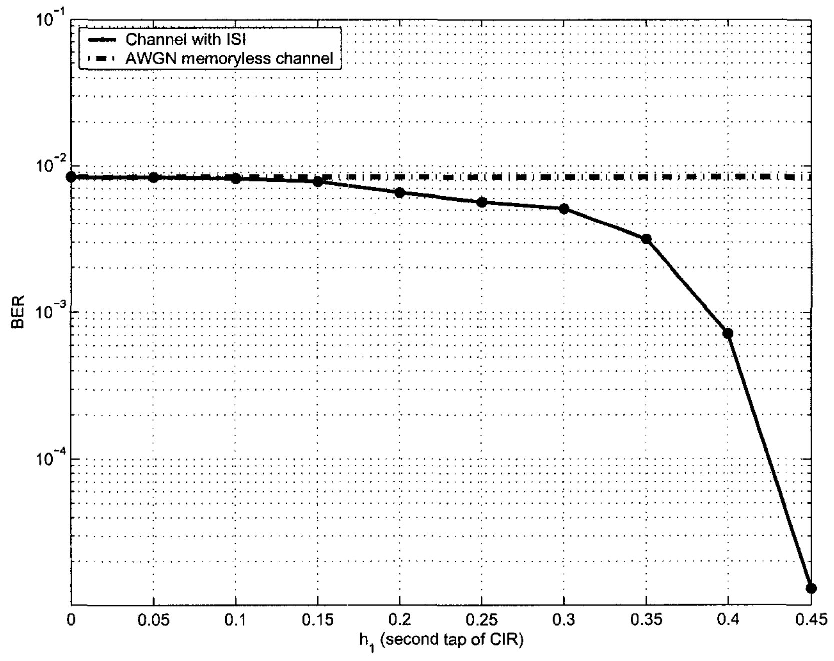

Figure 4.4: Effect of ISI on *Code A*, PAM-2 and SNIR = 3 dB.

model. However, Figure 4.11 shows that the BER of *Code E* at this SNIR is less (about a factor of 9) for the ISI channel than AWGN channel. Therefore, it can be concluded that the BER performance of *Code E* is better for this ISI model compared to that of the purely AWGN channel. Similar comparisons and conclusions can be made for the other LDPC codes used in this study.

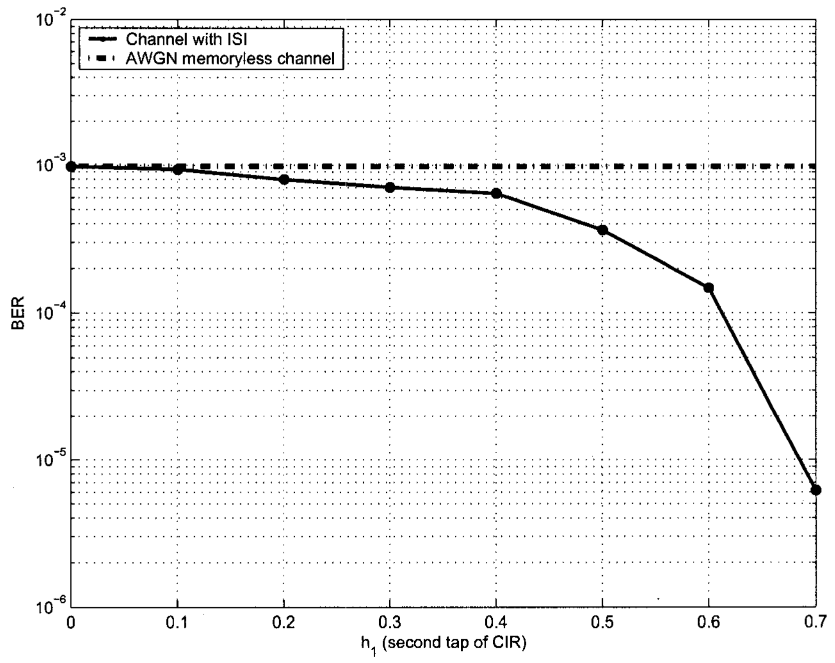

Figure 4.5: Effect of ISI on *Code C*, PAM-2 and SNIR = 2 dB .

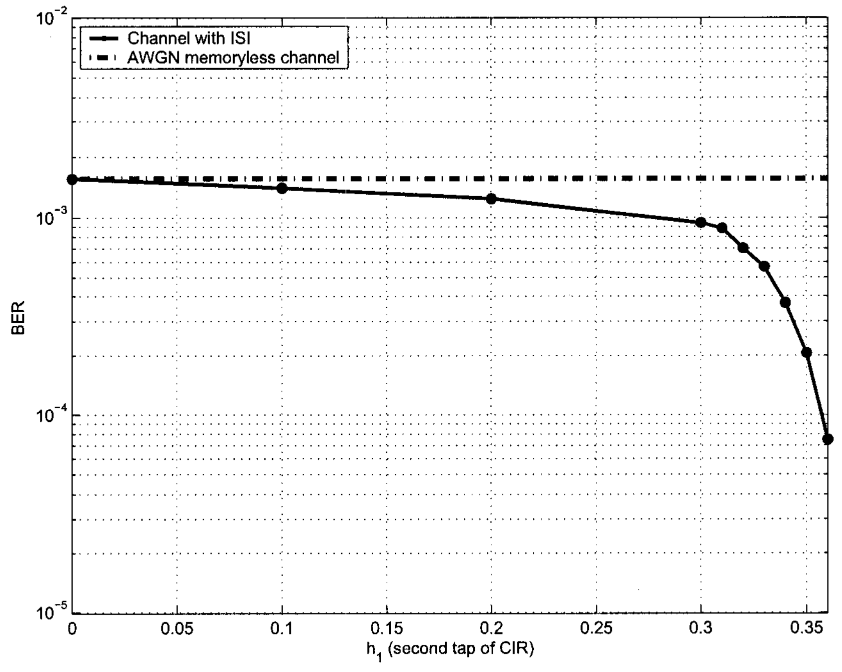

Figure 4.6: Effect of ISI on *Code C*, PAM-4 and SNIR = 5.75 dB.

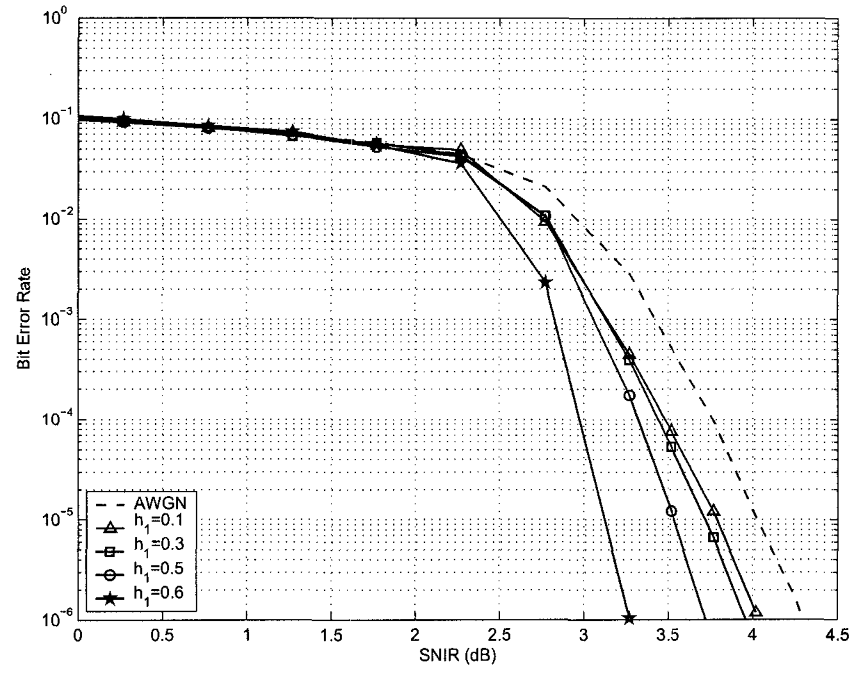

Figure 4.7: Effect of ISI on *Code A*.

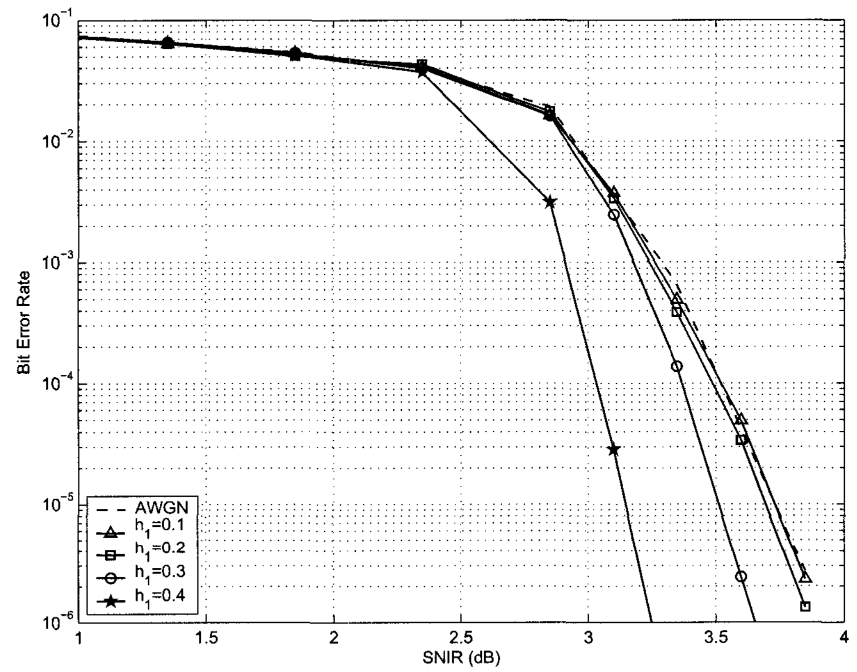

Figure 4.8: Effect of ISI on *Code B*.

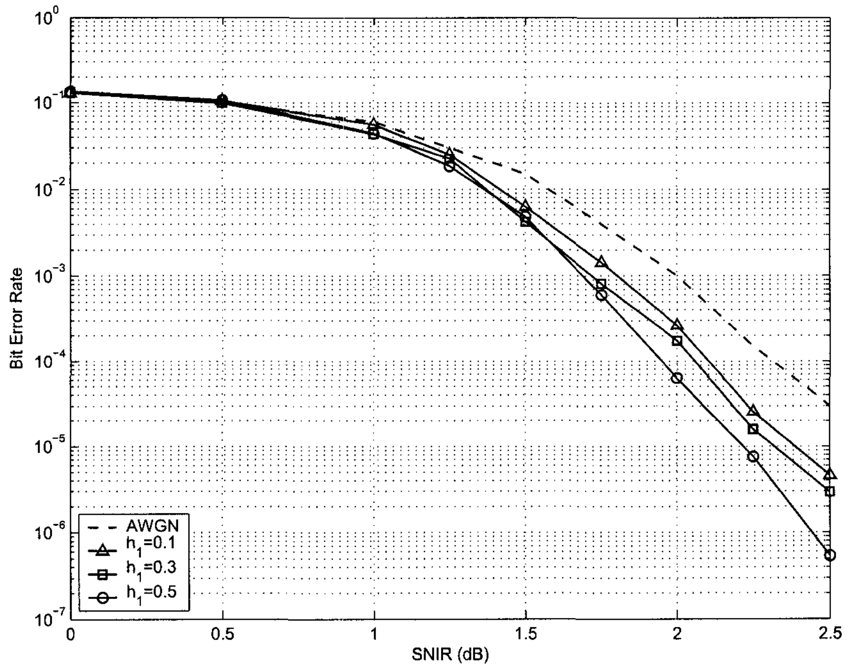

Figure 4.9: Effect of ISI on *Code C*.

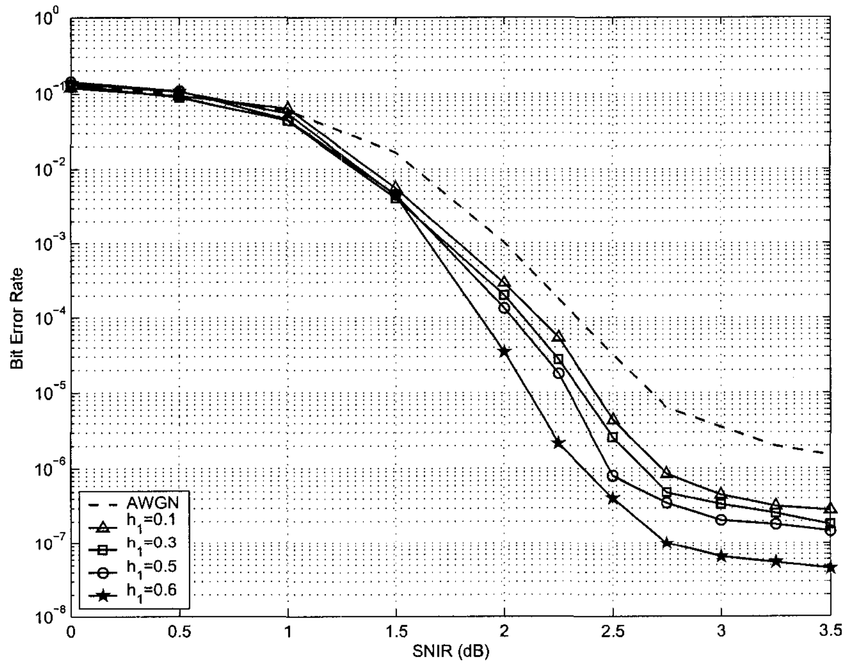

Figure 4.10: Effect of ISI on *Code D*.

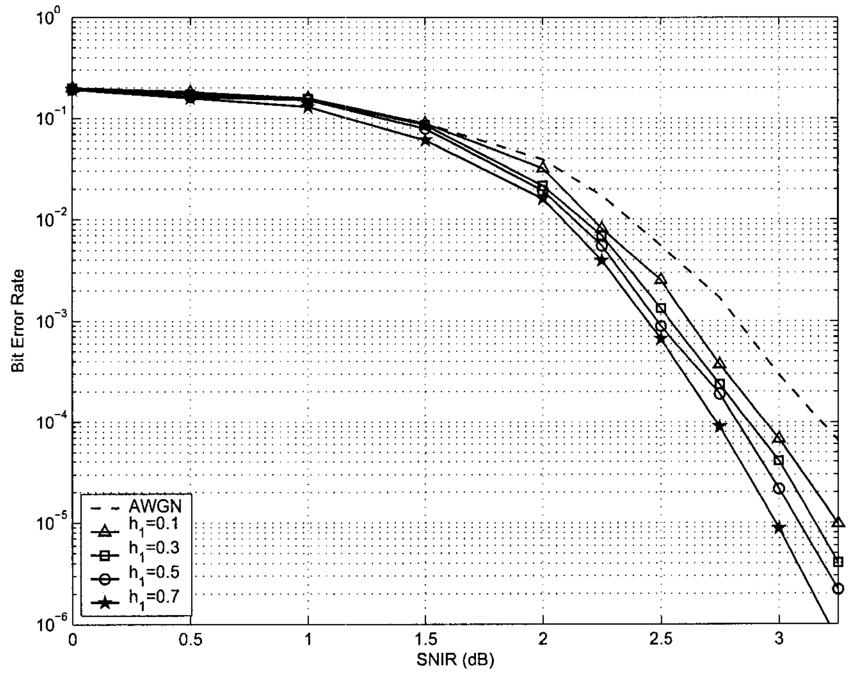

Figure 4.11: Effect of ISI on *Code E*.

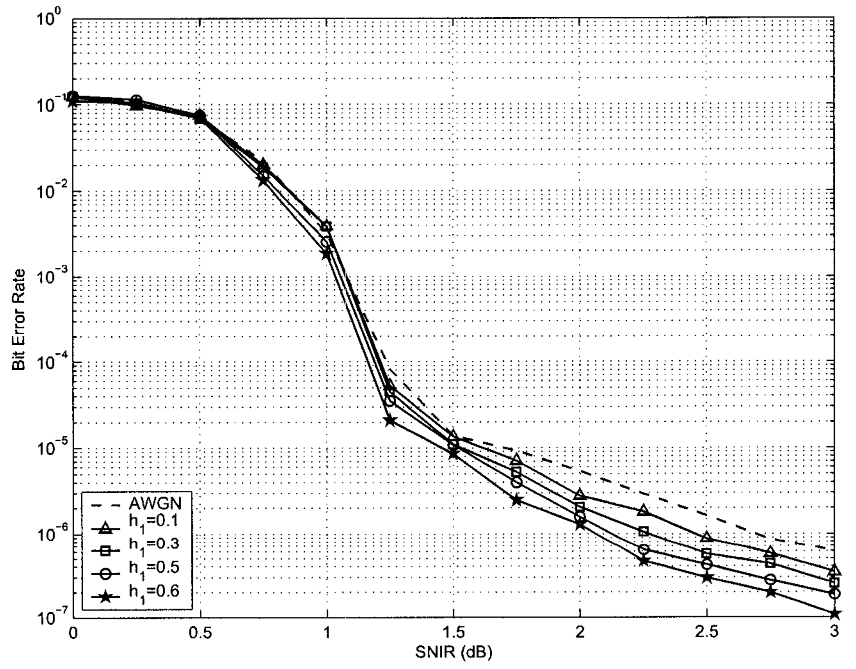

Figure 4.12: Effect of ISI on *Code F*.

4.4.1 Increasing the ISI Taps

The CIR model in the previous section had two coefficients. As mentioned previously, Equation 4.9 imposes a constraint on the maximum amount of ISI. This implies that in the proposed SNIR scenario, increasing the response length of CIR results in smaller CIR coefficients. As an instance, by assuming $L = 5$, equal CIR coefficients and the parameters of the example given in Section 4.3, the maximum CIR tap is lowered from 0.7079 to 0.3762. However, it is worth mentioning that as the CIR length L, increases, the BER performance gap between an ISI channel and an AWGN memoryless channel becomes smaller. When L becomes very large (*e.g.* more than 200), the gap is almost zero. This phenomenon is due to the fact that in the proposed SNIR scenario, a very long CIR, which must have very small coefficients, acts increasingly like an AWGN channel. This is in accordance with the Central Limit Theorem, since by assuming $h_0 = 1$ the amount of ISI added to the transmitted symbol at time t is obtained by:

$$\sum_{i=1}^{L-1} x(t-i)h_i \tag{4.15}$$

Therefore, when L is large, the ISI value can be considered as a random variable with Gaussian distribution. This effect is shown in Figure 4.13. In this figure, the BER performance of *Code D* for $h = \{1, 0.3\}$ is compared to its performance in CIR with $L = 100$ and exponentially decreasing coefficients. Both CIRs introduce the same amount of ISI energy to the transmitted sequence. As shown in the figure, the code performance in long CIR is much closer to its performance in AWGN.

4.4.2 Effect of ISI Pattern and Code Structure

It should be mentioned that in addition to the results presented in this chapter, I completed other simulation studies to study the effects of various ISI patterns and code structures on the performance of LDPC codes in the presence of ISI. However, the preliminary results did not show any change in the BER performance of the codes. For example, the performance of *Code A* was compared

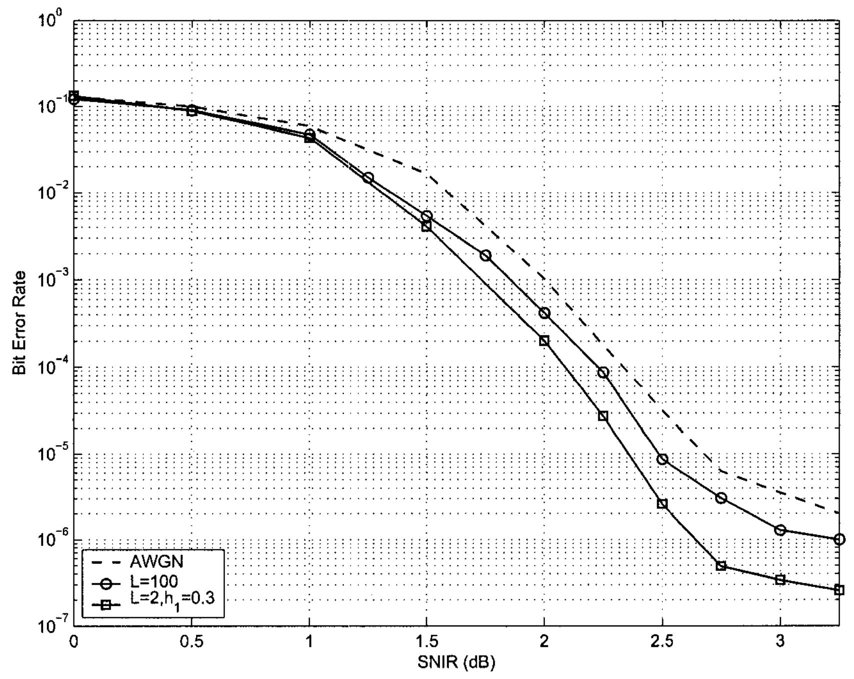

Figure 4.13: Effect of long ISI on *Code D*.

to other versions of this code with randomly permuted variable nodes[4] to see if the correlation introduced by ISI would have different effects on these codes. In another study, due to the special structure of the 10GBASE-T candidate codes, the position of the strongest ISI taps were changed in the CIR to introduce correlation between the inputs of variable nodes which are connected to the same check node. Neither of these studies showed a difference in the performance of the codes in the presence of ISI.

4.5 Conclusion

In this chapter, we reviewed performance evaluation results for the recent LDPC code candidates for 10GBASE-T Ethernet and for standard benchmark LDPC codes over AWGN and ISI channels. It was shown that at the same level of SNIR, LDPC codes are more sensitive to the AWGN than to ISI and the performance of LDPC codes is upper-bounded by their performance in the

[4]This was done by randomly exchanging the columns in its H matrix.

an AWGN channel.

An important application for the proposed results is the interpretation of the performance of LDPC codes in the channels in which the levels of AWGN and residual ISI are known or can be approximated. In another words, since the LDPC code is more sensitive to AWGN than to ISI, it is possible to characterize the AWGN equivalent for given amounts of residual ISI[5] in a channel and then estimate the worst case BER performance of the LDPC codes. This application can be used for systems, like 10GBASE-T, Ethernet in which a target BER must be guaranteed.

[5]As mentioned in Chapter 2, the amount of residual ISI depends on such parameters as the equalizer length and the finite precision of the equalizer coefficients and channel estimation error.

Chapter 5

Performance Evaluation of LDPC Codes in the Presence of Colored Noise

The previous chapter compared the performance of LDPC codes over AWGN and ISI channels. In an ISI channel, the received symbols are correlated but it is usually assumed that the noise model is AWGN. This chapter evaluates the effects of correlation between noise samples (*i.e.* colored noise) on the performance of LDPC codes while the channel does not introduce ISI. Two colored noise models are considered in this chapter: high-frequency Additive Colored Gaussian Noise (ACGN) and low-frequency ACGN. The effects of these noise models are discussed in Sections 5.1 and 5.2, respectively.

5.1 Performance Evaluation in the Presence of High-Frequency ACGN

This section compares the performance of LDPC codes over AWGN and high-frequency ACGN channels. The ACGN model used in this section is generated based on an Infinite Impulse Response (IIR) coloration filter that emphasizes high-frequencies. This filter model is described in Section 5.1.2. The LDPC codes used for this study are the same as the codes presented in Section 4.1.

5.1.1 System Model

Figure 5.1 illustrates the system model. In this model, the transmitted signal, $x(D)$, is transmitted using PAM-2 modulation. The transmitted signal is then corrupted by noise to form the received signal. In the case of ACGN, the noise is generated by passing an AWGN sequence through a coloration filter. The received signal in this model can be expressed by its D-transform as:

$$y(D) = x(D) + n_W(D)f_C(D) = x(D) + n_C(D) \tag{5.1}$$

where $f_C(D)$ is the D-transform of the impulse response of the coloration filter, $n_W(D)$ is the AWGN sequence, $n_C(D)$ is the corresponding ACGN sequence and $y(D)$ is the received signal. The LDPC decoder in this model performs standard soft-decision message-passing decoding and uses full *tanh* processing in the parity-check nodes.

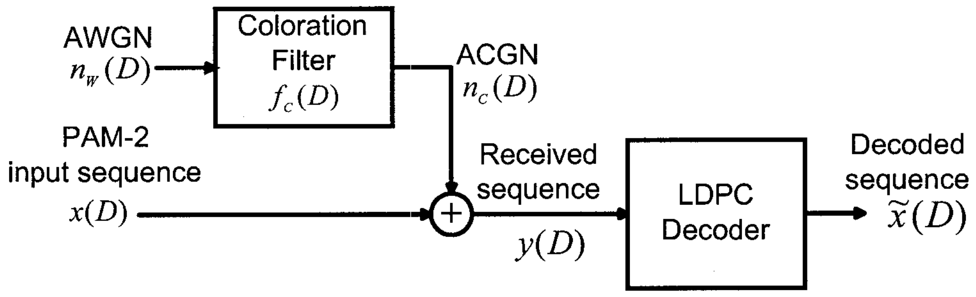

Figure 5.1: System model used for the performance evaluation in the presence of ACGN.

