NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DIRECT-SEQUENCE SPREAD-SPECTRUM MODULATION FOR UTILITY PACKET TRANSMISSION IN UNDERWATER ACOUSTIC COMMUNICATION NETWORKS

by

Peter Smith Duke

September 2002

Thesis Advisor:	Roberto Cristi
Co-Advisor:	Joseph Rice

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Direct-sequence Spread-Spectrum Modulation for Utility Packet Transmission in Underwater Acoustic Communication Networks			**5. FUNDING NUMBERS**
6. AUTHOR(S) Peter Duke			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis investigates the feasibility and performance of using Direct-Sequence Spread-Spectrum (DSSS) modulation for utility-packet transmission in Seaweb underwater wireless acoustic communications networks. Seaweb networks require robust channel-tolerant utility packets having a low probability of detection (LPD) and allowing for multi-user access. MATLAB code simulated the DSSS transmitter and receiver structures and a modeled channel impulse response represented the underwater environment. The specific modulation scheme implemented is direct-sequence, differentially encoded binary phase-shift keying (DS-DBPSK) with quadrature spreading. Performance is examined using Monte Carlo simulation. Bit error rates and packet error rates for various signal-to-noise ratios and channel conditions are presented and the use of a RAKE receiver, forward error-correction coding and symbol interleaving are examined for improving system performance.

14. SUBJECT TERMS Acoustic Communications, Underwater Communications, Underwater Networks, Undersea Warfare, Direct-Sequence Spread-Spectrum, Differential Binary Phase-Shift Keying, DSSS, DBPSK, DS/BPSK			**15. NUMBER OF PAGES** 151
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

DIRECT-SEQUENCE SPREAD-SPECTRUM MODULATION FOR UTILITY PACKET TRANSMISSION IN UNDERWATER ACOUSTIC COMMUNICATION NETWORKS

Peter S. Duke
Lieutenant Commander, Canadian Navy
B.S., University of Calgary, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author: Peter Duke

Approved by: Roberto Cristi
Thesis Advisor

Joseph Rice
Co-Advisor

John Powers
Chairman, Electrical and Computer
Engineering Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates the feasibility and performance of using Direct-Sequence Spread-Spectrum (DSSS) modulation for utility-packet transmission in Seaweb underwater wireless acoustic communications networks. Seaweb networks require robust channel-tolerant utility packets having a low probability of detection (LPD) and allowing for multi-user access. MATLAB code simulated the DSSS transmitter and receiver structures and a modeled channel impulse response represented the underwater environment. The specific modulation scheme implemented is direct-sequence, differentially encoded binary phase-shift keying (DS-DBPSK) with quadrature spreading. Performance is examined using Monte Carlo simulation. Bit error rates and packet error rates for various signal-to-noise ratios and channel conditions are presented and the use of a RAKE receiver, forward error-correction coding and symbol interleaving are examined for improving system performance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am very grateful to my wonderful wife Patsy and my children Alexander, Charlie and Kathryn for their loving support and the many sacrifices they made throughout my studies.

I would like to thank my advisors, Dr. Roberto Cristi and Joseph Rice for their guidance and patience and the many others who gave their advice and input into many aspects of this study. These include Mike Porter, Dale Green and Ethem Sozer.

Most importantly I thank God for his faithful provision throughout my life.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Underwater acoustic communications have a wide variety of applications in undersea warfare. These applications include wireless networked sensor telemetry in littoral areas, controlling of minefields and networking of surface vessels, submarines, Unmanned Underwater Vehicles (UUVs) and divers. The underwater communications channel is impaired by its band-limited nature and multipath propagation. Direct-Sequence Spread-Spectrum (DSSS) has proven to be very effective in radio frequency (RF) wireless networks, which experience similar difficulties. If the same robust performance can be achieved in the underwater acoustic channel, then the future design and development of underwater networks can incorporate many of the techniques that have proven so successful in RF wireless networks.

This thesis investigates the feasibility and performance of using DSSS modulation for utility packet transmission in Seaweb, an underwater wireless acoustic communications network. The Seaweb network requires robust channel-tolerant utility packets having a low probability of detection (LPD) and allowing for multi-user access. The transmitter and receiver structures are simulated in MATLAB and the underwater environment are represented by a modeled channel impulse response. The specific modulation scheme implemented is direct-sequence, differentially encoded binary phase-shift keying (DS-DBPSK) with quadrature spreading. The performance of DSSS modulation is examined using Monte Carlo simulation. Bit error rates and packet error rates for various signal-to-noise ratios and channel conditions are presented and the use of a RAKE receiver and forward error-correction coding are examined for improving system performance.

The results show that DSSS modulation is well suited for communications in the underwater environment. Bit error rates below 10^{-5} are achievable using soft decision Viterbi decoding and a RAKE receiver that adaptively places the RAKE taps to coincide with the multipath arrivals. The limitations in the modem's performance are associated with poor acquisition algorithm performance at low signal-to-noise ratios.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The primary focus of this research is to determine the feasibility and performance of using Direct-Sequence Spread-Spectrum (DSSS) as the modulation scheme for utility packet transmission in Seaweb. Seaweb is an experimental underwater acoustic communication network being developed by the Space and Naval Warfare Systems Center, San Diego [1]. The utility packets in the network are fixed-length, 72-bit sequences. They serve many functions in the network including initialization, probing the channel and controlling data transfer between nodes in the network. At present the utility packets, like the data packets, use M-ary Frequency-Shift Keying (MFSK) as their modulation scheme. For military communications applications, MFSK has two significant drawbacks in that it is easily detectable by unauthorized intercept receivers and requires time-division or frequency-division multi-access techniques. In the next evolution of Seaweb, the requirement is to have a utility packet that is highly reliable and channel tolerant, but also has a low probability of detection (LPD), low probability of intercept (LPI) and allows for code division multiple access. Spread-spectrum techniques naturally lend themselves to these requirements. Therefore either Frequency Hopped Spread-Spectrum (FHSS) or DSSS are candidate modulation schemes. Our research is confined to the design and analysis of a DSSS modem.

B. SEAWEB UTILITY PACKET REQUIREMENTS

As already mentioned, the utility packets are fixed-length sequences of 72 bits. The present acoustic bandwidth available to the Seaweb network is 9 to 14 kHz. A DSSS transmission is to be used to meet the LPD signaling requirements, which means that the chip rates and therefore the data rates are low. However, bit rates on the order of 10 to 100 bits per second are deemed acceptable. A ½-rate convolutional coder of constraint length 9 and soft decision de-coding with 4-bit quantization is desired. Communication ranges should be between three to five kilometers. Having bit error rates (BERs) less

than 10^{-5} is also desirable and is consistent with acceptable standards in RF communications.

C. GOALS AND METHODOLOGY

The goal of this thesis is to develop, in software, a DSSS modem, suitable for underwater acoustic communications and to examine its performance through Monte Carlo simulation. The experimental Seaweb modem presented here is only for utility packet transmissions and is not yet in use in the Seaweb system. The transmitter and receiver structures are implemented in MATLAB. The receiver software is designed to process the received signal bit by bit, just as would be done in a future hardware modem. The channel is modeled first as an ideal additive white Gaussian noise (AWGN) channel in order to compare simulation results to well-known theoretical performance of DSSS in such a channel. Next, an impulse response representative of an underwater acoustic channel is modeled to examine the system's performance in a multipath environment. BERs for various signal-to-noise ratios (SNRs) are measured for Direct-Sequence Differential Binary Phase-Shift Keying with quadrature spreading (DS-IQ-DBPSK). Lastly, two techniques for improving system performance are examined. These are convolutional error-correction coding and multipath diversity reception using a RAKE receiver.

D. BENEFITS OF THE STUDY

Underwater networks employing wireless acoustic communications have a wide variety of applications in undersea warfare. These applications include networked sensor telemetry in littoral areas, controlling of minefields and networking of surface vessels, submarines, Unmanned Underwater Vehicles (UUVs) and divers. With the drive to implement network centric warfare in all environments, reliable wireless underwater communication is important to the Navy. DSSS has proven to be very effective in radio frequency (RF) wireless networks and is therefore widely used. If the same robust performance can be achieved in the underwater acoustic channel, then the future design

and development of underwater networks can incorporate many of the techniques that have proven so successful in RF wireless networks.

E. THESIS ORGANIZATION

This thesis is organized into five remaining chapters. Chapter II discusses the characteristics of the underwater acoustic communication channel and addresses why it is arguably the most challenging communications medium in the battle space. Chapter III develops the theory behind DS-IQ-BPSK and the advantages that DSSS offers to communications in the undersea environment. This chapter also describes how the transmitter and receiver structures are implemented in this application. Chapter IV details the specific design parameters used in this Seaweb modem design. Chapter V examines both the theoretical and modeled performance of the modem in an ideal additive white Gaussian noise channel and a modeled channel. Finally, Chapter VI reviews the important results and concludes with recommendations for future study. Future work includes investigating improved signal processing algorithms in the receiver, comparing simulated and experimental performance, eventually implementing of the modem in hardware and integrating into the next generation of Seaweb modems.

THIS PAGE INTENTIONALLY LEFT BLANK

II. UNDERWATER ACOUSTIC COMMUNICATION CHANNEL CHARACTERISTICS

This chapter discusses the characteristics of the underwater acoustic channel and the difficulties it imposes on communications signals. Unlike radio frequency (RF) communications, which use electromagnetic waves, underwater communications use acoustic pressure waves to propagate signals through the medium. The challenges that this imposes ensue from three primary impairments. First, the channel is severely band-limited. High frequencies are strongly attenuated in the ocean, which result in relatively small transmission bandwidths and relatively low data rates compared to those achievable in RF communications. Second, ocean noise is non-Gaussian and results from numerous mechanisms including weather, surface wave action, biologics, shipping and industrial noise near the coastline. Lastly, severe fading occurs as a result of destructive interference. Reflections off the sea surface and the sea bottom, as well as scattering from inhomogeneities in the water column, result in multiple arrivals of the signal at the receiver. These multiple arrivals superimpose on each other and distort the signal in amplitude and phase. Likewise, the motion of the source, the receiver and the medium itself result in Doppler shifts and Doppler spreading, which further distort the signal. Figure 1 illustrates some of these time-varying processes.

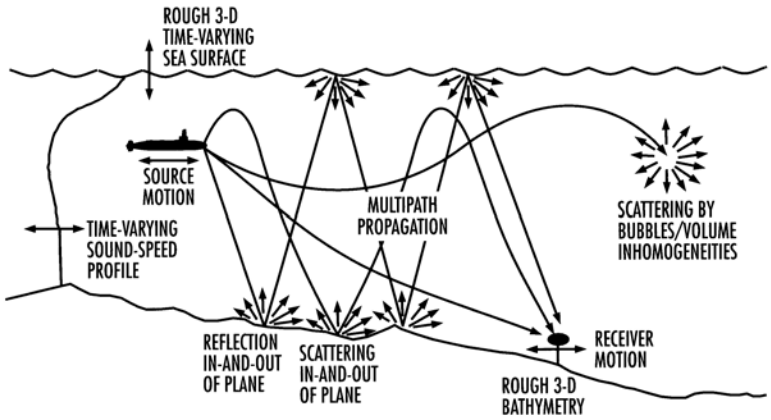

Figure 1. Some of the major processes affecting fading in the underwater acoustic communications channel [From Ref. 2].

A. BAND-LIMITED CHANNEL

The bandwidth of the underwater acoustic communication channel is severely restricted by the transmission loss (i.e., large-scale fading). The two dominant factors affecting transmission loss are spreading and attenuation [3]. Spreading is a geometric effect caused when the intensity of the sound field weakens as the field expands over a larger volume. Transmission loss due to spreading is proportional to $1/R^2$ for spherical spreading, or $1/R$ for cylindrical spreading where R is the range between the source and receiver. In shallow water channels, spreading behavior is typically between spherical and cylindrical and is here assumed to be $1/R^{3/2}$.

Attenuation involves the absorption and scattering of the sound wave as it propagates. Absorption is the most significant mechanism of the two and involves conversion of the acoustic energy into heat. Scattering occurs from the sea surface, ocean bottom and from inhomogeneities in the volume of the ocean. Scattering is generally a hindrance to propagation because scattering increases the attenuation of the signal due to poor directionality. In practice however, distinguishing between the two effects is impossible and therefore they are combined into one term. Figure 2 shows the strong dependence of attenuation with frequency and this dependence is divided into four regions.

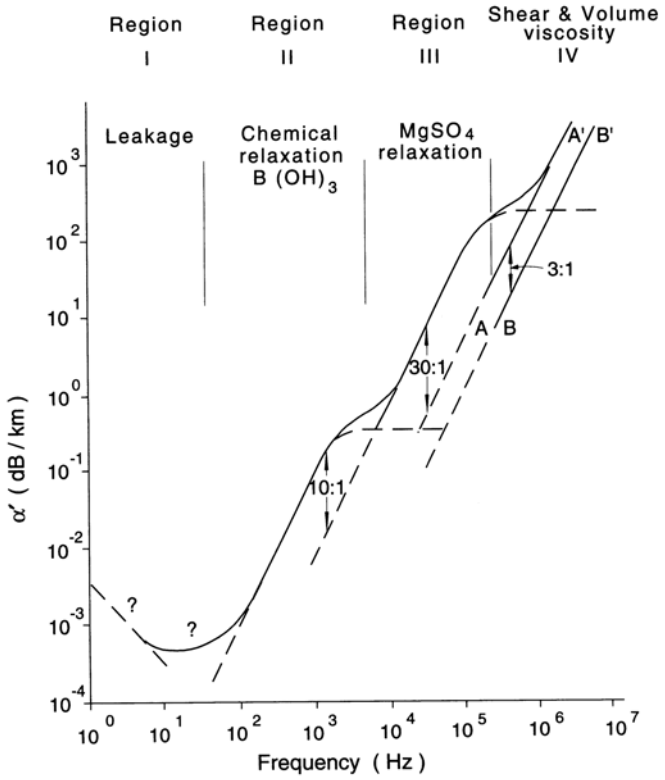

Figure 2. Attenuation coefficient for acoustic energy in seawater [From Ref. 4].

Regions II and III are dominated by the chemical relaxation of two constituents in seawater. The viscosity of the seawater dominates Region IV. The effects of Region I are not well understood.

An analytic expression for the attenuation coefficient can also be given as:

$$\alpha \simeq 3.3\times10^{-3} + \frac{0.11 f^2}{1+f^2} + \frac{44 f^2}{4{,}100+f^2} + 3.0\times10^{-4} f^2 \qquad (2.1)$$

where f is the frequency in kHz and α is in dB/km [4] and the four terms are sequentially associated with the four regions in Figure 2. With both spreading and absorption, the overall transmission loss (TL) expression is given by:

$$TL = 15\log R + \alpha R\times10^{-3} \qquad (2.2)$$

where R is in km and TL is in decibels referenced to 1 micro-Pascal (dB re 1 μPa) [3].

B. NON-GAUSSIAN NOISE CHANNEL

Noise in the underwater communication channel is non-Gaussian. Different sources dominate in different bands and noise levels vary greatly over time and geographic location. Therefore, developing good statistical representations of the noise is difficult. Experimental observations [3] show that at the lower frequencies (below 10 Hz) ambient noise is dominated by ocean turbulence. Noise between 50 Hz to 500 Hz is dominated by distant shipping and depends on the geographic location. At higher frequencies, 500 Hz to 50 kHz, the roughness of the sea surface dominates the noise spectrum. Sea surface roughness is directly related to the wind speeds at the sea surface and is therefore weather dependent. Lastly, at high frequencies above 50 kHz, the thermal noise, due to the motion of the molecules of the sea itself, is the dominant source of ambient noise. General expressions for ambient noise levels in the deep sea may be expressed as:

$$NL_1 = 17 - 30\log f \qquad \textit{Turblence Noise} \qquad (2.3)$$

$$NL_2 = 40 + 20(D - 0.5) + 26\log f - 60\log(f + 0.03) \qquad \textit{Shipping Noise} \qquad (2.4)$$

$$NL_3 = 50 + 7.5w^{1/2} + 20\log f - 40\log(f + 0.4) \qquad \textit{Surface Waves} \qquad (2.5)$$

$$NL_4 = -15 + 20\log f \qquad \textit{Thermal Noise} \qquad (2.6)$$

where NL is the ambient noise level in dB re 1 μPa , f is the frequency in Hz, D is the shipping density on a scale from 0 (very light) to 1 (heavy) and w is the wind speed in m/s [5]. Using the above expressions, Figure 3 shows one example of the cumulative effects of these sources over a broad frequency range, for a nominal wind speed of 10 m/s (20 knots) and light shipping (*D=0.5*). In the frequency band of interest, 9 to 14 kHz, the dominant noise source is wind-induced sea surface roughness.

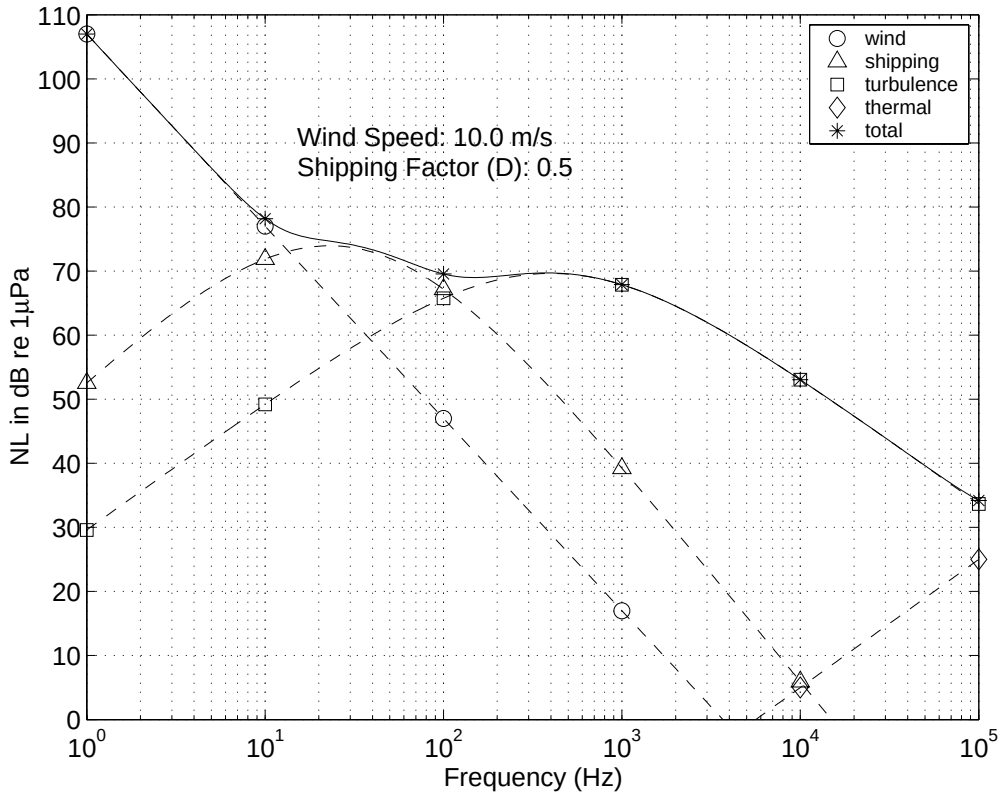

Figure 3. Deep-water ambient noise spectrum level, with light shipping and nominal sea surface wind speed of 10 m/s (Sea State 4).

In the shallow water channel additional sources of noise from biologics and coastal industry exist. The overall noise varies significantly between the times of day, the seasons, geographic locations, shipping density and weather and exhibits a large dynamic range. All this leads to a very noisy channel, which has characteristics that are quite difficult to represent statistically.

When NL and TL are combined, the overall range and frequency dependence of the channel becomes apparent as illustrated in Figure 4.

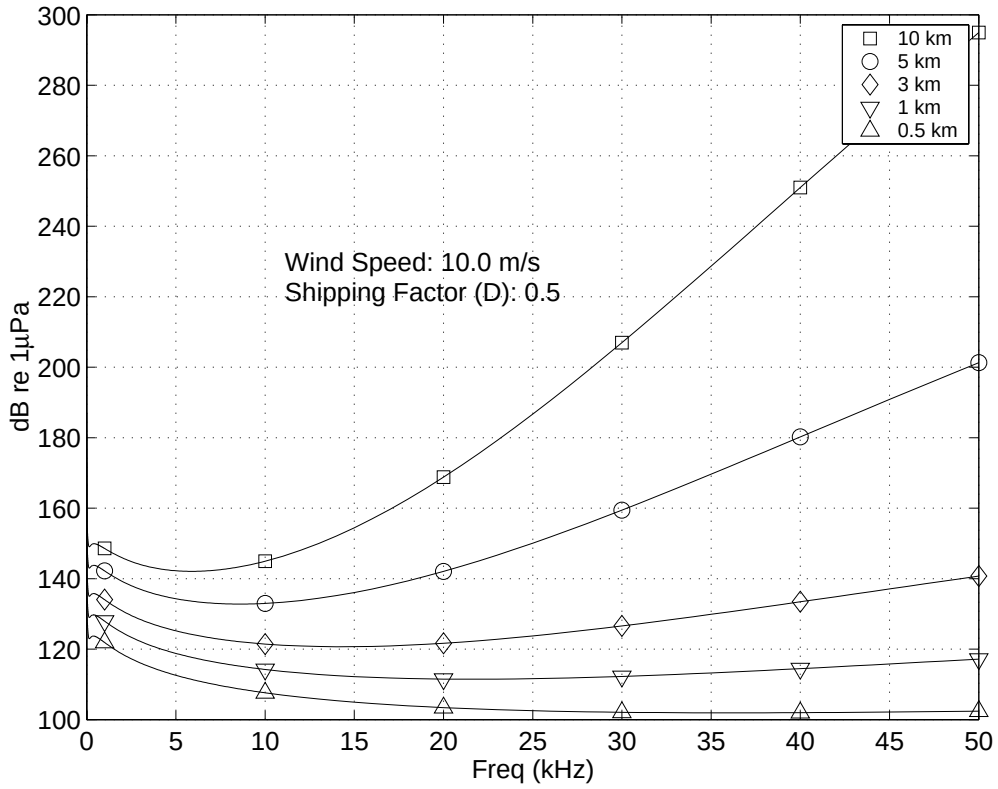

Figure 4. Frequency and range dependence of the underwater communications channel for the combined effects of noise level (NL) and transmission loss (TL).

We can see that the experimental Seaweb modem's 9 to 14 kHz band is consistent with its range requirements of 3 to 5 km. However, when compared to the radio frequency communications channel, these are extremely small bandwidths. For example, the IEEE 802.11b standard for wireless networks uses DSSS and operates in the 2.4 GHz band at a data rate of between 1 to 11 megabits per second (Mbps).

C. FADING CHANNEL

Small-scale fading in digital communications channels is the result of two mechanisms, the time spreading of the signal and the time-variant nature of the channel.

1. Time Spreading

Time spreading or time dispersion occurs when multiple versions of the transmitted signal arrive at the receiver. This is also called multipath propagation. In the underwater channel these multipath arrivals result from reflections off the sea surface and sea bottom, refraction and scatterers within the ocean volume. The reflection and refraction pattern of the sound waves is directly related to the geometry of the channel and the sound velocity profile. Figure 5 shows one example of sound wave propagation in the underwater channel, where z is the depth, c is the sound speed and r is the range.

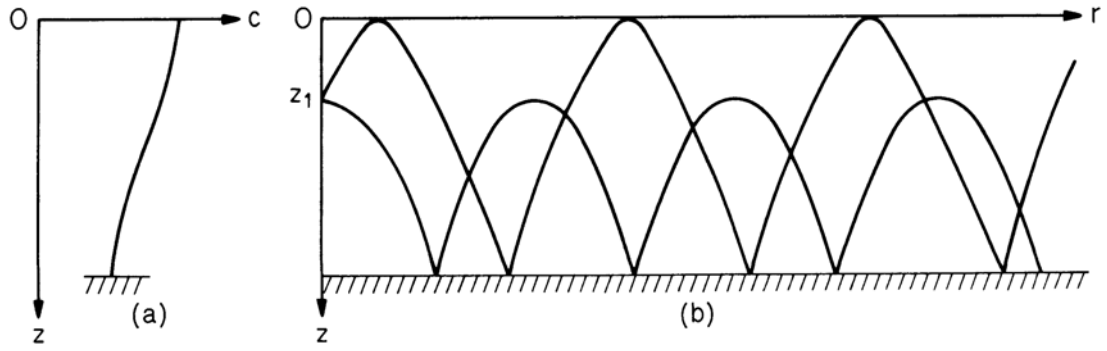

Figure 5. An example of sound propagation in a shallow water where (a) is the sound speed profile and (b) is a ray diagram representing two sound rays propagating from the source [From Ref. 6].

The random amplitude and phase of the multiple arrivals cause fluctuations in the received signal strength. The channel impulse response $h(\tau)$ is one way of characterizing the effect of multipath in a channel. However, in communication channels it is more common to refer to the multipath intensity profile (MIP), which is a measure of the average received power $S(\tau)$ as a function of the excess delay τ for a transmitted impulse. The excess delay is the time delay that occurs after the first arrival of the signal. An example of a MIP is seen in Figure 6.

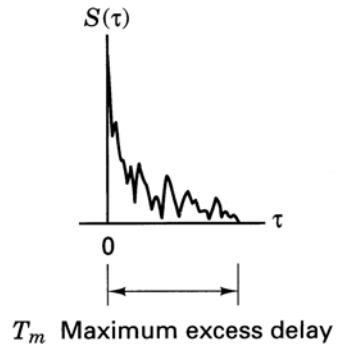

Figure 6. An example of a Multipath Intensity Profile [From Ref. 7].

The maximum excess delay T_m is the time during which the MIP is essentially non-zero. In the frequency domain, the time-spread signal can be classified by its coherence bandwidth B_{COH}. The B_{COH} is a statistical measure of the bandwidth over which the channel passes all frequency components with equal gain and equal phase. This also means that the frequency components are well correlated and that the channel's frequency-transfer function is essentially flat. T_m and B_{COH} are related by $B_{COH} \approx 1/T_m$. However, this is not the best way to classify a channel because the $S(\tau)$ may vary significantly for channels with the same T_m. Therefore a more useful parameter is the root mean squared delay spread σ_τ given by:

$$\sigma_\tau = \sqrt{\overline{\tau^2} - \overline{\tau}^2} \tag{2.7}$$

where $\overline{\tau}$ is the mean excess delay [8]. If B_{COH} is then defined as the frequency interval over which the channel's complex frequency transfer function has a correlation of at least 0.9, then B_{COH} becomes [8]:

$$B_{COH} \approx \frac{1}{50\sigma} \tag{2.8}$$

A time dispersive channel can be classified as *frequency selective* or *frequency non-selective*. In a *frequency selective* channel, the signal bandwidth W_S is larger than the coherence bandwidth $B_{COH} < W_S$ and therefore significant distortion occurs since the

spectral components of the signal are affected differently. In the time domain, this means that the symbol duration is much smaller than the maximum excess delay $T_m > T_S$; therefore successive data pulses will interfere with each other. This is called channel-induced inter-symbol interference (ISI).

In *a frequency non-selective* channel, the transmitted signal's bandwidth W_S is smaller than B_{COH}, (i.e., $B_{COH} > W_S$). Therefore the channel affects all spectral components of the signal equally. In the time domain, this means that all the symbol multipath components arrive within the symbol duration $T_m < T_S$. Consequently there is no channel induced ISI.

In the shallow water underwater communications channel, T_m can vary greatly but values in the order of ten milliseconds are not untypical [9]. This means that at chipping rates associated with the DSSS implementation in Seaweb (2400 chips per second), channel-induced ISI will occur. Later in Chapter III we will see that DSSS systems are particularly useful in rejecting interference, which includes channel-induced ISI. As a result, the experimental Seaweb modem will deal effectively with a frequency selective channel, given sufficient SNR.

2. Doppler Spreading

The delay spread and coherence bandwidth characterize the time dispersive properties of the channel but do not address its time-varying nature. If the channel was stationary and there was no motion in the transmitter and receiver, the channel would appear time invariant. The time variance is a result of the motion within the channel, specifically movement of the source, the receiver or the channel itself. This results in propagation paths that are different from moment to moment; therefore, the channel impulse responses varies over time. The parameters used to describe the channel's time-varying nature are its Doppler spread and coherence time. Doppler spread B_D is a measure of the spectral broadening of the signal caused by the channel's rate of change and is the range of frequencies over which the Doppler spectrum is essentially non-zero.

The coherence time T_{COH} of the channel is a statistical measure of the time duration during which the channel impulse response is essentially time-invariant. Doppler spread and coherence time are related by $T_{COH} \approx 1/B_D$. Sea surface roughness and source receiver motion are the dominant mechanisms resulting in Doppler spread [10]. If source and receiver are fixed, then the Doppler spread due to wind driven weather effects at the sea surface can be expressed as:

$$B_D = 2f_w\left[1+\frac{4\pi f_0 \cos\theta_0}{c}\right]h_w \tag{2.9}$$

where f_w is the wave frequency given by $f_w = 2/w$, w is the wind speed in m/s, f_0 is the carrier frequency, θ_0 is the angle of incidence and h_w is the wave height of the surface waves given by $h_w = 0.005w^{3/2}$ [10]. Again B_D can vary greatly, but without a moving source or receiver and in relatively low sea states, Doppler spreads of less than 10 Hz are not untypical.

A Doppler spread channel can be characterized as either *slow fading* or *fast fading*. In a slow fading channel, the symbol duration is less than the coherence time $T_{COH} > T_S$; therefore, the channel is essentially time invariant over the duration of the symbol. In a fast fading channel, the channel's characteristics (i.e., impulse response) changes faster than the symbol duration, $T_{COH} > T_S$. A slow fading channel is more desirable from a detection perspective, so symbol rates can be increased to meet this requirement, as long as there is a way to compensate for the channel-induced ISI that may result.

3. Doubly Spread Channels

Any channel, such as the shallow water communications channel, that undergoes both time and Doppler spread is said to be doubly spread. The product of $B_D T_M$ is called the spread factor. If $B_D T_M < 1$, then the channel is said to be underspread and if $B_D T_M > 1$ it is overspread. In underspread channels the channel impulse response can be

determined reliably and used to aid the receiver in demodulating the signal. In an overspread channel this is not possible and high data error rates occur. In the underwater acoustic channel, a spread of less than 10^{-3} is needed for coherent or differentially coherent detection [11].

In this chapter we have seen that the underwater acoustic channel imposes significant difficulties on communication signals. The channel is severely bandlimited, ocean noise in non-Gaussian and severe fading occurs due to both time spreading and Doppler spreading of the transmitted signal. In the next chapter, we will see that direct-sequence spread-spectrum modulation will help compensate for the multipath channel effects and provide the low probability of detection that Seaweb requires.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DIRECT-SEQUENCE SPREAD-SPECTRUM (DSSS) SYSTEMS

This chapter describes the key aspects of DSSS systems. First we explain the general advantages of using spread-spectrum communication techniques. Then we examine how these advantages are realized in DSSS. An overall block diagram of a DSSS system that uses quadrature spreading of a DBPSK signal is developed, with detailed explanations on how the individual blocks are implemented in practical systems. These blocks include the pseudo-random code generator, the modulator and demodulator and the channel coder. Finally, methods of compensating for the multipath channel effects are discussed.

A. DIRECT-SEQUENCE SPREAD-SPECTRUM OVERVIEW

1. Benefits of Spread-Spectrum Techniques

Spread-spectrum (SS) transmissions of digital communication signals are widely used in wireless and military applications because they are very effective at suppressing interference. This interference can occur from several sources. One source could be an adversary deliberately jamming the communications channel. Another source is the result of multiple access techniques in which many users simultaneously share the same transmission bandwidth thereby interfering with each other. Code Division Multiple Access (CDMA) in a cellular communication system is one example of this. Lastly, the interference may be the result of channel-induced ISI due to multipath arrivals in a band-limited channel.

