NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A CODED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING SIMULATION OF A HIGH DATA RATE, LINE-OF-SIGHT, DIGITAL RADIO FOR MOBILE MARITIME COMMUNICATIONS

by

David V. Roderick

June 1997

Thesis Advisor Paul H. Moose

Approved for public release; distribution is unlimited.

19980102 045

DTIC QUALITY INSPECTED 4

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 June 1997	3. REPORT TYPE AND DATES COVERED Engineer Degree

4. TITLE AND SUBTITLE TITLE OF THESIS A CODED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING SIMULATION OF A HIGH DATA RATE, LINE-OF-SIGHT, DIGITAL RADIO FOR MOBILE MARITIME COMMUNICATIONS

5. FUNDING NUMBERS

6. AUTHOR(S) David V. Roderick

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),
Naval Command, Control and Ocean Surveillance Center, San Diego, CA 92152-5001

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 unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)* The Naval Command, Control and Ocean Surveillance Center (NCCOSC), Research Development Test and Evaluation (RDT&E) Division's (NRaD) Communications Department is conducting applied research toward the development of a high-data-rate (HDR), line-of-sight (LOS), digital modem for ship-to-ship, ship-to-shore, and ship-to-relay communications. Development of bandwidth efficient HDR communications in a maritime radio environment is a challenging research problem due to the time-varying propagation effects within the marine layer. Marine layer propagation typically causes fading of the signal spectrum due to RF interference effects, and intersymbol interference because of multipath induced time spreading. The use of adaptive equalization to overcome distortions is difficult in this environment because of the dynamic nature of the signal propagation caused by transmitter and/or receiver motion and the maritime layer atmospheric effects. An alternative to channel equalization is the application of Coded Orthogonal Frequency Division Multiplexing (COFDM) which overcomes distortion effects without equalization through its orthogonality properties. This thesis explores the application of COFDM toward a HDR LOS maritime communications modem. The modem model is emulated in MATLAB and simulations are performed. Analysis of the simulations are conducted and evaluated as to the feasibility of a COFDM implementation in the presence of known noise and signal fading conditions.

14. SUBJECT TERMS. High-data-rate (HDR), Line-of-sight (LOS), Digital, Modem, Communications, Coded Orthogonal Frequency Division Multiplexing (COFDM), Frequency Division Multiplexing (FDM), MATLAB, Reed-Solomon Code, Differential Encoding, Interleaver, Multipath, Additive White Gaussian Noise (AWGN), Fast Fourier Transform (FFT), M-ary Phase Shift Keyed (M-PSK)

15. NUMBER OF PAGES
340

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

Approved for public release; distribution is unlimited.

A CODED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING SIMULATION OF A HIGH DATA RATE, LINE-OF-SIGHT, DIGITAL RADIO FOR MOBILE MARITIME COMMUNICATIONS

David V. Roderick

B.S.E.E., University of New Haven, West Haven CT, 1987
M.S.E.E., University of Maryland, College Park MD, 1992

Submitted in partial fulfillment of the
requirements for the degree of

ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
June 1997

Author: David V. Roderick

David V. Roderick

Approved by: Paul H.

Paul H. Moose, Thesis Advisor

R. Clark Robertson

R. C. Robertson, Second Reader

Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The Naval Command, Control and Ocean Surveillance Center (NCCOSC), Research Development Test and Evaluation (RDT&E) Division's (NRaD) Communications Department is conducting applied research toward the development of a high-data-rate (HDR), line-of-sight (LOS), digital modem for ship-to-ship, ship-to-shore, and ship-to-relay communications. Development of bandwidth efficient HDR communications in a maritime radio environment is a challenging research problem due to the time-varying propagation effects within the marine layer. Marine layer propagation typically causes fading of the signal spectrum due to RF interference effects and intersymbol interference because of multipath induced time spreading. The use of adaptive equalization to overcome distortions is difficult in this environment because of the dynamic nature of the signal propagation caused by transmitter and/or receiver motion and the maritime layer atmospheric effects. An alternative to channel equalization is the application of coded orthogonal frequency-division multiplexing (COFDM) which overcomes distortion effects without equalization through its orthogonality properties. This thesis explores the application of COFDM toward a HDR LOS maritime communications modem. The modem model is emulated in MATLAB and simulations are performed. Analysis of the simulations are conducted and evaluated as to the feasibility of a COFDM implementation in the presence of known noise and signal fading conditions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADC	Analog-to-Digital Conversion
A/D	Analog-to-Digital
AM	Amplitude Modulation
ARG	Amphibious Readiness Group
AWGN	Additive White Guassian Noise
BAA	Broad Agency Announcement
BER	Bit Error Rate
BG	Battle Group
CDL	Common Data Link
COFDM	Coded Orthogonal Frequency-Division Multiplexing
COTS	Commercial Off-the-Shelf
CW	Continuous Wave
DAC	Digital-to-Analog Conversion
dB	Decibel
DFT	Discrete Fourier Transform
FDM	Frequency Division Multiplexing
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FM	Frequency Modulation
HDR	High-Data-Rate
HDTV	High Definition Television
IC	Integrated Circuit
ICI	Intercarrier Interference
IFFT	Inverse Fast Fourier Transform
ISI	Intersymbol Interference
Kbps	Kilobits per second

kHz	kilohertz
Km	Kilometer
LOS	Line-of-Site
M-DPSK	M-ary Differential Phase Shift Keyed
Mbytes	Megabytes
MHz	megahertz
modem	Modulator/Demodulator
NCCOSC	Naval Command Control and Ocean Surveillance Center
OFDM	Orthogonal Frequency Division Multiplexing
PC	Personal Computer
PSK	Phase Shift Keyed
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keyed
RAM	Random Access Memory
RDT&E	Research Development Test and Evaluation
RF	Radio Frequency
R-S	Reed-Solomon
RSL	Received Signal Level
$SI_{ac}R$	Signal-to-adjacent interference channel ratio
SOI	Signal of Interest
µsec	Microseconds
UHF	Ultra High Frequency
Var	Variance

ACKNOWLEDGEMENTS

This work is supported by the Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E division's (NRaD) Communications Department. I thank Dr. Paul H. Moose, my thesis advisor, for his professional, theoretical and technical guidance. His communications experience and insight was highly motivating, and encouraged me to complete this research. I also thank Dr. Clark Robertson, my second reader, who offered assistance at various times along this journey. Lastly, I am grateful to the Naval Postgraduate School for establishing an exceptional educational program allowing students to enhance their technical expertise and promote educational career growth all within the spirit of the Navy military atmosphere.

I. INTRODUCTION

A. BROAD AGENCY ANNOUNCEMENT FOR A WIRELESS MODEM

The Naval Command, Control and Ocean Surveillance Center (NCCOSC), Research Development Test and Evaluation (RDT&E) Division's Communications Department is conducting applied research towards the development of a HDR, LOS, digital communications system for ship-to-ship (link 1), ship-to-shore (link 2), and ship-to-relay (link 3) type connectivity. The goal of the applied research is to develop a wireless communications network within a Battle Group (BG) or Amphibious Readiness Group (ARG) allowing high capacity voice, video and data transmissions among the platforms as well as the ability to link together the assets of each of the platforms. The added robustness of the asset connecting infra-structure will allow separate platforms without HDR capabilities to gain connectivity to a platform that does have HDR capability. (Fig. 1)

The primary objective of NRaD's Broad Agency Announcement (BAA) [1] is to develop a radio frequency (RF) modulator/demodulator (modem) with the capability of transmitting full-duplex 1536 kilo-bits-per-second (kbps) in the Naval maritime environment using a 600 kHz frequency channelization [1]. This requires that the 3dB bandwidth of the transmitted signal be less than 480 kHz. It is essential that the modem demonstrate reliable communication at useful ranges between mobile platforms such as Navy ships, helicopters, and sub-sonic fixed wing aircraft as well as various shore sites. Presently the modem is planned to transmit and receive HDR signals within the 225-400 MHz band, with the possibility of using the higher Ultra-High Frequency (UHF) spectrum 1350-1850 MHz band. It is also desirable to have an open non-proprietary system allowing for inter-operability with other existing Navy, Army and Air Force radio systems.

In order to allow reconfiguration of the HDR LOS modem for different applications and platforms, the corresponding RF system performance objectives for reliable communication are indicated as follows:

1. Continuous transmit/receive of a full-duplex n x 64 kbps link using two (one transmit, one receive) blocks of n x 25 kHz-wide contiguous frequency channels where the values of n are (at least) n = 1,2,4,8,9,12,16,24,32. This will require the 3 dB bandwidth of the transmitted signal to be $BW_{3dB} \leq n \times 20$ kHz or a 3 dB bandwidth efficiency of better than 3.2 bps/Hz.

2. 99% reliable, 1536 kbps data under each of the following three typical channels:
   a. Link 1: Ship-to-ship (Propagation Loss=130 dB, Cable/Misc. Loss = 5 dB)

      Path #1: Ricean, $F_d = 1$ Hz

      Path #2: Rayleigh, $T_{1-2} = 0.01$ μsec, $F_d = 10$ Hz, $L_{1-2} = -6$ dB

   b. Link 2: Ship-to-shore (Propagation Loss=130 dB, Cable/Misc. Loss = 5 dB)

      Path #1: Ricean, $F_d = 10$ Hz

      Path #2: Rayleigh, $T_{1-2} = 0.07$ μsec, $F_d = 10$ Hz, $L_{1-2} = -5$ dB

      Path #3: Rayleigh, $T_{1-3} = 0.80$ μsec, $F_d = 10$ Hz, $L_{1-3} = -15$ dB

   c. Link 3: Ship-to-relay (Propagation Loss=130 dB, Cable/Misc. Loss = 5 dB)

      Path #1: Ricean, $F_d = 25$ Hz

      Path #2: Rayleigh, $T_{1-2} = 0.9$ μsec, $F_d = 25$ Hz, $L_{1-2} = -3$ dB

      Path #3: Rayleigh, $T_{1-3} = 5.1$ μsec, $F_d = 25$ Hz, $L_{1-3} = -9$ dB

3. Bit Error Rate (BER) of the signal of interest (SOI) should be less than 10^{-6} when the signal-to-adjacent interference channel ratio ($SI_{ac}R$) = -15 dB for carrier-to-carrier spacing of D_{fc} = n x 25 kHz (adjacent channel), $SI_{ac}R$ = -40 dB for D_{fc} = 2n x 25 kHz (one channel between carriers), $SI_{ac}R$ = -65 dB for Dfc = 3n x 25 kHz (two channel between carriers), and $SI_{ac}R$ = -90 dB for D_{fc} = 4n x 25 kHz (three channel between carriers).

4. Accommodate a minimum transmit-receive carrier frequency separation of $|f_{c\text{-}Tx} - f_{c\text{-}Rx}| = 5n \times 25$ kHz on any platform without any significant degradation in system BER performance.
5. BER of SOI should be less than 10^{-6} with one in-band narrowband interferer (CW, FM voice, or AM voice) with SIR = -10 dB or with two in-band narrowband interferers each with SIR = -5 dB when tested in each of the channels described above for $n \geq 8$. Reduced performance is expected for FM and AM voice for $n < 8$; however, CW interference rejection should not degrade for $n < 8$.

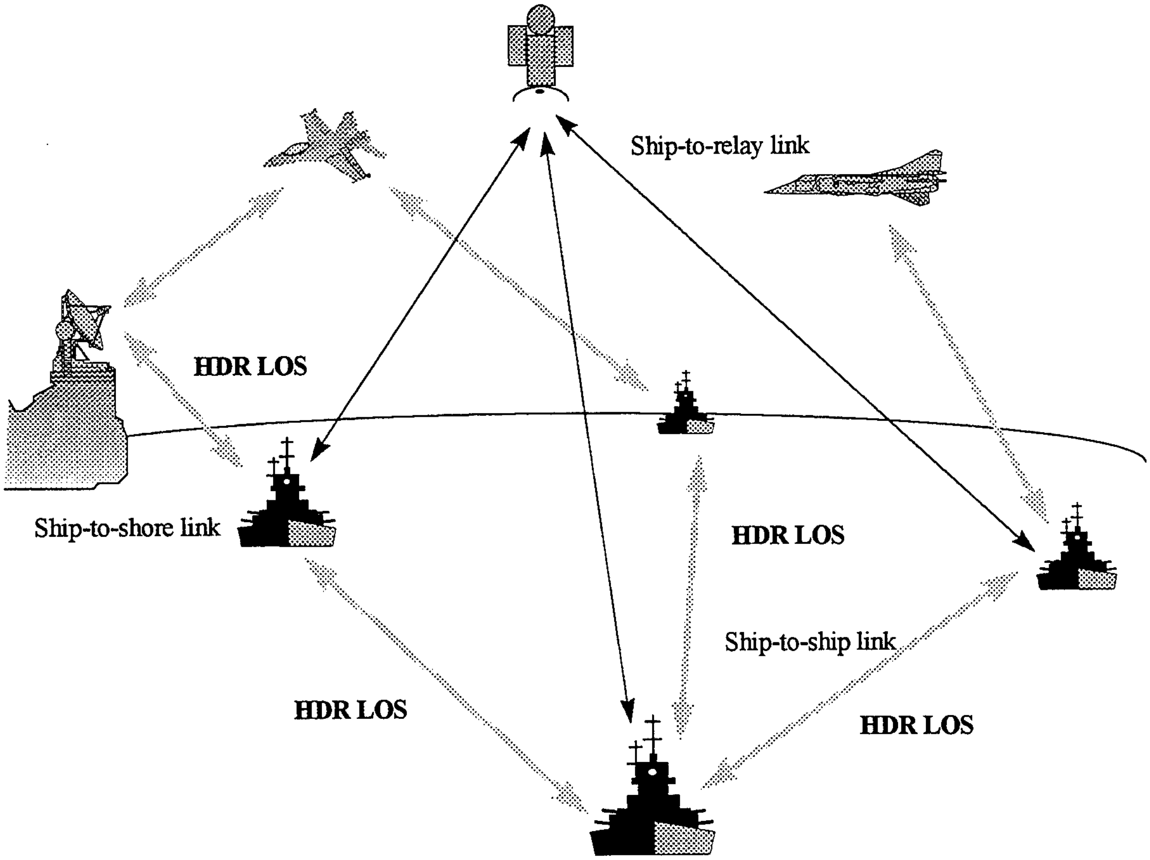


Fig. 1. Maritime Wireless Network

Some of the factors that will impact the performance parameters previously mentioned include the dynamics of the mobile communications channel, adjacent channel interference, co-channel interference, in-band narrowband interference, and the characteristics of the RF equipment. Within the maritime environment the LOS communication channel among mobile platforms in the UHF band is characterized by multiple propagation paths each with time-varying statistics. There is a wide variation of a few nanoseconds (nsec) up to a few microseconds (μsec) of the delay spread between propagation paths. This spread is based on the range between platforms and whether the connection is ship-to-ship, ship-to-shore, or a ship-to-air relay link. With the existence of multiple propagation paths within the maritime environment arriving at the receiver with varying time delays, referring to BAA notation, $T_{1\text{-}P}$ is the time delay in microseconds between the first and Pth path with respect to the particular communication link.

Typically, the short-term statistics of the received signal level (RSL) are characterized by a Rayleigh or Ricean distribution with possible fade rates ranging from 0-75 Hz, fade depths ranging from 0-40 dB, and maximum delay spreads of 0 to 15 microseconds. With regard to the multiple transmission paths and associated RSL power fading, $L_{1\text{-}P}$ is the difference in the mean signal level between the first and the Pth path. Furthermore, Ricean fading is defined by a non-fading LOS component, Doppler shifted by 0.7 F_d, with equal signal power to a Rayleigh fading signal with fade rate F_d, where F_d is the associated Doppler frequency. A Rayleigh fading signal is defined as in EIA/TIA IS-55. As carrier frequency increases, fade rates tend to increase and are also dependent on the absolute and relative velocities of the two communicating platforms and surrounding environmental conditions. It also has been demonstrated experimentally that fade rates are dependent on the range between platforms and become more severe with increased range. For purposes of thesis research, the RSL statistics specified in the BAA [1] including Doppler frequency shifting, F_d, multipath power fading, $L_{1\text{-}P}$, and multipath time delays, $T_{1\text{-}P}$, will represent the COFDM emulation model baseline with respect to the transmission links of interest (links 1 though 3) and their corresponding multipath parameters.

B. PROPOSED COFDM TECHNIQUE IN RESPONSE TO BAA

The design of a reliable HDR LOS modem in the maritime environment is challenging largely due to time-varying multipath propagation effects. As previously indicated, frequency dependent fading of the signal spectrum is a constant problem, adversely affecting reliable communications. In single carrier systems, multipath induced time spreading can cause intersymbol interference among data symbols and generate symbol errors during the message decoding process. Furthermore, random dynamic motions of the transmitter and/or receiver (moving communicating platforms) and/or atmospheric effects within the maritime layer make adaptive equalization of the channel difficult to accomplish.

In response and support of the BAA issued by NRaD and as an alternative to traditional single carrier compensation techniques such as adaptive equalization, a proposed solution is the design of a signaling scheme that does not require equalization for transmission in a spread channel. One such modulation technique is COFDM. COFDM is considered a practical effective method to meet requirements, and has already been implemented for digital broadcasting in Europe with further consideration for inclusion in terrestrial digital television and HDTV broadcasting [2]. In contrast to traditional frequency-division multiplexing (FDM), which occupies separate non-overlapping sub-bands of the overall spectrum bandwidth, COFDM utilizes orthogonal multiple carriers (typically in the order of hundreds) with mutually overlapping spectrums, thus providing for greater spectral efficiency. Interleaving the information symbols in time combined with forward error correction (FEC) permits recovery of corrupted symbols in the receiver after transmission through a Rayleigh fading channel. In this way COFDM represents a technique for bandwidth efficient, high data rate transmissions through a frequency selective fading channel with additive white Gaussian noise (AWGN).

C. THESIS OBJECTIVES AND ORGANIZATION

The objective and motivation for conducting this research is to document and evaluate computer simulation results of a software emulated communication system utilizing COFDM techniques designed to operate in a maritime environment. The compiled results will be used by NRaD to evaluate hardware design proposals responding to the BAA in addition to judging the overall feasibility of a COFDM based system. NRaD has provided the maritime channel model specifications indicative of actual environmental conditions that the physical hardware (transmitters and receivers) will be subjected to and operated within [1]. These specifications are incorporated into the simulated channel models. Emulation and simulation of the communication system model is performed at the Naval Postgraduate School using MATLAB® on a personal computer (PC) platform (Pentium® 200 MHz PC) with 64 megabytes (Mbytes) of RAM and using the Windows 95® operating system.

MATLAB is a mathematical software package developed by the Math Works, Inc. of Natick, MA which enables numerical analysis, matrix computation and signal processing through a graphical interface. As part of the basic software package, MATLAB includes elementary signal processing functions in a tool box kit in addition to basic mathematical commands. Using these rudimentary functions and commands, one can create higher level modules to form subroutine “m-files”. The hierarchy of m-files are combined in such a way as to represent the functional blocks of the system and emulate the overall communication model through which baseband signal level simulations are performed. Simulation results are typically presented in graphical form displaying error distributions and symbol error rates (SER) for different channel configurations and with different system input parameters and constraints. Comparisons of simulation results to known theoretical and previously modeled system performance standards enable the formation of a software model baseline standard. From the baseline, conclusions can be drawn concerning implementation feasibility and, consequently, comparisons made of future physical modem hardware to the emulated model simulation results.

This thesis is organized as follows: Chapter II discusses the maritime channel model background and theory in terms of AWGN and multipath; Chapter III reviews conventional FDM concepts and introduces COFDM theory and related topics; Chapter IV presents the COFDM system level block diagram used in the emulation model and discusses the operation of each functional block; Chapter V presents the MATLAB coded block descriptions that form the software model as well as higher level simulation diagnostic and batch programs; Chapter VI discusses the simulation test plan methodology and reports simulation trial results; Chapter VII concludes with an evaluation of the simulation results, discusses overall feasibility and presents possible future related work; Appendix A. lists the complete MATLAB code used including system macros, diagnostic verification programs and batch m-files; Appendix B. presents a compilation of performance results for various system configurations.

II. NOISE AND THE MARITIME CHANNEL

A. MULTIPATH CHANNEL DESCRIPTION AND MODEL

Maritime environmental factors can have an adverse impact on reliable HDR LOS communications between ship-to-ship, ship-to-shore and ship-to-air platforms. The most prominent type of error producing phenomenon present is multipath fading which is frequency dependent. Multipath fading exists when there is more than one transmission path between transmitter and receiver and is characterized by variations of the receive signal level (RSL) from the free-space calculated level for a particular far end transmitter output. The propagation mechanisms that affect fading are atmospheric refraction, reflections from objects, scattering of radio energy, focusing attenuation, and other various meteorological and geographical factors. The received signal may consist of several discrete paths, each with a different attenuation and time delay, or a continuum of paths all of which either constructively and/or destructively combine at the receiver. At times the multiple delayed signals add destructively to reduce the power level of the received signal, while at other times they add constructively and augment the signal. In extreme situations multipath induced deep fading, known as power fading, can result in complete loss of the signal. Another manifestation of multipath in a digital carrier receiver is a form of signal interference referred to as intersymbol interference (ISI), causing detection errors [3].

Power fading is characterized as dramatic decreases from the free-space signal level for extended periods of time. It is possible for multipath power fading to exhibit fades greater than 30 dB for periods of seconds or minutes. This type of fading occurs during quiet, windless and foggy nights, when temperature inversion near the surface occurs and not enough wind turbulence is present to mix the air. The result is the formation of elevated or surface based stratified layers. The maritime environment is particularly conducive to multipath due largely to the high number of unobstructed ocean reflections. In contrast to land, the open ocean typically does not contain protruding vegetation or other projecting obstructions that tend to break up the reflection components as is often the case with terrestrial based radio link paths. [4]

Multipath propagation also manifests itself as time dispersion resulting from differences in transmitter and receiver transit times among multiple propagation paths with different lengths. Time dispersion is characterized by a delay power spectrum and is measured as multipath delay spread in microseconds. Time dispersion is particularly harmful to digital communications since excessive signal delay spread causes ISI and prevents correct bit detection in the receiver, degrading overall bit error rate (BER) performance. One possible way to overcome the effects of delay time and resulting signal spread is to decrease the bit rate.

The effects of multipath spread causes variations in amplitudes and phases of the signal frequency spectrum due to the continuous interference from multipath wave components. When the fluctuations are correlated within the signal bandwidth so that all the spectral components behave in a similar fashion, one has what is referred to as frequency non-selective, or flat fading. If the fluctuations have little correlation across the band, then the result is frequency-selective fading.

Another type of dispersion is frequency dispersion or Doppler spreading which usually is present along with time dispersion. Doppler frequency spread can be due to atmospheric conditions as well as relative continuous motions between transmitters and receivers and is measured in Hertz. If the rate at which the received signal is slowly changing with time, then the Doppler frequency spread is relatively small; conversely, if there is rapid time change then the Doppler frequency spread is large.

One frequently used model to represent a time-variant multipath channel is depicted in Fig. 2 [5]. This model is composed of a delay line with multiple taps. Tap coefficients are typically modeled as complex-valued, Gaussian random processes that are mutually uncorrelated. The delay line length, T_m, corresponds to the amount of time dispersion in the multipath channel, also known as the multipath spread, and is given by,

$$T_m = \sum_{i=1}^{l-1} \tau_i \, , \tag{1}$$

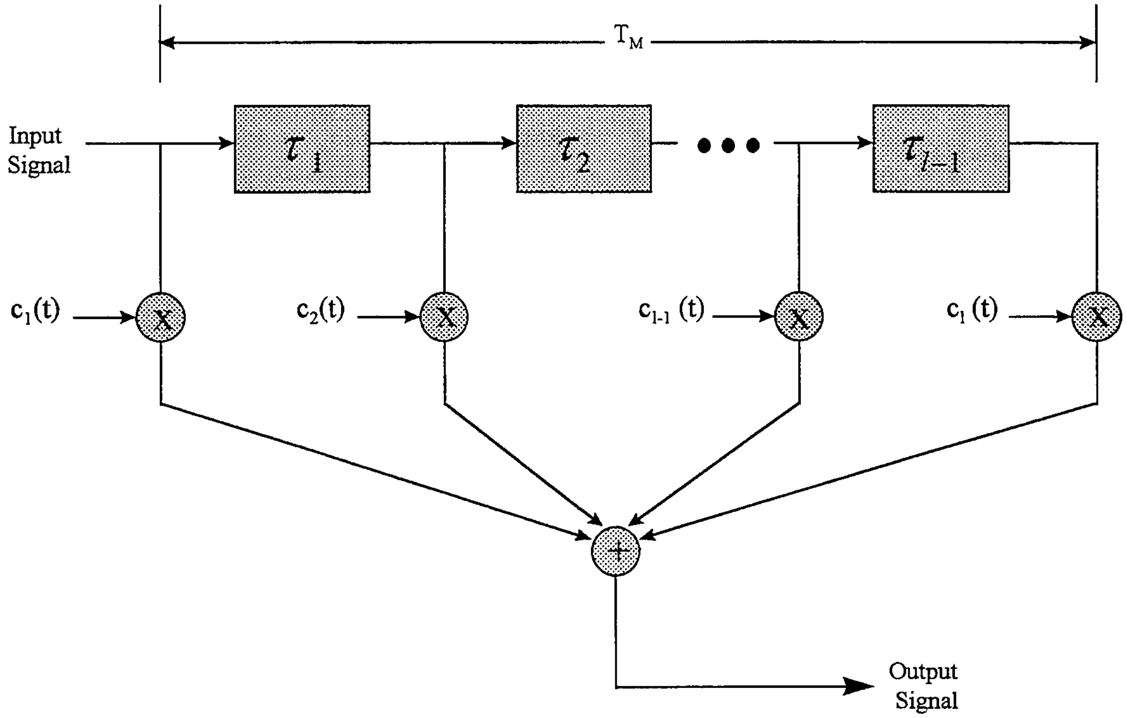


Fig. 2. Time Varying Multipath Channel Model Using Tapped Delay Line

B. ADDITIVE WHITE GAUSSIAN NOISE DESCRIPTION AND MODEL

Along with the distortions caused by a multipath channel exhibiting memory, the maritime environment also includes the effects of thermal noise normally modeled as additive white Gaussian noise (AWGN). AWGN can be described as a zero mean Gaussian process, n(t), with a uniform power spectral density given as

$$S_n(f) = \frac{N_o}{2} \quad \text{watts/Hz} \tag{2}$$

The Var(n(t)) is defined as the variance of the Gaussian noise process, n(t). The noise is described as "additive" because it is simply added to the signal transmitted through the channel [6]. Fig. 3 demonstrates the baseband model used for AWGN where s(t) is the transmitted signal through the channel, n(t) is the added noise process and r(t) is the received signal.

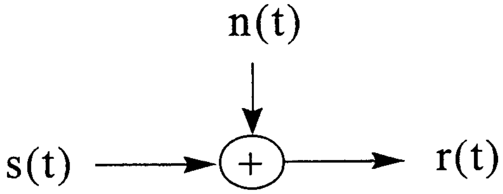


Fig. 3. AWGN Baseband Model

It can also be shown that s(t) and n(t) may be represented on a symbol interval $[0,T_s]$ in terms of a suitable orthonormal basis set $\left\{\Psi_j(t)\right\}$. For a bandpass signal representation such as MPSK, one such basis set is $\left\{\Psi_I(t), \Psi_Q(t\right\}$, where,

$$\Psi_I(t) = \left(2/T_s\right)^{1/2} \cos\left(2\pi f_c t\right) \tag{3}$$

and

$$\Psi_Q(t) = -\left(2/T_s\right)^{1/2} \sin\left(2\pi f_c t\right). \tag{4}$$

Thus, bandpass noise that interferes with the signal can be represented in terms of the signal space basis functions as

$$\tilde{n}(t) = n_I \Psi_I(t) + n_Q \Psi_Q(t) \tag{5}$$

with $\tilde{n}(t)$ being the projection of the noise n(t) onto the signal space

$$n_I = \int_0^{T_s} n(t)\Psi_I(t)dt \tag{6}$$

and

$$n_Q = \int_0^{T_s} n(t)\Psi_Q(t)dt\,. \tag{7}$$

For AWGN it can be shown that $\{n_I, n_Q\}$ are uncorrelated, zero mean Gaussian random variables with variances, $\sigma^2_{I,Q}$, equal to $N_o/2$ [6]. Thus,

$$N_o = \sigma_I^2 + \sigma_Q^2. \tag{8}$$

The noise vector, after sampling at a rate of f_s samples per second for a period of time T_s, contains discrete complex values and is of length $N = f_sT_s$. Each complex valued element is an independent and identically distributed (iid) Gaussian random variable with real and imaginary parts that are also iid. The means are zero and the variances are all $\sigma_N^2 = N_0/2 = \sigma^2_{I,Q}$. With the signal symbol energy E_s defined in terms of s(t) as

$$E_s = \int_0^{T_s} s^2(t)dt = s_I^2 + s_Q^2 \tag{9}$$

the ratio of symbol energy to noise power is defined as

$$\mathrm{Es}/_{\mathrm{No}} = \frac{\int_0^{T_s} s^2(t)dt}{2\sigma_N^2} \qquad (10)$$

where it is also assumed that all symbols have the same average energy.

C. MARITIME ENVIRONMENTAL IMPACT ON LOS COMMUNICATION

The three LOS communication links of interest, ship-to-ship, ship-to-shore and ship-to-relay (high elevation), have been studied by NRaD personnel in San Diego, California. Data has been gathered on environmental conditions affecting reliable communications through a modem test model experiment. The reliability of the LOS digital radio is a function of the magnitude and variations of the received signal level (RSL). Reliability is defined as:

$$\% \text{ reliability} = 100\% - \% \text{ outage}, \qquad (11)$$

where outage is defined as either a CCITT G.821 Severely Errored Second or as a second during which synchronization of the information is lost [7]. Alternatively, outage may also be described as the percentage of a second in which the bit error rate (BER) is worse than 10^{-6}. For purposes of this research while conducting simulations, outage will be measured in terms of observed symbol error rates (SER) and total symbol interval lengths in seconds. Furthermore as part of the BAA objective, a 99% reliable link is defined as having 1% of outage time over 1 hour of operation (i.e., 36 seconds total accumulated outage time over 3600 seconds of test time).

The median RSL is determined by the total transmit power, free-space propagation loss, diffraction due to the earth radius, cable losses, antenna heights, losses and/or gains; however, over air the RSL randomly fluctuates about the median value. Empirical data gathered from the NRaD experiment indicate that the primary atmospheric conditions that affect the RSL are:

- Variations caused by the formation of an evaporation duct near the surface of the water acting as a transmission waveguide and whether or not the antenna is located within this duct.
- Refraction of the signal influenced by the troposphere causes flat, fast fading. This multipath interference tends to become more severe as the path length is increased.
- Multipath interference is also due to water surface reflections of the transmitted signal. These reflections tend to induce frequency-selective, slow fading which can also be a function of the sea state and the ocean reflection coefficient.
- Diffraction caused by the earth's radius tends to generate a shadowing effect as the signals bend away from the earth.

Based on data gathered and observations made in the oceanic vicinity of San Diego, it is inferred that the most prevalent influence on the RSL is due to reflective multipath with a strong likelihood of refractive multipath and vapor ducting occurring during the winter months. Fig. 4 depicts the atmospheric and maritime conditions that affect the RSL as described above.

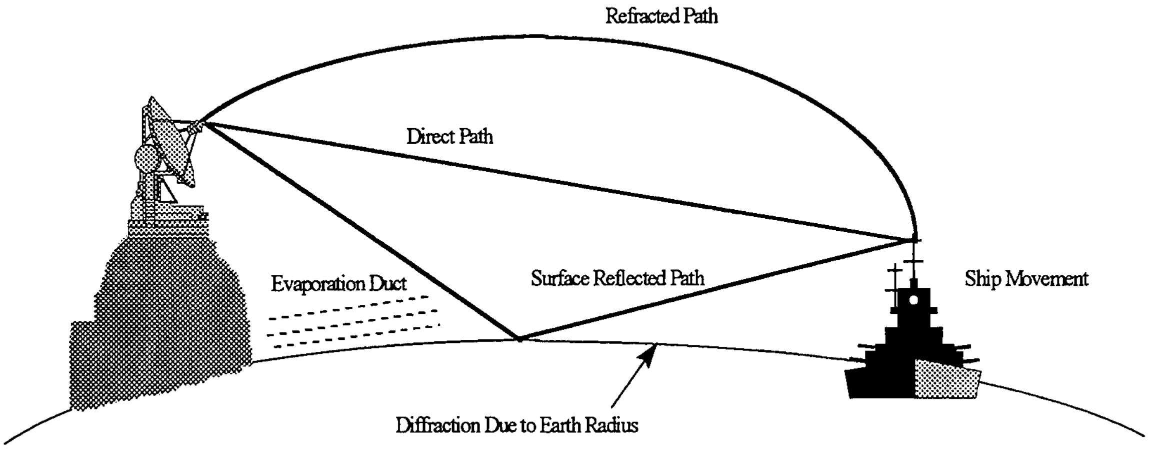


Fig. 4. Multipath in the Maritime Environment

Reflective multipath can be modeled as a two-path channel model consisting of a primary direct path and a secondary indirect surface reflected path. In reality, the location of the surface reflection area of the indirect path as well as the differential path length is a function of the earth's curvature. However, for calculation purposes, a flat earth approximation is assumed to determine the differential path length, $d_\Delta^{reflected}$, (meters) and channel delay spread, τ_Δ, (seconds) as follows:

$$d_\Delta^{reflected} = d_{reflected} - d_{direct} = \sqrt{(h_1 + h_2)^2 + d^2} - \sqrt{(h_1 - h_2)^2 + d^2}\ , \tag{12}$$

for $d >> h_1, h_2$,

$$d_\Delta^{reflected} \cong 2\frac{h_1 h_2}{d}\ , \tag{13}$$

and

$$\tau_\Delta = \frac{d_\Delta^{reflected}}{c} \tag{14}$$

where h_1 and h_2 are the two antenna heights, d is the horizontal range between antennas, and c is the speed of light ($\approx 3\text{x}10^8$ meters per second). Observations further indicate that reflected multipath delay spread decreases with increasing path distance. Reflections off smooth surfaces such as the ocean tend to cause small scattering which results in frequency-selective fading of the channel frequency response. For this type of fading, the null separation is given by

$$\text{null separation} = \frac{1}{\tau_\Delta}\ \text{(Hz)} \tag{15}$$

and the spectral peak-to-null difference is given by

$$\text{peak-to-null} = 20\log_{10}\frac{1+|\Gamma|}{1-|\Gamma|}\ \text{(dB)} \tag{16}$$

where $|\Gamma|$ is the magnitude of the reflection coefficient. Previous measurements have indicated that for calm sea the reflection coefficient can be as high as 0.98 resulting in a maximum spectral peak-to-null difference of 40 dB. For this experiment using transmission links 1 and 2, it was verified that the transmission spectrum is much less than the null separation given by equation (12), thus the channel is characterized primarily as a flat fading channel with as much as 40 dB of RSL variations depending on the sea state.

To a lesser degree, atmospheric refractive multipath also has a detrimental impact. The largest possible refraction angle is measured to be about three degrees. The differential path length between the maximum refraction angle, $\theta_{refracted}$, and the direct ray is given by:

$$d_{\Delta}^{refracted} = d_{refracted} - d_{direct} \leq d\left(\frac{1}{\cos\left(\theta_{refracted}^{o}\right)} - 1\right), \tag{17}$$

with increasing delay spread occurring with increasing distance.

Finally, ship and antenna movements also affect the RSL causing significant fade rates. On land, a common fade rate condition occurs when a vehicle is traveling at a hypothetical speed of 27 knots (50 km/hr) and transmitting to a fixed receiver at a carrier frequency of 300 MHz, resulting in a predicted fade rate of about 13 Hz. The determination of exact frequency fade rates for moving maritime vessels transmitting at specific carrier frequencies are apparently unknown at this time. Thus, for purposes of this study and thesis research, fade rates associated with maritime vehicle speeds are assumed to be comparable to land vehicle rates operating at similar carrier frequencies. At higher operating frequency ranges, the fade rates tend to increase. For this study a UHF transmission bandwidth of 225-400 MHz, is assumed.

III. CODED ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING CONCEPTS

A. SERIAL AND PARALLEL COMMUNICATION SYSTEMS

In a traditional serial data digital communication system, data is sent as a serial pulse train of information symbols. During the sequential transmission of each symbol through the channel, the symbol frequency spectrum is allowed to occupy the entire available bandwidth. However, in a multipath environment such as the maritime one, scattered reflections due to the ocean surface, platform structures, nearby obstacles and atmospheric factors, in addition to Doppler shifts caused by transmitter/receiver relative motions, make the signal envelope fluctuate. The time dispersion nature of the multipath channel also causes adjacent symbols of the serial stream to interfere when the symbols are short compared to the time spread. [2]

A parallel communication system differs from the serial counterpart by allowing the simultaneous transmission of several sequential data streams using much longer symbols. At any instance in time, there are many data elements (symbols) being transmitted through the channel. With this type of system, the individual spectrums of each data symbol occupy only a small portion of the overall available bandwidth. This approach is advantageous in spreading out the frequency-selective fade over many different symbols. Thus, instead of there being a high concentration of errors with several adjacent symbols being completely destroyed by the fade, the errors are spread out over many symbols and appear less bursty. In this situation, precise reconstruction of a majority of the symbols is possible even without the addition of error correcting codes. Additionally, in a parallel system, by partitioning the entire bandwidth into multiple non-overlapping frequency sub-bands (sub-channels), equalization of each sub-channel is much easier than the serial system because the symbols are now much longer than the time dispersion of the channel, which greatly reduces the effects of ISI.

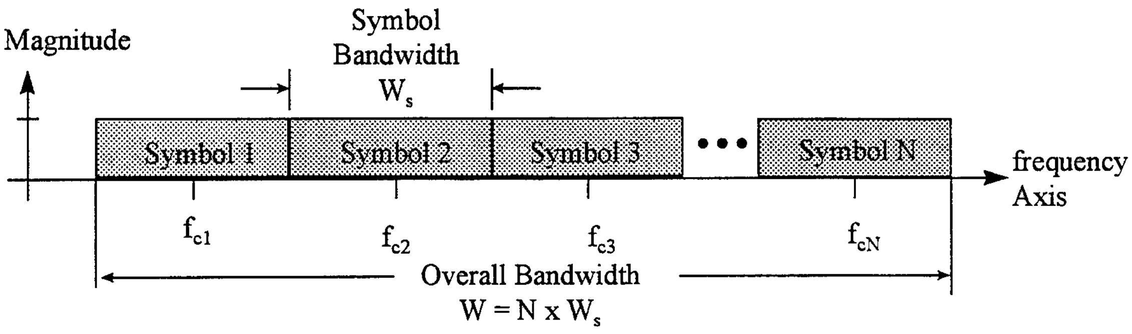


Fig. 5. Ideal Frequency-Division Multiplexing Spectrum

The approach to implementing a parallel communication system is done in different ways. In a classical parallel data system using conventional FDM technology (Fig. 5), the total signal frequency bandwidth is partitioned into N non-overlapping sub-channels. Each subchannel is modulated by a separate data symbol, and then each of the N sub-channels are frequency-division multiplexed for transmission. At the receiving end, separation of the sub-bands traditionally is accomplished by a bank of bandpass filters. However, due to the roll-off effect of physically realizable filters, the actual bandwidth of each sub-channel must be further widened. Sufficient guard bands must be inserted in the frequency spectrum between adjacent sub-channels to permit effective filtering without in-band signal attenuation and adjacent band signal interference. This method, with the addition of guard bands, does not offer the best possible spectrum efficiency (Fig. 6) since now the overall bandwidth is lengthened by multiple guard bands that do not carry any useful information. [3]

An alternative to traditional FDM is to implement a system utilizing staggered quadrature amplitude modulation (QAM) to increase efficiency of band usage [2]. Staggered QAM mimics the traditional FDM concept by using N sub-channels of modulated carriers with an additional excess sub-channel bandwidth of α. However, each adjacent subchannel overlaps their neighbors at the -3 dB frequency point allowing for a flatter, slightly more compact composite spectrum. Additionally, the filter design requirements for staggered QAM is less critical than FDM.