5.1.2 Colored Noise Generation

To generate the high-frequency colored noise, AWGN is filtered by a 1st-order high-pass Infinite Impulse Response (IIR) filter with a Z-transform of:

$$F_C(z) = \frac{a}{1 + bz^{-1}} \qquad 0 \leq b < 1 \tag{5.2}$$

where a is a gain factor and b is a correlation coefficient. This filter correlates each AWGN sample with the preceding noise samples. When $b = 0$ the filter does not introduce any correlation and the noise is strictly AWGN. By increasing b, the correlation between noise samples increases. This correlation noise can be expressed for the k-th ACGN sample as:

$$n_C(k) = a \sum_{i=0}^{k-1} (-b)^i n_W(k-i) \qquad k = 1, 2, ... \tag{5.3}$$

To fairly compare the performance of LDPC codes with respect to high-frequency ACGN and AWGN, the ACGN should have the same noise power as the AWGN. Hence the variance of the noise should be same for both cases. Consequently, a and b in Equation 5.2 must be determined so as not to introduce any gain in the channel model. The noise variance in the system can be obtained according to the SNR as:

$$\sigma^2_{n_W} = \sigma^2_{n_C} = \frac{N_0}{2} = \frac{1}{2}(10^{(-\text{SNR}/10)} E_b) \tag{5.4}$$

where E_b is the average energy per bit, N_0 represents the AWGN variance for two dimensions, and $\sigma^2_{n_W}$ and $\sigma^2_{n_C}$ are the variances of the AWGN and ACGN, respectively. According to Parsaval's relation for the Fourier transform [71], the gain of $F_C(z)$, G, can be computed as:

$$G = \frac{1}{2\pi j} \oint_S F_C(v) F_C(v^{-1}) v^{-1} d_v \tag{5.5}$$

where S is a closed contour in the region of convergence of $F_C(z)$. To have a unity gain filter, G in Equation 5.5 was set to 1 and the filter coefficients were calculated as follows:

$$G = \frac{a^2}{1-b^2} = 1 \rightarrow a = \pm\sqrt{1-b^2} \rightarrow F_C(z) = \frac{\sqrt{1-b^2}}{1+bz^{-1}} \tag{5.6}$$

Figure 5.2 shows the frequency response of $F_C(z)$ for different values of b. As illustrated, by increasing the b coefficient, the filter becomes increasingly high-pass. Thus as b is increased, the colored noise resulting from filtering AWGN becomes increasingly "blue".

5.1.3 Simulation Results

Figures 5.3 to 5.10 show the BER performance of the codes used in the presence of high-frequency ACGN. The BER was calculated based on at least 100 frame

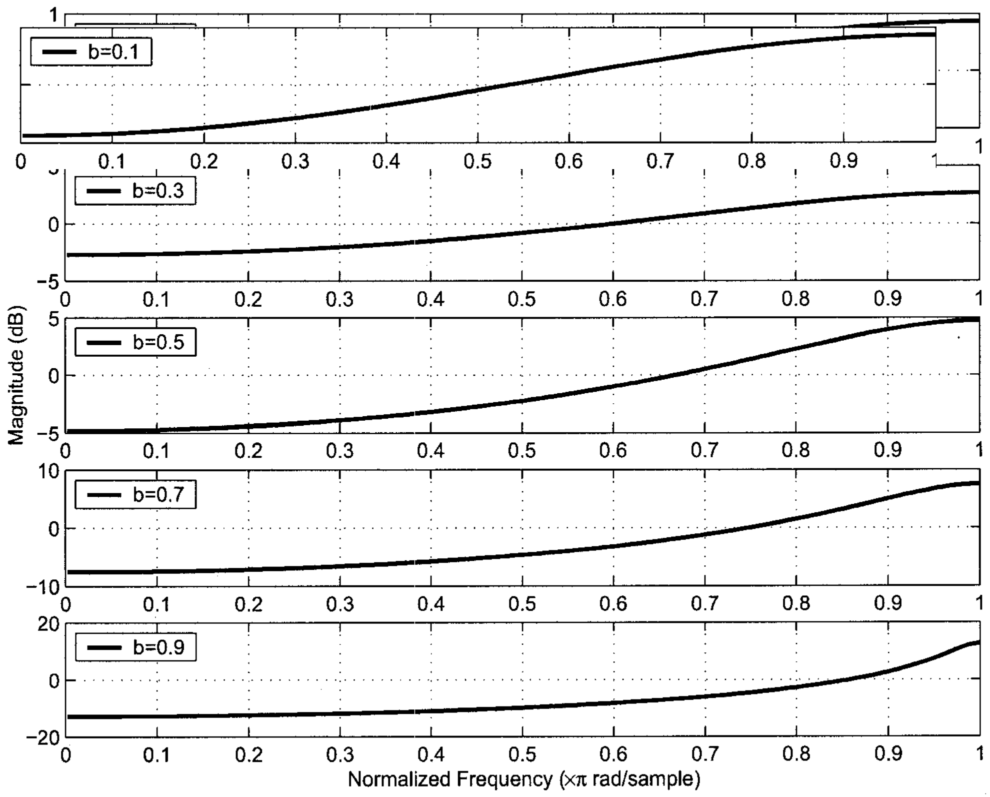

Figure 5.2: Frequency response of high-pass $F_C(z)$ for different values of coefficient b.

errors immediately after the 50-th decoding iteration. The dashed line in each figure shows the BER for an AWGN memoryless channel (*i.e.*, $b = 0$ and $a = 1$). Figures 5.3 and 5.4 compare the BER performance for *Code A* and *Code C* with respect to colored noise versus AWGN at fixed SNRs of 3.5 dB and 2 dB, respectively. The x-axis in these figures corresponds to the b coefficient in $F_C(z)$. These figures show that in comparison with the BER performance over an AWGN channel, more highly blue ACGN more severely degrades the performance of the LDPC decoder. As b increases and the correlation in the ACGN samples increases, the relative performance loss increases. It should also be noted here that the BER degradation for small values of b is approximately log-linear in b. However, for higher values of b the degradation is more rapid than linear. This can be easily seen by comparing the BERs at points $b = 0.5$ and $b = 0.9$ in Figure 5.4.

Figures 5.5 to 5.10 show the same effect for *Code A* to *F*, respectively, in their BER "waterfall" plots over a wide range of SNRs. As these figures show, the SNR performance loss[1], for a certain value of b, for all of the codes is comparable. For example, the performance loss at $b = 0.4$ is about 0.25 dB for all of the codes. Figure 5.5 and 5.6 confirm the above mentioned effects for the 10GBASE-T candidate codes, *Code A* and *B*.

5.1.3.1 Effect of Colored Noise on Error Floor

One of the important properties of LDPC codes is their error floor. An error floor is an undesirable lower limit on the BER that dominates at high SNRs. It is known that the existence of cycles in the structure of an LDPC code can affect its error floor. A cycle correlates the messages in the code Tanner graph and LDPC codes with cycles tend to have a higher error floor. As an example, *Code C* has an error floor that shows up approximately at BER $= 10^{-7}$ and *Code D* has an error floor approximately at BER $= 10^{-5}$ in an AWGN channel. The only difference between these two codes is the length of their shortest cycle[2]. The reason for investigating these two codes for this simulation study was to see if there is any similarity between the effect of correlation introduced by cycles and the effect of colored noise on the BER performance and error floor of short LDPC codes.

Figure 5.7 shows the effect of high-frequency ACGN on the error floor of *Code C*. This figure indicates that, with respect to BER, the high-frequency ACGN raises the error floor. This rise for the worst case coloration is about a factor of 10. The same effect can be seen in Figure 5.8 for *Code D*. By comparing the performance of *Code C* and *Code D*, it can be seen that the short-cycle difference (6-cycle-free to 2-cycle-free) results in superior performance as well as a lower error floor for *Code C* in the presence of ACGN. This is similar to the result for the case of AWGN. Therefore, it can be concluded

[1]The SNR performance loss refers to the amount of SNR loss at a certain BER, whereas the BER performance loss refers to the amount of BER loss at a certain SNR

[2]As mentioned in Section 4.1, *Code C* is a 6-cycle-free code and *Code D* is 2-cycle-free.

that this characteristic of LDPC codes remains the same even in the presence of high-frequency ACGN. Finally, Figure 5.10 shows the effect of ACGN for *Code F*, which is an irregular code with a relatively high error floor. The rise in BER in the error floor region of this code is comparable to the rise observed for the the regular codes that were considered. These results show that the BER curve for high-frequency ACGN channel can be considered as a shifted version of the BER curve for AWGN channel. This means that the LDPC codes have worse BER performance in the presence of high-frequency ACGN, but the overall BER trend (in both the waterfall and error floor regions) over high-frequency ACGN and AWGN channels is qualitatively similar.

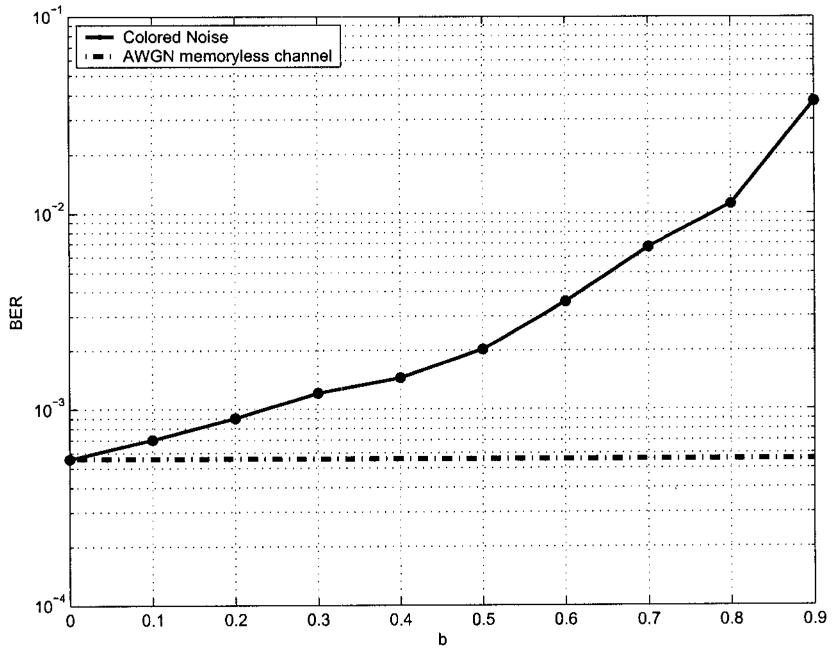

Figure 5.3: Effect of high-frequency ACGN on *Code A* at SNR = 3.5 dB.

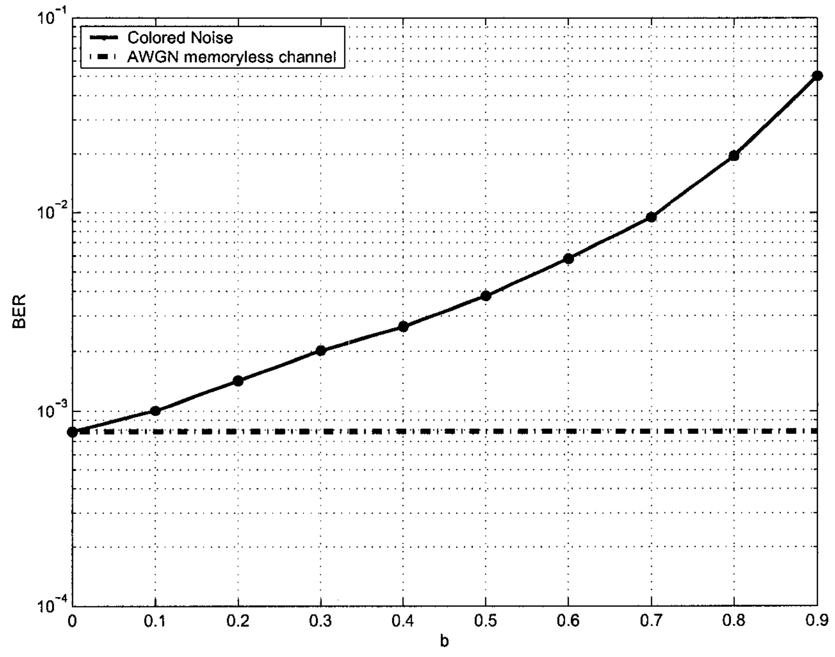

Figure 5.4: Effect of high-frequency ACGN on *Code C* at SNR = 2 dB.

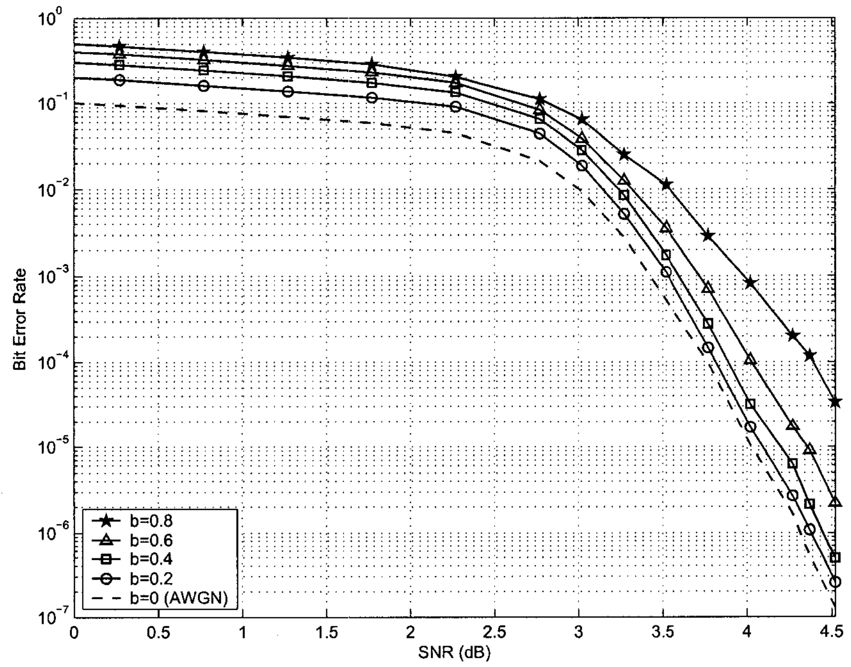

Figure 5.5: Effect of high-frequency ACGN on *Code A*.

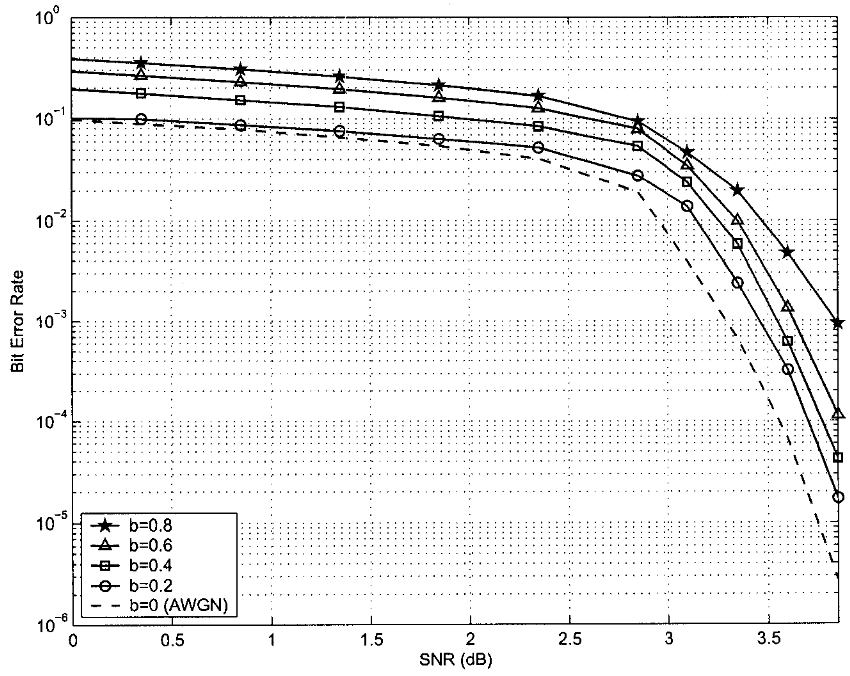

Figure 5.6: Effect of high-frequency ACGN on *Code B*.

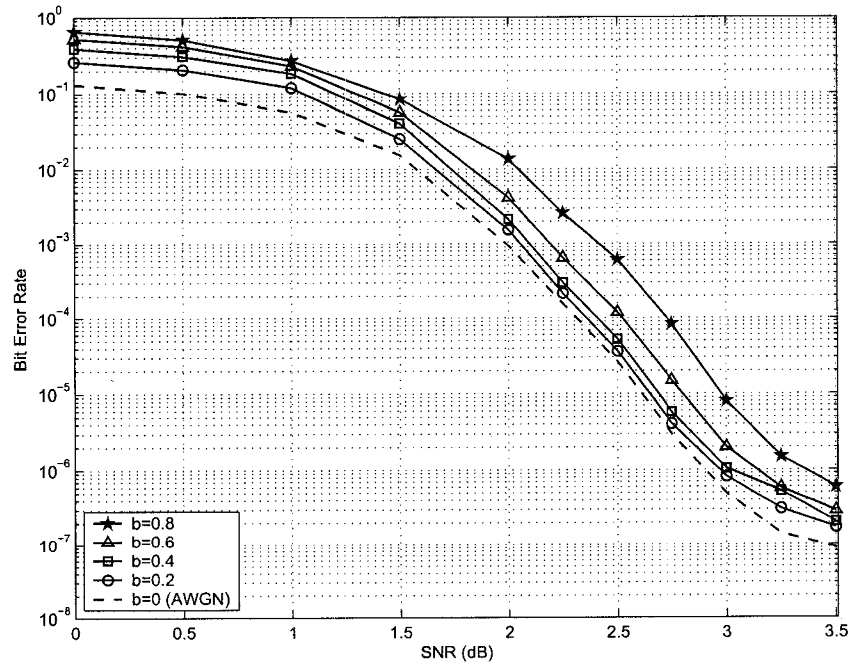

Figure 5.7: Effect of high-frequency ACGN on *Code C*.

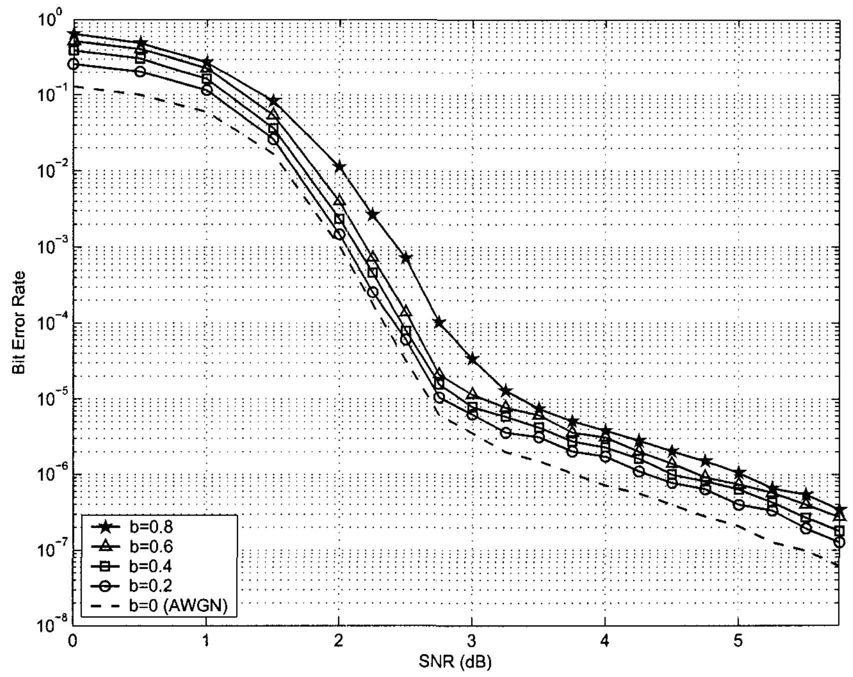

Figure 5.8: Effect of high-frequency ACGN on *Code D.*

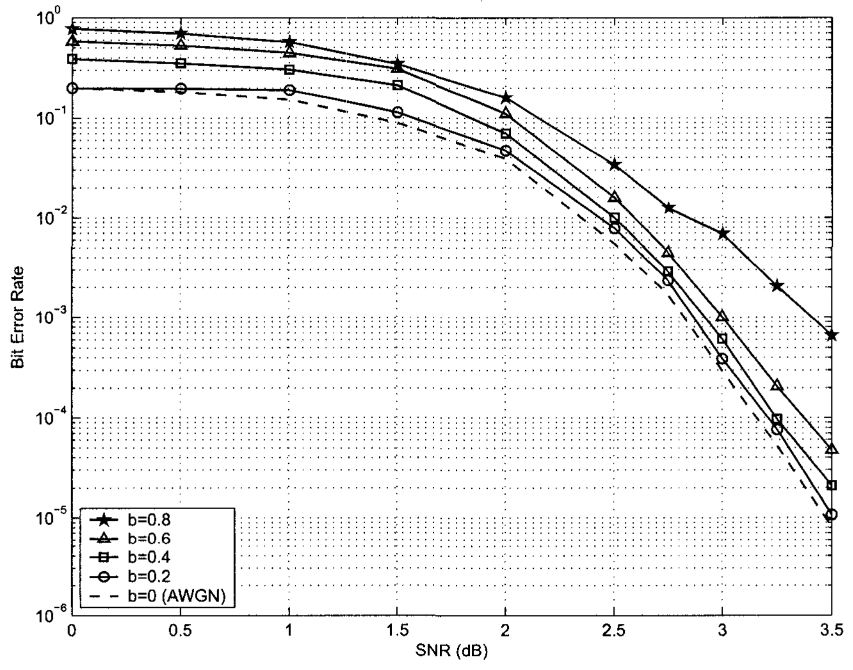

Figure 5.9: Effect of high-frequency ACGN on *Code E.*

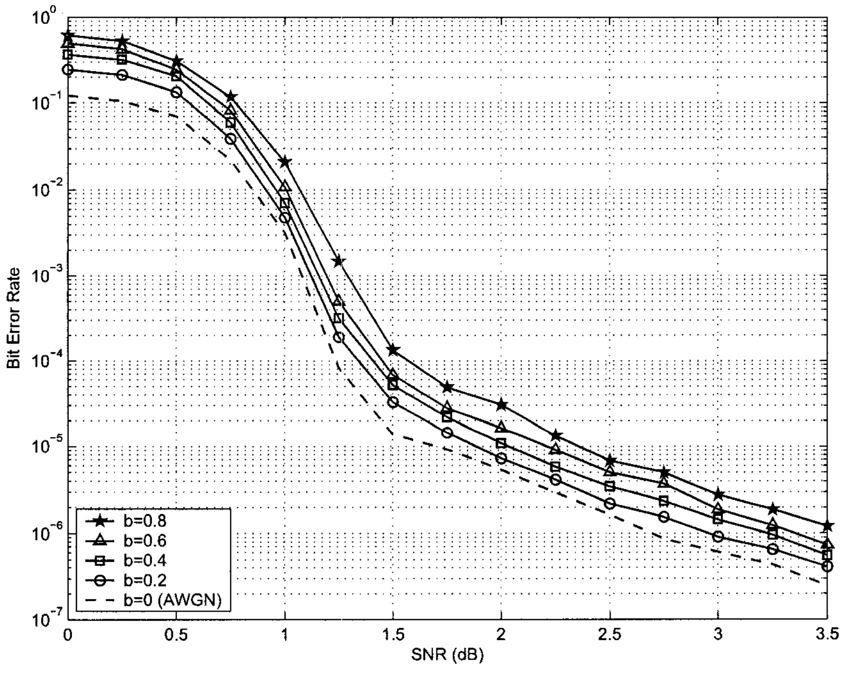

Figure 5.10: Effect of high-frequency ACGN on *Code F*.

5.1.3.2 Effects of ACGN on PAM Signal Constellations with More Signal Levels

The constellation used in the previous section was PAM-2. However, as mentioned in Chapter 2, the candidate constellations for 10GBASE-T are actually PAM-8, PAM-12 and PAM-16. Therefore, a brief study of the effects of ACGN on PAM constellations with more signal levels was also carried out. The purpose of this study was to see if code performance in the presence of ACGN, while using a PAM constellation with more signal levels, is comparable to their behavior with PAM-2.

Figure 5.11 shows the effects of high-frequency ACGN on *Code C* at fixed SNR = 6 dB using a PAM-4 constellation. By comparing the BER performance shown in this figure with that shown in Figure 5.4, it can be seen that the high-frequency ACGN causes comparable BER loss for both constellations.

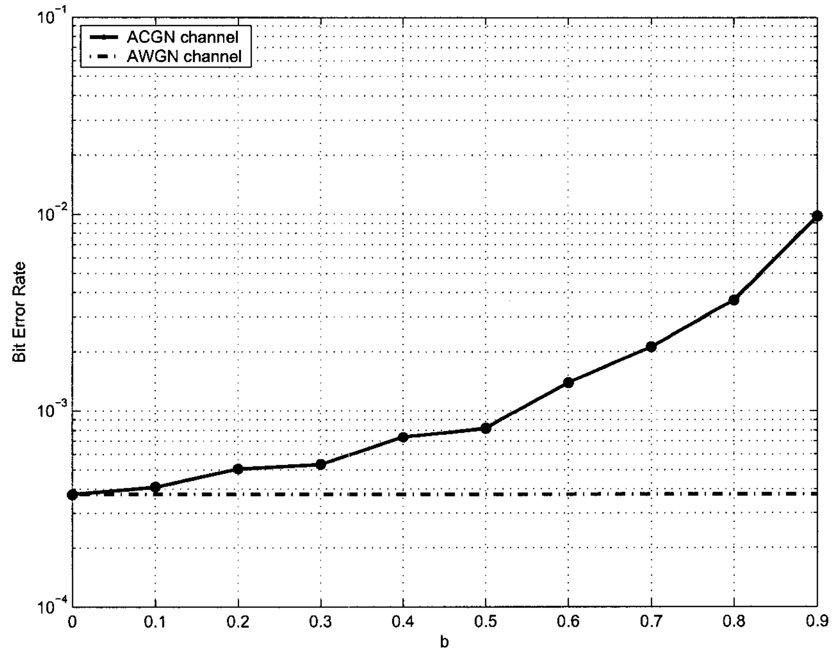

Figure 5.11: Effect of high-frequency ACGN on *Code C* at SNR = 6 dB, PAM-4.

5.2 Performance Evaluation in the Presence of Low-Frequency ACGN

The results in the previous section showed the effect of high-frequency ACGN on the BER performance of LDPC codes. In this section, the effects of low-frequency ACGN are investigated. The system model used in the study is same as the system model in Section 5.1.1 except for the coloration filter. The coloration filter in this study is an IIR filter that emphasizes the low-frequencies. Thus this filter produces "reddish" as opposed to the "blueish" noise studied earlier.