Spread-spectrum techniques can also be used to hide a signal by transmitting it at low power. By spreading the signal energy over the widest available bandwidth and using the minimum power needed, the signal can be hidden in the channel noise. This means that any unauthorized interceptor will have a low probability of detecting the signal relative to the intended receiver. Likewise because of the pseudo-random properties of the spreading sequence, even if the signal is detected a lower probability of

it being intercepted exists. As a result, spread-spectrum signals are called *low probability of detection* (LPD) and *low probability of intercept* (LPI) signals.

There are three primary motivations for implementing spread-spectrum in Seaweb. First is to reduce the effects of channel-induced ISI in the severely band-limited underwater channel, second is to allow for multiple users in an undersea network and finally we want to transmit LPD and LPI signals so that the network can operate covertly.

To be considered a spread-spectrum technique, a transmission must have two characteristics: First, the transmission bandwidth of the signal must be much larger than the minimum bandwidth associated with the information data rate $W \gg R$. The second requirement is that the signal's bandwidth must be spread by using a spreading signal or code that is independent of the data. This code has pseudo-random properties, which allows the receiver to know *a priori* what the code is. Demodulation is then accomplished by correlating the received code with a synchronized replica in the receiver and thereby despreading the signal.

There are two basic methods for implementing spread-spectrum: DSSS involves spreading using phase modulation, FHSS involves rapidly changing the carrier frequency. A basic block diagram of a spread-spectrum system is seen in Figure 7. Only DSSS is examined in this thesis.

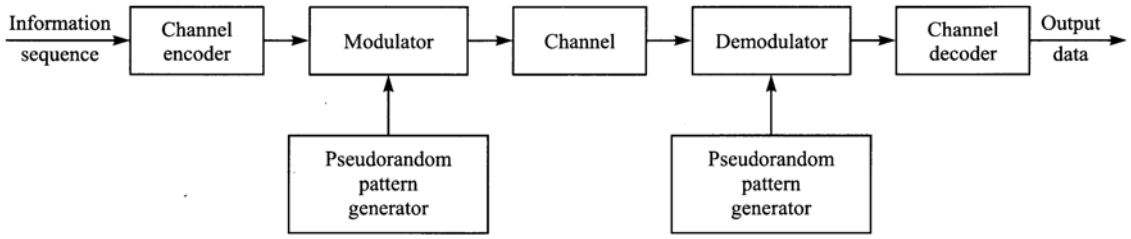

Figure 7. Block diagram of a Spread-Spectrum digital communication system [From Ref. 12].

2. Direct-Sequence Spread-Spectrum

In DSSS the spreading of the signal bandwidth occurs at baseband by multiplying the baseband data pulses with a chipping sequence. This chipping sequence is a pseudo-random binary waveform with a pulse duration of T_C and a chipping rate of $R_C = 1/T_C$. Each pulse is called a chip and T_C is the chip interval. For a given information symbol of duration T_S and a symbol rate of $R_S = 1/T_S$, the duration of each chip is much less than the pulse length of the information symbol (i.e., $T_C \ll T_S$) and R_C is much higher than the symbol rate (i.e., $R_C \gg R_S$). In practical systems, the number of chips per symbol N_C must be an integer number with the transition of the data symbols and the chips occurring at the same time. The ratio of chips to symbols is called the spreading gain k or bandwidth expansion factor B_e where:

$$k = B_e = N_C = \frac{T_S}{T_C} = \frac{R_C}{R_S}$$

A PN code has a fixed-length of N chips and can be classified as either long or short. In a *short code* the entire chip sequence is transmitted within every data bit. In a *long code* only a portion of the sequence is transmitted within each data bit and typically $N / N_C \gg 1$. The chipping sequence and the data sequence are combined by modulo-2 summing the binary sequences or by multiplying the two pulsed waveforms. The relationship between chips and symbols and the resulting spread data sequence is seen in Figure 8.

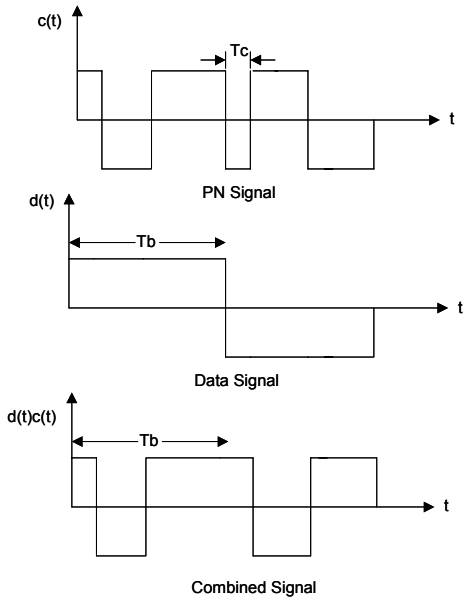

Figure 8. The relationship between the spreading sequence c(t) and the information sequence d(t) for a DSSS signal with six chips per bit.

The chip duration is chosen in order to spread the signal over the maximum available bandwidth of the channel. A rectangular pulse $g(t)$ of length T, has a null-to-null bandwidth $B_{nn} = 2/T$. Therefore for a channel bandwidth W, $T_C = 2/W$ or $R_C = W/2$, meaning that the chip rate should be half the available channel bandwidth. Higher chipping rates, approaching the bandwidth (i.e., $W/2 < R_C < W$) are possible using more sophisticated techniques like root-raised cosine pulse-shaping of the data waveform, but these are not addressed here. Despreading of the DSSS signal in the receiver is accomplished by again multiplying the signal by the same PN sequence.

3. Interference Suppression

We can now examine how spreading the signal bandwidth helps suppress interference. To simplify the description, consider only baseband communication and wideband interference, which is consistent with barrage noise jamming, multi-user applications or multipath arrivals.

As seen in the block diagram in Figure 9, multiplication with the PN sequence in the transmitter spreads the data signal over the entire bandwidth. At the receiver, multiplication with the same PN sequence gives a selective despreading of the data signal. Yet the interference signal is not despread since it is uncorrelated with the PN sequence and continues to occupy the entire bandwidth. This increases the received signal-to-noise ratio over the no-spreading case.

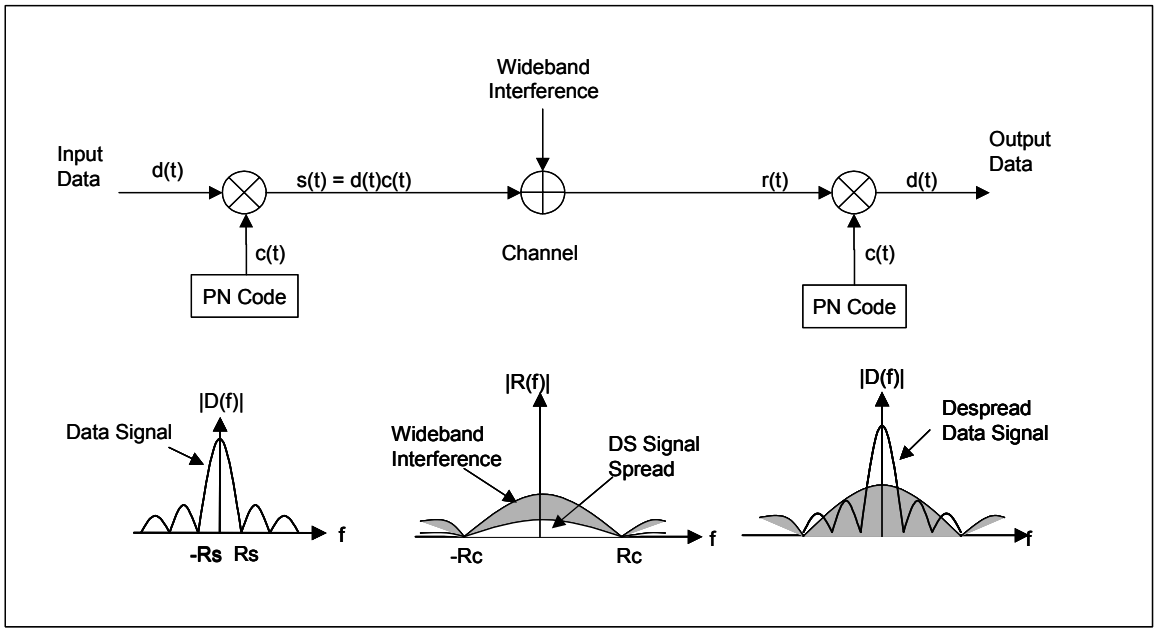

Figure 9. Effects of wideband interference on DSSS signal [After Ref. 13].

If the signal were not spread, then the interference would still occupy the same bandwidth as the transmitted signal and would severely degrade performance. This is illustrated in Figure 10.

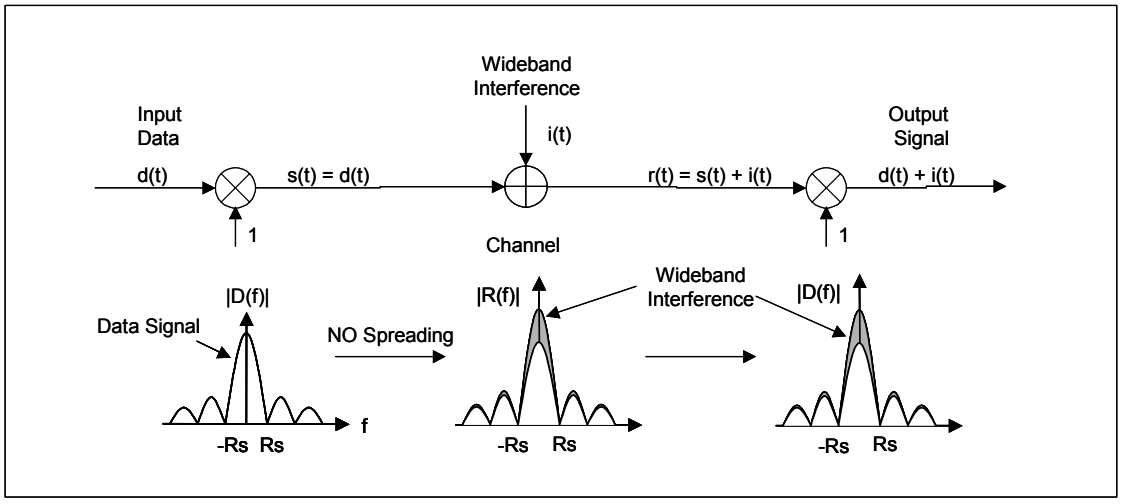

Figure 10. Effects of wideband interference on a BPSK signal without spreading.

Figure 11 shows the case of no interference and only additive white Gaussian noise (AWGN). In this case DSSS offers no advantage. The SNR of the output data signal is the same whether spreading is implemented or not. This is because the noise remains uniform across the entire spectrum even after despreading.

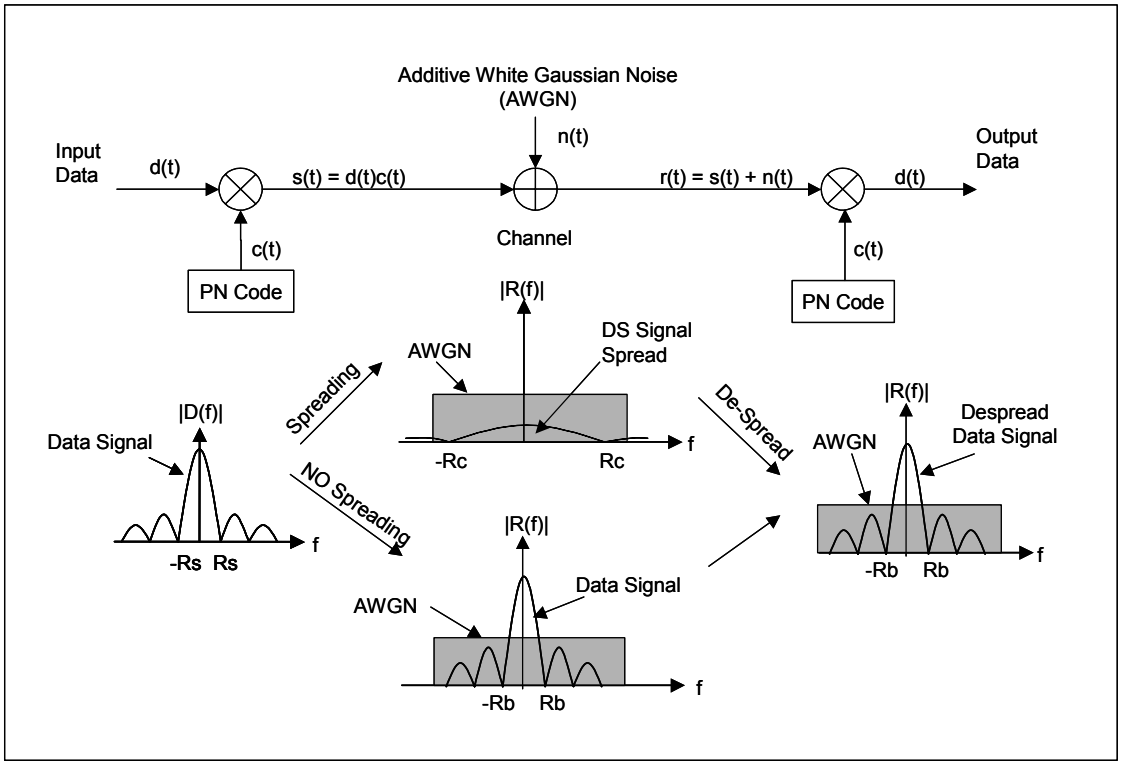

Figure 11. Effects of AWGN on a DSSS signal [After Ref. 13].

B. PSEUDO-RANDOM NOISE (PN) SPREADING SEQUENCES

The chipping waveform $c(t)$ is modeled as a zero mean, polar random binary wave, in that it can assume the state (+1 or –1) with equal probability. Since each $c(t)$ has a duration of T_C seconds, an infinite length sequence has an autocorrelation function given in Figure 12.

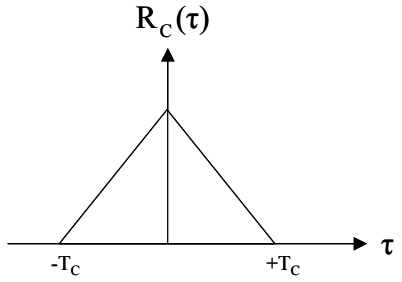

Figure 12. The correlation function for a polar random binary wave.

In practice $c(t)$ must be finite and deterministic because the receiver must know the sequence *a priori* in order to produce a replica of $c(t)$ and to despread the data. The chip sequence is generated from a pseudo-random noise (PN) sequence generator, which should have the same autocorrelation as a polar random binary wave.

1. Maximal Length Sequences

The most common method of generating the PN code is using a series of n-shift registers. Depending on the specific implementation, the output of each shift register may or may not be fed back to the input through an exclusive-OR (XOR) operator. If the feedback is designed correctly, the output of the shift registers will produce a binary series in which the output will cycle through the maximum number of states before repeating itself. This particular code is called a maximal length sequence or m-sequence. The output taken at any of the shift registers is an m-sequence. Therefore, a family or set of m-sequences is generated from any given configuration. An example of a four-register m-sequence generator is seen in Figure 13.

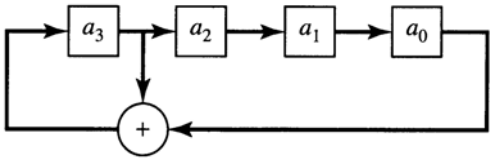

Figure 13. A four stage m-sequence generator [From Ref. 14].

For any given configuration of n-stage registers, the length of the m-sequence will be $N = 2^n - 1$ (15 in this example) and a set of n different m-sequences can be generated. Because the length of the sequence is odd, there will not be an equal number of ones and zeros and an additional one is present. The configuration of the feedback connections is defined by the generator polynomial, which for the above example is:

$$g(D) = 1 + D + D^4. \tag{3.1}$$

This indicates that the output D^1 of the first shift register a_3 and the output D^4 of the fourth shift register a_0 are fed back to the input D^0. The autocorrelation function of an m-sequence is seen in Figure 14.

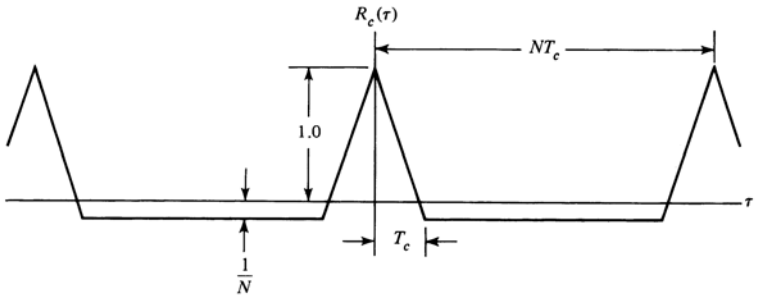

Figure 14. The autocorrelation function for an arbitrary m-sequence [From Ref. 14].

Although m-sequences have good autocorrelation properties, they can have very large cross-correlations. These poor cross-correlation properties make them unsuitable for multi-access applications. When multi-access codes are required, we want the cross-correlation of two independent PN sequences to be small. In this case, Gold codes are often used.

2. Gold Codes

In a set of m-sequences of length N, some will have "good" cross-correlation properties. These sequences are called "preferred m-sequences." A set of new PN sequences can then be generated from two preferred m-sequences by modulo-2 summing them in a specific manner. These new sequences will exhibit the same good cross-correlation properties as the original two preferred m-sequences used to generate them. These new sets of PN sequences are called Gold codes. One example of a Gold code generator is seen in Figure 15. The two m-sequence generators seen here are shown in their *high-speed* implementations rather than as shown in Figure 13 above.

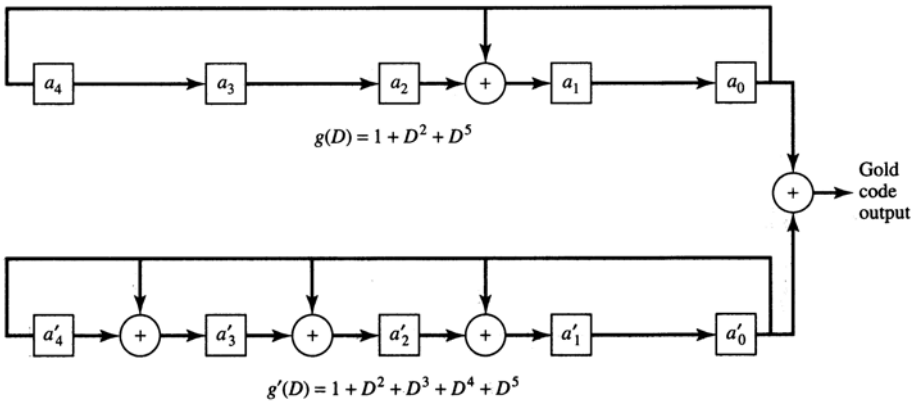

Figure 15. A typical Gold code generator [From Ref. 14].

Other PN codes like Kasami codes, Hadamard codes, Barker codes and others exist and are widely discussed in technical literature but are not addressed here. Likewise, only the auto- and cross-correlation properties between full-length sequences have been discussed. In the case in which long codes are used for chipping, only a portion of the sequence is used to chip any given bit and therefore partial period cross-correlation properties are important. In the experimental Seaweb modem implementation long Gold codes are used.

C. DIRECT-SEQUENCE DIFERENTIALLY ENCODED BINARY PHASE-SHIFT KEYING WITH QUADRATURE SPREADING (DS-IQ-DBPSK)

This section addresses how the chipped data sequence is modulated and demodulated using differentially encoded binary phase-shift keying with quadrature spreading. First the general theory behind DBPSK is examined. Then we examine how the signal is spread and modulated using a balanced in-phase and quadrature-phase modulator. Finally we will look at how the DSSS waveform is demodulated at the receiver with particular emphasis on using a RAKE receiver for diversity reception.

1. Binary Phase-Shift Keying (BPSK)

In BPSK, the data sequence modulates the phase of a constant amplitude carrier. Typically the two phases are 0^o and 180^o. Figure 16 shows a typical BPSK waveform in the time domain.

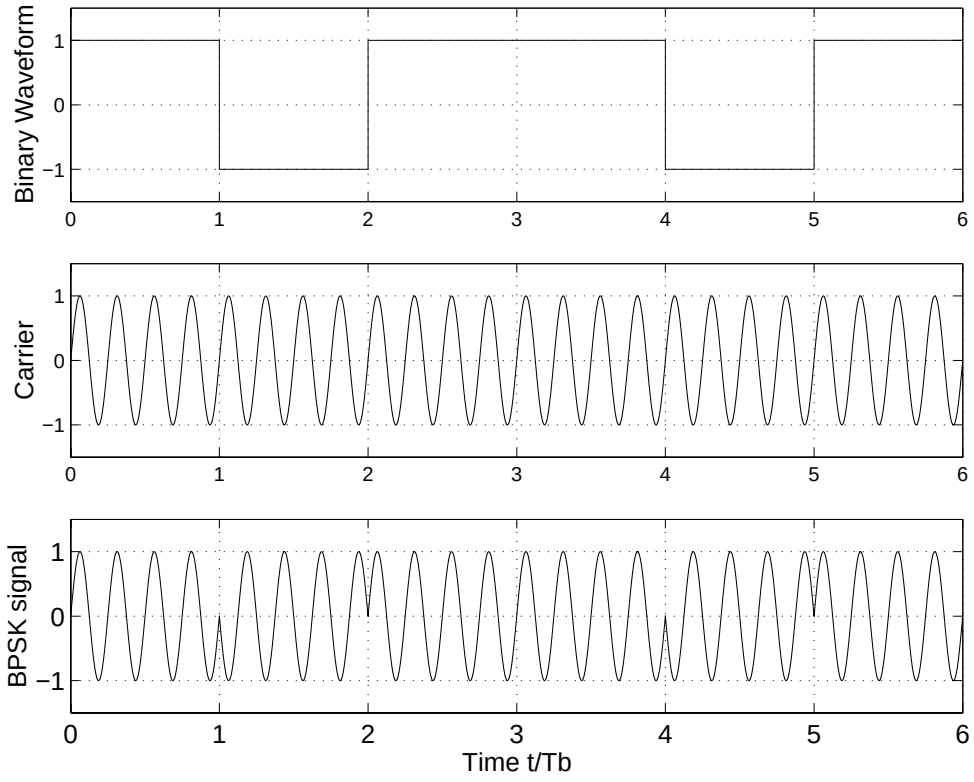

Figure 16. A BPSK signal in the time domain.

If the carrier has an amplitude A_C and the bit duration is T_b then the energy per bit is $E_b = \frac{1}{2} A_c^2 T_b$ and the transmitted BPSK signal can be expressed as:

$$\begin{aligned} s_i(t) &= A_C \cos(2\pi f_c t + \theta_i(t)) \quad 0 \le t \le T_b \\ &= \sqrt{\frac{2E_b}{T_b}} \cos(2\pi f_c t + \theta_i(t)) \\ \theta_i &= \begin{cases} 0, & i = 1 \\ \pi, & i = 2 \end{cases} \end{aligned} \tag{3.2}$$

where f_c is the carrier frequency and $\theta_i(t)$ is the phase modulation term. Since $\theta_i(t)$ is restricted to 0 or π the above expression can be rewritten as:

$$\begin{aligned} s(t) &= b(t)\sqrt{\frac{2E_b}{T_b}} \cos \omega_c t \\ &= b(t)\sqrt{E_b}\Psi_1(t) \end{aligned} \tag{3.3}$$

where $b(t) = \pm 1$ represents the polar random binary data waveform, $\omega_C = 2\pi f_c$ and $\Psi_1(t) = \sqrt{2/T_b} \cos \omega_c t$. The signal can also be represented graphically as a vector on a polar plot where the axes represent the in-phase and quadrature-phase components of the signal. This plot is called the signal constellation. The vector's magnitude is the signal amplitude and the direction corresponds to the phase. The constellation for BPSK is seen in Figure 17.

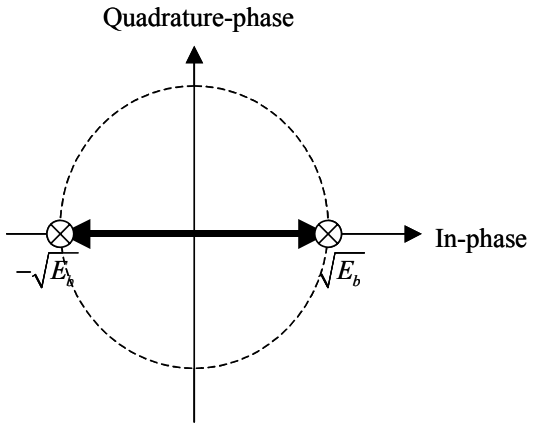

Figure 17. The signal constellation for BPSK.

The frequency domain representation of the BPSK signal can be seen in Figure 18.

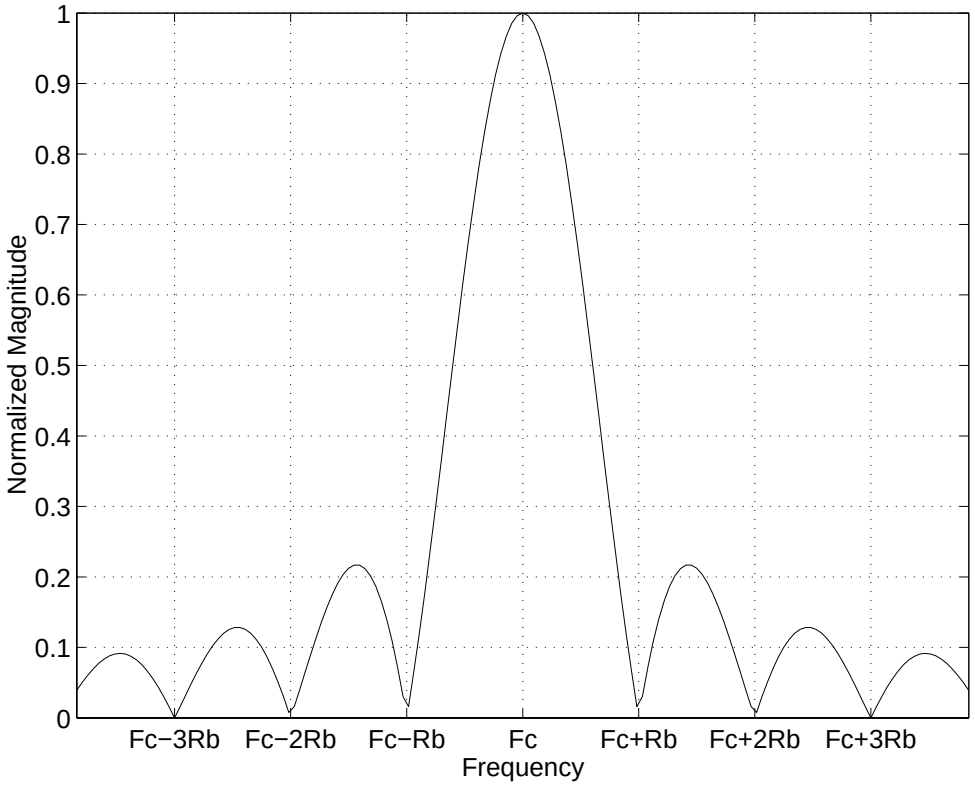

Figure 18. A BPSK signal in the frequency domain.

BPSK can be coherently or non-coherently demodulated. Non-coherent detection is often used because it has only slightly inferior performance (less than 3 dB) when compared to coherent detection, but it does not require elaborate methods of tracking the phase of the received signal. When non-coherent detection is used, the data bits are differentially encoded in what is known as Differentially Encoded Binary Phase-Shift Keying (DBPSK).

Whether DBPSK should be considered a non-coherent or coherent detection scheme can be confusing. Most authors [7] call DBPSK a non-coherent technique because the phase of the received signal is not tracked continuously. Some authors [12] point out that although DBPSK is a non-coherent scheme, it does involve the assumption that the phase is constant over at least two successive bits and consequently can be thought of as an extreme case of coherent detection (i.e., a type of semi-coherent detection). But because DBPSK is not an energy detection method but a phase detection method, other authors [15] simply define it as differentially coherent. For our purposes DBPSK will be strictly called non-coherent.

2. Differential Encoding

In DBPSK, the data must be differentially encoded before modulation. The output of the differential encoder is dependant on whether the present symbol is the same or different than the previous symbol. The first bit in the sequence is arbitrarily chosen. There are several methods of implementing the coding, for example:

$$d(k) = \overline{d(k-1) \oplus b(k)} \tag{3.4}$$

where $\oplus$ represents the XOR operation, $b(k)$ is the present data bit and $d(k-1)$ is the previous differentially encoded bit. Table 1 shows an example of this operation for the bit stream 1 1 0 1 1. Although the differential bit sequence will always have an extra bit at the start, a DBPSK transmission will, in all other respects, resemble a BPSK transmission.

sample k	0	1	2	3	4	5
b(k)		1	1	0	1	1
d(k)	1	1	1	0	0	0
θ(k)	0	0	0	π	π	π

Table 1. Generation of the phase angles from the input data sequence for DBPSK.

3. Chipping the DBPSK Signal

We now consider how the chipping sequence effects the modulator. The DBPSK transmission can be expressed as:

$$s(t) = d(t)A_C \cos \omega_C t \tag{3.5}$$

where $d(t) = \pm 1$ and represents the differentially encoded polar binary waveform derived from the bit sequence $d(k)$. In Section A.2 we saw that $c(t)$ and $d(t)$ had to be synchronized so that there are an integer number of chips per bit and the transitions of chips and bits had to occur at the same time. The DSSS signal was spread at baseband by multiplying the two signals, with the result:

$$s_{DS}(t) = c(t)d(t)A_C \cos \omega_C t. \tag{3.6}$$

The product $c'(t) = c(t)d(t)$ is just another random polar binary wave but with a pulse duration T_C and therefore:

$$s_{DS}(t) = c'(t)A_C \cos \omega_C t. \tag{3.7}$$

This is equivalent to a BPSK transmission in which the "bit" duration is T_C.

4. Pulse-shaping

When rectangular pulses propagate through a band-limited channel, they are spread in time. This is because the sharp transitions at the edges of the pulse contain the high frequency components, which are smoothed by filtering. This spreading causes successive pulses to interfere with each other and also results in poor correlation of the signal with its replica in the receiver.

These adverse effects can be mitigated by shaping the amplitude of the pulse before it is transmitted. One simple method is a band-limiting filter equal to the bandwidth of the channel. This ensures that the band-limiting distortion at the transmitter is known and can therefore be compensated for in the receiver. Other more sophisticated techniques, such as raised root-cosine or Gaussian pulse-shaping filters, can also be used for specific applications but are not addressed here.

5. Balanced Quadrature Modulation of the DS DBPSK Signal

In spread-spectrum applications it is common to send the DBPSK signal over both an in-phase (I) and quadrature-phase (Q) channel, as this is known to have better performance in some jamming environments. Yet, more importantly for the Seaweb application is that, when different chipping sequences are used for the I and Q channels, it will be more difficult for a unauthorized interceptor to detect the signal, as will be shown later. This modulation technique is often called balanced QPSK or quadrature spread DBPSK. In this thesis, balanced QPSK will be referred to as IQ-DBPSK to stress the fact that it is a DBPSK signal sent over an I and Q channel. A block diagram of this implementation is seen in Figure 19.