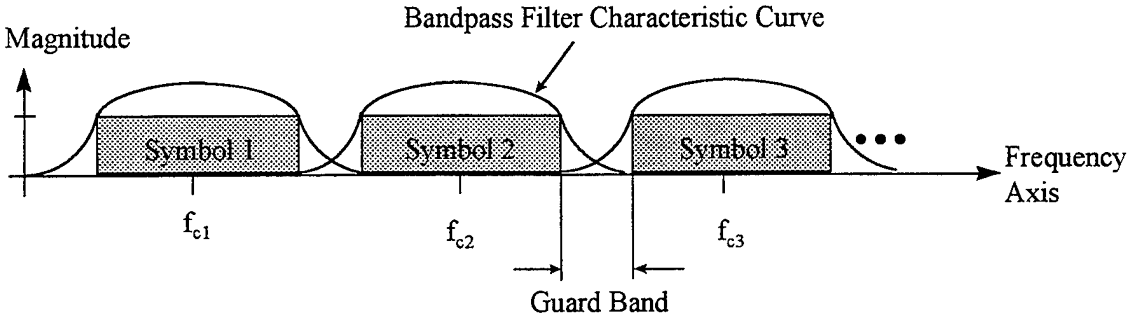


Fig. 6. Additional Guard Bands In Frequency Spectrum

Both these methods require individual sinusoidal generators to represent each of N sub-channels in the transmitter (carrier tones) and corresponding demodulators at the receiver. For a large number of sub-channels (N very large), the arrays of sinusoidal generators and coherent demodulators becomes unreasonably expensive, complex and space consuming, hence, not very practical. Thus, the principal objections to the use of parallel systems are the complexity and cost of the equipment required to implement the system. [8]

An alternative approach to conventional FDM and staggered QAM is a system that uses the discrete Fourier transform (DFT) to modulate and demodulate parallel data. Using the DFT in the transmitter, the individual sub-channel spectra can be represented with sinc functions which are not band-limited. Multiplexing of the sub-channels is accomplished by base-band processing instead of bandpass filtering.

One such technique which uses the DFT for implementation is orthogonal frequency-division multiplexing (OFDM), which is defined as a form of multi-carrier modulation where the carrier spacing is carefully selected so that each sub-carrier (tone) is orthogonal to the other sub-carriers. In order for a signal set to be orthogonal, any pair of sub-carriers must have a frequency separation of a multiple of $1/T_s$ [3]. OFDM differs from traditional FDM by allowing the OFDM spectrum of individual orthogonal subcarriers to mutually overlap; thus, a more optimum spectrum efficiency is gained over FDM. With the inclusion of coherent detection at the receiver and the use of orthogonal subcarrier tones separated by the reciprocal of the signaling element duration, independent separation of the multiplexed tones is possible, specifically by using the DFT.

B. OFDM THEORY

Consider a data sequence (D_0, D_1, D_2, … D_{N-1}), where each D_n is a complex number of the form $D_n = A_n + jB_n$. If a DFT is performed on the sequence, the result is a vector $\underline{d}$ = (d_0, d_1, d_2, … d_{N-1}) of N complex numbers with

$$d_m = \sum_{n=0}^{N-1} D_n \exp(-j(2\pi nm / N)) = \sum_{n=0}^{N-1} D_n \exp(-j(2\pi f_n t_m)), \qquad m = 0, 1, 2, \ldots N-1, \tag{18}$$

where

$$f_n \triangleq \frac{n}{N\Delta t}, \tag{19}$$

$$t_m \triangleq m\Delta t, \tag{20}$$

$$\Delta t \triangleq \frac{T_s}{N}, \tag{21}$$

and T_s is an arbitrary chosen symbol duration of the serial data sequence D_n [2],[8]. Taking the real part only of the $\underline{d}$ vector, we get the following components:

$$y_m = \sum_{n=0}^{N-1} A_n \cos(2\pi n f_n t_m) + \sum_{n=0}^{N-1} B_n \sin(2\pi n f_n t_m) \qquad m = 0, 1, 2, \ldots N-1. \tag{22}$$

Applying these components to an ideal low-pass filter with cutoff frequency $\frac{f_s}{2} = \frac{1}{2\Delta t}$, we now obtain the frequency division multiplexed signal

$$y(t) = \sum_{n=0}^{N-1} A_n \cos(2\pi n f_n t) + \sum_{n=0}^{N-1} B_n \sin(2\pi n f_n t) \qquad 0 \le t \le T_s. \tag{23}$$

As an illustration of a general OFDM based communication system using the orthogonality principle, Fig. 7 represents a block diagram of major system components with substitutions of more efficient fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) algorithms to reduce the number of operations from N^2 in the DFT down to approximately $\frac{N}{2}\log_2 N$ for the radix two FFT [9]. Initially, the incoming serial data bit stream is grouped to form symbols, q bits long, in preparation for a M-ary digital modulation scheme, where $M = 2^q$. Each symbol passes through a signal constellation mapper, such as 16-phase shift keyed (16-PSK) for example (for this case, $q = \log_2 M = \log_2 16 = 4$), to generate a complex modulation value, $\{D_N\}$, corresponding to a particular 4-bit symbol. The sequence of complex modulation values are converted from serial to parallel format by a multiplexer to form a block size of N symbols, where each member of N corresponds to a subcarrier frequency tone. The N complex modulation values are then modulated in baseband fashion by the IFFT performing the mapping into the time domain. Finally a multiplexer converts from parallel format to a serial data stream suitable for upconversion and RF transmission. Before the upconversion process can be accomplished, an analog-to-digital (A/D) converter is used to convert the discrete values to the analog equivalent and perform low-pass filtering. After transmission through the channel, the OFDM receiver portion of the system performs the inverse process of the transmitter. Specifically, downconversion and low-pass filtering is initially performed to recreate the baseband transmitted signal. The baseband serial data stream is converted to parallel forming N paths, which are fed to an FFT block. The N-point FFT operation recovers the complex modulation values, allowing the inverse signal mapper to generate the corresponding symbol bit pattern. The q-bit length symbols are multiplexed into a serial data stream to complete the process and recover the original information.

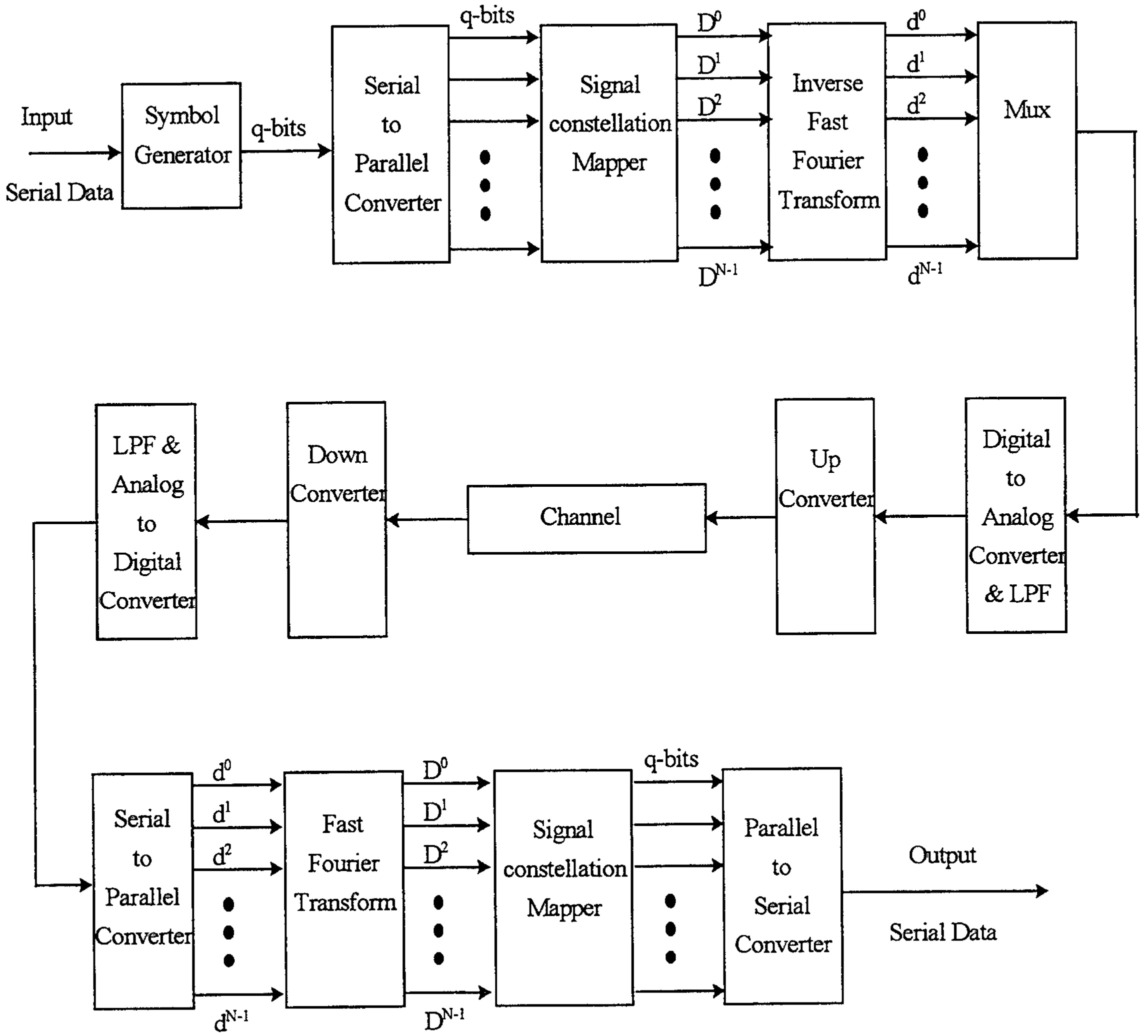

Input
Serial Data
Symbol Generator
q-bits
Serial to Parallel Converter
q-bits
Signal constellation Mapper
D0
D1
D2
DN-1
Inverse Fast Fourier Transform
d0
d1
d2
dN-1
Mux
Digital to Analog Converter & LPF
Up Converter
Channel
Down Converter
LPF & Analog to Digital Converter
Serial to Parallel Converter
d0
d1
d2
dN-1
Fast Fourier Transform
D0
D1
D2
DN-1
Signal constellation Mapper
q-bits
Parallel to Serial Converter
Output
Serial Data

Fig. 7. General OFDM Communications Model

During the signal constellation mapping stage, each data symbol is encoded as a truncated sinusoid within the interval $(0,T_s)$. Signal truncation causes the frequency response of y(t) to be a sinc function. As seen in Fig. 8, the spectral shape of an OFDM subchannel contains zero crossings at multiples of $1/T_s$. The other sub-carriers are generated by the IDFT in such a way that their spacing generates a nearly flat overall spectrum with no interference among individual spectra. For example, an OFDM spectrum would be similar to the one depicted in Fig. 9. In this figure the orthogonality of the subcarriers is demonstrated by the overlapping of individual subcarrier spectra at their respective zero crossings; thus, the spectra of the individual subchannels are zero at the other subcarrier frequencies.

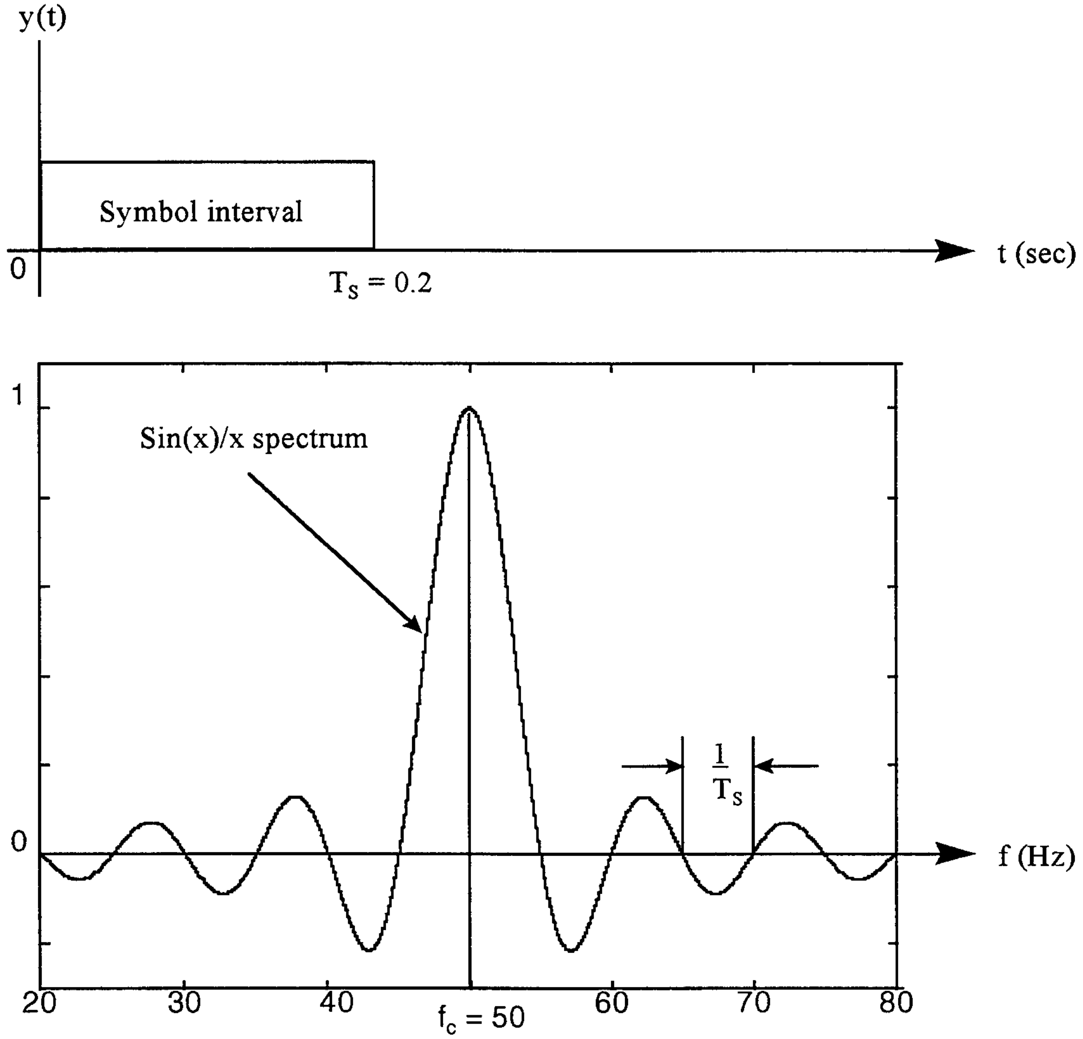


Fig. 8. Spectrum For Single Symbol With Length T_S

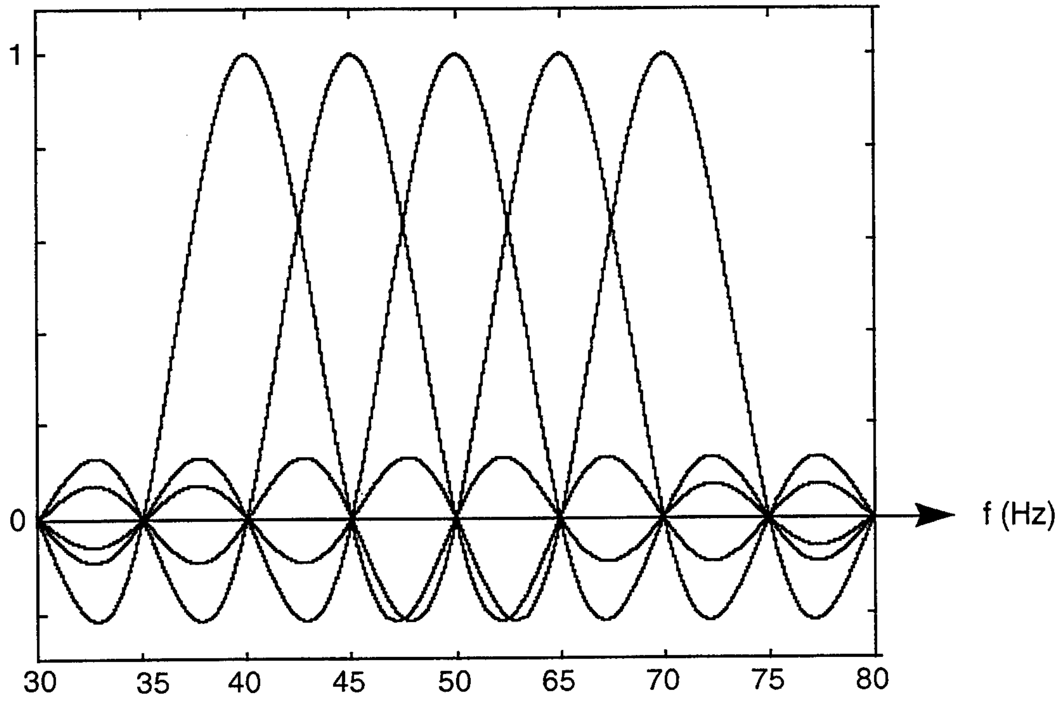


Fig. 9. Complete OFDM Spectrum For Five Symbols Showing Overlap

As previously mentioned, generation of this orthogonal structure is accomplished by using the IFFT, and assuming a distortionless channel, orthogonality is maintained after transmission with each individual subchannel completely separable by the FFT process in the receiver. Unfortunately, in practice, ideal distortionless channel conditions cannot be guaranteed and are typically nonexistent in actual RF transmission environments. Also, since each OFDM symbol spectrum is not band limited, channel distortions such as multipath cause each subchannel to spread energy into the adjacent subchannels causing intercarrier interference (ICI).

C. SYMBOL GUARD INTERVAL INSERTION

One method to overcome ISI and ICI for linear time-invariant channels is to append a guard interval precursor to the symbol interval itself between consecutive symbols prior to transmission [2]. This guard space is a periodic extension of the signal and contains no useful information. The total symbol duration then becomes $T_{total} = T_g +$

T_s, where T_g is the guard interval length and T_s is the useful information bearing symbol duration (Fig. 10). The addition of the guard interval is considered overhead and reduces the overall data throughput, therefore T_g is kept as short as possible to preserve high transmission rates, but long enough to be an effective channel compensator. The guard interval length is dependent upon the channel impulse response and the multipath delay spread. It compensates for the channel's memory [10; p. 42].

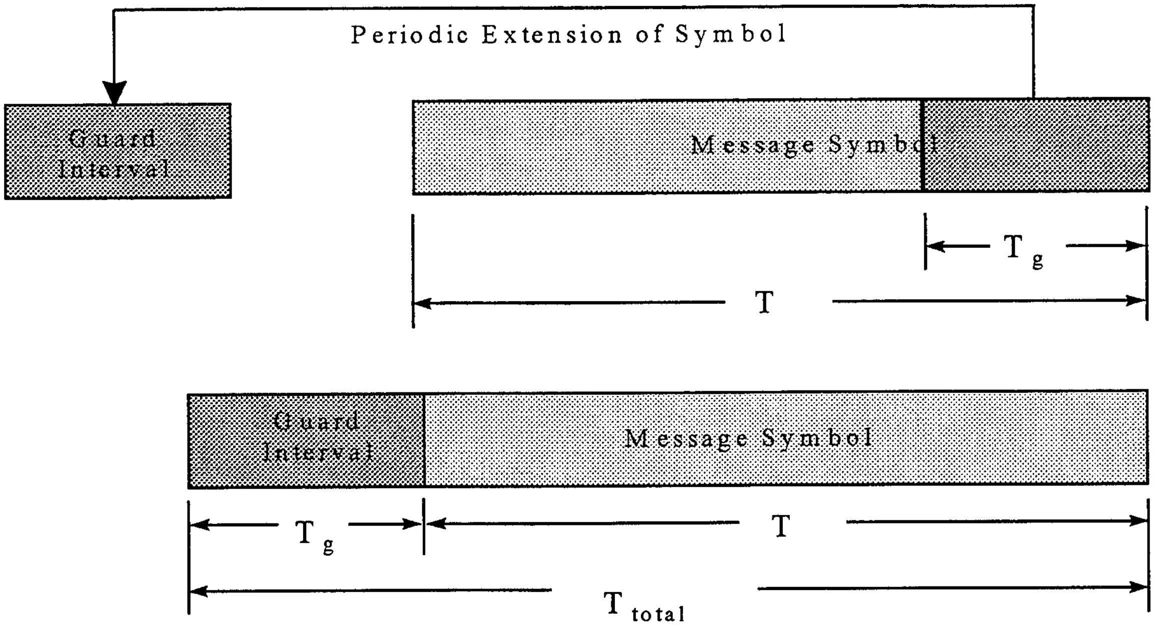


Fig. 10. Creation of the Guard Interval

During the guard interval duration, as the symbol is initially being transmitted, there occurs a "buildup" period within the channel as the impulse response reaches steady-state after the symbol's initial transmission excitation. Following time T_g, and during the T_s period, the channel is in "steady-state". Following the T_{total} period, there is a residual "decay" of the channel response after transmission of the information symbol. The "decay" period of the current transmitted symbol coincides with the "buildup" period of the next transmitted symbol; thus, an overlap occurs between the decaying impulse response of the previously transmitted symbol and the rise time of the next transmitted symbol during time, T_g. This overlap allows minimization of guard interval times as depicted in Fig. 11. The preconditioning of the channel by the inclusion of periodically extended guard intervals allows for the channel to adequately prepare for each transmitted symbol's characteristic waveform without causing ISI or ICI.

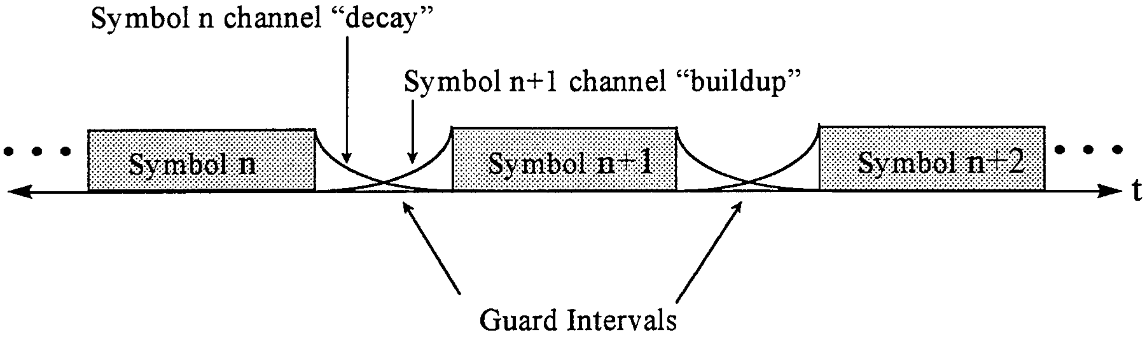


Fig. 11. Guard Interval Insertion Into Symbol Stream

The total information symbol duration, T_s, determines the subcarrier spacing, $f_s = 1/T_s$. However, the symbol rate is

$$r_S = \frac{1}{T_{Total}} \le f_s . \tag{24}$$

While keeping a fixed signal constellation and maintaining the data throughput, a longer useful symbol duration increases the number of OFDM subcarriers and the number of points in the FFT operation. However, carrier offset and phase stability may affect how close two subcarriers can be placed. In addition, in the case of mobile reception where transmitter and/or receiver are in motion, subcarrier spacing must be large enough to account for Doppler frequency shifts. Since the number of OFDM subcarriers correspond to the number of complex points being processed during the FFT operation, consideration must also be given to processing time delays incurred during FFT calculations. [2]

D. FORWARD ERROR CORRECTION COMBINED WITH OFDM

OFDM represents an efficient method to transmit information in parallel in a frequency-selective channel. However, parallel transmission does not suppress the fading directly, since individual subcarriers (OFDM tones) within the channel can be affected by fading depending on their frequency. Instead, frequency diversity coupled with channel coding combine to protect transmitted data. With the addition of overhead non-information bearing bit redundancy to the data (parity bits), COFDM symbols are created for transmission, permitting possible error detection and correction in the receiver. [2]

Various familiar coding techniques provide practical means of error correction. One subclass of nonbinary BCH block codes known as Reed-Solomon (R-S) codes are considered in this thesis. In particular, R-S codes are effective in burst-error environments. For a specified block size with n code symbols being sent for each k information symbols, the R-S code is capable of correcting t arbitrary symbol errors with

$$t = \frac{n-k}{2}. \qquad (25)$$

Thus, no more than 2t parity check symbols are required as error correction overhead [11]. The appended parity symbols offer redundancy that do not carry any useful information, hence, the number of parity symbols increases the overhead and affects the system transmission information rate for a given fixed bandwidth. Consequently, there is a practical limit to the number of additional parity symbols that can be added to the message word k.

The ratio of information symbols to code symbols is called the code rate, R, and can be interpreted as the fraction of the code word that actually carries information [12; pp. 4-5], [4; pp. 416-421]:

$$R = \frac{k}{n} \tag{26}$$

Thus, there is a trade-off between error performance and bandwidth, or information rate, in terms of higher vice lower code rates. The performance improvement gained by adding coding is often measured in terms of coding gain. Coding gain in terms of symbols is defined as the reduction of required $^{Es}/_{No}$ in dB to achieve a specified error performance of an FEC system as compared to an uncoded system with an identical modulation scheme.

The combination of R-S coding with the frequency diversity property of OFDM to obtain COFDM represents a suitable method for providing effective data transmission over a frequency selective channel. However, additional compensation techniques must be employed to further augment reliable communication and combat the presence of symbol errors caused by multipath channel impulsive burst noise and flat fading.

E. SYMBOL INTERLEAVING

One technique using time diversity is symbol interleaving. In a multipath environment the channel is characterized as having memory; thus, errors cannot be considered as randomly distributed error events whose occurrences are independent from symbol to symbol. Consequently, the error events among adjacent symbols as a result of random signal fluctuations or pulsed noise are highly correlated and tend to occur in concentrated bursts. Interleaving the coded message before transmission and deinterleaving after reception allows error bursts to be spread out in time over the entire message block, effectively decorrelating short error bursts. In this way the error events appear more randomly distributed upon reception at the decoder which allows for maximum error correcting performance [13; pp. 357-362].

In most practical situations, the channel memory decreases with increased time separation. With the interleaving method, the code word symbols are separated in time, with intervening times filled with symbols of other codewords. Thus, by separating the symbols in time, a channel exhibiting memory can effectively be changed to a memoryless one, allowing for random-error-correcting-codes to be more effective in correcting the errors. The interleaving process shuffles coded symbols over a span of many symbol lengths known as the interleaver block length; consequently, longer block lengths promote more effective interleaving. The actual interleaving block length required depends on the stochastic burst duration characteristic of the particular channel. However, regardless of block size requirements, complete knowledge of the interleaving pattern algorithm performed inside the transmitter must also be known by the receiver so that proper reordering of the symbols and correct message reconstruction can be accomplished.

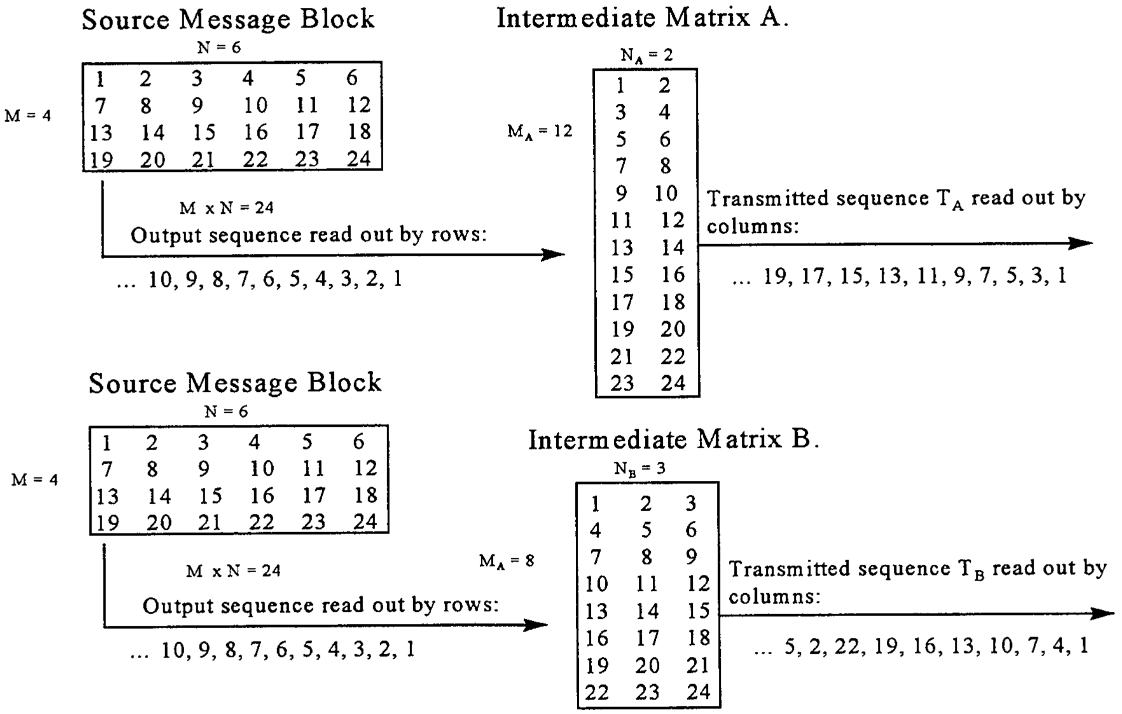
1. Block Interleaver

Different interleaving methods exist, the most basic being the conventional block interleaver. [13; pp. 357-364] A block interleaver requires temporary formation of an intermediate rectangular array whose product of matrix dimensions (matrix row number times the matrix column number) equals the product of the initial source message matrix dimensions. The intermediate matrix is completely filled by the symbols taken from the message matrix, which are read in by rows. Afterwards the symbols are read out of the intermediate matrix by columns, producing the interleaving effect. It is intuitively apparent and will be later demonstrated by example that the degree of symbol interleaving and spacing depends on intermediate matrix dimensions. However, there is a practical limit to the intermediate matrix dimensions.

Aside from added system complexity, including the block interleaving operation introduces transmission and decoding latencies. The dimensions of the intermediate matrix determine the total symbol count in the message block; thus, larger arrays contain more symbols. Considering that the block interleaver and deinterleaver cannot begin their respective interleaving processes until all symbols are available, there is a transmission delay period or latency as the intermediate matrix is filled. If more symbols are required to fill larger intermediate matrices, longer delays occur. Excessive latency is undesirable for a full duplex communication system such as the proposed modem since prolonged transmission delays can adversely affect communications at the application level. Obviously a compromise must be reached, allowing for the benefit of interleaving and burst error decorrelation at the minimized expense of slightly increased latency and negligible application performance impact.

Fig. 12 presents a block interleaver example demonstrating the effects of interleaving a message prone to burst errors prior to transmission through the channel. In this example, the symbol coded source message block is structured as a M by N matrix, S, with M = 4 rows and N = 6 columns and the dimension product of S equal to M x N = 24. As part of the interleaving algorithm an intermediate matrix must be temporarily constructed using the symbols taken from S. Therefore, the dimension product of the intermediate matrix, L, (# of columns times # of rows) also equals M x N. Given the value of S for this example, all possible row and column intermediate matrix dimension pairs are: (1,24), (2,12), (3,8), (4,6), (6,4), (8,3), (12,2), and (24,1). During the formation and subsequent filling of the intermediate arrays having each of these dimensions, the symbols provided by matrix S are read out row by row and into L row by row until S is empty. After matrix L becomes full, the individual symbols within are read out column by column, representing the transmission sequence. From this discussion, it is evident that effective decorrelation of adjacent errored symbols within the transmitted message sequence depends on selective formation of intermediate matrices using appropriate array dimensions. Varied matrix dimensions tends to space the errors differently throughout the message block after deinterleaving is performed.

Fig. 12 supports this example pictorially. It is instructive to note that formation of intermediate arrays with dimensions (1,24) (row vector) and (24,1) (column vector) are not generally implemented since no effective interleaving occurs. For instructional purposes, this example uses intermediate matrix interleaver dimension pairs: (2,12), (3,8) and (4,6) only. From Fig. 12, the dimensions of intermediate matrix A are (12,2), having 12 rows and 2 columns. After being filled completely with the symbols taken from the source message block read in row by row, the transmitted sequence, T_A, is read out of matrix A column by column. During transmission through the channel, hypothetical burst noise occurs corrupting a group of three adjacent symbols in the sequence. Upon reception, the receiver deinterleaves the sequence to reconstruct the original source message. It is apparent from the figure that the burst errors become decorrelated from the group after deinterleaving, becoming isolated non-adjacent symbol errors spaced every other symbol apart.



Source Message Block

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Output sequence read out by rows:

... 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Intermediate Matrix A.

1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16
17	18
19	20
21	22
23	24

Transmitted sequence T_A read out by columns:

... 19, 17, 15, 13, 11, 9, 7, 5, 3, 1

Source Message Block

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Output sequence read out by rows:

... 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Intermediate Matrix B.

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24

Transmitted sequence T_B read out by columns:

... 5, 2, 22, 19, 16, 13, 10, 7, 4, 1

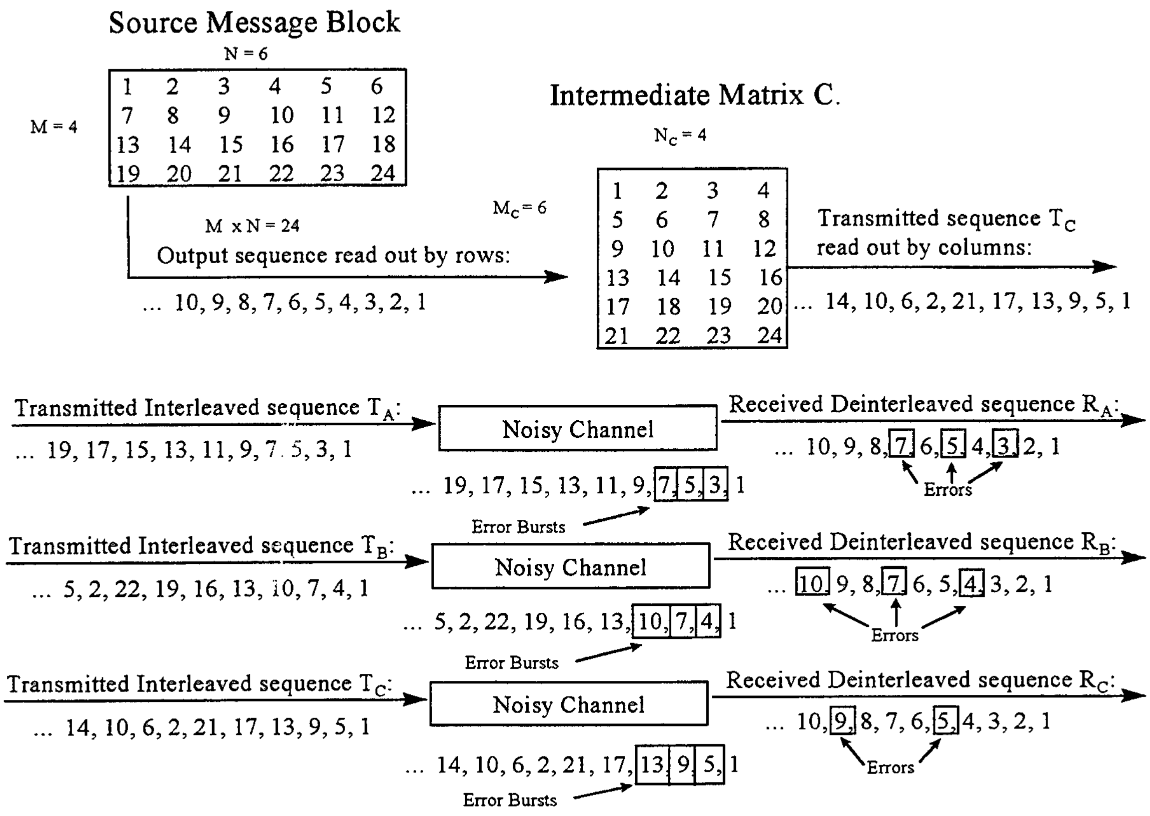


Fig. 12. Conventional Block Interleaver Example

In a similar example, using intermediate matrix B with 8 rows and 3 columns, the identical channel burst error event once again affects a group of three symbols in the transmitted sequence, T_B. Following deinterleaving in the receiver, the group of contiguous errors become decorrelated forming isolated symbol errors in the received sequence, R_B, spaced every two symbols apart. Similarly, for the last intermediate matrix C example, following transmission of the interleaved sequence, T_C, through the channel and deinterleaving in the receiver, the group of errored symbols afflicted by burst noise in the channel, become singly occurring error events spread out in the received message sequence, R_C, and are spaced every third symbol apart. If this example continued for every possible interleaver intermediate matrix dimension, it becomes apparent that the spacing of isolated errors appearing in deinterleaved message sequences are directly related to the intermediate matrix dimensions.