In order to generate a low-frequency ACGN, it is sufficient to negate the value of b coefficient in Equation 5.2 and make the coloration filter low-pass. Therefore, the low-pass coloration filter model can be expressed as follows

$$F_C(z) = \frac{a}{1 + bz^{-1}} \qquad -1 < b \leq 0 \tag{5.7}$$

The gain factor, a, can be also calculated according to Equation 5.6.

Figure 5.12 shows the frequency response of the low-pass $F_C(z)$ for different values of b. As illustrated, by increasing the b coefficient, the filter becomes increasingly low-pass, *i.e.* increasingly red.

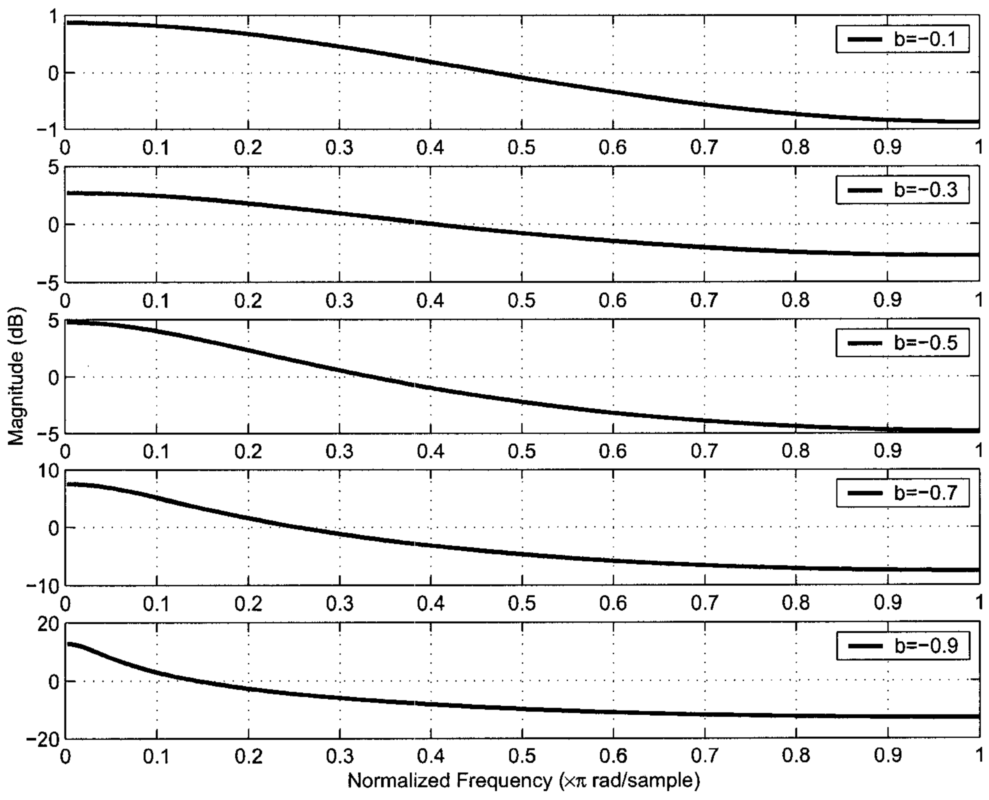

Figure 5.12: Frequency response of low-pass $F_C(z)$ for different values of coefficient b.

5.2.1 Simulation Results

Figures 5.13 to 5.18 show the BER performance of *Code A* to *Code F*, respectively, in the presence of low-frequency ACGN. These figures show that as $|b|$ increases and the ACGN becomes more colored, the relative performance loss increases. This behavior is similar to the effect of high-frequency the ACGN on LDPC codes. However, in comparison with high-frequency ACGN, the BER performance loss is more severe for low-frequency ACGN. This is more apparent for more colored low-frequency ACGN models (*i.e.*, $|b| = 0.6$ or $|b| = 0.8$).

As an instance, the BER loss for 10GBASE-T candidate codes, Figures 5.13 and 5.14, for $|b| = 0.6$ at SNR = 4 dB is about a factor of 20. This is almost twice the BER loss observed for the high-frequency ACGN scheme in Figures 5.5 and 5.6 with $b = 0.8$.

The SNR performance loss is also more degraded in the presence of low-frequency ACGN. The results show that the SNR performance loss for $|b| = 0.4$ can be up to 0.4 dB. This was about 0.25 dB for high-frequency ACGN with $b = 0.4$. As $|b|$ increases, the performance gap between low-frequency ACGN and high-frequency ACGN becomes more dominant. This is shown in Figures 5.19 and 5.20, where BER performance of *Code C* and *Code F* in the presence of low-frequency ACGN are compared to their performance in the presence of corresponding high-frequency ACGN.

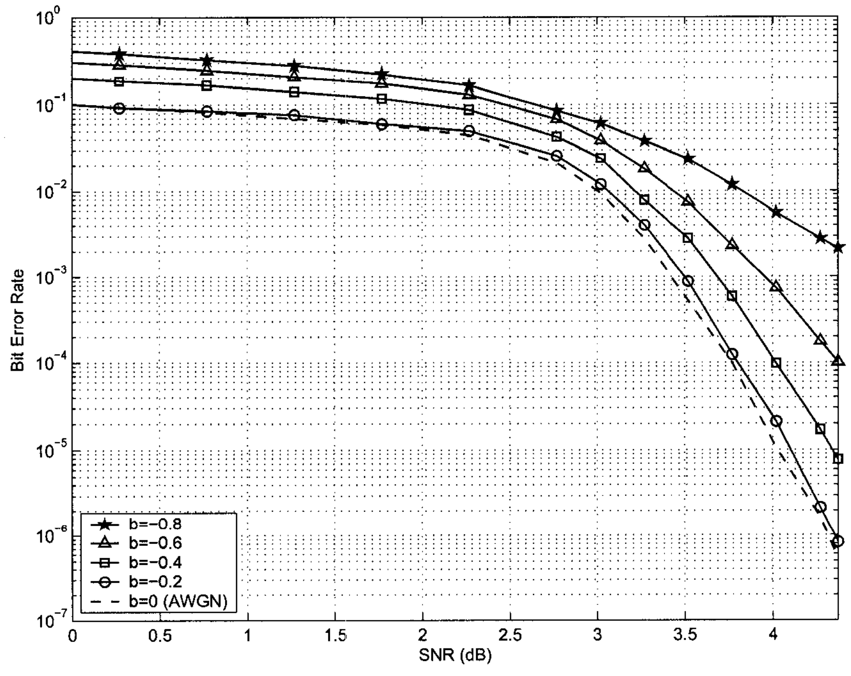

Figure 5.13: Effect of low-frequency ACGN on *Code A*.

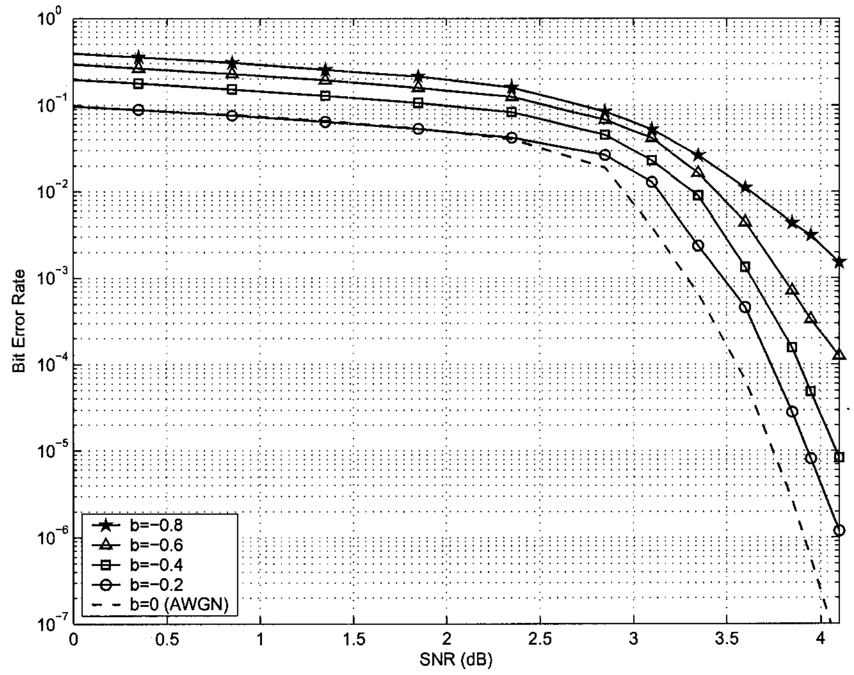

Figure 5.14: Effect of low-frequency ACGN on *Code B*.

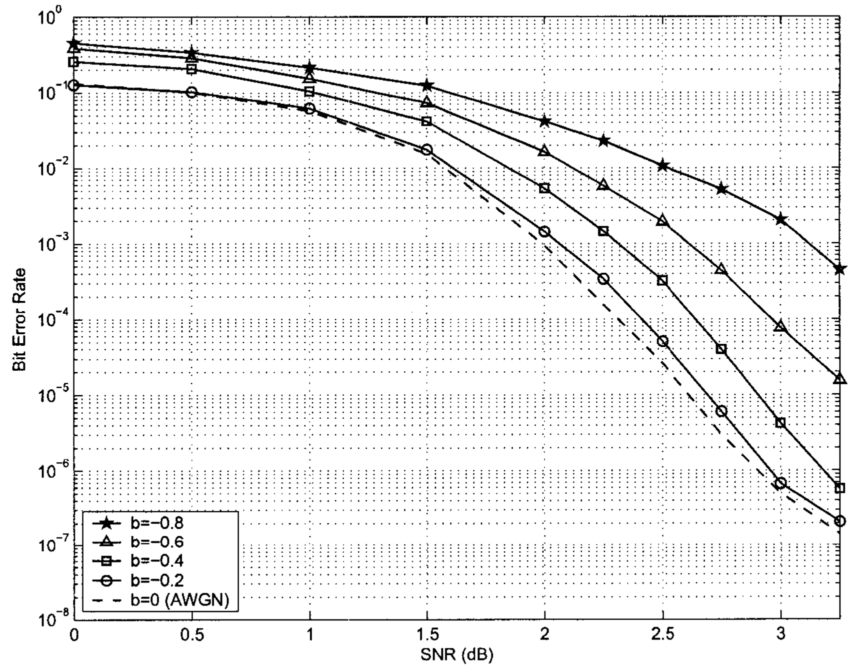

Figure 5.15: Effect of low-frequency ACGN on *Code C*.

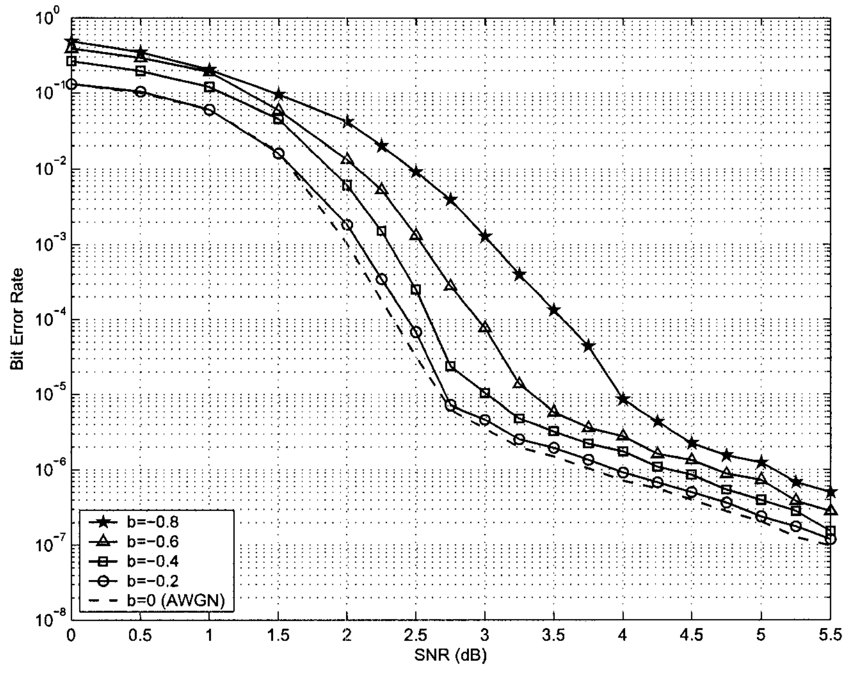

Figure 5.16: Effect of low-frequency ACGN on *Code D*.

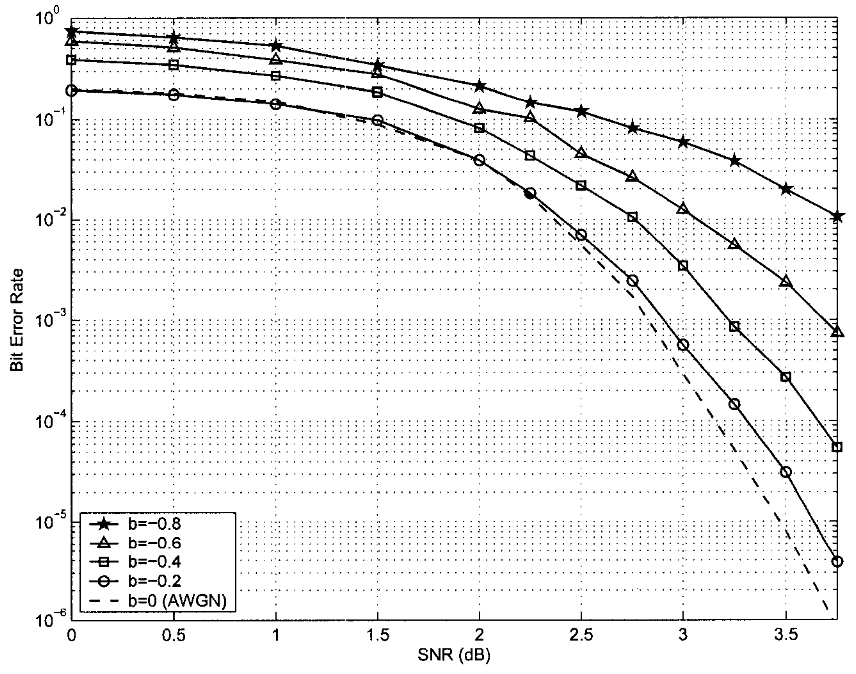

Figure 5.17: Effect of low-frequency ACGN on *Code E*.

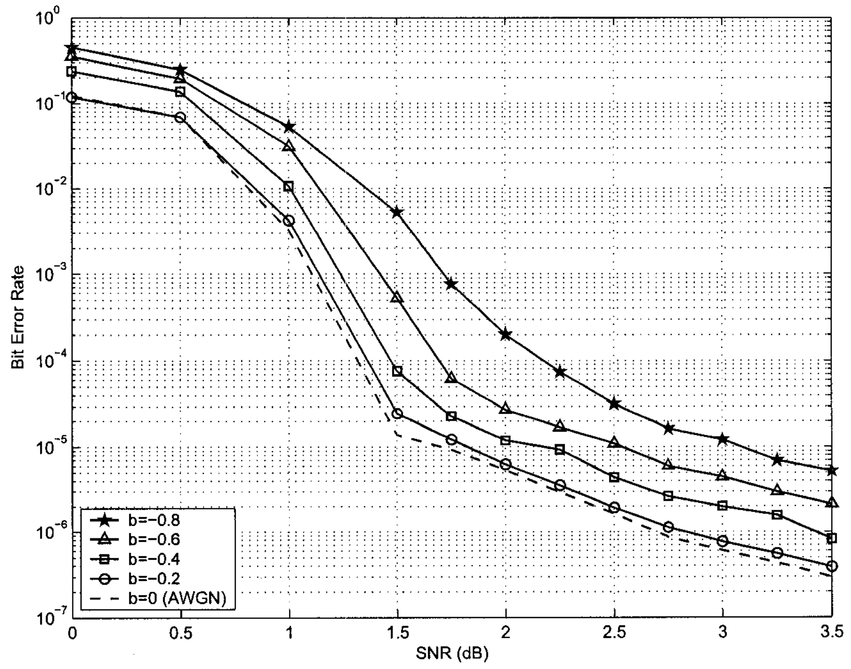

Figure 5.18: Effect of low-frequency ACGN on *Code F*.

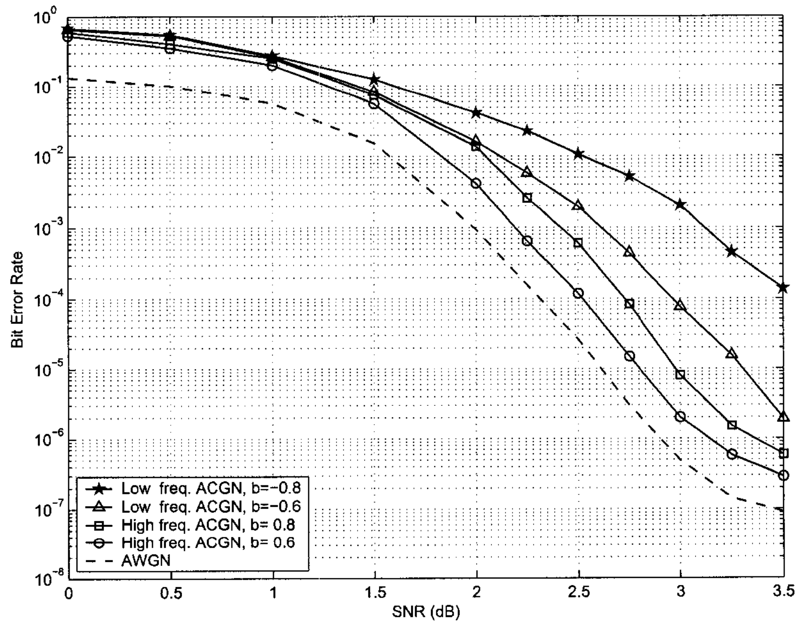

Figure 5.19: Comparison between BER performance under low-frequency and high-frequency ACGN, *Code C*.

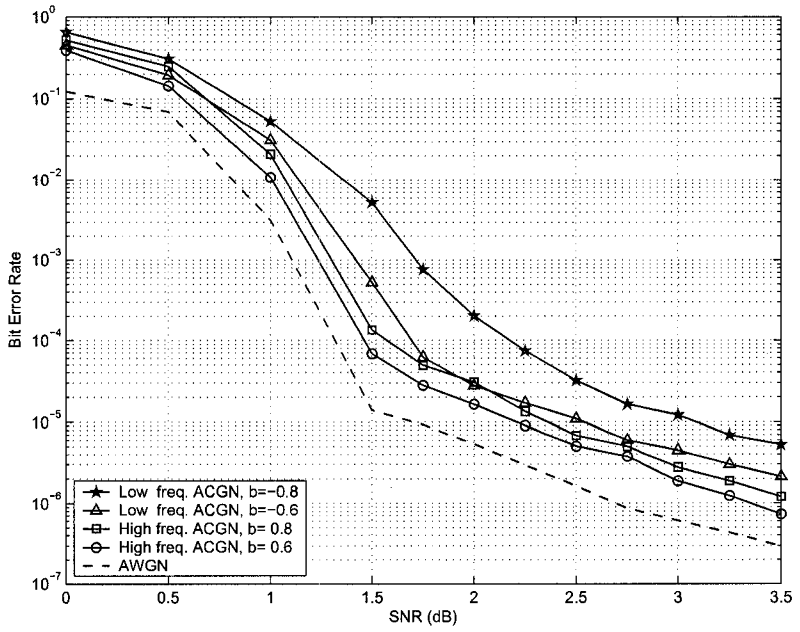

Figure 5.20: Comparison between BER performance under low-frequency and high-frequency ACGN, *Code F*.

5.3 Conclusion

In this chapter, the BER performance of LDPC codes in the presence of ACGN were investigated. Two filter models were used to generate low-frequency and high-frequency ACGN. The results for both the low-pass and high-pass schemes showed that the BER performance of LDPC codes degrades in the presence of ACGN, and that as the coloration increases, the performance degradation becomes worse. Such degradation is more severe for low-frequency ACGN. However, regardless of the performance degradation in the presence of ACGN, the overall performance behavior (in both the waterfall and error floor regions) is qualitatively similar to the behavior in AWGN.

The filter models used in this chapter generated correlations between noise samples. This enabled us to investigate the effects of this simple coloration model on the BER performance. In the next chapter, the effects of $1/f$ noise are studied.

Chapter 6

Performance Evaluation of LDPC Codes in the Presence of $1/f$ Noise

As mentioned in Section 2.2.5, $1/f$ noise is associated with clock noise in digital systems and considered as an important impairment that is observed in solid-state circuits [13, 22–25, 28]. This chapter investigates the effects of $1/f$ noise on the performance of LDPC codes. The system model used in this chapter is different from the system model used in the previous chapter. In Chapter 5, only ACGN noise exists in the channel. However, in the system model used in this chapter, both AWGN and $1/f$ noise exist in the channel model.

6.1 System Model

Figure 6.1 illustrates the system model used in the simulation study. In this model, the transmitted signal, $x(D)$, is transmitted using PAM-2 modulation. The transmitted signal is then corrupted by AWGN and $1/f$ noise to form the received signal. The received signal, $y(D)$, in this model can be expressed by its D-transform as:

$$y(D) = x(D) + n_W(D) + n_{1/f}(D) \tag{6.1}$$

where $n_W(D)$ is the AWGN sequence and $n_{1/f}(D)$ is the $1/f$ noise sequence. Similar to the previous chapters, the LDPC decoder in this model performs standard soft-decision message-passing decoding and uses full *tanh* processing in the parity-check nodes.

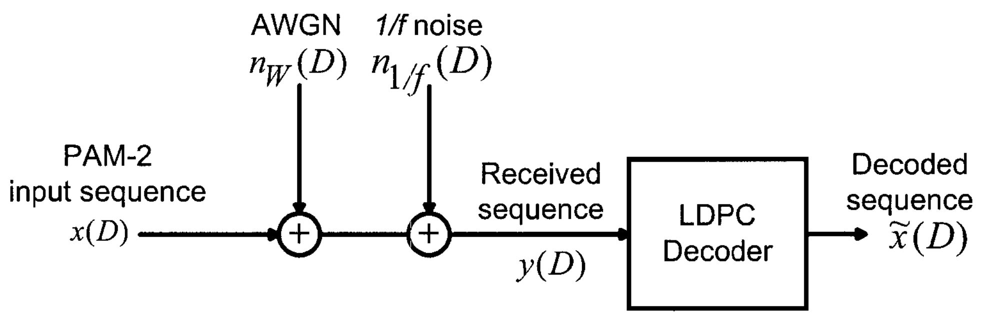

Figure 6.1: System model used for the performance evaluation in the presence of $1/f$ noise.

In the simulation study, the performance of the LDPC codes in the $1/f$ noise model is compared to the performance of the equivalent pure AWGN channel. To do so, the total power of the noise (*i.e.*, the sum of the AWGN variance and the $1/f$ noise variance) is kept equal to the power of AWGN in a pure AWGN channel:

$$\sigma_{1/f}^2 + \sigma_{n_2}^2 = \sigma_{n_1}^2 \tag{6.2}$$

where $\sigma_{1/f}^2$ is the variance of the $1/f$ noise, $\sigma_{n_2}^2$ is the variance of the AWGN in the $1/f$ noise model, and $\sigma_{n_1}^2$ is the variance of the AWGN in a pure AWGN channel of the same total noise power. It should be noted that $\sigma_{n_1}^2$ can be computed according to the average bit energy of the transmitted sequence and the operating SNR using Equation 5.4.

6.1.1 $1/f$ Noise Generation

There are various methods in the literature for generating $1/f$ noise. The method used in this study is based on IIR filtering of an AWGN noise sequence [28] (see Figure 6.2). The output of the filter is a $1/f$ noise sequence that has a spectrum with -10 dB drop per decade[1]. The transfer function of the $1/f$

[1]See Figure 2.12 in Chapter 2.

coloration filter, $F_{1/f}(z)$, is [28]:

$$F_{1/f}(z) = \frac{a}{(1-z^{-1})^{1/2}} \tag{6.3}$$

where a is a gain factor. $F_{1/f}(z)$ can be expanded as

$$F_{1/f}(z) = \frac{a}{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_k z^{-k} + \ldots.} \tag{6.4}$$

where $\{b_0, b_1, \ldots\}$ are the coefficients of the denominator, which can be computed in a recursive way as follows:

$$\begin{aligned} b_0 &= 1 \\ b_k &= (k - \tfrac{1}{2})\tfrac{b_{k-1}}{k} \end{aligned} \tag{6.5}$$

In the simulation study, the denominator was expanded to 30 coefficients. Figure 6.3 shows the resulting frequency response of $F_{1/f}(z)$ for $a = 1$ and 30 coefficients in the denominator. Note how the filter strongly emphasizes the low-frequencies, more strongly even than the reddened white noise considered in the previous chapter.

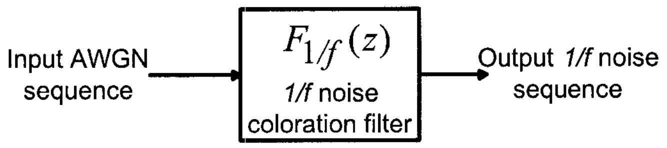

Figure 6.2: $1/f$ noise generator.

In comparison with the ACGN coloration filter, $F_C(z)$, in Chapter 5, the $1/f$ coloration filter, $F_{1/f}(z)$, produces a stronger correlation between noise samples. This can be seen in Figure 6.4, where the impulse responses of $F_C(z)$ and $F_{1/f}(z)$ are depicted. Figure 6.4(A) shows the impulse response for $F_C(z)$ with $b = -0.8$ and $a = 0.6$. Figure 6.4(B) shows the impulse response for $F_{1/f}(z)$ with $a = 0.6580$. Both filters have a unity gain.