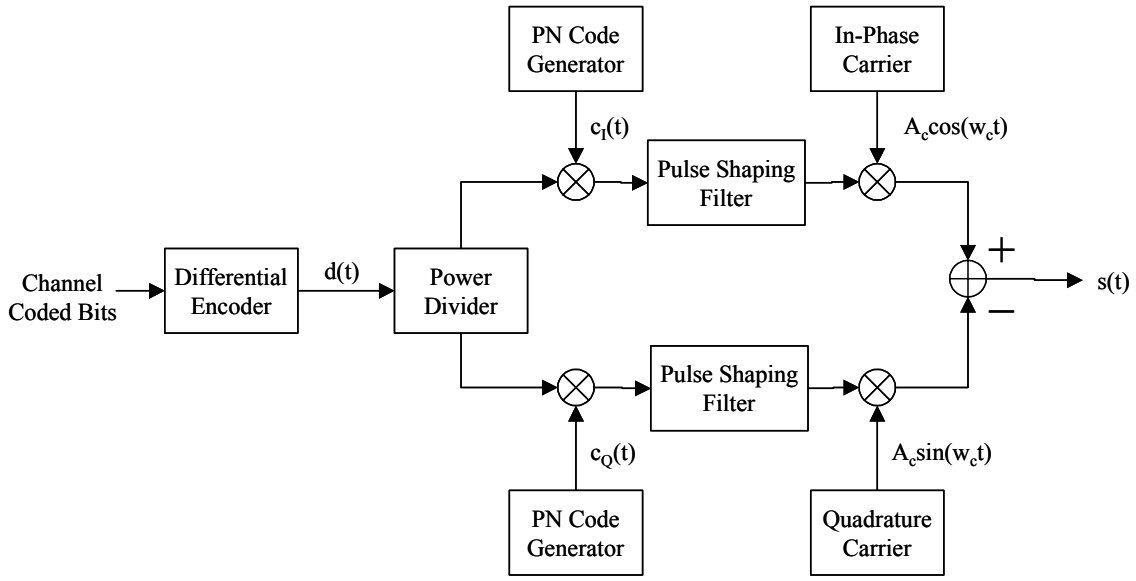

Figure 19. Block diagram for DS-IQ-DBPSK modulator.

The DS-IQ-DBPSK signal can be expressed as:

$$s_{DS-IQ-BPSK}(t) = \frac{1}{\sqrt{2}} A_C \left[c_I(t)\cos[\omega_C t + \theta_i(t)] - c_Q(t)\sin[\omega_C t + \theta_i(t)] \right] \qquad (3.8)$$

where $\theta_i(t) = 0, \pi$ is the phase modulation term which is the same for both the I and Q channels, $c_I(t)$ and $c_Q(t)$ are the spreading waveforms used on each channel and the $1/\sqrt{2}$ is a scaling factor so that the signal power is evenly split between the two channels.

6. Synchronization of the PN Sequence in the Receiver

In a DSSS system, it is critical that the receiver's replica of the spreading code be aligned or synchronized with the received signal in order to demodulate it successfully. A misalignment of even one chip will result in a loss of the signal. The process of synchronizing the codes is done in two stages. The first is called "acquisition" and involves detecting the presence of the signal and bringing the codes into rough alignment. The second step is called "tracking," which involves fine synchronization between the codes. The codes are kept in alignment by using a feedback control loop called a Delay Lock Loop (DLL).

a. Acquisition

Several methods exist to acquire DSSS signals and all involve correlating the received signal with a replica and comparing the output with a threshold to determine if the signal is present and roughly synchronized. Instead of implementing these typical methods found in [7] and [12], our initial Seaweb utility packet design follows the work presented in [16]. In this implementation an acquisition frame is added to the start of each transmission.

The frame consists of three short duration PN sequences. These sequences are modulated in the same manner as the data, that is, they are DS-IQ-DBPSK signals. In the receiver these pulses are passed through a filter matched to one of the PN sequences. When three correlation peaks of similar amplitude and separation are detected, the signal

is determined to be present. An estimate of the start of the first data bit and first chip is determined from the distance between the correlation peaks. A non-coherent correlator is used in this application.

b. Tracking

Now that the start of the data has been determined and the signal has been coarsely aligned, the DLL is used to perform fine synchronization. An implementation of a non-coherent DLL is seen in Figure 20.

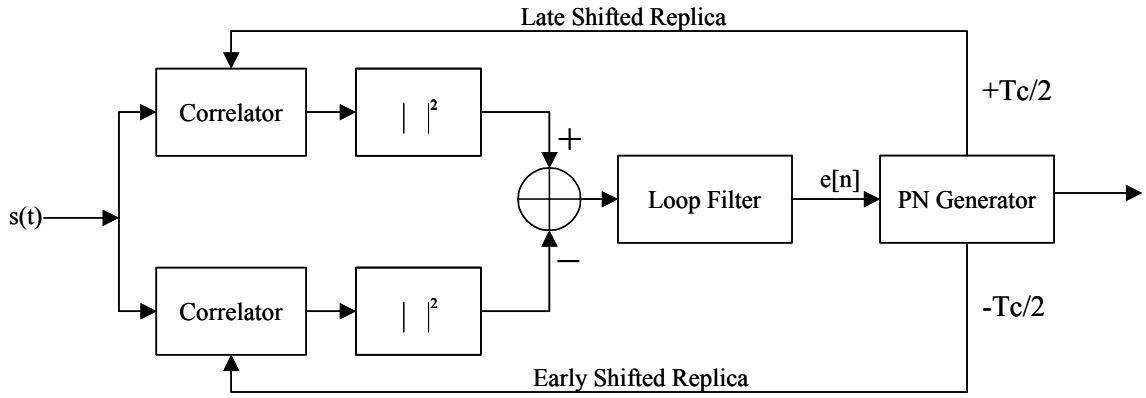

Figure 20. Block diagram of the DLL implementation [After Ref. 16].

The basic principle behind this is to compare the received PN sequence $s(t)$ to an early and late shifted version of the receiver's replica and to generate an error signal $e[n]$, which is proportional to the amount by which the two signals are out of alignment. This error signal is fed back to the PN sequence generator in the receiver, causing it to advance or delay by a number of samples proportional to the error. The goal is to drive the error to zero at which point the two sequences will be precisely aligned. Correlating the incoming code with an early and late shifted version of the replica generates the error signal. The two correlator outputs are then subtracted from each other, creating what is called an "S-curve," as seen in Figure 21.

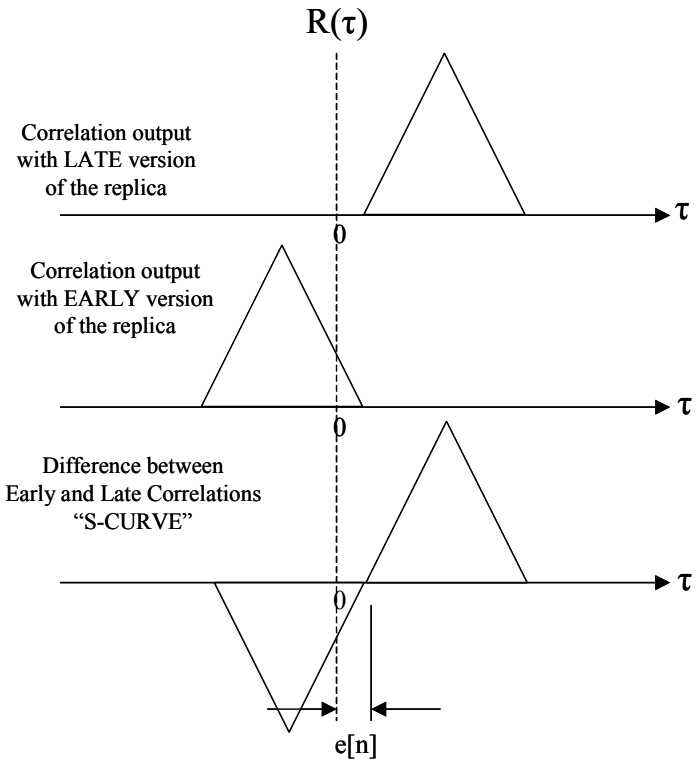

Figure 21. "S-curve" generated by the DLL where τ is the correlation lag.

If there is no misalignment between the PN codes, the zero crossing of the S-curve will occur in the center of the correlation output. Any misalignment will result in a zero crossing that is either left or right of center. The number of samples that the zero crossing is offset is equal to the number of samples that the replica PN sequence must be shifted in order to be aligned with the incoming signal. This process is done continuously for each bit.

One common problem with DLLs is that at low SNRs the error signal will experience noise jitter and will be erratic. A loop filter is therefore used to limit the variance of the noise in the error signal. The loop filter is a lowpass filter whose bandwidth must be designed to maximize the performance of the DLL. Small bandwidths result in less jitter but larger bandwidths allow for quicker adjustments to any misalignments.

7. Demodulation of DS-IQ-DBPSK

Once the receiver is synchronized with the incoming waveform, there are several possible methods of demodulating the DS-IQ-DBPSK signal. The model chosen for the experimental Seaweb modem is seen in Figure 22 and is also called a quadrature demodulator.

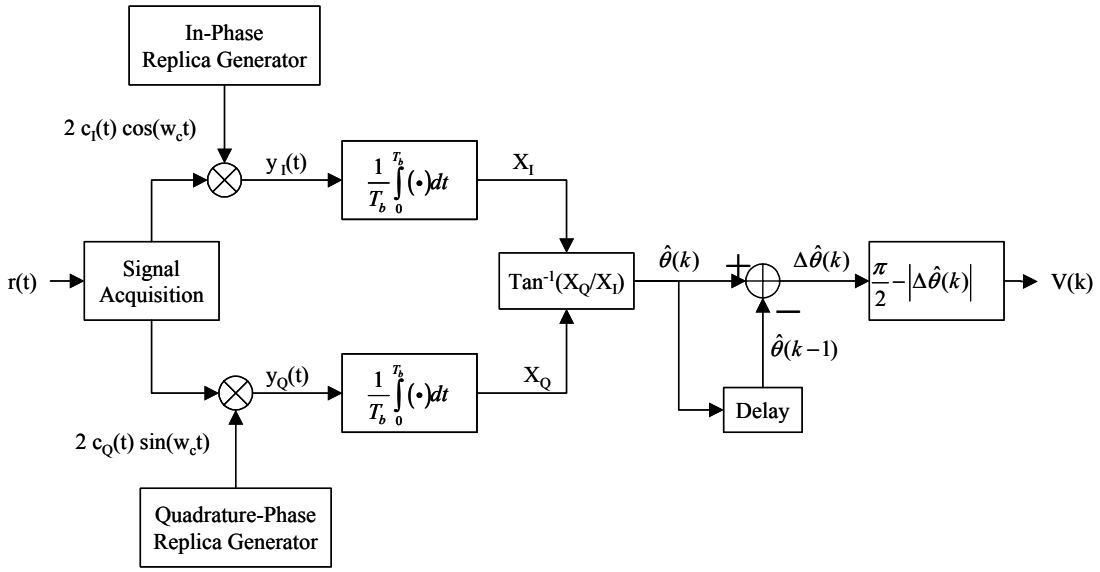

Figure 22. Block diagram of a non-coherent DS-IQ-BPSK receiver [After Ref. 12].

Ignoring the effects of the channel, the received signal will be the same as the transmitted signal $s(t)$ given by Equation 3.6 above. The input to the I-channel integrator is given by:

$$\begin{aligned} y_I(t) &= 2c_I(t)\cos(\omega_C t)\bullet s(t) \\ &= 2c_I(t)\cos(\omega_C t)\frac{A_C}{\sqrt{2}}\Big[c_I(t)\cos[\omega_C t+\theta_i(t)]-c_Q(t)\sin[\omega_C t+\theta_i(t)]\Big]. \end{aligned} \tag{3.9}$$

The output of the integrator can be expressed as:

$$x_I = \frac{1}{T_b}\int_0^{T_b} 2c_I(t)\cos(\omega_C t)\frac{A_C}{\sqrt{2}}\Big[c_I(t)\cos[\omega_C t+\theta_i(t)]-c_Q(t)\sin[\omega_C t+\theta_i(t)]\Big]\,dt. \tag{3.10}$$

Due to the properties of the PN sequences, $\int_0^{T_b} c_I(t)c_I(t)dt = 1$ and $\int_0^{T_b} c_I(t)c_Q(t)\,dt = 0$ and Equation 3.10 becomes:

$$x_I = \frac{1}{T_b}\int_0^{T_b} 2\frac{A_C}{\sqrt{2}}\cos(\omega_C t)\cos[\omega_C t + \theta_i(t)]\;dt. \tag{3.11}$$

Recall from Equation 3.2 that because $\theta_i(t) = 0, \pi$ Equation 3.11 can be rewritten as:

$$\begin{aligned} x_I &= \frac{1}{T_b}\int_0^{T_b} 2\frac{A_C}{\sqrt{2}}\cos(\omega_C t)d(t)\cos[\omega_C t]\;dt \\ &= \frac{1}{T_b}\int_0^{T_b} \frac{A_C}{\sqrt{2}}d(t)\left[1+\cos(2\omega_C t)\right]\;dt \\ &= \frac{A_C d(t)}{\sqrt{2}T_b}\left[T_b + \frac{\sin(4\pi f_C T_b)}{4\pi f_C T_b}\right] \end{aligned} \tag{3.12}$$

where $d(t) = \pm 1$. Assuming that $f_C T_b \gg 1$ and $f_C = nR_b$ (both of which are true for the experimental Seaweb modem), then x_I becomes:

$$x_I = d(t)\frac{A_C}{\sqrt{2}}. \tag{3.13}$$

Using exactly the same approach for the Q-channel the output of the integrator is:

$$x_Q = d(t)\frac{A_C}{\sqrt{2}}. \tag{3.14}$$

An estimate of the true phase of the k^{th} received differentially encoded bit $\hat{\theta}(k)$ can then be determined from the expression:

$$\hat{\theta}(k) = \tan^{-1}\left(\frac{x_Q}{x_I}\right) = \tan^{-1}\left(\frac{\pm A_C/\sqrt{2}}{\pm A_C/\sqrt{2}}\right) = \frac{\pi}{4}, \frac{5\pi}{4} \tag{3.15}$$

and the magnitude of the k^{th} differentially encoded bit is given by:

$$\left|x[k]\right| = \sqrt{x^2{}_I + x^2{}_Q}. \tag{3.16}$$

The differential encoding is removed by comparing the phase of the present bit $\hat{\theta}(k)$ to the phase of the previous bit $\hat{\theta}(k-1)$, which yields:

$$\Delta\hat{\theta}(k) = \hat{\theta}(k) - \hat{\theta}(k-1) = 0, \pi \tag{3.17}$$

where $\Delta\hat{\theta}(k)$ is the phase of the k^{th} channel bit. This can be converted to an output voltage V that is then used as the decision variable to determine the original polar binary data sequence $b(k)$ by taking:

$$V = \frac{\pi}{2} - \left|\Delta\hat{\theta}(k)\right| \tag{3.18}$$

$$b(k) = \begin{cases} 0, & \text{if } V < 0 \\ 1, & \text{if } V \geq 0. \end{cases} \tag{3.19}$$

In the case of soft decision decoding, this decision is not made in the receiver, instead the voltage level V is passed to the soft decision decoder.

8. LPD Properties of DS-IQ-DBPK

With an understanding of how a DS-IQ-DBPSK signal is generated and detected, we can now briefly examine its LPD and LPI properties. First we define what is meant by LPD and LPI, since this definition can differ between references. The probability of detecting a signal is associated with determining if the signal is present of not. As discussed in Section A.1, DSSS is inherently LPD because it can hide the signal below the channel noise by spreading its energy over a larger bandwidth. The probability of intercepting a signal is associated with determining its spreading code. Reference [15] defines an LPI signal as a spread-spectrum signal whose code is unknown to the interceptor. Therefore, even if the presence of a DS signal is detected, it may still be LPI. Although more sophisticated (and classified) means are available to detect and to intercept DSSS signals, a simple method, discussed further here, is to use a square-law device.

If a DS-BPSK signal is squared, then:

$$\begin{aligned} s^2(t) &= 2\frac{A_C{}^2}{2}c^2(t)d^2(t)\cos^2(\omega_c t+\theta_0) \\ &= \frac{A_C{}^2}{2}\underbrace{c^2(t)d^2(t)}_{=1}[1+\cos(2\omega_c t+2\theta_0)] \\ &= \frac{A_C{}^2}{2}+\frac{A_C{}^2}{2}\cos(2\omega_c t+2\theta_0)]. \end{aligned} \tag{3.20}$$

Therefore by simply squaring the signal, it can be despread and detected its above the noise, as well as determine its carrier frequency. Also, only one code sequence needs to be intercepted.

If, however, the transmitted signal uses quadrature spreading and different PN codes for the I and Q channels, then:

$$\begin{aligned} s^2(t) = A_C{}^2\underbrace{c_I{}^2(t)d^2(t)}_{=1}\cos^2(\omega_c t+\theta_0) + A_C{}^2\underbrace{c_Q{}^2(t)d^2(t)}_{=1}\sin^2(\omega_c t+\theta_0) \\ -2A_C{}^2c_I(t)c_Q(t)d^2(t)\cos(\omega_c t+\theta_0)\sin(\omega_c t+\theta_0) \\ = A_C{}^2 - 2A_C{}^2c_I(t)c_Q(t)d^2(t)\sin(2\omega_c t+2\theta_0). \end{aligned} \tag{3.21}$$

Since $c_I(t)$ and $c_Q(t)$ are different but synchronized, the result is another DS signal with the same processing gain but at a carrier frequency of $2\omega_c$. Therefore a DS-IQ-DBPSK signal cannot be detected using a simple square-law device.

D. RAKE RECEIVER

It should be intuitive from Chapter II that a receiver's performance will be poorer in fading environments (although this will be proven analytically in Chapter V). In a fading channel, the signal's energy is spread over several multipath arrivals. Therefore if the receiver only demodulates the first arrival, usually associated with the main path, much of the signal energy is lost to the receiver. Even in the case of SS, in which we can

reject the ISI interference caused by these other arrivals, we do not exploit the energy in them to improve detection.

One method to improve performance in fading channels is "diversity reception." Diversity means receiving several versions of the signal. This can be done using several antennas or in our case hydrophones (spatial diversity), transmitting the signal on several frequencies (frequency diversity) or repeating the same transmission at different times (time diversity). By summing these diversity receptions, we can improve performance in a fading channel. In the case of multipath, there is inherent time diversity due to the multiple arrivals of the signal. If, as in the case of DSSS, the multipath arrivals are separated by time intervals greater than T_C, then these arrivals will be resolvable and can be summed. A receiver that performs this is called a RAKE. The RAKE can also be thought of as a finite impulse response (FIR) filter, "matched" to the channel impulse response.

In the case of DS-IQ-DBPSK, the RAKE is implemented essentially as a bank of demodulators processing successive time delays of the received signal [12]. The processing delays are T_C seconds apart. Figure 23 shows a block diagram of this implementation.

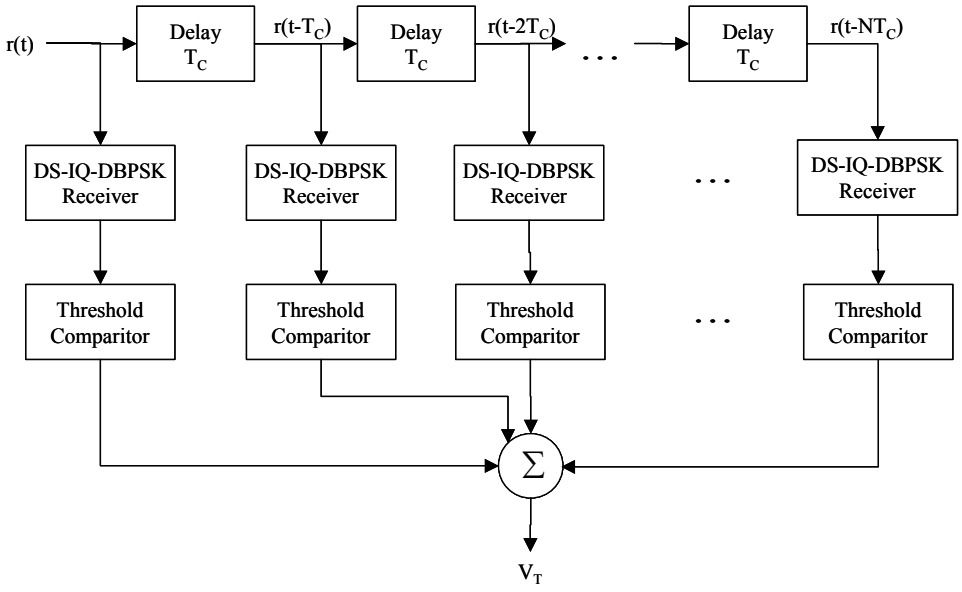

Figure 23. Block diagram of a RAKE receiver for DS-IQ-DBPSK.

Each processing leg of the RAKE is called a "tap." The output of each of the n taps is a voltage level V_n and in our case the outputs of all taps are linearly combined (i.e., no weighting is used) to give an overall output voltage V_T. This can be expressed as:

$$V_T = \sum_{n=1}^{N} V_n . \tag{3.22}$$

This new output voltage V_T is then used as the decision variable to determine the original binary data sequence.

$$b(k) = \begin{cases} 0, & \text{if } V_T < 0 \\ 1, & \text{if } V_T \geq 0. \end{cases} \tag{3.23}$$

Again, in the case of soft decision decoding, this decision is not made in the receiver and the voltage level V_T is output to the soft decision decoder.

Two problems occur with this implementation of the RAKE. First, if there is no signal present in a given tap, the output voltage will be due to noise only. When this noise is added with all the other tap outputs, the SNR will drop and the performance will suffer. This is typically overcome by using a threshold at the outputs of each tap. If the voltage is below the threshold, the voltage is deemed to be noise and the output from that tap will be zeroed. If the output is above the threshold, the voltage is considered to have a signal component and it will be summed with the other taps.

The second problem with this RAKE implementation is that it is unlikely that the multipath arrivals will be separated by exactly integer multiples of T_C. This means that the multipath energy will be spread over two successive taps, which will further degrade performance. This problem can be addressed by adaptively determining the tap locations to coincide with the multipath arrivals. If the channel impulse response can be estimated, then these optimal tap positions can be determined. The estimated channel impulse response must be updated at sufficient intervals so that it tracks the changes in the channel.

E. CHANNEL CODING

1. Overview of Channel Coding

The purpose of channel coding is to detect, to correct and to limit errors caused by noise and fading in the channel. This improved performance typically comes at the expense of computational load and or bandwidth and involves adding redundancy to the data. A trivial example would be to send a data bit repeated three times, i.e., a 1 would become 1 1 1. The receiver would then do a majority vote on the three coded bits to determine the actual bit sent. For example, if a 1 0 1 sequence were received, the receiver would interpret that as a 1. Therefore, in this case we can tolerate one in every three bits being in error.

In this simple example, we can see that the improved performance comes at the expense of a larger bandwidth since the coded bits must be sent three times as fast. Channel coding is a rich field of research with many different coding algorithms available. We will restrict our discussion to two techniques, convolutional encoding and block interleaving. Convolutional coding is a forward error-correction (FEC) code technique in which errors are detected and corrected. Block interleaving is a method of sequencing the data to limit the effects of channel burst errors.

Other candidate schemes like block-coding or turbo-coding do not meet the immediate requirements for Seaweb utility packets. Block codes in general do not perform as well as convolutional codes and turbo codes need long bit sequences, which are not possible because the Seaweb utility packets are short fixed-length (72-bit) sequences.

2. Convolutional Coding

A convolutional encoder is composed of k sets of K-length shift registers whose outputs are selectively modulo-2 summed by n adders. An example of an encoder with $k = 1$ and $K = 3$ and $n = 2$ is seen in Figure 11.

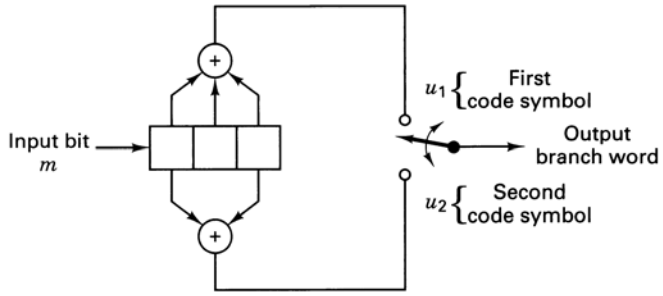

Figure 24. $k=1,\ K=3,\ n=2$ convolutional encoder [From Ref. 7].

The output of this encoder will have 2 coded bits for every input data bit. Three parameters characterize a convolutional encoder. First the coding rate r given by:

$$r=\frac{k}{n}. \tag{3.24}$$

In other words, r expresses the decrease in the information rate. The next is its constraint length, which is usually defined as the length of the shift registers K [7, 12]. The last parameter is its free distance d_{free}, which is a measure of the code's performance and will addressed further in Chapter V. The above example is therefore a ½-rate convolutional encoder of constraint length 3. As the encoder configurations become very large, being able to indicate how the registers and modulo-2 summers are connected is very important. This is done by defining a connection vector. In the above example, the connection vector would be:

$$g_1=111 \quad g_2=101. \tag{3.25}$$

This indicates that the first code symbol is generated by connecting the outputs of all three shift registers, while the second code symbol is generated by connecting only the first and third registers. These code vectors are often combined and expressed in their octal form. For example, the above code vector would become $g=[7,5]$. Optimal configurations have been developed for a variety of code rates and constraint lengths and are available from several references [7, 12, 17].

One point that impacts the practical implementation of a convolutional encoder is that the number of coded bits will be more than just r times the number of input bits.

This is because we also need to add "flush bits" to the input so that the last data bit will be pushed all the way through the shift registers. This means that the number of channel bits will be:

$$\text{number of channel bits} = r \times \left[N_{input\,bits} + (K-1) \right] \tag{3.26}$$

In the case of Seaweb in which the utility packets are only 72 bits, the total number of coded channel bits, for a ½-rate constraint length 9 coder, is $2(72+8) = 160$ bits.

To better understand how the convolutional decoder works, seeing what the output of the encoder looks like is important. Using the encoder described above and for an input bit stream 1 1 0 1 1, the output of the decoder is seen in Table 2 below. The state is defined as the value of the last $K-1$ (2 in this example) shift registers. It is assumed that the registers are initialized to 000.

i	input bit at t_i	registers at t_i	state at t_i	output at t_i
1	1	100	00	11
2	1	110	10	01
3	0	010	11	01
4	1	101	01	01
5	1	110	10	01

Table 2. Register contents, states and output code words for a given set of inputs, using the convolutional coder in Figure 24 above [After Ref. 7].

The relationship between inputs, states and outputs shown in Table 2 above can also be shown graphically as a path through a trellis diagram seen in Figure 25. The trellis shows all possible states and code words for a given series of inputs. The states are indicated by the letters "a" through "d" and the input bit is indicated by either a dashed or a solid line and the output code word is given by the two-bit word indicated on the codeword branch.

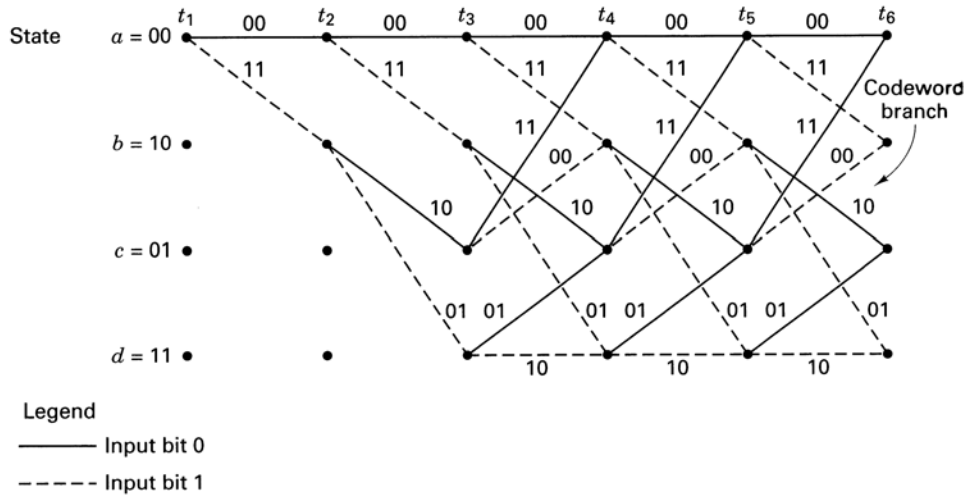

Figure 25. The trellis diagram for a $K = 3$, $k = 1$, $n = 2$ convolutional encoder [From Ref. 7].

To explain the trellis diagram we start at time t_1. The state at t_1 is $a = 00$. If a 1 were input, the state would transition, along the dashed line, to $b = 10$ and the code word would be 11. If a 0 were input, the state at t_2 would remain at $a = 00$ and the output code word would be 00. From Table 1 above, we see that the first input is a 1; therefore, the output is 11 and the new state is $b = 10$. This same process can be performed for all input bits as they cycle through the encoder.

3. Hard and Soft Decision Decoding

Once the coded sequence has passed through the channel, it must be decoded. This is performed by comparing what the received data sequence is, to what it should be, as predicted by the decoder. Several algorithms exist to do this decoding. One of the most common methods in the technical literature is the Viterbi algorithm, which can be used for both hard decision decoding (HDD) and soft decision decoding (SDD).

In HDD the input to the decoder is either a 1 or 0. This decision is made at the output of the demodulator. For example, in the case of a binary modulation schemes (i.e., BPSK) a negative voltage at the output of the demodulator, whether small or large, would

be detected as a zero. In the case of SDD, the demodulator does not make the decision. Instead the output voltage is quantized into 2^n levels and represented by n bits and sent through the soft decision decoder. We will restrict ourselves to describing HDD. For further information on SDD, the interested reader is directed to references such as, [7, 12, 18] which give in-depth information and analysis of both coding schemes and their variants.

Providing an example is the easiest way to explain how HDD works. Using the same encoding example from Section 2 above, the top of Figure 26 shows the input data sequence, the transmitted code word and one possible received sequence where the fourth received code word is in error.

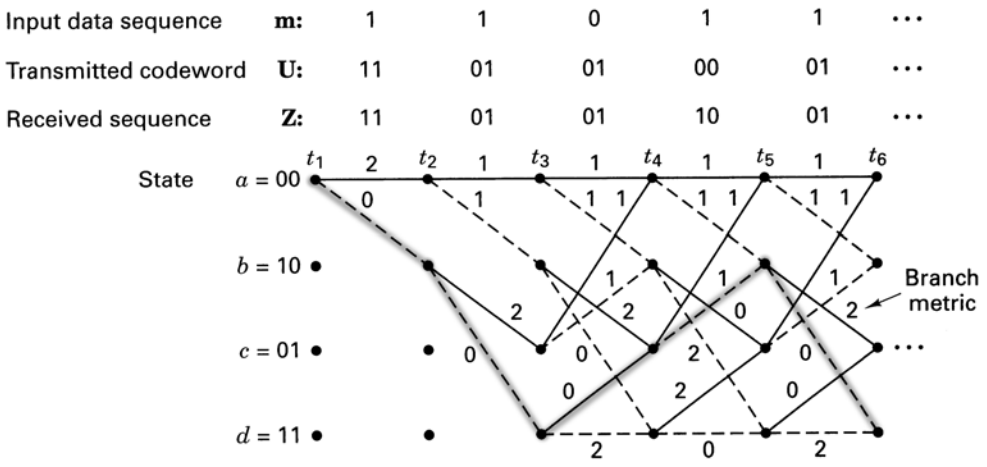

Figure 26. The trellis diagram for a $K = 3$, $k = 1$, $n = 3$ convolutional decoder using HDD [From Ref. 7].

The Viterbi algorithm finds a path through the trellis that the original coder "most likely" took. In HDD, "most likely" means the path with the smallest cumulative Hamming distance. The Hamming distance between two code words is the number of positions in which they differ.