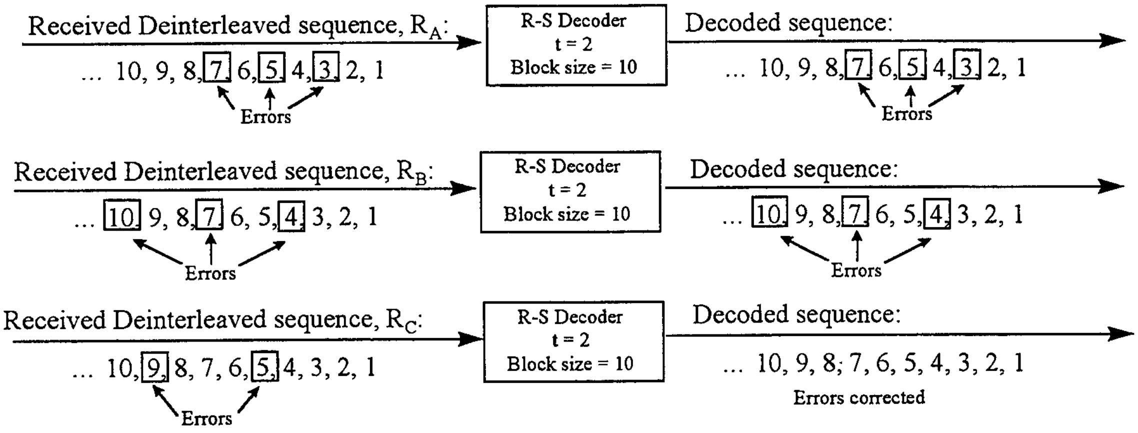


Fig. 13. Block Interleaver With R-S Decoding

2. Interleaving And R-S Error Correction

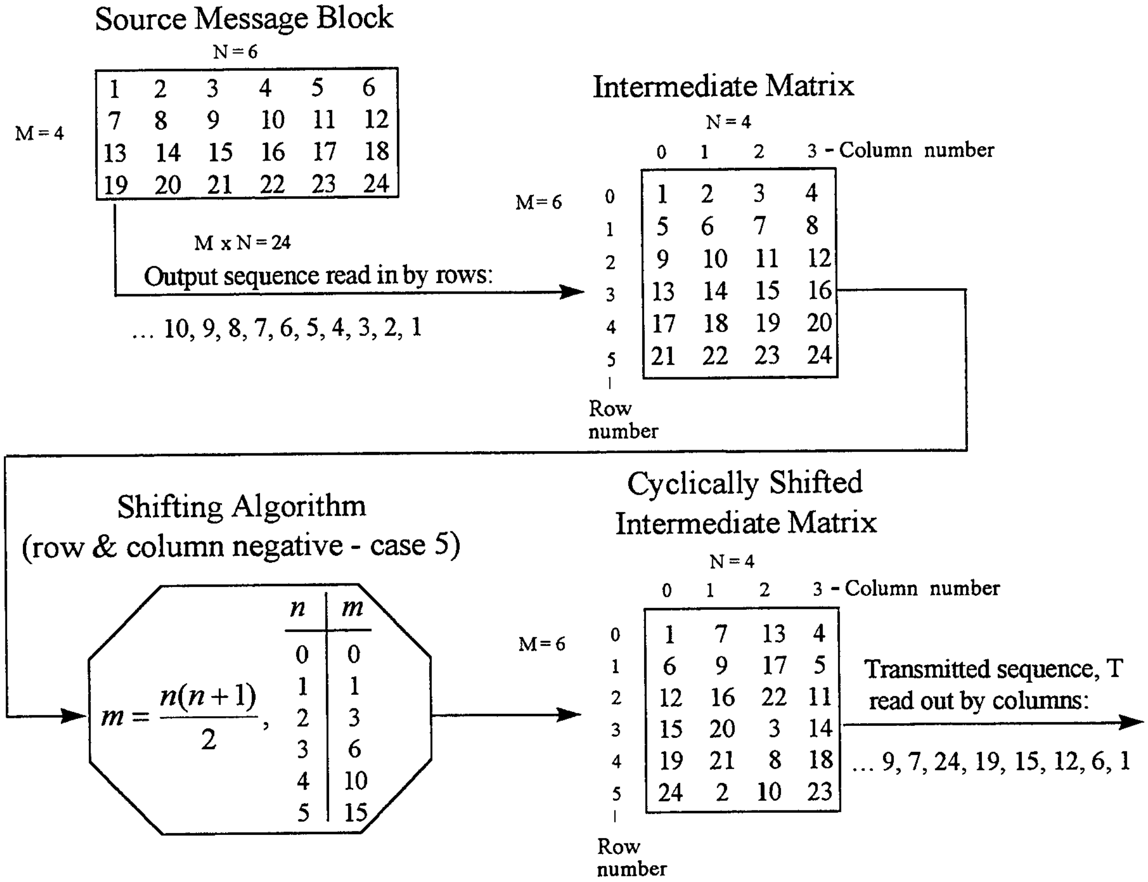
The spacing and locations of received symbol errors are important considerations during implementation of efficient block error correction codes such as R-S coding. As previously mentioned, the R-S code is effective in correcting t errors within a message block. Therefore, the ability of the R-S code to all correct errors depends on the error concentration within the block and the strength of the code. Referring to Fig. 13 as a continuation of the example presented in Fig. 12, the R-S code is not effective in correcting errors in received sequences R_A and R_B since the code is limited to correcting only two errors in a message block size of ten symbols. Since both the R_A and R_B sequences contain three errors within a ten symbol block, the symbol error count exceeds the error correction ability of the R-S code. Thus, no correction is possible for these sequences and the errors remain, corrupting those portions of the message. However, the configuration and interleaving/deinterleaving processes performed on the received message sequence R_C, successfully redistributes the individual errors across the entire transmitted block permitting only two errors to exist in a coding block length of ten symbols. Consequently, the R-S code is effective in correcting these two errors and recover all the lost information within this ten symbol block. Thus, the guarantee of reliable symbol transmission not only depends on the R-S coding strength and block length, but also on the selection of suitable interleaving configuration parameters for a given noisy channel. During the simulation research conducted for this thesis, the R-S coding and interleaving parameters are frequently adjusted and corresponding results recorded to identify optimal combinations for communication with minimal symbol error rates and maximized performance.

3. Cyclical Shifting Block Interleaver

In addition to the ordinary block interleaver, a cyclical shifting block interleaver disperses burst errors by redistributing symbols within an intermediate matrix according to a predefined shifting algorithm prior to transmission. One such type of interleaver previously incorporated in the Common Data Link (CDL) Simulation is referred to as the CDL interleaver and is based upon a Unisys Corporation proprietary design [6], [14]. The CDL interleaver also relies on the formation of an arbitrary sized intermediate matrix similar to the block interleaver. However, as an additional operation within the intermediate matrix, symbols along rows and/or columns are shifted cyclically according to the formula:

$$m = \frac{n(n+1)}{2}, \qquad (27)$$

where n is the corresponding matrix row or column number and m is the number of positions the symbols are shifted, either row and/or column positive and/or negative. It is also possible to define other shifting algorithms besides the one defined in (27), perhaps offering better statistical channel performance under certain recognizable error pattern behavior. Regardless of the type of algorithm implemented, apriori knowledge of the algorithm implemented must also be known to the receiver so that proper reordering of symbols can be accomplished by the decoder.



Source Message Block (M=4, N=6)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Intermediate Matrix (M=6, N=4)

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16
4	17	18	19	20
5	21	22	23	24

n	m
0	0
1	1
2	3
3	6
4	10
5	15

Cyclically Shifted Intermediate Matrix (M=6, N=4)

	0	1	2	3
0	1	7	13	4
1	6	9	17	5
2	12	16	22	11
3	15	20	3	14
4	19	21	8	18
5	24	2	10	23

Fig. 14. Cyclical Shifting Block Interleaver Example

As an example of a cyclical shifting interleaver, Fig. 14 duplicates the familiar source message matrix and intermediate matrix, C, previously presented in the Fig. 12 examples. In this example, cyclical symbol shifting is performed as an additional operation acting on the symbols within the intermediate matrix according to (27). For this particular interleaving case (case 5), shifting is first performed on row symbols followed by column symbols in the negative directions (to the left for rows, toward the top for columns). Other interleaving cases are also permitted (total of eight cases) allowing for multiple combinations of row and column shifting directions and are defined in Chapter V. Following cyclical shifting, the symbols are once again read out of the intermediate matrix by columns forming transmitted sequence.

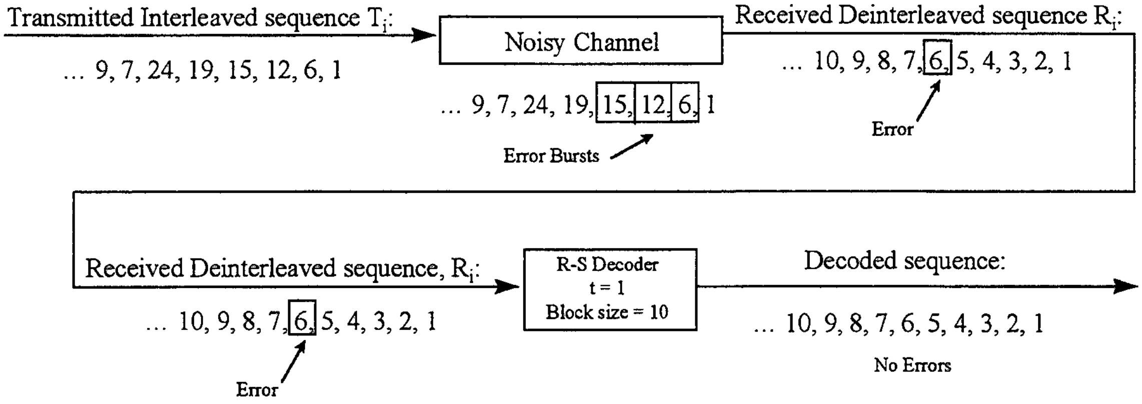


Fig. 15. Cyclical Shifting Block Interleaver With R-S Decoding

The output sequence is transmitted through the multipath channel and encounters burst noise, corrupting portions of the signal. The received signal plus noise is decoded and the corresponding symbols are recreated, one of which is erroneous. The symbol sequence next enters the R-S decoder, attempting to correct single symbol errors within a code block length of 10 symbols. From Fig. 15, it is apparent that the error present in the deinterleaved sequence code block is successfully corrected by the R-S decoder. Hence in this particular example, the cyclical shifting block interleaver sufficiently redistributed errored symbols for effective R-S error correction while using a reduced code strength.

In the previous block interleaver example involving intermediate matrix C, a stronger R-S code capable of correcting two symbol errors is required to generate an error free sink message sequence. However, in contrast to the cyclical shifting interleaver example also using intermediate matrix C, a reduced strength R-S code capable of correcting a single error is all that is necessary to ensure a message sequence devoid of errors. In comparison of the two examples, the cyclical shifting interleaver is as effective as the block interleaver in producing an error free message with a reduced code strength, hence, less code overhead is required and a greater information rate is possible.

Not all cyclical shifting interleaving cases are as effective, as there is dependence upon channel characteristics as well as intermediate matrix dimensions for optimal performance. During this research, multiple simulations are performed using various interleaving configurations in an attempt to determine the most effective cases.

IV. MATLAB COFDM SYSTEM MODEL

The next step in the research was the development of an COFDM computer system model upon which simulations are based. For purposes of this thesis, all signal processing and channel transmissions through the simulated links are performed at baseband. In consideration of this thesis representing a computer system emulation and simulation not requiring physical implementation, the functions normally associated with RF up-conversion and down-conversion are not necessary to generate meaningful tradeoff results. Thus, filtering, digital-to-analog conversion (DAC), up/down frequency translation and analog-to-digital conversion (ADC) functional sub-blocks necessary for actual implementation are not included in the computer model.

A. COFDM TRANSMITTER

The COFDM transmitter functional block diagram is illustrated in Fig. 16 with each of the sub-blocks subsequently described.

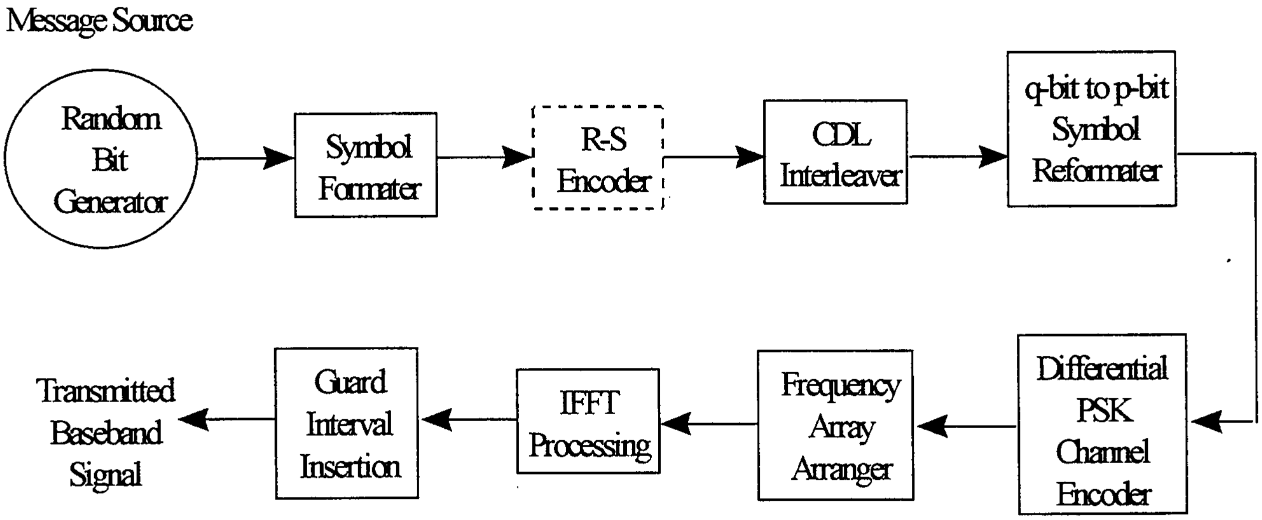


Fig. 16. COFDM Transmitter Functional Block Diagram

Random Bit Generator: This functional block originates a random message bit pattern representing the information source. The bit sequence length is variable as defined by the user. The random property of each binary element is determined by a seed parameter setting the internal computer's random number generator seed. If multiple simulations are performed using the same seed values, identical results occur. This property is useful when comparing and contrasting simulation outputs with different system configurations. By fixing seed values, optimal system configurations can be ascertained based upon superior SER performance while using consistent channel characteristics and source message symbol patterns. It is also possible to set the seed randomly by the internal PC processor.

Symbol Formatter: The random binary message stream enters the symbol formatter where bits are collected and grouped together to form q-bit long words and are referred to as OFDM symbols. For most of the simulations conducted, an initial message symbol word length of eight bits is used. In consideration of eventual COFDM system implementation, selecting eight bit symbol words allows compatibility with existing commercial off-the-shelf (COTS) R-S error correction hardware which is typically physically designed to operate on 8-bit words. Furthermore, using 8-bit symbols instead of 4-bit symbols for example, permit longer R-S code word formations and greater number of possible symbol corrections within a code word. Thus, this step is included in the signal path solely with future implementation and R-S coding in mind. At a later stage in the transmitter after R-S encoding, the symbols are resized using shorter length words to accommodate the preferred constellation mapping scheme and are referred to as PSK symbols.

The formatted symbol sequence is reshaped into a matrix forming the source message block. The number of matrix columns corresponds to the desired number of COFDM subcarriers generated for transmission and must be even. The number of matrix rows is entirely arbitrary depending on the specified message block length. The total number of symbols randomly generated for simulations equals the source message block column number times the row number.

Error Correction Encoder: Symbol error correction is incorporated into the COFDM system model using a Reed-Solomon (R-S) correction method, allowing for t or fewer errors to be corrected as governed by (25). The information word size, n, the code word size, k, and the code block length are user defined to select the coding strength and coding gain as required, with consideration given to overall code rates. While provisions for the error detection and correction functionality is included in the COFDM system model, symbol encoding and parity bit generation is not actually performed in the transmitter and exists virtually; hence, the dashed outline representing the R-S encoder block in Fig. 16. Instead, the error detection and correction functionality is emulated in the receiver without the additional step of individual R-S symbol decoding.

Absence of the R-S encoder is justified under the assumption that effective R-S encoders have previously been demonstrated and verified. Therefore, it is not necessary to emulate the actual encoding process since R-S encoding verification is not the focus of this research, and this additional operation slows down simulation run times. For COFDM simulation purposes, it is only necessary to emulate an error correction mechanism to correct received symbol errors at some stage in the communication link. Consequently, the R-S error correction functionality is actually emulated in the receiver and is based upon comparisons of the source message to the received message. Further discussion regarding this will be included in the R-S decoding section of the OFDM receiver.

CDL Interleaving: As a time diversity mechanism, a symbol interleaver is included and operates on the source message symbol array to redistribute the symbol locations. As mentioned in the previous Chapter this process aids in randomizing burst errors in a multipath channel. The extent of symbol interleaving is primarily determined by the user specified dimensions of the intermediate matrix as given by the row number and column number. However, all interleaver row and column parameters must correspond to the original message array dimensions as previously discussed.

In addition to conventional block interleaving, more sophisticated cyclical shift interleaving is also included to redistribute symbols according to a predefined shifting algorithm. The shifting algorithm is a function of the intermediate matrix row number and/or column number and is based upon a Unisys Corporation proprietary design [14]. A total of eight separate configurations (interleaving cases) are incorporated to cyclically shift symbols in a selective pattern within the two dimensional intermediate matrix.

Symbol Reformatter: In preparation for the appropriate N-ary modulation scheme, N-PSK, the incoming q-bit OFDM symbols are resized into p-bit PSK symbols, where $N = 2^P$ (Note: the N used for N-ary signaling is not the same N used for N-point FFT calculations). Since 16-PSK and 4-PSK (QPSK) are predominately used during simulation runs, accordingly symbol lengths are resized as either 4-bit ($p = 4$) or 2-bit ($p = 2$) length words. If necessary, zero bit padding may be required during the reformatting process to account for incomplete word formations.

As a result of symbol reformatting, the dimensions of the original source message array may change to compensate for the addition or deletion of redefined symbols. Regardless of the number of new PSK symbols formed, the number of matrix columns corresponding to OFDM subcarriers remain fixed. Hence, any necessary message symbol quantity adjustment is accommodated by increasing or decreasing the number of matrix symbol rows instead. For example, if during the symbol reformatting process the OFDM symbols are changed from 8-bits to 4-bits, then the total number of message symbols double from their original amount. Consequently after reformatting, the number of message matrix rows double while the number of message matrix columns remain constant.

Channel Encoding: PSK is the preferred modulation technique for channel encoding in multipath channels. Prior to signal constellation mapping, differential encoding is performed on the symbols within the message matrix. Two types of differential encoding are included. Considering differential encoding along the time dimension (symbol rows), a cumulative summation down each column of the message symbol array is calculated. For differential encoding along the frequency dimension (OFDM frequencies), a cumulative summation across each row of the message symbol array is calculated. Recall that construction of the message block matrix is designed so that columns represent OFDM frequencies (frequency dimension), while rows represent symbols generated in time (time dimension). During subsequent simulation trials, either frequency and/or time differential encoding may be selected to evaluate system performance.

The differential encoding/decoding technique introduces memory into the system and allows for decoding of the current received symbol with respect to the previously decoded symbol. Consequently, detection decisions are based upon relative differences between consecutively received symbols. This technique may be advantageous in a slowly fading multipath channel where the variations among successive received symbols is negligible. A cumulative summation can be best illustrated through an example.

$$\text{Given:} \quad \underline{V} = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]' \tag{28}$$

$$\text{then,} \quad \underline{\text{CsumV}_{16}} = [1\ 3\ 6\ 10\ 15\ 5\ 12\ 4\ 13]'. \tag{29}$$

$\underline{V}$ is a column vector whose elements represent message symbols taken from the set of N integers, where $N = 2^P$. $\underline{\text{CsumV}}$ is formed by consecutively adding in modulo-N fashion successive elements in $\underline{V}$ beginning with one to the current running total in $\underline{\text{CsumV}}$ beginning with zero. For this example N = 16; thus, 0 + 1 = 1, 1 + 2 (the next element in $\underline{V}$) = 3, 3 + 3 = 6, 6 + 4 = 10, 10 + 5 = 15, 15 + 6 = 21 = 5 (modulo-16) and so on. In this way, all the elements in CsumV are calculated with respect to the first element in V. A more concise expression is:

$$\text{CsumV}_k = \text{V}_k \oplus \text{CsumV}_{k-1}, \qquad (30)$$

where $\{\text{V}_k\}$ is a modulo-N message sequence into the differential encoder, $\{\text{CsumV}_k\}$ is the encoder output sequence, and ⊕ denotes modulo-N addition [13; pp. 108-109].

Following differential encoding, each symbol in the differentially encoded message array is channel encoded as a complex modulation value with unit magnitude and one of N possible phases (N-PSK modulation); that is,

$$D_K = e^{2\pi j \frac{CsumV_K}{N}} . \qquad (31)$$

In continuation of the previous example ending with (29), the corresponding vector of 16-ary complex modulation value phase angles are,

$$\text{Ang}[\underline{\text{D}}] = [22.5^\circ\ 67.5^\circ\ 135^\circ\ 225^\circ\ 337.5^\circ\ 112.5^\circ\ 270^\circ\ 90^\circ\ 292.5\,^\circ]' \qquad (32)$$

A row of ones representing zero phase complex modulation values is appended to the top of the message array during time differential encoding, representing a decoding reference for the receiver. For frequency differential encoding, a pair of columns containing ones elements is appended to the extreme left side of the message array as a similar decoding reference. Two ones columns are included instead of a single column to maintain an even number of OFDM frequencies (even number of columns).

Frequency Array Arranger: This sub-block conditions the previously generated array of complex modulation values for eventual IFFT processing. Due to the manner in which the N-point FFT algorithm is performed by MATLAB, the resulting signal spectrum is graphically displayed symmetrical about the point N/2 instead of the origin, with frequency indexing from 0 to N-1 instead of the usual baseband range of -N/2 to N/2-1, as depicted in Fig. 18. Since MATLAB does not allowing negative indexing, the left half negative portion of a correct spectrum shown in Fig. 17 is shifted positive N positions as a result of a FFT. To account for the swapping of the left and right half portions of the spectrum, an index correction shift must subsequently be performed to relocated the upper half samples above N/2 back to their normal positions, left of the origin at the receiver. Recall that the IFFT/FFT are periodic functions with period 2π; thus, the spectrum infinitely repeats itself as shown in Fig. 19. Consequently, after bandpass filtering the spectrum with cutoffs N/2 and 3N/2, a complete symmetrical spectrum can be obtained that matches a baseband one suitable for transmission.

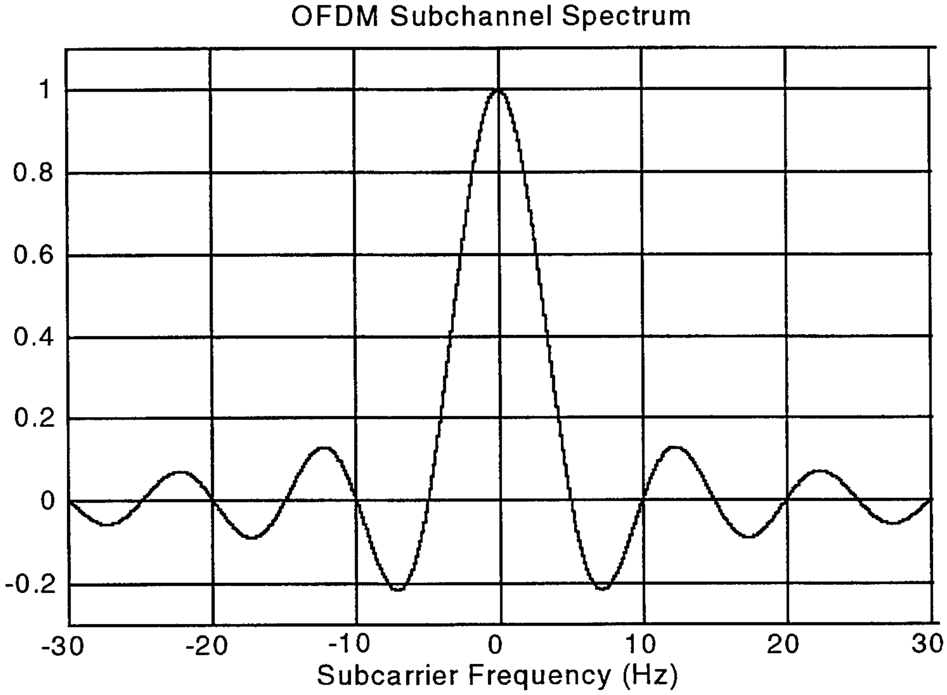


Fig. 17. Correct Fourier Transform Frequency Spectrum

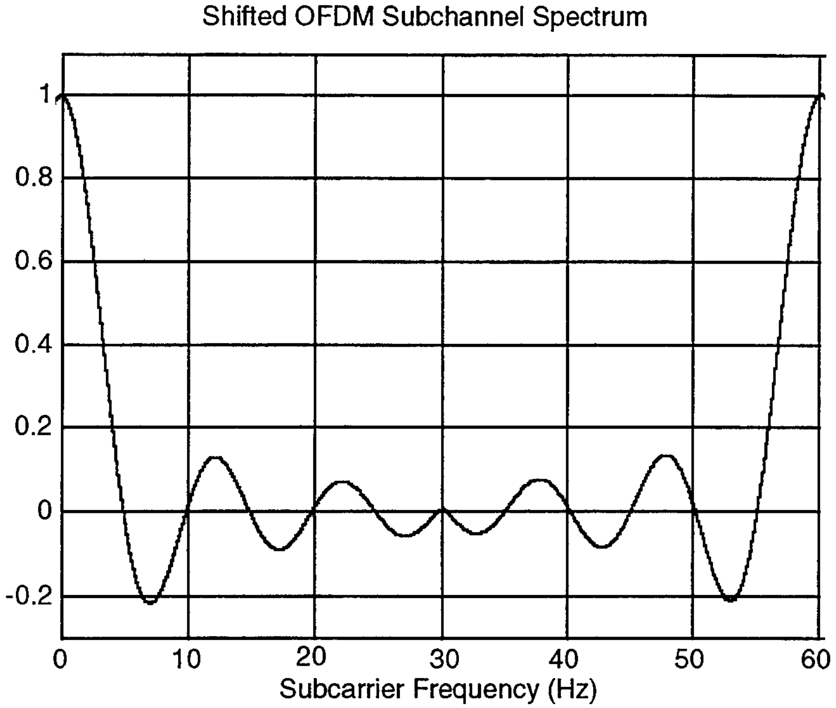


Fig. 18. Shifted Frequency Spectrum Resulting From MATLAB FFT

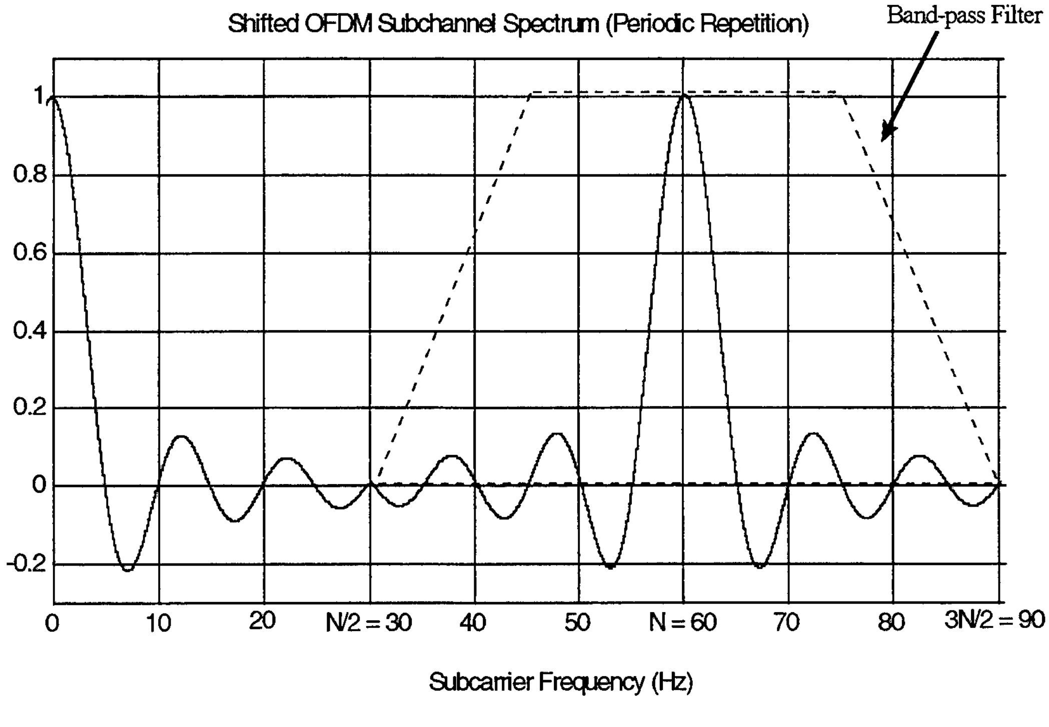


Fig. 19. Complete Spectrum Recovery Using Bandpass Filter

With this in mind, the frequency array arranger block accepts the input vector containing K elements of complex modulation values and pre-shifts the first half to the right and takes the second half and shifts them to the left. To account for filter roll-off slopes, a padding of P zeros are inserted between the upper and lower halves. Thus, the total sample length becomes N, where N = K + P.

IFFT Processing: To convert the frequency array to time domain representation, an N-point IFFT is performed producing a corresponding output sequence of time domain samples. The input array of complex modulation values have the left and right half swapped by the previous frequency arranger block to account for the automatic frequency index shift that results from the IFFT.

Guard Interval Insertion: A guard interval composed of a periodic extension of the symbol is inserted at the beginning of each symbol for channel impulse response compensation purposes. The length of the guard interval is variable to account for multipath delays and may be lengthened or shortened as required. The guard interval is represented by additional time domain samples added to the resulting sequence derived from IFFT processing.

B. COFDM RECEIVER

The receiver functional block diagram is illustrated in Fig. 20. The blocks in the receiver perform the reciprocal functions of the transmitter and are described below.

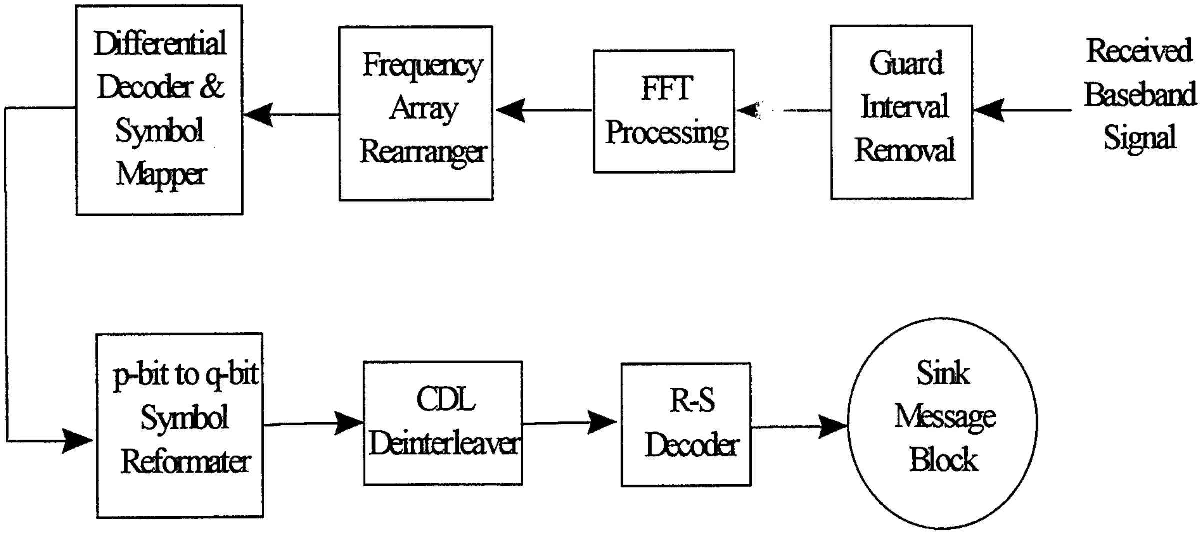


Fig. 20. COFDM Receiver Functional Block Diagram

Guard Interval Removal: The guard interval precursor appended to each symbol in the transmitter is initially removed, leaving behind the remaining information portion of the symbol for further processing. The information symbol consists of a sequence of N time domain samples.

FFT Processing: The sequence of time domain samples are transformed into the frequency domain using an N-point FFT to recover the OFDM frequency tone information. In a linear time-invariant channel, the orthogonality of carriers is preserved; however, in a multipath environment with frequency Doppler shifting, this is not always the case. The output is an array of complex modulation values with the left half portion shifted to the right N positions as a result of the FFT operation (Fig. 18).

Frequency Array Rearranger: To reconstruct the array of complex modulation values properly after FFT processing and maintain correct spectral ordering, the frequency rearranger block swaps the upper half portion of the complex values to the lower half indexed positions and the lower half values get shifted to the upper indexed positions. In effect, the two halves exchange their positions.

Channel Decoding: Differential decoding in first performed either in the frequency dimension (matrix columns) or time dimension (matrix rows), maintaining compatibility with the transmitter differential encoding method. In addition, the previously appended reference ones elements are removed. Afterwards, channel decoding is accomplished, inverse mapping each received complex modulation value with magnitude and phase into a corresponding N-ary symbol representation composed of p bits. Considering 16-PSK, 4 bit long symbols are reconstructed.

Symbol Reformatter: The signal inverse mapper recreates p-bit long symbol words represented in decimal notation. To prepare each symbol for eventual R-S decoding, each p-bit symbol is first converted to it's binary equivalent, then reformatted as eight bit symbols. Once again the message matrix dimensions may change according to the symbol reformatting scheme in an inverse manner from the way discussed in the transmitter. The final output sink message matrix has the same dimensions as the source message matrix in the transmitter.

CDL Deinterleaving: The message array of eight bit symbols is next deinterleaved to reconstruct proper ordering of the information symbol stream according to the particular interleaving case configured in the transmitter. After deinterleaving, any corrupted symbol errors caused by burst noise in the channel should be sufficiently redistributed within the message array, creating a more random, uncorrelated error distribution.

R-S Decoder: The Reed-Solomon decoder detects and corrects t symbol errors where t is given by (25). As the number of parity check symbols in the code word of n symbols increases within a practical limit, more errors are corrected and the coding gain is improved. The R-S decoder performs error detection and correction functionality without actual decoding since no parity symbol encoding is performed in the transmitter for reasons previously discussed. The symbol detection and correction scheme looks for errored symbols within a code block by comparing source and sink messages. Any symbol miscompares appearing within a code block indicate errors, and an attempt at correcting the errors is performed depending upon the configured coding strength and code block length. Error quantities within a code block exceeding the correction ability of the code are left unaltered and result in message corruption.

Received Message: The output of the receiver represents the received sink message block. After transmission through the system channel model and prone to noise and multipath distortions, symbol errors may exist. The distribution of error events within a message array is recorded and the symbol error rates calculated to generate corresponding performance curves. The resulting simulation data is compared to the theoretical performance criteria for evaluation.

C. CHANNEL MODELS

Noisy Channel: Three channel models are emulated as part of the overall communication system model and used during simulations (a noise free channel 0 model is also included for system functional verification) (Fig. 21). One emulated channel type is the AWGN model and represents additive noise only. The second is the multipath channel model and is characterized by power fading (loss) in dB, Doppler frequency shifting in Hz and multipath time delays in microseconds which vary for each transmission link according to the specified multipaths. For example, the ship-to-ship link has two multipaths, one Ricean distributed and one Rayleigh distributed. The ship-to-shore link has three multipaths, one Ricean distributed and two Rayleigh distributed, and finally the ship-to-relay (air) link also has three multipaths, one Ricean distributed and two Rayleigh distributed. The composite channel 3 model is a combination of the first two models;

thus, the AWGN model is added to the multipath model representing the actual maritime communication environment.

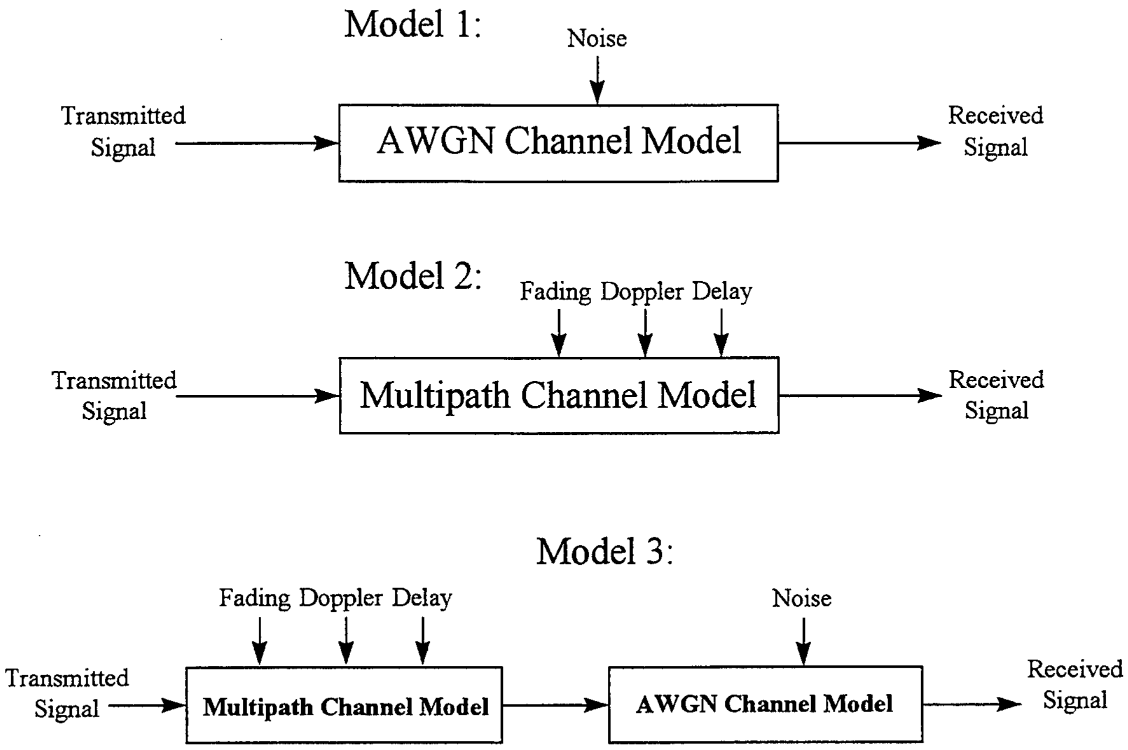


Fig. 21. Three Channel Models

D. SYSTEM DESIGN METHODOLOGY

Initial design of the overall system originates from an understanding of the COFDM concept as well as from an attempt to meet the specified system objectives for bit rate and performance. According to the BAA requirements, the desired modem operational bit rate is 1.536 Mbps. Additionally, the usable 3-dB bandwidth of operation is 480 KHz, with primary consideration for operation in the 225-400 MHz UHF band. From this information, the calculated bits per Hz is found to be

$$b_{Hz} = \frac{1536kbps}{480kHz} = 3.2 \,{}^{bits}\!/_{Hz} \approx 4 \,{}^{bits}\!/_{Hz}. \quad (33)$$

Rounding b_{Hz} up to the next whole number, 4 bits per Hz results; hence, 16-ary signaling is used, since $M = 2^q = 2^4 = 16$. In consideration of relatively fast fading rates in addition to minimal bandwidth usage, PSK is selected as the baseline modulation technique for this emulation; thus, 16-PSK is included. It is instructive to note that by using 16-PSK the calculated bit rate is 1.92 Mbps, a rate in excess of the specified bit rate. However, it will become apparent later that this added throughput diminishes once FEC is included and guard intervals are appended to individual symbols as required non-information bearing overhead. Thus, the useful information bit rate is reduced.

Consideration is next given to the required symbol interval length. It is undesirable to use excessively long symbol interval lengths in a multipath channel so as to avoid power fading within the symbol. Similarly, it is not advantageous to minimize the symbol interval length too much since a guard interval precursor must later be appended to the information symbol for channel compensation purposes. For optimal efficiency, guard interval overhead should be minimized as much as possible while also maintaining effectiveness in negating multipath distortions. In consideration of the BAA's specified worst case multipath time delay of 5.1 μsec (link 3, ship-to-relay link), a guard interval of 10 μsec is considered sufficient compensation. As a general rule of thumb to minimize overhead, the guard interval occupies no more than two percent of the information bearing symbol length. Consequently, the calculated symbol length, T_s, becomes 500 μsec ($T_s = T_g/2\% = 10$ μsec/.02 = 500 μsec), while the total symbol length, T_{total}, is 510 μsec.

Using the calculated symbol length, T_s, and keeping in mind that the spacing of OFDM sub-carriers is the reciprocal of the symbol interval, one may obtain the OFDM tone spacing as follows:

$$f_s = \frac{1}{T_S} = \frac{1}{500\mu\sec} = 2000Hz\,. \tag{34}$$

From f_s, the maximum number of OFDM frequency tones allowed within the available 3 dB, 480kHz bandwidth, BW_{sdB}, is

$$N_{OFDM} = \frac{BW_{3dB}}{f_S} = \frac{480kHz}{2000Hz} = 240\,. \tag{35}$$

Consequently, as shown in Fig. 22, the complete overlapping spectrum of OFDM symbols are mapped to 240 corresponding OFDM sub-carrier tones occupying a 480 kHz bandwidth. In the time domain, the symbol interval appears similar to Fig. 23 with information interval, T_s, occupying 98% of the entire symbol length, T_{total}, equaling 510 μsec. Accounting for 2% guard interval overhead, the adjusted bit rate without R-S coding is 1.8816 Mbps. If R-S FEC is included with a 0.85 code rate, then the overall bit rate further reduces to 1.5936 Mbps, still above the specified objective. It will be demonstrated later during simulation discussions for certain links and system configurations that the actual bit rate decreases further as additional FEC strength is required to accommodate channel induced noise and perform acceptably.