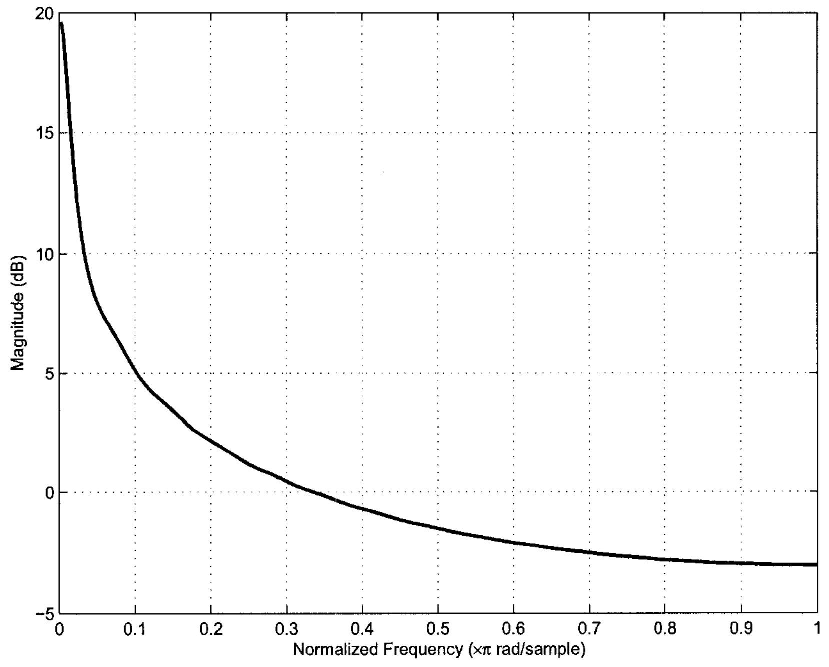

Figure 6.3: Frequency response of $F_{1/f}(z)$, $a = 1$.

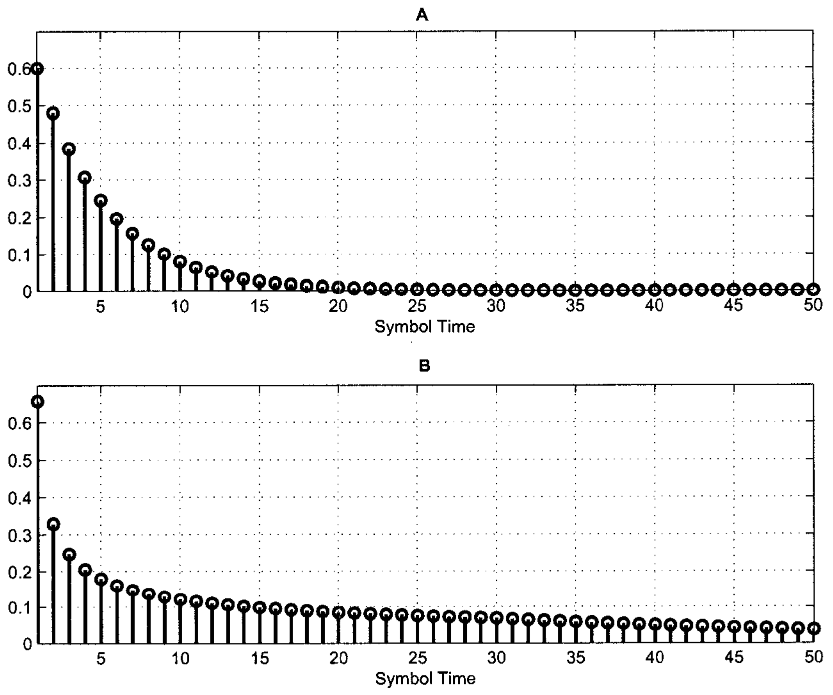

Figure 6.4: Impulse response of A) $F_C(z)$ with $b = -0.8$ and unity gain and, B) $F_{1/f}(z)$ with unity gain.

6.2 Simulation Results

Figures 6.5 to 6.10 show the BER performance of *Code A* to *Code F*, respectively, for the channel model of Figure 6.1. Each figure also shows the BER performance for pure AWGN as well as for pure $1/f$ noise channel. In the simulation study, the BER performance was measured for various levels of $1/f$ noise as a percentage of the total noise power in the channel. The total noise power was kept constant, at each SNR, in association with Equation 6.2. This means that by increasing the power of the $1/f$ noise in the channel, the power of the AWGN is decreased to keep the total noise power constant. As an example, according to Equation 5.4, the total variance of the noise, $\sigma^2_{n_1}$, at SNR = 2 dB for *Code C* with the rate of 0.5 is 0.6310. For the case that 10% of the noise in the channel is $1/f$ noise, the variance of the $1/f$ noise, $\sigma^2_{1/f}$, is 0.0631 and the variance of the AWGN in the channel, $\sigma^2_{n_2}$, is 0.5679.

These figures show that as the percentage of $1/f$ noise increases, the BER loss becomes more severe. The BER loss for 10% $1/f$ noise in the channel is more than a factor of 10 in the waterfall region (*e.g.* see Figures 6.7 and 6.10). The BER performance loss increases when a higher level of $1/f$ noise exists in the channel. As an example, when 50% of the total noise in the channel is $1/f$ noise, the BER loss of *Code A* at SNR = 4 dB and *Code C* at SNR = 3 dB is about a factor of 30. This trend can be also seen in the SNR performance loss[2] of these codes. The SNR performance loss for 25% $1/f$ noise is more than 1 dB while for 50% $1/f$ noise, the SNR performance loss is more than 2.5 dB (*e.g.* Figure 6.10).

Interestingly, the results show that the BER loss in the error floor region is less than in the waterfall region. This effect can be easily seen in Figures 6.7 and 6.8. This is different from the behavior of LDPC codes in the presence of ACGN. However, it should be noted that for the highly colored ACGN

[2]It should be noted that the SNR performance comparison is more common in the waterfall region, since at the error floor the BER performance is almost constant and a small gap between BERs of the codes, at a same SNR, in the error floor region can be interpreted as a huge SNR performance loss.

with $|b| = 0.8$, a similar effect (but less severe), was also seen (see Figure 5.16). It appears that the effects of highly colored noise, such as $1/f$ noise or ACGN with $|b| = 0.8$, is more detrimental in the waterfall region, and that at higher SNRs, in which the power of the colored noise is decreased, the BER performance is less degraded.

In general, when compared with the effects of ACGN, $1/f$ noise causes more degradation. However, such an increased degradation is in accordance with the results of Chapter 5. The reason is that $1/f$ noise is more strongly colored than the ACGN model that we assumed and, as observed in Chapter 5, the more the coloration of noise, the worse the performance. In addition, the comparison between low-pass (*i.e.* reddish) ACGN and high-pass ACGN (*i.e.* blueish) in Chapter 5 showed that low-pass ACGN is more detrimental to performance. The $1/f$ noise model is a very low-pass model.

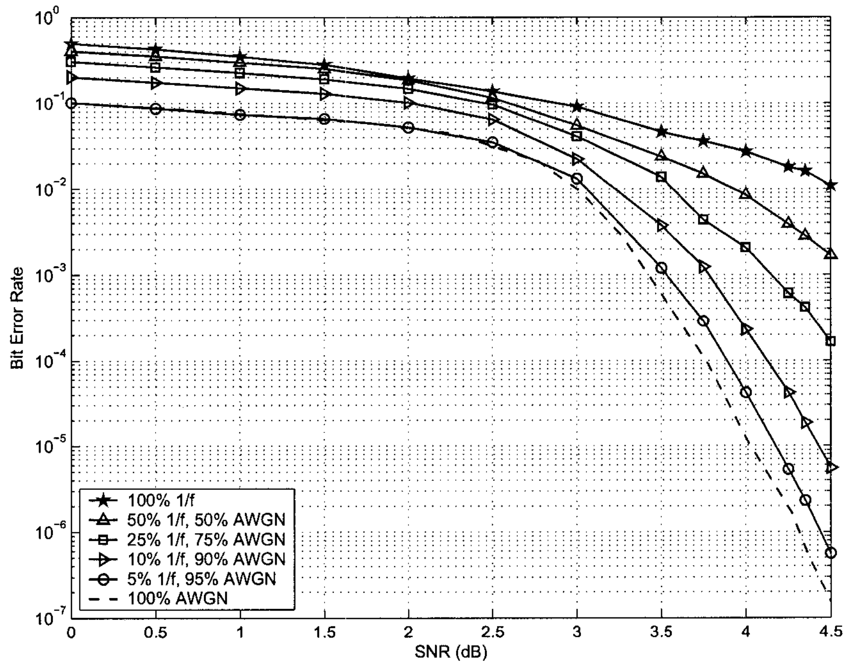

Figure 6.5: Effect of $1/f$ noise on *Code A*.

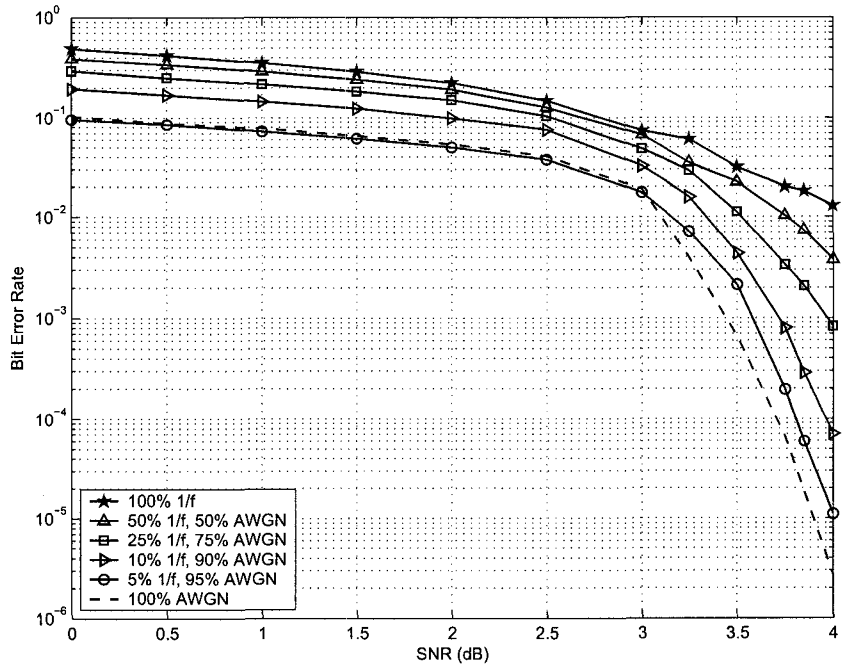

Figure 6.6: Effect of $1/f$ noise on *Code B*.

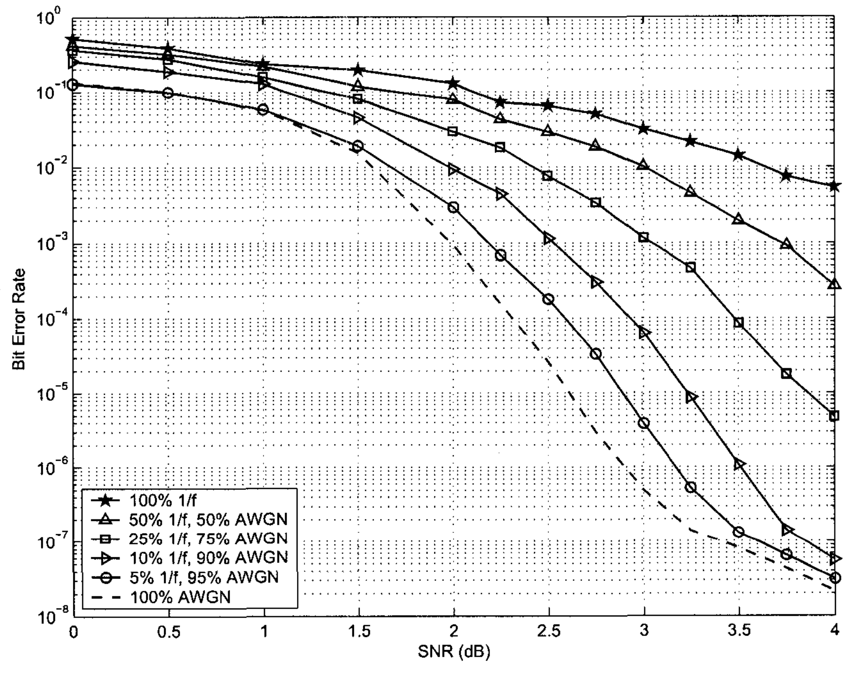

Figure 6.7: Effect of $1/f$ noise on *Code C*.

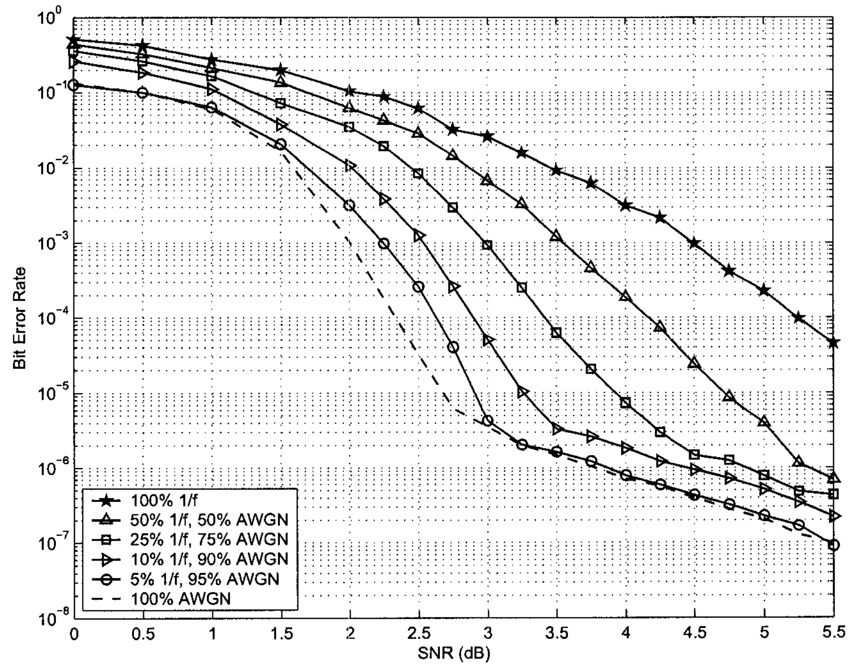

Figure 6.8: Effect of $1/f$ noise on *Code D*.

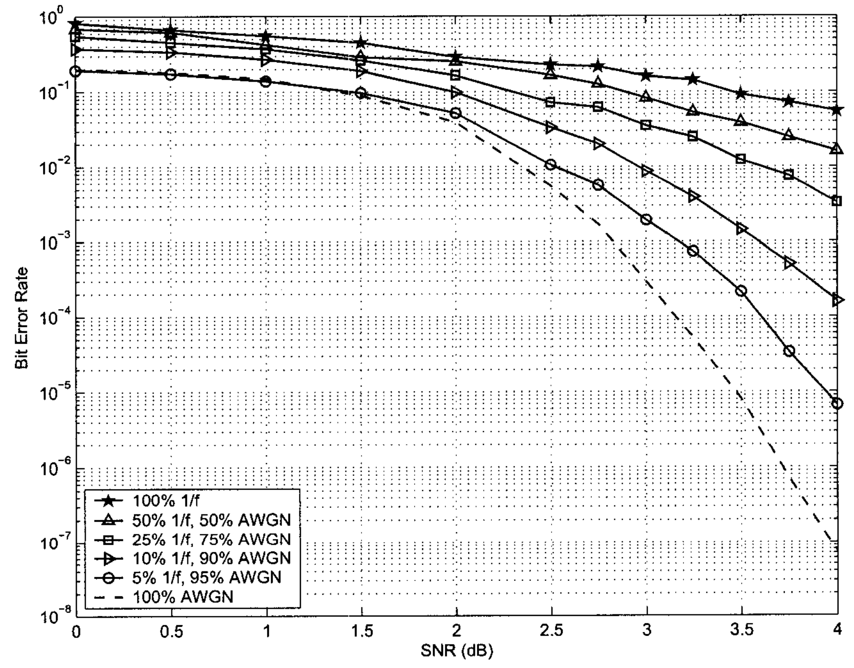

Figure 6.9: Effect of $1/f$ noise on *Code E*.

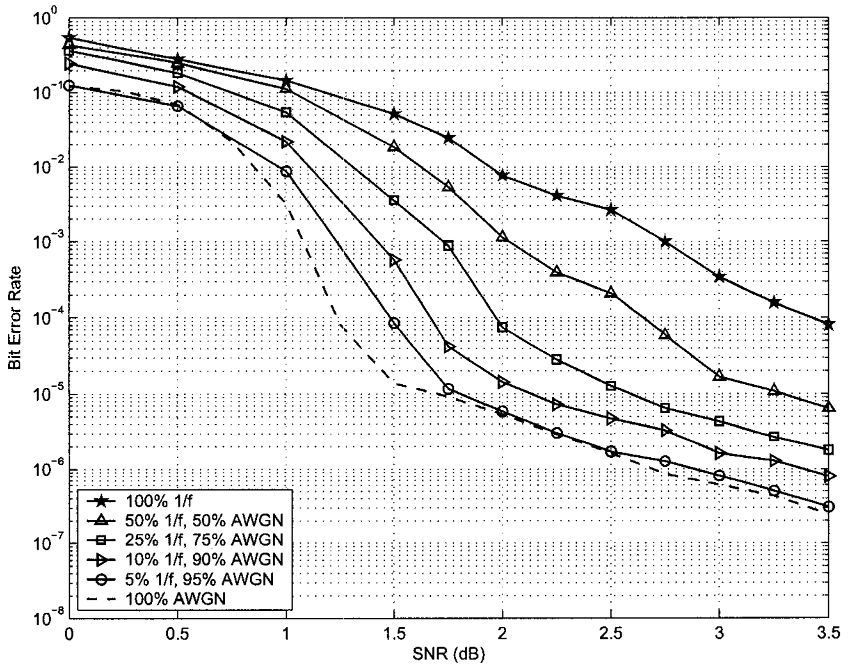

Figure 6.10: Effect of $1/f$ noise on *Code F*.

6.3 Conclusion

In this chapter, the BER performance of LDPC codes in the presence of $1/f$ noise was investigated. In comparison with the ACGN model used in Chapter 6, the $1/f$ noise model considered in this chapter was a more strongly colored model. The BER performance was measured for various percentages of $1/f$ noise in the channel. The simulation results showed that as the percentage of $1/f$ noise increases in the channel, the performance loss increases. The observed BER loss was more in the waterfall region than in the error floor region. It was concluded that highly colored noise, such as $1/f$ noise or ACGN with $|b| \geq 0.8$, is more detrimental on the performance in the waterfall region. Also, at higher SNRs, in which the power of colored noise is decreased, the BER performance of the LDPC codes is less degraded. The simulation results suggest that, in general, compared to the effects of ACGN, $1/f$ noise causes more performance degradation.

Chapter 7

Conclusions and Future Work

7.1 Main Contributions

Low-Density Parity-Check codes are among the most powerful error control codes known. They can produce error-correcting performance close to the Shannon limit and can be decoded using iterative decoding algorithms with only linear complexity. These key advantages have made LDPC codes a candidate coding scheme for various novel applications and upcoming standards. The next generation of Ethernet, 10GBASE-T, is among such standards. This standard aims to provide a data rate of 10 Gbps over four-pair UTP cabling with a minimum bit error rate of 10^{-12}. This data rate is 10 times faster than the data rate of the existing wireline Ethernet standard, 1000BASE-T. Despite its numerous advantages, 10GBASE-T transmission suffers from numerous impairments that must be tackled.

This thesis first reviewed the organization of 1000BASE-T and 10GBASE-T systems and explained the major impairments that arise in their communication media. Some of these impairments are common between these two standards, and some of them, such as ANEXT, are either unique to or more severe in 10GBASE-T. The thesis has also reviewed the alternative coding and equalization schemes for 1000BASE-T and 10GBASE-T systems. It provided simulation results for the performance of alternative coding and equalization schemes, such as decision-feedback equalization and delayed-decision-feedback-sequence estimation, that are being considered for 1000BASE-T. The advantages of LDPC codes over these schemes, with respect to the performance and complexity in high-speed applications such as 10GBASE-T, were then explained.

In Chapter 4, performance evaluation results were presented for recent candidate LDPC codes for 10GBASE-T Ethernet and some standard LDPC codes over AWGN and ISI channels. It was shown that at the same level of signal-to-noise-and-interference ratio, LDPC codes appear to be more vulnerable to AWGN than to ISI. It was also shown in the simulations that, given this scenario, the performance of LDPC codes over an ISI channel is upper-bounded by its performance in the AWGN channel. An important application for the proposed results is the interpretation of the performance of LDPC codes in the channels in which the levels of AWGN and residual ISI are known or can be approximated. In another words, since LDPC codes appear to be more sensitive to AWGN than to ISI, it is possible to characterize the AWGN equivalent for given amounts of residual ISI in a channel and then estimate the worst-case BER performance of the LDPC codes.

Chapter 5 presented the BER performance evaluation of LDPC codes in the presence of ACGN. Two simple filter models were used in this chapter to generate low-frequency and high-frequency ACGN. The results for both low-pass and high-pass schemes showed that the BER performance of LDPC codes degrades in the presence of ACGN, and as the degree of coloration increases, the performance degradation becomes worse. Such degradation is more severe for low-frequency ACGN. It was also observed that the overall performance loss, in both the waterfall and error floor regions, is comparable in both ACGN scenarios.

The filter models used in Chapter 5 introduce a simple correlation between noise samples. In Chapter 6, the effects of $1/f$ noise, which is a more strongly colored noise, were studied. It was shown that as the percentage of $1/f$ noise in the channel increases, the BER loss increases. The BER performance loss can be very severe when a high level of $1/f$ noise exists. This trend was also observed in the SNR performance loss of LDPC codes. In general, the observed performance loss was greater in the waterfall region than in the error floor region. It was concluded that highly colored noise types, such as $1/f$ noise or ACGN with $|b| \geq 0.8$, are more detrimental to performance in the waterfall region. Also by decreasing the power of the colored noise at higher SNRs, the BER performance of the LDPC codes is less degraded. In comparison with the effects of ACGN, $1/f$ noise causes more degradation. However, such an increased degradation is in accordance with the results of Chapter 5. The reason is that $1/f$ noise is more colored than the ACGN considered, and as observed in Chapter 5, the more the coloration of the noise, the worse the performance. In addition, the comparison between low-pass ACGN and high-pass ACGN in Chapter 5 showed that low-pass ACGN is more detrimental to the performance. The $1/f$ noise model is also a very low-pass model.

7.2 Possible Future Work

Possible future work, which the author would like to undertake shortly, is the investigation of the effects of ANEXT in 10GBASE-T systems, specifically, the effects on the BER performance of LDPC codes. Such an investigation would be useful for understanding the actual performance of 10GBASE-T candidate LDPC codes in more realistic situations. This study can be done by using a similar method used in Chapter 6. In [4], ANEXT spectrum has been measured in a 10GBASE-T system and modeled as a Finite Impulse Response (FIR) filter. However, the length of the FIR filter is long (*i.e.* more than 2500 taps). But it is possible to approximate this filter with an IIR filter with less taps, and model an ANEXT sequence by passing an AWGN sequence, which has a flat spectrum, through this IIR filter. The output ANEXT sequence can be then treated as colored noise in the system for the sake of the BER performance evaluation.

The results provided in this thesis give insight into the performance of LDPC codes in real practical applications. Theoretical analysis of the effects studied in this thesis should be undertaken as a follow-up project. For example, the possible relationships between vulnerability to colored noise and the Tanner graph structure of the LDPC codes should be investigated.

The modulation scheme used in the majority of simulations of the thesis was PAM-2. PAM-2 is a modulation scheme that is used in most of the LDPC code literature and in related simulation studies. Although the preliminary results of this thesis showed that the same error-correcting behavior should be expected for PAM modulations with more signaling levels, it would be worth carrying out a more in-depth study of the effects of these impairments when the PAM modulation has more signalling levels. Such a study might be useful for the 10GBASE-T standard, in which PAM-8, PAM-12 and PAM-16 are candidate modulation schemes. In addition to this, it would be also interesting to study the effects of impairments for the case when, instead of using the Sum-Product algorithm, the LDPC decoder uses more implementable approximate methods, such as the Min-Sum algorithm.

Another interesting possible research direction would be the investigation of the effects of SNR mismatch at the receiver on the performance of LDPC codes in the presence of AWGN as well as colored noise. Similar studies have been recently done for some special LDPC codes used in magnetic recording [72]; however, the effects of SNR mismatch, to the best of our knowledge, have not yet been studied for colored channel models such as the ones used in this thesis.

Bibliography

[1] *Ethernet darwing*, www.ethermanage.com/ethernet, July 2005.

[2] E. F. Haratsch and K. Azadet, "A 1-Gb/s joint equalizer and trellis decoder for 1000BASE-T Gigabit Ethernet," *IEEE Journal of Solid-State Circuits*, no. 3, pp. 374–384, March 2001.

[3] G. Malhotra, J. H. Jeong, and M. Kavehrad, "Joint MIMO equalization and decoding for 10GBASE-T transmissions," in *Proceedings of the IEEE Global Telecommunications Conference*, Dallas, Texas, USA, Dec. 2004, pp. 918–922.

[4] P. Anand and S. Bates, "Characterization of alien-next in 10GBASE-T systems," in *Proceedings of 2005 IEEE Canadian Conference on Electrical and Computer Engineering*, Saskatoon, SK, Canada, May 2005, pp. 1059–1062.

[5] M. Kavehrad, J. F. Doherty, J. H. Jeong, A. Roy, and G. Malhotra, "10Gbps transmission over standard category-5 copper cable," in *Proceedings of the IEEE Global Telecommunications Conference*, San Fransico, CA, USA, Dec. 2003, pp. 4106–4110.

[6] R. Seifert, *Gigabit Ethernet.* MA: Addison-Wesley, 1998.

[7] *Physical Layer Parameters and Specifications for 1000 Mb/s Operation Over 4-Pair of Category-5 Balanced Copper Cabling, Type 1000BASE-T.* IEEE Standard 802.3ab-1999, 1999.