Starting at t_1, we note that if a 1 was sent, the received sequence should be 11, which it is; therefore, the Hamming distance between 11 and 11 is 0. This difference is

noted on the branch as the branch metric. If instead a 0 had been sent, the received word should have been 00, it was 11; therefore, the Hamming distance between 00 and 11 is 2, which is also on the branch. This same procedure is carried on through the trellis with the Hamming distance marked on each branch. The cumulative Hamming distance for all paths is found and the surviving path with the smallest cumulative Hamming distance is chosen. In fact the Viterbi algorithm does not calculate the cumulative Hamming distance for all paths but will drop paths along the way, thus decreasing the computational load and preventing the problem of several paths having the same sum at the end. In the above case, the "most likely" path is highlighted in gray and has a cumulative sum of 1. The correct decoded output can then be determined by tracing back along the winning path using the convention that a 1 is represented by a dashed line and a 0 by a solid line. The decoded output for the above example is therefore 1 1 0 1 1.

As previously mentioned, the approach for SDD is slightly different, but it still uses the principle of finding the most likely path through the decoding trellis. Although SDD has typically 2 to 3 dB better performance than HDD, it is more computationally expensive.

4. Block Interleaving

Convolutional codes are designed to correct random independent errors. Errors that result from channel interference (whether it is channel-induced ISI due to multipath fading or hostile jamming) are not random and often occur in bursts. Likewise, errors in DBPSK occur in pairs, which also are not random. Convolutional coders do not perform well with these non-random errors. Therefore a block interleaver is used in most communication systems to help randomize these errors.

A block interleaver shuffles the convolutionally encoded bits so that successive bits are separated from each other. This is performed by reading the encoded bits into rows of an $M(rows) \times N(cols)$ matrix and then reading the data out column by column. In the receiver, the output of the demodulator is de-interleaved by reading the data in by columns and out by rows. This ensures that each input bit is separated by a distance of

N bit periods from adjacent bits before entering the channel. Therefore, channel burst errors of up to N bits long will appear as random single bit errors at the input of the convolutional decoder. The effects of the block interleaving are seen in Figure 27.

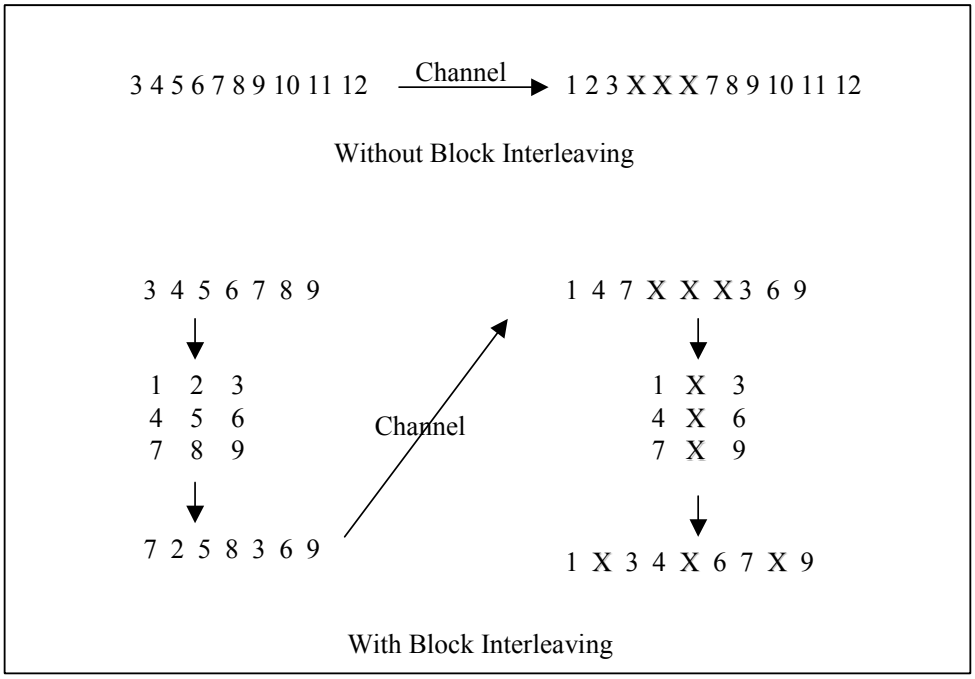

Figure 27. The effects of block interleaving (channel errors are indicated by an X).

F. PERFORMANCE ANALYSIS OF COMMUNICATIONS SYSTEMS

In general, when discussing the performance of signal processing methods, the commonly used figure of merit is the *average signal power to average noise power ratio,* S/N or SNR. In digital communications, it is more common to use E_b/N_0 as the figure of merit where E_b is the energy per bit and N_0 is the noise power spectral density. These two ratios are related by:

$$\frac{E_b}{N_0} = \frac{S\,T_b}{N/W} = \frac{S\,W}{N\,R_b} \tag{3.27}$$

where W is the channel bandwidth and $R_b = 1/T_b$ is the bit rate. Therefore E_b/N_0 is just a normalized version of SNR, normalized by the signal bit rate and bandwidth.

The most important metric used in comparing the performance of different digital communication methods and receiver processing algorithms is the probability of bit error P_b as a function of E_b / N_0. Theoretical expressions for P_b versus E_b / N_0 are widely available in the technical literature for all modulation schemes and processing techniques used in this thesis.

In practice, Monte Carlo simulations are performed to estimate P_b for a specific modem design. This involves sending random data bits through the simulated system and measuring the number of bit errors generated at the receiver output. The bit errors are determined by comparing the output decoded data sequence with the original information sequence over many packets. The bit error rate (BER) is then found using the expression:

$$BER = \frac{total\ bits\ in\ error}{total\ bits}$$

The BER should correspond well with the expected P_b from theory as long as sufficient statistics are used.

In this chapter, we have presented the reasons for choosing DS-IQ-DBSK as the modulation scheme for the experimental Seaweb modem. Also, overall block diagrams for generating and detecting DS-IQ-DBSK signals were developed. In the next chapter we will present the specific design parameters used by the modem to meet the Seaweb design requirements.

IV. MATLAB IMPLEMENTATION OF THE SEAWEB TRANSMITTER AND RECEIVER

This chapter presents the MATLAB implementation of the experimental Seaweb modem design, discussed in Chapter III. Block diagrams of the transmitter and receiver structures were shown in Figures 19 though 23. Each block is discussed separately here, with particular emphasis on the specific parameters used in the design. A software flow diagram as well as a description of MATLAB code is presented in Appendix A as a user's manual. Also, a full copy of the source code is included in Appendix B.

First we recap the Seaweb design requirements. The modem is required to send 72-bit utility packets using DSSS. It must have a maximum bandwidth of 5 kHz with a center frequency near 12 kHz. A ½-rate convolutional coder of constraint length 9 is implemented using soft decision decoding and 4-bit quantization. The bit rates can be in the tens of bits per second and minimal complexity is desired so the system can be implemented on a Digital Signal Processing (DSP) chip operating at an energy-conserving power state.

A. TRANSMITTER

The DS-IQ-DBPSK signal is generated using the block diagram seen in Figure 19 developed in Chapter III. The 72 information bits in the utility packet are channel encoded, differentially encoded and then passed through both an I and Q channel where they are chipped, pulse shaped and modulated. The two channel outputs are summed together to form one data frame and an acquisition frame is then appended to the start of the data frame. The resulting signal is sent through the simulated channel.

1. Channel Coding

Channel coding is performed using both a convolutional coder and block interleaver as described in Chapter III. A rate ½-code with constraint length 9 is used,

which transforms the 72 information bits in the utility packet into 160 channel bits. An optimal code connection vector [753, 561] was taken from [17].

Both hard and soft decision Viterbi decoding are performed on the received data. Since the Seaweb specification allows for a 4-bit word, $2^4 = 16$ quantization levels are used over an interval of $[-\pi, \pi]$. MATLAB's communications tool box functions are used to perform the coding and decoding functions.

Since there are 160 channel bits, the block interleaver matrix is 16×10. This means that at the output of the interleaver, the input bits are separated by ten bits. The 160-bit sequence is differentially encoded resulting in 161 channel bits. The encoded sequence is then passed through an I and Q channel.

2. Modulation

Two different 2047 length Gold codes are used as the PN sequences to chip the data, one for the in-phase and the other for the quadrature-phase channel. These codes are taken from work done by [19] and the actual sequences were downloaded directly from his web site. Since the bandwidth of operation for Seaweb is only $W = 5$ kHz, the maximum T_C is found to be: $T_C = 2/W = 2/5 \text{ kHz} = 0.0004 \text{ s} = 0.4 \text{ ms}$. This is equal to a chipping rate $R_C = 1/T_C = 2500$ chips per second (cps).

In the actual implementation, however, a lower chipping rate is used to provide more flexibility in the channel encoded bit rates that could be implemented. At 2500 cps only bit rates 10, 50 and 100 bits per second (bps) allow for an integer number of chips per bit. However, 2400 cps bit rates of 10, 20, 30, 40, 50, 60, 80 and 100 can be used. Simulation results for a channel encoded bit rate of 40 bits per second (i.e., an information bit rate of 18 bits per second) are presented in Chapter V. An $R_C = 2400$ cps is equal to a null-to-null bandwidth $B_{nn} = 4800$ Hz and a chip duration of $T_C = 0.417$ ms.

After chipping the data, the signal is up-sampled to a sampling frequency $f_S = 48\,kHz$. This exceeds the required Nyquist rate of:

$$f_{Nyquest} = 2f_{\max} = 2(f_c + \frac{R_C}{2}) = 2(12\text{ kHz} + 2.4\text{ kHz}) = 28.8\text{ kHz} \qquad (4.1)$$

At this sampling rate and for a bit rate of 40 bits per second, there are 20 samples per chip and 800 samples per bit.

The output of the up-sampled signal is passed through the pulse-shaping filter. This is a digital lowpass FIR filter of order $n = 64$ and is based on a weighted equi-ripple filter design. The pass-band frequency is $f_{pass\,band} = 2.5$ kHz and the stop band is $f_{stop\,band} = 3$ kHz. The low filter order is chosen so that it can be more easily implemented in hardware. The magnitude response of the filter is seen in Figure 28.

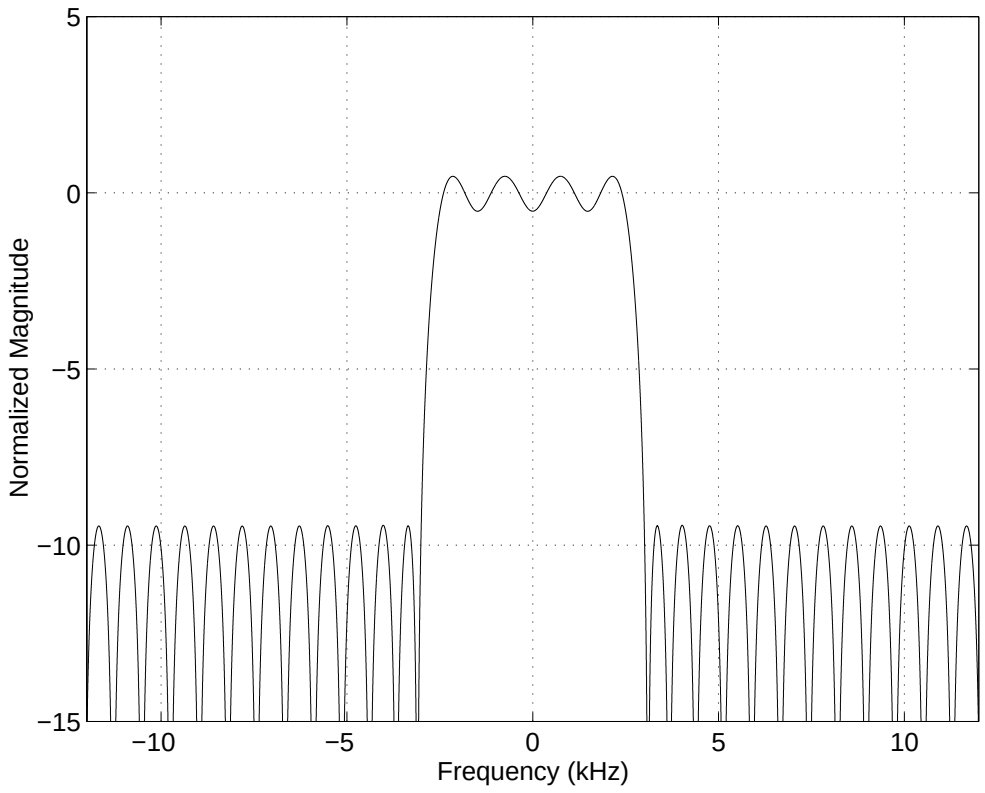

Figure 28. Magnitude response of the pulse-shaping filter.

The resulting effect, in the frequency domain, on the chipped data signal, is shown in Figure 29.

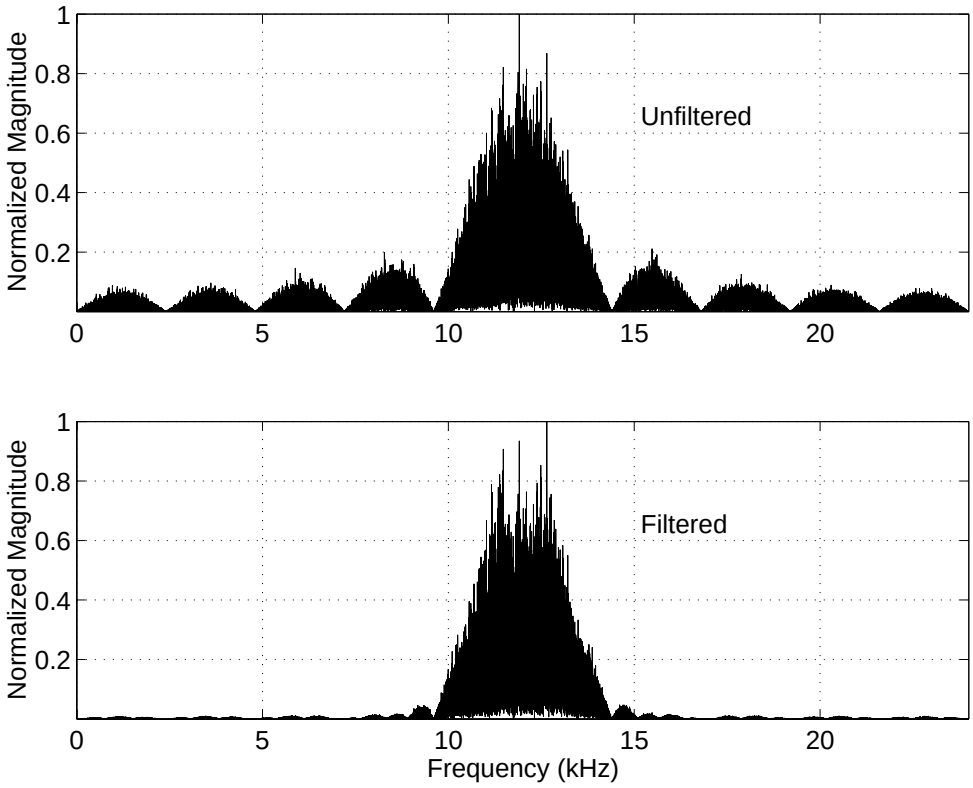

Figure 29. Effect of the pulse-shaping filter on the frequency-domain magnitude response of the transmitted signal, where the upper plot represents the unfiltered signal and the lower plot represents the filtered signal.

The output of the pulse-shaping filter is modulated by both an in-phase and quadrature-phase carrier, as detailed in Chapter III. The carriers have a center frequency of $f_C = 12$ kHz and amplitude $A_C = 1$. The two channels are then summed to generate one signal.

At this point in the simulation, the signal power S is determined from the expression:

$$S = \frac{\sum |x[n]|^2 T_s}{L T_s} = \frac{\sum |x[n]|^2}{L} \tag{4.2}$$

where $x[n]$ is the amplitude of each of the samples in the modulated signal and L is the total number of samples. Without pulse-shaping $S = A_C^2 / 2 = 1/2$, but with pulse-shaping this is no longer true. Instead it is observed from the simulation that $S \simeq 0.455$. This is consistent with the frequency domain observations in Figure 29. The power contained in the main lobe of a *sinc* function is 90.3 percent of the total signal power. Since the pulse-shaping filter has a bandwidth just a little wider than the null-to-null bandwidth of the signal, the filter will pass just over 90.3 percent of the power.

3. Acquisition Frame

After modulation, an acquisition frame described in Chapter III.C.6.a is appended to the start of the data packet. This frame is used to later acquire the signal in the receiver. The three pulses are 240 chips long and are separated by a period of 200 chips. The chipping sequence used is the same as the one used to chip the data. The total length of the frame is therefore 1320 chips, 26400 samples or 0.55 seconds.

A representation of the overall transmitted signal is seen in Figure 30.

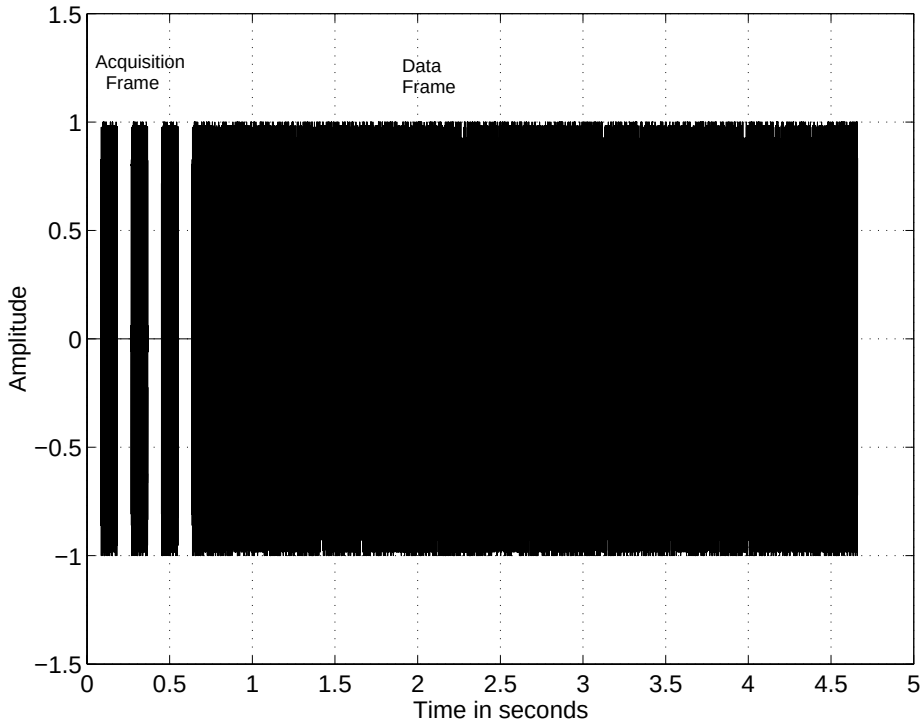

Figure 30. Transmitted signal in the time domain showing the acquisition and data frame components in the utility packet for bit rate of 40 bps.

B. CHANNEL MODELS

Two channel models were implemented. The first is an ideal AWGN channel and was chosen in order to compare the simulation results to well-known theoretical performance of a DS-IQ-DBPSK signal. The second channel is generated using an underwater acoustic propagation model. Doppler spreading is not considered in any of the channels and only AWGN is simulated. There is no simulation of non-Gaussian sea noise.

1. Ideal AWGN Channel

The AWGN channel is ideal in the sense that there is no multipath spread and only noise is added to the signal. The received signal $r(t)$ (i.e., the input signal to the

receiver) can therefore be expressed as:

$$r(t) = s(t) + n_{AWGN}(t). \tag{4.3}$$

In the simulation, the noise to be added to the signal is defined by the user and, for the purposes of our analysis, is called $E_b / N_{0\ desired}$. The noise power σ^2, that must then be added to the signal to generate this specified E_b / N_0, can be calculated from the expression:

$$\sigma^2 = N = \frac{N_0}{2T_S} \tag{4.4}$$

where T_S is the sampling period. From:

$$N_0 = \frac{E_b}{E_b / N_{0\ desired}} \tag{4.5}$$

and from Equation 3.2:

$$E_b = \frac{A_C{}^2}{2} T_b = S\ T_b \tag{4.6}$$

where S is the signal power determined from Equation 4.2, the noise power spectral density is obtained from

$$N_0 = \frac{S\,T_b}{E_b / N_{0\ desired}}. \tag{4.7}$$

The noise power to be added to the signal is then determined using Equation 4.4 above.

As an example, for a bit rate of $R_b = 40\,\text{bps}$ and a desired $(E_b / N_{0\ desired}) = 0$ dB the signal power in the data frame portion of the signal as returned by the simulation is $S \simeq 0.455$. Therefore N_0 from Equation 4.7 above is:

$$N_0 = \frac{0.455 \times \frac{1}{40}}{10^{\frac{0}{10}}} = 0.011375 \tag{4.8}$$

and the noise power σ^2, from Equation 4.4 above is therefore:

$$\sigma^2 = \frac{0.011375}{2\ (1/48000)} = 273. \tag{4.9}$$

The SNR in this case becomes:

$$\frac{S}{N} = \frac{S}{\sigma^2} = \frac{0.0455}{273} = 0.00167 = -27.8 \text{ dB}. \tag{4.10}$$

If the bit rate were halved, then T_b would be doubled and the resulting SNR would decrease by 3 dB for the same defined $E_b / N_{0\ desired}$. This demonstrates two principles of DSSS discussed in Chapter II. First, for a fixed chipping rate, slower bit rates will have more processing gain. Second, this higher processing gain means that a lower SNR can be used and will better hide the signal in the channel noise. Table 3 below summarizes the relationships for various bit rates.

bit rate (bps)	chips per bit (cps)	SNR (dB)
20	120	-31.8
40	60	-27.8
80	30	-24.8

Table 3. Relationships between channel encoded bit rates, chip rates and SNR for a DSSS signal with a sampling rate of 48 kHz, chip rate of 2400 cps, and $E_b / N_0 = 0$ dB.

2. Modeled Multipath Channel

The second channel model is a multipath channel whose impulse response is generated using the *Bellhop* underwater acoustic propagation model [20]. *Bellhop* is a Gaussian ray model which, according to [10] and [2], are the class models best suited for broadband communication signals. The details of the model are not addressed here but the interested reader can find more information on *Bellhop*, including a users' manual, at [20]. Gaussian ray acoustic propagation models are also discussed in detail in [4].

Bellhop requires an input file that describes the physics of the channel. This includes the channel depth and depths of the transmitter and receiver. A sound speed profile of the water column also needs to be defined as this determines the refractive properties of the underwater channel. The specific input file used represents a real channel observed on May 10, 2002 in waters off San Diego, California, while conducting underwater communications experiments. The input file is shown in Figure 31.

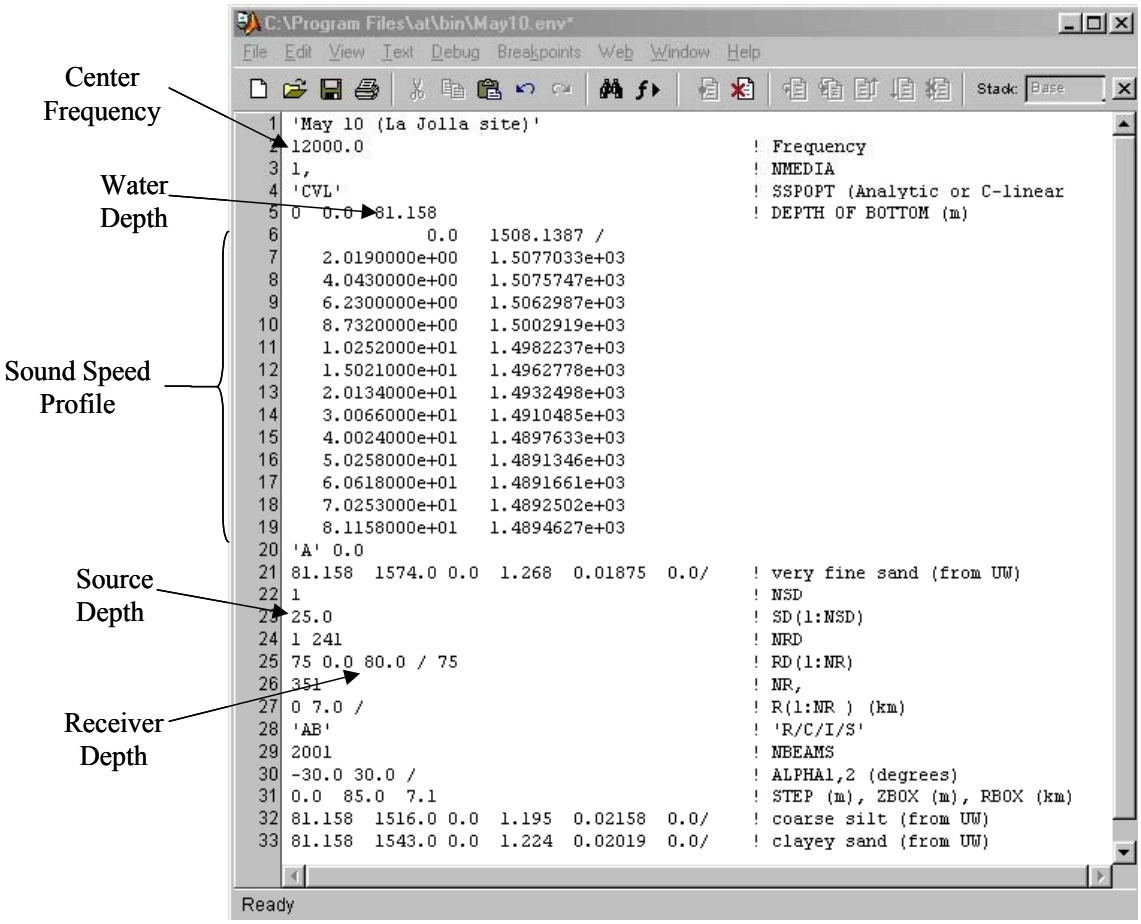

Figure 31. Input file for *Bellhop*, which describes the channel dimensions and physics.

The other parameters listed in the input file define the number of beams used in computing the channel response and the output range and depth resolutions. In addition, the sea bottom boundary layer properties are also defined.

The model generates the channel impulse response shown in Figure 32. This response is simplified to a series of delta-functions occurring at time intervals corresponding to the peaks in the impulse response and is shown in Figure 33.

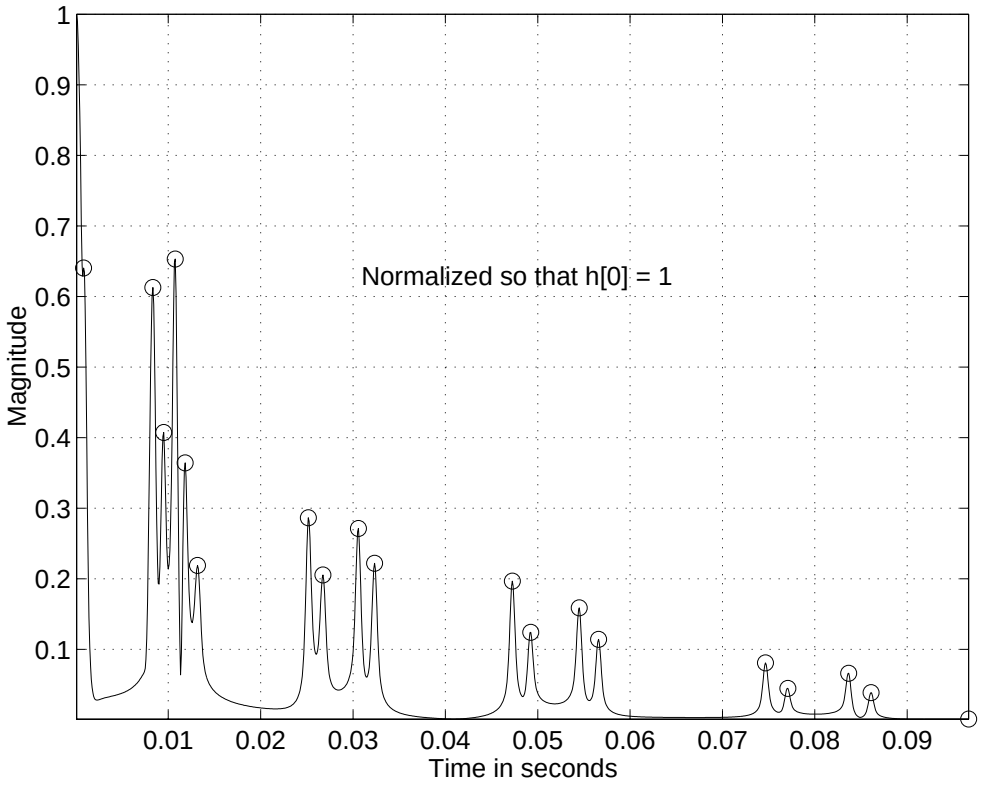

Figure 32. Normalized impulse response for modeled channel generated by *Bellhop* for a range of 4.0 kilometers between source and receiver .

We can see that this is a very challenging channel. First there are many multipath arrivals and the time spread is almost 100 milliseconds. However, most of the energy arrives within the first 15 milliseconds. Another observation is that all arrivals are more than T_C seconds apart and are therefore resolvable.

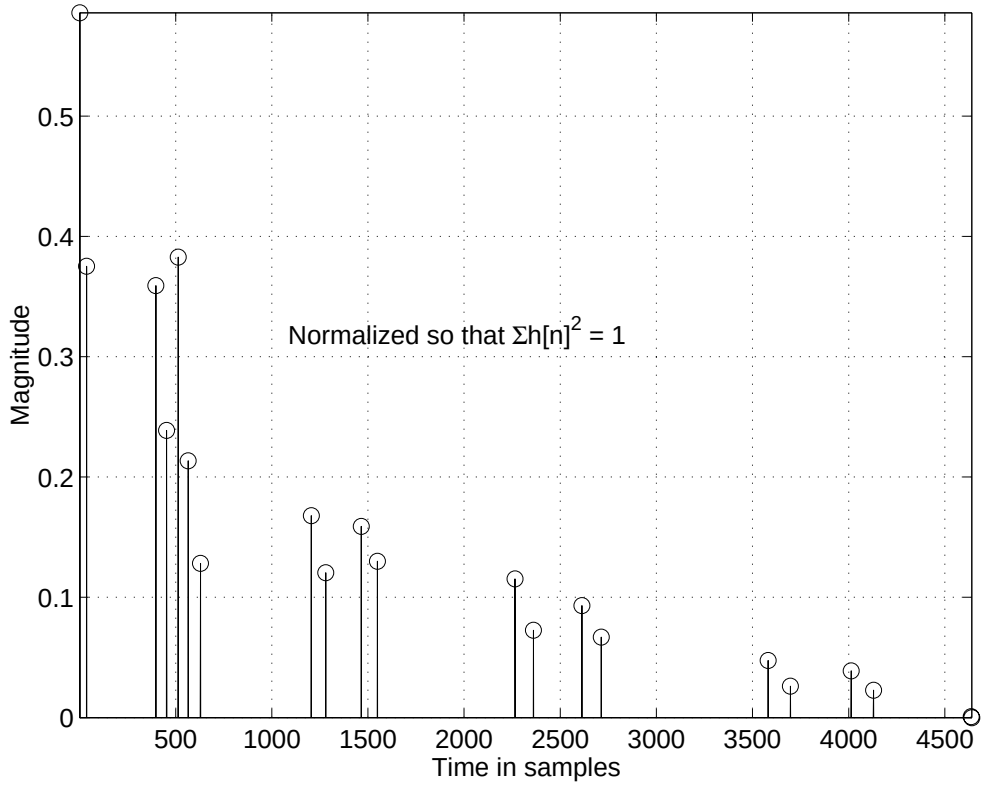

Figure 33. Simplified impulse response for modeled channel normalized to ensure energy conservation in the channel.