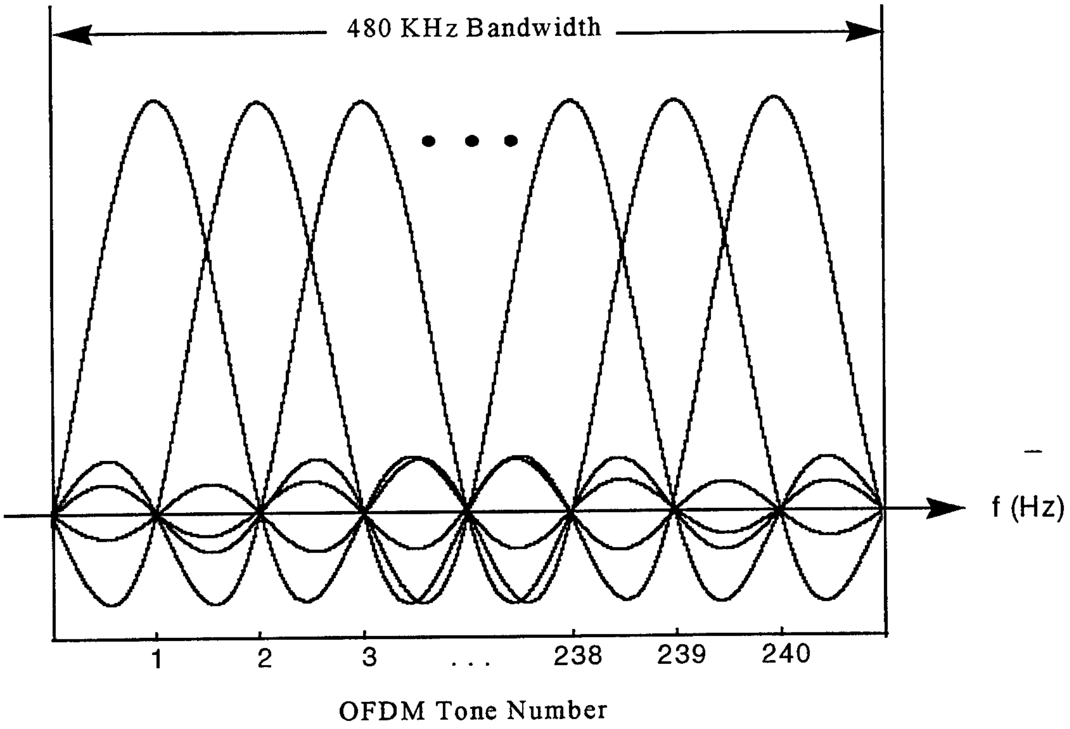
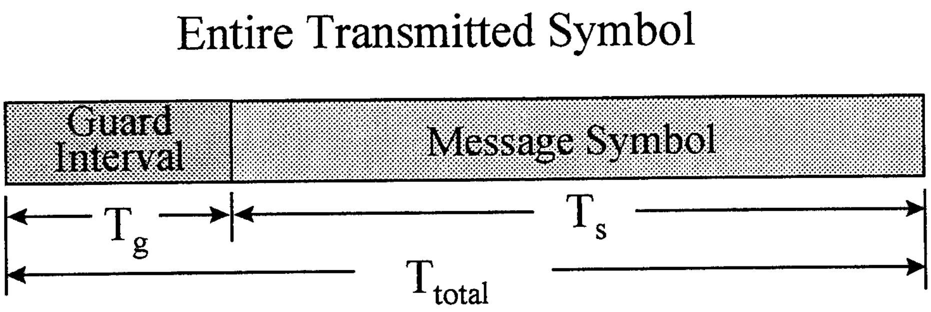


Fig. 22. Emulation Partial OFDM Spectrum



$$T_g = 10\ \mu\text{sec}$$

$$T = 500\ \mu\text{sec}$$

$$T_{total} = 510\ \mu\text{sec}$$

Fig. 23. Single Symbol Interval with Precursor

Nonetheless, having preliminarily determined the modulation scheme, the guard interval length, the proposed number of OFDM frequency tones and the OFDM frequency spacing while considering the bit rate objective and bandwidth constraints, construction of the software COFDM system emulation model can begin. A block diagram of the complete system model which is emulated in MATLAB and simulations performed is presented in Fig. 24.

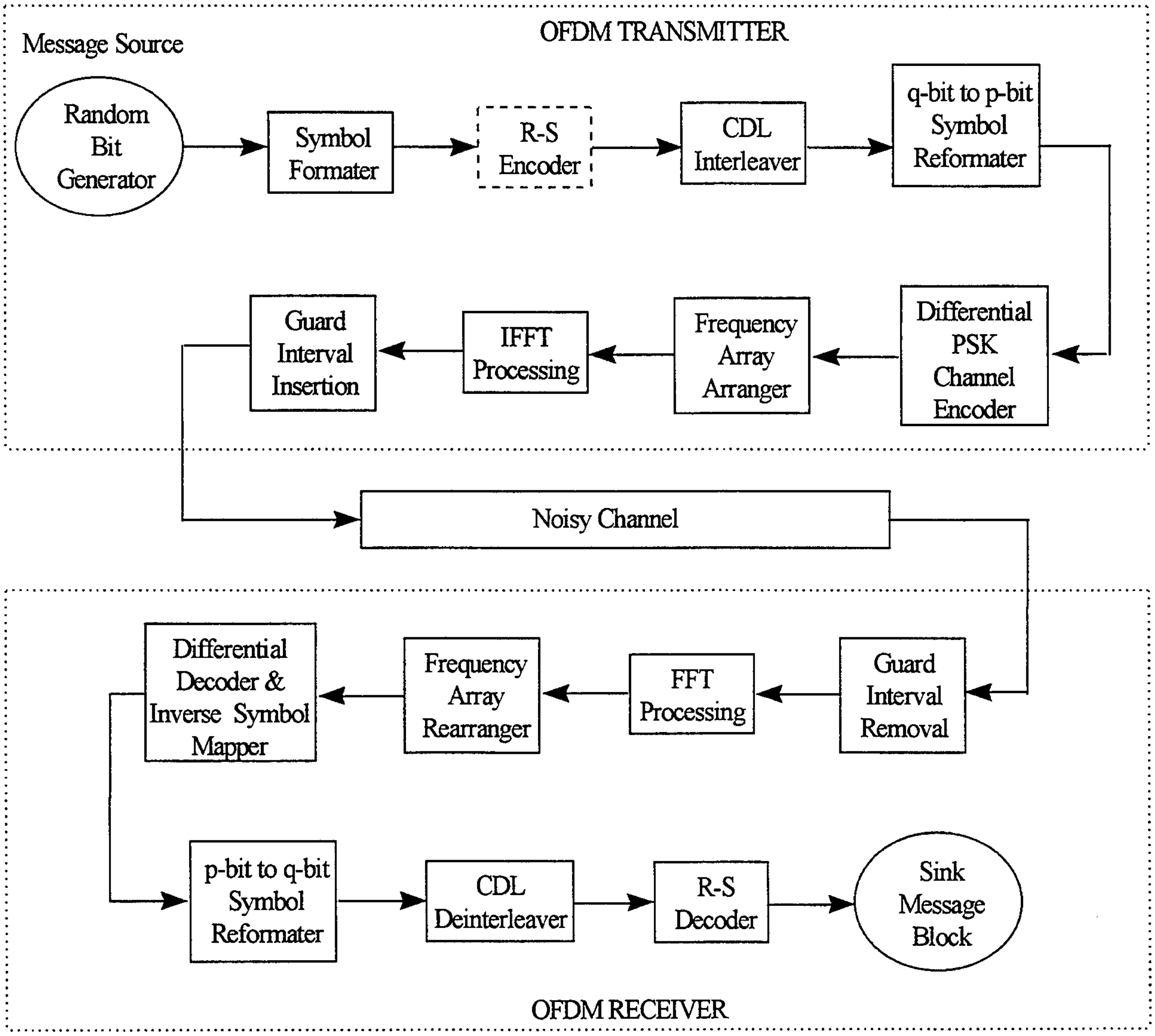


Fig. 24. Complete OFDM System Model

V. MATLAB PROGRAMMING AND MACRO DEVELOPMENT

A. GENERAL MATLAB FUNCTION FORMAT

The MATLAB language provides various standard built-in functions and commands as well as additional higher level tool box functions. It is possible to define custom macro functions with multiple input and output arguments. The user can define and program macros by using a standard ASCII text editor such as Notepad or MS Word. The programmed files are created and named with a **.m** suffix appended to each macro name; appropriately functions are referred to as "m-files". To invoke a macro within the MATLAB command window, a user types the function name and includes the input argument variables in parentheses, and sets the function equal to the output variables in brackets. The variables may be real or complex, scalars, vectors, or matrices. For example, the function **function.m** is an ASCII file in a directory in the MATLAB path and has the form:

[out1,out2,out3] = function(in1,in2,in3), (36)

where out1, out2 and out3 are output variables and in1, in2 and in3, are input arguments. Macro functions may also be equivalently represented as a functional module or block as shown in Fig. 25.

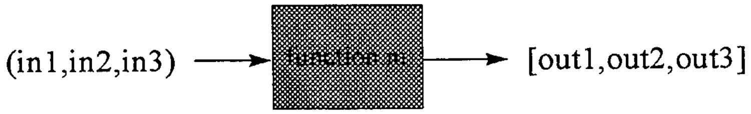


Fig. 25. A MATLAB Functional Block

B. OFDM SYSTEM MODEL CONSTRUCTION OF FUNCTIONAL BLOCKS

With an understanding of the basic structure for creating macros, emulation of the COFDM communication system depicted in Fig. 24 is done by initially partitioning the overall system according to functionality and forming functional interconnecting sub-blocks. The COFDM system model consists of three rudimentary components: a COFDM transmitter, the channel and a COFDM receiver. Within the transmitter are two separate functional blocks, a source encoder block and an IFFT processing block. The channel consists of four separate models: the channel 0 model, the channel 1 model, the channel 2 model and the channel 3 model. Each channel model corresponds to a different type of noise (except for channel 0 model which is noise free). The receiver block consists of two blocks: the FFT processing block and the message decoding block. Recall that all simulations are performed at baseband; therefore, no additional blocks associated with RF bandpass transmissions are required nor included in the model.

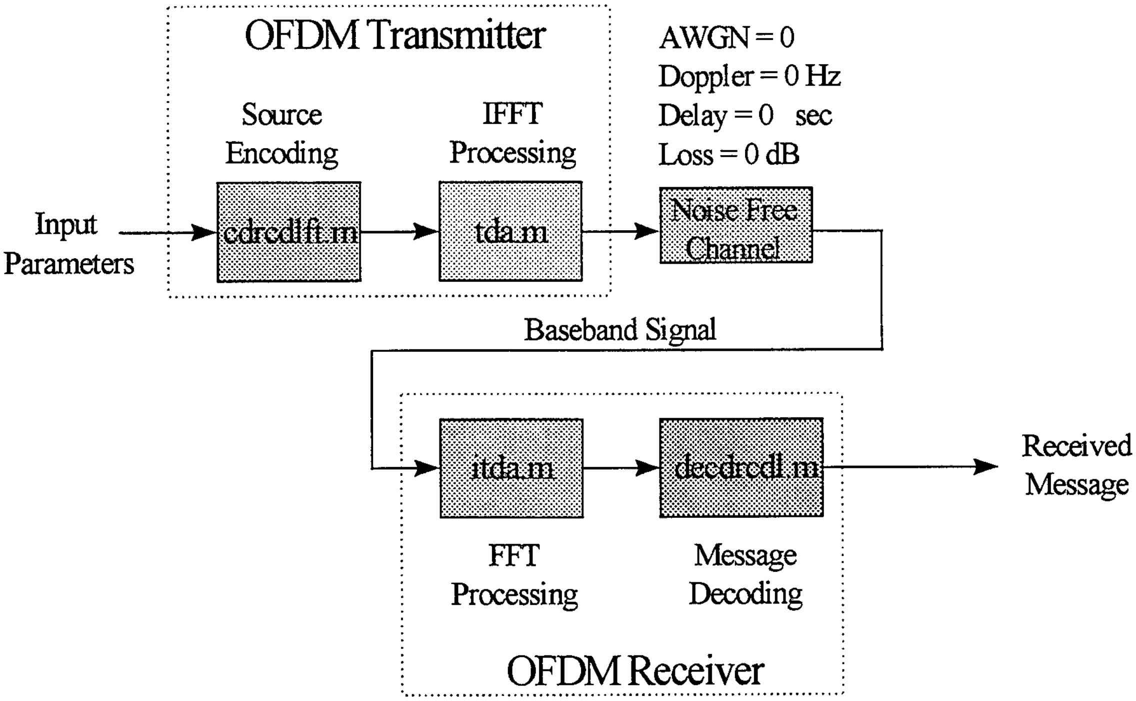


Fig. 26. Model 0 Block Diagram

1. COFDM Model 0 System

The model 0 block diagram is shown in Fig. 26 and represents a noise free perfect channel, i.e., the absence of AWGN and any multipath influences within the channel. Transmitter source encoding is performed within the m-file macro, **cdrcdlft.m**. The functional sub-blocks associated with **cdrcdlft.m** are depicted in Fig. 27. The IFFT processing block responsible for generating OFDM frequency tones and appending guard intervals is represented by the m-file macro, **tda.m**. Correspondingly in the receiver, the inverse functions of the transmitter are performed, namely FFT processing and guard interval removal is accomplished by the **itda.m** m-file, while signal decoding is accomplished by macro **decdrcdl.m**.

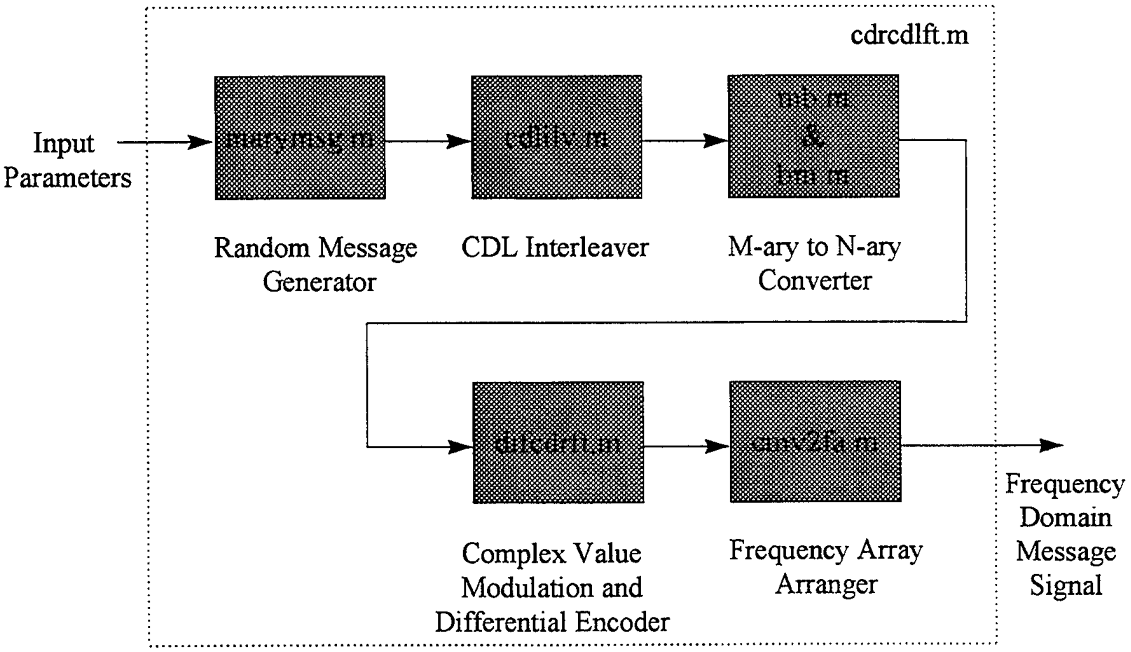


Fig. 27. M-file Cdrcdlft.m Functional Sub-blocks

a. COFDM Transmitter

The hierarchical arrangement of m-files within **cdrcdlft.m,** including subroutine macros, are presented in Fig. 28 and are subsequently described in detail.

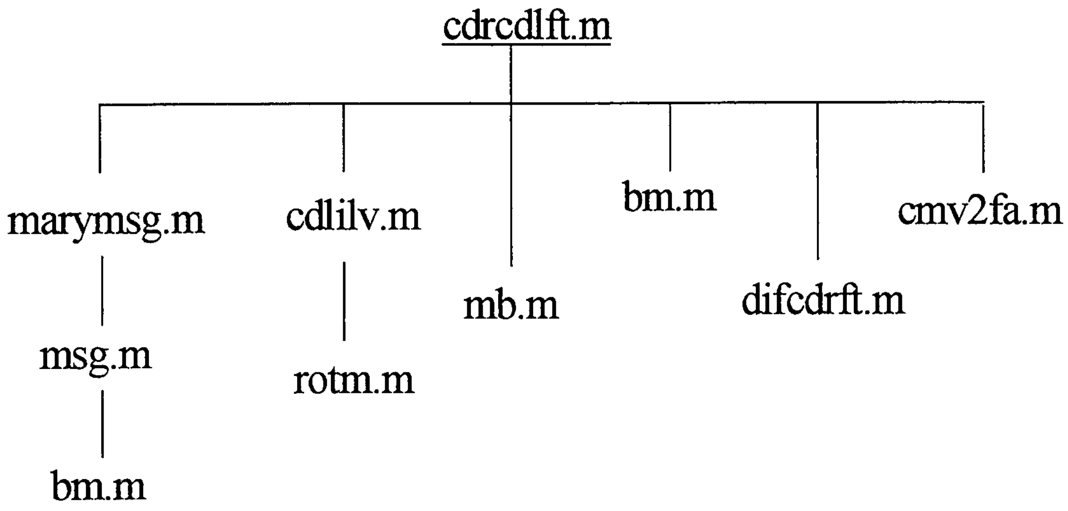
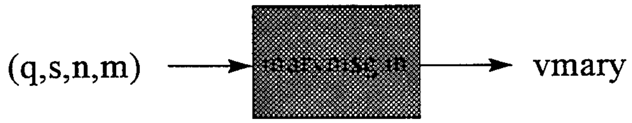


Fig. 28. M-file Hierarchy for Cdrcdlft.m

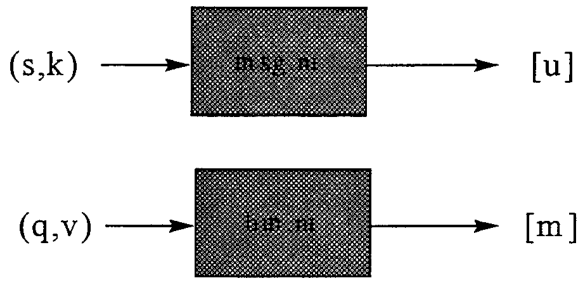
The source message is randomly generated by the m-file **marymsg.m**. The general form of the function is depicted by the functional block shown below. As Table 1 demonstrates, this function generates an array of randomly generated *q*-bit long symbols displayed in decimal notation which represent the random message source, *vmary*. The input arguments, *n* and *m*, determine the overall output message matrix dimensions, where *n* is the number of rows and *m* is the number of columns. The value selected for *m* also represents the number of OFDM frequency tones and must be an even positive integer so as to completely fill the available transmission bandwidth without spectral cutoff of the endpoint symbols. The value selected for *n* is any arbitrary positive integer and represents



rows of symbols generated in time. The input argument, *s*, is the seed parameter used for setting the seed of the internal MATLAB random number generator function. The remaining input argument, *q*, represents the number of bits contained in each of the symbol words. Considering M-ary signaling , M = 2^q. The function **marymsg.m** requires two other subroutine m-files, **msg.m** and **bm.m**.

```
» msg1 = marymsg(3,10,6,6)  % Random 6 row by 6 column message source using 8-ary
                              symbols and a seed of 10.

msg1 =

   6   1   6   6   4   0

   6   6   1   0   1   6

   0   0   1   4   2   6

   1   2   0   6   6   0

   0   2   1   0   2   6

   1   0   4   2   6   3


» msg2 = marymsg(4,20,3,8)  % Random 3 row by 8 column message source using 16-ary
                              symbols and a seed of 20.

msg2 =

   8    9   6    9   5   10   9   7

   2   12   9    2   8    7   8   9

   5   13   7   12   3    1   0   9


» msg3 = marymsg(8,30,5,4)  % Random 5 row by 4 column message source using 256-ary
                              symbols and a seed of 30.

msg3 =

   14   228   148   105

  109   152   123     4

   34   221    32    68

  177   101    84   152

   93   215   184   171
```

Table 1. Marymsg.m Sample Outputs



The function **msg.m** randomly generates a *k*-length binary output sequence, *u*, with the random number generator seed set by parameter, *s*. The function **bm.m**, representing a binary to M-ary converter, transforms a variable length binary input sequence, *v*, into an equivalent M-ary output sequence, *m*, depending on the value selected for *q*, the word bit length. By accepting as an input the random binary output generated by m-file **msg.m**, **bm.m** groups bits together *q*-bits at a time to form words representing M-ary symbols whose output is a vector of equivalent decimal numbers. Padding with zeros may be necessary to ensure a complete *q*-bit word formation. Table 2 and Table 3 illustrate the function operations through examples.

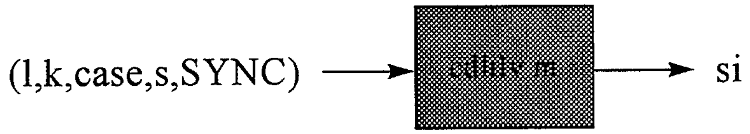
```
» bin1 = msg(10,20)   % Random binary sequence of length 20 with seed of 10.

bin1 = 0   1   1   1   0   0   0   1   1   0   1   1   0   0   1   0   0   0   0   1

» bin2 = msg(11,13)   % Random binary sequence of length 13 with seed of 11.

bin2 = 1   0   1   0   1   1   0   1   0   0   0   0   1
```

Table 2. Msg.m Sample Outputs

```
» mary1 = bm(3,bin1) )   % 3-ary equivalent of  binary test pattern, bin1.

mary1 =  6    1    6    6    4    0    2
```

```
» mary2 = bm(4,bin1) )   % 4-ary equivalent of  binary test pattern, bin1.

mary2 = 14    8    13    4    8

» mary3 = bm(3,bin2) )   % 3-ary equivalent of  binary test pattern, bin2.

mary3 = 5    6    2    0    1

» mary4 = bm(4,bin2) )   % 4-ary equivalent of  binary test pattern, bin2.

mary4 = 5    11    0    1
```

Table 3. Various Bm.m Sample Outputs



After the randomly generated source message of M-ary symbols displayed in decimal notation is formed, the array is next interleaved by the m-file function **cdlilv.m**. This m-file has a five argument input and a single output. Parameters, *l* and *k*, determine the dimensions of the interleaver intermediate matrix where *l* is the number of rows and *k* is the number of columns. The parameter, *case*, is an input that selects which desired interleaving method should be included. There are nine different interleaving cases. Case 0 represents a conventional block interleaver where the message symbol array is read into the intermediate matrix by rows, then immediately read out of the same matrix by columns as demonstrated in Fig. 12. Cases 1 through 8 represent cyclically shifted interleaver cases of the type previously incorporated in the Common Data Link (CDL) simulation (Fig. 14). After the message block is read into the intermediate matrix by rows, the symbols are then cyclically shifted by rows and/or columns within the intermediate matrix as a function of the case number. The rows may be shifted to the left (row negative) or to the right (row positive). Similarly, the columns may be shifted upward (column negative) or downward (column positive). The algorithm that determines the amount of row and/or column shift is given by (27), and is a proprietary Unisys Corporation design previously included in the CDL simulation model [6], [14]. The eight cases indicating shift direction are summarized as follows:

Case 1 - Column negative,
Case 2 - Column positive,
Case 3 - Row negative,
Case 4 - Row positive,
Case 5 - Row negative/column negative,
Case 6 - Row negative/column positive,
Case 7 - Row positive/column negative,
Case 8 - Row positive/column positive.

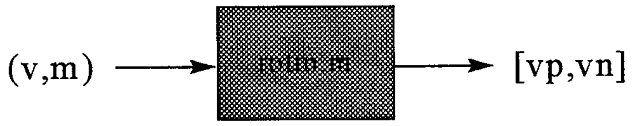
After appropriate cyclical shifting is performed inside the intermediate matrix as determined by the *case* number, the interleaved symbols are read out of the intermediate matrix by columns and represent the output interleaved sequence, *si*. The remaining input parameter, *SYNC*, is a left over from the CDL simulation model and represents the interleaver synchronization code word which overwrites the last sixteen bits of the interleaved output. For purposes of this COFDM model, *SYNC* is not necessary and therefore is not used by setting it equal to an empty vector in this simulation.

It is also possible to bypass the interleaving operation entirely by appropriately selecting the number of rows, *l*, or the number of columns, *k*, of the intermediate matrix dimensions to be one. In this way, the construction of intermediate matrix actually takes the form of either a row or column vector. Column or row vectors cannot be effectively interleaved using this method; thus, the output sequence of **cdlilv.m** is identical to the input sequence with no effective interleaving accomplished. Table 4 demonstrates CDL interleaving by example.

```
» msg2   % Original source message array with 3 rows and 8 columns (16-ary symbols).


msg2 =
    8    9    6    9    5   10    9    7
    2   12    9    2    8    7    8    9
    5   13    7   12    3    1    0    9

» msg2t = msg2'   % Transpose msg2 array into msg2t with 8 rows and 3 columns.


msg2t =
    8    2    5
    9   12   13
    6    9    7
    9    2   12
    5    8    3
   10    7    1
    9    8    0
    7    9    9

» msg2vect = msg2t(:)'   % Msg2 reformatted as a row vector with 1 row and 24 columns.


msg2vect =
  Columns 1 through 12
    8    9    6    9    5   10    9    7    2   12    9    2
  Columns 13 through 24
    8    7    8    9    5   13    7   12    3    1    0    9
» si = cdlilv(6,4,0,msg2vect,[])   % Perform block interleaving (case 0) on msg2 using a (6,4)
                                     intermediate matrix.

Intermediate_mx =
   8    9    6    9
   5   10    9    7
   2   12    9    2
   8    7    8    9
   5   13    7   12
   3    1    0    9


si =
  Columns 1 through 12
    8    5    2    8    5    3    9   10   12    7   13    1
  Columns 13 through 24
    6    9    9    8    7    0    9    7    2    9   12    9

» msg2intlv = reshape(si,8,3)'   % Reshape the interleaved message output sequence with the
                                     original msg2 array dimensions, 3 rows by 8 columns.
msg2intlv =
       8    5    2    8    5    3    9   10
      12    7   13    1    6    9    9    8
       7    0    9    7    2    9   12    9

si = cdlilv(6,4,5,msg2vect,[])  % Perform cyclical shift interleaving (case 5) on msg2 using a
                                  (6,4) intermediate matrix.
Intermediate_mx =
   8    9    6    9
   5   10    9    7
   2   12    9    2
   8    7    8    9
   5   13    7   12
   3    1    0    9


si =
  Columns 1 through 12
    8   10    2    8    7    9    9    2    9   12    3    9
```

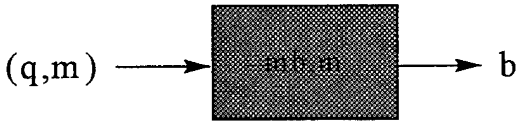
```
 Columns 13 through 24
   8    5    1    6    7   12    9    5    9    7   13    0

» msg2intlv  = reshape(si,8,3)'  % Reshape the interleaved message output sequence with the
                                    original msg2 array dimensions, 3 rows by 8 columns.
msg2intlv  =
   8   10    2    8    7    9    9    2
   9   12    3    9    8    5    1    6
   7   12    9    5    9    7   13    0
```

Table 4. Cdlilv.m Interleaving Examples



Cdlilv.m makes use of the subroutine m-file, **rotm.m.** The m-file, **rotm.m**, accepts two input arguments, variable *v*, being the input vector to be rotated, and variable *m*, the number of positions the columns and/or rows are to be shifted. The outputs, *vp* and *vn*, are vectors which represent either positive shifts, *vp*, or negative shifts, *vn* acting on the input vector symbols.

After the interleaving operation, the interleaved message array is converted from an M-ary format to a N-ary format suitable for N-PSK modulation. The symbol format conversion process is accomplished by two separate m-file routines, **mb.m** and **bm.m**. The function **mb.m** accepts two input variables and represents a M-ary to binary converter. The input *q* is the number of bits defining the M-ary symbols where $M = 2^q$. The remaining input, *m*, represents the incoming M-ary message array. The single output from this block, *b*, is a binary data sequence whose information content is equivalent to the coded M-ary symbols.



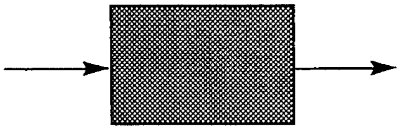
The binary output sequence generated by **mb.m** is next fed as an input to **bm.m**. Recall that the function **bm.m** converts a variable length binary input sequence, *v*, into an equivalent N-ary output symbol sequence, m, where N = 2^q. In this way, the combination of m-files **mb.m** and **bm.m** functions effectively convert the interleaved message information block from an array containing M-ary symbols to one consisting of N-ary symbols.

With the desired bit values determining M and N chosen by the user, the size of the N-ary message array may change since additional symbols may be formed, or likewise there may be a reduction in the number of symbols. However, the number of columns in the final symbol message matrix consistently remains unaltered as they represent the number of OFDM sub-carrier tones and remain fixed for each simulation. If the message block size must increase or decrease as a result of M-ary to N-ary symbol format conversion, the adjustment is accomplished by increasing or decreasing the number of rows in the message block only, never the number of columns. For example, given any arbitrary message array to be converted form M-ary symbol format to equivalent N-ary symbol format, if M is chosen to be 256 and N is 16, then the number of equivalent N-ary symbols representing the input message array will increase two-fold from the original total. Consequently, the number of rows forming the output matrix doubles, while the column number remains the same. As a function of the desired M-ary and N-ary configuration, a pad of zero symbols may be automatically inserted to ensure a full array. In the receiver, the zero pad is removed, leaving behind the randomly generated message source. Table 5 depicts a M-ary to N-ary conversion example.

```
mary_msg = marymsg(8,20,3,8)   % Random 3 row by 8 column message source using 256-ary
                                 (M-ary) symbols and a seed of 20.

mary_msg =

 152  150  165  121  194   41  120  152

 213  199   19  144  231   33  224  229

   1   73  171  142  214   51  102   81

» nary_msg = reshape(bm(4,mb(8,mary_msg)),8,6)'   % Reformat the message source symbols
                         as a 6 row by 8 column array using equivalent 8-ary (N-ary) symbols.

nary_msg =

   8    9    6    9    5   10    9    7

   2   12    9    2    8    7    8    9

   5   13    7   12    3    1    0    9

   7   14    1    2    0   14    5   14

   1    0    9    4   11   10   14    8

   6   13    3    3    6    6    1    5
```

Table 5. M-ary to N-ary Symbol Conversion Example

After the interleaving and M-ary to N-ary conversion operations are accomplished, the message array containing information symbols represented in decimal notation, is differentially encoded then channel encoded as an array of complex modulation values suitable for N-PSK modulation. The symbol-to-complex-modulation-value mapping process is accomplished using the m-file, **difcdrft.m**. This function has a three argument input and a single output consisting of differentially encoded complex modulation values, *MD*, in array format.



The input, *fort*, determines if the array, *m*, of decimal numbers is to be differentially encoded in time or in frequency by performing a cumulative summation of array elements. If *fort* is zero, time differential encoding is performed on the message array, *m*, by executing a cumulative summation down each column. If *fort* is one, frequency differential encoding is performed by similarly performing a cumulative summation across each row in the message array, *m*. Recall that array columns correspond to OFDM frequencies, while array rows represent information symbols generated in time.

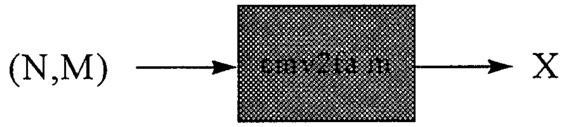
Cumulative summations of the input array are accomplished by adding in modulo-N fashion the first element of the appropriate column or row vector to the next adjacent element, replacing the second element by the current summation, then adding this current sum to the third element and replacing that element with the current sum. This process is repeated until all elements in the row (frequency differential encoding) or column (time differential encoding) are exhausted. The cumulative summation process then repeats beginning with the first element of the next row or column respectively.

After differential encoding with modulo-N cumulative summations, the array, *m*, is next channel encoded as N-ary complex modulation values. The input, *p*, indicates the number of unit circle phase partitions formed based upon the N-PSK modulation scheme where $N = 2^p$. The mapping process begins by accepting the input symbol message array, *m*, and generating corresponding complex modulation values, MD, with unit magnitude and one of N possible phases. Recall that complex modulation numbers are described by magnitude, A, and phase, , of the form Ae^{j} . For example, if N = 16, then $p = \log_2 16 = 4$. Thus, each input symbol is a decimal number with a range of values from zero to fifteen. After modulation, each symbol becomes a complex modulation value with a magnitude of one (A = 1) and possible phase values selected from the set, {± 22.5, ± 45, ± 67.5, ± 90, ± 112.5, ± 135, ± 157.5, 0, 180} degrees.

As a final step, a reference row of ones (zero phase angles) are appended to the message array, *m*, at the top to provide a reference starting point for the differential decoding performed in the COFDM receiver. Similarly, for frequency differential encoding, a reference column pair of ones (zero phase angles) are appended to the message array, *m*, at the left. Two reference ones columns are appended to maintain an even number of OFDM frequencies. Consequently, output *MD* includes the additional reference ones within the complex modulation array. In the receiver, these reference values are stripped off during differential decoding. Table 6 provides an example.

```
» nary_msg = marymsg(4,30,3,4)  % Random 3 row by 4 column message source using 16-ary
                                  symbols and a seed of 30.

nary_msg =
   14    0    4   14
    4    9    9    6
   13    6    8    9

» complex_mod_vals = difcdrft(4,nary_msg,0)  % Corresponding complex modulation values for
                                               16-ary symbols and time differential encoding.

complex_mod_vals =
   1.0000             1.0000             1.0000             1.0000
   0.7071 - 0.7071i   1.0000             0.0000 + 1.0000i   0.7071 - 0.7071i
   0.7071 + 0.7071i  -0.9239 - 0.3827i   0.3827 - 0.9239i   0.0000 + 1.0000i
   0.9239 - 0.3827i   0.9239 - 0.3827i  -0.3827 + 0.9239i   0.3827 - 0.9239i

» complex_mod_vals = difcdrft(4,nary_msg,1) % Corresponding complex modulation values for
                                              16-ary symbols and frequency differential encoding.

complex_mod_vals =
   1.0000   1.0000   0.7071 - 0.7071i   0.7071 - 0.7071i   0.7071 + 0.7071i   1.0000 - 0.0000i
   1.0000   1.0000   0.0000 + 1.0000i   0.3827 - 0.9239i  -0.7071 + 0.7071i   0.0000 - 1.0000i
   1.0000   1.0000   0.3827 - 0.9239i   0.3827 + 0.9239i  -0.3827 - 0.9239i   0.0000 + 1.0000i
```

Table 6. Complex Modulation Value Example



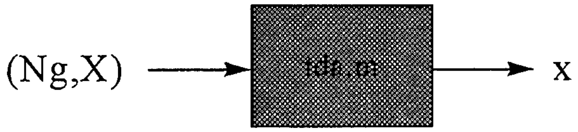
As a final step in the source encoding block, and in preparation of OFDM frequency generation through the IFFT, the input array of complex modulation values, *M*, are rearranged into a special frequency array by the m-file **cmv2fa.m**. The second input variable, *N*, is the number of FFT points used which must be larger than the number of columns of complex modulation values in the array (number of OFDM frequencies). This function also swaps the positions of the modulation values by grouping the left half portion of the matrix elements and shifting them to the rightmost positions, and likewise grouping the right half portion of the matrix elements and shifting them to the leftmost positions. Swapping is performed in anticipation of the frequency spectrum shifting that automatically results from FFT processing. When the MATLAB FFT command is invoked, the negative spectral frequencies to be shifted to the rightmost positive locations by *N* positions. Thus, the spectrum is no longer symmetrical about the origin; but, instead becomes symmetrical about the frequency point *N*/2. If the frequency halves are swapped before IFFT processing, then the frequencies can be later recovered in their correct orientation by filtering.

The shifted frequency array output is represented by *X*. A pad of zeros is included in the middle of the array whose amount is the difference between the number of FFT points, *N*, and the number of modulation values. The zero pad is included as a guard band to account for filter slopes during subsequent bandpass filtering after up-conversion and RF transmission. This filtering is not actually performed for the thesis simulations; however, the guard band is included for actual implementation purposes. Table 7 gives an example.

```
» freq_array = cmv2fa(8,complex_mod_vals)  % Frequency array formation using 8 point FFT
                                             and time differential encoding.

freq_array =

 Columns 1 through 4

  1.0000            1.0000            0        0
  0.0000 + 1.0000i  0.7071 - 0.7071i  0        0
  0.3827 - 0.9239i  0.0000 + 1.0000i  0        0
 -0.3827 + 0.9239i  0.3827 - 0.9239i  0        0

 Columns 5 through 8

   0        0        1.0000            1.0000
   0        0        0.7071 - 0.7071i  1.0000
   0        0        0.7071 + 0.7071i -0.9239 - 0.3827i
   0        0        0.9239 - 0.3827i  0.9239 - 0.3827I

freq_array = cmv2fa(8,complex_mod_vals)  % Frequency array formation using 8 point FFT
                                           and frequency differential encoding.

freq_array =

 Columns 1 through 4

  0.7071 - 0.7071i  0.7071 + 0.7071i  1.0000 - 0.0000i   0
  0.3827 - 0.9239i -0.7071 + 0.7071i  0.0000 - 1.0000i   0
  0.3827 + 0.9239i -0.3827 - 0.9239i  0.0000 + 1.0000i   0

 Columns 5 through 8

   0      1.0000      1.0000      0.7071 - 0.7071i
   0      1.0000      1.0000      0.0000 + 1.0000i
   0      1.0000      1.0000      0.3827 - 0.9239I
```

Table 7. Frequency Array Example

After source encoding, the complex modulation frequency array, *X*, is IFFT processed within the m-file, **tda.m,** generating the OFDM frequencies. The **tda.m** function also prepares the transmitted symbols for channel compensation by first appending the periodic guard interval whose length is indicated by the input, *Ng*. *Ng* represents the number of additional time domain waveform samples to add to the beginning of the information symbol interval. The output, x , is the time domain samples suitable for transmission and consisting of an array of complex samples. This functional block is the final block the message signal enters before transmission through the channel. Again, for purposes of this thesis, DAC and up-conversion of the signal is not included, permitting all simulations to be performed at baseband. See Table 8 for an example.