[8] The IEEE P802.3an 10GBASE-T Task Force, www.ieee802.org/3/an, Feb. 2005.

[9] P. Izzo, *Gigabit Networks.* John Wiley and Sons Inc., 2000.

[10] M. Hatamian, O. E. Agazzi, J. Creigh, H. Samueli, A. J. Castellano, D. Kruse, A. Madisetti, N. Yousefi, K. Bult, P. Pai, M. Wakayama, M. M. McConnell, and M. Colombatto, "Design considerations for Gigabit Ethernet 1000BASE-T twisted pair transceivers," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, Santa Clara, CA, May 1998, pp. 335–342.

[11] S. Bates, "VLSI issues for the implementation of 10GBASE-T Ethernet," in *Proceedings of the Very Large Scale Integrated Circuuts Conference*, 2004.

[12] K. Azadet, "Gigabit Ethernet over unshielded twisted pair cables," in *Proceedings of the International Sympousium VLSI Technology, Systems, and Applications*, Taipei, Taiwan, Jun. 1990, pp. 167–170.

[13] A. Hajimiri and T. H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE Journal of Solid-State Circuits*, no. 2, pp. 179–194, Feb. 1998.

[14] J. G. Proakis, *Digital Communication.* New York: McGraw-Hill, 2001.

[15] J. E. Smee and N. C. Beaulieu, "Error-rate evaluation of linear equalization and decision feedback equalization with error propagation," *IEEE Transactions on Communications*, no. 5, pp. 656–665, May 1998.

[16] W. Gerstacker, R. Fischer, and J. Huber, "A transmission scheme for twisted pair lines with coding, precoding, and blind equalization," in *Proceedings of the IEEE Global Telecommunications Conference*, Phoenix, Arizona, USA, Nov. 1997, pp. 52–56.

[17] V. Stojanovic and M. Horowitz, "Modeling and analysis of high-speed links," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, Sept. 2003, pp. 589–594.

[18] J. D. Wang and H. Y. Chung, "Trellis coded communication systems-colored noise and the swapping technique," *IEEE Transactions on Communications*, no. 9, pp. 1549–1556, Sept. 1990.

[19] M. Tomlinson, "New automatic equaliser employing modulo arithmetic," *Electron. Lett.*, vol. 7, pp. 138–139, March 1971.

[20] H. Harashima and H. Miyakawa, "Matched-transmission technique for channels with intersymbol interference," *IEEE Transactions on Communications*, vol. COM-20, pp. 774–780, Aug 1972.

[21] M. S. Keshner, "$1/f$ noise," *The Proceedings of the IEEE*, no. 3, pp. 212–218, March 1982.

[22] G. Ghibaudo, "Low frequency noise and fluctuations in advanced CMOS devices," in *Proceedings of SPIE - Int. Soc. Opt. Eng.*, 2003, pp. 16–28.

[23] E. Simoen and C. Clays, "On the flicker noise in submicron silicon MOSFETs," *Solid-State Electron.*, pp. 865–882, 1999.

[24] M. Marin, Y. A. Allogo, M. Murcia, P. Llinares, and J. C. Vildeuil, "Low frequency noise characterization in 0.13mm p-MOSFET. impact of scaled-down 0.25, 0.18 and 0.13mm technologies on 1/f noise," *Microelectron. Reliab.*

[25] D. Rold, M. Simoen, E. Mertens, S. Schaekers, M. Badenes, and G. Decoutere, "Impact of gate oxide nitridation process on $1/f$ noise in 0.18mm CMOS," *Microelectron. Reliab.*, pp. 1933–1938, 2001.

[26] R. Whittle, "DSP generation of pink (1/f) noise," www.firstpr.com.au/dsp/pink-noise, July 2003.

[27] M. Gardner, "White and brown music, fractal curves and one-over-f fluctuations," *Scientific American*, p. 288, 1978.

[28] N. J. Kasdin, "Discrete simulation of colored noise and stochastic processes and $1/f^{\alpha}$; power law noise generation," *The Proceedings of the IEEE*, no. 5, pp. 803–827, May 1995.

[29] J. G. Proakis, M. Salehi, and G. Bauch, *Contemporary Communication Systems using MATLAB and Simulink.* Thomson-Brooks/Cole, 2004.

[30] S. Haykin, *Adaptive Filters Theory.* Prentice Hall, 2001.

[31] M. Reuter, J. C. Allen, J. R. Zeidler, and C. North, "Mitigating error propagation effects in a decision feedback equalizer," *IEEE Transactions on Communications*, no. 11, pp. 1064–1075, Nov. 2001.

[32] J. G. Proakis and M. Salehi, *Fundementals of Communication Systems.*

Pearson Prentice Hall, 2005.

[33] P. Monsen, "Adaptive equalization of the slow fading channel," *IEEE Transactions on Communications*, pp. 1064–1075, Aug. 1974.

[34] ——, "Theoretical and measured performance of a DFE modem on a fading multipath channel," *IEEE Transactions on Communications*, pp. 1144–1153, Oct. 1977.

[35] R. Fischer, W. Gerstacker, and J. Huber, "Dynamics limited precoding, shaping, and blind equalization for fast digital transmission over twisted pair lines," *IEEE J. Select. Areas. Commun.*, vol. SAC-13, pp. 1622–1633, Dec. 1995.

[36] A. Duel-Hallen and C. Heegard, "Delayed Decision-Feedback Sequence Estimation," *IEEE Transactions on Communications*, vol. 37, no. 5, pp. 428–436, May 1989.

[37] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT-13, pp. 260–269, April 1967.

[38] J. G.D. Forney, "The Viterbi algorithm," *The Proceedings of the IEEE*, vol. 61, pp. 268–278, March 1973.

[39] J. Hayes, "The Viterbi algorithm applied to digital data transmission," *IEEE Communications Magazine*, vol. 13, no. 2, pp. 15–20, Mar 1975.

[40] H. Kobayashi, "Correlative level coding and maximum likelihood decoding," *IEEE Transactions on Information Theory*, vol. IT-17, pp. 586–594, Sept. 1971.

[41] J. Omura, "Maximum likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *Info. Sci.*, vol. 3, pp. 243–266, July 1971.

[42] J. G. D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 363–378, May 1972.

[43] R. G. Gallager, *Low Density Parity Check Codes.* Cambridge, MA: MIT

Press, 1963.

[44] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, 1996.

[45] M. Sipser and D. Spielman, "Expander codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 1710–1722, 1996.

[46] T. J. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, Feb. 2001.

[47] T. J. Richardson and R. Urbanke, "The capacity of low-density paritycheck codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, Feb. 2001.

[48] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, pp. 58–60, Feb. 2001.

[49] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. IT-42, pp. 533–547, 1981.

[50] C. B. Schlegel and L. C. Perez, *Trellis and Turbo Coding.* IEEE Press, 2004.

[51] A. Blanksby and C. Howland, "A 690-mw 1-Gb/s 1024-b rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, March 2002.

[52] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Department of Electrical Engineering, Linkoping University, Sweden, 1996.

[53] M. P. C. Fossorier, M. Mihaljevic, and I. Imai, "Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation," *IEEE Transaction on Communication*, vol. 47, no. 5, pp. 673–680, May 1999.

[54] A. Anastasopoulos, "A comparison between the sum-product and the

min-sum iterative detection algorithms based on density evolution," in *Proceedings of the IEEE Global Telecommunications Conference*, vol. 2, Nov. 2001, pp. 1021–1025.

[55] F. Guilloud, E. Boutillon, and J.-L. Danger, "λ-min decoding algorithm of regular and irregular LDPC codes," in *Proceedings of 3rd International Symposium on Turbo Codes (ISTC 03)*, Brest, France, 1-5 Sept. 2003, pp. 451–454.

[56] S. Howard, C. Schlegel, and V. Gaudet, "A degree-matched check node approximation for LDPC decoding," in *Proceedings of the IEEE International Symposium on Information Theory*, Adelaide, Australia, 4-9 Sept. 2005.

[57] J. Heo, "Analysis of scaling soft information on low density parity check codes," *Electronics Letters*, vol. 39, no. 2, pp. 219–221, Jan. 2003.

[58] G. Battail and H. M. S. El-Sherbini, "Coding for radio channels," *Ann. Telecommun*, vol. 37, pp. 75–96, Jan./Feb. 1982.

[59] A. J. Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, Sept. 1999.

[60] A. Sridharan and D. Costello, "A new construction method for low density parity check convolutional codes," in *Proceedings of the IEEE Information Theory Workshop*, Bangalore, India, Oct. 2002, pp. 20–25.

[61] S. B. Z. Chen and X. Dong, "Performance of low-density parity-check convolutional codes," available from www.ece.ualberta.ca/sbates, 2004.

[62] S. Bates and G. Block, "A memory based architecture for for FPGA implementations of low-density parity-check convolutional decoders," in *Proceedings of the IEEE Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, May 2005.

[63] R. Swamy, S. Bates, and T. L. Brandon, "Architectures for ASIC implementations of low-density parity-check convolutional encoders and decoders," in *Proceedings of the IEEE Symposium on Circuits and Systems (ISCAS), Kobe, Japan*, 2005.

[64] Y. Gu and K. K. Parhi, "Interleaved trellis coded modulation and decoding for 10 Gigabit Ethernet over copper," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,* May 2004, pp. V25–V28.

[65] J. H. Jeong and M. Kavehrad, "Turbo equalizations for broadband transmission on category 6 cable," in *Proceedings of the IEEE Global Telecommunications Conference*, Dallas, Texas, USA, Dec. 2004, pp. 1081–1085.

[66] P. R. Chevillat and E. Eleftheriou, "Decoding of trellis-encoded signals in the presence of intersymbol interference and noise," *IEEE Transactions on Communications*, vol. 37, no. 7, pp. 669–676, July 1989.

[67] E. Baccarelli and S. Galli, "New results about analysis and design of TCM for ISI channels and combined equalization/decoding," *IEEE Transactions on Communications*, vol. 46, no. 4, pp. 417–420, April 1998.

[68] K. J. Kerpez, "Viterbi receivers in the presence of severe intersymbol interference," in *Proceedings of the IEEE Global Telecommunications Conference*, Dec. 1990, pp. 2009–2013.

[69] K. A. Hamied and G. L. Stuber, "Performance of trellis-coded modulation for equalized ISI channels," in *Proceedings of the 43rd IEEE Vehicular Technology Conference*, May 1993, pp. 9–12.

[70] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on reed-solomon codes with two information symbols," *IEEE Communications Letters*, vol. 7, no. 7, pp. 317–319, July 2003.

[71] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[72] W. Tan and J. R. Cruz, "Signal-to-noise ratio mismatch for low-density parity-check coded magnetic recording channels," *IEEE Transactions on Magnetics*, no. 2, pp. 498–506, March 2004.

Appendix A

Programs

A.1 Decision Feedback Equalizer for 1000BASE-T Ethernet

A.1.1 Header Files

A.1.1.1 simulation.h

```
#if !defined(SIMULATION_H)
#define SIMULATION

const int     NumChannels=4;
const int     CIR_Length=14;

const double CIR[CIR_Length]={
    1.00000000000000,
    0.90909090909091,
    0.36363636363636,
    0.18181818181818,
   -0.18181818181818,
   -0.22727272727273,
   -0.36363636363636,
   -0.20000000000000,
   -0.18181818181818,
   -0.10909090909091,
   -0.09090909090909,
   -0.07272727272727,
   -0.05454545454545,
   -0.03636363636364
};

const double LearningConst = 1e-4;               // Learning Constant
const double ForgettingConst= 1.0;               // Forgetting Constant
const int     FFE_Length = 14;
const int     FBE_Length = 13;
const int     DFE_Length= FFE_Length+FBE_Length;

#endif
```

A.1.1.2 Trellis1000BaseT.h

```
#if !defined(TRELLIS1000BASET_H)
#define TRELLIS1000BASET

const short NumCodeStates=8;
const short Num4DSubsets =8;
const short TrellisDepth =15;
const short Trellis_InFrameLength=TrellisDepth-1;
const short NumInBranches =4;
const short NumPaths=NumCodeStates;
const short CodeDimention =4;
const short NumSubsetTypes=2;

typedef struct TrellisState
{
  double StateMetric;
  short SurvivorBranch;
  short Survivor4DSymbol [NumInBranches][CodeDimention];
  double InBranchMetric [NumInBranches];
};

static TrellisState  Trellis [TrellisDepth][NumCodeStates]={0,0,{0},{0}};

const short InBranchState [NumCodeStates][NumInBranches]={
        {0,2,4,6},{0,2,4,6},{0,2,4,6},{0,2,4,6},
        {1,3,5,7},{1,3,5,7},{1,3,5,7},{1,3,5,7}};

const short InBranch4DSubset[NumCodeStates][NumInBranches]={
        {0,2,4,6},{2,0,6,4},{4,6,0,2},{6,4,2,0},
        {1,3,5,7},{3,1,7,5},{5,7,1,3},{7,5,3,1}};

const short Subset4DPattern[Num4DSubsets][NumSubsetTypes][CodeDimention]={
     { {0,0,0,0},  {1,1,1,1} },
     { {0,0,0,1},  {1,1,1,0} },
     { {0,0,1,1},  {1,1,0,0} },
     { {0,0,1,0},  {1,1,0,1} },
     { {0,1,1,0},  {1,0,0,1} },
     { {0,1,1,1},  {1,0,0,0} },
     { {0,1,0,1},  {1,0,1,0} },
     { {0,1,0,0},  {1,0,1,1} },
};


/* D0:  XXXX + YYYY */
const short constell_D0[64][4] = {
              { 0,  0,  0,  0}, {-2, 0,  0,  0}, { 0,-2,  0,  0}, {-2,-2, 0, 0},
              { 0,  0,-2,  0}, {-2, 0,-2,  0}, { 0,-2,-2,  0}, {-2,-2,-2, 0},
              { 0,  0,  0,-2}, {-2, 0,  0,-2}, { 0,-2,  0,-2}, {-2,-2, 0,-2},
              { 0,  0,-2,-2}, {-2, 0,-2,-2}, { 0,-2,-2,-2}, {-2,-2,-2,-2},
              { 1,  1,  1,  1}, {-1, 1,  1,  1}, { 1,-1,  1,  1}, {-1,-1, 1, 1},
              { 1,  1,-1,  1}, {-1, 1,-1,  1}, { 1,-1,-1,  1}, {-1,-1,-1, 1},
              { 1,  1,  1,-1}, {-1, 1,  1,-1}, { 1,-1,  1,-1}, {-1,-1, 1,-1},
              { 1,  1,-1,-1}, {-1, 1,-1,-1}, { 1,-1,-1,-1}, {-1,-1,-1,-1},
              { 2,  0,  0,  0}, { 2,-2,  0,  0}, { 2,  0,-2,  0}, { 2,-2,-2, 0},
              { 2,  0,  0,-2}, { 2,-2,  0,-2}, { 2,  0,-2,-2}, { 2,-2,-2,-2},
              { 0,  0,  2,  0}, {-2, 0,  2,  0}, { 0,-2,  2,  0}, {-2,-2, 2, 0},
              { 0,  0,  2,-2}, {-2, 0,  2,-2}, { 0,-2,  2,-2}, {-2,-2, 2,-2},
              { 0,  2,  0,  0}, {-2, 2,  0,  0}, { 0,  2,-2,  0}, {-2, 2,-2, 0},
              { 0,  2,  0,-2}, {-2, 2,  0,-2}, { 0,  2,-2,-2}, {-2, 2,-2,-2},
              { 0,  0,  0,  2}, {-2, 0,  0,  2}, { 0,-2,  0,  2}, {-2,-2, 0, 2},
              { 0,  0,-2,  2}, {-2, 0,-2,  2}, { 0,-2,-2,  2}, {-2,-2,-2, 2} };

/* D1:  XXXY + YYYX */
const short constell_D1[64][4] = {
              { 0,  0,  0,  1}, {-2, 0,  0,  1}, { 0,-2,  0,  1}, {-2,-2, 0, 1},
        { 0, 0,-2, 1}, {-2, 0,-2, 1}, { 0,-2,-2, 1}, {-2,-2,-2, 1},
        { 0, 0, 0,-1}, {-2, 0, 0,-1}, { 0,-2, 0,-1}, {-2,-2, 0,-1},
        { 0, 0,-2,-1}, {-2, 0,-2,-1}, { 0,-2,-2,-1}, {-2,-2,-2,-1},
        { 1, 1, 1, 0}, {-1, 1, 1, 0}, { 1,-1, 1, 0}, {-1,-1, 1, 0},
        { 1, 1,-1, 0}, {-1, 1,-1, 0}, { 1,-1,-1, 0}, {-1,-1,-1, 0},
        { 1, 1, 1,-2}, {-1, 1, 1,-2}, { 1,-1, 1,-2}, {-1,-1, 1,-2},
        { 1, 1,-1,-2}, {-1, 1,-1,-2}, { 1,-1,-1,-2}, {-1,-1,-1,-2},
        { 2, 0, 0, 1}, { 2,-2, 0, 1}, { 2, 0,-2, 1}, { 2,-2,-2, 1},
        { 2, 0, 0,-1}, { 2,-2, 0,-1}, { 2, 0,-2,-1}, { 2,-2,-2,-1},
        { 0, 0, 2, 1}, {-2, 0, 2, 1}, { 0,-2, 2, 1}, {-2,-2, 2, 1},
        { 0, 0, 2,-1}, {-2, 0, 2,-1}, { 0,-2, 2,-1}, {-2,-2, 2,-1},
        { 0, 2, 0, 1}, {-2, 2, 0, 1}, { 0, 2,-2, 1}, {-2, 2,-2, 1},
        { 0, 2, 0,-1}, {-2, 2, 0,-1}, { 0, 2,-2,-1}, {-2, 2,-2,-1},
        { 1, 1, 1, 2}, {-1, 1, 1, 2}, { 1,-1, 1, 2}, {-1,-1, 1, 2},
        { 1, 1,-1, 2}, {-1, 1,-1, 2}, { 1,-1,-1, 2}, {-1,-1,-1, 2} };

/* D2: XXYY + YYXX */
const short constell_D2[64][4] = {
        { 0, 0, 1, 1}, {-2, 0, 1, 1}, { 0,-2, 1, 1}, {-2,-2, 1, 1},
        { 0, 0,-1, 1}, {-2, 0,-1, 1}, { 0,-2,-1, 1}, {-2,-2,-1, 1},
        { 0, 0, 1,-1}, {-2, 0, 1,-1}, { 0,-2, 1,-1}, {-2,-2, 1,-1},
        { 0, 0,-1,-1}, {-2, 0,-1,-1}, { 0,-2,-1,-1}, {-2,-2,-1,-1},
        { 1, 1, 0, 0}, {-1, 1, 0, 0}, { 1,-1, 0, 0}, {-1,-1, 0, 0},
        { 1, 1,-2, 0}, {-1, 1,-2, 0}, { 1,-1,-2, 0}, {-1,-1,-2, 0},
        { 1, 1, 0,-2}, {-1, 1, 0,-2}, { 1,-1, 0,-2}, {-1,-1, 0,-2},
        { 1, 1,-2,-2}, {-1, 1,-2,-2}, { 1,-1,-2,-2}, {-1,-1,-2,-2},
        { 2, 0, 1, 1}, { 2,-2, 1, 1}, { 2, 0,-1, 1}, { 2,-2,-1, 1},
        { 2, 0, 1,-1}, { 2,-2, 1,-1}, { 2, 0,-1,-1}, { 2,-2,-1,-1},
        { 1, 1, 2, 0}, {-1, 1, 2, 0}, { 1,-1, 2, 0}, {-1,-1, 2, 0},
        { 1, 1, 2,-2}, {-1, 1, 2,-2}, { 1,-1, 2,-2}, {-1,-1, 2,-2},
        { 0, 2, 1, 1}, {-2, 2, 1, 1}, { 0, 2,-1, 1}, {-2, 2,-1, 1},
        { 0, 2, 1,-1}, {-2, 2, 1,-1}, { 0, 2,-1,-1}, {-2, 2,-1,-1},
        { 1, 1, 0, 2}, {-1, 1, 0, 2}, { 1,-1, 0, 2}, {-1,-1, 0, 2},
        { 1, 1,-2, 2}, {-1, 1,-2, 2}, { 1,-1,-2, 2}, {-1,-1,-2, 2} };

/* D3: XXYX + YYXY */
const short constell_D3[64][4] = {
        { 0, 0, 1, 0}, {-2, 0, 1, 0}, { 0,-2, 1, 0}, {-2,-2, 1, 0},
        { 0, 0,-1, 0}, {-2, 0,-1, 0}, { 0,-2,-1, 0}, {-2,-2,-1, 0},
        { 0, 0, 1,-2}, {-2, 0, 1,-2}, { 0,-2, 1,-2}, {-2,-2, 1,-2},
        { 0, 0,-1,-2}, {-2, 0,-1,-2}, { 0,-2,-1,-2}, {-2,-2,-1,-2},
        { 1, 1, 0, 1}, {-1, 1, 0, 1}, { 1,-1, 0, 1}, {-1,-1, 0, 1},
        { 1, 1,-2, 1}, {-1, 1,-2, 1}, { 1,-1,-2, 1}, {-1,-1,-2, 1},
        { 1, 1, 0,-1}, {-1, 1, 0,-1}, { 1,-1, 0,-1}, {-1,-1, 0,-1},
        { 1, 1,-2,-1}, {-1, 1,-2,-1}, { 1,-1,-2,-1}, {-1,-1,-2,-1},
        { 2, 0, 1, 0}, { 2,-2, 1, 0}, { 2, 0,-1, 0}, { 2,-2,-1, 0},
        { 2, 0, 1,-2}, { 2,-2, 1,-2}, { 2, 0,-1,-2}, { 2,-2,-1,-2},
        { 1, 1, 2, 1}, {-1, 1, 2, 1}, { 1,-1, 2, 1}, {-1,-1, 2, 1},
        { 1, 1, 2,-1}, {-1, 1, 2,-1}, { 1,-1, 2,-1}, {-1,-1, 2,-1},
        { 0, 2, 1, 0}, {-2, 2, 1, 0}, { 0, 2,-1, 0}, {-2, 2,-1, 0},
        { 0, 2, 1,-2}, {-2, 2, 1,-2}, { 0, 2,-1,-2}, {-2, 2,-1,-2},
        { 0, 0, 1, 2}, {-2, 0, 1, 2}, { 0,-2, 1, 2}, {-2,-2, 1, 2},
        { 0, 0,-1, 2}, {-2, 0,-1, 2}, { 0,-2,-1, 2}, {-2,-2,-1, 2} };

/* D4: XYYX + YXXY */
const short constell_D4[64][4] = {
        { 0, 1, 1, 0}, {-2, 1, 1, 0}, { 0,-1, 1, 0}, {-2,-1, 1, 0},
        { 0, 1,-1, 0}, {-2, 1,-1, 0}, { 0,-1,-1, 0}, {-2,-1,-1, 0},
        { 0, 1, 1,-2}, {-2, 1, 1,-2}, { 0,-1, 1,-2}, {-2,-1, 1,-2},
        { 0, 1,-1,-2}, {-2, 1,-1,-2}, { 0,-1,-1,-2}, {-2,-1,-1,-2},
        { 1, 0, 0, 1}, {-1, 0, 0, 1}, { 1,-2, 0, 1}, {-1,-2, 0, 1},
        { 1, 0,-2, 1}, {-1, 0,-2, 1}, { 1,-2,-2, 1}, {-1,-2,-2, 1},
        { 1, 0, 0,-1}, {-1, 0, 0,-1}, { 1,-2, 0,-1}, {-1,-2, 0,-1},
        { 1, 0,-2,-1}, {-1, 0,-2,-1}, { 1,-2,-2,-1}, {-1,-2,-2,-1},
        { 2, 1, 1, 0}, { 2,-1, 1, 0}, { 2, 1,-1, 0}, { 2,-1,-1, 0},
        { 2, 1, 1,-2}, { 2,-1, 1,-2}, { 2, 1,-1,-2}, { 2,-1,-1,-2},
        { 1, 0, 2, 1}, {-1, 0, 2, 1}, { 1,-2, 2, 1}, {-1,-2, 2, 1},
        { 1, 0, 2,-1}, {-1, 0, 2,-1}, { 1,-2, 2,-1}, {-1,-2, 2,-1},
        { 1, 2, 0, 1}, {-1, 2, 0, 1}, { 1, 2,-2, 1}, {-1, 2,-2, 1},
        { 1, 2, 0,-1}, {-1, 2, 0,-1}, { 1, 2,-2,-1}, {-1, 2,-2,-1},
        { 0, 1, 1, 2}, {-2, 1, 1, 2}, { 0,-1, 1, 2}, {-2,-1, 1, 2},
        { 0, 1,-1, 2}, {-2, 1,-1, 2}, { 0,-1,-1, 2}, {-2,-1,-1, 2} };

/* D5: XYYY + YXXX */
const short constell_D5[64][4] = {
        { 0, 1, 1, 1}, {-2, 1, 1, 1}, { 0,-1, 1, 1}, {-2,-1, 1, 1},
        { 0, 1,-1, 1}, {-2, 1,-1, 1}, { 0,-1,-1, 1}, {-2,-1,-1, 1},
        { 0, 1, 1,-1}, {-2, 1, 1,-1}, { 0,-1, 1,-1}, {-2,-1, 1,-1},
        { 0, 1,-1,-1}, {-2, 1,-1,-1}, { 0,-1,-1,-1}, {-2,-1,-1,-1},
        { 1, 0, 0, 0}, {-1, 0, 0, 0}, { 1,-2, 0, 0}, {-1,-2, 0, 0},
        { 1, 0,-2, 0}, {-1, 0,-2, 0}, { 1,-2,-2, 0}, {-1,-2,-2, 0},
        { 1, 0, 0,-2}, {-1, 0, 0,-2}, { 1,-2, 0,-2}, {-1,-2, 0,-2},
        { 1, 0,-2,-2}, {-1, 0,-2,-2}, { 1,-2,-2,-2}, {-1,-2,-2,-2},
        { 2, 1, 1, 1}, { 2,-1, 1, 1}, { 2, 1,-1, 1}, { 2,-1,-1, 1},
        { 2, 1, 1,-1}, { 2,-1, 1,-1}, { 2, 1,-1,-1}, { 2,-1,-1,-1},
        { 1, 0, 2, 0}, {-1, 0, 2, 0}, { 1,-2, 2, 0}, {-1,-2, 2, 0},
        { 1, 0, 2,-2}, {-1, 0, 2,-2}, { 1,-2, 2,-2}, {-1,-2, 2,-2},
        { 1, 2, 0, 0}, {-1, 2, 0, 0}, { 1, 2,-2, 0}, {-1, 2,-2, 0},
        { 1, 2, 0,-2}, {-1, 2, 0,-2}, { 1, 2,-2,-2}, {-1, 2,-2,-2},
        { 1, 0, 0, 2}, {-1, 0, 0, 2}, { 1,-2, 0, 2}, {-1,-2, 0, 2},
        { 1, 0,-2, 2}, {-1, 0,-2, 2}, { 1,-2,-2, 2}, {-1,-2,-2, 2} };

/* D6: XYXY + YXYX */
const short constell_D6[64][4] = {
        { 0, 1, 0, 1}, {-2, 1, 0, 1}, { 0,-1, 0, 1}, {-2,-1, 0, 1},
        { 0, 1,-2, 1}, {-2, 1,-2, 1}, { 0,-1,-2, 1}, {-2,-1,-2, 1},
        { 0, 1, 0,-1}, {-2, 1, 0,-1}, { 0,-1, 0,-1}, {-2,-1, 0,-1},
        { 0, 1,-2,-1}, {-2, 1,-2,-1}, { 0,-1,-2,-1}, {-2,-1,-2,-1},
        { 1, 0, 1, 0}, {-1, 0, 1, 0}, { 1,-2, 1, 0}, {-1,-2, 1, 0},
        { 1, 0,-1, 0}, {-1, 0,-1, 0}, { 1,-2,-1, 0}, {-1,-2,-1, 0},
        { 1, 0, 1,-2}, {-1, 0, 1,-2}, { 1,-2, 1,-2}, {-1,-2, 1,-2},
        { 1, 0,-1,-2}, {-1, 0,-1,-2}, { 1,-2,-1,-2}, {-1,-2,-1,-2},
        { 2, 1, 0, 1}, { 2,-1, 0, 1}, { 2, 1,-2, 1}, { 2,-1,-2, 1},
        { 2, 1, 0,-1}, { 2,-1, 0,-1}, { 2, 1,-2,-1}, { 2,-1,-2,-1},
        { 0, 1, 2, 1}, {-2, 1, 2, 1}, { 0,-1, 2, 1}, {-2,-1, 2, 1},
        { 0, 1, 2,-1}, {-2, 1, 2,-1}, { 0,-1, 2,-1}, {-2,-1, 2,-1},
        { 1, 2, 1, 0}, {-1, 2, 1, 0}, { 1, 2,-1, 0}, {-1, 2,-1, 0},
        { 1, 2, 1,-2}, {-1, 2, 1,-2}, { 1, 2,-1,-2}, {-1, 2,-1,-2},
        { 1, 0, 1, 2}, {-1, 0, 1, 2}, { 1,-2, 1, 2}, {-1,-2, 1, 2},
        { 1, 0,-1, 2}, {-1, 0,-1, 2}, { 1,-2,-1, 2}, {-1,-2,-1, 2} };

/* D7: XYXX + YXYY */
const short constell_D7[64][4] = {
        { 0, 1, 0, 0}, {-2, 1, 0, 0}, { 0,-1, 0, 0}, {-2,-1, 0, 0},
        { 0, 1,-2, 0}, {-2, 1,-2, 0}, { 0,-1,-2, 0}, {-2,-1,-2, 0},
        { 0, 1, 0,-2}, {-2, 1, 0,-2}, { 0,-1, 0,-2}, {-2,-1, 0,-2},
        { 0, 1,-2,-2}, {-2, 1,-2,-2}, { 0,-1,-2,-2}, {-2,-1,-2,-2},
        { 1, 0, 1, 1}, {-1, 0, 1, 1}, { 1,-2, 1, 1}, {-1,-2, 1, 1},
        { 1, 0,-1, 1}, {-1, 0,-1, 1}, { 1,-2,-1, 1}, {-1,-2,-1, 1},
        { 1, 0, 1,-1}, {-1, 0, 1,-1}, { 1,-2, 1,-1}, {-1,-2, 1,-1},
        { 1, 0,-1,-1}, {-1, 0,-1,-1}, { 1,-2,-1,-1}, {-1,-2,-1,-1},
        { 2, 1, 0, 0}, { 2,-1, 0, 0}, { 2, 1,-2, 0}, { 2,-1,-2, 0},
        { 2, 1, 0,-2}, { 2,-1, 0,-2}, { 2, 1,-2,-2}, { 2,-1,-2,-2},
        { 0, 1, 2, 0}, {-2, 1, 2, 0}, { 0,-1, 2, 0}, {-2,-1, 2, 0},
        { 0, 1, 2,-2}, {-2, 1, 2,-2}, { 0,-1, 2,-2}, {-2,-1, 2,-2},
        { 1, 2, 1, 1}, {-1, 2, 1, 1}, { 1, 2,-1, 1}, {-1, 2,-1, 1},
        { 1, 2, 1,-1}, {-1, 2, 1,-1}, { 1, 2,-1,-1}, {-1, 2,-1,-1},
        { 0, 1, 0, 2}, {-2, 1, 0, 2}, { 0,-1, 0, 2}, {-2,-1, 0, 2},
        { 0, 1,-2, 2}, {-2, 1,-2, 2}, { 0,-1,-2, 2}, {-2,-1,-2, 2} };


#endif
```