The simplified response in Figure 33 is used as the $h(t)$ for the channel simulation. It is also important to note that the impulse response must be normalized such that $\sum h^2[n] = 1$ [12]. This is done to ensure that the channel does not add or remove energy by artificially amplifying or attenuating the signal. AWGN is added in the same manner as the AWGN channel. The output of the channel is determined by convolving the transmitted signal with the channel impulse response and the received signal is given by:

$$r(t) = h(t) \otimes s(t) + n_{AWGN}(t). \tag{4.11}$$

C. RECEIVER

A DS-IQ-DBPSK receiver was implemented using the block diagrams developed in Figures 20, 22 and 23 of Chapter III.

Acquisition is achieved using a matched filter to detect the three short PN sequence acquisition frame described in Chapter III.C.6.a. The matched filter output for an ideal channel (i.e., no noise or multipath) is seen in Figure 34.

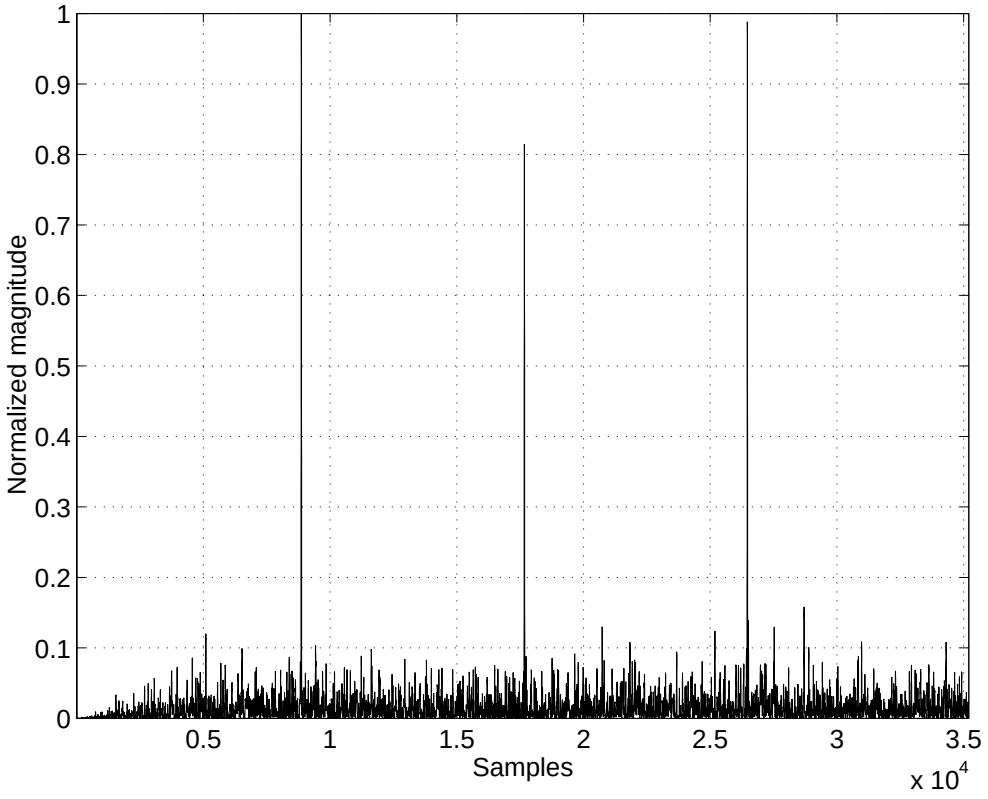

Figure 34. Acquisition frame and the Matched Filter output for an ideal channel (i.e., no noise, no multipath).

The start of the data frame is determined from the average distance between the correlation peaks in Figure 34.

The non-coherent DLL is implemented as detailed in Chapter III.C.6.b. The early and late delays are kept as $\pm T_C / 2$. Consequently, any misalignments of more than half a

chip (or 10 samples) will result in a loss of track and a loss of the signal. The loop filter is implemented as weighted averaging of the present shift $s[n]$ with the previous shift $s[n-1]$ as given by:

$$s_{filtered}[n] = (1-a)\cdot s[n] + a\cdot s[n-1] \tag{4.12}$$

where a is the weighting factor. A weighting factor of 0.9 is used in the simulations. A plot of the S-curve for the implemented DLL is shown in Figure 35.

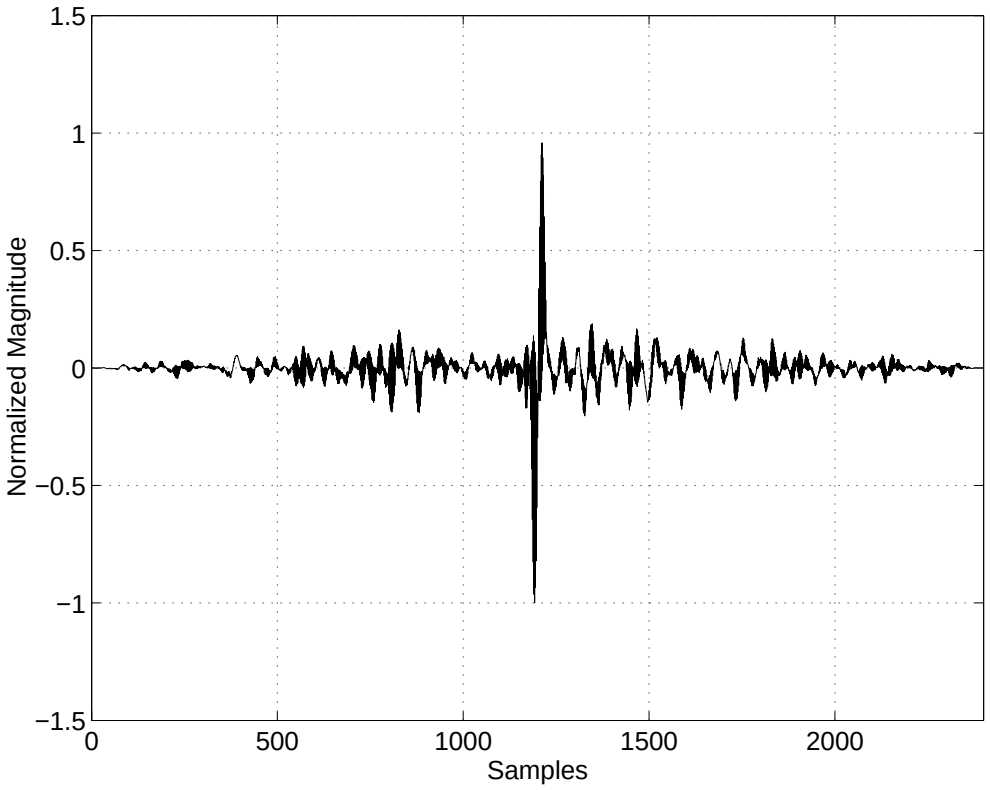

Figure 35. S-Curve of the DLL for an ideal AWGN channel (this example is for an $E_b / N_0 = 18\,\text{dB}$ ($SNR \simeq -10\,\text{dB}$))

After achieving synchronization, the received signal is demodulated using the DS-IQ-DBPSK receiver seen in Figure 21 of Chapter III. A typical signal constellation at the output of the receiver, but before differential decoding, is shown in Figure 36.

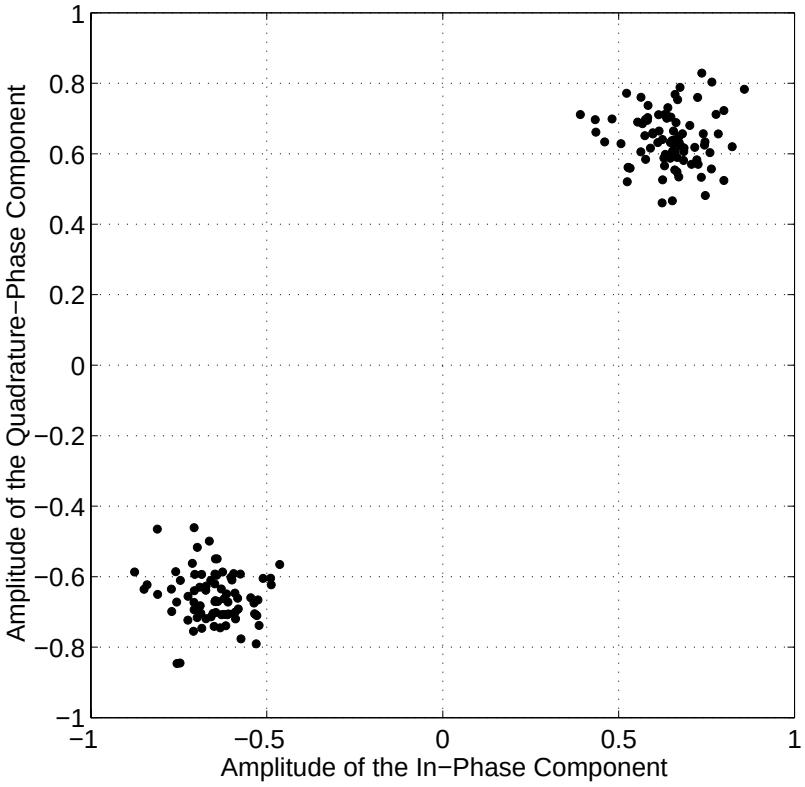

Figure 36. Typical received signal constellation, after despreading and before differential decoding. in the ideal AWGN channel (this example is for an $E_b / N_0 = 18\,\text{dB}$ ($SNR \simeq -10\,\text{dB}$)).

The RAKE portion of the receiver is implemented using both a fixed-tap and adaptive tap design as discussed in Chapter III.D. In the fixed-tap case, thirty taps are implemented and spaced at fixed intervals of T_C or 20 samples apart. This is sufficient to process the first 12.5 ms of multipath arrivals. The tap thresholds are set at 0.3. This means that, if the main path arrival (i.e., the signal at the first tap) has a magnitude of M, then only those taps that have a signal component greater than $0.3M$ are summed together. All other tap outputs are set to zero. In the case of the adaptive tap implementation of the RAKE, the tap locations were artificially supplied to the receiver, based on the locations of the first five multipath arrivals. The demodulated and detected bits are compared to the original transmitted information bits to determine the resulting bit error rate.

In this chapter, we have presented the specific design parameters used in the experimental Seaweb modem and the modeled channel characteristics. In the next chapter, we will examine this modem's performance in both the ideal AWGN channel and the modeled underwater channel and compare these to theoretical results.

THIS PAGE INTENTIONALLY LEFT BLANK

V. THEORETICAL PERFORMANCE AND MONTE CARLO SIMULATION RESULTS

In this chapter, both the theoretical and simulated performance of DS-IQ-DBPSK are presented and discussed. In examining the theoretical performance, analytic expressions for the probability of bit error P_b versus E_b / N_0 are developed for AWGN channel and a Rayleigh fading channel. Expressions are also developed for P_b which incorporate the expected improvements in performance by using hard and soft decision error-correction coding in the case of AWGN.

For the Seaweb modem simulation, three different aspects of performance were examined. First, a plot of the BER versus E_b / N_0 was generated using Monte Carlo simulation on the two channels presented in Chapter IV. The Monte Carlo simulation involved sending an adequate number of packets through the transmitter and receiver to generate sufficient statistics. Sufficient statistics were taken to be enough bits or packets to generate error rates that are an order of magnitude greater than the theoretically predicted values. This plot is compared to the expected theoretical performance. Next, because the modem's purpose is to send utility packets, a plot of the probability of packet error P_p versus E_b / N_0 is generated.

A. THEORECTICAL PERFORMANCE OF DS-IQ-DBPSK IN AWGN AND RAYLEIGH FADING

1. Performance in AWGN

The expression for the probability of bit error for DBPSK whether sent on a single channel or on an I and Q channel is given as [12]:

$$P_b = \frac{1}{2}\exp\left(-\frac{E_b}{N_0}\right). \tag{5.1}$$

A plot of this expression is seen in Figure 37.

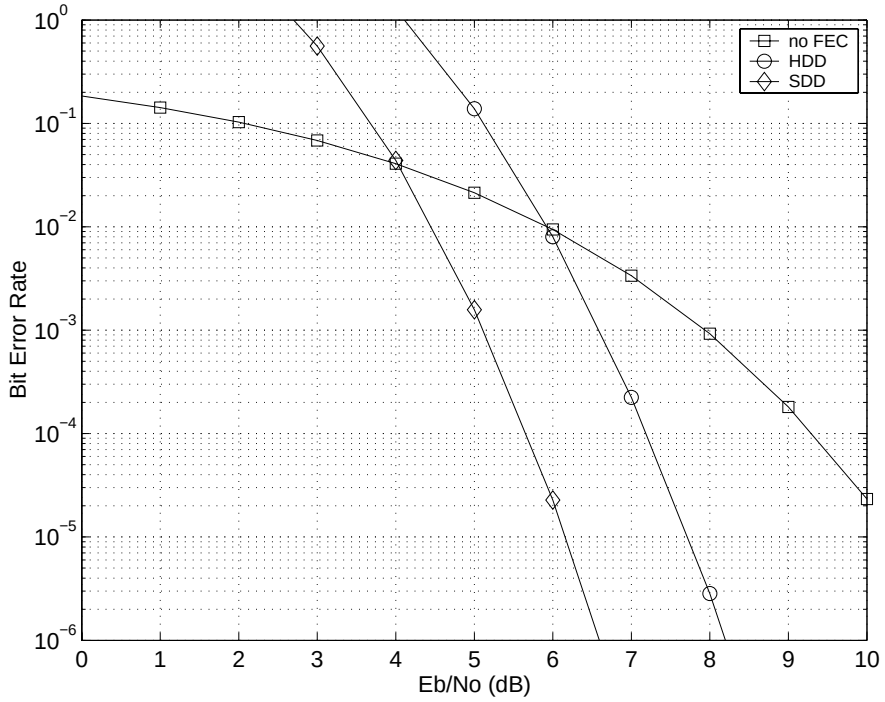

Figure 37. Theoretical bit error rates versus E_b / N_0 for DS-IQ-DBPSK in AWGN.

When a rate $r = k / n$ convolutional coding is used, an upper bound on the probability of bit error is given as [12]:

$$P_b < \frac{1}{k} \sum_{d=d_{free}}^{\infty} B_d P_d. \tag{5.2}$$

The free distance d_{free} is the minimum Hamming distance that can exist between the received signal and any branch in the decoding trellis. The parameter B_d is the sum of all possible bit errors that can occur when the all-zero code word is transmitted and a path of weight *d* is selected. Finally P_d is the channel transition probability. Typically, the first four terms in the above summation dominate and therefore need to be considered. The values for B_d and d_{free} are specific to the convolutional code chosen and are

available from tables and for our case [21] gives them as:

$$\begin{aligned} &k = 1 \\ &d_{free} = 12 \\ &B_d: \ B_{12} = 33, B_{13} = 0, \ B_{14} = 281, \ B_{15} = 0, \ B_{16} = 2179. \end{aligned} \tag{5.3}$$

In the case of HDD, P_d is independent of the modulation scheme chosen and can be expressed as [12]:

$$P_d = \begin{cases} \frac{1}{2}\binom{d}{d/2}\left[p(1-p)^{d/2}\right] + \sum_{i=1+d/2}^{d}\binom{d}{i}p^i(1-p)^{d-1} & \text{for } d \text{ even} \\ \sum_{i=(d+1)/2}^{d}\binom{d}{d/2}p^i(1-p)^{d-1} & \text{for } d \text{ odd.} \end{cases} \tag{5.4}$$

where p is the probability of bit error without encoding. When Equations 5.2, 5.3 and 4.4 are combined, the resulting P_b curve is also seen in Figure 37.

For SDD, P_d is dependent on the modulation technique. Although no expression was found for DBPSK specifically, [21] states that the performance of DBPSK (again whether sent on a single or an I and Q channel) is simply 3 dB better than noncoherent frequency-shift keying (NC-FSK) modulation when using SDD. The performance of NC-FSK is well published and [21] gives the expression for P_d as:

$$P_d = \frac{1}{2^{2d-1}}\exp\left(\frac{-drE_b}{2N_0}\right)\sum_{n=0}^{d-1}c_n\left(\frac{drE_b}{2N_0}\right) \tag{5.5}$$

where:

$$c_n = \frac{1}{n!}\sum_{m=0}^{d-1-n}\binom{2d-1}{m}. \tag{5.6}$$

This is the same as NC-FSK with d diversity receptions and, in the case of SDD, $d = d_{free}$ and r is the code rate. Combining Equations 5.5 and 5.6 and shifting the results left by 3 dB, the resulting P_b curve is seen in Figure 37.

Observing Figure 37, we can see three standard results from using FEC in an AWGN. First, the upper bound on the probability of bit error for both HDD and SDD is only valid for $P_b < 10^{-3}$. Second, at low values of E_b / N_0, performance with FEC can be worse than without FEC. Lastly, SDD is typically 2 to 3 dB better than HDD and in our case the difference is just less than 2 dB.

The theoretical packet error rates in all cases can be determined from the expression:

$$P_p = 1 - (1 - p)^N \tag{5.7}$$

where P_p is the probability of packet error, p is the probability of bit error for each bit in the packet and N is the number of bits per packet. The theoretical P_p curves for 72 bit packets in AWGN are shown in Figure 38.

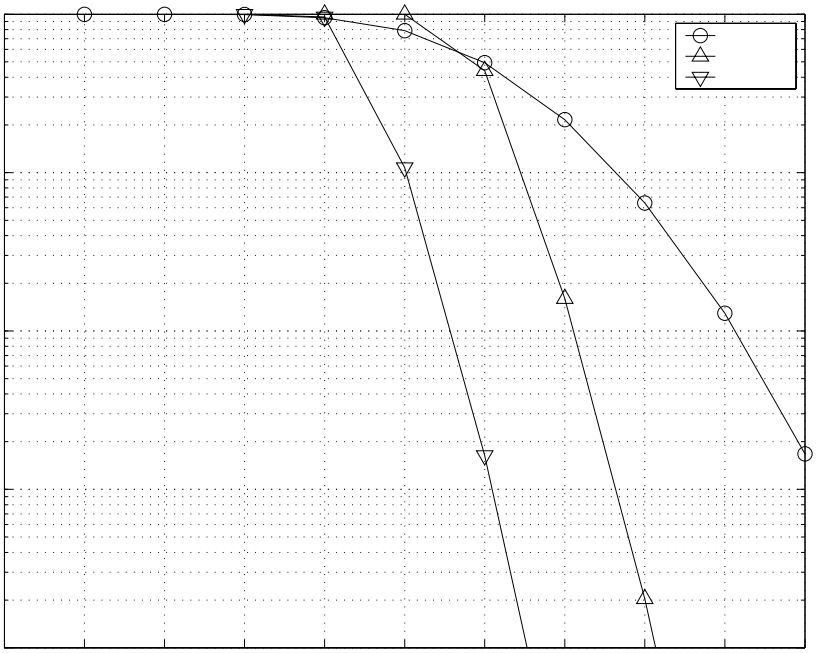

Figure 38. Theoretical packet error rates P_p versus E_b / N_0 for DS-IQ-DBPSK for 72 information bit packets in AWGN.

2. Performance in a Rayleigh Fading Channel

In a fading channel the received signal amplitude is no longer fixed but varies over time. Therefore the expression for the received DS-IQ DBPSK signal given in Equation 3.8 becomes:

$$s_{DS-IQ-BPSK}(t) = \frac{1}{\sqrt{2}} a_C \left[c_I(t)\cos[\omega_C t + \theta_i(t)] - c_Q(t)\sin[\omega_C t + \theta_i(t)] \right] \tag{5.8}$$

where a_C is modeled as a random variable. The new probability of bit error for IQ-DBPSK from Equation 5.8 is now dependant on the random variable a_C and can be expressed as:

$$P_B(\gamma_b) = \frac{1}{2}\exp[-\gamma_b] \tag{5.9}$$

where $\gamma_b = a_C{}^2 T_b / N_0$ and is a function of the random variable a_C. The two most widely used channel fading models are Ricean and Rayleigh. Rayleigh is a special case of Ricean where there is no direct or main path between the transmitter and receiver, i.e., all the received power is due to multipath. The Rayleigh model is used most often because it is the "worst case" and it has analytic expressions that are easier to solve. We will restrict our analysis to the Rayleigh channel.

For any function $g(x)$ of a random variable x, the expected value of the function is given by:

$$E[g(x)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx \tag{5.10}$$

where $f_X(x)$ is the probability density function (PDF) of the random variable. The PDF for Rayleigh fading is given as:

$$f_\Gamma(\gamma_b) = \frac{1}{\overline{\gamma}_b}\exp\left[-\frac{\gamma_b}{\overline{\gamma}_b} \right] u(\gamma_b) \tag{5.11}$$

where $\overline{\gamma}_b = \overline{a}_C{}^2 T_b / N_0$ and is analogous to E_b / N_0. Therefore the average (or expected) probability of bit error is:

$$\begin{aligned} P_b &= \int_0^\infty P_B(\gamma_b) f_\Gamma(\gamma_b) d\gamma_b \\ &= \int_0^\infty \frac{1}{2}\exp[-\gamma_b]\frac{1}{\overline{\gamma}_b}\exp\left[-\frac{\gamma_b}{\overline{\gamma}_b}\right] u(\gamma_b) d\gamma_b \\ &= \frac{1}{2(\overline{\gamma}_b + 1)} . \end{aligned} \tag{5.12}$$

A plot of this expression is seen in Figure 39 and compared to performance in AWGN.

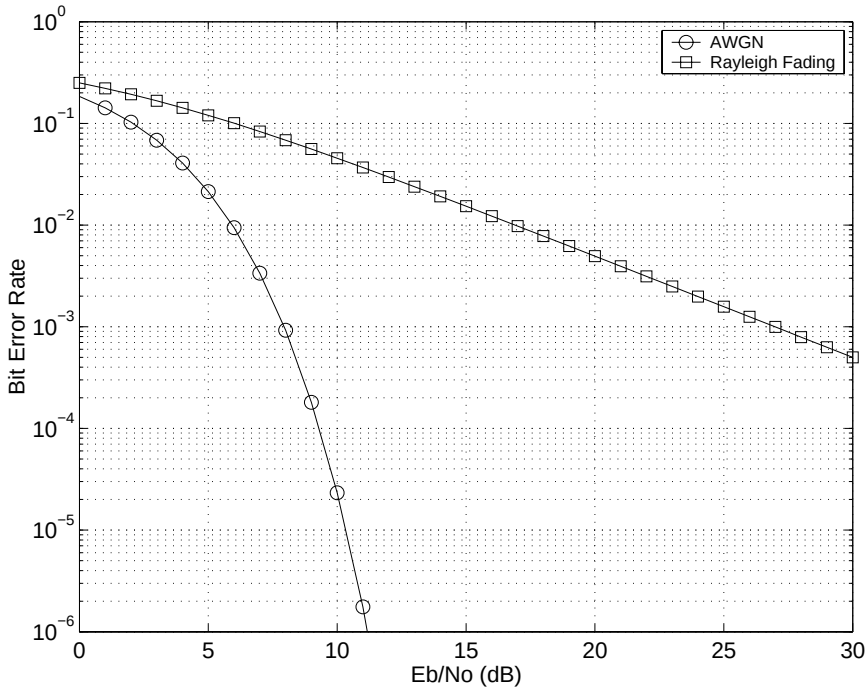

Figure 39. Theoretical bit error rates versus E_b / N_0 for DS-IQ-DBPSK in Rayleigh fading compared to performance in AWGN.

From Figure 39, we see that the performance of DS-IQ-BPSK is significantly degraded in a fading environment. Further expressions for the theoretical improvements in P_b when using HDD, SDD or diversity reception are not developed here but are available in [12].

B. SIMULATION RESULTS FOR AWGN CHANNEL

1. Receiver Performance – Bit Error Rate

Figure 40, 41 and 42 show the results of the Monte Carlo simulation for the experimental Seaweb modem in AWGN. Bit error rates are again plotted against E_b / N_0 for a bit rate of 40 bits per second. Although Seaweb requires only 72-bit packets (160 channel coded bits) data packets, performance is also analyzed for 1242-bit packets (2500 channel coded bits). In the case of 1242-bit packets, the block interleaver matrix becomes 50×50. The larger packet size is implemented in order to determine if the packet size affected the performance of either the HDD or SDD algorithms. For comparison purposes, the theoretical P_b also plotted. The performance curves for the simulation results in all figures simply connect the data points.

Figure 40 shows the results for the case when no forward error-correction coding is used in the receiver.

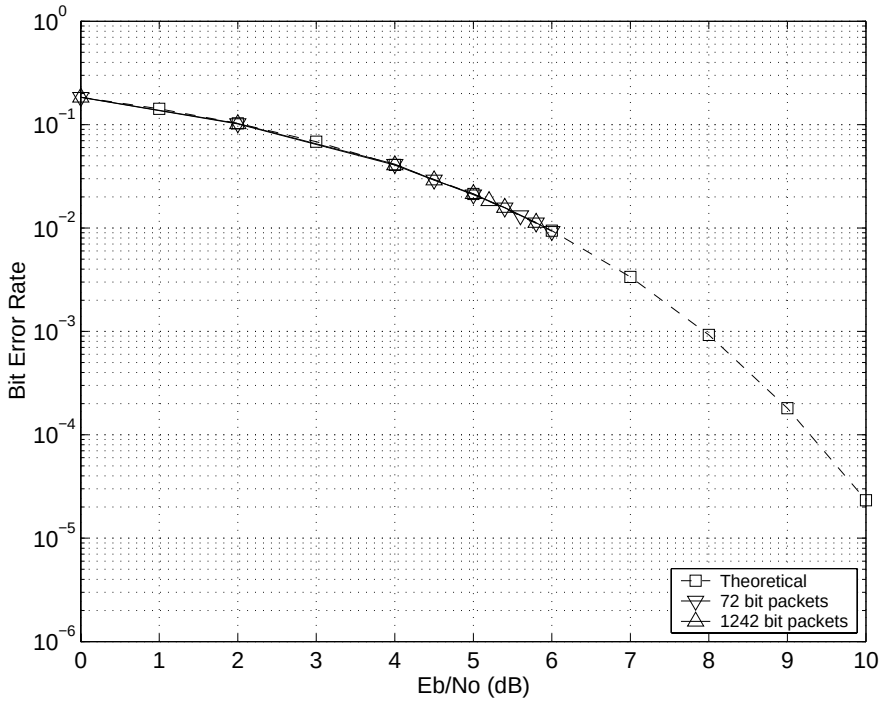

Figure 40. Bit error rate in an AWGN channel with no error-correction coding.

Two observations can be made from these results. First, the simulation results are quite close to those predicted from theory. This means that the experimental Seaweb modem implementation accurately generates and correctly demodulates the DS-IQ-BPSK signal. Second, the smaller packet size does not degrade performance.

Figure 41 shows the results for the case when HDD is used in the receiver. Three observations can be made from these results. First, the simulation BERs are below the theoretical upper bound curve as expected. Second, the BERs appear to asymptotically approach the upper bound limit as E_b / N_0 increases, again as expected. Finally we see that the longer 1242 bit packets have a slightly better performance than the shorter 72 bit packets.

Figure 42 shows the Monte Carlo simulation results for SDD. The results here are similar to those of HDD. The BERs for the simulation are below the theoretical upper bound but approach the bound as E_b / N_0 increases. We also see that at higher values of E_b / N_0 the larger packets have fewer errors than the short packets. The difference is small, on the order of 1 dB. Since Seaweb utility packets are constrained to the smaller packet size, we will confine our subsequent analysis to it.

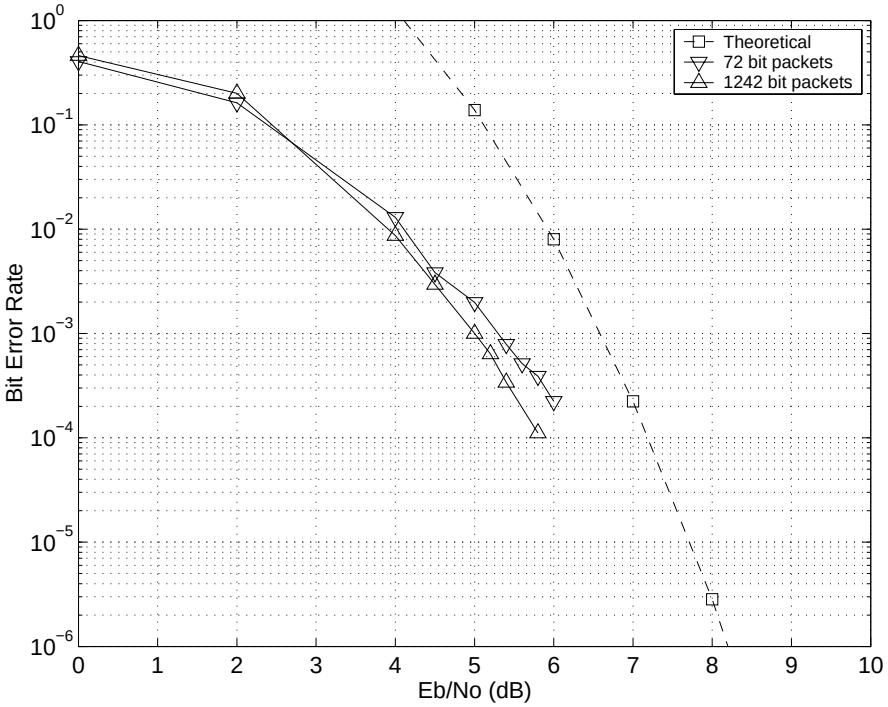

Figure 41. Bit error rate in an AWGN channel with HDD.

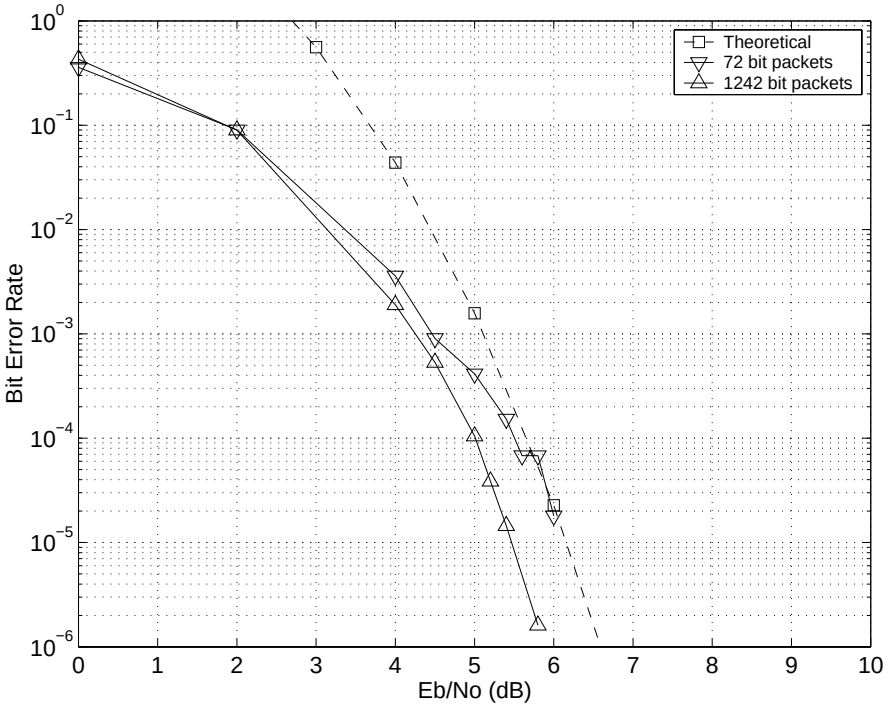

Figure 42. Bit error rate in an AWGN channel with SDD.