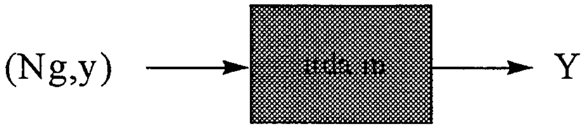
```
» Time_domain_sig = tda(5,freq_array)  % Generate time domain OFDM signal using IFFT and
                                         add 5 sample point precursor.

Time_domain_sig =


 Columns 1 through 4
 -0.0518 + 0.1250i        0              -0.0518 - 0.1250i        0
  0.0000 + 0.2500i  -0.1250 + 0.1250i  -0.3018 + 0.1250i  -0.1768 + 0.2500i
 -0.0811 + 0.0000i   0.2517 - 0.1043i   0.3401 - 0.3401i   0.1323 - 0.3194i
 -0.0676 + 0.2986i  -0.0957 + 0.2310i  -0.2590 + 0.1633i  -0.2310 + 0.2310i

 Columns 5 through 8
  0.3018 + 0.1250i   0.5000             0.3018 - 0.1250i        0
  0.1768 + 0.1768i   0.3018 - 0.0518i   0.1250 - 0.0518i   0.0000 + 0.1768i
  0.0000 - 0.0542i   0.0207 + 0.0501i  -0.0676 - 0.0676i  -0.2134 - 0.0884i
  0.0676 + 0.1633i   0.2310 - 0.0957i   0.0676 - 0.1633i  -0.0957 + 0.0957i

 Columns 9 through 13
 -0.0518 + 0.1250i        0              -0.0518 - 0.1250i        0              0.3018 + 0.1250i
  0.0000 + 0.2500i  -0.1250 + 0.1250i  -0.3018 + 0.1250i  -0.1768 + 0.2500I   0.1768 + 0.1768i
 -0.0811 + 0.0000i   0.2517 - 0.1043i   0.3401 - 0.3401i   0.1323 - 0.3194I   0.0000 - 0.0542i
```

```
-0.0676 + 0.2986i  -0.0957 + 0.2310i  -0.2590 + 0.1633i  -0.2310 + 0.2310I  0.0676 + 0.1633i


» Time_domain_sig = tda(2,freq_array)   % Generate a time domain OFDM signal using IFFT
                                          and add 2 sample point precursor.
Time_domain_sig =


 Columns 1 through 5
     0               0.3018 + 0.1250i  0.5000            0.3018 - 0.1250I      0
 -0.1768 + 0.2500i   0.1768 + 0.1768i  0.3018 - 0.0518i  0.1250 - 0.0518I  0.0000 + 0.1768i
  0.1323 - 0.3194i   0.0000 - 0.0542i  0.0207 + 0.0501i -0.0676 - 0.0676I  -0.2134 - 0.0884i
 -0.2310 + 0.2310i   0.0676 + 0.1633i  0.2310 - 0.0957i  0.0676 - 0.1633I  -0.0957 + 0.0957i

 Columns 6 through 10
-0.0518 + 0.1250i       0              -0.0518 - 0.1250I       0              0.3018 + 0.1250i
 0.0000 + 0.2500i  -0.1250 + 0.1250i  -0.3018 + 0.1250I  -0.1768 + 0.2500i  0.1768 + 0.1768i
-0.0811 + 0.0000i   0.2517 - 0.1043i   0.3401 - 0.3401I   0.1323 - 0.3194i  0.0000 - 0.0542i
-0.0676 + 0.2986i  -0.0957 + 0.2310i  -0.2590 + 0.1633I  -0.2310 + 0.2310i  0.0676 + 0.1633i
```

Table 8. Time Domain Signal Example

b. Channel 0 Model

As previously mentioned, the model 0 channel is noise free. The overall model 0 system can be viewed as the OFDM transmitter directly connected to the OFDM receiver by a direct link since there is no intervening stimulus affecting the signal. Thus, the channel 0 functional block can be pictorially represented simply as a wire connecting the transmitter to the receiver without a separate additional MATLAB functional block.



c. COFDM Receiver

The receiver portion of the model 0 system depicted in Fig. 24 contains two primary functional blocks, **itda.m** and **decdrcdl.m**. The m-file **itda.m** transforms the time domain received complex signal, *y*, into an equivalent frequency domain representation by performing the FFT. The FFT allows recovery of the OFDM frequencies generated in the transmitter block. Prior to FFT processing, the *Ng* point precursor guard interval is removed from each of the symbols. The format of the output, *Y*, is a frequency array of complex modulation values with the left and right half portions of the array interchanged in frequency index positions as exemplified in Table 9.

```
» Recvd_CMV_array = itda(5,Time_domain_sig)  % Remove 5 sample precursor and perform FFT to
transform received time domain samples into corresponding orthogonal complex modulation values with
left and right halves swapped.

Recvd_CMV_array =
  Columns 1 through 4
   1.0000              1.0000 - 0.0000i           0 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 1.0000i    0.7071 - 0.7071i    0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3827 - 0.9239i    0.0000 + 1.0000i    0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3827 + 0.9239i    0.3827 - 0.9239i    0.0000 + 0.0000i   0.0000 - 0.0000i
  Columns 5 through 8
        0              0.0000 + 0.0000i    1.0000 - 0.0000i   1.0000 - 0.0000i
   0.0000              0.0000 - 0.0000i    0.7071 - 0.7071i   1.0000 - 0.0000i
   0.0000 + 0.0000i    0.0000 - 0.0000i    0.7071 + 0.7071i  -0.9239 - 0.3827i
        0              0.0000 + 0.0000i    0.9239 - 0.3827i   0 .9239 - 0.3827i
```

Table 9. Received Frequency Array Example of Complex Modulation Values

The remaining receiver decoding functions are performed within the **decdrcdl.m** block by multiple sub-blocks which are presented in Fig. 29. The hierarchical arrangement of m-files within **decdrcdl.m** are presented in Fig. 30.

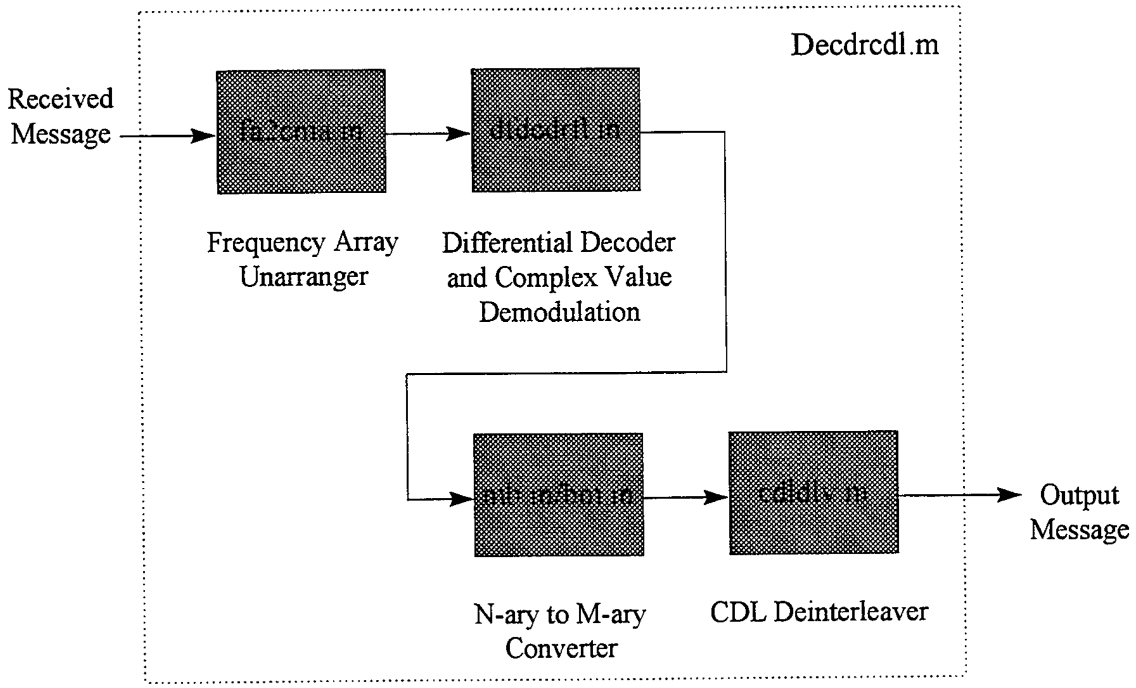


Fig. 29. M-file Decdrcdl.m Functional Sub-blocks

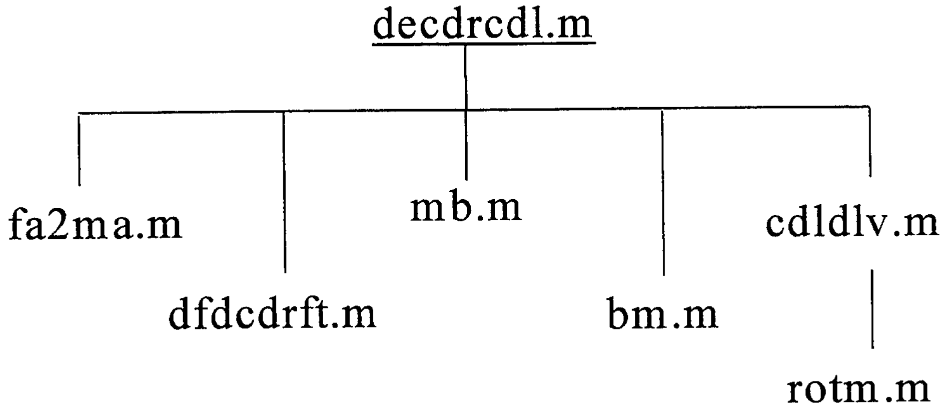
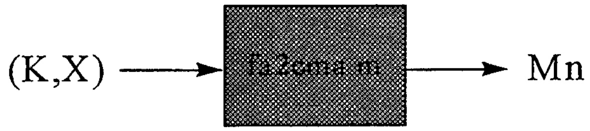


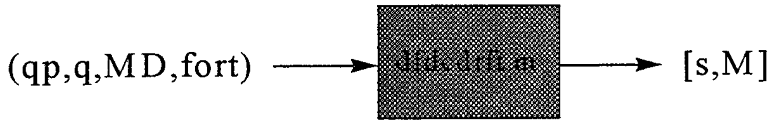
Fig. 30. M-file Hierarchy for Decdrcdl.m

The frequency array is restructured back into the proper complex modulation array format by the **fa2cma.m** m-file within **decdrcdl.m**. The function **fa2cma.m** accepts the input *K* indicating half the number of OFDM frequency tones (corresponds to frequencies occupying one-half of the frequency array). The remaining input, *X*, are the complex frequency array values to be rearranged. The output, *Mn*, is the equivalent complex modulation array representation with the correct ordering of frequencies seen in Table 10.



```
» Freq_unarranger = fa2cma(2,Recvd_CMV_array)  % Unarrange the left and right halves of the
                                                  frequency array to the correct orientations.

Freq_unarranger =
  1.0000 - 0.0000i  1.0000 - 0.0000i  1.0000            1.0000 - 0.0000i
  0.7071 - 0.7071i  1.0000 - 0.0000i  0.0000 + 1.0000i  0.7071 - 0.7071i
  0.7071 + 0.7071i  -0.9239 - 0.3827i  0.3827 - 0.9239i  0.0000 + 1.0000i
  0.9239 - 0.3827i  0.9239 - 0.3827i  -0.3827 + 0.9239i  0.3827 - 0.9239i
```

Table 10. Unarranged Frequency Array Example

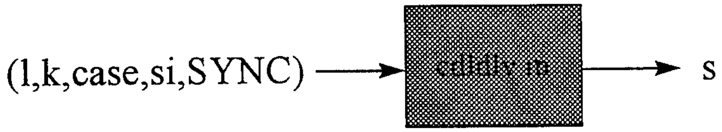


After the **fa2cma.m** block, the complex modulation values are differentially decoded either in time or in frequency, then hard decoded into corresponding N-ary symbols. This functionality is accomplished by the m-file **dfdcdrft.m**. The complex modulation values, *MD*, from **fa2cma.m** are accepted as an input, and inverse mapping of the complex numbers to N-ary symbols is performed based upon the value of *q*, where N = 2^q. If *fort* is equal to one, frequency differential decoding is performed. If *fort* is equal to zero then time differential decoding is performed. Differential decoding is the inverse operation performed in the transmitter; however, regardless of the type of differential decoding, all reference ones values are removed after decoding allowing the received message matrix to remain. The input, *qp*, is used for soft decoding and allows for extra functionality not included in this system model; hence, it is not used. The output, *s*, indicates phase sector numbers corresponding to N-ary demodulation also representing corresponding inverse mapped symbols in decimal notation. The remaining output, *M*, is the differentially decoded modulation array. Table 11 presents an example.

```
» [symbols DiffDecd_values] = dfdcdrft(4,4,Freq_unarranger,0)  % Differentially decode in time and
demodulate the complex modulation values into corresponding 16-ary symbols.

symbols =
   14    0    4   14
    4    9    9    6
   13    6    8    9

DiffDecd_values =
   0.7071 - 0.7071i   1.0000 - 0.0000i   0.0000 + 1.0000i   0.7071 - 0.7071i
   0.0000 + 1.0000i  -0.9239 - 0.3827i  -0.9239 - 0.3827i  -0.7071 + 0.7071i
   0.3827 - 0.9239i  -0.7071 + 0.7071i  -1.0000 + 0.0000i  -0.9239 - 0.3827i
```

Table 11. Demodulated Received Signal Example

With the reception of the message in N-ary format consisting of PSK symbols, a reformatting of symbols to M-ary is next performed to form OFDM symbols. Once again the functions **mb.m** and **bm.m** perform the reformatting procedure as previously described in the transmitter section.



As a final operation in the receiver, the message symbol array is deinterleaved by the function **cdldlv.m** which performs the inverse operation of **cdlilv.m**. The input, *si*, is the received interleaved message, while *case* determines which deinterleaving case to follow (refer to the discussion on **cdlilv.m** for case descriptions) . It is important that the case number for the deinterleaver match that of the interleaver or numerous errors will occur as a result of unmatched deinterleaving. Inputs *l* and *k* indicate the intermediate matrix dimensions and must be identical to the intermediate dimensions used for the interleaver in the transmitter. The input vector, *si*, is read into the (*l*,*k*) matrix by columns, while the columns and/or rows are rotated in reverse direction and reverse order then they were rotated in **cdlilv.m** for the same case number. The output, *s*, provides the final message array read out of the intermediate matrix by rows. **Cdldlv.m** calls the subroutine m-file, **rotm.m** which performs the array rotations as previously described in **cdlilv.m.** The example shown in Table 12 demonstrates operation of the CDL deinterleaver corresponding to case 5.

```
» Intrlvd_symbols =  % Decoded M-ary symbol array.
   14    0    4   14
    4    9    9    6
   13    6    8    9

» Intrlvd_symbolst = Intrlvd_symbols'  % Transposed symbol array.



Intrlvd_symbolst =
   14    4   13
    0    9    6
    4    9    8
   14    6    9

» Intrlvd_symbol_vect = Intrlvd_symbolst(:)'  % Equivalent symbol vector.

Intrlvd_symbol_vect =
  14    0    4   14    4    9    9    6   13    6    8    9

» Deintrlvd_sym_vect = cdldlv(2,6,5,Intrlvd_symbols_vec,[]) % Deinterleave the input symbol vector
using a 2 row by 6 column intermediate matrix and case 5 type deinterleaving (note: SYNC is an empty
vector).

Deintrlvd_sym_vect =
  14   14    9    9   13    9    8    0    4    4    6    6

» Deintlvd_symbols = reshape(Deintrlvd_sym_vect,3,4)' % Reshape the resulting deinterleaved vector
into an array of message symbols corresponding to 4 OFDM tones. This output array represents the sink
message.

Deintlvd_symbols =
  14   14    9
   9   13    9
   8    0    4
   4    6    6
```

Table 12. CDL Deinterleaver Example

2. COFDM Model 1 System

The previously presented m-file functional blocks represent the model 0 system. However, the same transmitter and receiver blocks are also common to the model 1, model 2 and model 3 systems. The only differences are in the channel models. The model 1 block diagram is shown in Fig. 31 and represents channel 1 model consisting of the AWGN channel, which is implemented using the m-file **awgn.m**.

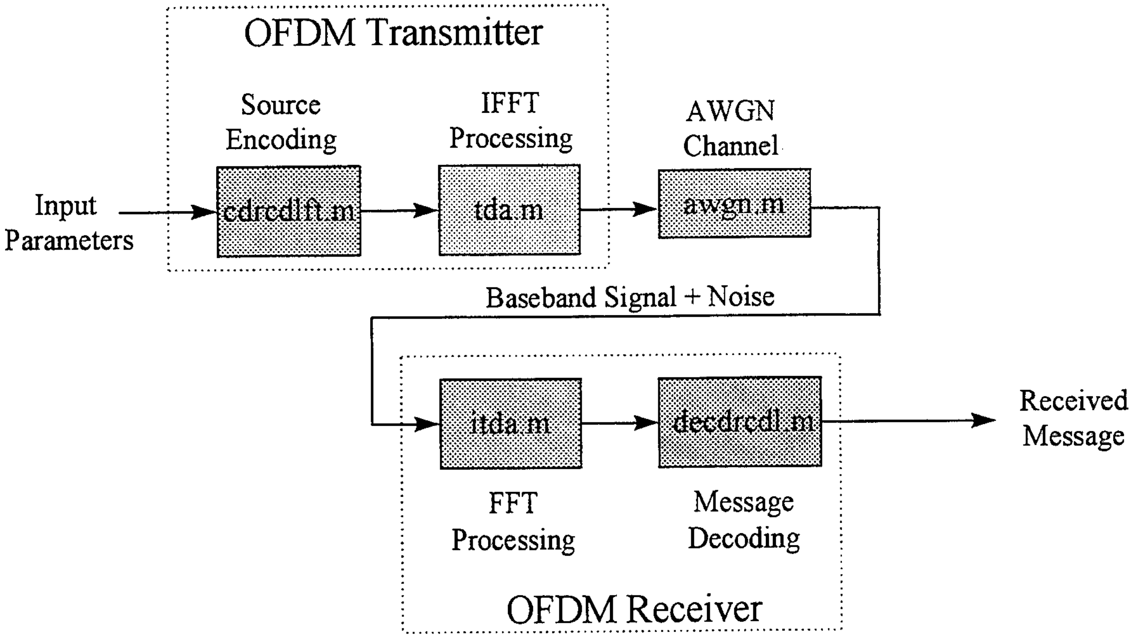
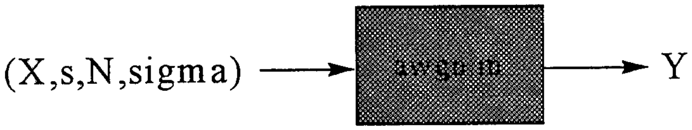


Fig. 31. System Model 1 Block Diagram



a. Channel 1 Model

Awgn.m adds complex white Gaussian noise to the input signal, *X*, consisting of time domain samples with real and imaginary parts. The seed parameter, *s*, sets the seed with the random phase and amplitude parameters being independently generated. The input variable, *sigma* ( ), determines the noise power spectral density, N_o, according to the formula

$$N_o = 2 \quad ^2 , \tag{37}$$

and is multiplied by the random noise vector to weight its strength before finally being added to the input signal. The input *N* represents the number of time domain samples of the input *X* and is also equal to the number of FFT points used for OFDM generation. The output *Y* consists of the original signal plus AWGN noise represented as complex real and imaginary number pairs. Recall that the complex signal amplitudes are fixed at unity and, therefore, so are the symbol energies; consequently, any power adjustments for bit error rate (BER) performance curve calculations is accounted for by varying *sigma*.

3. COFDM Model 2 System

The COFDM model 2 system is presented in Fig. 32 and is has identical transmitter and receiver components as the model 1 system, differing only in the channel model. The channel 2 model consists of the multipath channel exclusively which is implemented using the **chuhf.m** m-file. No other types of noise such as AWGN is added to this model; thus, the multipath effects on the transmitted signal can be individually analyzed.

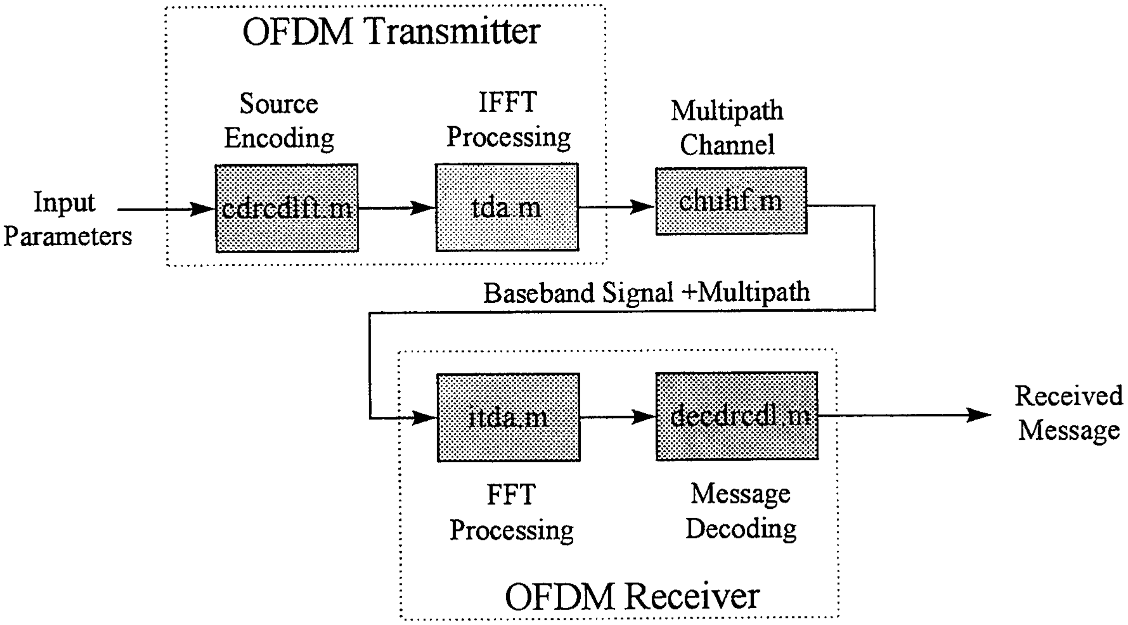
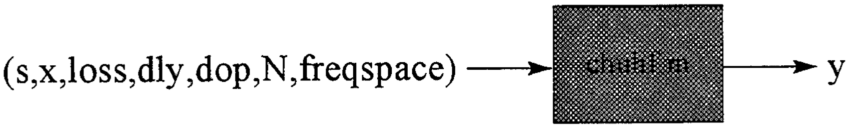


Fig. 32. System Model 2 Block Diagram

a. Channel 2 Model

M-file **chuhf.m** represents the channel 2 multipath model. The hierarchy for **chuhf.m** is shown in Fig. 33.



This m-file accepts as inputs the RSL power loss, *loss* (dB), time delays, *dly* (msec.), and Doppler frequency shifting, *dop* (Hz), characteristic of the maritime multipath channel. The transmitted signal, *x*, represents the time domain output of the COFDM transmitter consisting of complex numbers and is the input signal parameter to the channel model. Initially, the m-file **dline.m** is called

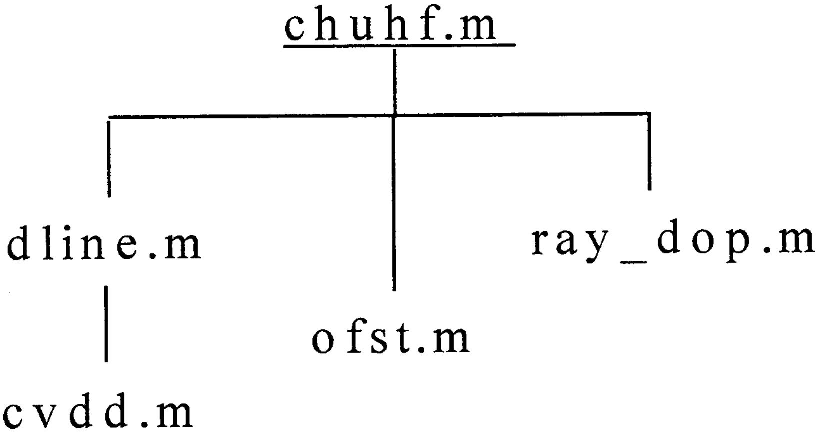


Fig. 33. M-file Hierarchy for Chuhf.m

to set-up the multiple delayed paths. Since the input, *dly*, can be a vector of delays, the number of delay lines corresponds to the number of elements in the vector. **Dline.m** in turn calls the subroutine m-file **cvdd.m** which implements a "continuously variable digital delay element" [15]. This m-file filters the *x* input using an eight-tap Finite Impulse Response (FIR) filter whose tap coefficients are a function of the desired delay.

Later, the m-file **ray_dop.m**, calculates the maximum Doppler shift frequency as a fraction of OFDM tone spacing as provided by the input, *freqspace*. This m-file generates a random sequence of length L*N independent points (L bauds of N samples per baud) of complex numbers with zero mean, and 0.5 variance real and imaginary parts. The envelope is Rayleigh with a mean square value of one. *N* is the number of FFT points. The amount of Doppler shifting is randomly calculated up to the maximum allowed using the seed parameter, *s*, to set the seed of the random number generator. The real and imaginary parts are independently generated, and it is acceptable to enter a vector of Doppler shift values equal to the number of delay paths. Additionally, the direct path is offset by 0.7 of the maximum input Doppler shift which is calculated by m-file **ofst.m**. As a final step in **chuhf.m**, the power losses for the individual multipaths are accounted for by multiplying each loss amount times the respective delay line output vectors. The output, *y*, is a time domain representation of the transmitted signal plus multipath effects, presented as an array of complex received time domain samples.

4. COFDM Model 3 System

The COFDM model 3 system is depicted in Fig. 34. In agreement with COFDM system models 1 and 2, the OFDM transmitter and OFDM receiver are identical to the functional sub-blocks presented in the COFDM model 0 system discussion. The only differences are in the channel model 3.

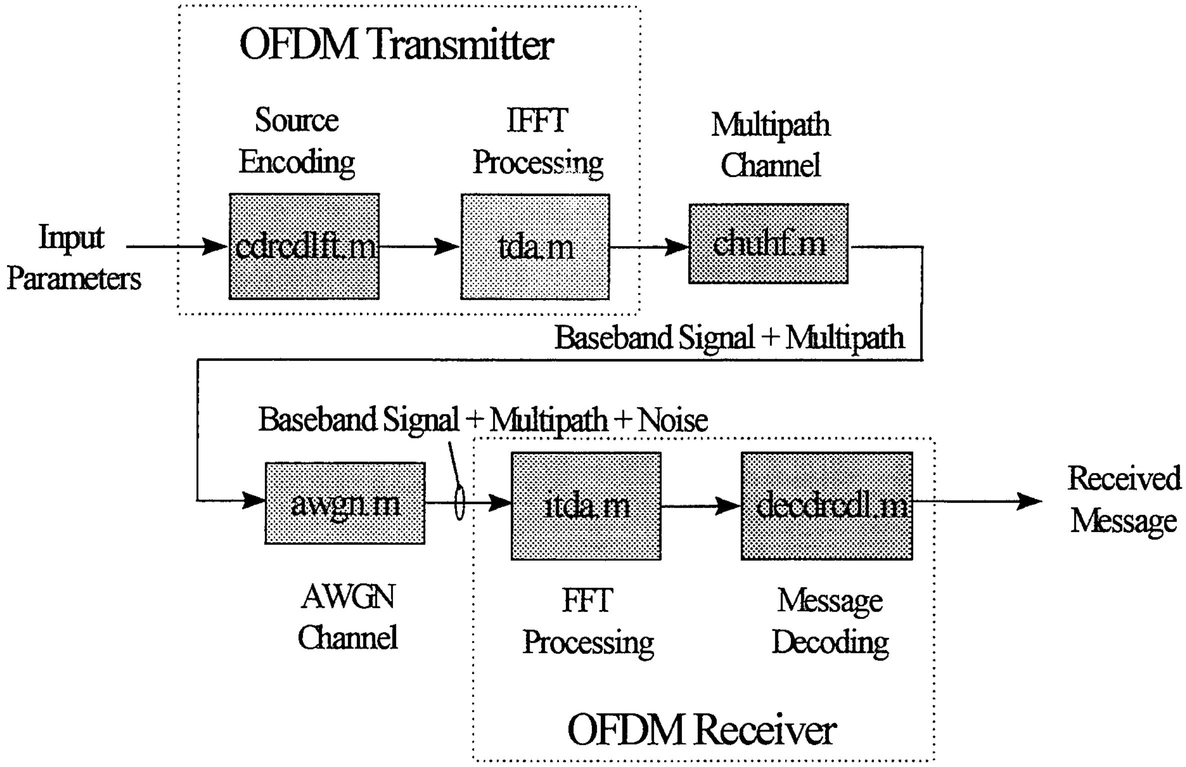


Fig. 34. System Model 3 Block Diagram

a. Channel 3 Model

The channel 3 model consists of the channel 1 model (AWGN) combined with the channel 2 model (multipath) to form an overall complete channel model representing the actual maritime environment described in the BAA specification and further described in reference [1]. Both the channel 1 model and channel 2 model have been previously described in detail, being implemented by m-files **awgn.m** and **chuhf.m,** respectively. The channel 3 model is used extensively in system performance analysis presented in the next chapter.

C. MATLAB VERIFICATION PROGRAM DEVELOPMENT

Upon completion of the system model construction using custom m-file functional blocks, additional diagnostic m-files were created aiding in debug and system functional verification. Most of the diagnostic m-files are not part of the COFDM model development with the exception of **check.m** which not only compares the source message to the decoded sink message and ascertains discrepancies but also includes the receiver R-S error correcting functionality to correct symbol errors depending on the configured R-S code strength.

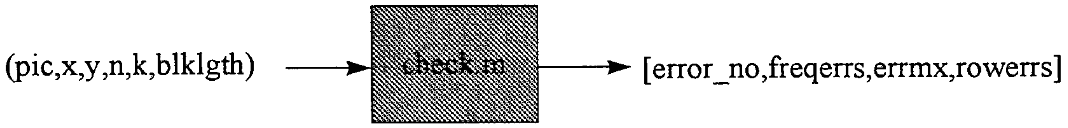
1. Source And Sink Message Comparer

To compare the randomly generated source message with the decoded received message the m-file **check.m** is used. This diagnostic m-file also includes the R-S error correcting code functionality. Recall that R-S FEC is a necessary part of the OFDM system model; however, the encoding portion of R-S coding is notably absent from the OFDM transmitter for all the system models. As previously mentioned, the omission of R-S symbol parity encoding in the transmitter using textbook coding algorithms was chosen to reduce computation and simulation run times. Since R-S encoding using well known algorithms has been previously demonstrated and implemented numerous times in communication systems, for purposes of this thesis it is not necessary to include the encoding operation in the model to prove functionality. Instead, R-S decoding is required in the receiver as a necessary functional block to enable symbol error correction capability as determined by the user.

Check.m is a six variable input, four variable output function. The input, *pic*, is a loop indicator variable useful in setting the figure numbers for plot displays during iterative simulation cycles to ensure that previously generated figures are not overwritten by subsequent plots. The inputs, *x*, and, *y*, are the two symbol message arrays of identical dimensions to be compared, *x* being the original source message array and *y* being the received message array often corrupted by channel induced noise errors. The remaining inputs *n*, *k*, and *blklgth* are used for symbol error correction operating on the received message array, *y*.

The pair (*n,k*) determine the error correcting strength, with *n* being the information word size and *k* the code word size. The maximum number of symbols errors, t, that can be corrected within a code block length, *blklgth*, is given by (25). The code block length is formed using symbols taken from message array columns starting with the first symbol (top left of the array). Hence, the error correction is accomplished across OFDM frequencies row by row as opposed to down symbol rows. For code block lengths that exceed the number of OFDM frequencies (number of message array columns), additional symbols are taken from the next lower adjacent row until the block is completely filled with symbols. The FEC parameters can be completely defined by the user including no FEC. If the number of symbol error occurrences within a code block length exceed the number of symbols that can possibly be corrected, t, then the entire block remains unchanged, including the symbol error locations, and the next code block is processed.

If the number of symbol error occurrences is less than or equal to the number of errors that can possibly be corrected, t, then all errored symbols are fixed to their correct value, resulting in an error-free block. Intuitively, it is apparent that the code block length in addition to the coding strength is an important parameter in ensuring effective symbol error correction. However, it is also important to be mindful of the code rate factor, since increasing code strength for a given block size causes a reduction of the information rate and, hence, reduced transmission efficiency. Table 13 presents examples.



```
» msg1
msg1 =
   9    4   13    3
```

```
    9    4    1    0
    1    2    7   15

» msg2
msg2 =
    9    4   13    5
    9    3    1    0
    2    2    7    5

» [err_no freqerr errmx rowerr] = check(0,msg1,msg2,4,4,4)
WARNING! Errors were detected!

WARNING!: Since n = k, there is no R-S error correcting possible.

For the given input parameters: n = 4 and k = 4, the Reed-Solomon code is capable
of correcting 0 errors.

OOOPS!: The Reed-Solomon code did not correct any errors.
Perhaps a more powerful R-S code is required.

The total number of error occurrences is: 4

The error number distribution per block number is:
    1    2    3
    1    1    2

err_no = 4

freqerr = 1    1    0    2

errmx =
    0    0    0    1
    0    1    0    0
    1    0    0    1
```

```
rowerr = 1    1    2

» [err_no.freqerr errmx rowerr] = check(0,msg1,msg2,4,2,4)
WARNING! Errors were detected!

For the given input parameters: n = 4 and k = 2, the Reed-Solomon code is capable
of correcting 1 errors.

OOOPS: The Reed-Solomon code corrected some detected errors, but not all.
Originally the error total was: 4

After R-S decoding, the error number was reduced to: 2

The total number of correct symbols are: 10

The error number distribution per block number is:
   1    2    3
   0    0    2

err_no = 2

freqerr = 1    0    0    1

errmx =
   0    0    0    0
   0    0    0    0
   1    0    0    1

rowerr = 0    0    2
```

Table 13. Check.m Example

2. Differential Encoder/Decoder and Frequency Array Checker

To verify proper operation of the differential encoder and decoder along with the frequency array arranger and unarranger, the m-file **cmvdifck.m** is used. This file generates a random M-ary message test pattern using **marymsg.m**, performs differential encoding and forms the frequency array. The inverse operations are later performed, namely frequency array unarranger and differential decoding. A comparison of the source message and the decoded sink message is performed by **check.m** to determine any discrepancies. As the following MATLAB example demonstrates (Table 14), both the differential encoder/decoder and frequency array arranger/unarranger function correctly for both time and frequency differential encoding cases.

```
» cmvdifck(10,5,10,16,4,4)
This m-file checks the correctness of the differential encoder/decoder & the
frequency arrangers.
To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the
time version: 1

Source_msg =
  14   8  13   4   8  13   1   4  12   0
   4   8   2   7   4   0  11   1   0   5
   0   2   7   0   4   9   7   2   2  10
   4   9   3  13   4  15   9   3  11  10
   5  11  15   9   1   4   5   0  12  14

Sink_Msg =
  14   8  13   4   8  13   1   4  12   0
   4   8   2   7   4   0  11   1   0   5
   0   2   7   0   4   9   7   2   2  10
   4   9   3  13   4  15   9   3  11  10
   5  11  15   9   1   4   5   0  12  14

GREAT!!! there are no errors.
```

```
TEST PASSED!!!
```

Table 14. Cmvdifchk.m Example

3. CDL Interleaver Verification

To confirm correct functional operation of the interleaver and deinterleaver, m-file **intlvchk.m** is used. Again a random source message array is formed using **marymsg.m**. The message block is then interleaved according to the desired case using the CDL interleaver, **Cdlilv.m**. Also included is m-file **Cdldlv.m** to deinterleave the symbol array using consistent case parameters. Finally **check.m** is used to verify correctness. This diagnostic m-file was run multiple times for all cases and confirmed proper functionality. The following Table 15 sample run depicts some interleaving examples.

```
» intlvchk(10,5,8,4,10,4,0)


Random_msg =

    14     8    13     4     8    13     1     4
    12     0     4     8     2     7     4     0
    11     1     0     5     0     2     7     0
     4     9     7     2     2    10     4     9
     3    13     4    15     9     3    11    10


Interleaved_array =

    14     4     0     4     8     8     2     9
    13     2     7     3     4     7     0    13
     8     4     4     4    13     0     9    15
     1    11     7     9     4     1     2     3
    12     0     2    11     0     5    10    10


Deinterleaved_array =

    14     8    13     4     8    13     1     4
```

```
  12   0   4   8   2   7   4   0
  11   1   0   5   0   2   7   0
   4   9   7   2   2  10   4   9
   3  13   4  15   9   3  11  10

GREAT!!! there are no errors.


» intlvchk(10,5,8,4,10,4,5)


Random_msg =
  14   8  13   4   8  13   1   4
  12   0   4   8   2   7   4   0
  11   1   0   5   0   2   7   0
   4   9   7   2   2  10   4   9
   3  13   4  15   9   3  11  10


Interleaved_array =
  14   8   0   9   2   4   3   8
  11  13   7   9   7  10   4   4
   2   4   8   0   9  13  11   2
   1  10   3   1   4   0   0  13
  12   5   2   4   4   7  15   0


Deinterleaved_array =
  14   8  13   4   8  13   1   4
  12   0   4   8   2   7   4   0
  11   1   0   5   0   2   7   0
   4   9   7   2   2  10   4   9
   3  13   4  15   9   3  11  10

GREAT!!! there are no errors.
```

Table 15. CDL Interleaver Checker Example

4. System Model 0 Checker

As previously indicated, the model 0 system consists of the OFDM transmitter and receiver interfaced to a perfect noise free channel. Functional verification of the sub-blocks contained in the transmitter and receiver and common to system models 1, 2 and 3 is accomplished by m-file **chn0cdl.m**. This function contains all OFDM fundamental system components contained in the system models with the exclusion of a channel. Basically this m-file can be thought of as the OFDM transmitter connected directly to the OFDM receiver. The purpose of building and using this m-file is to confirm proper operation and interaction of all of the sub-blocks connected together. This is accomplished by verification of source and sink messages using the **check.m** program. The m-file subroutine hierarchy for **chn0cdl.m** is shown in Fig. 35.

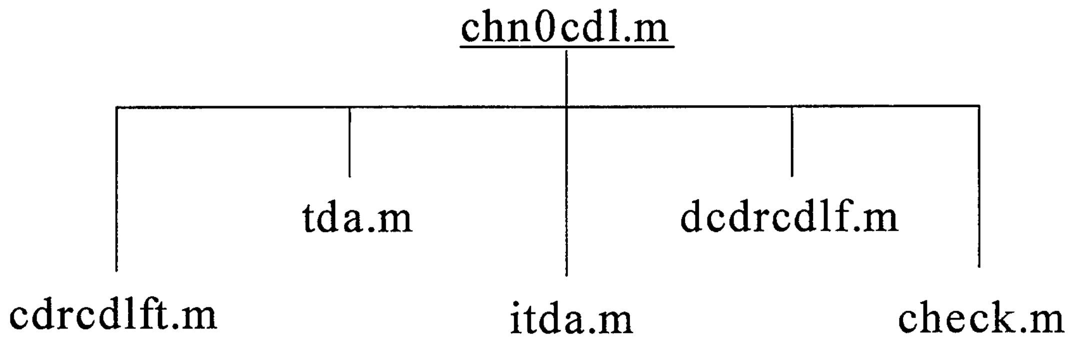


Fig. 35. M-file Hierarchy for Chn0cdl.m

Upon presentation of the model 3 system functional block diagram along with the MATLAB m-file programs emulating the system model, Fig. 36 once again displays the complete system model together with the corresponding m-files representing the emulated functional sub-blocks within. Appendix A. provides complete documentation of all m-file programs emulating system sub-blocks as well as system diagnostic programs and batch m-files. The next section discusses batch m-file creation used to perform system simulations.

D. MATLAB SIMULATION BATCH M-FILES

Simulation trials are accomplished using batch m-files which include some or all of the COFDM system model functional blocks. These m-files also include input requests displayed in the MATLAB command window for system configuration, as well as to query the user the option of generating figures, enabling data plots at various stages of processing, as well as printing hard-copies. Some batch programs permit multiple input argument variables (vector inputs) for certain configuration parameters, allowing multiple batch simulation repetitions using a different variable element for each loop. A reason for allowing vector inputs, for example, is to generate simulation data corresponding to different seed values while also choosing multiple interleaver cases. Batch files requesting input data are included to promote a more user-friendly simulation interface allowing for easier reconfigurations.