A.1.2 DFE Program

```
/*****************************************************************************************
  PURPOSE: This program evaluates the BER performance of DFE in a
  1000BASE-T system.

  INPUTS:
    RunLength: Number of 4D symbols to be generated.
    SNR: Signal-to-noise ration in dB.

  NOTES: Program reads DFE coefficients from "DFECoeffs.txt"
*****************************************************************************************/
#include <fstream.h>
#include <iostream.h>
#include <math.h>
#include <cstdlib>
#include <iomanip.h>
#include <time.h>
#include "simulation.h"
#include "trellis1000BaseT.h"

/************************************
/* Function Definition */
/************************************/
void  IArrayShift(int A[][NumChannels] ,int A_Length, int * NewCell);
void  FArrayShift(double A[][NumChannels] ,int A_Length, double * NewCell);
double noise( double variance, double u1, double u2 );
double PAM5_Detector(double x);

main(int argc,char* argv[])
{
/************************************
/* Parameters and Variables */
/************************************/
    long RunLength = (int) (atof(argv[1]));
    double SNR = (double) atof(argv[2]);
    //  double LearningConst = (double) atof( argv[3] );
    //  double ForgettingConst= (double) atof( argv[4] );

    double Symb [NumChannels]={0};
    const short bits_per_word=8;
    short buf[bits_per_word+1];
    short Sdn6, Sdn7, Sdn8;
    short Delay0=0, Delay1=0, Delay2=0,Delay0_Temp=0; /* initialize encoder state */
    short point,subset;

    double ChannelBuf[CIR_Length][NumChannels]={0};
    double SumVar[NumChannels]={0};

    double u1, u2;
    double noise_power;
    double avg_point_power = 4*1.8125; // Average Power for 4D-PAM5 Symbols
    // Noise variance to be added to 4D-PAM5 Symbols
    double variance = avg_point_power / exp((SNR/10.0)*log(10.0));
    // Noise variance to be added to 1D-PAM5 Symbols
    variance=variance/4;
    const double N = (double)(RAND_MAX)+1.0;

    time_t seed,Start_Time,End_Time;

    const int Delay= (int) ceil((FFE_Length+FBE_Length-1)/2);
    double SymbOut[NumChannels]={0};
    double Error[NumChannels]={0};

    double FFE_Coeffs [FFE_Length][NumChannels]={0};
    double FBE_Coeffs [FBE_Length][NumChannels]={0};
    double DFE_Coeffs [DFE_Length][NumChannels]={0};
    double FFE_InVector [FFE_Length][NumChannels]={0};
    double FBE_InVector [FBE_Length][NumChannels]={0};
    double DFE_InVector [DFE_Length][NumChannels]={0};
    double Psi [DFE_Length] [FBE_Length][NumChannels]= {0};
    double NPsi [DFE_Length][NumChannels] = {0};

    double    OrgSymbBuffer[FFE_Length][NumChannels]={0};
    int     i,j,k,r,c;
    int Error_Num=0;
    double SER,BER;

    /*************************************
    /* Initialization */
    /*************************************/
    srand( unsigned (time(&seed)));

    // Reading DFE coefficients from DFECoeffs.txt
    ifstream DFECoeffsFile;
    DFECoeffsFile.open("DFECoeffs.txt");

    for (k=0;k<DFE_Length;k++) {

        DFECoeffsFile>>DFE_Coeffs[k][0]>>DFE_Coeffs[k][1]>>DFE_Coeffs[k][2]>>DFE_Coeffs[k][3];

        if (k<FFE_Length){
            FFE_Coeffs[k][0]=DFE_Coeffs[k][0];
            FFE_Coeffs[k][1]=DFE_Coeffs[k][1];
            FFE_Coeffs[k][2]=DFE_Coeffs[k][2];
            FFE_Coeffs[k][3]=DFE_Coeffs[k][3];
        }
        else{
            FBE_Coeffs[k-FFE_Length][0]=DFE_Coeffs[k][0];
            FBE_Coeffs[k-FFE_Length][1]=DFE_Coeffs[k][1];
            FBE_Coeffs[k-FFE_Length][2]=DFE_Coeffs[k][2];
            FBE_Coeffs[k-FFE_Length][3]=DFE_Coeffs[k][3];
        }
    }
    DFECoeffsFile.close();

/****************************** SIMULATION MAIN LOOP ***************************/
    time(&Start_Time);

    for(k=1;k<RunLength+Delay;k++){

        //Random Bit Sequence Generation and word forming

        for (i=0;i<bits_per_word;i++)
            buf[i]=rand() & 01;

        //1000BASE-T Convolutional Encoding

        Sdn6 = buf[ bits_per_word - 2 ];
        Sdn7 = buf[ bits_per_word - 1 ];
        Sdn8 = Delay0;

        /* append the new bit (Sdn8) to the right end of the vector */
        buf[ bits_per_word ]  = Sdn8;

        /* update the encoder state */
        Delay0_Temp=Delay0;

        if (Sdn7==Delay1)
        {Delay0=0; }
        else
        {Delay0=1;}
```

```
if (Sdn6==Delay2)
{Delay1=0; }
else
{Delay1=1;}
Delay2= Delay0_Temp;

//1000BASET 4D Symbol Mapping
point  = 32*buf[5] + 16*buf[4] + 8*buf[3] + 4*buf[2] + 2*buf[1] + buf[0];
subset = 4*buf[6] + 2*buf[7] + buf[8];

switch (subset) {
case 0:    Symb[0] = constell_D0[point][0];
    Symb[1] = constell_D0[point][1];
    Symb[2] = constell_D0[point][2];
    Symb[3] = constell_D0[point][3];
break;

case 1:    Symb[0] = constell_D1[point][0];
    Symb[1] = constell_D1[point][1];
    Symb[2] = constell_D1[point][2];
    Symb[3] = constell_D1[point][3];
break;

case 2:    Symb[0] = constell_D2[point][0];
    Symb[1] = constell_D2[point][1];
    Symb[2] = constell_D2[point][2];
    Symb[3] = constell_D2[point][3];
break;

case 3:    Symb[0] = constell_D3[point][0];
    Symb[1] = constell_D3[point][1];
    Symb[2] = constell_D3[point][2];
    Symb[3] = constell_D3[point][3];
break;

case 4:    Symb[0] = constell_D4[point][0];
    Symb[1] = constell_D4[point][1];
    Symb[2] = constell_D4[point][2];
    Symb[3] = constell_D4[point][3];
break;

case 5:    Symb[0] = constell_D5[point][0];
    Symb[1] = constell_D5[point][1];
    Symb[2] = constell_D5[point][2];
    Symb[3] = constell_D5[point][3];
break;

case 6:    Symb[0] = constell_D6[point][0];
    Symb[1] = constell_D6[point][1];
    Symb[2] = constell_D6[point][2];
    Symb[3] = constell_D6[point][3];
break;

case 7:    Symb[0] = constell_D7[point][0];
    Symb[1] = constell_D7[point][1];
    Symb[2] = constell_D7[point][2];
    Symb[3] = constell_D7[point][3];
break;

} /* end switch */

FArrayShift(OrgSymbBuffer,FFE_Length,Symb);

//Passing symbols through AWGN channel with ISI
```

```
FArrayShift(ChannelBuf,CIR_Length,Symb);

for(c=0;c<NumChannels;c++){
    SumVar[c]=0;
    for (i=0;i<CIR_Length;i++)
    {
        SumVar[c]=ChannelBuf[i][c]*CIR[i]+SumVar[c];
    }
    Symb[c]=SumVar[c];
}

//AWGN
for(c=0;c<NumChannels;c++){
    u1 = double(rand()+1) /N;
    u2 = double(rand()+1) /N;
    Symb[c] = Symb[c] + noise(variance, u1, u2 );
}

//DFE

FArrayShift(FFE_InVector, FFE_Length,Symb);

if(k>Delay){
    for (i=0;i<NumChannels;i++){
        for (j=0;j<DFE_Length;j++){
            if (j<FFE_Length){
                DFE_InVector[j][i]=FFE_InVector[j][i];
            }
            else{
                DFE_InVector[j][i]=FBE_InVector[j-FFE_Length][i];
            }
        }
    }//for i


    for (i=0;i<NumChannels;i++){
        SymbOut[i]=0;
        for (j=0;j<DFE_Length;j++){
            SymbOut[i]=DFE_InVector[j][i]*DFE_Coeffs[j][i]+SymbOut[i];
        }
        Error[i]=PAM5_Detector(SymbOut[i])-SymbOut[i];
        SymbOut[i] =PAM5_Detector(SymbOut[i]);
    }

    for (i=0;i<NumChannels;i++){
        for (j=0;j<DFE_Length;j++){
            NPsi[j][i]= DFE_InVector[j][i];
        }
    }

    for (i=0;i<NumChannels;i++){
        for (r=0;r<DFE_Length;r++){
            for (c=0;c<FBE_Length;c++){
                NPsi[r][i]= NPsi[r][i] + FBE_Coeffs[c][i] * Psi[r][c][i];
            }
        }
    }

    for (i=0;i<NumChannels;i++){
        for (j=0;j<DFE_Length;j++){
          DFE_Coeffs[j][i]=ForgettingConst*DFE_Coeffs[j][i]+LearningConst*Error[i]*NPsi[j][i];
        }
    }

    for (i=0;i<NumChannels;i++){
        for (r=0;r<DFE_Length;r++){
```

```
                    for (c=FBE_Length-2;c>=0;c--){
                        Psi[r][c+1][i]= Psi[r][c][i];
                    }
                }
            }

            for ( i=0;i<NumChannels;i++){
                for ( r=0;r<DFE_Length;r++) {
                    Psi[r][0][ i]=NPsi[r][i];
                }
            }

            for (i=0;i<NumChannels;i++){
                for (j=0;j<DFE_Length;j++){
                    if (j<FFE_Length)
                    {
                        FFE_Coeffs[j][i]=DFE_Coeffs[j][i];
                    }
                    else{
                        FBE_Coeffs[j-FFE_Length][i]=DFE_Coeffs[j][i];
                    }
                }
            }//for i

            FArrayShift(FBE_InVector, FBE_Length,SymbOut);

            for (i=0;i<NumChannels;i++){
                if (PAM5_Detector(SymbOut[i]) != OrgSymbBuffer[Delay][i]){
                    Error_Num++;
                }
            }
        }
    };//for k


    //SER and BER Calculation

    //SER for 1D-PAM5 symbols
    SER= double (Error_Num)/double(k-Delay);
    //On "average", 1.32 bits of 2 bits  in a 1D-PAM5 symbol have error.
    BER= SER*1.32/8;

    time(&End_Time);
    double Elapsed_Time= (End_Time-Start_Time)/60;

    cerr<<endl<<"******************************************"<<endl;
    cerr<<"Number of Symbols: "<<k-Delay<<endl;
    cerr<<"Number of Errors : "<<Error_Num<<endl;
    cerr<<"Symbol Error Rate (SER) for 1D-PAM5: "<<SER<<endl;
    cerr<<"Bit  Error Rate (BER): "<<BER<<endl;
    cerr<<"Simulation Time: "<<Elapsed_Time<<" min."<<endl;
    cerr<<"******************************************"<<endl<<endl;
};//main

/*************************************
/* Functions' Body *
/*************************************/
 void IArrayShift(int A[][NumChannels] ,int A_Length, int * NewCell)
 {
 /*
 PURPOSE:
 Shift-right each row of A and place the NewCell as the first element of A.
 NOTES:
 A should be 2D array of Integer.
 */
     for (int k=0;k<NumChannels;k++){
        for (int j=A_Length-2;j>=0;j--) {
            A[j+1][k]=A[j][k];
        }
        A[0][k]=NewCell[k];
    }
}

void FArrayShift(double A[][NumChannels] ,int A_Length, double * NewCell)
{
/*
   PURPOSE: Shift-right each row of A and place the NewCell as the first element of A.
   NOTES: A should be 2D array of Float.
*/

    for (int k=0;k<NumChannels;k++){
        for (int j=A_Length-2;j>=0;j--) {
            A[j+1][k]=A[j][k];
        }
        A[0][k]=NewCell[k];
    }
}

double noise( double variance, double u1, double u2 )
{
    double pi = 3.14159265358979;
    return sqrt((-2)*variance*log(u1)) * cos(2*pi*u2);
}

double PAM5_Detector(double x)
{
/* PURPOSE:PAM5 Detector (Slicer)*/

    double y=0;
    if (x>1.5)
    {y =+2;}
    else {
        if (x>=0.5)
        {y=1;}
        else {
            if (x>=-0.5)
            {y=0;}
            else
            {
                if(x>=-1.5)
                {y=-1;}
                else
                {y=-2;}
            }
        }
    }
    return(y);
};
```