2. Receiver Performance - Packet Error Rate

Figures 43, 44 and 45, show the packet error rates for the AWGN channel. Only the results for the 72 bit packets are shown. In all three cases, we see that the simulation results are better than the theoretical results for E_b / N_0 between 2 to 6 dB. Also the simulation approaches the theoretical upper bounds of performance at higher values of E_b / N_0.

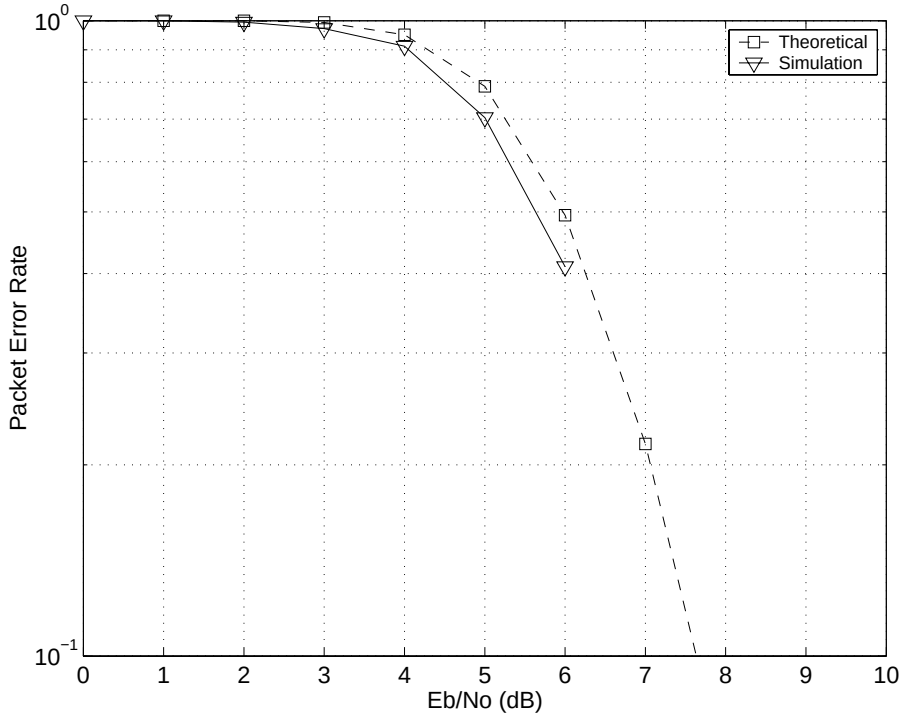

Figure 43. Packet error rate in an AWGN channel with no error-correction coding.

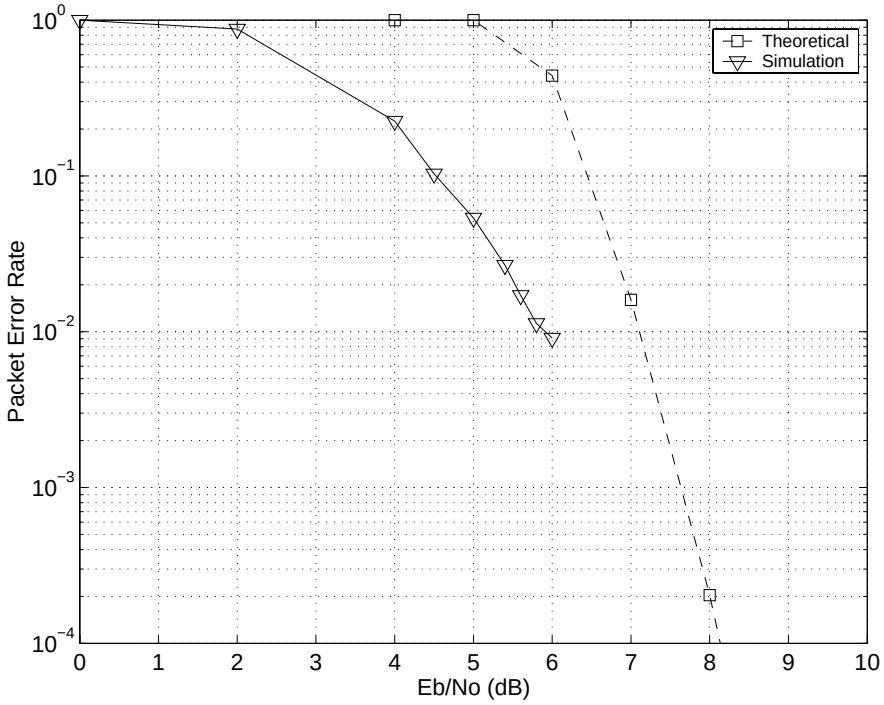

Figure 44. Packet error rate in an AWGN channel with HDD.

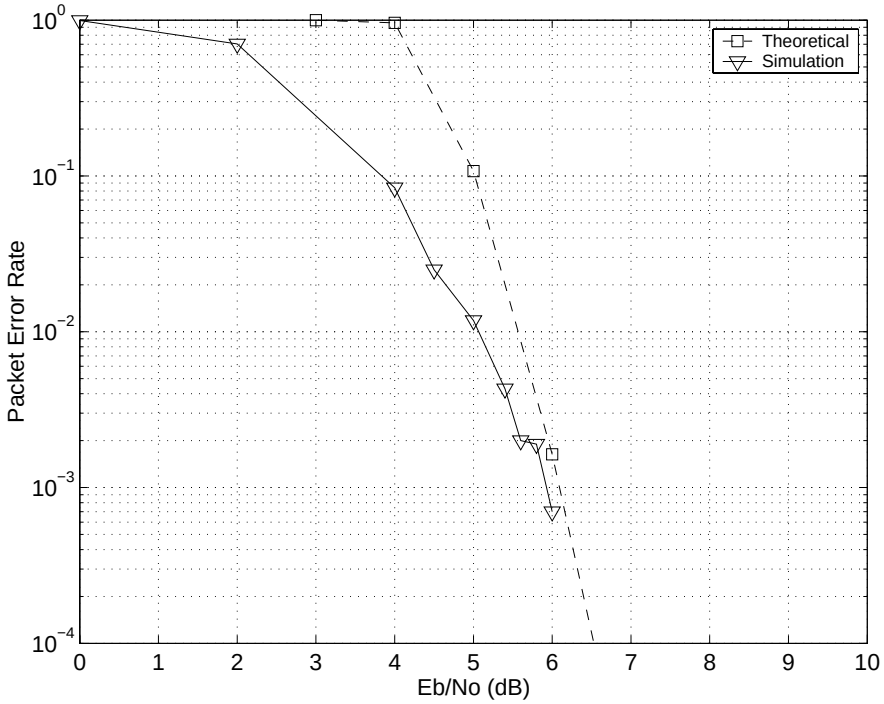

Figure 45. Packet error rate in an AWGN channel with SDD.

3. Performance of Synchronization Algorithms

One very important result to come out of the above simulations is that the acquisition block could not acquire the signal consistently for values of $E_b / N_0 < 6$ dB. In order to get results at these signal levels, the acquisition frame needed a much larger SNR than the data frame. To generate the results for Figures 40 to 45 the acquisition frame SNR was given an SNR $30\,\text{dB}$ larger than the data frame SNR. The value of E_b / N_0 in the figures is, however, based on the bit energy in the data frame only.

Having to send one portion of the utility packet at a much higher SNR than the other defeats the goal of trying to transmit LPD signals. As a result, the simulation was also run to show performance when the SNRs of the two frames are kept the same. Figure 46 shows this result for AWGN.

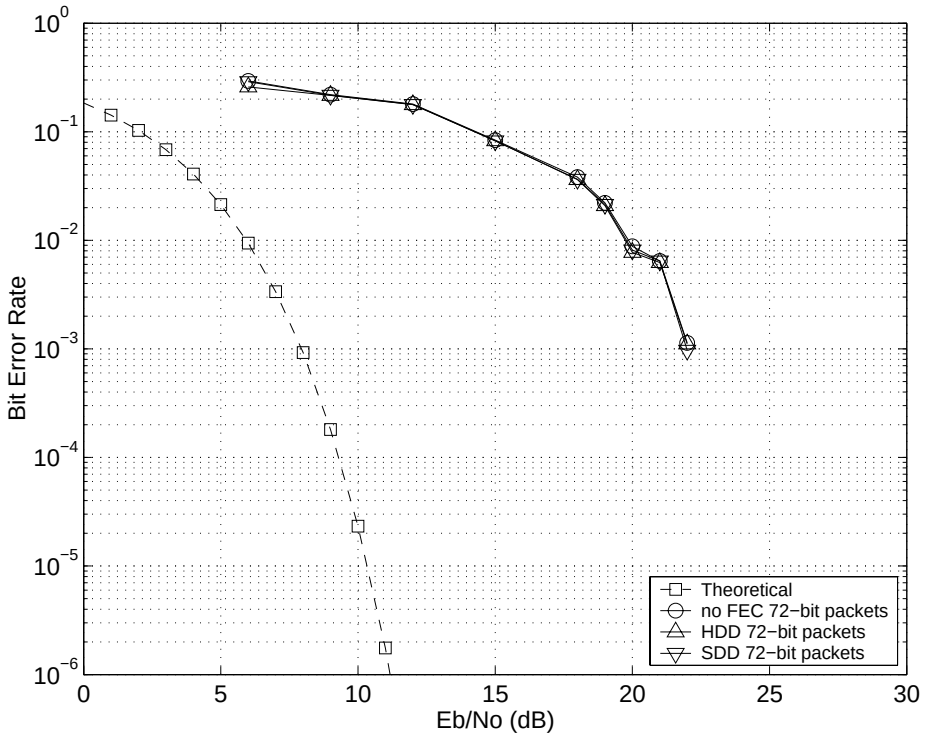

Figure 46. Bit error rate in AWGN when the acquisition frame and the data frame have the same SNR.

We see that this implementation causes severe degradation in performance. Again synchronization could not be achieved consistently for $E_b / N_0 < 6$ dB. For E_b / N_0 between 6 and 22 dB, BERs are still large. It is only above 22 dB that we lock onto the signal at which point, because the data SNR is very high, the error rate drops to zero. There were no significant differences in performance between any of the processing schemes. A closer look at the data packet-by-packet shows that SDD did perform slightly better than HDD.

4. Low Probability of Detection

Two techniques were used to assess the LPD properties of the signal in the AWGN channel. For the presence of the signal to be undetectable to an unauthorized intercept receiver, the transmitted signal should not be audible in the channel. It was determined that a transmitted signal with an $SNR < -6\,\text{dB}$ could not be audibly detected above the background noise, by the casual listener. Second the transmitted signal should not be visible by simply looking at the time or frequency domain of the transmitted signal in channel. Again for signals with an $SNR < -6\,\text{dB}$ the signals presence could not be detected. This means that for a 40 bps signal to be undetectable to an unauthorized interceptor yet still be detectable to the receiver, the E_b / N_0 threshold should be less than approximately 22 dB. In the AWGN channel $BERs < 10^{-5}$ are easily achievable below the LPD threshold. Other more sophisticated means are available to detect the signal (as discussed in Chapter III) but this is certainly a minimum threshold above which the signal's presence could easily be detected

C. SIMULATION RESULTS FOR THE MODELED SHALLOW WATER CHANNEL

To examine the performance of the simulated Seaweb modem in the modeled shallow water channel, three different modem configurations are used in the Monte Carlo simulations. In the first case, no RAKE receiver is used. In the second case, a RAKE

receiver with 30 equally spaced taps and a tap spacing of T_C is used to cover the first 12.5 ms of multipath arrivals. Lastly, a 5-tap RAKE is used where the taps are placed to coincide with the 5 largest multipath arrivals occurring within the first 12.5 milliseconds. The time index of the arrivals is artificially supplied to the receiver based on the channel impulse response. As in the case of the AWGN channel, when the acquisition frame and data frame SNRs are kept equal, performance is very poor. Simply increasing the SNR of the acquisition frame does not work in the case of the multipath channel. This is because the multipath interference caused by the acquisition frame overwhelms the lower SNR data frame causing excessive bit errors. Instead, initial results are obtained by artificially providing the start of the data frame to the receiver. Results for BERs and PERs are plotted versus E_b / N_0 for a channel encoded bit rate of 40 bits per second. For each simulation run, a maximum of 1000 packets were transmitted through the channel for E_b / N_0 ranging from 0 to 30 dB. In all cases, the theoretical performance of DS-IQ-DBPSK in a Rayleigh fading channel is also shown for comparison. The performance curves for the simulation results in all figures simply connect the data points.

1. Receiver Performance with No RAKE

Figure 47 and 48 show the performance of the experimental Seaweb modem receiver when no RAKE is used and the start of the data is artificially supplied to the receiver.

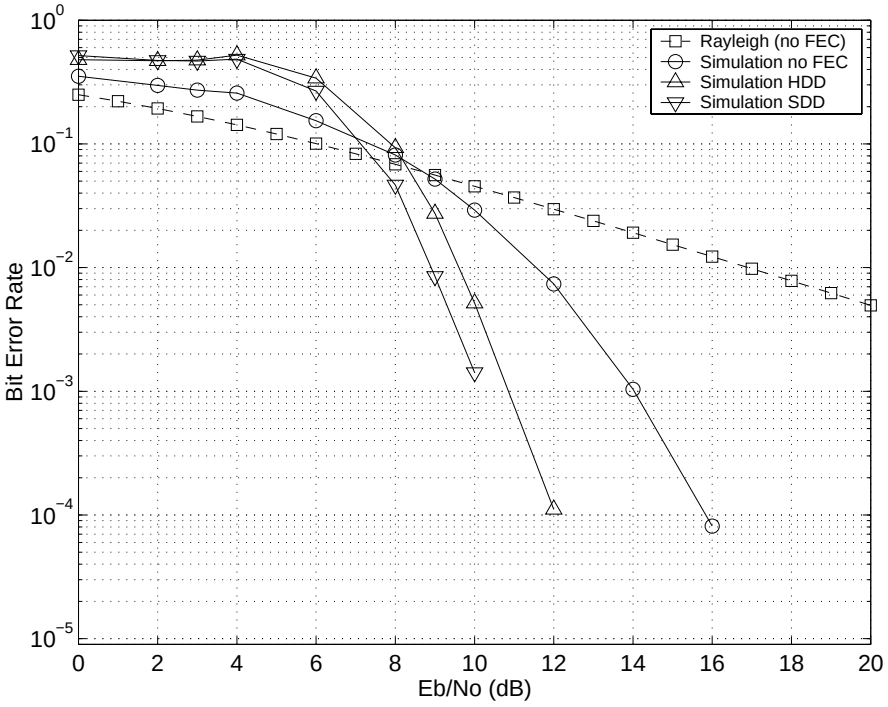

Figure 47. Bit error rate without using the RAKE receiver for the modeled channel.

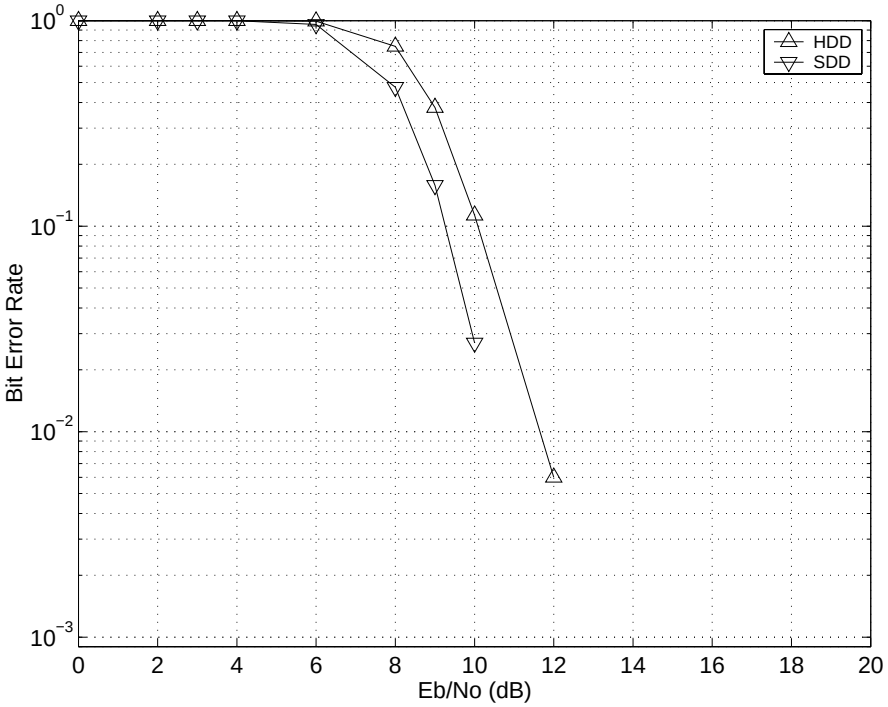

Figure 48. Packet error rate without using the RAKE receiver for the modeled channel.

From Figures 47 and 48, we see that the modeled channel is not as severe as a Rayleigh fading channel, this is because we have a large main path component. Also we see that HDD and SDD significantly improve performance for $E_b / N_0 > 8$ dB, with SDD providing the largest gain. It is important to note the last data point in each of the curves indicates that this was the last integer value of E_b / N_0 that resulted in any errors. For example in Figure 47, SDD corrected all the simulation errors for values of $E_b / N_0 \geq 11$ dB.

2. Receiver Performance Using RAKE with 30 Fixed Spaced Taps

Figure 49 and 50 show the performance of the experimental Seaweb modem receiver when 30 equally spaced taps are used in the RAKE and the start of the data is artificially supplied to the receiver.

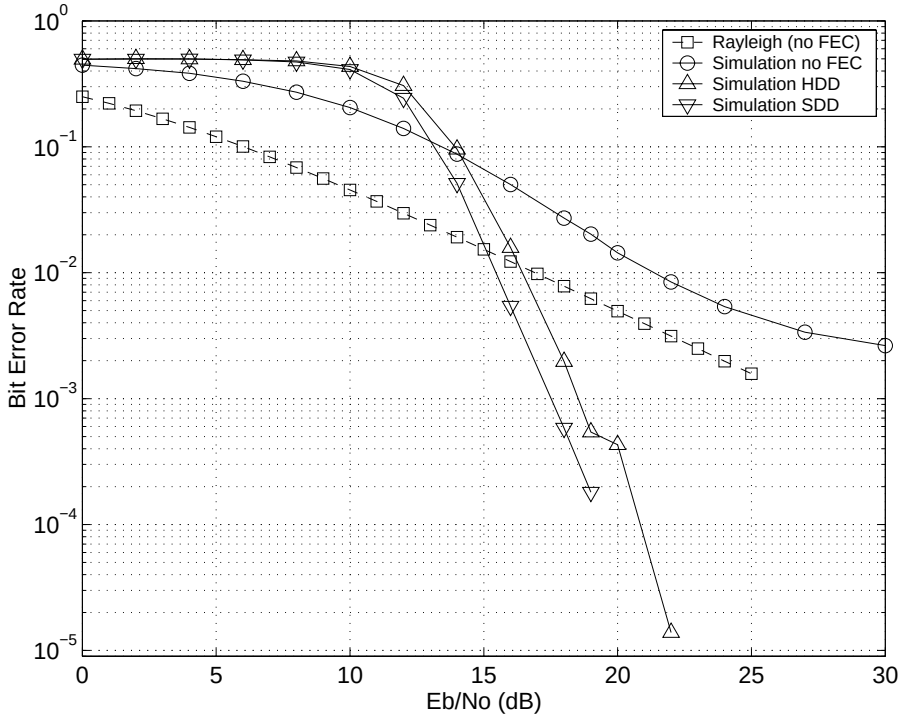

Figure 49. Bit error rate for the RAKE receiver using 30 taps with fixed spacing, in the modeled channel.

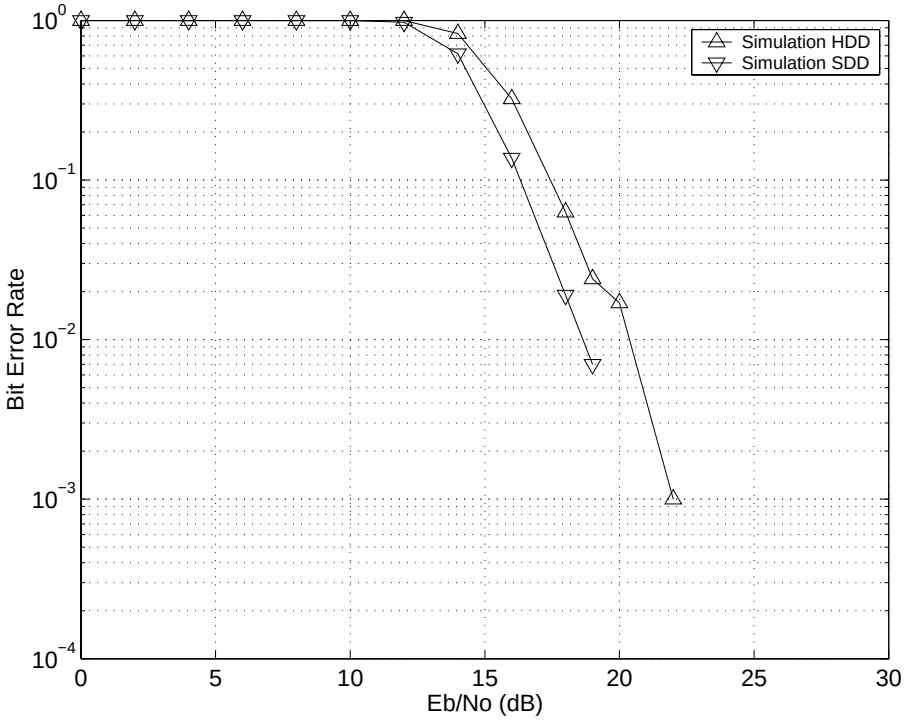

Figure 50. Packet error rate for the RAKE receiver using 30 taps with fixed spacing, in the modeled channel.

By fixing the tap spacing in the RAKE, performance is significantly worse than when the receiver uses no RAKE. In this configuration the RAKE cannot effectively detect and utilize the energy in the multipath arrivals. Therefore at low SNRs, the tap outputs are essentially noise. SDD still outperformed HDD and improved overall performance for $E_b / N_0 \geq 14$ dB.

3. Receiver Performance Using RAKE with 5 Adaptively Spaced Taps

Figure 51 and 52 show the performance of the experimental Seaweb modem receiver when 5 adaptively spaced taps are used in the RAKE and the start of the data is artificially supplied to the receiver. The tap locations are artificially supplied to the receiver and correspond to the first 5 peaks of the channel impulse response.

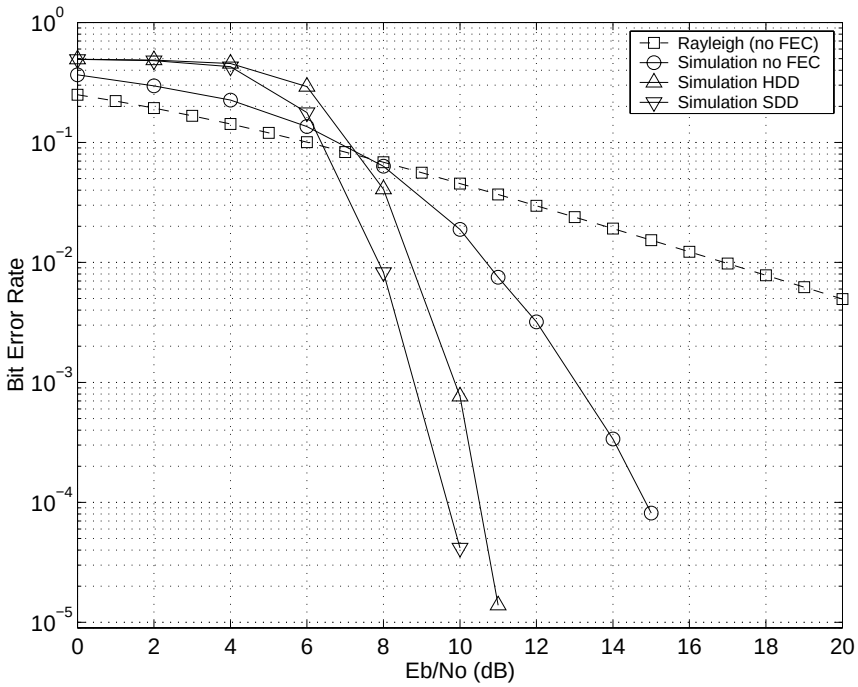

Figure 51. Bit error rate for the RAKE receiver using 5 adaptively spaced taps, in the modeled channel.

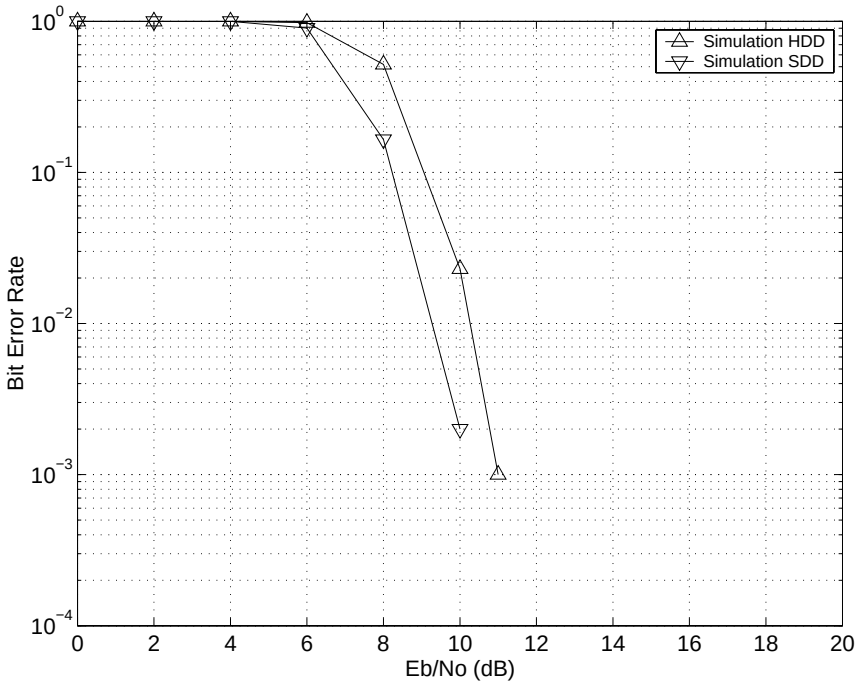

Figure 52. Packet error rate for the RAKE receiver using 5 adaptively spaced taps, in the modeled channel.

This implementation of the RAKE has the best performance of the three configurations. Likewise SDD was the optimal decoding method. Figure 53 and 54 illustrate the comparative results for each of the three receiver configurations using SDD We see that the improvement provided by using the 5 adaptive taps in the RAKE is at best 1 dB better than using no RAKE at all.

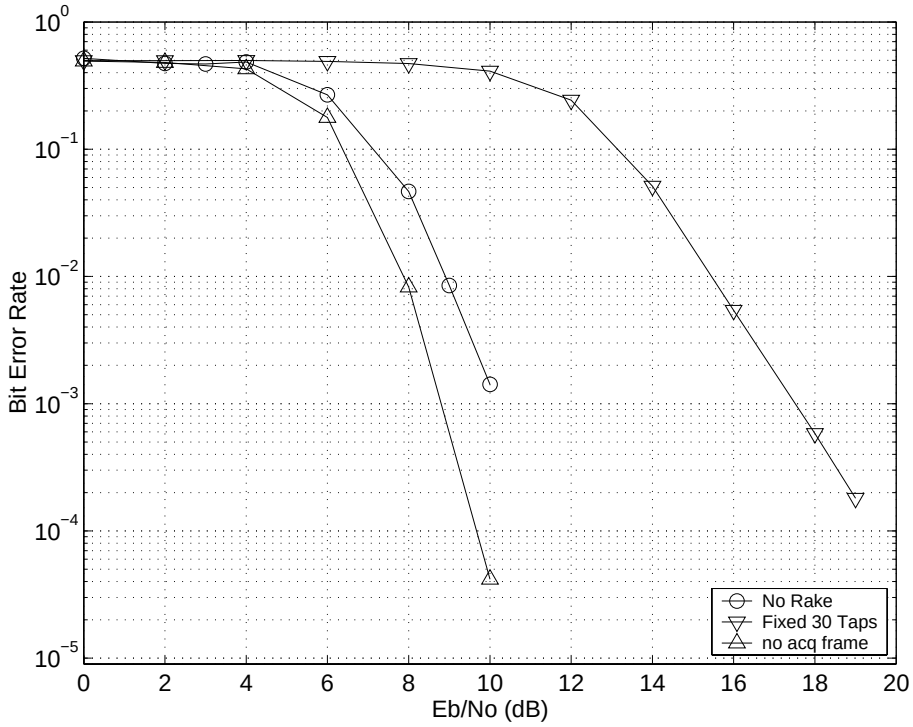

Figure 53. Comparison of bit error rate performance for the three receiver configurations using SDD, in the modeled channel.

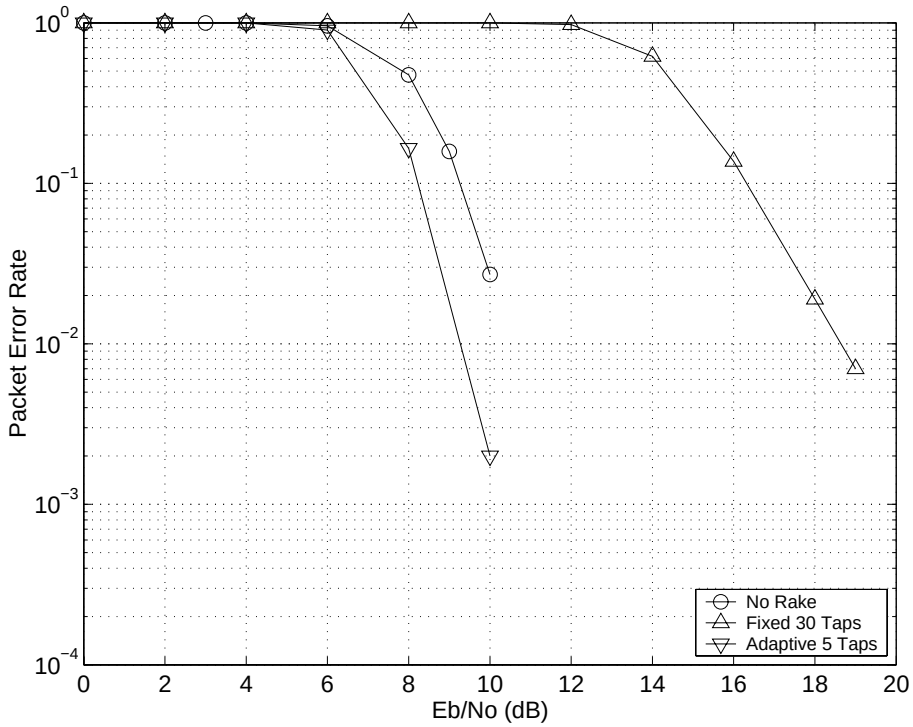

Figure 54. Comparison of packet error rate performance for the three receiver configurations using SDD, in the modeled channel.

4. Receiver Performance When Acquisition Frame and Data Frame Have Equal SNRs.

Figures 55 and 56 show the performance of the experimental Seaweb modem implementation when the acquisition frame and data frame SNR are equal. Only the results for SDD are presented, because it had the best performance.