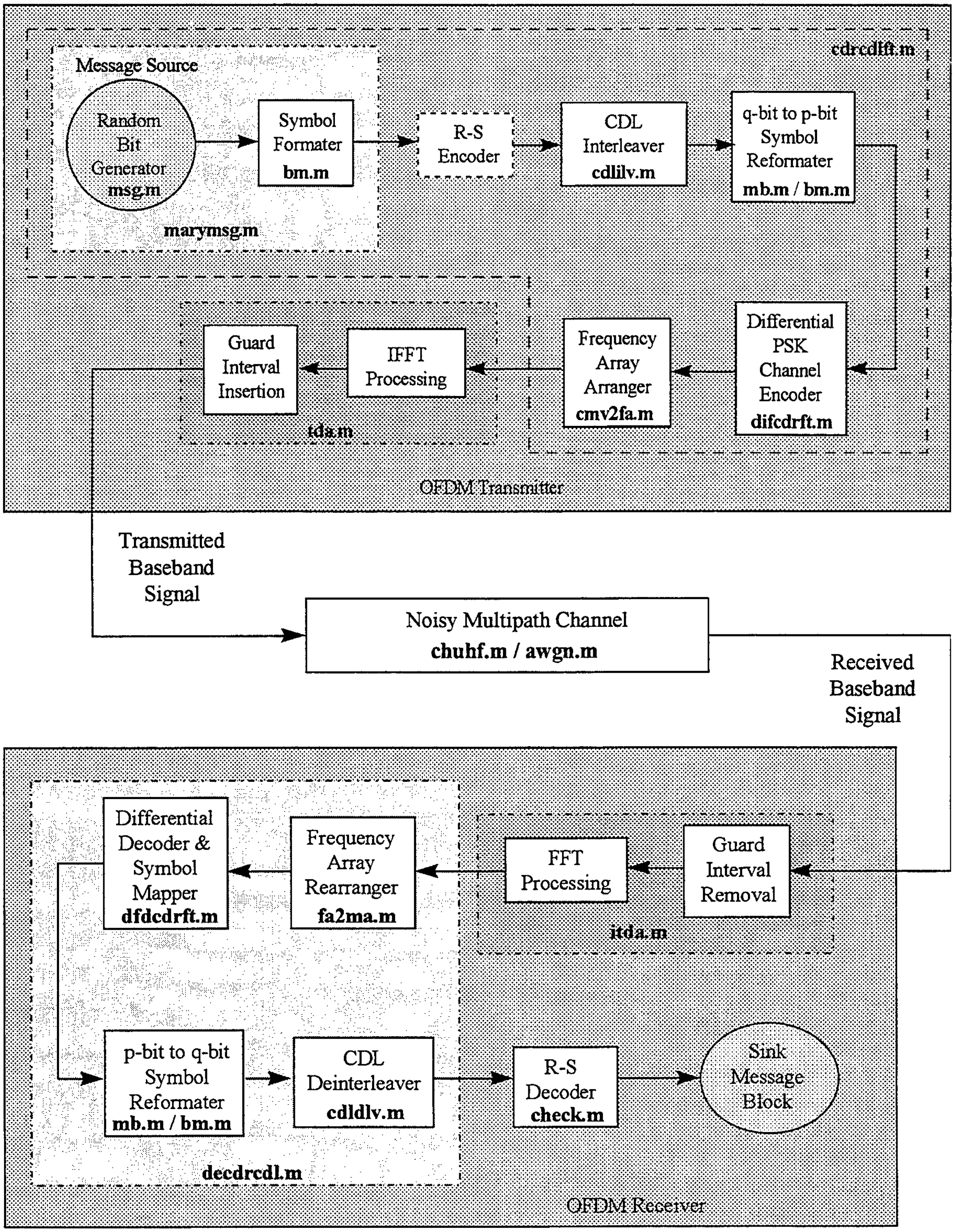


Fig. 36. Complete OFDM System Model With Corresponding M-files

1. Seed Evaluation Batch File

The batch m-file **uhfseeds.m** performs numerous COFDM system simulations using the channel 2 model (**chuhf.m**) with different seed configurations. The goal of this function is to test the multipath channel and identify the worst case channel conditions as determined by the seed parameter. Recall that the seed parameter is used by m-file **chuhf.m** to randomly generate multipath perturbations in the channel; thus, certain seed values generate a more severe burst noise environment than others do. Consequently, specific seeds tend to create a poor multipath channel and generate more errors during simulations. Error prone seeds are identified and used during subsequent model 3 system simulations to promote performance results reflecting worst case multipath channel behaviors. This helps to ensure that model 3 simulation data will represent worst case scenarios with respect to the channel conditions in the maritime environment. To isolate the channel 2 model, no interleaving or error correction is performed during **uhfseeds.m** trials as these techniques compensate for the multipath and affect the results. Different pre-defined transmission links (links 1 through 3) may be configured, as well as a custom user defined link. Table 16 gives an example of simulation initialization.

The output is a compilation of integer seed values beginning with one and increasing to a user defined maximum value, with corresponding error totals displayed in graphical form and based on simulations using either time or frequency differential encoding/decoding. Additionally, an error distribution plot is compiled and graphically displayed showing best case seeds growing toward worst case seeds as well as an error histogram.

This method of testing the channel to identify "bad" seeds is not extended to the channel 1 AWGN model. Typically, AWGN is uniformly distributed throughout message blocks and does not exhibit burst error behavior. Therefore, regardless of the chosen seed input to **awgn.m** used during model 1 simulations, on average the AWGN channel generates consistent performance results.

```
» uhfseeds   % Run this batch file to simulate channel 2, link 3, with 240 OFDM tones using 500
             seeds.

To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time
version: 1
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 3
Do you want print outs? (1 = yes, 0 = no): 0
Enter the minimum number of symbols to test: 10000
Enter the number of OFDM frequencies (NOTE: Must be even): 240
Enter the number of FFT points (NOTE: Must be larger than # of OFDM frequencies): 256
All tested seeds begin with one and end with a max number.  Enter Smax (Integer #): 500
Tested seed range is 1 - 500 ...
```

Table 16. Batch M-file Uhfseeds.m Example

The m-file hierarchy of **uhfseeds.m** is depicted Fig. 37. This m-file introduces the **uhfift.m** function used to emulate a version of system model 2 minus the interleaver in the transmitter and deinterleaver in the receiver. Interleaving/deinterleaving operations are omitted since, as previously mentioned, these blocks are designed to compensate for burst errors and improve overall model 2 performance; thus, including interleaving/deinterleaving defeats the purpose of this test. Consequently, the **uhfift.m** system structure is identical to the model 2 system without the interleaving/deinterleaving functionality. The remaining subroutine m-files comprising **uhfift.m** are consistent with the functional blocks previously described for the model 2 system.

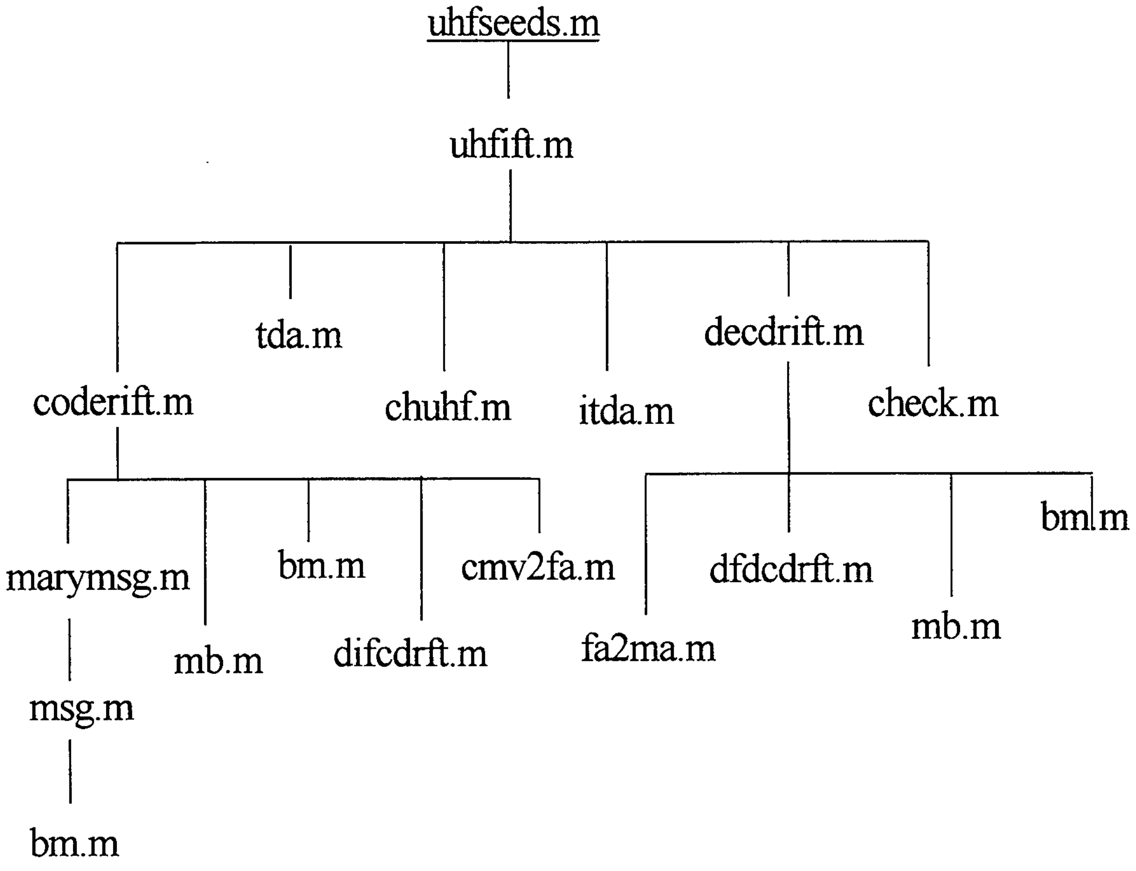


Fig. 37. M-file Hierarchy for Uhfseeds.m

2. COFDM System Simulation Batch File

To simulate the overall COFDM system model including all four channels, batch file **cofdmsim.m** is invoked. This m-file is created to test complete system models 0 through 3, comprehensively, and generate BER performance curves for various $^{Es}/_{No}$ values with respect to channels 1 and 3. The simulation performance results are compared to the theoretical curves and judgments made as to the accuracy of the model as well as the overall feasibility of various system configurations based upon performance merits. Table 17 provides a further example.

```
» cofdmsim
____________________________________________________________

This batch m-file runs OFDM simulations using different channel models.
To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or to run both enter 2
(two): 0
Enter the # of OFDM frequencies (Note: Must be even): 60
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 64
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter 0,1,2
or 3): 3
Channel model 3 simulation performed.
Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003):
linspace(0,0.01,10)
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 4
Custom link simulation...
Enter the link loss in dB (Ex. [0 4 7]): [0 2 4]
Enter the doppler frequency in Hertz (Ex. [30 20 15]): [10 20 30]
Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): [0 1.5 7.9]
Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): 0
Enter specific case numbers from (0 to 8) (Ex. [0 4 5 8]): [0 4 5 8]
Do you want to find optimal interleaver case(s)? (1 = yes, 0 = no): 0
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10020
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10020], or [10020 1], offers no
interleaving functionality): [60 167]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 2
Enter the guard interval length (Number of sample points): 6
Do you want to include error correction coding? (1 = yes, 0 = no): 1
Enter n,k and error correction block length (Ex. [240 200 240]): [200 180 200]
Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): [184 268 109]
Do you want signal plots? (1 = yes, 0 = no): 1
How many seconds of delay between pictures? 5
```

```
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 17. Batch M-file Cofdmsim.m Example

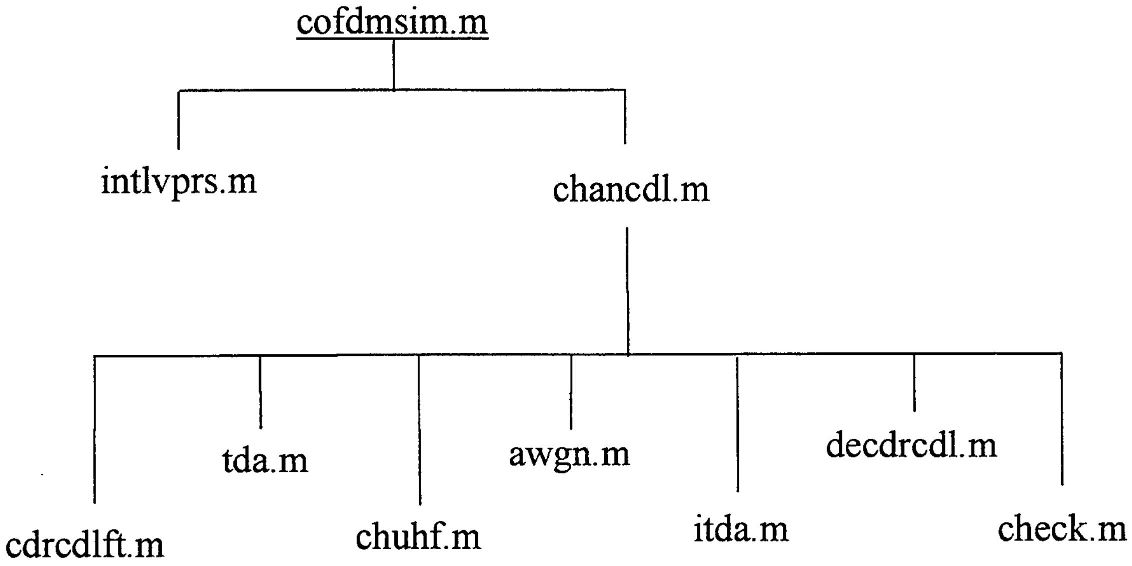
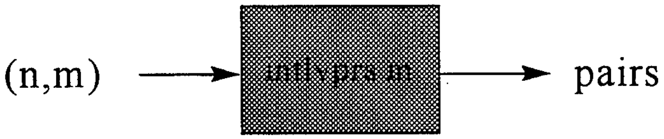


Fig. 38. M-file Hierarchy for Cofdmsim.m

This batch m-file includes two additional sub-routines not previously introduced, specifically **intlvprs.m** and **chancdl.m**. The m-file **chancdl.m** is identical to **chn0cdl.m** with respect to the transmitter and receiver segments; however, **chancdl.m** includes all four channel models (channel 0 through channel 3) with the option of selective configuration by the user. Thus, the m-files **chuhf.m** and **awgn.m** are included with the function. The hierarchy of m-files composing **cofdmsim.m** are shown in Fig. 38.



The m-file **intlvprs.m**, determines all suitable intermediate matrix interleaver dimension pairs as a function of the inputs *n* and *m*, where *n* is the number of rows and *m* is the number of columns representing source message matrix dimensions. Initially, all positive whole number multiples of the product formed by multiplying *n* by *m* are calculated. Recall that the product of the intermediate matrix dimensions must equal the product of the source message matrix dimensions. This m-file is useful for the CDL block interleaving function in order to identify acceptable intermediate matrix dimensions. The output, *pairs*, is an array indicating all possible intermediate matrix dimension pair choices for the given inputs based upon the whole number multiples. The dimension pairs are duplicated by the function in inverse order since intermediate matrix row number and column number dimensions are interchangeable. The following example (Table 18) demonstrates further.

```
» Multiples1 = intlvprs(5,10)

Multiples1 =
    1   50
    2   25
    5   10
   10    5
   25    2
   50    1

» Multiples2 = intlvprs(4,8)

Multiples2 =
    1   32
    2   16
    4    8
    8    4
   16    2
```

```
    32    1

» Multiples3 = intlvprs(10,80)

Multiples3 =
     1  800
     2  400
     4  200
     5  160
     8  100
    10   80
    16   50
    20   40
    25   32
    32   25
    40   20
    50   16
    80   10
   100    8
   160    5
   200    4
   400    2
   800    1
```

Table 18. M-file Intlvprs.m Example

3. Interleaver Case Optimization Batch File

As a means of identifying which interleaver cases promote reduced error concentrations under different system simulation configurations, m-file **chancase.m** is used. This function may be configured to individually simulate channel models 0 through 3 and provide an output plot summarizing the total error distributions per message block row (note that running model 0 is for diagnostic purposes only). Knowledge of maximum error totals within any single row is necessary to determine the minimum required R-S code strength needed to correct all errors. Recall that the R-S error correction function operates on row symbols; thus, a preference for lower overall row error totals is desirable to maximize code rates and minimize coding overhead. Furthermore, multiple simulations of **chancase.m** are repeated with identical system configurations but with different interleaver cases to identify which cases tend to disperse errors more effectively and minimize the total number of errors appearing in any one row of the message block. This batch program uses the **chancdl.m** subroutine m-file. An example of program initialization is provided in Table 19.

```
» chancase   % Simulate channel 2, link 3 for optimal interleaver cases using 240 OFDM tones.
```

```
To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time version: 1
Enter the # of OFDM frequencies (Note: Must be even): 240
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 256
Enter specific integer seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 279
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter 0,1,2
or 3): 2
Channel model 2 simulation performed.
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom):  3
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10080
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10080], or [10080 1], offers no
interleaving functionality): [240 42]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Test all interleaver cases(yes) or specific ones(no)? (1 = yes, 0 = no): 1
All cases, 0 through 8, will be tested.
Enter the guard interval length (Number of sample points): 6
```

```
Do you want pictures? (1 = yes, 0 = no): 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 19. M-file Chancase.m Example

VI. SYSTEM SIMULATION METHODOLOGY AND TEST RESULTS

A. GENERAL TEST PLAN

After construction of the various system models and functional verification of the partially integrated sub-blocks is accomplished, the research progresses to the simulation test phase where complete integrated system simulation trials are performed using different channel models and corresponding performance curves generated. The general system simulation test plan is presented in Fig. 39 along with the associated m-files governing each respective test phase.

As indicated in the figure, there are six independent test phases, advancing in the level of channel difficulty starting from the easiest, channel model 0, to the most challenging and complex, channel model 3. Through the collection and evaluation of simulation data, the hierarchical test approach from simple to complex aids in isolating poor system performance during initial simulation stages and also allows for careful study and evaluation of each channel model output individually. Consequently, early evidence of inadequate performance due to missing or poorly functioning sub-blocks can help the designer deduce the necessary corrective measures needed in the form of system redesign and/or reconfiguration to finalize an optimal system. By conducting multiple system simulations and evaluating output data, it is possible to perform iterative configuration adjustments on the model to strengthen and improve the system for added robustness and optimal system model 3 performance. The eventual goal is to optimize the system and generate performance results using the final channel 3 model that demonstrate a satisfactorily working COFDM based communication system compatible with BAA specified performance and throughput objectives.

For most simulation trials, a total quantity of 10,000 + 1% symbols representing the source message are simulated through the various channel models, with the exact quantity depending on the configured number of OFDM frequency tones as well as interleaver intermediate matrix dimensions. The 10,000 symbols, typically 8-bits long, represent OFDM symbols and are configurable to any bit length. However, the number of transmitted symbols, referred to as PSK symbols, may increase or decrease as a result of symbol reformatting and the conversion from M-ary symbols to N-ary symbols. Since most simulations involve 16-PSK type modulation, the N-ary symbol word size typically reduces to 4-bits from a M-ary symbol word size of 8-bits. As a result of M-ary to N-ary conversion from 8-bits to 4-bits, the transmitted source message array doubles in the number of PSK symbols to 20,000 + 1%. Thus, roughly 20,000 PSK 4-bit symbols are simulated through the various channels.

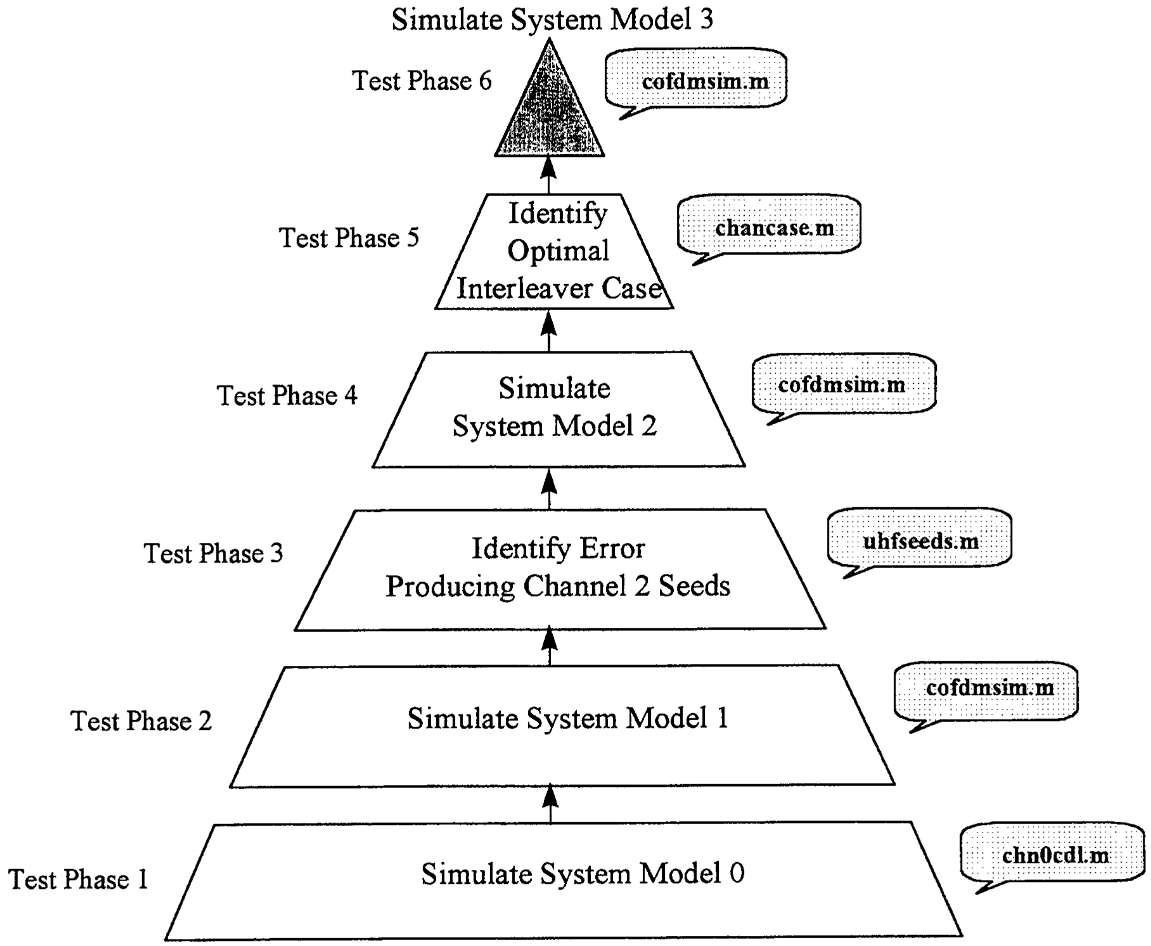


Fig. 39. Simulation Test Plan Hierarchy

The OFDM symbol quantity comprising the source message array is selected for practical software simulation considerations as well as for minimizing interleaving latency delays within the system. For example, if a configuration of 10,080 OFDM symbols are simulated through the model using 240 OFDM frequency tones with a symbol interval length of 500μsec, then the total symbol message block interleaving processing latency is 21msec as calculated below:

$$T_{Latency} = 500\mu\sec\times\frac{10{,}080 symbols}{240} = 21m\sec. \qquad (38)$$

Delays beyond this are considered unacceptable for full duplex system communications; hence, the arrival at a 10,000 +1% OFDM symbol limit per interleaver block.

Furthermore, there is a practical limit as to how many symbols can be efficiently processed by the PC microprocessor during simulations. Even though the chosen PC platform for system emulation and simulation is presently state of the art, multiple simulations using source message block sizes much in excess of 10,000 OFDM symbols tend to noticeably slow processing times and promote hard drive thrashing. Hard drive thrashing occurs during processor calculations as data values are continuously swapped out of main memory to secondary storage and vice versa. Coincidentally, the selection of a maximum of 10,000 +1% OFDM symbols in consideration of minimal system transmission latency also works well with the PC hardware configuration for simulation purposes.

B. TEST PHASE 1 - SYSTEM MODEL 0 SIMULATIONS

With reference to the test plan hierarchy, initially system model 0 simulations are performed to verify proper integration of all system sub-blocks and to ensure a correctly working overall model. Recall that the COFDM model 0 system incorporates the channel 0 model, representing a perfect noise free channel without AWGN and multipath distortions (Fig. 21). Hence, this model can be viewed simply as the OFDM transmitter output connected directly to the OFDM receiver input with no intervening channel block. With the prior assumption that the transmitter and receiver are functioning correctly according to design, then the source and sink message blocks should have identical content without symbol errors since there can not be any channel noise influences corrupting the signal. Consequently, any symbol error occurrences in the sink message must be the result of an incorrectly implemented m-file program model. For instance, if the deinterleaving sub-block did not perform functionally correct in the receiver, and numerous symbol errors resulted as detected by m-file **check.m** verification, then system debug would follow and corrective measures taken to provide necessary functional sub-block repairs before proceeding to the next test phase.

With this in mind, numerous system model 0 simulations were repeatedly conducted using m-file **chn0cdl.m** with various input configurations, and the resulting data collected and evaluated (it is also possible to perform the identical system verification test using batch m-file **cofdmsim.m** configured for a model 0 simulation). After initial simulation failures and subsequent system corrective debug, final test results indicated that the transmitter and receiver functional blocks were indeed constructed properly and functioning accurately since no resulting symbol errors were identified in the receiver after repeated runs. A table of sample results reflecting model 0 system simulations with various input configurations is presented in Table 20.

```
» chn0cdl(0,0,0,1,1,4,4,6,8,4,4,8,8,8,6,0);  % Run a model 0 functional check with 4 OFDM
frequencies and 6 rows.  The number of FFT points is 8, while the M-ary number is 16.  No FEC
is used; however, time differential encoding is included with a 6 sample guard interval.

Random_Source_Msg =
    1    2    0    6
   14    0    2    5
    6   13    9    7
```

```
  10   5   6   4
   1   1  15   4
  15  10   4   2

Sink_msg =
   1   2   0   6
  14   0   2   5
   6  13   9   7
  10   5   6   4
   1   1  15   4
  15  10   4   2

GREAT!!! there are no errors.
Test Passed!!!

» chn0cdl(0,0,0,1,1,10,8,5,16,4,4,8,8,8,8,1);  % Run a model 0 functional check with 10 OFDM
frequencies and 4 rows.  The # of FFT points is 16, while the M-ary number is 16.  No FEC is
used; however, frequency differential encoding is included with a 8 sample guard interval.

Random_Source_Msg =
   1   2   0   6  14   0   2   5   6  13
   9   7  10   5   6   4   1   1  15   4
  15  10   4   2   4  13   2   5   8  11
   4  10   4  14  14   6   3   4   9   2

Sink_msg =
   1   2   0   6  14   0   2   5   6  13
   9   7  10   5   6   4   1   1  15   4
  15  10   4   2   4  13   2   5   8  11
   4  10   4  14  14   6   3   4   9   2

GREAT!!! there are no errors.
Test Passed!!!
```

Table 20. System Model 0 Verification Example

With the conclusion of transmitter and receiver functional verification, the remaining system test simulations include channel noise and multipath and are oriented around the channel 1, channel 2 and channel 3 models. Recall from Fig. 21 that the channel 1 model represents AWGN only, while the channel 2 model includes multipath exclusively, and the channel 3, includes both the multipath channel 2 model and the AWGN channel 1 model. The channel 3 model is the most difficult error producing channel model since it adds AWGN to the signal distortions induced by the multipath channel. However, channel 3 emulates actual maritime transmission environmental phenomenon; hence, the channel model 3 is most indicative of the types of channel influences that will affect real-time RF communication during transmissions by the proposed COFDM modem.

C. TEST PHASE 2 - SYSTEM MODEL 1 SIMULATIONS

Test phase 2 performs channel 1 model simulations exclusively (AWGN channel) and compares the trial results to theoretical performance values. Recall that AWGN is emulated in MATLAB using m-file **awgn.m** and is part of the COFDM model 1 system. Fig. 31 depicts the complete model 1 system consisting of the OFDM transmitter and OFDM receiver interfaced to the AWGN channel 1 block. During this test phase, batch m-file **cofdmsim.m** is configured for system model 1 simulations and used to generate numerous trial data. The data results are presented graphically in the form of performance curves representing symbol error rates (SER) versus the ratio of symbol energy to noise power ($^{Es}/_{No}$). Simulation data are compared to theoretical AWGN performance curves with similar system configurations. Evaluations of the results are conducted to measure the integrity of the system in the presence of AWGN.

The theoretical performance plots for differentially encoded coherent M-PSK are depicted in Fig. 40 and are based on

$$P_E(M) \cong 2Q\left(\sqrt{\frac{2E_S}{N_O}}\sin\frac{\pi}{\sqrt{2}M}\right) \qquad (39)$$

where $P_E(M)$ is the SER and $M = 2^q$. In Fig. 40, curves are generated for M = 4 (q = 2, QPSK), M = 8 (q = 3) and M = 16 (q = 4) [13; p. 177]. These curves represent the performance approximation reference baseline to which all subsequent system simulation trial data will be compared.