A.2 Delayed Decision Feedback Sequence Estimator for 1000BASE-T Ethernet

```
/**********************************************************************************
PURPOSE: This program evaluates the BER performance of DDFSE in a
1000BASE−T system.

INPUTS:
  NUM_SYMB: Number of 4D symbols to be generated.
  SNR: Signal−to−noise ration in dB.

NOTES: Program reads DFE coefficients from "DFECoeffs.txt"
**********************************************************************************/
#include <stdlib.h>
#include <fstream.h>
#include <iostream.h>
#include <cmath>
#include <cstdlib>
#include <iomanip.h>
#include <ctime>
#include "simulation.h"
#include "trellis1000BaseT.h"
/***********************************
/* Function Definition */
/***********************************/
void  FArrayShift(double A[][NumChannels] ,int A_Length, double * NewCell);
double noise( double variance, double u1, double u2 );
short  PAM_X_Detector(double a);
short  PAM_Y_Detector(double a);

main(int argc,char* argv[])
{
/***********************************
/* Parameters and Variables */
/***********************************/
    long NUM_SYMB = (int) (atof(argv[1]) + FFE_Length);
    double SNR = (double) atof(argv[2]);

    const short bits_per_word=8;
    short buf[bits_per_word+1];

    static short  Sdn6, Sdn7, Sdn8;
    static short  Delay0=0, Delay1=0, Delay2=0,Delay0_Temp=0; /* initialize encoder state */

    short point, subset;
    static double Symb[NumChannels]={0};
    const int     Delay= (int) ceil((FFE_Length+FBE_Length−1)/2);
    static double OrgSymbBuf[FFE_Length+Trellis_InFrameLength−1][NumChannels]={0};
    static double ChannelBuf[CIR_Length][NumChannels]={0};
    static double SumVar[NumChannels]={0};
    static double u1, u2;
    static double noise_power;

    // Average Power for 4D−PAM5 Symbols
    const double avg_point_power = 4*1.8125;
    double variance = avg_point_power / exp((SNR/10.0)*log(10.0));
    // Noise variance to be added to 4D−PAM5 Symbols
    variance=variance/4;

    const double N = (double) (RAND_MAX)+1.0;

    static double FFE_Coeffs [FFE_Length][NumChannels]={0};
    static double FFE_InVector[FFE_Length][NumChannels]={0};
    static double FBE_Coeffs [FBE_Length][NumChannels]={0};
    static double FBE_InVector[FBE_Length][NumChannels]={0};
    static double SymbTemp [NumChannels]={0};
    static short SymbOut [NumChannels]={0};

    static short QuanSymb1D [NumPaths][NumChannels][NumSubsetTypes] ={0};
    static double Metric1D    [NumPaths][NumChannels][NumSubsetTypes] ={0};
    static double Metric4D    [NumPaths][NumChannels][NumSubsetTypes] ={0};
    static short DecodedPath [NumPaths][Trellis_InFrameLength][CodeDimention]={0};
    static double ISI_Estimate [NumPaths][NumChannels]={0};
    short InBS,InBSubset,X,X0,X1,X2,X3,Y0,Y1,Y2,Y3;
    short min_index,Symb4DPatternTemp,SurvivirPathNum;
    double min,temp,tempx,tempy;

    time_t seed,Start_Time,End_Time;
    srand( unsigned (time(&seed)));

    int i,k=0,t,s,p,b,d,c,j=0,Error=0;
    double SER=0;
    double BER=0;

/*************************************
/* Initialization */
/*************************************/
    // Reading DFE coefficients from DFECoeffs.txt
    ifstream DFECoeffsFile;
    DFECoeffsFile.open("DFECoeffs.txt");

    for (i=0;i<DFE_Length;i++) {
        if (i<FFE_Length)
            DFECoeffsFile>>FFE_Coeffs[i][0]>>FFE_Coeffs[i][1]>>FFE_Coeffs[i][2]>>FFE_Coeffs[i][3];
        else
            DFECoeffsFile>>FBE_Coeffs[i-FFE_Length][0]>>FBE_Coeffs[i-FFE_Length][1]>>
            FBE_Coeffs[i-FFE_Length][2]>>FBE_Coeffs[i-FFE_Length][3];
    }

/***************************** SIMULATION MAIN LOOP ***************************/
    time(&Start_Time);
    while (k<NUM_SYMB){
        k++;
        for (i=0;i<bits_per_word;i++)
            buf[i]=rand() & 01;

        Sdn6 = buf[ bits_per_word - 2 ];
        Sdn7 = buf[ bits_per_word - 1 ];
        Sdn8 = Delay0;

        /* append the new bit (Sdn8) to the right end of the vector */
        buf[ bits_per_word ]  = Sdn8;

        /* update the encoder state */
        Delay0_Temp=Delay0;

        if (Sdn7==Delay1)
        {Delay0=0; }
        else
        {Delay0=1;}
        if (Sdn6==Delay2)
        {Delay1=0; }
        else
        {Delay1=1;}
        Delay2= Delay0_Temp;

        //1000BASET 4D Symbol Mapping

        point  = 32*buf[5] + 16*buf[4] + 8*buf[3] + 4*buf[2] + 2*buf[1] + buf[0];
        subset = 4*buf[6] + 2*buf[7] + buf[8];
```

```
switch (subset) {
case 0:     Symb[0] = constell_D0[point ][0];
     Symb[1] = constell_D0[point ][1];
     Symb[2] = constell_D0[point ][2];
     Symb[3] = constell_D0[point ][3];
break;

case 1:     Symb[0] = constell_D1[point ][0];
     Symb[1] = constell_D1[point ][1];
     Symb[2] = constell_D1[point ][2];
     Symb[3] = constell_D1[point ][3];
break;

case 2:     Symb[0] = constell_D2[point ][0];
     Symb[1] = constell_D2[point ][1];
     Symb[2] = constell_D2[point ][2];
     Symb[3] = constell_D2[point ][3];
break;

case 3:     Symb[0] = constell_D3[point ][0];
     Symb[1] = constell_D3[point ][1];
     Symb[2] = constell_D3[point ][2];
     Symb[3] = constell_D3[point ][3];
break;

case 4:     Symb[0] = constell_D4[point ][0];
     Symb[1] = constell_D4[point ][1];
     Symb[2] = constell_D4[point ][2];
     Symb[3] = constell_D4[point ][3];
break;

case 5:     Symb[0] = constell_D5[point ][0];
     Symb[1] = constell_D5[point ][1];
     Symb[2] = constell_D5[point ][2];
     Symb[3] = constell_D5[point ][3];
break;

case 6:     Symb[0] = constell_D6[point ][0];
     Symb[1] = constell_D6[point ][1];
     Symb[2] = constell_D6[point ][2];
     Symb[3] = constell_D6[point ][3];
break;

case 7:     Symb[0] = constell_D7[point ][0];
     Symb[1] = constell_D7[point ][1];
     Symb[2] = constell_D7[point ][2];
     Symb[3] = constell_D7[point ][3];
break;
} /* end switch */

FArrayShift(OrgSymbBuf,FFE_Length+Trellis_InFrameLength-1,Symb);

//Passing symbols through AWGN channel with ISI
FArrayShift(ChannelBuf,CIR_Length,Symb);

for(c=0;c<NumChannels;c++){
     SumVar[c]=0;
     for (i=0;i<CIR_Length;i++){
          SumVar[c]=ChannelBuf[i][c]*CIR[i]+SumVar[c];
     }
     Symb[c]=SumVar[c];
}

//AWGN
for(c=0;c<NumChannels;c++){
    u1 = (rand()+1) /N;
    u2 = (rand()+1) /N;
    Symb[c] = Symb[c] + noise(variance, u1, u2 );
}

//DDFSE

FArrayShift(FFE_InVector,FFE_Length,Symb);

for (c=0;c<NumChannels;c++)
{
    SumVar[c]=0;
    for (i=0;i<FFE_Length;i++)
    {
        SumVar[c]=FFE_InVector[i][c]*FFE_Coeffs[i][c]+SumVar[c];
    }
    Symb[c]=SumVar[c];
}

if (k>=FFE_Length)
{
    j++;
    for(t=0;t<TrellisDepth-1;t++){
        for(s=0;s<NumCodeStates;s++){
            Trellis [t][s].StateMetric     = Trellis [t+1][s].StateMetric;
            Trellis [t][s].SurvivorBranch = Trellis [t+1][s].SurvivorBranch;
            for(b=0;b<NumInBranches;b++)
                Trellis [t][s].InBranchMetric[b] = Trellis[t+1][s].InBranchMetric[b];
            for(b=0;b<NumInBranches;b++){
                for(d=0;d<CodeDimention;d++)
                    Trellis [t][s].Survivor4DSymbol[b][d]= Trellis[t+1][s].Survivor4DSymbol[b][d];
            }
        }
    }

    for(s=0;s<NumCodeStates;s++){
        Trellis [TrellisDepth-1][s].StateMetric     = 0;
        Trellis [TrellisDepth-1][s].SurvivorBranch = 0;

        for(b=0;b<NumInBranches;b++)
            Trellis [TrellisDepth-1][s].InBranchMetric[b] = 0;
        for(b=0;b<NumInBranches;b++){
            for(d=0;d<CodeDimention;d++)
                Trellis [TrellisDepth-1][s].Survivor4DSymbol[b][d] = 0;
    }   }

    if (j==1){
        Trellis [TrellisDepth -2][0].StateMetric = 0;
        for(s=1;s<NumCodeStates;s++)
        {
            Trellis [TrellisDepth-2][s].StateMetric = 1000;
        }
    }

    // 1D Metric Computation
    for(p=0;p<NumPaths;p++){
        for(c=0;c<NumChannels;c++){
            SymbTemp[c]=Symb[c]+ISI_Estimate[p][c];

            QuanSymb1D[p][c][0]=PAM_X_Detector(SymbTemp[c]);
            QuanSymb1D[p][c][1]=PAM_Y_Detector(SymbTemp[c]);
            Metric1D[p][c][0]=pow(SymbTemp[c]- QuanSymb1D[p][c][0],2);
            Metric1D[p][c][1]=pow(SymbTemp[c]- QuanSymb1D[p][c][1],2);
        }
    }
```

```
// 4D Metric Computation
for(s=0;s<NumCodeStates;s++){
    for(b=0;b<NumInBranches;b++){
        InBS=InBranchState[s][b];
        InBSubset=InBranch4DSubset[s][b];
        X0=Subset4DPattern[InBSubset][0][0];
        X1=Subset4DPattern[InBSubset][0][1];
        X2=Subset4DPattern[InBSubset][0][2];
        X3=Subset4DPattern[InBSubset][0][3];
        Y0=Subset4DPattern[InBSubset][1][0];
        Y1=Subset4DPattern[InBSubset][1][1];
        Y2=Subset4DPattern[InBSubset][1][2];
        Y3=Subset4DPattern[InBSubset][1][3];

        tempx=Metric1D[InBS][0][X0]+ Metric1D[InBS][1][X1]+
                Metric1D[InBS][2][X2]+Metric1D[InBS][3][X3];
        tempy=Metric1D[InBS][0][Y0]+ Metric1D[InBS][1][Y1]+
                Metric1D[InBS][2][Y2]+Metric1D[InBS][3][Y3];

        if (tempx<tempy){
            Metric4D[s][b][0]=tempx;
            Metric4D[s][b][1]=0;
        }
        else {
            Metric4D[s][b][0]=tempy;
            Metric4D[s][b][1]=1;
        }
    }
}//for s

// Calculating code-state metric for the whole trellis except the last stage
if(j>Trellis_InFrameLength){
    for(t=1;t<TrellisDepth-1;t++){
        for(s=0;s<NumCodeStates;s++){
            min=1e300;
            for(b=0;b<NumInBranches;b++){
                temp= Trellis[t-1][InBranchState[s][b]].StateMetric +
                        Trellis [t][s].InBranchMetric[b];
                if (temp<min){
                    min=temp;
                    min_index=b;
                }
            }
            Trellis [t][s].StateMetric=min;
            Trellis [t][s].SurvivorBranch=min_index;
        }
    }
}

//Calculating code-state metrics of the last stage (for current symbol) of the trellis
for(s=0;s<NumCodeStates;s++){
    min=1e300;
    for(b=0;b<NumInBranches;b++){
        Trellis [TrellisDepth-1][s].InBranchMetric[b]=Metric4D[s][b][0];
        temp= Trellis[TrellisDepth-2][InBranchState[s][b]].StateMetric +
                Trellis [TrellisDepth-1][s].InBranchMetric[b];
        if (temp<min){
            min=temp;
            min_index=b;
        }
    }
    Trellis [TrellisDepth-1][s].StateMetric=min;
    Trellis [TrellisDepth-1][s].SurvivorBranch=min_index;

    for(b=0;b<NumInBranches;b++){
        InBS=InBranchState[s][b];
```

```
                InBSubset=InBranch4DSubset[s][b];
                Symb4DPatternTemp=int (Metric4D[s][b][1]);

                for(d=0;d<CodeDimention;d++) {
                    X=Subset4DPattern[InBSubset][Symb4DPatternTemp][d];
                    Trellis [ TrellisDepth-1][s].Survivor4DSymbol[b][d]= QuanSymb1D[InBS][d][X];
                }
            }
        }

        // Updating Paths
        for(p=0;p<NumPaths;p++){
            s=p;
            for(t= TrellisDepth-1;t>0;t--){
                b=Trellis[t][s].SurvivorBranch;
                for(d=0;d<CodeDimention;d++) {
                    DecodedPath[p][t-1][d]=Trellis[t][s].Survivor4DSymbol[b][d];
                }
                s=InBranchState[s][b];
            }
        }

        // Decoding Symbol
        if(j>=Trellis_InFrameLength){
            min=1e300;
            for(s=0;s<NumCodeStates;s++){
                temp= Trellis[TrellisDepth-1][s].StateMetric;
                if (temp<min)
                {
                    min=temp;
                    min_index=s;
                }
            }

            SurvivirPathNum=min_index;

            for(d=0;d<CodeDimention;d++){
                SymbOut[d]=DecodedPath[SurvivirPathNum][0][d];
            }

            for (d=0;d<CodeDimention;d++){
                if (SymbOut[d] != OrgSymbBuf[FFE_Length+Trellis_InFrameLength-2][d])
                {
                    Error++;
                }
            }
        }

        for (p=0;p<NumPaths;p++){
            for (c=0;c<NumChannels;c++){
                for(t=0;t<Trellis_InFrameLength;t++){
                    FBE_InVector[Trellis_InFrameLength-t-1][c]=DecodedPath[p][t][c];
                }
            }

            for (c=0;c<NumChannels;c++){
                temp=0;
                for (t=0;t<FBE_Length;t++)
                {
                    temp=FBE_InVector[t][c]*FBE_Coeffs[t][c]+temp;
                }
                ISI_Estimate[p][c]=temp;
            }
        }//p
    } //if k
};//for k
```

```
    SER= double (Error)/double(CodeDimention*j);
    //On "average", 1.32 bits of 2 bits  in a 1D-PAM5 symbol have error.
    BER= double (Error*1.32)/double(CodeDimention*j*2);

    time(&End_Time);
    cerr<<endl<<"******************************************"<<endl;
    cerr<<"SNR: "<<SNR<<" dB"<<endl;
    cerr<<"Number of Symbols: "<<j<<endl;
    cerr<<"Number of Errors : "<<Error<<endl;
    cerr<<"Symbol Error Rate (SER) for 1D-PAM5: "<<SER<<endl;
    cerr<<"Bit   Error Rate (BER): "<<BER<<endl;
    cerr<<"Simulation Run Time: "<<difftime (End_Time,Start_Time)<<" s"<<endl;
    cerr<<"******************************************"<<endl<<endl;

};//main

/*************************************
/* Functions' body*/
/*************************************/

void FArrayShift(double A[][NumChannels] ,int A_Length, double * NewCell){
/*
PURPOSE: Shift-right each row of A and place the NewCell as the first element of A.
NOTES: A should be 2D array of Float.
*/
    for (int k=0;k<NumChannels;k++){
        for (int j=A_Length-2;j>=0;j--) {
            A[j+1][k]=A[j][k];
        }
        A[0][k]=NewCell[k];
    }
}

double noise( double variance, double u1, double u2){
    double pi = 3.14159265358979;
    return sqrt((-2)*variance*log(u1)) * cos(2*pi*u2);
}


short PAM_X_Detector(double a){
/* PURPOSE: PAM5 Detector for X={-1,+1} subset.*/
    if (a>=0)
        return(1);
    else
        return(-1);
}

short PAM_Y_Detector(double a){
/* PURPOSE: PAM5 Detector for Y={-2,0,+2} subset.*/
    if (a>=1)
        return(2);
    else {
        if (a<=-1)
            return(-2);
        else
            return(0);
    }
}
```