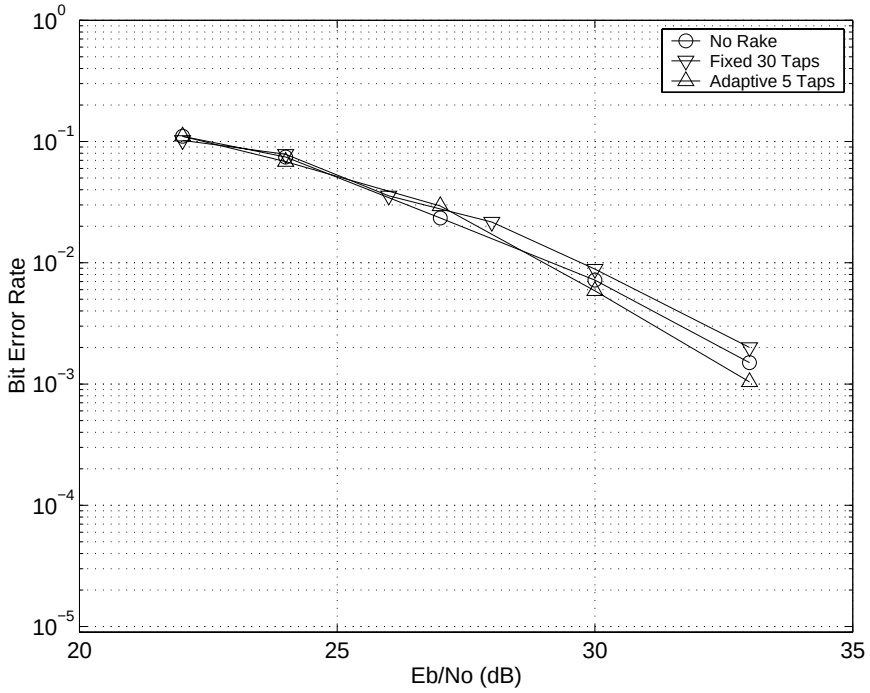

Figure 55. Bit error rate for the RAKE receiver using SDD and 5 adaptively spaced taps located at the dominant multipath arrivals, in the modeled channel.

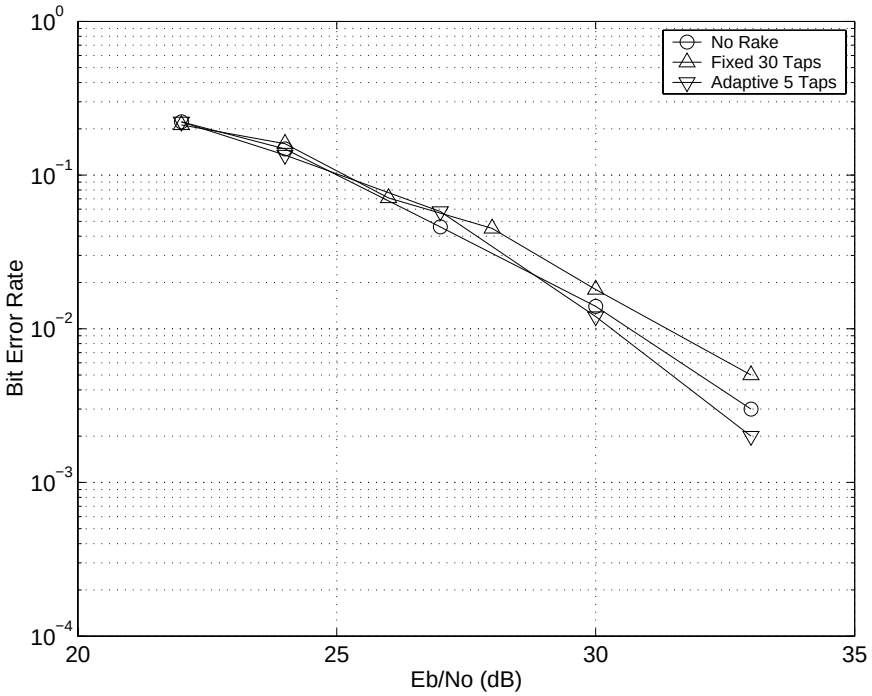

Figure 56. Packet error rate for the RAKE receiver using SDD and 5 adaptively spaced taps located at the dominant multipath arrivals, in the modeled channel.

As in the case of AWGN, performance is significantly reduced. The receiver is not able to acquire the signal consistently for values of $E_b / N_0 \leq 20$ dB. For $22 \leq E_b / N_0 \leq 30$ dB, all three configurations have equally poor performance. Above 30 dB the adaptive RAKE's performance is best, but only on the order of less than 2 dB. When $E_b / N_0 > 33$ dB, the receiver can effectively acquire the signal and because of the high SNR, the error rate immediately drops to zero.

5. Summary of Performance

In this chapter, we demonstrated that our receiver performed within theoretical expectations for the ideal AWGN channel. SDD was shown to outperform HDD, which outperformed the case when no forward error correction coding was used. In the modeled multipath channel when the starting location of the data frame is artificially provided, the receiver can achieve BERs below 10^{-4} for $E_b / N_0 < 22$ dB when SDD is used in any of the RAKE configurations, which is below the LPD threshold. When the SNRs of the two frames are kept equal, we only achieve $\text{BER} < 10^{-4}$ when $E_b / N_0 > 33$ dB. This is well above the LPD threshold.

In the next and final chapter, we summarize the important results and present areas for follow-on work.

VI. CONCLUSION

The goal of this thesis was to develop in software a DSSS modem suitable for underwater acoustic communications and to examine its performance through Monte Carlo simulation. The specific application of the modem is for utility packet transmission in Seaweb. The transmitter and receiver were implemented in software using MATLAB and the receiver was specifically designed to process the received transmission bit by bit. The channel was represented as an impulse response generated by a shallow-water acoustic propagation model. The modem was successfully implemented and our analysis of the modeled channel demonstrates that our design meets the utility packet requirements for Seaweb with some limitations.

A. FINDINGS

Our receiver was shown to perform according to theoretical expectations for an ideal additive white Gaussian channel and SDD was found to be the optimum decoding scheme. This gave us confidence that the results obtained for the modeled channel would be accurate.

In the modeled multipath channel, when the data frame starting location was artificially provided, the receiver could detect and demodulate signals for values of E_b / N_0, well below the 22 dB LPD threshold. Error-correction coding significantly improved performance with SDD providing the greatest gain. The RAKE receiver did improve performance when the tap locations were chosen to coincide with the occurrence of the multipath arrivals. But this improvement was small, less than 2 dB. This was much smaller than expected since diversity receptions should provide significantly greater gain. Using RAKE taps with a fixed spacing and in a sufficient number to cover the multipath arrivals proved to be the worst performing configuration.

The most significant limitation of the experimental modem was the acquisition algorithm. When its SNR of the acquisition frame was kept equal to the SNR of the data

frame, performance was very poor in all cases and values of $E_b / N_0 > 33$ dB were required to achieve low bit and packet error rates. At these values of E_b / N_0, the signal is easily audible at the receiver, violating the requirements for a low probability of detection signal. It must be stressed that with sufficient SNR the acquisition algorithm performed well and bit errors and packet error dropped to zero.

B. FOLLOW-ON WORK

There are four primary areas identified for follow-on work. First, improvements in the acquisition process should be pursued. Although the algorithm presented here worked well for signals with large SNRs, a better method of acquiring the signal must be found if the modem is to meet the LPD requirements.

Second, the detection algorithms should be improved. The RAKE receiver did not perform as well as hoped and much higher gains should theoretically be achievable by using diversity receptions. Closer analysis of the effects of incrementally increasing the number of RAKE taps, changing the tap threshold and performing a weighted combining of the tap outputs are just a few areas that could be examined. Also, alternate methods such as blind equalization may prove to be better at undoing the channel impulse response and being able to use the energy present in the multipath arrivals.

Third, the performance of the experimental Seaweb modem must be measured at sea. Although [11] and [16] have presented experimental results showing the general viability of DSSS, these experiments were performed at relatively high signal-to-noise ratios. There is little published work demonstrating performance at low SNRs (i.e., below 0 dB), which is the region of particular interest if DSSS is to meet the LPD requirements for Seaweb.

The fourth and final area for future work is in developing better channel models. The channels used in this thesis were relatively simple. The most detailed channel was generated using the ocean impulse response produced by the oceanographic acoustic propagation model called *Bellhop*. The output of *Bellhop* was further simplified by converting it to a set of delta functions. This model is not time invariant nor did it

introduce Doppler spreading, both of which are characteristics of a real ocean channel. Also, there is little published work comparing the predicted performance of broadband underwater acoustic communication signals using modeled channels with actual experimental results. This work is being pursued in part by [2], but without validated models that introduce time invariance and Doppler spreading, research is confined to expensive and lengthy experimental analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SOFTWARE USERS MANUAL

This Appendix acts as a software manual for the experimental Seaweb modem simulation. MATLAB 6.1 release 12 was used to implement all aspects of the transmitter, receiver and channel. A modular approach was taken when developing the software so that each significant process has its own function (i.e., MATLAB m-file). The focus in developing the MATLAB code was not on computational efficiency but instead on ease of understanding the implementation. It is assumed that the reader has a working knowledge of MATLAB.

A software flow diagram of the code is seen in Figures 58 and 59. Each of the blocks in the figures represents a separate MATLAB function written as an m-file. The overall controlling m-file is called *seaweb_modem_simulation.m*. Running this single MATLAB m-file will call all the other necessary functions and execute them.

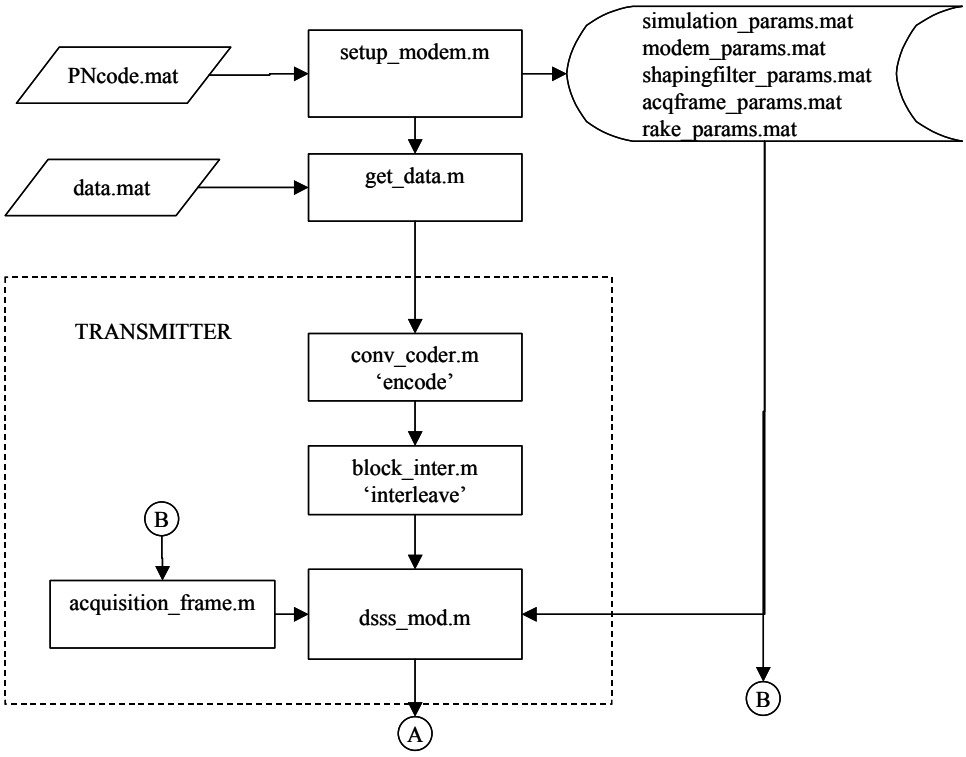

Figure 57. Software flow diagram of the transmitter portion.

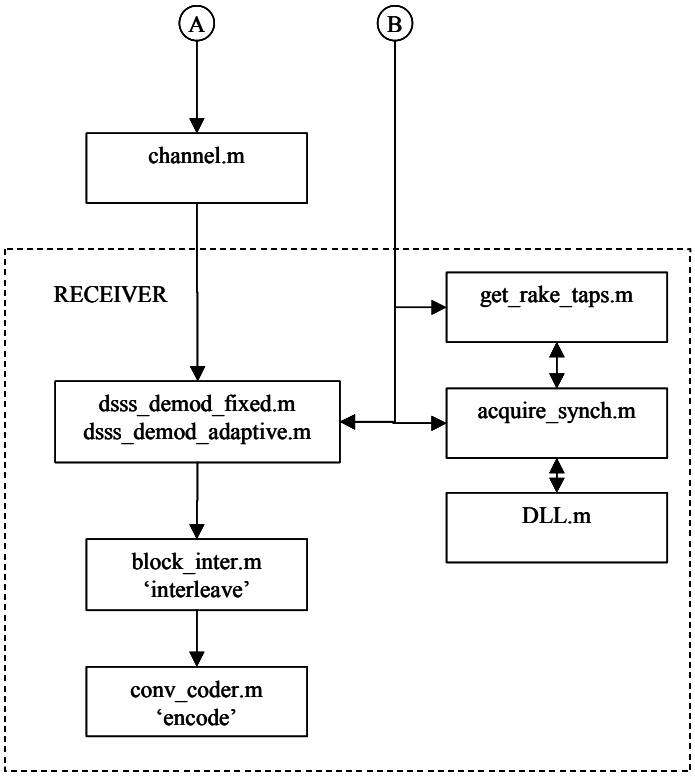

Figure 58. Software flow diagram for the channel and the receiver portions.

The key software blocks identified in the above figures are now discussed in detail. This is not a comprehensive listing of all the functions used in the simulation, since many built-in MATLAB functions are called. A complete listing of the Seaweb simulation source code is included in Appendix B.

seaweb_modem_simulation.m – This function is the overall controlling m-file. There is no input to this function and the output is a file called:

- *simulation_results.mat,* which contains a variable called *summary.* This variable summarizes the simulation errors and indicates the simulation parameters used, i.e. number of packets, SNR, E_b / N_0 and the channel.

setup_modem.m – This function defines all the design parameters. These include everything from the sampling frequency and packet length to the tap thresholds used in the RAKE receiver. The function has no inputs or outputs, instead the parameters are

saved as five individual mat files that are later loaded by other functions. These mat files are:

- *simulation_params.mat* – defines the parameters to be used in the overall simulation. These include the number of packets and the values of E_b / N_0 to be used.
- *modulation_params.mat* – defines the parameters needed for the transmitter and receiver including bit rate, chip rate, sampling rate, center frequency and packet length.
- *shapingfilter_params.mat* – defines the filter order and filter coefficients.
- *acqframe_params.mat* – defines the length and separation of the acquisition frame in chips.
- *rake_params.mat* – defines the rake settings including the tap thresholds and number of taps.

get_data.m – This function generates the information bit sequence. A case statement determines whether the data comes from a previously generated mat file (*'savedata'*) or is generated as a random bit sequence (*'random'*). The output of this function is a bit sequence.

convolutional_coder.m – This function codes or decodes the input bit sequence depending on the case statement indicated. The code rate and constraint length are defined within the function. To encode the data, the case statement is *'encode'*, to decode with HDD use *'decode_hard'* and to do SDD use *'decode_soft'*.

block_interleaver.m – This function interleaves or deinterleaves the bit sequence passed to it, depending on the case statement indicated. To interleave, the case statement is *'interleave'*. To deinterleave the case statement is *'deinterleave'*. The matrix size is defined within the function and must be manually changed by the user if packet sizes, other than the 72-bit Seaweb packets, are used. The input and output are both bit sequences.

dsss_mod.m – This function implements all the transmitter stages including chipping the data, modulating the data, appending the acquisition frame to the start of the

packet. The input is a bit sequence and the output is the transmitted signal. All the required parameters are loaded from mat files generated by *setup_modem.m*.

acquisition_frame.m – This is a sub-function, which generates the acquisition frame that is to be appended to the start of the data frame. It is called separately by *dsss_mod.m.* This is the frame that is later used to acquire the signal in the receiver.

shapingfilter.m – This is also a sub-function called by *dsss_mod.m, dsss_demod_fixed.m, dsss_demod_adaptive.m* and *acqistion_frame.m*. Its role is to filter the data passed to it using the pulse-shaping filter parameters defined in *setup_modem.m*. The output is the filtered data waveform.

channel.m – This function defines each of the channels used by the simulation. It accepts three inputs: the transmitted signal, the noise power to be added to the signal and a case statement indicating which channel to use. The function then convolves the input signal with the appropriate channel and adds AWGN to the result. The output is a single time series data set that becomes the input to the receiver.

dsss_demod_fixed.m / dsss_demod_adaptive.m – These functions perform all aspects of demodulating the received signal. One function or the other is used depending on whether fixed or adaptive tap spacing is desired. The required parameters are loaded from the *rake_params.mat* file generated by *setup_modem.m*. Two sub-functions, *acquire_synch.m* and *DLL.m*, are called to perform acquisition and tracking of the signal. The input is the received signal. The outputs are both hard decision bits and voltage levels to be used by the soft decision decoder.

acquire_synch.m – This function determines the start of the data sequence. The input is the received signal and the outputs are pointers indicating the location of the first data sample, an index of the correlation peaks and the output of the matched filter.

DLL.m – This function executes the delay lock loop. The function inputs are the received signal, the receiver generated replica and four pointers that indicate the start and end of the bit to be processed and the start and end of the receiver generated replica. The output of the DLL is the number of samples that the replica must be shifted in order to be aligned with the incoming signal.

get_rake_taps.m – This function uses the output of the *acquire_synch.m* block to estimate the channel impulse response. It then outputs the sample index of the multipath arrivals to the adaptive rake receiver. This function calls a p-file named *mmpeaks.p,* which was downloaded from the Mastering Matlab web site [22] and is used to find the peaks of the matched filter output in the acquisition block. These peaks represent an estimate of the multipath arrival times.

upsampler.m – This function, although not indicated in the software flow diagram, is used throughout the simulation. It lengthens the binary sequence passed to it by a user-defined amount. This serves the purpose of increasing the number of samples in the bit or chip. One use of this function is up-sampling the chipping sequence to the simulation sampling rate so that it can be modulated.