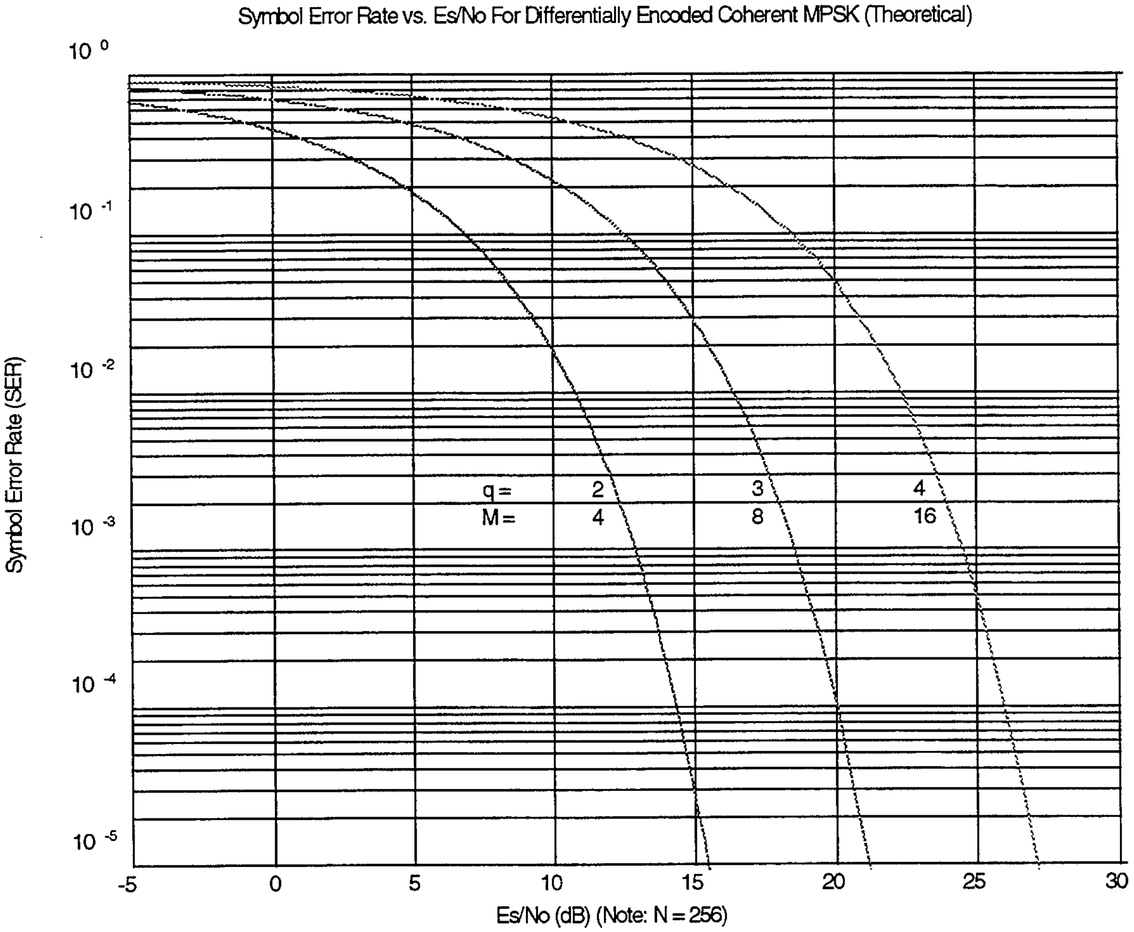


Fig. 40. Theoretical Performance Graph Showing SER Verse $^{Es}/_{No}$

```
» cofdmsim

This batch m-file runs COFDM simulations using different channel models.
To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or to run both enter 2
(two): 1
Enter the # of OFDM frequencies (Note: Must be even): 240
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 256
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter 0,1,2
or 3): 1
Channel model 1 simulation performed.
Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003):
[linspace(0,0.018,20), linspace(0.018,0.02,20)]
Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): 0
Enter specific case numbers from (0 to 8) (Ex. [0 4 5 8]): 0
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10080
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10080], or [10080 1], offers no
interleaving functionality): [240 42]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Enter the guard interval length (Number of sample points): 6
Do you want to include error correction coding? (1 = yes, 0 = no): 0
Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 0
Do you want signal plots? (1 = yes, 0 = no): 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 21. System Model 1 Simulation Example Using Cofdmsim.m

As previously mentioned, the magnitudes of each randomly generated message symbol and corresponding complex modulation value are fixed at unity and represent the signal energies. However, the noise power, N_O, is variable and configurable by the user. Consequently, during simulation configurations, judicious selection of noise powers by setting suitable noise variance ranges (*sigma* parameter) promotes the generation of meaningful performance plots and allows for comparisons among various trial configurations. Table 21 presents a portion of a **cofdmsim.m** simulation configured for system model 1 (AWGN channel) using 240 OFDM frequency tones and frequency differential encoding, while Fig. 41 depicts the corresponding performance plot associated with the configured inputs. A total of 10,080 8-bit OFDM symbols are randomly generated and represent the source message block during these simulations using 240 OFDM tones. Since 16-PSK is the configured modulation scheme for this trial, a total of 20,160 4-bit PSK symbols are transmitted through the channel (double the OFDM symbol message block size).

Comparison of the simulated SER performance graph in Fig. 41 to the corresponding 16-PSK theoretical graph shown in Fig. 40 indicates a system performance result approximately 1 dB worse than theoretical AWGN. For example, from Fig. 41 using 16-PSK, a simulated SER of 10^{-3} occurs when the $^{Es}/_{No}$ is approximately 25.25 dB. Comparing this to Fig. 40, a theoretical SER of 10^{-3} corresponds to an $^{Es}/_{No}$ of roughly 24.5 dB. The dissimilarity of the simulated system model 1 result and estimated theoretical performance is approximately 0.75 dB, or approximately 3.1% error from theoretical. Although the difference between the simulated performance curve and theoretical is within 1 dB, it is desirable to investigate why there is a loss of precision.

In recognition of (39) being an approximation and after careful review of system construction, it is apparent that the 0.75 dB discrepancy is largely a result of the R-S FEC sub-block location in the receiver signal path and the manner in which R-S FEC functionality is performed. Recall that source message is composed of 10,080 OFDM symbols configured as eight bits words. Before transmission, the symbols are converted into 4-bit PSK symbol words. This format conversion causes the message block symbol number to double to 20,160. Upon reception of the transmitted message, the symbols are reformatted from 4-bit PSK symbols back into 8-bit OFDM symbols. R-S FEC operates on the 8-bit OFDM symbols when the source and sink messages miscompare.

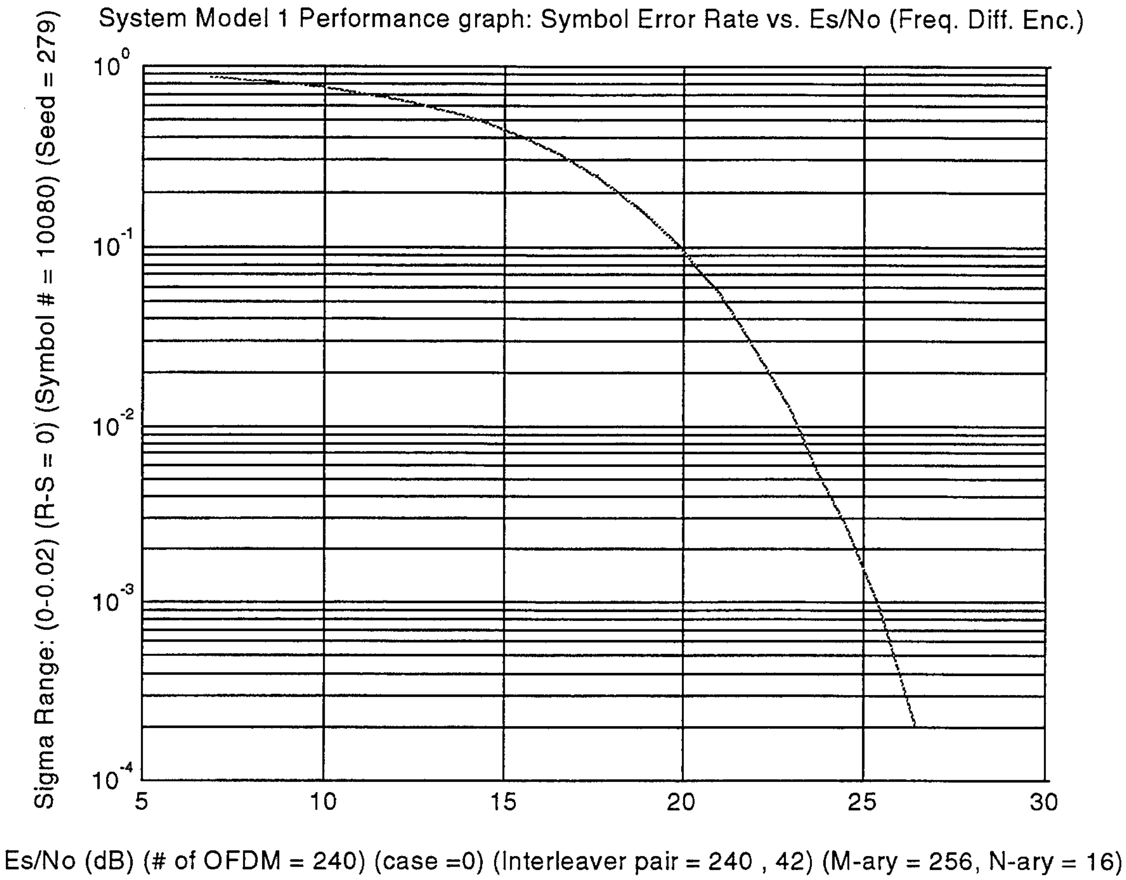


Fig. 41. System Model 1 Simulation Performance Graph Showing SER vs. $^{Es}/_{No}$ (Frequency Differential Encoding and 240 Tones)

Since channel noise can affect 4-bit PSK symbols, it is possible for two PSK symbols, one of which is in error and one which is correct to be mapped into a single 8-bit OFDM symbol, appearing to the R-S decoder as a single 8-bit symbol error. From this discussion it is apparent that the mapping of symbols during N-ary to M-ary format conversion may extend the symbol error length across a single PSK symbol boundary into an adjacent correct PSK symbol. The consequence is OFDM symbol performance calculations that appear slightly worse than expected.

At first a possible solution could be to perform the R-S FEC operation earlier in the receiver signal processing sequence such as before the symbol reformatter. However, the system design requires that the interleaving function be performed using 8-bit symbols before R-S FEC can be accomplished. The alternative is to perform 4-bit PSK symbol interleaving and deinterleaving, then perform 4-bit R-S symbol error correction afterwards. This approach was initially considered but deemed undesirable as it requires additional interleaver matrix manipulations using larger matrix dimensions causing increased latency and added complexity. Additionally, performing R-S FEC using 4-bit symbols instead of 8-bit symbols reduces maximum possible code block lengths from 255 symbols down to 15 symbols [12; p. 171]. This is undesirable for an effective COFDM design since code block lengths consisting of 15 symbols would no longer include all OFDM sub-carriers along one row of message symbols and would not take full advantage of the frequency diversity property and combined FEC offered by 240 OFDM tones. Thus, a system design using 8-bit OFDM symbol interleaving and deinterleaving is the preferred choice.

In terms of channel induced error pattern scenarios, the best case condition exists when PSK symbol errors are adjacent to each other and paired together. A worst case condition exists when PSK symbol errors are paired with correct symbols and mapped into OFDM symbols. The lower bound on this symbol mapping error phenomenon is zero, when no channel errors are generated and are absent from the sink message. The upper bound is two, when every other 4-bit PSK symbol is in error, half of the entire sink message, causing all 8-bit OFDM symbols or the entire message block to appear in error. Thus, the corresponding performance curve would indicate double the actual number of PSK symbols to be in error.

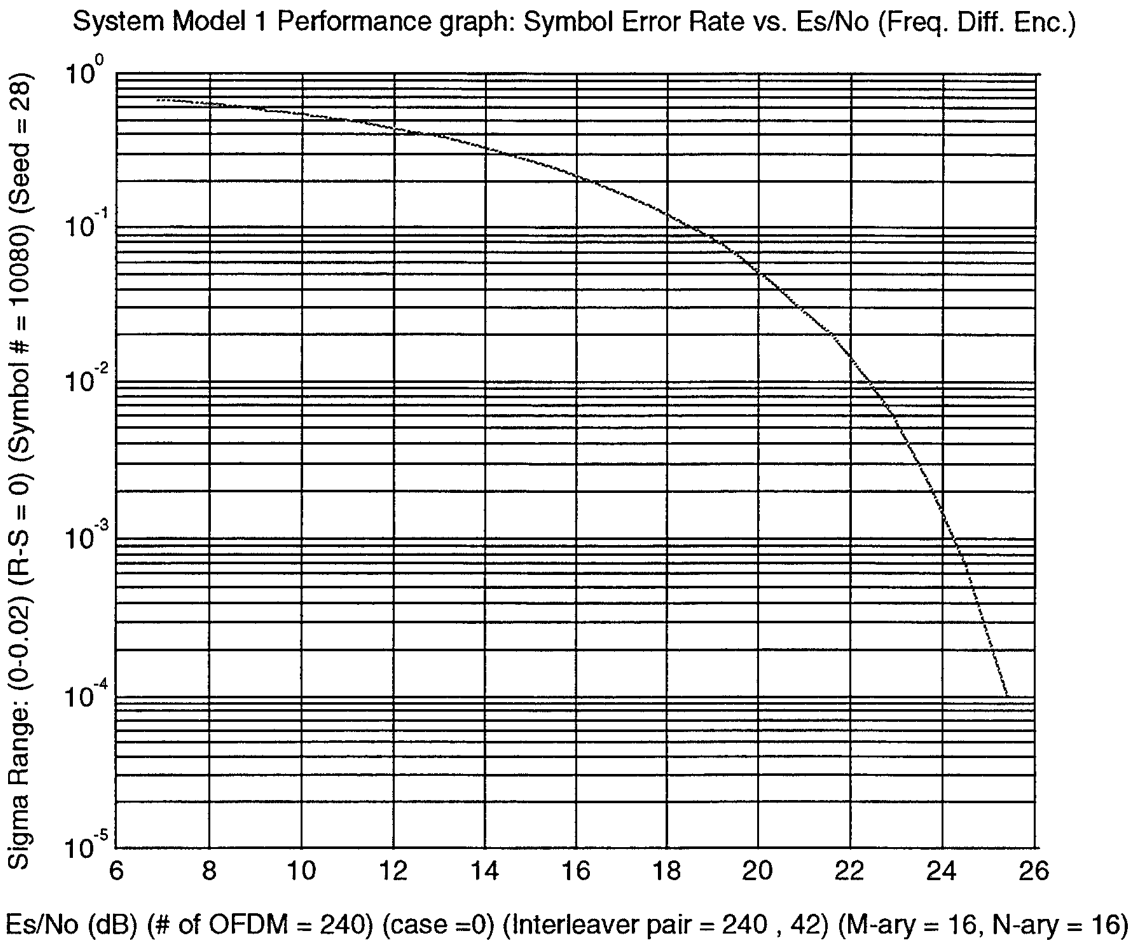


Fig. 42. System Model 1 Simulation Performance Graph Showing SER vs. $^{Es}/_{No}$ Using OFDM Symbol Size Equal to PSK Symbol Size (4-bits) (240 Tones)

To verify this hypothesis accounting for the simulation performance curve discrepancy from theoretical, a simulation trial using a 4-bit OFDM symbol format and 4-bit PSK symbol format is included with observation of the output SER and comparison to theoretical. From Fig. 42, the simulation output curve using 240 tones and frequency differential encoding/decoding is nearly identical to the theoretical performance curves of Fig. 40 for 16-PSK. This result supports the aforementioned discrepancy hypothesis, as there is negligible difference between the approximated theoretical AWGN and simulated curves. Hence, a correctly functioning system model in the presence of AWGN is confirmed. Through this experiment, it is also recognized that the curves depicted in Fig. 40 are based upon the approximation given by (39) and, hence, are not exact.

The implementation errors induced by the system design and appearing in the performance curves permeate all subsequent simulation trials including system models 1 through 3. While no exact error offset compensation applied to simulation outputs to counteract the implementation error is possible, it is helpful to be aware of the error bounds inherent in the simulation performance curves to help gauge the merit of the results.

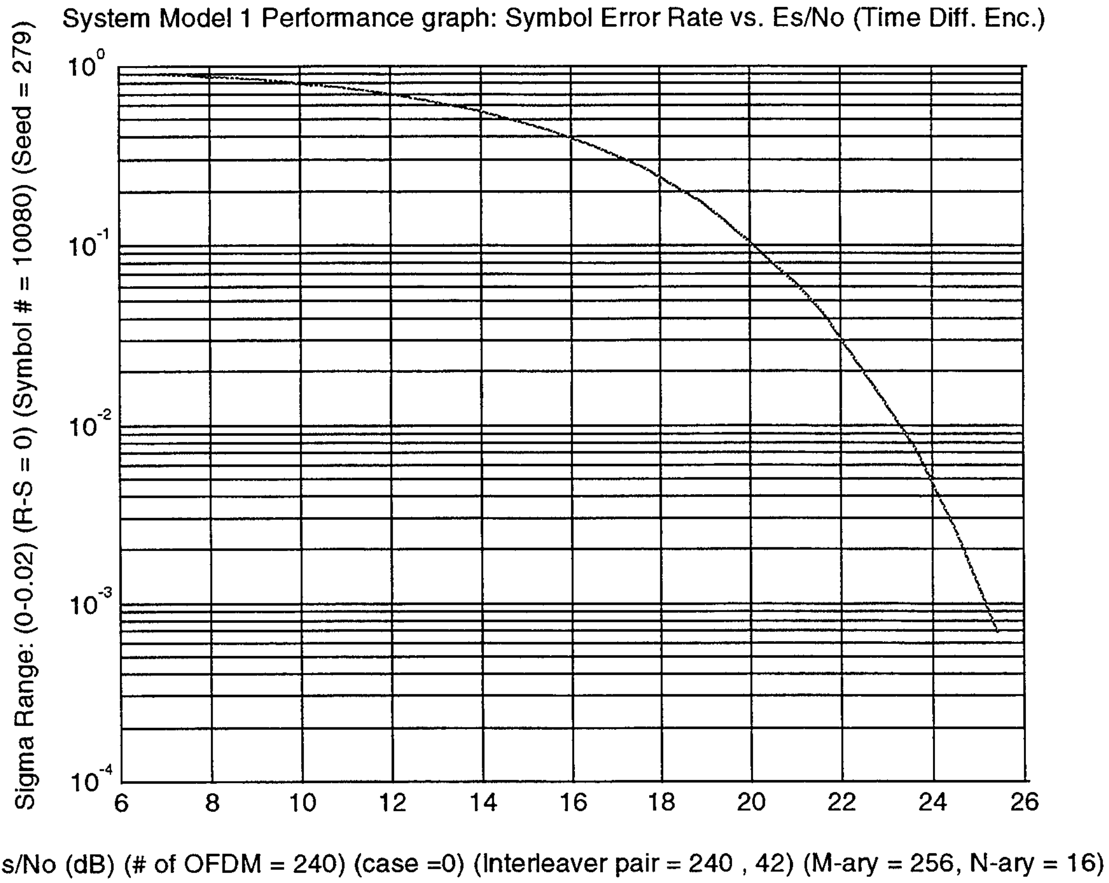


Fig. 43. System Model 1 Simulation Performance Graph Showing SER vs. $^{Es}/_{No}$ (Time Differential Encoding and 240 Tones)

Numerous simulations configured with 240 OFDM frequency tones, 8-bit OFDM symbols and different noise parameters were conducted, each demonstrating similar acceptable performance results as indicated earlier. For example, Fig. 43 displays performance results for the same system model 1 simulation with identical Fig. 41 configurations; however, time differential encoding is included instead of frequency differential encoding. The performance results between Fig. 43 and Fig. 41 at a SER of 10^{-2} are similar, within 0.5 dB, and correspond to the theoretical curve in Fig. 40 within 0.5 dB. Multiple system model 1 simulations consistently verified performance results similar to the estimated theoretical within the 1 dB error tolerance. Consequently, phase one testing of the AWGN channel one model concluded successfully.

As an added caveat, it is apparent at least from phase 1 test results that system model 1 simulations using 4-bit OFDM and 4-bit PSK symbols yield results close to theoretical AWGN. Consequently, in the conclusions, discussion and complete system model 3 simulations oriented around 4-bit OFDM symbols and 4-bit PSK symbols are included for comparison purposes to the system baseline.

D. TEST PHASE 3 - IDENTIFY ERROR PRODUCING CHANNEL 2 SEEDS

With the successful completion of system model 1 simulations and verification of system performance within an acceptable range of theoretical AWGN, the test plan progresses to phase three. During phase three testing, channel 2 simulations are performed without interleaving, and a statistical record of resulting error totals are compiled using various system seed configurations. This test phase makes use of batch m-file **uhfseeds.m** to perform numerous COFDM system simulations using the channel 2 model (**chuhf.m**) with various input seed configurations. This test step is performed to excite the channel and record the total errors appearing in the sink message as caused by multipath burst noise distorting the transmitted signal. Since the multipath channel randomly distorts the transmitted signal, the simulations behave differently with different seed configurations. Thus, certain seeds affect transmitted symbols differently by generating more message errors than other seeds. Consequently, by identifying the worst case channel conditions as a function of the seed parameter, these "bad" seeds can be included during subsequent system model 3 testing, ensuring that a worst case channel model is created during the complete system model simulation. The performance results derived from system model 3 simulations using "bad" seeds will represent a COFDM system operating in a worst case multipath channel environment under extreme conditions.

The range of seed values available for simulations are elements of the positive integer set; hence, there are an infinite number of possible seeds. Obviously it is impossible to simulate all conceivable seeds during test phase 3. However, a suitable subset of all possible seeds with behavioral characteristics indicative of the infinite set are adequate to convey general statistical error distribution information. It is desirable to select 500 different integer seeds for test using system model 2 and be confident within an acceptable percentage that the tested seed set represents 99% or more of all possible seeds generating z or fewer errors. With this goal in mind, application of the law of large numbers and the Chebychev inequality to ascertain a confidence parameter are included in the estimation. [16; pp. 107-108] The following events and probabilities are defined next.

Event A_z: Equals the event that z symbol errors occur in N symbols, where $\Pr(A_z) = P_{Az}$.
Event B: Equals the event that z, or z+1, or z+2 or ... N symbol errors occur in N symbols, or in other words, there are z or more errors in N symbols. Hence,

$$B = A_z \bigcup A_{z+1} \bigcup A_{z+2} \bigcup \ldots \bigcup A_N \text{ ; and,} \qquad (40)$$

$$P_B = \Pr(B) = P_z + P_{z+1} + P_{z+2} + \ldots + P_N = \sum_{i=z}^{N} P_i \, . \qquad (41)$$

Furthermore, let the estimate of the relative frequency of event B, $\hat{P}_B$, be a lattice random variable and equal to