A.3 LDPC Decoder

```
/*************************************************************************************
PURPOSE: LDPC decoder. This program works for both full-rannk and not fullrank code.

INPUTS: A-list file name, BER file name, error limit,maximum iteration, Number of SNR points,
        SNR points, info

NOTE1: This program is orginally porvided by the HCDC labratorty at the University of Alberta
       and then modidied by the author for the purposes of this thesis.

NOTE2: Lch is a long double.Lch values will be Lch values used in LDPCdec, which are
       opposite sign to initially calculated Lch values where Lch=log(p(v=1)/p(v=0))) Prints
       screen output every 100000 frames All-zeros codeword is sent as usual.
*************************************************************************************/

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
#define DISPLAY 0

FILE *filein;
FILE *berfileout;
FILE * ferhistfileout ;
FILE *ERRpos_file;

double MAX_R48= 2147483648.0;

char *filein_name;
char *berfileout_name;

int itermax;
int SNRnum;
float *SNR;
long double *N0;
unsigned short *randseed;

int n;
int m;
int info;
int maxvardegree;
int maxpardegree;
int *vardegree;
int *pardegree;
int *varnode;
int *parnode;
int *varindex;
int *parindex;

int *bits;
int *x;
long double *y;

long *errors;
long *cwerrors;
long *cwdeterrors;
long *cwundeterrors;

long *frameerrors;
long *cwdetframeerrors;
long *cwundetframeerrors;
float *cwdetber;
float *cwundetber;
float *fer;
float *cwdetfer;
float *cwundetfer;

float *cwber;
/*output of LDPC decoder */
long *nerrors;
long *cwerrs;
long *chksum;
int  *iters;

float *ber;
long double *varmsg;
long double *parmsg;

long double *channelmetric;
long double *Lch;

int* itmppointer;
long* ltmppointer;
float* ftmppointer;
double* dtmppointer;
long double *ldtmppointer;

/*
 statistics memory allocation variables initial allocation 1000 memory blocks
expansion of 1000 ok says if expansion was successful
*/

long *fe;
long fe_length=1000;
int  ok_fe;
long fe_count=0;

long *cwdetfe;
long cwdetfe_length=1000;
int  ok_cwdetfe;
long cwdetfe_count=0;

long *cwundetfe;
long cwundetfe_length=1000;
int  ok_cwundetfe;
long cwundetfe_count=0;

int  expansionhist=1000;

int      *ERRpos;
float    *ERRout;
int       ERRok;

clock_t starttime, endtime;
double cpu_time_used;
```

```
/* for termination signal handling */
struct sigaction act, oact;
void termhandler(int sig);

void AWGN(long double *out, int *inp, long double var, int len);
void BPSKmetrics(long double *out, long double *inp,long double mean, long double var, int len);
void probtoLLR(long double *out, long double *inp, int len, int highis);
long double _atanhl(long double x);
void TransposeM(int* Mx, int maxrow, int maxcol);
void hist(long* buffer, long length, char* printtext, FILE* filehist );
void swap(long a[], long i, long j);
int Random(long i, long j);
void quicksort(long a[], long left, long right);

void LDPCdec(
            long *nerrors,
            long *cwerrs,
            long *chksum,
            int * iters ,

            int *varnode,
            int *varindex,
            int *parnode,
            int *parindex,
            long double *Lch,
            int *bits,
            int itermax,
            int L,
            int M,
            int info,
            int maxvardeg,
            int maxpardeg
            );

int main(int argc, char *argv[ ], char *envp[ ]) {

    /* Read Alist == Code*/
    int numclosed;
    long i;
    long j;
    long k;  /*for loop counter*/
    long double rate;
    int errlim;
    int detecterr;
    unsigned short ss[3]={0,0,0};

    if (argc ==1){
        printf("|                                                          |\n");
        printf("|   Error: The program needs input parameters see below: |\n");
        printf("|                                                          |\n");
        printf("|   Help of the Unix/Mac OS version LDPC decoder          |\n");
        printf("|   LDPCdecoder {A list file name}, {BER file name},...   |\n");
        printf("|               {error limit},{maximum iteration},        |\n");
        printf("|               {Number of SNR points},{SNR points}       |\n");
        printf("|               {info}                                    |\n");
        printf("|                                                          |\n");
        printf("|   This simulator is good for error rates up to 1e-12    |\n");
        printf("|   HCDC LAB. Dec. 2004                                   |\n");
        printf("|                                                          |\n");
        return(0);
    }
    else {
        if  ((argv[1]=="h")|| (argv[1]=="H") || (argv[1]=="-h") ||
             (argv[1]=="-H") || (argv[1]=="/h") || (argv[1]=="/H")||
             (argv[1]=="/?")||(argv[1]=="-?") || (argv[1]=="?"))
        {
```

```
        printf("|     Help of the Unix/Mac OS version LDPC decoder       |\n");
        printf("|     LDPCdecoder {A list file name}, {BER file name},... |\n");
        printf("|                {error limit},{maximum iteration},      |\n");
        printf("|                {Number of SNR points},{SNR points},     |\n");
        printf("|                {info}                                   |\n");
        printf("|                                                         |\n");
        printf("|     This simulator is good for error rates up to 1e-12 |\n");
        printf("|     HCDC LAB. Dec. 2004                                 |\n");
        printf("|                                                         |\n");
}

filein_name=argv[1];
printf("A list file              : %s\n",filein_name);
berfileout_name=argv[2];
printf("BER file output to        : %s\n",berfileout_name);
sscanf(argv[3],"%d",&errlim);
printf("Error limit               : %d\n",errlim);
sscanf(argv[4],"%d",&itermax);
printf("Maximum Iteration         : %d\n",itermax);
sscanf( argv[5],"%d",&SNRnum);
printf("Number of SNR points  : %d\n",SNRnum);
SNR = (float *)calloc( SNRnum, sizeof( float ) );
for (i=0;i<SNRnum;i++){
     sscanf( argv[6+i], "%f", (SNR+i) );
     printf("at SNR                    : %f\n",*(SNR+i));
}
/* Input random seed as parameter - is short int array of 3  */
randseed=(unsigned short *)calloc(3, sizeof(unsigned short));
for (i=0;i<3;i++){
     sscanf( argv[6+SNRnum+i], "%hd", (randseed+i));
     printf("random seed               : %hd\n",*(randseed+i));
}
sscanf( argv[9+SNRnum],"%d",&info);
printf("Number of info bits k     : %d\n",info);
/*abort();*/

printf("|     LDPC All zero decoder for regular/irregular code |\n");
printf("|     By Sheryl Howard                                  |\n");
printf("|        Siavash Sheikh Zeinoddin                       |\n");
printf("|        December 2004                                  |\n");
printf("|                                                       |\n");
printf("|     Caution: Using 48 bit random generator            |\n");
printf("|                                                       |\n");
printf("|     This simulator is for error rates up to 1e-12    |\n");
printf("|     HCDC LAB. Dec. 2004                               |\n");
printf("|                                                       |\n");


/* Input seed as parameter */
seed48(randseed);
starttime = clock();

errors=(long *)calloc( SNRnum*itermax, sizeof( long ) );
if( errors == NULL )
     printf( "Can't allocate memory\n" );
cwerrors=(long *)calloc( SNRnum*itermax, sizeof( long ) );
if( cwerrors == NULL )
     printf( "Can't allocate memory\n" );
cwdeterrors=(long *)calloc( SNRnum, sizeof( long ) );
if( cwdeterrors == NULL )
     printf( "Can't allocate memory\n" );
cwundeterrors=(long *)calloc( SNRnum, sizeof( long ) );
if( cwundeterrors == NULL )
     printf( "Can't allocate memory\n" );
ber=(float *)calloc( SNRnum*itermax, sizeof( float ) );
```

```
if( ber == NULL )
     printf( "Can't allocate memory\n" );

cwber=(float *)calloc( SNRnum*itermax, sizeof( float ) );
if( cwber == NULL )
     printf( "Can't allocate memory\n" );

frameerrors=(long *)calloc( SNRnum, sizeof( long ) );
if( frameerrors == NULL )
     printf( "Can't allocate memory\n" );

cwdetframeerrors=(long *)calloc( SNRnum, sizeof( long ) );
if( cwdetframeerrors == NULL )
     printf( "Can't allocate memory\n" );
cwundetframeerrors=(long *)calloc( SNRnum, sizeof( long ) );
if( cwundetframeerrors == NULL )
     printf( "Can't allocate memory\n" );

cwdetber=(float *)calloc( SNRnum, sizeof( float ) );
if( cwdetber == NULL )
     printf( "Can't allocate memory\n" );
cwundetber=(float *)calloc( SNRnum, sizeof( float ) );
if( cwundetber == NULL )
     printf( "Can't allocate memory\n" );

fer=(float *)calloc( itermax, sizeof( float ) );
if( fer == NULL )
     printf( "Can't allocate memory\n" );
cwdetfer=(float *)calloc( itermax, sizeof( float ) );
if( cwdetfer == NULL )
     printf( "Can't allocate memory\n" );
cwundetfer=(float *)calloc( itermax, sizeof( float ) );
if( cwundetfer == NULL )
     printf( "Can't allocate memory\n" );
/*for decoder output*/
nerrors=(long *)calloc( itermax, sizeof( long ) );
if( nerrors == NULL )
     printf( "Can't allocate memory\n" );
cwerrs=(long *)calloc( itermax, sizeof( long ) );
if( cwerrs == NULL )
     printf( "Can't allocate memory\n" );
chksum=(long *)calloc( 1, sizeof( long ) );
if( chksum == NULL )
     printf( "Can't allocate memory\n" );
iters=(int *)calloc( 1, sizeof( int ) );
if( iters == NULL )
     printf( "Can't allocate memory\n" );

/* histogram statisrtic variables*/
fe=(long *)calloc( fe_length, sizeof( long ) );
if( fe == NULL ){
     printf( "Can't allocate memory\n" );
     ok_fe=0;

}else
     ok_fe=1;

cwdetfe=(long *)calloc( cwdetfe_length, sizeof( long ) );
if( cwdetfe == NULL ){
     printf( "Can't allocate memory\n" );
     ok_cwdetfe=0;

}else
     ok_cwdetfe=1;

cwundetfe=(long *)calloc( cwundetfe_length, sizeof( long ) );
```

```
if( cwundetfe == NULL ){
    printf( "Can't allocate memory\n" );
    ok_cwundetfe=0;

}else
    ok_cwundetfe=1;


/* Open for read (will fail if file "data" does not exist) */

if( ( filein   = fopen( filein_name, "r" )) == NULL )
{
    printf( "The Alist file was not opened check the name and existence\n" );
    exit (2);
}
else{
    printf( "The file Alist was opened\n" );
    fscanf( filein ,"%d",&n);
    fscanf( filein ,"%d",&m);
    /*      info=n-m; */
    fscanf( filein ,"%d",&maxvardegree);
    fscanf( filein ,"%d",&maxpardegree);

    vardegree = (int *)calloc( n, sizeof( int ) );
    if( vardegree == NULL )
        printf( "Can't allocate memory\n" );
    itmppointer=vardegree;
    for(i=0;i<n;i++,itmppointer++)
        fscanf( filein ,"%d",itmppointer);

    pardegree = (int *)calloc( m, sizeof( int ) );
    if( pardegree == NULL )
        printf( "Can't allocate memory\n" );
    itmppointer=pardegree;
    for( i=0;i<m;i++,itmppointer++)
        fscanf( filein ,"%d",itmppointer);

    varnode = (int *)calloc( n*maxvardegree, sizeof( int ) );
    if( varnode == NULL )
        printf( "Can't allocate memory\n" );
    itmppointer=varnode;
    for( i=0;i<(n*maxvardegree);i++,itmppointer++)
        fscanf( filein ,"%d",itmppointer);

    parnode = (int *)calloc( m*maxpardegree, sizeof( int ) );
    if( parnode == NULL )
        printf( "Can't allocate memory\n" );
    itmppointer=parnode;
    for( i=0;i<m*maxpardegree;i++,itmppointer++)
        fscanf( filein ,"%d",itmppointer);

    varindex = (int *)calloc( n*maxvardegree, sizeof( int ) );
    if( varindex == NULL )
        printf( "Can't allocate memory\n" );

    parindex = (int *)calloc( m*maxpardegree, sizeof( int ) );
    if( parindex == NULL )
        printf( "Can't allocate memory\n" );

    /* Newly added                                    */
    ERRpos=(int *)calloc( n, sizeof( int ) );
    if( ERRpos == NULL )
        printf( "Can't allocate memory\n" );
    ERRout=(float *)calloc( n*SNRnum, sizeof( float ) );
    if( ERRout == NULL )
        printf( "Can't allocate memory\n" );
        //index matrix for variables

        int varrow;
        long lindtmp;
        for(i=0;i<n;i++){
            varrow=0;
            if (DISPLAY) printf("\n");
            for(j=0;j<maxvardegree;j++)
                if ((*(varnode+i*maxvardegree+j))!=0)
                    varrow++;

                for(k=0;k<varrow;k++){
                    /* all these numbers are indexed one more than C*/
                    lindtmp= (*(varnode+i*maxvardegree+k));

                    for(j=0;j<maxpardegree;j++)
                        if ((*(parnode+(lindtmp-1)*maxpardegree+j))==i+1){
                            *(varindex+i*maxvardegree+k)= j+1;
                            if (DISPLAY) printf("%d ",j+1);
                        }
                }
        }

        for(i=0;i<n;i++){
            if (DISPLAY) printf("\n");
            for(j=0;j<maxvardegree;j++)
                if (DISPLAY) printf("%d ",*(varindex+i*maxvardegree+j));
        }
        /*index matrix for parity nodes*/
        int parrow;

        for(i=0;i<m;i++){
            parrow=0;
            if (DISPLAY) printf("\n");
            for(j=0;j<maxpardegree;j++)
                if ((*(parnode+i*maxpardegree+j))!=0)
                    parrow++;

                for(k=0;k<parrow;k++){
                    /* all these numbers are indexed one more than C*/
                    lindtmp= (*(parnode+i*maxpardegree+k));

                    for(j=0;j<maxvardegree;j++)
                        if ((*(varnode+(lindtmp-1)*maxvardegree+j))==i+1){
                            *(parindex+i*maxpardegree+k)= j+1; /* original matlab index starting from 1*/
                            if (DISPLAY) printf("%d ",j+1);
                        }
                }
        }

        for(i=0;i<m;i++){
            if (DISPLAY) printf("\n");
            for(j=0;j<maxpardegree;j++)
                if (DISPLAY) printf("%d ",*(parindex+i*maxpardegree+j));
        }

}/*else*/

/* initializing message in variable and parity check node*/
varmsg=(long double *)calloc(n*maxvardegree, sizeof( long double ) );
if( varmsg == NULL )
printf( "Can't allocate memory\n" );
parmsg=(long double *)calloc(m*maxpardegree, sizeof( long double ) );
```

```
if( parmsg == NULL )
printf( "Can't allocate memory\n" );

printf("berfilename: %s \n",berfileout_name);

if( ( berfileout   = fopen( berfileout_name, "a" )) == NULL )
{
    printf( "The file 'BERdata' was not opened\n" );
    exit (3);
}
else{
    printf( "\nThe file for Biterrorrate was opened\n" );
}


/* Signal Handling Parameters */
act.sa_handler = termhandler;
sigemptyset(&act.sa_mask);
act. sa_flags =0;
/* Signal Termination Handler */
sigaction (SIGTERM,&act,NULL);

/*Noise parameters*/
rate=(long double )(info)/(long double)n;
N0=(long double *)calloc( SNRnum, sizeof(long double ) );

if( N0 == NULL )
printf( "Can't allocate memory\n" );
else {
                ldtmppointer=N0;
                for(i=0;i<SNRnum;i++,ldtmppointer++)
                    *ldtmppointer=pow(10,(long double)(-1*SNR[i]/10))/rate;
}
/*Encoding*/
/*send all zeros*/
bits=(int *)calloc( n, sizeof( int ) );

if( bits == NULL )
printf( "Can't allocate memory\n" );

/*Transmit as BPSK*/
x=(int *)calloc( n, sizeof( int ) );

if( x == NULL )
printf( "Can't allocate memory\n" );
else {
                itmppointer=x;
                for(i=0;i<n;i++,itmppointer++)
                    *itmppointer=2*(*(bits+i))-1;
}

y=(long double *)calloc( n, sizeof( long double) );

if( y == NULL )
printf( "Can't allocate memory\n" );

channelmetric=(long double *)calloc( 2*n, sizeof( long double) );
if( channelmetric == NULL )
printf( "Can't allocate memory\n" );

Lch=(long double *)calloc( n, sizeof( long double) );
if( Lch == NULL )
printf( "Can't allocate memory\n" );


TransposeM(varnode, n, maxvardegree);
TransposeM(varindex, n, maxvardegree);
TransposeM(parnode, m, maxpardegree);
TransposeM(parindex, m, maxpardegree);


/*************************************************************************
*                                SNR LOOP
*************************************************************************/
int s;
int nloops;

for(s=0;s<SNRnum;s++)
{
    nloops =0; /* Reset number of loops for new SNR */
    fprintf ( berfileout ,"SNR=%4.2f\n",*(SNR+s));

    *(frameerrors+s)=0;
    *(cwdeterrors+s)=0;
    *(cwundeterrors+s)=0;
    *(cwdetber+s)=0.0;
    *(cwundetber+s)=0.0;
    *(cwdetframeerrors+s)=0;
    *(cwundetframeerrors+s)=0;

    memset(fe,0,sizeof(long)*fe_length);
    fe_count=0;
    memset(cwdetfe,0,sizeof(long)*cwdetfe_length);
    cwdetfe_count=0;
    memset(cwundetfe,0,sizeof(long)*cwundetfe_length);
    cwundetfe_count=0;

    while(*(frameerrors+s)<errlim){

        /*send over AWGN channel*/

        AWGN(y, x, *(N0+s)/2, n);
        BPSKmetrics(channelmetric, y, 1, *(N0+s)/2, n);
        probtoLLR(Lch, channelmetric, n, 1); /* 1 negates the Lch so neg LLR means x=1 likely */
        /*message-passing Decoding*/
        LDPCdec(nerrors, cwerrs, chksum, iters,
            varnode, varindex, parnode, parindex,
            Lch, bits, itermax, n, m, info,
            maxvardegree, maxpardegree);
        /*Calculate Error rate*/
        /* nerrors are info bits errors*/
        ERRok=0;
        for(i=0; i<info; i++ ){
            if (abs(*(ERRpos+i))){
                ERRok=1;
                fprintf (ERRpos_file,"%d ",i);
                *(ERRout+s*n+i)= *(ERRout+s*n+i)+(float)(abs(*(ERRpos+i)));
            }
        }
        if (ERRok) {
            for(i=info; i<n; i++ ){
                if (abs(*(ERRpos+i))){
                    fprintf (ERRpos_file,"%d ",i);
                    *(ERRout+s*n+i)= *(ERRout+s*n+i)+(float)(abs(*(ERRpos+i)));
                }
            }
        }
        if (ERRok) fprintf(ERRpos_file,"\n");
        if (ERRok) {
            for(i=0; i<n; i++) {
                fprintf (ERRpos_file,"%Lf ",*(Lch+i));
            }
        fprintf(ERRpos_file,"\n");
    }
    /* check to see if fell out early, possibly from undetected codeword error */
    if ((*iters) < itermax-1)
        for( i=(*iters);i<itermax;i++){
            *(nerrors + i)=*(nerrors+(*iters));
            *(cwerrs + i)=*(cwerrs+(*iters));
        }


        for( i=0;i<itermax;i++) *(errors+s*itermax+i)+=*(nerrors+i);

        nloops++;

        for( i=0;i<itermax;i++) *(ber+s*itermax+i)=
            ((float)(*( errors+s*itermax+i)))/((float)nloops*(float)info);

        /*cwerrs is coded bit errors*/
        detecterr=0;
        if (*(cwerrs+itermax-1)>0) /* error at last iteration */
            if ((*chksum) !=0) {/* error detected*/
                *(cwdeterrors+s)+= (*(cwerrs+itermax-1));
                *(cwdetber+s)= ((float)(*(cwdeterrors+s)))/((float)n*(float)nloops);
            }
            else {/*undetected error*/
                *(cwundeterrors+s)+= (*(cwerrs+itermax-1));
                *(cwundetber+s)= ((float)(*(cwundeterrors+s)))/((float)n*(float)nloops);
            };

            for( i=0;i<itermax;i++){
                *(cwerrors+s*itermax+i)+= (*(cwerrs+i));
                *(cwber+s*itermax+i)=((float)(*(cwerrors+s*itermax+i)))/((float)n*(float)nloops);
            };

            if (*(nerrors+itermax-1)>0) {
                *(frameerrors+s)+=1;

                /*building a hist list processed later which expands itself as memory is needed*/
                if (ok_fe) {
                    *(fe+fe_count)=*(nerrors+itermax-1);
                    fe_count++;
                    if (fe_count==fe_length) {
                        if( (fe =(long *) realloc(fe, ((fe_length+expansionhist)*sizeof( long )) ))== NULL )
                            ok_fe=0;
                        else {
                            ok_fe=1;
                            fe_length +=expansionhist;
                        }
                    }
                }else
                    printf("I am not able to record more histogram data on frame errors");

                printf("Number of loops=%d\n",nloops);
                printf("Number of errors are:\n");
                for (i=0;i<itermax;i++) printf("%7d, ",(*(nerrors+i)));

                printf("\n");
                printf("Total errors at SNR=%6.2f are:\n",*(SNR+s));
                for (i=0;i<itermax;i++) printf("%15d ",(*(errors+s*itermax+i)));
                printf("\n");
                printf("Total frame errors are:  ");
                printf("%15d \n",*(frameerrors+s));
            }

            /*if codeword frame has errors*/
            if (*(cwerrs+itermax-1) >0)
```

```
                    if ((*chksum) !=0){
                        detecterr =1;
                        (*(cwdetframeerrors+s))++;
                        /*build statistics of errors per frame*/

                        if (ok_cwdetfe) {
                            *(cwdetfe+cwdetfe_count)= (*(cwerrs+itermax-1));
                            cwdetfe_count++;
                            if (cwdetfe_count==cwdetfe_length) {
                                if( (fe =(long *) realloc(cwdetfe,
                                    ((cwdetfe_length+expansionhist)*sizeof(long))))== NULL)
                                    ok_cwdetfe=0;
                                else {
                                    ok_cwdetfe=1;
                                    cwdetfe_length+=expansionhist;
                                }
                            }
                        }else
                            printf("I am not able to record more histogram on codeword frame errors");
                    }else {
                        (*(cwundetframeerrors+s))++;

                        if (ok_cwundetfe) {
                            *(cwundetfe+cwundetfe_count)= (*(cwerrs+itermax-1));
                            cwundetfe_count++;
                            if (cwundetfe_count==cwundetfe_length) {
                                if( (fe =(long *) realloc(cwundetfe,
                                    ((cwundetfe_length+expansionhist)*sizeof(long))))== NULL )

                                    ok_cwundetfe=0;
                                else {
                                    ok_cwundetfe=1;
                                    cwundetfe_length+=expansionhist;
                                }
                            }
                        }else
                            printf("I am not able to record more histogram on codeword frame errors");
                    }
                    if ((nloops % 100000)==1) {
                        printf("Number of frames = %d\n",nloops);

                        printf("Total errors:\n");
                        for (i=0;i<itermax;i++) printf(" %d ",(*(errors+s*itermax+i)));

                        printf("\n");
                        printf(" Bit Error rate:\n");
                        for (i=0;i<itermax;i++) printf(" %lf ",(*(ber+s*itermax+i)));

                        printf("\n Frame Errors = %d\n ",*(frameerrors+s));
                    }/* if nloops*/


    }/*while frameerr...*/
    *(fer+s)=(((float)*(frameerrors+s))/(float)nloops);
    *(cwdetfer+s)=(((float)*(cwdetframeerrors+s))/(float)nloops);
    *(cwundetfer+s)=(((float)*(cwundetframeerrors+s))/(float)nloops);


    /* print error count to BER file */
    fprintf( berfileout ,"%d %d %d\n",itermax,info,n);
    fprintf( berfileout ,"%d\n",nloops);
    for (i=0;i<itermax;i++) fprintf(berfileout," %d ",(*(errors+s*itermax+i)));
    fprintf( berfileout ,"\n\n");

    fprintf( berfileout ,"info errorrate= %8.6e, errors=%d, %d iterations\n",*(ber+s*itermax+itermax-1),
            *(errors+s*itermax+itermax-1),itermax);
```

```
}/*for s in SNR*/

free( vardegree );
free( pardegree );
free( errors );
free( cwerrors );
free( cwdeterrors );
free( cwundeterrors );
free( frameerrors );
free( cwdetframeerrors );
free( cwundetframeerrors );
free( cwdetber );
free( cwundetber );
free( ber );
free( varmsg );
free( parmsg );
free( N0 );
free( SNR );
free( bits );
free( x );
free( y );
free( channelmetric );
free( Lch );
free( nerrors );
free( cwerrs );
free( chksum );
free( iters );
free( cwber );
free(ERRout);
free(ERRpos);

/* Close stream */

if( fclose( filein ) )
   printf( "The file 'data 1' was not closed\n" );

/* All other files are closed: */

if ( fclose( berfileout ))
   printf( "The file 'data 2' was not closed\n" );

endtime = clock();
cpu_time_used = ((double) (endtime - starttime)) / CLOCKS_PER_SEC;

printf("\nTime elapsed = %lf\n", cpu_time_used);
}
return(0);
}/* if argc*/

/*********************************************************************/
long double _atanhl(long double x) {
    return(0.5*log((1+x)/(1-x)));

}
/*********************************************************************/
void AWGN(long double *out, int *inp, long double var, int len){
    int i;

    long double *a;
    long double *b;

    a=(long double *)calloc( len, sizeof( long double) );
    if( a == NULL )
                    printf( "Can't allocate memory\n" );
    b=(long double *)calloc( len, sizeof( long double) );
    if( b == NULL )
                    printf( "Can't allocate memory\n" );

    for(i=0;i<len;i++){
        *(a+i)=(long double)lrand48() / (1.0+(long double) MAX_R48);
        *(b+i)=(0.5+(long double)lrand48()) / (1.0+(long double) MAX_R48);
        *(out+i)= (long double) *(inp+i);
        *(out+i)+=(long double)cos(2.0*M_PI* (*(a+i))) *
                    sqrt(-2.0*log((*(b+i))))*(long double)sqrt(var);

    }

    free( a );
    free( b );
}
/**********************************************************************/
void BPSKmetrics(long double *out, long double *inp,long double mean, long double var, int len){
    int i;
    long double tmp1;
    long double tmp2;
    /*for antipodal signalling*/
    for(i=0;i<len;i++){
        tmp1=(long double) exp(pow((*(inp+i)+mean),2)/(-2*var));
        tmp2=(long double) exp(pow((*(inp+i)-mean),2)/(-2*var));
        *(out+2*i)  =tmp1/(sqrt(M_PI*2*var));
        *(out+2*i+1)=tmp2/(sqrt(M_PI*2*var));
    }
}
/**********************************************************************/

void probtoLLR(long double *out, long double *inp, int len, int highis){

    int i;

    for(i=0;i<len;i++){
        if ((*(inp+2*i))!=0.0)
            *(out+i)= *(inp+2*i+1)/(*(inp+2*i));
        else
            *(out+i)=exp(50);

        if ((*(out+i))<= 1e-22)
            *(out+i)=(long double)exp(-50);
        else if ((*(out+i))>=1e22)
            *(out+i)=(long double)exp(50);

        *(out+i)=(long double) -1* highis*log(*(out+i));
    }
}
/**********************************************************************/
void TransposeM(int* Mx, int maxrow, int maxcol){
    int *tempi;
    int j;
    int i;

    tempi = (int *)calloc( maxcol*maxrow, sizeof( int ) );
    if( tempi == NULL ){
                    printf( "Can't allocate memory\n" );
                    exit (1);
    }

    for(j=0;j<maxrow;j++)
        for(i=0;i<maxcol;i++)
            *(tempi+i*maxrow+j)= *(Mx+j*maxcol+i);

        for(i=0;i<maxrow*maxcol;i++)
```

```
                *(Mx+i)= *(tempi+i);

            free(tempi);

}
/**********************************************************************/
void swap(long a[], long i, long j) {
    long tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
/**********************************************************************/
int Randij(long i, long j) {
    return i + rand() % (j-i+1);
}
/**********************************************************************/
void quicksort(long a[], long left, long right) {
    int last = left, i;

    if ( left >= right) return;

    swap(a,left,Randij(left,right));
    for (i = left + 1; i <= right; i++)
        if (a[i] < a[left])
            swap(a,++last,i);
        swap(a,left,last);
        quicksort(a, left, last-1);
        quicksort(a, last+1,right);
}
/***********************************************************************/
void hist(long* buffer, long length, char* printtext, FILE* filename){
/* fe_length=1000 length if the buffer in general
fe_count length used till now which is passed to thehist graph
    */

    long* hist_buffer;
    long max_bin;
    long i;
    /*
    printf("before quicksort\n");
    for(i=0;i<length;i++)
    printf("%d ", *(buffer+i));
    */
    quicksort(buffer, 0, length-1);

    /*  printf("\nAfter quicksort\n");
    for(i=0;i<length;i++)
    printf("%d ", *(buffer+i));
    */

    max_bin= *(buffer+length-1);

     hist_buffer =(long *)calloc( max_bin,sizeof( long double) );
    if(  hist_buffer  == NULL )
                    printf( "Can't allocate memory buffer for histogram\n" );

    for(i=0;i<length;i++)
        if (*(buffer+i)!=0)
            *( hist_buffer +(*(buffer+i))-1)+=1;


        for(i=0;i<max_bin;i++)
            if (*( hist_buffer +i)!=0)
                fprintf (filename, printtext ,(i+1),*( hist_buffer +i));

            free( hist_buffer );
```

```
}
/*********************************************************************/
void LDPCdec(
          long *nerrors,
          long *cwerrs,
          long *chksum,
          int *iters,

          int *varnode,
          int *varindex,
          int *parnode,
          int *parindex,
          long double *Lch,
          int *bits,
          int itermax,
          int L,
          int M,
          int info,
          int maxvardeg,
          int maxpardeg)
{

   int     iter;               /* iteration counter */
   int     iterfinal ;         /* final iteration value when all parity checks=0 */
   int     var;                /* variable node counter */
   int     par;                /* parity check node counter */

   int     j,i;                   /* edge connection (message) counter */
   int     Hrow;               /* parity check H row counter */
   int     xhat;               /* LDPC decoded codeword bit (hard decision) */
   int     diff;               /* difference between xhat and bits */
   int     checksumall;        /* sum of all parity check sums */
   int     varrow;             /* row pos in parmsg matrix to variable node var */
   int     varcol;             /* col pos in parmsg matrix to variable node var */
   int     parcol;             /* col pos in varmsg matrix to parity check node par */

   long *checksum;           /* vector of parity check sums */
   int *parrow;              /* row pos in varmsg matrix to parity node par */
   long *errors;              /* number of info errors in LDPC decoding hard decision */
   long *cwerrors;           /* number of cw bit errors in LDPC decoding hard decision */

   long double LLRtanh;            /* prod(tanh(LLR msgs/2)) at parity nodes */
   long double extrinsic;       /* divide LLRtanh by tanh(msg/2) to get extrinsic */


   long double *varnodemsg;    /* parity msgs to each variable node */
   long double *LLRx;         /* sum of LLR messages at variable nodes */
   long double *parmsg;            /* LLR messages out of parity check nodes */
   long double *parnodemsg;   /* variable msgs to each parity node */
   long double *varmsg;        /* LLR messages into parity check nodes */

   /* Computational Section ---------- */

   /* Initialize with zeros */
   varnodemsg = (long double *)calloc(maxvardeg,sizeof(long double));
   varmsg = (long double *)calloc(L*maxvardeg,sizeof(long double));
   LLRx = (long double *)calloc(L,sizeof(long double));
   parrow = (int *)calloc(maxpardeg,sizeof(int));
   parnodemsg = (long double *)calloc(maxpardeg,sizeof(long double));
   parmsg = (long double *)calloc(M*maxpardeg,sizeof(long double));
   checksum = (long *)calloc(M,sizeof(long));
   errors  = (long *)calloc(itermax,sizeof(long));
   cwerrors  = (long *)calloc(itermax,sizeof(long));
    iterfinal =itermax; /* initialize to itermax */
```

```
/* Message-Passing Decoding Iterative Loop */
for (iter=0; iter<itermax; iter++) { /* iterate for itermax loops */

    /* Variable Node Processing: Sum LLR messages */
    for (var=0; var<L; var++) LLRx[var] = Lch[var];

    for (var=0; var<L; var++) { /* each variable node */
        for (j=0; j<maxvardeg; j++) { /* each edge connection */
            varrow=varnode[j*L + var]; /* row in parmsg, 1 to M */
            varcol=varindex[j*L + var]; /* col in parmsg, 1 to maxpardeg */
            /*printf("\n %d %d", varrow, varcol);*/

            if (varrow) { /* 0 denotes no edge connection for that degree */
                varnodemsg[j]=parmsg[(varcol-1)*M + varrow-1];
            } else {
                varnodemsg[j]=0.;
            }
            LLRx[var]+=varnodemsg[j]; /* sum all msgs to variable node var */
        }
        for (j=0; j<maxvardeg; j++) { /* each edge connection */
            varrow=varnode[j*L + var]; /* row in parmsg, 1 to M */
            /* subtract off same edge to get extrinsic msg from var to parity chk varrow */
            if (varrow) varmsg[j*L + var]=LLRx[var]-varnodemsg[j];
        }
    } /* end of variable node processing loop */

    /* Hard Decision and Errors */

    memset(checksum,0,sizeof(long)*M);

    xhat=0;
    for (var=0; var<L; var++) { /* each variable node */
        if (LLRx[var]<=0) xhat=1; /* negative LLR means a 1 is most likely */
        if (LLRx[var]>0) xhat=0; /* positive LLR means a 0 is most likely */
        /* check parity check sum */
        for (j=0; j<maxvardeg; j++) {
            if (varnode[j*L + var]) {
                Hrow=varnode[j*L + var];
                checksum[Hrow-1]+=xhat;
            }
        }
        /*  if ((var<5)||(var>L-5)) printf("checksum[1]=%d\n",checksum[0]); */
        /* check for errors */
        diff=xhat-bits[var];

        /* (*(ERRpos+var))=diff; */
        /*printf("%d ",diff);*/
        if ( diff<0) diff=-diff;

        *(ERRpos+var)=diff;

        if (var<info) errors[iter]+=diff; /* info bit errors - systematic code */
        cwerrors[iter]+=diff;
    }


    /* see if all parity check sums are satisfied or we have undetected err */
    checksumall=0;
    /* take modulo 2 of checksum */
    for (Hrow=0; Hrow<M; Hrow++) checksum[Hrow]%=2;
    /* then add together; if checksum ==0, all checks are satisfied */
    for (Hrow=0; Hrow<M; Hrow++) checksumall+=checksum[Hrow];
    /*  printf("total check sum=%d\n",checksumall); */
    /* if all parity check sums are satisfied, stop decoding now */
    if (checksumall==0) iterfinal=iter;
```

```
        if (checksumall==0) break;

        /* Parity Check Node Processing: full tanh processing */

        for (par=0; par<M; par++) { /* each parity check node */
            LLRtanh=1.0;
            for (j=0; j<maxpardeg; j++) { /* each edge connection */
                parrow[j]=parnode[j*M + par]; /* row in varmsg, 1 to L */
                parcol=parindex[j*M + par]; /* col in varmsg, 1 to maxvardeg */
                if (parrow[j]>0)  {  /* 0 denotes no edge connection for that degree */
                    parnodemsg[j]=varmsg[(parcol-1)*L + parrow[j]-1];
                    /* prod(tanh(all msg to parity node par)) */
                    LLRtanh*=tanh(parnodemsg[j]/2.0);
                }
            }
            for (j=0; j<maxpardeg; j++) { /* each edge connection */
                if (parrow[j]>0) {
                    /* divide out same edge to get extrinsic msg from parity to variable */
                    extrinsic =LLRtanh/tanh(parnodemsg[j]/2.0);
                    if ( extrinsic  > 0.9999)
                        parmsg[j*M + par]=10.0;
                    else if ( extrinsic  < -0.9999)
                        parmsg[j*M + par]=-10.0;
                    else parmsg[j*M + par]=2.0* atanh(extrinsic);
                }
            }
        } /* end of parity check node processing loop */

    }  /* end of iteration  loop */


    /* Output */

    for ( iter =0; iter<itermax; iter++) {
        nerrors[ iter ]  = errors[ iter ];
        cwerrs[ iter ]   = cwerrors[ iter ];
    }
    *chksum=checksumall;
    /*     printf("parity check sums satisfied at iteration %d\n",iterfinal); */
    * iters = iterfinal ;

    free (varnodemsg);
    free (varmsg);
    free (LLRx);
    free (parrow);
    free (parnodemsg);
    free (parmsg);
    free (checksum);
    free ( errors );
    free (cwerrors);
}

/*   Signal Handling Routine */
void termhandler(int sig)  {
    fflush (NULL);
    exit (0);
}
```

A.4 ISI Channel Model

```
/***********************************************************************************
PURPOSE: The ISI function applies the channel ISI (as an FIR filter) to the input sequnce.

 INPUTS and OUTPUTS:
 out: Output sequence
 inp: Input sequence
 len: sequence length
/***********************************************************************************/

/**************
The following command lines should be placed before the "main" function (i.e. as GLOBAL variables etc)
/**************/
#include <fstream.h>
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>

// Channel Impulse Response (CIR)
#define     CIR_Length 30
long double CIR[CIR_Length]={0};

// Reading CIR coefficients from CIRCoeffs.txt
ifstream CIRCoeffsFile;
CIRCoeffsFile.open("CIRCoeffs.txt");
for (int k=0;k<CIR_Length;k++){
    CIRCoeffsFile>>CIR[k];
}
CIRCoeffsFile.close();

/***********************************************************************************/
void ISI(long double *out, long double *inp, int len)
{
    void FArrayShift(long double A[],int A_Length, long double NewCell);

    long double ChannelBuf[CIR_Length]={0};
    long double S;

    for(int i=0;i<len;i++){

        FArrayShift(ChannelBuf,CIR_Length,*(inp+i));

        S=0;
        for (int j=0;j<CIR_Length;j++){
            S += (long double) ChannelBuf[j]*CIR[j];
        }
        *(out+i)=S;
    }
}
/***********************************************************************************/
void FArrayShift(long double A[],int A_Length, long double NewCell)
{
    /* PURPOSE: Shifts each row of A and places the NewCell as the first element of A.*/
    for (int j=A_Length-2;j>=0;j--){
        A[j+1]=A[j];
    }
    A[0]=NewCell;
}
```

A.5 ACGN Generator

```
/**************************************************************************************
PURPOSE: This function generates ACGN sequence and add it to the input sequence.

INPUTS and OUTPUTS:
  out: Output sequence
  inp: Input sequence
  var: ACGN variance
  len: sequence length
  a  : gain factor in coloration  filter
  b  : coloration  filter  coefficient
**************************************************************************************/
void ACGN(long double *out, int *inp, long double var, int len, long double a, long double b)
{
    int i;
    long double *p;
    long double *q;

    p=(long double *)calloc( len, sizeof( long double) );
    if( p == NULL )
                printf( "Can't allocate memory\n" );

    q=(long double *)calloc( len, sizeof( long double) );
    if( q == NULL )
                printf( "Can't allocate memory\n" );

    for(i=0;i<len;i++){
        *(p+i)=(long double)lrand48() / (1.0+(long double) MAX_R48);
        *(q+i)=(0.5+(long double)lrand48()) / (1.0+(long double) MAX_R48);
        *(out+i)=(long double)cos(2.0*M_PI* (*(p+i))) * sqrt(-2.0*log((*(q+i))))*(long double)sqrt(var);
    }

    *(out+0)=(long double) a * (*(out+0));
    for(i=1;i<len;i++){
        *(out+i) =(long double) a * (*(out+i)) - (long double) b * (*(out+i-1));
    }

    for(i=0;i<len;i++){
        *(out+i)+= (long double) *(inp+i);
    }

    free( p );
    free( q );
}
```

A.6 $1/f$ Noise Generator

```
/***********************************************************************************
PURPOSE: This PINK function generates 1/f (pink) noise sequence and add it to the
input sequence.

INPUTS and OUTPUTS:
  out: Output sequence
  inp: Input sequence
  var: 1/f variance
  len: sequence length
/***********************************************************************************/

/*************
 The following command lines should be placed before the "main" function (i.e. as GLOBAL variables etc)
/*************/

#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

double MAX_R48= 2147483648.0;

// Calculating coloration  filter  coefficients

#define     Coloration_Filter_Length 30
long double Coloration_Filter[Coloration_Filter_Length]={0};
long double Buf[Coloration_Filter_Length]={0};
long double alpha=1;
long double T=1;
for (int j=1;j<Coloration_Filter_Length;j++)
{
    T = (long double) ((long double)j-1-(alpha/2))*T/(long double)j;
    Coloration_Filter[j-1]=T;
}

/***********************************************************************************/
void PINK(long double *out, long double *inp, long double var, int len)
{
    void FArrayShift(long double A[],int A_Length, long double NewCell);
    long double gas_dev(long double noise_var);

    long double w;
    long double pink_noise_sample;
    long double SumTemp;

    for(int i=0;i<len;i++){
        w=gas_dev(1);

        SumTemp=0;
        for (int j=0;j<Coloration_Filter_Length;j++){
            SumTemp += (long double) Buf[j]*Coloration_Filter[j];
        }
        pink_noise_sample=w-SumTemp;
        FArrayShift(Buf,Coloration_Filter_Length,pink_noise_sample);

        // 3.2 is the output variance of coloration  filter , sqrt(1/3.2)=0.65653216429861
        pink_noise_sample= pink_noise_sample * (long double) sqrt(var) * 0.65653216429861;
        *(out+i)= *(inp+i) + pink_noise_sample;
    }

}
/*************************************************************************************/
void FArrayShift(long double A[],int A_Length, long double NewCell)
{
/* PURPOSE: Shifts each row of A and places the NewCell as the first element of A.*/
    for (int j=A_Length-2;j>=0;j--){
        A[j+1]=A[j];
    }
    A[0]=NewCell;
}
/*************************************************************************************/
long double gas_dev(long double noise_var)
{
/* PURPOSE: Generates white Gaussian noise.*/

    int i;
    long double a,b,out;

    a  = (long double)lrand48() / (1.0+(long double) MAX_R48);
    b  = (0.5+(long double)lrand48()) / (1.0+(long double) MAX_R48);
    out= (long double)cos(2.0*M_PI*a) * sqrt(-2.0*log(b))*(long double)sqrt(noise_var);
    return(out);
}
```