error_sum.m – This function generates the bit error rates and packet error rates for the simulation. The inputs are the decoded information bit sequence and the bit sequence originally generated. These are compared and error rates are generated.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB CODE

```
%****************************************
% SEAWEB MODEM SIMULATION
%
% This Program is the overall controlling
% software for the SeaWeb modem simulation
% all other functions are called by executing
% this m-file
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************
clear all; close all; clc;

tic;                    % starts clock to measure simulation run time
disp('started');

setup_modem;            % sets up all simulation settings
load('simulation_params')
load('modulation_params');

for m = 1:length(EbNo_dB);  %loop through each of the EbNo values

    for n = 1:N             %loop through each of the n  packets

        [m n]

        % generate the information data bit sequence
        d = get_data('random');

        % channel encode the data
        dc = convolutional_coder('encode',d);     %convolutional code the bits
        de = block_interleaver('interleave',dc);  %interleave the bits

         % modulate the data
        [tx, signal_power] = dsss_mod(de);

        % generate the noise power associated with EbNo
        EbNo(m) = 10.^(EbNo_dB(m)/10);
        Eb = signal_power.*Tb;
        No = Eb./EbNo(m);
        sigma_n = sqrt(No./(2*Ts));
        noise_power = sigma_n.^2;
        SNR_dB = 10.*log10(signal_power./noise_power);

        % pass the transmit signal through the channel
        if channel_type == 0; ch = channel('awgn',tx,noise_power);
        elseif channel_type ==1; ch = channel('multinoise1',tx,noise_power);
        elseif channel_type ==2; ch = channel('multinoise2',tx,noise_power);
        end
```

```
        % demodulate the received data in hard and soft decisions
        [hb, sb, shifts] = dsss_demod_adaptive(ch);
        % [hb, sb, shifts] = dsss_demod_fixed(ch);

        % channel decode the received hard bits
        h1 = block_interleaver('deinterleave',hb);    %de-interleave
        h2 = convolutional_coder('decode_hard',h1);   %HDD
        mh = h2;

        % channel decode the received soft bits
        s1 = block_interleaver('deinterleave',sb);    %de-interleave
        s2 = convolutional_coder('decode_soft',s1);   %SDD
        ms = s2;

         % determine the bit errors
        channel_errors = (hb - de)~=0;
        total_channel_errors(n,:) = sum(channel_errors);

         % determine the packet errors
        packet_errors_hdd(n,:) = sum((mh - d)~=0);
        packet_errors_sdd(n,:) = sum((ms - d)~=0);

    end

    % generate a summary of all the simulation results
    summary(:,m) = error_sum(channel_type, N, EbNo_dB(m), SNR_dB, ...
                                channel_errors, total_channel_errors, ...
                                packet_errors_hdd, packet_errors_sdd)

end

% save the simulation results
save simulation_results summary

disp('finished');
t = toc

% load handel;
% wavplay(y,8000);
```

```
function setup_modem

%****************************************
% SETS UP THE TX AND RX MODEMS
%
% This function is sets all modem parameters
% and saves them as separate *.mat files
% Five mat files are generated
%   1. simulation_params.mat
%   2. modulation_params.mat
%   3. acqframe_params.mat
%   4. shapingfilter_params.mat
%
% developed by Peter Duke September 2002
%****************************************

%--------------------------------------
% Set up Simulation Parameters
%disp('   setting up the modulation parameters');

N = 1;                %number of packets
EbNo_dB = 20;         %EbNo values to use
channel_type = 1;     %channel type 0 = AWGN

save simulation_params          N EbNo_dB channel_type;


%-----------------------------------------------------
% Set up the RAKE Receiver Parameters

Ntaps = 1;             % number of taps to use in the fixed RAKE
tap_threshold = 0.3;   % RAKE output threshold
ht_threshold = 0.05;   % threshold for determining the multipath peak
a = 0.9;               % weight factor for the DLL loop filter

   save rake_params          Ntaps tap_threshold ht_threshold a;


%---------------------------------------
% Set up Modulation Parameters

load gold2047_1;            % PN chipping sequence file.mat, variable name must be "code"

Rb = 40;              % bit rate
Rc = 2400;     % chip rate
Rs = 48000;           % sampling rate
fs = Rs;       % sampling frequency
Tb = 1/Rb;          % bit length
Tc = 1/Rc;  % chip length
Ts = 1/fs;     % sampling interval
fcarrier = 12000;         % carrier frequency

samples_per_bit  = Rs/Rb;     % samples per bit
samples_per_chip = Rs/Rc;   % samples per chip
chips_per_bit = Rc/Rb;          % chips per bit
```

```
% %------------------------
% Rc = 2560;      %chip rate
% Rs = 40960;        %sampling rate
% fs = Rs;        %sampling frequency
% Tc = 1/Rc;       %chip length
% fcarrier = 12000;                 %carrier frequency
% samples_per_bit  = 496*2; %samples per bit
% samples_per_chip = 16;          %samples per chip
% chips_per_bit = 31*2;                %chips per bit
% Rb = Rs/samples_per_bit; Tb = 1/Rb;
% %------------------------

packet_length = 160;              % number of bits transmitted per packet

% set modulation method: 1 = dbpsk, 2 = balanced qpsk (iq-dbpsk), 3 = dqpsk
modulation_method = 2;         % modulation method

% save settings to "modulation_params.mat" file
save modulation_params        Rb Rc Rs fs Tb Tc Ts fcarrier code ...
                              samples_per_bit samples_per_chip chips_per_bit...
                              packet_length ...
                              modulation_method;


%-------------------------------------------------------
% Set up Pulse Aquistion Frame Parameters

% set acquisition method:   1 = 3 PN sequences
%                           2 = none
acq_method = 1;

% Acquisition method 1 = 3 PN sequences
if acq_method == 1;
   len = 240;   % length in chips of each PN acquistion sequences
   sep = 200;   % separation in chips between acquistion sequences

% Acquisition method 2 = no aacquisition fram
% this is used to speed up simulations with AWGN only
elseif acq_method ==2;
   len = 0;
   sep = 0;
else
   error('ERROR not a valid acquistion method');
end

   save acqframe_params       Rb Rc Rs fs Tb Tc Ts fcarrier code ...
                              samples_per_bit samples_per_chip chips_per_bit ...
                              acq_method ...
                              len sep;
```

```
%-----------------------------------------------------
% Set up Pulse Shaping Filter Parameters

% set pulse shaping filter types: 1 = LPF filter, 2 = RRC
shapingfilter_type = 1;

% Shaping Filter type 1 = LPF to band limit to +/- 2500 Hz
if shapingfilter_type == 1;
        n = 64;                         % filter order
        fpass = 2500;               % passband; ought to be a bit bigger than needed
        fstop = 3000;               % stopband
        fmax = fs./2;               % maximum transmittable frequency at that rate
        w = [1 10];          % weighting factor
        f = [0 fpass/fmax fstop/fmax 1];      % frequency band edges
        A = [1 1 0 0];          % desired spectrum amplitues at band edges
        b = remez(n,f,A);
        a = 1;

        save shapingfilter_params      Rb Rc Rs fs Tb Tc Ts fcarrier code ...
                                       shapingfilter_type ...
                                       n b a;

else
    disp('ERROR not a valid Pulse Shaping Filter type');
end
```

```
function [varargout] = get_data(oper);

%****************************************
% This function either
%   1. generates random data, or
%   2. loads data from a specific data file
%****************************************

% disp('getting data');

switch oper

% generate random data
case 'random'
   d = randint(1,72); %generate the random data bit sequence

% load data from a previously saved data file
case 'savedata'

   load savedata
   d = savedata

end

varargout{1} = d;
```

```
function [varargout] = convolutional_coder(oper, varargin)

%****************************************
% CONVOLUTIONAL ENCODER/DECODER
%
% This funciton performs both
% Convolutional Coding and Decoding of the Data
% depending on the operator passed
% input - the information bit sequence
% output - the convolutionally encoded bit sequence
%
% developed by Peter Duke September 2002
%****************************************

%---------------------------------------
% use constraint length 9 coder/decoder
% generator polynomial is the same as for IS-95

K = 9; % contraint length
flushbits = ones(1, K-1);
trel = poly2trellis(K,[753 561]);

%---------------------------------------
switch oper

%---------------------------------------
case 'encode'

uncoded_data = varargin{1};
message = [uncoded_data flushbits];

coded_data = convenc(message,trel);
varargout = {coded_data};

%---------------------------------------
case 'decode_hard'

% Use hard decision decoding

coded_data = varargin{1};
tbl = 2;
decoded_data = vitdec(coded_data, trel, K-1, 'trunc', 'hard');

varargout = {decoded_data(1:end-K+1)};
```

```
%---------------------------------------
case 'decode_soft'

% Use 4-bit soft decision decoding
% To prepare for soft decision decoding,map to decision values

coded_data = varargin{1};
% [x,qcode] = quantiz(coded_data, [-1.65:2*1.65/14:1.65],[0:1:15]); % Values in qcode are between 0 and
2^3-1.
partition = [-pi/2 : (pi/16) : pi/2];
partition = partition(2:end-1);
codebook = [0:1:15];
[x,qcode] = quantiz(coded_data, partition, codebook); % Values in qcode are between 0 and 2^3-1.
decoded_data = vitdec(qcode, trel, K-1, 'cont', 'soft', 4);

varargout = {decoded_data(K:end)};

end
```

```
function [varargout] = block_interleaver(oper,varargin)

%****************************************
% BLOCK INTERLEAVER / DE-INTERLEAVER
%
% this function perfroms both block Interleaving
% and de-Interleaving depending on the operator passed
% input - the convolutionally encoded bit sequence
% output - the channel encoded bit sequence
%
% developed by Peter Duke September 2002
%****************************************

%-------------------------------------
% set number of rows and columns
rows = 16; cols = 10;

%-----------------------------------------
switch oper

%-----------------------------------------
case 'interleave'

uninterleaved_data = varargin{1};

block = reshape(uninterleaved_data,rows,cols);
interleaved_data = reshape(block',1,prod(size(block)));

varargout = {interleaved_data};

%-----------------------------------------
case 'deinterleave'

interleaved_data = varargin{1};

block = reshape(interleaved_data',cols,rows);
deinterleaved_data = reshape(block',1,prod(size(block)));

varargout = {deinterleaved_data};

end
```

```
function [varargout] = dsss_mod(varargin)

%*********************************************************
% MODULATOR
%
% Modulate the Data Using DSSS and
% Differential Binary phase-shift keying with  Quadrature Spreading (DS-IQ-DBPSK)
%
% developed by Peter Duke September 2002
% last modified 2/9
%*********************************************************

%****************************************
% Initialize the Modem

% Read in the data
data = varargin{1};

% Setup modem using settings in modem_params.mat
% and check that it is set up correctly

load ('modulation_params');
config_error_check('modulator',data);

%****************************************
% DS-IQ-DBPSK Modulator

%----------------------------------------------
% generate the differential bit sequence

bi = [NaN data];  % data bits
dbi(1) = 1;           % differential data bits
for i = 2:length(bi)
   dbi(i) = not(xor(dbi(i-1), bi(i)));
end

%----------------------------------------------
% convert differential data from 0/1 to -1/+1

nrz_dbi = (dbi*2)-1;
number_of_dbits = length(nrz_dbi);

%----------------------------------------------
% up sample the data to the chip rate

data_sequence = upsampler(nrz_dbi, chips_per_bit);

%----------------------------------------------
% determine repetitions/portion of orignal gold code required
% to chip entire output

total_chips_needed = number_of_dbits.*chips_per_bit;
number_of_repetitions = ceil(total_chips_needed./length(code));
longcode = repmat(code, 1, number_of_repetitions);
```

```
%---------------------------------------------
% chip the data

% generate chip sequence, chip all data with one sequence
chip_sequence_ip = longcode(1, 1:(chips_per_bit.*number_of_dbits));
chip_sequence_qp = longcode(2, 1:(chips_per_bit.*number_of_dbits));

% chip the data
chipped_data_sequence_ip = chip_sequence_ip .* data_sequence;
chipped_data_sequence_qp = chip_sequence_qp .* data_sequence;

%---------------------------------------------
% Up sample Chipped Data Sequence to fs and  shape the chip sequence using the pulse shaping filter

sampled_chips_ip = upsampler(chipped_data_sequence_ip, samples_per_chip);
sampled_chips_qp = upsampler(chipped_data_sequence_qp, samples_per_chip);

shaped_data_ip = shapingfilter(sampled_chips_ip);
shaped_data_qp = shapingfilter(sampled_chips_qp);
shaped_data = [shaped_data_ip; shaped_data_qp];

%---------------------------------------------
% modulate with both and I and Q carrier

t = [0:1/fs:(length(shaped_data)/fs)-1/fs];
carrier_ip = (1./sqrt(2)).*cos(2.*pi.*fcarrier.*t);
carrier_qp = (-1./sqrt(2)).*sin(2.*pi.*fcarrier.*t);

if modulation_method==1;
   modulated_data = shaped_data(1,:).*carrier_ip ;
elseif modulation_method==2
   modulated_data = shaped_data(1,:).*carrier_ip + shaped_data(2,:).*carrier_qp;
end

%---------------------------------------------
% add the acquistion frame

load acqframe_params;

if acq_method == 1
   acq_frame = acquisition_frame;
elseif acq_method ==2;
   acq_frame = [];        % used only to speed up the AWGN simulations
end

tx_signal = [acq_frame modulated_data];

tx_signal_power = (sum(modulated_data.^2))./length(modulated_data);

%*****************************************
% Generate Modem Output

varargout{1} = tx_signal;
varargout{2} = tx_signal_power;
```

```
function [varargout] = acquisition_frame

%****************************************
% ACQUISITION FRAME GENERATOR
%
% This function gemerates the acqisition frame
% which is to be appended to the date frame
% in the transmiltter
% input - none
% output - acquisiton frame
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************

load ('acqframe_params');

% geenerate the acquistion frame bit sequence
c = code(:,1:len);
z = zeros(2,sep);
chip_sequence = [ z c z c z c z ];

% upsample to the sampling frequency
sampled_chips_ip = upsampler(chip_sequence(1,:), samples_per_chip);
sampled_chips_qp = upsampler(chip_sequence(2,:), samples_per_chip);

% pass the sequence through the pulse shaping filter
shaped_chips_ip = shapingfilter(sampled_chips_ip);
shaped_chips_qp = shapingfilter(sampled_chips_qp);

% modulate the sequence
t = 0: 1/fs : length(shaped_chips_ip)/fs - 1/fs;
carrier_ip = (1./sqrt(2)).*cos(2.*pi.*fcarrier.*t);
carrier_qp = (-1./sqrt(2)).*sin(2.*pi.*fcarrier.*t);

% add the two channels to generate the acquistion frame
acq_frame = shaped_chips_ip.*carrier_ip + shaped_chips_qp.*carrier_qp;

% save an index to the first data sample for later comparison
first_data_sample = length(acq_frame) + 1;
save first_data_sample first_data_sample;

% generate the output
varargout{1} = acq_frame;
```

```
function [varargout] = shapingfilter(varargin);

%****************************************
% SHAPING FILTER
%
% This function filters the sampled data sequence
% passed to it using the pulse shaping filter
% parameters generated in the function setup_mode.m
% it is used to pulse shape the PN chipping waveform
% in both the transmitter and receiver
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************

%----------------------------------------
% read in the data to be filtered and
% the filter parameters
% filter parameters are defined from setup_modem.m
% where:    n = filter order
%                   b,a are the filter coefficients
%          alpha = rolloff factor for the Root Rasied Cosine

data_in = varargin{1};
load ('shapingfilter_params')

%----------------------------------------
% Use LPF filter - type 1
if shapingfilter_type ==1;

        % add trailing zeros to get all data through filter
        data_in = [data_in zeros(1,n+1)];

        % filter the input signal
        data_out = filter(b, a, data_in);


%----------------------------------------
% Use RRC filter - type 2
elseif shapingfilter_type ==2;

        data_out = rcosflt(data_in, 1, samples_per_chip, 'sqrt/fs', rolloff_factor)';

end

%----------------------------------------
% gernerate output

varargout{1} = data_out;
```

```
function [varargout] = channel(oper,varargin);

%*********************************************************
% CHANNEL SIMULATOR
%
% This funciton loads the channel ht
% convolves it with the transmitted signal
% and adds additive white gaussian noise
% input - transmitted signal and channel to be used
% output - input to the receiver
%
% developed by Peter Duke September 2002
% last modified 2/9
%*********************************************************

%-------------------------------------------------
% define and load variables

tx = varargin{1};
load modulation_params;

switch oper

%-------------------------------------------------
% add AWGN only

case 'awgn'

        tx = [tx zeros(1,fs*0.2)];  % add 0.2sec of silence to the end
        noisepower = varargin{2};
        noisestd = sqrt(noisepower);
        noise = randn(1,length(tx)).*noisestd;
        np = var(noise);
        out = tx + noise;

%-------------------------------------------------
% load the channel ht
% convolve it with the transmitted signal and add AWGN

case 'multinoise1'

        tx = [tx zeros(1,fs*0.2)];  % add 0.2sec of silence to back
        noisepower = varargin{2};
        noisestd = sqrt(noisepower);

        Tc = 20;                                % chip spacing
        ht = [1.0 zeros(1,1.5*Tc-1) 0.6 zeros(1,2*Tc-1) 0.4]; % generate ht
        ht = ht./sqrt(sum(ht.^2));                      % normalize ht
        taplocations = find(ht~=0)
        save taplocations taplocations;

        out = conv(tx,ht);
        out = out + randn(1,length(out))*noisestd;
```

```
%-------------------------------------------------
% load the channel ht
% convolve it with the transmitted signal and add AWGN

case 'multinoise2'

        noisepower = varargin{2};
        noisestd = sqrt(noisepower);

        load sigex_ht;
        ht = ht3;
        taplocations = find(ht~=0);
        save taplocations taplocations;

        out = conv(tx,ht);
        out = out + randn(1,length(out))*noisestd;

end

%-------------------------------------------------
% generate output

varargout{1} = out;
```

```
function [varargout] = dsss_demod_fixed(varargin)

%*********************************************************
% DEMODULATOR - USING A FIXED-TAP RAKE RECEIVER
%
% This functions Demodulates the DS-IQ-DBPSK signal
% using a RAKE receiver
% input - output of the channel simulator
% output - hard and soft bit sequences
%
% developed by Peter Duke September 2002
% last modified 2/9
%*********************************************************

%****************************************
% Initialize the Demodulaoter

%----------------------------------------
% Read in the data  to be processed by the RAKE receiver
data = varargin{1};

%----------------------------------------------
% Setup modem using settings in modem_params.mat
% and check that it is set up correctly

load ('modulation_params');
load ('rake_params');          %loads the number of taps and threshold

% check that tx and rx are conFigured the same
config_error_check('demod')

% initialize variables
shift = 0;
shifts = 0;
allx = [ ];
ally = [ ];

%****************************************
% Acquire the Packet Data

load ('acqframe_params');

if acq_method == 1;

   [first_data_sample, multipath, window, index_corr_peak] = acquire_sync(data);
   %   load first_data_sample
   data = data(first_data_sample:end);

elseif acq_method ==2;

   data = data;

end
```

```
%*****************************************
% Demodulate the Data using a non-coherent RAKE receiver

%-----------------------------------------------
% determine repetitions/portion of orignal gold code required
% to chip entire output

number_of_dbits = packet_length + 1;
total_chips_needed = number_of_dbits.*chips_per_bit;
number_of_repetitions = ceil(total_chips_needed./length(code));
longcode = repmat(code, 1, number_of_repetitions);

%-----------------------------------------------
% Generate replica of entire modulated and shapped chipping sequence

% generate chip sequence for all data
chip_sequence_ip = longcode(1,1:(chips_per_bit.*number_of_dbits));
chip_sequence_qp = longcode(2,1:(chips_per_bit.*number_of_dbits));

% Upsample chip sequence to fs
sampled_chips_ip = upsampler(chip_sequence_ip, samples_per_chip);
sampled_chips_qp = upsampler(chip_sequence_qp, samples_per_chip);

% pulse shape the upsampled sequence using shapingfilter.m
shaped_chips_ip = shapingfilter(sampled_chips_ip);
shaped_chips_qp = shapingfilter(sampled_chips_qp);

% modulate with carrier
t = [0:1/fs:(length(shaped_chips_ip)/fs)-1/fs];
carrier_ip = 2.*cos(2.*pi.*fcarrier.*t);
carrier_qp = -2.*sin(2.*pi.*fcarrier.*t);

replica_ip = shaped_chips_ip.*carrier_ip;
replica_qp = shaped_chips_qp.*carrier_qp;
replica = [replica_ip; replica_qp];

%-----------------------------------------------
% cycle through data bit by bit

clear phi dphi;

for bn = 1:number_of_dbits;

   %-----------------------------------------------
   % generate pointers to the samples in the chip sequence
   % that indicate the start and end of the bit
   % d1, d2 = pointers to first and last samples of data bit
   % r1, r2 = pointers to first and last samples in replica bit
```

```
if bn==1;
   % need to account for filter transient in first bit
   load ('shapingfilter_params');
   filterlag = n/2;
   d1 = filterlag + 1;
        d2 = d1 + samples_per_bit - 1;
   r1 = d1;
   r2 = d2;
else
   d1 = d1 + samples_per_bit;
   d2 = d1 + samples_per_bit - 1;
   r1 = r1 + samples_per_bit;
   r2 = r1 + samples_per_bit - 1;
end

 %---------------------------------------
 % run the Delay Lock Loop
 [shift] = DLL(data,replica,d1,d2,r1,r2);
  % shift = 0;

 % LPF the shift to remove noise jitter
 shift_lpf = fix(shifts(end)*a + shift*(1-a));   %filtered shift

 shifts = [shifts shift_lpf];

 r1 = r1 + shift_lpf;
 r2 = r2 + shift_lpf;

    %-----------------------------------------------
    %cycle through each of the fixed rake taps

    for tn = 1:Ntaps;   % cycle through each tap

       % generate tap delay
       td = (tn-1).*samples_per_chip;

       % correlate data to replica for each tap delay
       % integrate and dump
       xv = data(d1+td:d2+td).*replica_ip(r1:r2);
        x(tn) = sum(xv)./length(xv);
        yv = data(d1+td:d2+td).*replica_qp(r1:r2);
        y(tn) = sum(yv)./length(yv);


       allx = [allx x(1)];
       ally = [ally y(1)];

    end                  % get next tap

% determine the mag and phase at the output of each tap
j = sqrt(-1);
c = x + j*y;
cmag = abs(c);
cphase = angle(c);
```

```
   %--------------------------------------------
   % Determine the voltage level at each tap output for each bit
   % by Differentially demodulating each tap before combining

   phi(bn,:) = cphase;
   mag(bn,:) = cmag;

   % for the first channel bit initialize the variable vectors
   if bn==1
      dphi(bn,1:length(c)) = NaN;
      dx(bn,1:length(c)) = NaN;
      dy(bn,1:length(c)) = NaN;
      tapvoltage(bn,1:length(c)) = NaN;

   % for the remaining bits determine the voltage at each tap output
   else
      dphi(bn,:) = phi(bn,:) - phi((bn-1),:);

      dx(bn,:) = cos(dphi(bn,:));
      dy(bn,:) = sin(dphi(bn,:));
      dphi_new(bn,:) = atan2(dy(bn,:),dx(bn,:));

      tapvoltage(bn,:) = (pi/2) - abs(dphi_new(bn,:));

      % zero the tap voltages that are below the noise threshold
      % that is they have a mag above a noise threshold
      below_thresh = (mag(bn,:)<(tap_threshold.*(mag(bn,1))));
      p = find(below_thresh==1);
      tapvoltage(bn,p) = 0;
   end

   %--------------------------------------------
   % Sum the outputs of all taps

   op_volts(bn) = sum(tapvoltage(bn,:));

end   % get the next bit

%-----------------------------------------------
% Make hard and soft decision

hardbits = (sign(op_volts) + 1)./2;
softbits = op_volts;

%*****************************************
% generate outputs

varargout{1} = hardbits(2:end);
varargout{2} = softbits(2:end);
varargout{3} = shifts;
```

```
function [varargout] = dsss_demod_adaptive(varargin)

%*********************************************************
% DEMODULATOR - USING A ADAPTIVE TAP RAKE RECEIVER
%
% This functions Demodulates the DS-IQ-DBPSK signal
% using a RAKE receiver
% input - output of the channel simulator
% output - hard and soft bit sequences
%
% developed by Peter Duke September 2002
% last modified 2/9
%*********************************************************

%****************************************
% Initialize the Demodulator

%----------------------------------------
% Read in the data  to be processed by the RAKE receiver
data = varargin{1};

%----------------------------------------------
% Setup modem using settings in modem_params.mat
% and check that it is set up correctly

load ('modulation_params');
load ('rake_params');          %loads the number of taps and threshold

% check that tx and rx are conFigured the same
config_error_check('demod')

% initialize variables
shift = 0;
shifts = 0;
allx = [ ];
ally = [ ];

%****************************************
% Acquire the Packet Data

load ('acqframe_params');

if acq_method == 1;

   [first_data_sample, multipath, window, index_corr_peak] = acquire_sync(data);
   % load first_data_sample

   %----------------------------------------------
   % determine where the adaptive taps should be placed

      [taplocations] = get_rake_taps(multipath, window, index_corr_peak, ht_threshold);
      %  load taplocations;
      %  taplocations
   %----------------------------------------------
```

```
   data = data(first_data_sample:end);

elseif acq_method ==2;

   data = data;

end

%*****************************************
% Demodulate the Data using DBPSK or I-Q DBPSK

%-----------------------------------------------
% determine repetitions/portion of orignal gold code required
% to chip entire output

number_of_dbits = packet_length + 1;
total_chips_needed = number_of_dbits.*chips_per_bit;
number_of_repetitions = ceil(total_chips_needed./length(code));
longcode = repmat(code, 1, number_of_repetitions);

%-----------------------------------------------
% Generate replica of entire modulated and shapped chipping sequence

% generate chip sequence for all data
chip_sequence_ip = longcode(1,1:(chips_per_bit.*number_of_dbits));
chip_sequence_qp = longcode(2,1:(chips_per_bit.*number_of_dbits));

% Upsample chip sequence to fs
sampled_chips_ip = upsampler(chip_sequence_ip, samples_per_chip);
sampled_chips_qp = upsampler(chip_sequence_qp, samples_per_chip);

% pulse shape the upsampled sequence using shapingfilter.m
shaped_chips_ip = shapingfilter(sampled_chips_ip);
shaped_chips_qp = shapingfilter(sampled_chips_qp);

% modulate with carrier
t = [0:1/fs:(length(shaped_chips_ip)/fs)-1/fs];
carrier_ip = 2.*cos(2.*pi.*fcarrier.*t);
carrier_qp = -2.*sin(2.*pi.*fcarrier.*t);

replica_ip = shaped_chips_ip.*carrier_ip;
replica_qp = shaped_chips_qp.*carrier_qp;
replica = [replica_ip; replica_qp];

%-----------------------------------------------
% cycle through data bit by bit

clear phi dphi;

for bn = 1:number_of_dbits;
```

```
%---------------------------------------------
% generate pointers to the samples in the chip sequence
% that indicate the start and end of the bit
% d1, d2 = pointers to first and last samples of data bit
% r1, r2 = pointers to first and last samples in replica bit

if bn==1;
   % need to account for filter transient in first bit
   load ('shapingfilter_params');
   filterlag = n/2;
   d1 = filterlag + 1;
   d2 = d1 + samples_per_bit - 1;
   r1 = d1;
   r2 = d2;
else
   d1 = d1 + samples_per_bit;
   d2 = d1 + samples_per_bit - 1;
   r1 = r1 + samples_per_bit;
   r2 = r1 + samples_per_bit - 1;
end

 %--------------------------------------
 % run the Delay Lock Loop
 [shift] = DLL(data,replica,d1,d2,r1,r2);
 % shift = 0;

 % LPF the shift to remove noise jitter
 shift_lpf = fix(shifts(end)*a + shift*(1-a));    %filtered shift

 shifts = [shifts shift_lpf];

 r1 = r1 + shift_lpf;
 r2 = r2 + shift_lpf;

    %---------------------------------------------
    %cycle through each of the adaptive rake taps

    for tn = 1:length(taplocations)  %cycle through each tapfor adaptive taps

      % generate tap delay
      td = taplocations(tn)-1;

      % correlate data to replica for each tap delay
      % integrate and dump
      xv = data(d1+td:d2+td).*replica_ip(r1:r2);
       x(tn) = sum(xv)./length(xv);
       yv = data(d1+td:d2+td).*replica_qp(r1:r2);
       y(tn) = sum(yv)./length(yv);


      allx = [allx x(1)];
      ally = [ally y(1)];

    end                        %get next tap
```

```
   % determine the mag and phase at the output of each tap
   j = sqrt(-1);
   c = x + j*y;
   cmag = abs(c);
   cphase = angle(c);

   %--------------------------------------------
   % Determine the voltage level at each tap output for each bit
   % by Differentially demodulating each tap before combining

   phi(bn,:) = cphase;
   mag(bn,:) = cmag;

   % for the first channel bit initialize the variable vectors
   if bn==1
      dphi(bn,1:length(c)) = NaN;
      dx(bn,1:length(c)) = NaN;
      dy(bn,1:length(c)) = NaN;
      tapvoltage(bn,1:length(c)) = NaN;

   % for the remaining bits determine the voltage at each tap output
   else
      dphi(bn,:) = phi(bn,:) - phi((bn-1),:);

      dx(bn,:) = cos(dphi(bn,:));
      dy(bn,:) = sin(dphi(bn,:));
      dphi_new(bn,:) = atan2(dy(bn,:),dx(bn,:));

      tapvoltage(bn,:) = (pi/2) - abs(dphi_new(bn,:));

      % zero the tap voltages that are below the noise threshold
      % that is they have a mag above a noise threshold
      below_thresh = (mag(bn,:)<(tap_threshold.*(mag(bn,1))));
      p = find(below_thresh==1);
      tapvoltage(bn,p) = 0;
   end

   %--------------------------------------------
   % Sum the outputs of all taps

   op_volts(bn) = sum(tapvoltage(bn,:));

end % get the next bit

%--------------------------------------------
% Make hard and soft decision
hardbits = (sign(op_volts) + 1)./2;
softbits = op_volts;

%*******************************************
% generate outputs

varargout{1} = hardbits(2:end);
varargout{2} = softbits(2:end);
varargout{3} = shifts;
```

```
function [varargout] = acquire_sync(varargin);

%****************************************
% ACQUIRE SYNCHRONIZATION
% This function acquires the 3 PN sequences
% that were appended to the start of the data frame
% and determines the start of the first data sample
%
% based on code developed by Etham Sozer
% modifed by Peter Duke September 2002
% last modified 2/9
%****************************************

%-----------------------------------------------
% Read in the data

data = varargin{1};

%-----------------------------------------------
% Get acqustion frame settings from acqframe_params.mat

load ('acqframe_params');
load ('shapingfilter_params');

%-----------------------------------------------
% define and initialize additional variables, flags and counters

window = (len+sep).*samples_per_chip;  % window length in samples
filter_delay = n + 1;

found = 0;                    % flag set to 1 when 3 PN sequences found
acq_sample = 0;   % accumulate filtered data in y_acc vector

%-----------------------------------------------
% generate the Matched Filter coefficients

c = code(:,1:len);

sampled_chips_ip = upsampler(c(1,:),samples_per_chip);
sampled_chips_qp = upsampler(c(2,:),samples_per_chip);

MF_ip = fliplr(shapingfilter(sampled_chips_ip));
MF_qp = fliplr(shapingfilter(sampled_chips_qp));

% initialize the initial conditions of the filter
Zic_ip = zeros(1,(length(MF_ip)-1));
Zic_qp = zeros(1,(length(MF_qp)-1));


%-----------------------------------------------
% Slide windows through the incoming data
% the second window offset by half the window length
% looking for the 3 generated the gold code sequences
% stops when the 3 correlation peaks have been found
% and results in "found" variable flag = 1
```

```
for  split_window = 0:1

   first = 1;                    % pointer to first sample in the window
   if   split_window == 1;       % offset window
      first = first + window/2;
   end

   Npeak = 1;                    % number of peaks found
   y_acc    = [];           % vector for accumulating envelope

   %-----------------------------------
   while (~found)

      last = first + window - 1;% pointer to last sample in the window
      if last > length(data)
         error('CANNOT ACQUIRE THE SYNCHRONIZATION FRAME');
         return;                              % jump out of loop when end of the data reached
      end;

      %-----------------------------------
      % demodulate the signal in the window
            t = first:last;
            carrier = exp(-j.*2.*pi.*fcarrier.*t./fs );

            signal_demod = data(first:last) .* carrier;
            [y1, Zic_ip] = filter( MF_ip, 1,    signal_demod, Zic_ip);
            [y2, Zic_qp] = filter( MF_qp, 1, j.*signal_demod, Zic_qp);

            y = abs( y1 - y2 ).^2 ./ length(signal_demod).^2; % he had length(signal_demod)^2
            y_acc = [ y_acc y ];

      mfoutput = y_acc;

      %-----------------------------------
      % find index and magnitude of correlation peak in the window

            peak_subindex = find( y == max(y) );                        % find indices of peaks
            index_corr_peak(Npeak) = peak_subindex(1) + first - 1;      % store the index of the first peak
            mag_corr_peak(Npeak) = max(y);

      %-----------------------------------
      % checking distance between 1st/2nd peaks and 2nd/3rd peaks
      % are within 2% of the window lenth

      if  (Npeak >= 2);
         if ( abs(index_corr_peak(Npeak) - index_corr_peak(Npeak-1) - window ) > 0.02.*window )
            index_corr_peak(1) = index_corr_peak(Npeak);        % put the new peak in the first position
            Npeak = 1;     % if not within tolerance reset the peak counter
         end;
      end;

      %-----------------------------------
      % once all 3 peaks are found
      % determine the index of the start of the data packet = mean of the 3 peak indices
```

```
      if (Npeak == 3)     % ie if we have found all  peaks
         first_data_sample = round( mean(index_corr_peak(1:3)) ) + ((2.*sep) + len).*samples_per_chip+ 1;
         mean_mag_corr_peak = mean(mag_corr_peak(1:3));
         found = 1;
         multipath = y_acc;
      end;

      % advance the peak counter if we have a nonzero peak
      if mag_corr_peak(Npeak) ~= 0
         Npeak = Npeak + 1;
      end

      first = first + window;     % advance the sample counter to the next window

   end;                % next window


end;   % next offset window

%-------------------------------------------------
% generate the output

varargout{1} =  first_data_sample;
varargout{2} = multipath;
varargout{3} = window;
varargout{4} = index_corr_peak;
```

```
function [varargout] = get_rake_taps(varargin);

%*********************************************************
% DETERMINE THE  LOCATION FOR THE ADAPTIVE RAKE TAPS
%
% This determines where the adaptive RAKE taps
% should be located
% it uses the estimate of the channel impulse reponse
% found from the acquisition functions
% matched filter output
%
% developed by Peter Duke September 2002
% last modified 2/9
%*********************************************************

%-------------------------------------------------
% define and load variables
multipath = varargin{1};
window = varargin{2};
index_corr_peak = varargin{3};
ht_threshold = varargin{4};

%-------------------------------------------------
% average the three matched filter outputs passed from acquire_sync
s1 = multipath(index_corr_peak(1):index_corr_peak(1)+window/2-1);
s2 = multipath(index_corr_peak(2):index_corr_peak(2)+window/2-1);
s3 = multipath(index_corr_peak(3):index_corr_peak(3)+window/2-1);
stotal = (s1+s2+s3)./3;
x1 = 1:(window/2);

% find the 5 or fewer largest multipath components
% that are within 5% of the first arrival
i1 = mmpeaks(stotal);
m1 = stotal(i1);
m2 = fliplr(sort(m1));
m3 = m2(1:5);
i2 = find(m3>ht_threshold*m3(1));

% loop through to find the index of these largest components
% the is required because there may be less than 5
n=1;
i3=1;
while n<length(i2);
   n=n+1;
   i3 = [i3 find(stotal == m3(n))];
end

taplocations = i3;

%-------------------------------------------------
% generate output

varargout{1} = taplocations
```

```
function [varargout] = DLL(varargin);

%****************************************
% DELAY LOCKED LOOP
% This function keeps the receiver generate PN replica aligned
% with the received signals PN sequence
% input - receiver generated relplica and received data
% output - error signal indicating number of samples
%          reciever replica need to be shifted
%
% modifed by Peter Duke September 2002
% last modified 2/9
%****************************************


%----------------------------------------
% define input variables

data = varargin{1};
replica = varargin{2};
d1 = varargin{3};
d2 = varargin{4};
r1 = varargin{5};
r2 = varargin{6};

load modulation_params;
load shapingfilter_params;
Tc = samples_per_chip;

%----------------------------------------
% get first bit of data

signal = data(d1:d2);

%----------------------------------------
% generate the left and right shifted replicas

replica_ip = replica(1,:);
replica_qp = replica(2,:);

%generate replica
delay = 0; %delay in samples;
g1 = replica_ip(r1+delay:r2+delay);
g2 = replica_qp(r1+delay:r2+delay);

%generate a left shifted replica
delay = -Tc/2; %delay in samples;
dm1 = replica_ip(r1+delay : r2+delay);
dm2 = replica_qp(r1+delay : r2+delay);

%generate a right shifted replica
delay = Tc/2; %delay in samples;
dp1 = replica_ip(r1+delay:r2+delay);
dp2 = replica_qp(r1+delay:r2+delay);
```

```
gc = g1 + j*g2;
dp = dp1 + j*dp2;
dm = dm1 + j*dm2;

%----------------------------------------
% determine the required shift

% generate cross-correlations
corr_left  = abs(xcorr(signal,dm));
corr_right = abs(xcorr(signal,dp));

% generate the S-curve
diff = corr_left.^2 - corr_right.^2;
vv = diff/max(diff);
save plot7 vv;

% determine the zero crossing and translate that
% to the number of samples to shift
lag0 = (length(vv)+1)./2;
[vp,pp] = max(vv(lag0-Tc:lag0));
pp;
pp = pp+(lag0-Tc)-1;
[vm,pm] = min(vv(lag0:lag0+Tc));
pm;
pm = pm+lag0-1;
xcross = pp + ((pm-pp)/2);
shift = round(lag0-xcross);

%----------------------------------------
% generate the output

varargout{1} = shift;
```

```
function [varargout] = error_sum(varargin)

%****************************************
% ERROR SUMMER
%
% This function calculate the bit and packet error rates
% by comparing the received sequence
% to the transmitted sequence
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************


%-------------------------------------------------
% get input variables

channel_type = varargin{1};
N = varargin{2};
EbNo = varargin{3};
SNR = varargin{4};
channel_errors = varargin{5};
total_channel_errors = varargin{6};
packet_errors_hdd = varargin{7};
packet_errors_sdd = varargin{8};

%-------------------------------------------------
%generate an error output file

title = ['n   '; 'ce  '; 'he  '; 'se  '];
nn = [1:N];
errors = num2str([nn; total_channel_errors'; ...
                        packet_errors_hdd'; packet_errors_sdd']);
error_summary = [title errors];

results = [1:N; total_channel_errors'; ...
              packet_errors_hdd'; packet_errors_sdd'];

c = sum(results(2,:));
chd = sum(results(3,:));
csd = sum(results(4,:));
p = length(find(results(2,:)>.1));
phd = length(find(results(3,:)>.1));
psd = length(find(results(4,:)>.1));

details = results;
summary = [channel_type; N; EbNo; SNR; ...
                  c; chd; csd; p; phd; psd];

% generate output
varargout{1}=summary;
```

```
function varargout = upsampler(varargin)

%****************************************
% UP SAMPLER
%
% This function upsamples the binary sequences passed to it
% by the user defined amount
% no filtering is performed instead
% 1s or 0s are appropriately inserted into the sequence
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************


data = varargin{1};
M = varargin{2};

samples = ones(1,M);
matrix = samples' * data;
out = reshape(matrix,1,prod(size(matrix)));

varargout{1} = out;
```

```
function [varargout] = config_error_check(oper,varargin);

%****************************************
% CONFIGURATION ERROR CHECHER
%
% This function confirms that the transmitter and receiver
% modulation parameters have been setup the same
%
% developed by Peter Duke September 2002
% last modified 2/9
%****************************************

load ('modulation_params');
load ('acqframe_params');

switch oper

%-------------------------------------------------
case 'modulator'

data = varargin{1};

% check correct modulation
if modulation_method~=1 & modulation_method~=2
   error(sprintf('ERROR\n Incorrect Modulator Being Used\n change modem_setup.m file or use another
modulator'));
end

% check spreading factor is an integer
if ( chips_per_bit ~= fix( chips_per_bit ) );
   error(sprintf('ERROR\n spreading factor is not an integer\n change Rb and/or Rc in the
modem_setup.m'));
end

% check packet length
if ( packet_length ~= length(data) );
   error(sprintf('ERROR\n Receiver is expecting different packet length\n packet length must be =
2*(Ndata bits + constraint length -1)\n option1 - change packet length in modem_setup.m\n option2 -
change the number of data bits\n'));
end

%-------------------------------------------------
case 'demod'

% check correct modulation
if modulation_method~=1 & modulation_method~=2
   error(sprintf('ERROR\n Incorrect Modulator Being Used\n change modem_setup.m file or use another
modulator'));
end

end
```

LIST OF REFERENCES

[1] J. A. Rice, R. K. Creber, C. L. Fletcher, P .A. Baxley, D .E. Rogers, and D. C. Davsion, "Seaweb Underwater Acoustic Nets," *Space and Naval Warfare Systems Center San Diego Biennial Review 2001*, pp. 234-243, 2001.

[2] P. A. Baxley, H. Bucker, V. K. McDonald, and J. A. Rice, "Shallow-Water Acoustic Communications Channel Modeling using Three-Dimensional Gaussian Beams," *Space and Naval Warfare Systems Center San Diego Biennial Review 2001*, pp. 251-262, 2001.

[3] R. J. Urick, *Principles of Underwater Sound,* 3rd ed. New York: McGraw-Hill, 1983.

[4] F. B. Jensen, W. A. Kuperman, M. B. Porter, and H. Schmidt, *Computational Ocean Acoustics,*. New York: Springer-Verlag, 2000.

[5] R. Coates, *Underwater Acoustic Systems.* New York: John Wiley & Sons Inc, 1989.

[6] L. Brekhovskikh and Y. Lysanov, *Fundamentals of Ocean Acoustics.* Berlin: Springer-Verlag, 1982.

[7] B. Sklar, *Digital Communications,* 2nd ed. New Jersey: Prentice Hall, 2001.

[8] T. S. Rappaport, *Wireless Communications Principles and Practice,* 2nd ed. New Jersey: Prentice Hall, 2002.

[9] J. A. Catipovic, "Performance Limitations in Underwater Acoustic Telemetry," *IEEE Journal of Oceanic Engineering,* Vol. 15, pp. 205-216, 1990.

[10] D. B. Kilfoyle and A. B. Baggeroer, "The State of the Art in Underwater Acoustic Telemetry," *IEEE Journal of Oceanic Engineering,* Vol. 25, pp. 4-27, 2000.

[11] M. Stojanovic, J. G. Proakis, J. A. Rice, and M. D. Green, "Spread Spectrum for Underwater Acoustic Telemetry," presented at IEEE OCEANS'98 Conference, Nice, France, September 1998.

[12] J. G. Proakis, *Digital Communications,* 4th ed. New York: McGraw-Hill, 2001.

[13] J. De Nayerlaan, "Spread Spectrum (SS) introduction," [http://www.sss-mag.com/pdf/Ss_jme_denayer_intro_print.pdf], September 2002.

[14] R. L. Peterson, R. E. Ziemer, and D. E. Borth, *Introduction to Spread Spectrum Communications.* New Jersey: Prentice Hall, 1995.

[15] R. E. Ziemer, *Introduction to Digital Communication,* 2nd ed. New Jersey: Prentice Hall, 2001.

[16] E. M. Sozer, J. G. Proakis, M. Stojanovic, J. A. Rice, A. Benson, and M. Hatch, "Direct Sequence Spread Spectrum Based Modem for Under Water Acoustic Communication and Channel Measurement," presented at IEEE OCEANS'99 Conference, Seattle, WA, September 1999.

[17] A. J. Viterbi, *CDMA Principles of Spread Spectrum Communication.* Massachusetts: Addison-Wesley, 1995.

[18] S. B. Wicker, *Error Control Systems for Digital Communications and Storage.* New Jersey: Prentice Hall, 1995.

[19] K. Karkkainen, "Optimized PN Sequences Available for Simulation of CDMA Systems," [http://www.ee.oulu.fi/~kk/], September 2002.

[20] M. B. Porter, "Ocean Acoustics Library," [http://oalib.saic.com/], September 2002.

[21] G. C. Clarke, Jr and J. B. Cain, *Error-Correction Coding for Digital Communications.* New York: Plenum Press, 1981.

[22] D. Hanselman and B. Littlefield, "Mastering Matlab Toolbox Version 6," [http://www.eece.maine.edu/mm/MM6/tbx.html], September 2002.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman, Code EC
   Electrical and Computer Engineering Department
   Naval Post Graduate School
   Monterey, California

4. Dr. Roberto Cristi, Code EC/Cx
   Electrical and Computer Engineering Department
   Naval Postgraduate School
   Monterey, California

5. Joe Rice, Code PH/Rj
   Physics Department
   Naval Post Graduate School
   Monterey, California