$$\hat{P}_B = \frac{k_z + k_{z+1} + \ldots + k_N}{n} = \frac{1}{n}\sum_{i=z}^{N} k_i \text{, where } 0 \le \hat{P}_B \le 1 \text{ and} \tag{42}$$

$k_z + k_{z+1} + \ldots + k_N$ is the summation of the number of times z, or z+1, or z+2 or … N errors occur in n trials. Now let $k_z + k_{z+1} + \ldots + k_N = \alpha n$ which implies that $\hat{P}_B = \alpha$, or in other words, let the estimate of the relative frequency of z or more error events occurring in N symbols be α.

Event C: Equals the event that there are 0, 1, 2, … , z-1 errors in N symbols; or in other words, there are fewer than z errors in N symbols. Thus, C = B^c, or

$$P_C = \Pr(C) = P_0 + P_1 + P_2 + \ldots + P_{z-1}. \tag{43}$$

Let the estimate of the relative frequency of event C, $\hat{P}_C$, be a lattice random variable equal to,

$$\hat{P}_C = \frac{k_0 + k_1 + \ldots + k_{z-1}}{n} = \frac{n - (k_z + k_{z+1} + \ldots + k_N)}{n} \text{, where } 0 \le \hat{P}_C \le 1. \tag{44}$$

Consequently, $\hat{P}_C = 1 - \hat{P}_B = 1 - \alpha$, or in other words, the relative frequency estimate of fewer than z error events occurring in N symbols.

Now let z be equal to the largest error total corresponding to the worst case seed out of the 500 simulated seeds. Then $\hat{P}_C$ is the relative frequency estimate after n trials that the channel will generate z-1 or fewer errors. Making use of the Chebyshev inequality and the *law of large numbers* [16], then

$$\Pr(\left|\hat{Pc}-Pc\right| \geq \varepsilon) \leq \frac{Pc(1-Pc)}{n\varepsilon^2} \qquad (45)$$

which is an upper bound on the chance that $\hat{P}_C$ and P_C differ by more than ε after n trials. Obviously P_C is an unknown quantity since it represents the actual occurrence of fewer than z errors for all possible seeds existing in the universe. However, the estimate $\hat{P}_C$ can be determined as a function of the total number of tested seed trials whose error totals are less than z errors.

For this experiment, only the worst case seed out of 500 is considered and represents the greatest error total, z; therefore 499 out of 500 tested seeds produce error totals less than z. Consequently from (44),

$$\hat{P}_C = \frac{499}{500} = 0.998\,. \qquad (46)$$

By letting $\varepsilon = 0.008$, we can use (45) to compute the probability that P_C is between 0.99 and 1.00. That is,

$$\Pr(\left|\hat{Pc}-Pc\right| \geq 0.008) \leq \frac{(0.002)(0.998)}{500(0.008)^2} = 6.23\%\,, \qquad (47)$$

where we have used $\hat{P}_C$ on the right side of (45) since P_C is unknown. This result indicates a 93.76% confidence that the worst case seed in 500 seeds is in a subset of 1% of all seeds that generate z or more errors.

In continuation of the original system configuration of 240 OFDM tones and 16-PSK used during test phase 2, **uhfseeds.m** simulation trials are performed using 10,080 OFDM symbols with both time and frequency differential encoding/decoding. The selected integer seed range used for system configuration and evaluation are from 1 to 500. Also, link 3 is initially included during trials as it represents the most challenging multipath channel with relatively severe power fading, path delays and Doppler shifting typical of maritime ship-to-relay link communications.

An initial simulation using time differential encoding is performed with the corresponding seed error report presented in Fig. 44, and the corresponding error distribution displayed in Fig. 45. The seed error report provides simulation information regarding the total number of message error occurrences as a function of a specific seed value. The error distribution plot orders the seeds from least to greatest with respect to the number of errors generated in each corresponding message block. Included in the margin within the error distribution figures are the top 5 error producing seeds (Top 1% of entire tested seed set). Fig. 46 displays the error histogram for 240 tones using ten error bins and reveals that the average number of errors per seed for this configuration is 405. From this result it is apparent that out of 10080 OFDM message symbols transmitted, 405 on average would be in error at the receiver, approximately a 4% average symbol error rate.

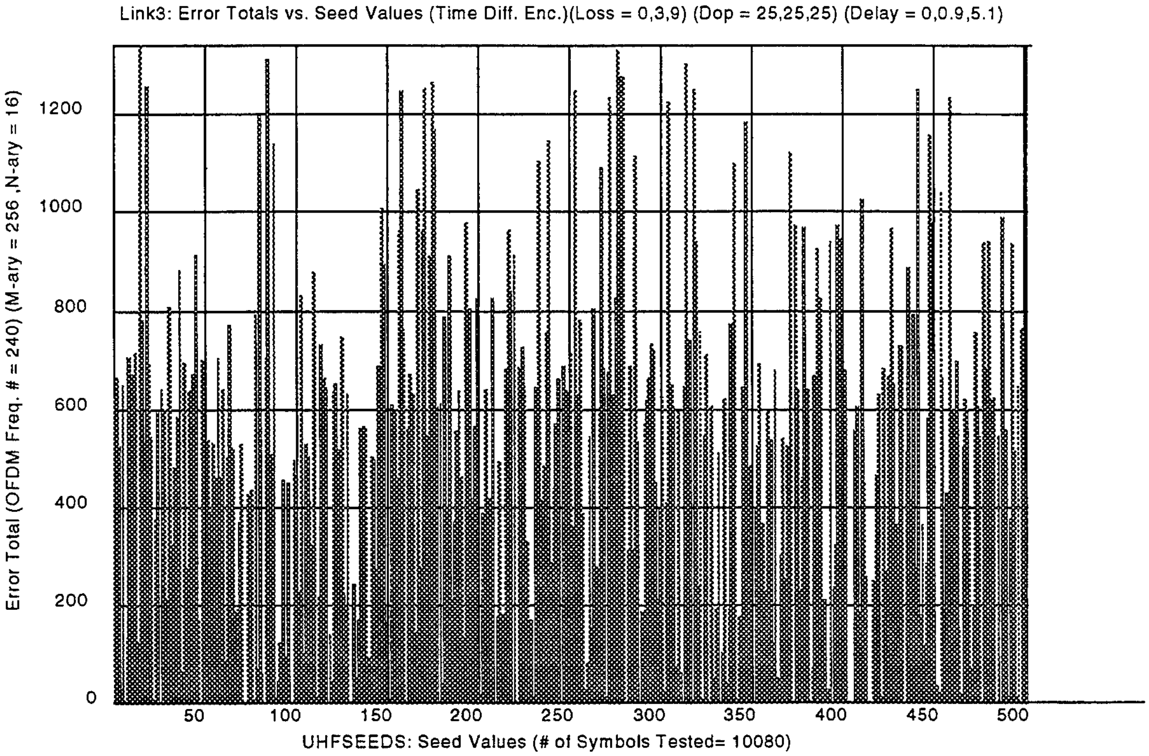


Fig. 44. Link 3 Seed Error Report (240 Tones, Time Differential Encoding)

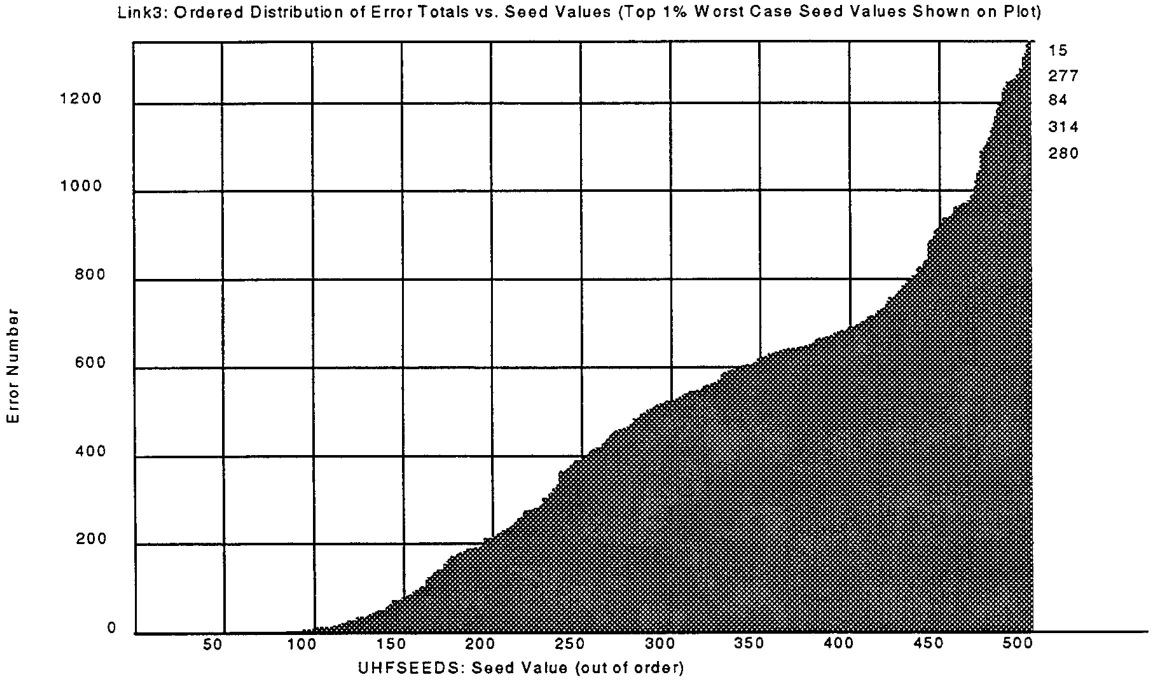


Fig. 45. Ordered Distribution of Error Totals Verse Seed Values, 240 Tones

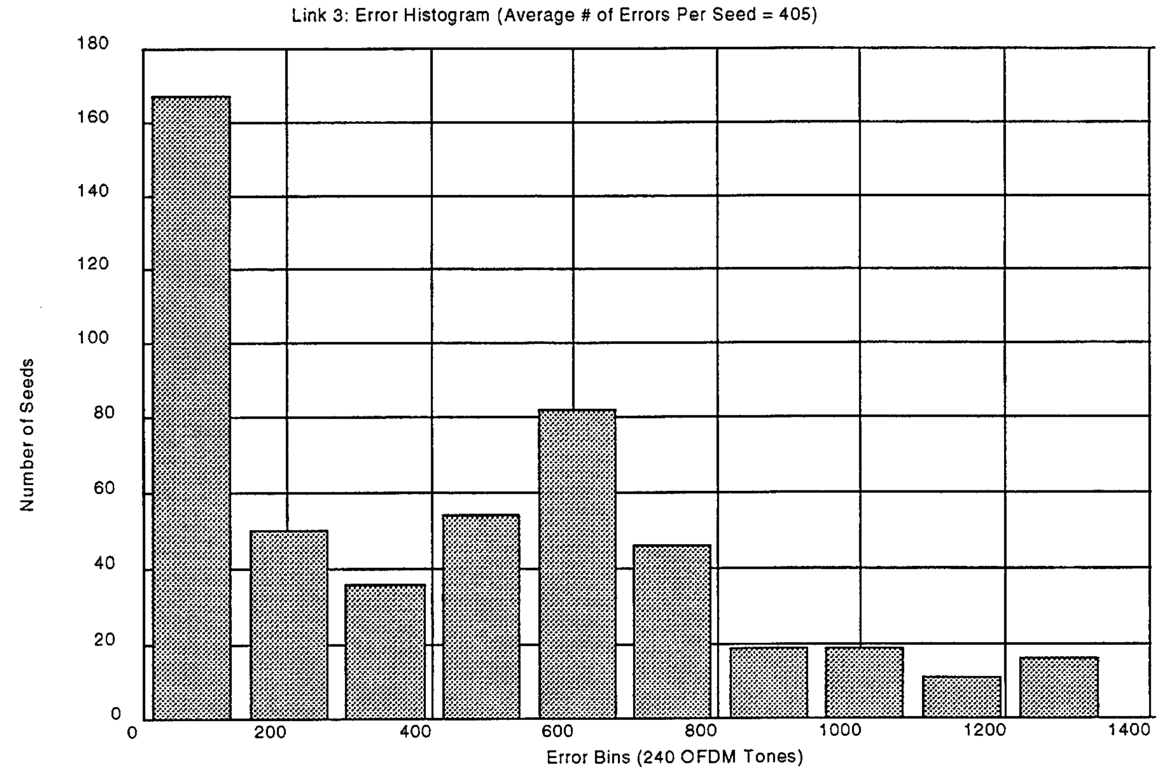


Fig. 46. Error Histogram for 240 Tones and Time Differential Encoding

A second **uhfseeds.m** simulation trial is repeated using 240 OFDM frequencies, 16-PSK and a 10,080 OFDM symbol message test pattern. However, in this case frequency differential encoding/decoding is included instead of time differential encoding. The resulting error report is presented in Fig. 47, and the corresponding error distribution displayed in Fig. 48. From the error histogram shown in Fig. 49, the average number of errors per seed is 35, or approximately a 0.34% average symbol error rate. In contrast to Fig. 46, there is a dramatic reduction of average errors when using time differential encoding over frequency differential encoding. In addition, the worst seed, value 279, generates a total of only 404 errors as opposed to the previous simulation's worst seed, 15, which produces 1341 total errors. Apparently frequency differential encoding combined with a 240 OFDM tone configuration and 16-PSK results in better overall system performance with a minimum average error total. Consequently, this optimal configuration will be utilized for remaining system test phases using 240 OFDM tones.

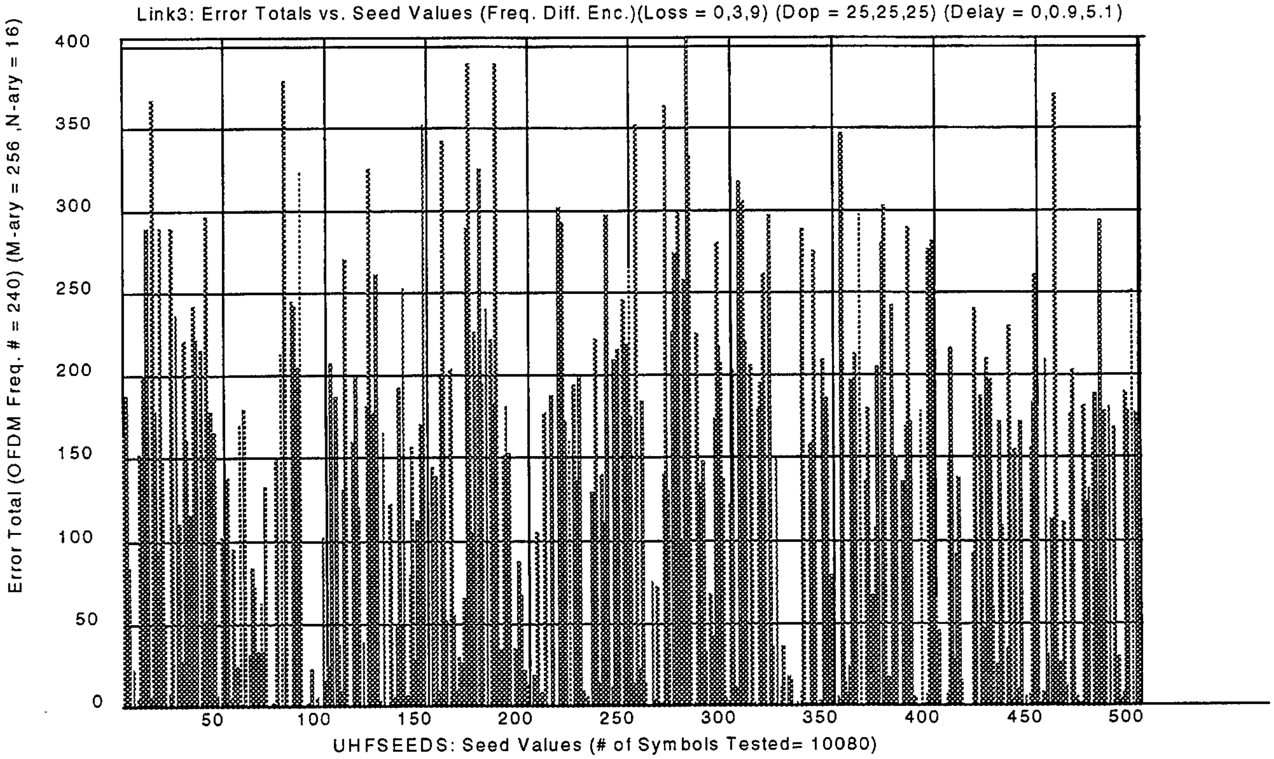


Fig. 47. Link 3 Seed Error Report (240 Tones, Frequency Differential Encoding)

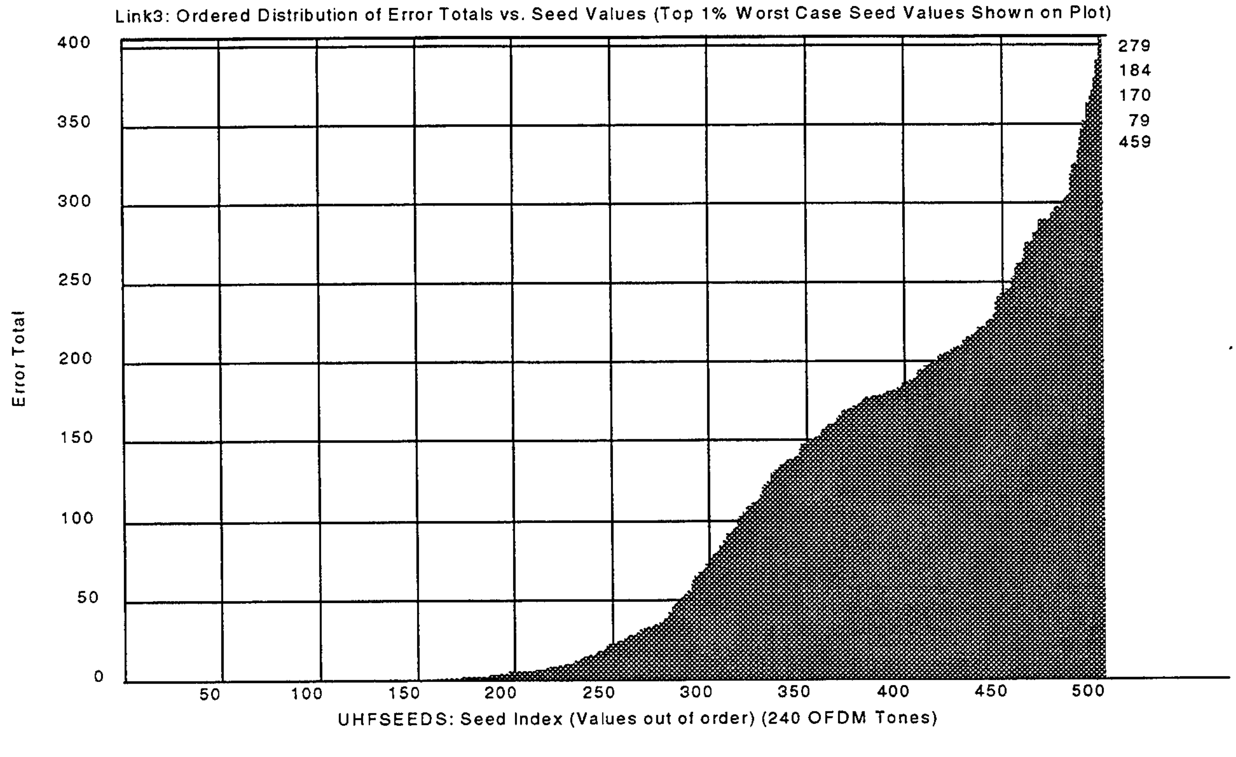


Fig. 48. Ordered Distribution of Error Totals Verse Seed Values, 240 Tones

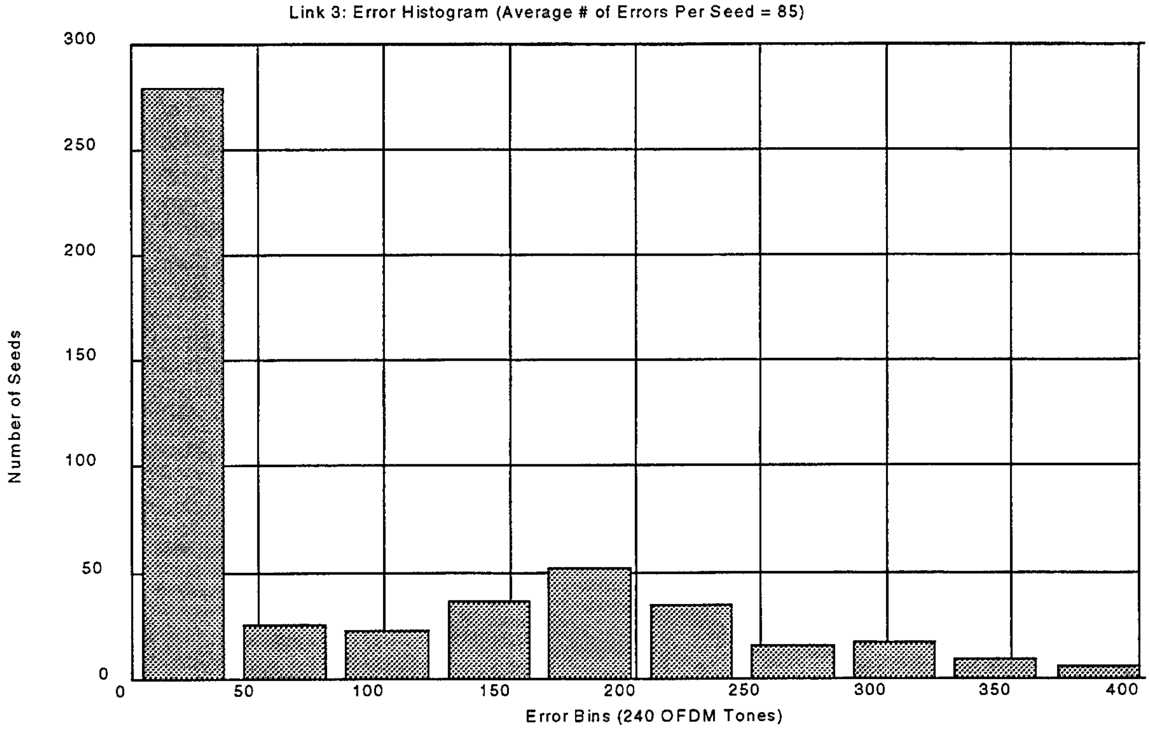


Fig. 49. Error Histogram for 240 Tones and Frequency Differential Encoding

The results of these initial phase 3 test trials indicate large variations in average error totals among system configurations using time verse frequency differential encoding/decoding. From these observations, it is deemed worthwhile to modify the initial system configuration from 240 OFDM tones to a system using other tone quantities and investigate the corresponding performance results produced by batch m-file **uhfseeds.m** using the two differential encoding methods. Phase 3 simulations are repeated with the original system configuration extended to include 30, 60 120 and 480 OFDM frequency tones using both time and frequency differential encoding/decoding. After simulation data is collected and evaluation of resulting error totals are performed, a determination is made as to which OFDM tone number combined with the corresponding system configuration is optimal in terms of producing minimal average errors per seed.

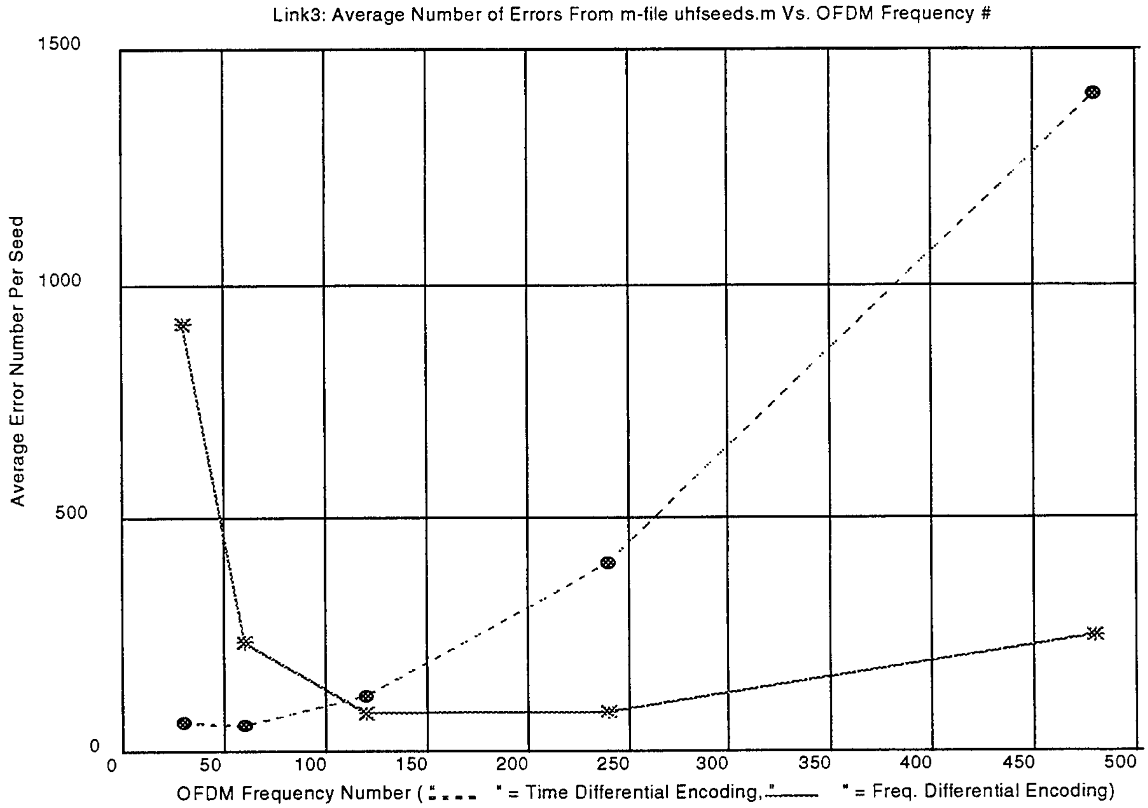


Fig. 50. Average Error Totals Vs. Number of OFDM Tones for Frequency and Time Differential Encoding

Numerous simulations were performed generating error reports, error distribution plots and histograms with similar formats as previously presented for 240 tones above. Fig. 50 displays a comprehensive plot of the average error totals for the OFDM tones of interest. It is readily apparent from the figure that in general time differential encoding/decoding performs better with the system configured for fewer OFDM frequency tones; conversely, a configuration using more OFDM frequency tones typically performs better using frequency differential encoding/decoding. However, in observation of the optimal tone quantities and their corresponding differential encoding/decoding methods, both curves tend to exhibit upswings near the endpoints. Apparently, 60 OFDM tones and time differential encoding/decoding performs slightly better than 30 OFDM tones with the same encoding/decoding method. Similarly, 240 OFDM tones and frequency differential encoding performs slightly better than 480 OFDM tones. Also, 120 OFDM tones performs nearly identical for both time and frequency differential encoding/decoding and only slightly better than 240 OFDM tones using frequency differential encoding, suggesting a "cross-over" point.

Evaluation of Fig. 50 concludes that 60 OFDM tones generates a minimum average error total using time differential encoding and, therefore, will also be included in subsequent system simulations. Based upon these simulation results, the COFDM system design initially configured for 240 tones using frequency differential encoding/decoding is expanded to also include 60 OFDM tones using time differential encoding/decoding and test phase 3 seed error reports are repeated.

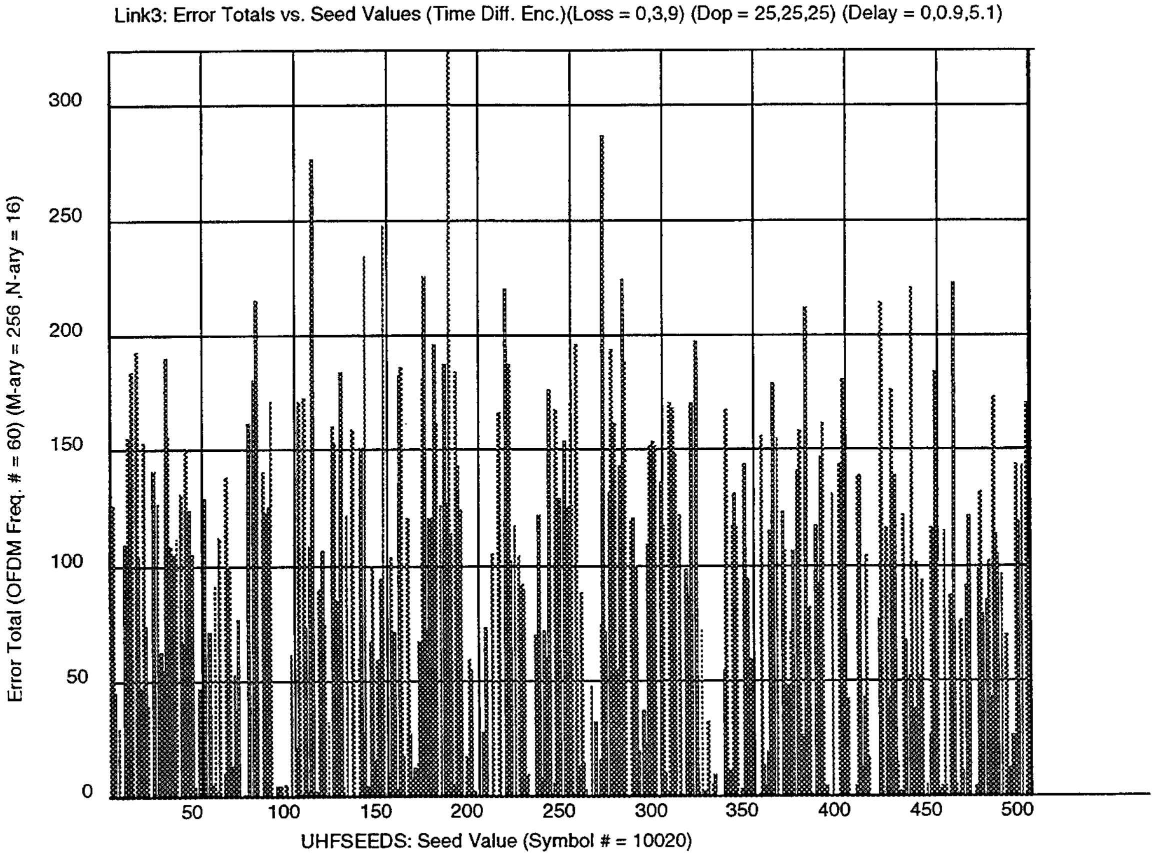


Fig. 51. Link 3 Seed Error Report (60 Tones, Time Differential Encoding)

Using 60 OFDM tones in the system, the sub-carrier (tone) spacing within the 480 kHz channel bandwidth calculates to be 8 kHz (480KHz/60 = 8KHz). Correspondingly the information symbol length is found to be 125μsec (1/8 kHz = 125μsec). Using a 2% guard interval, the complete symbol length, T_{total}, becomes 127.5μsec. Thus, the system configured with 60 OFDM tones uses shorter total symbol lengths than for 240 OFDM tones; however, sub-carrier spacing with 60 tones is 4 times larger than with 240 tones.

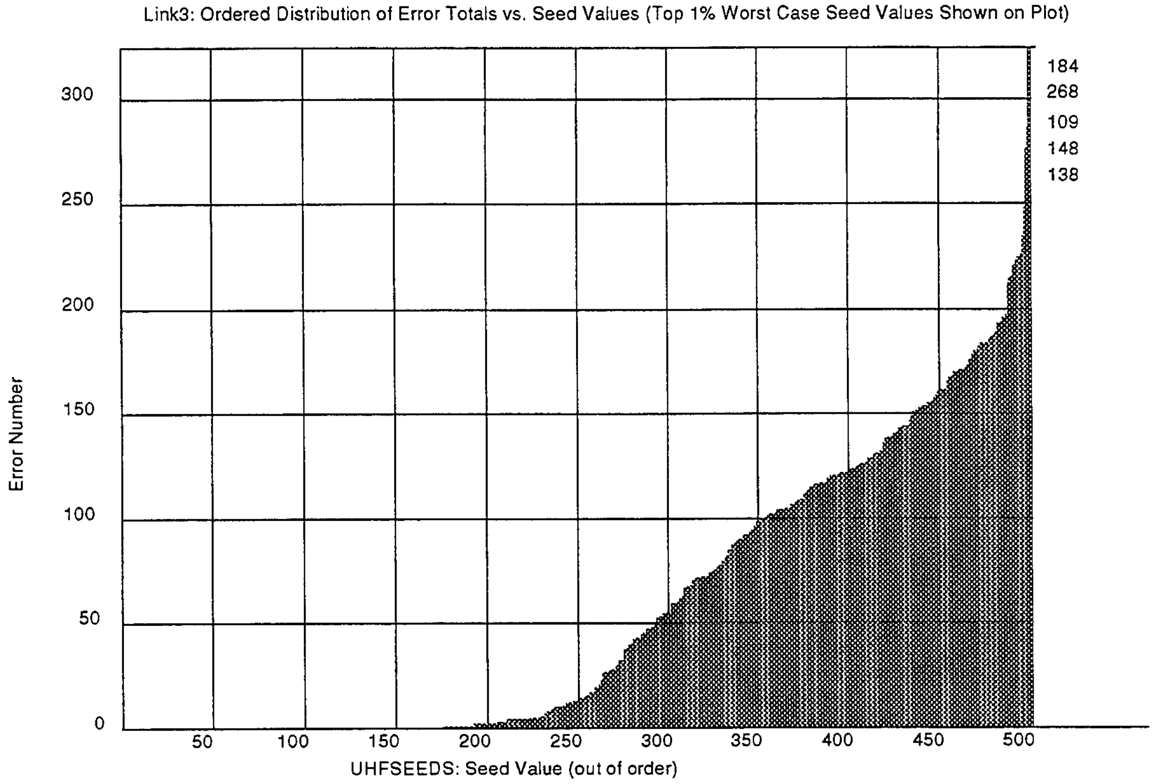


Fig. 52. Link 3 Ordered Distribution of Error Totals Verse Seed Values, 60 Tones

With the system reconfigured to accommodate 60 OFDM tones and time differential encoding/decoding, test phase 3 is repeated and corresponding link 3 seed error reports generated. The results of **uhfseeds.m** simulations using 60 tones, 16-PSK and a message block size of 10,020 symbols are presented in Fig. 51, the seed error report, and Fig. 52, the seed error distribution plot identifying 184 as the worst case seed generating 323 total errors. Fig. 53 depicts the corresponding error histogram indicating the average total errors per seed is 57, an average lower than the 85 errors per seed for 240 OFDM tones using frequency differential encoding/decoding. From these multiple simulations it is apparent that 60 and 240 OFDM tones are near optimal in terms of generating minimal errors per seed and, therefore, will be the focus of subsequent simulation trials during test phases 4, 5 and 6.

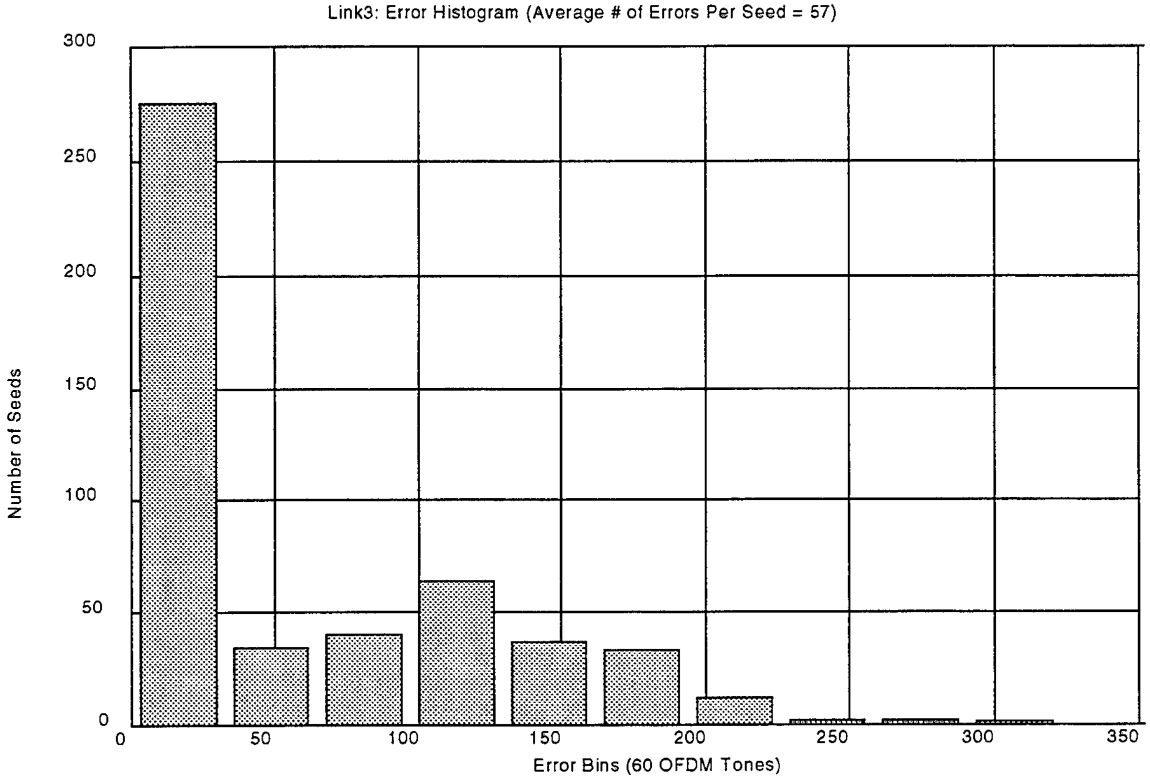


Fig. 53. Link 3 Error Histogram for 60 Tones Using Time Differential Encoding

From this point on, simulations oriented around 60 OFDM tones use time differential encoding/decoding, while simulations oriented around 240 OFDM tones use frequency differential encoding/decoding. The corresponding link 1 and link 2 seed error reports, error distributions and error histograms for 60 and 240 OFDM tones and their respective differential encoding/decoding methods are presented in Fig. 54 through Fig. 65. A record of the worst case seed for each configured link is reserved for later system model 2 and model 3 simulations.

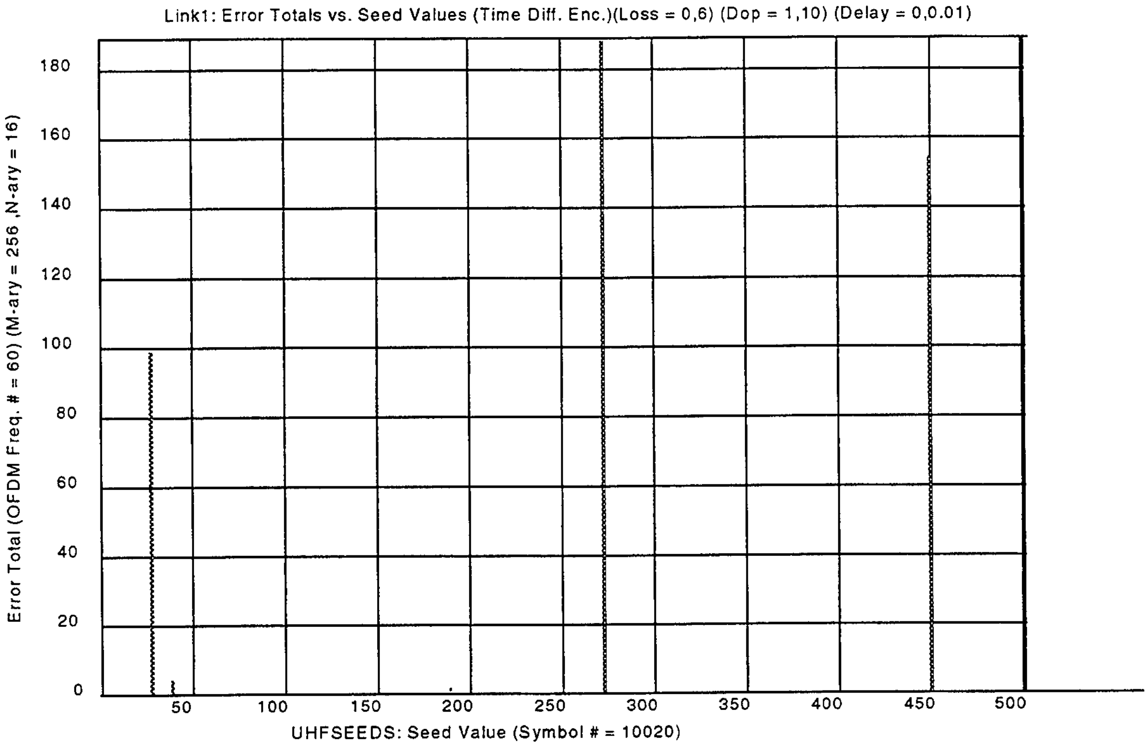


Fig. 54. Link 1 Seed Error Report (60 Tones, Time Differential Encoding)

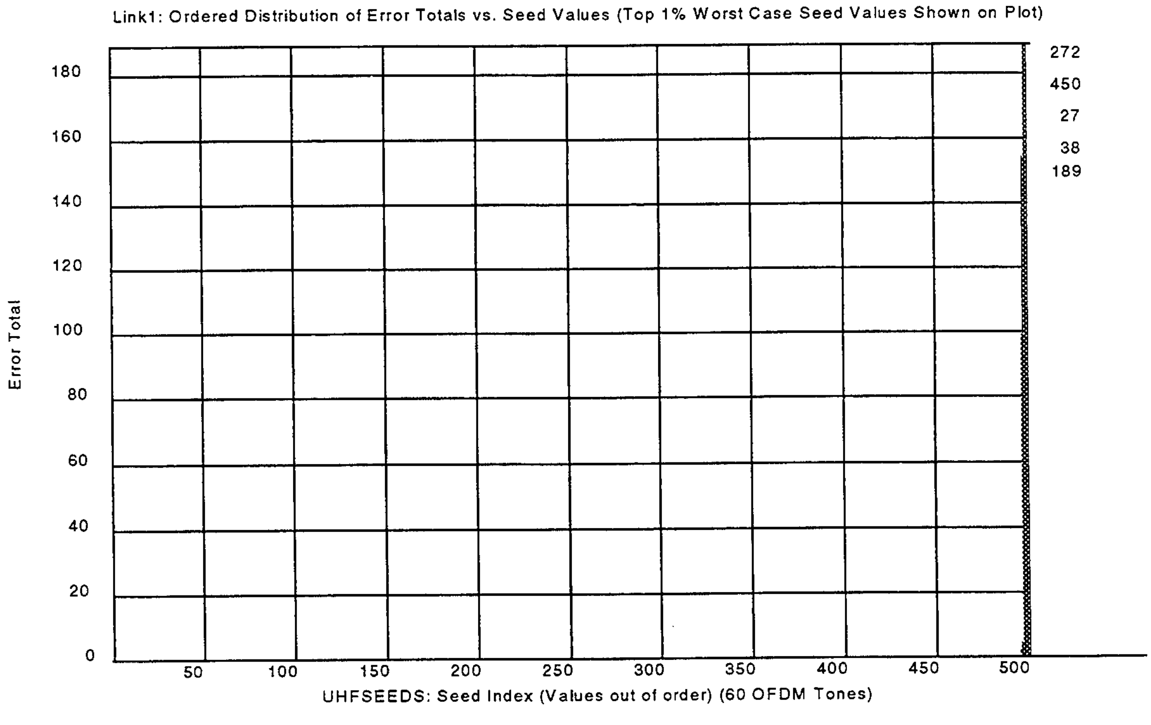


Fig. 55. Link 1 Ordered Distribution of Error Totals Verse Seed Values, 60 Tones

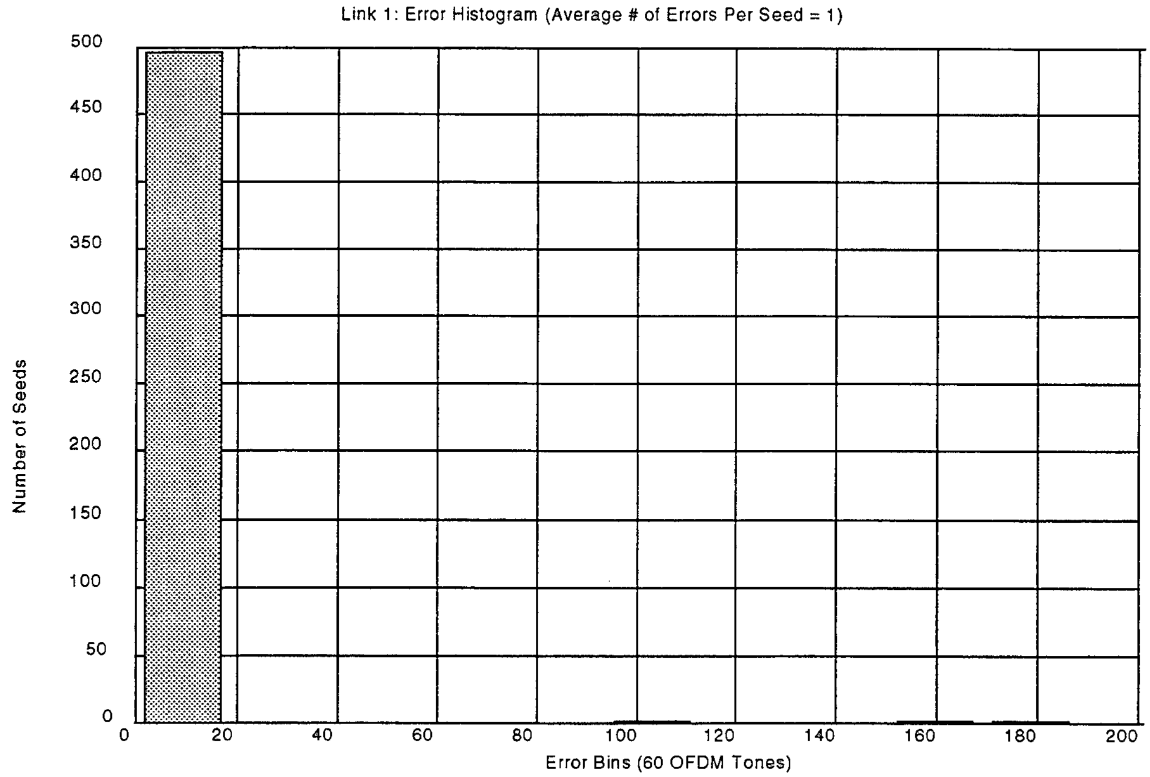


Fig. 56. Link 1 Error Histogram for 60 Tones Using Time Differential Encoding

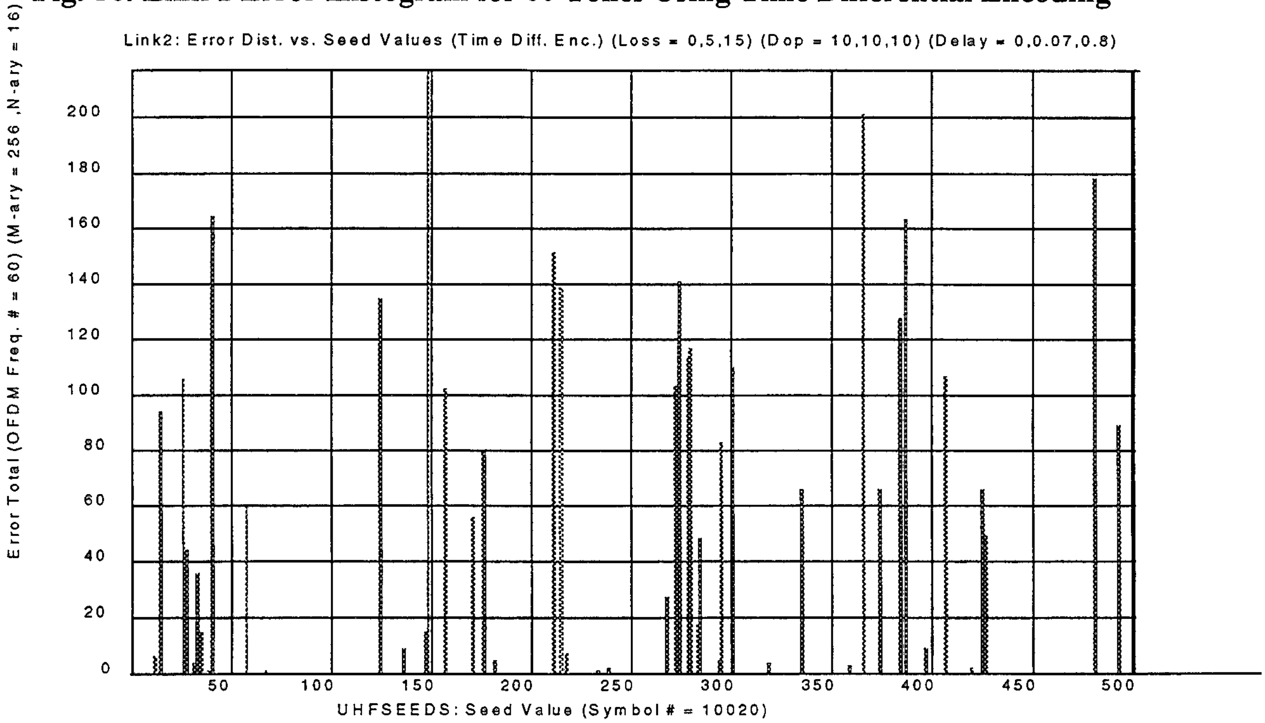


Fig. 57. Link 2 Seed Error Report (60 Tones, Time Differential Encoding)

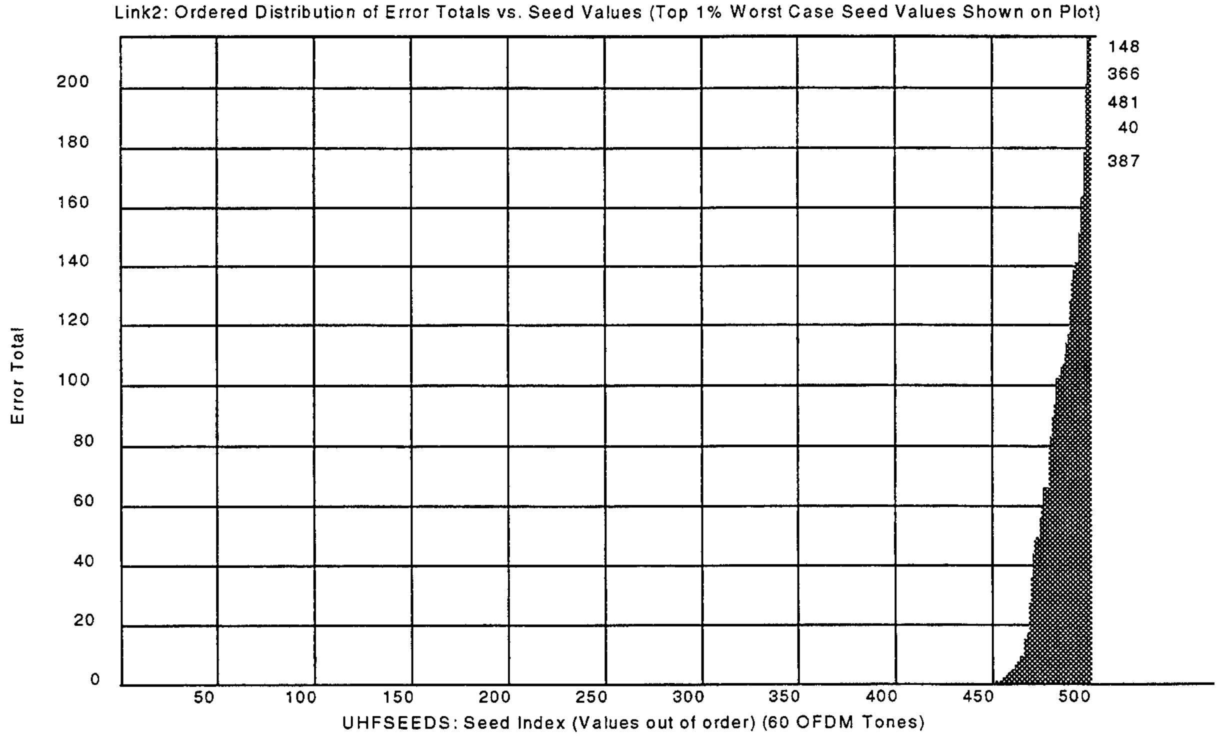


Fig. 58. Link 2 Ordered Distribution of Error Totals Verse Seed Values, 60 Tones

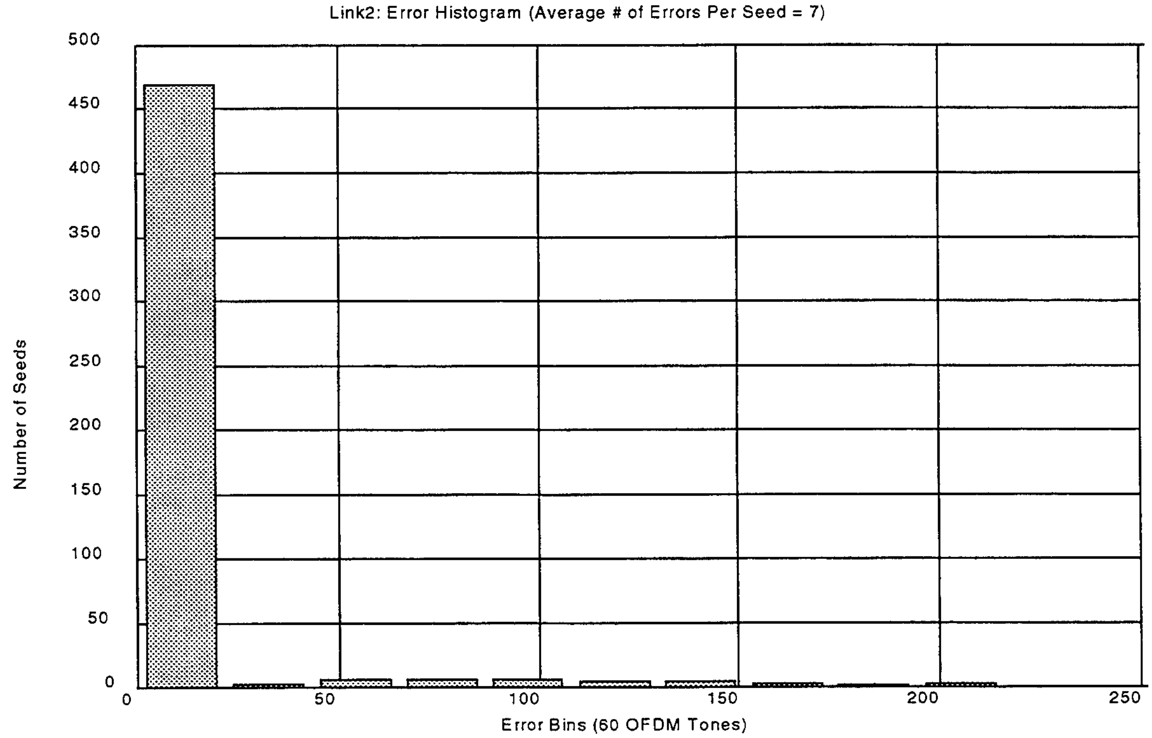


Fig. 59. Link 2 Error Histogram for 60 Tones and Time Differential Encoding

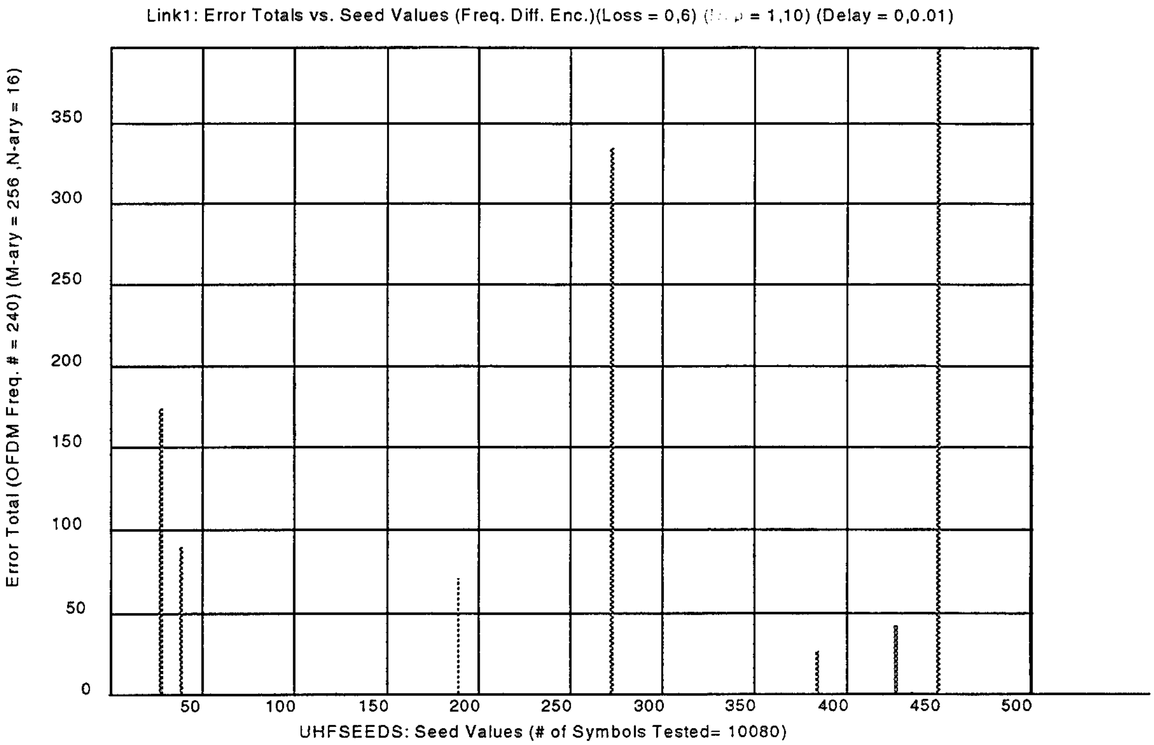


Fig. 60. Link 1 Seed Error Report (240 Tones, Frequency Differential Encoding)

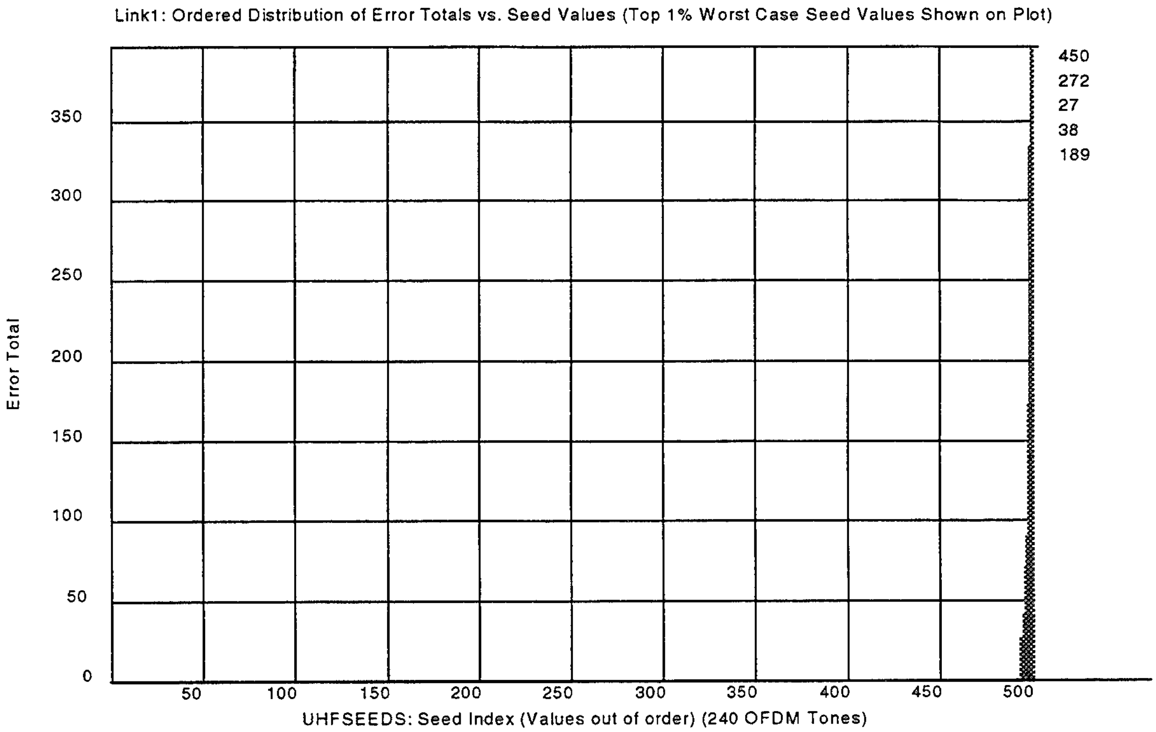


Fig. 61. Link 1 Ordered Distribution of Error Totals Verse Seed Values, 240 Tones

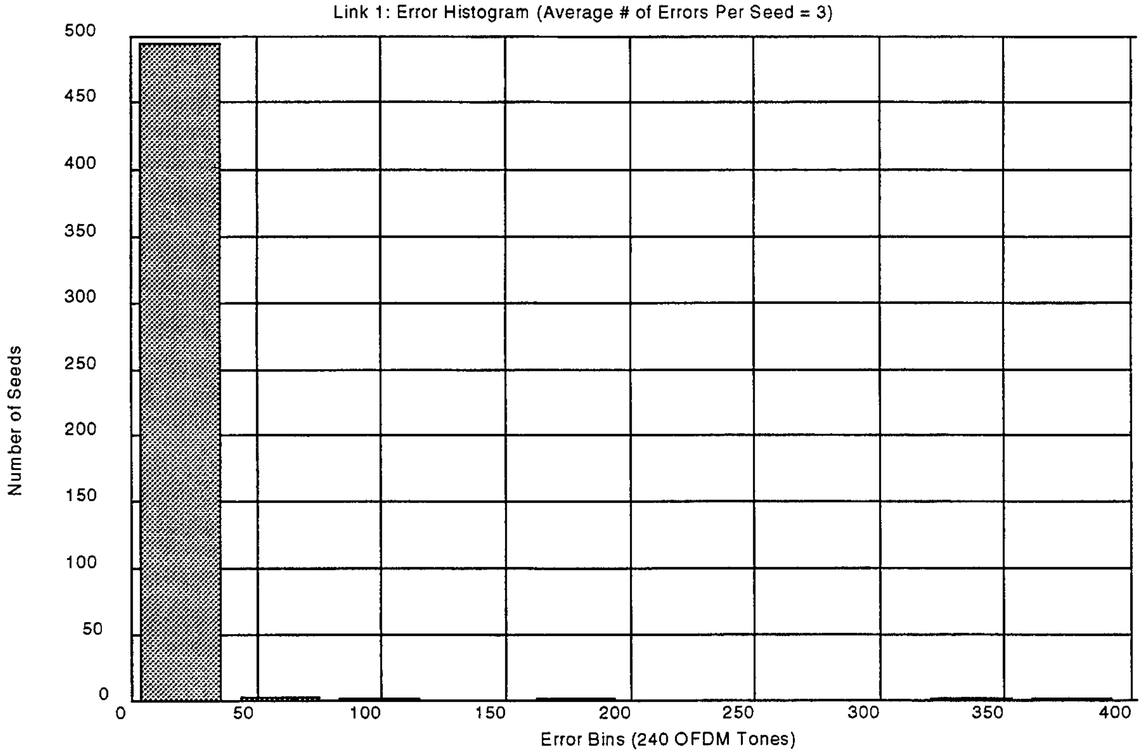


Fig. 62. Link 1 Error Histogram for 240 Tones and Frequency Differential Encoding

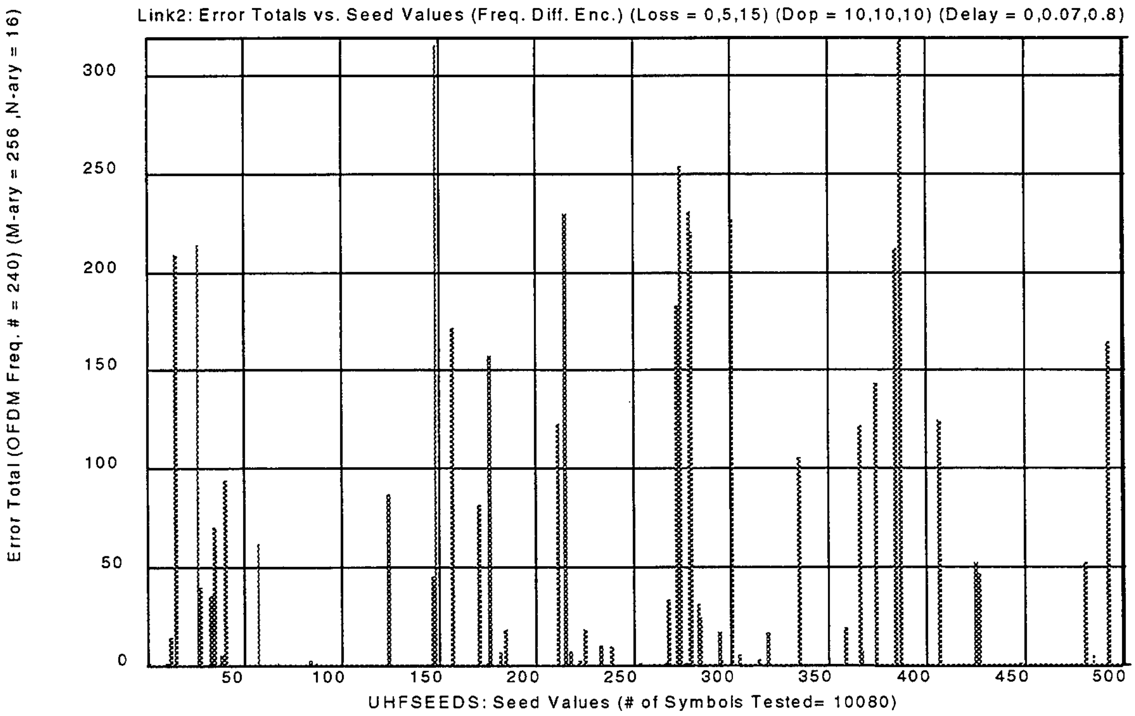


Fig. 63. Link 2 Seed Error Report (240 Tones, Frequency Differential Encoding)

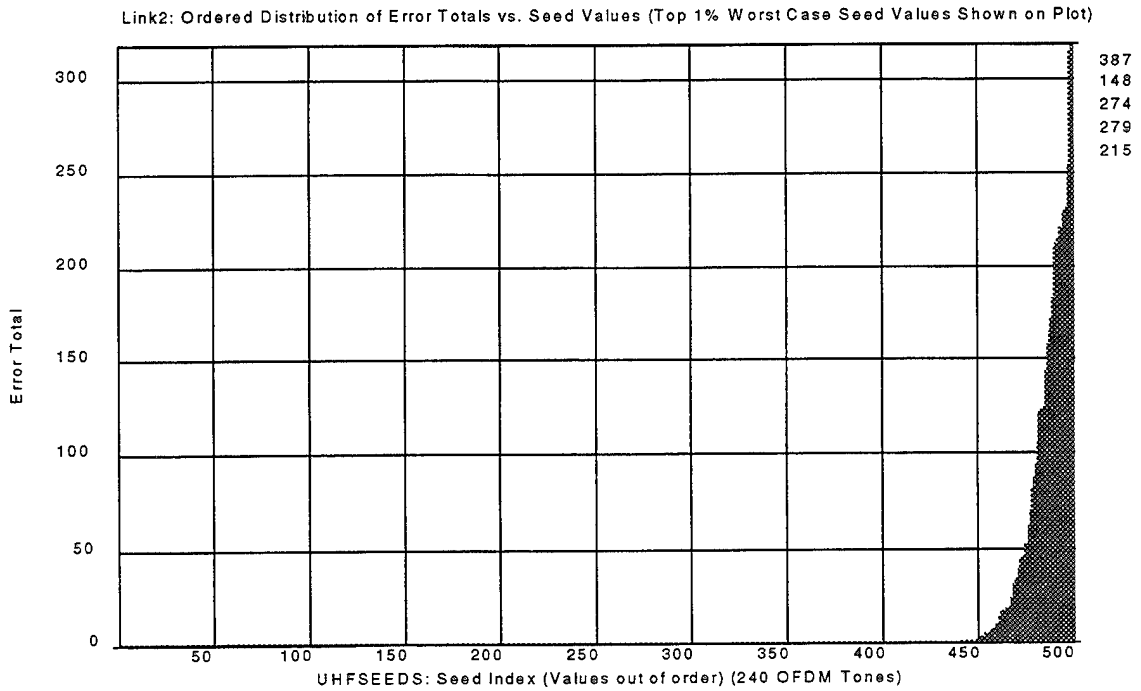


Fig. 64. Link 2 Ordered Distribution of Error Totals Verse Seed Values, 240 Tones

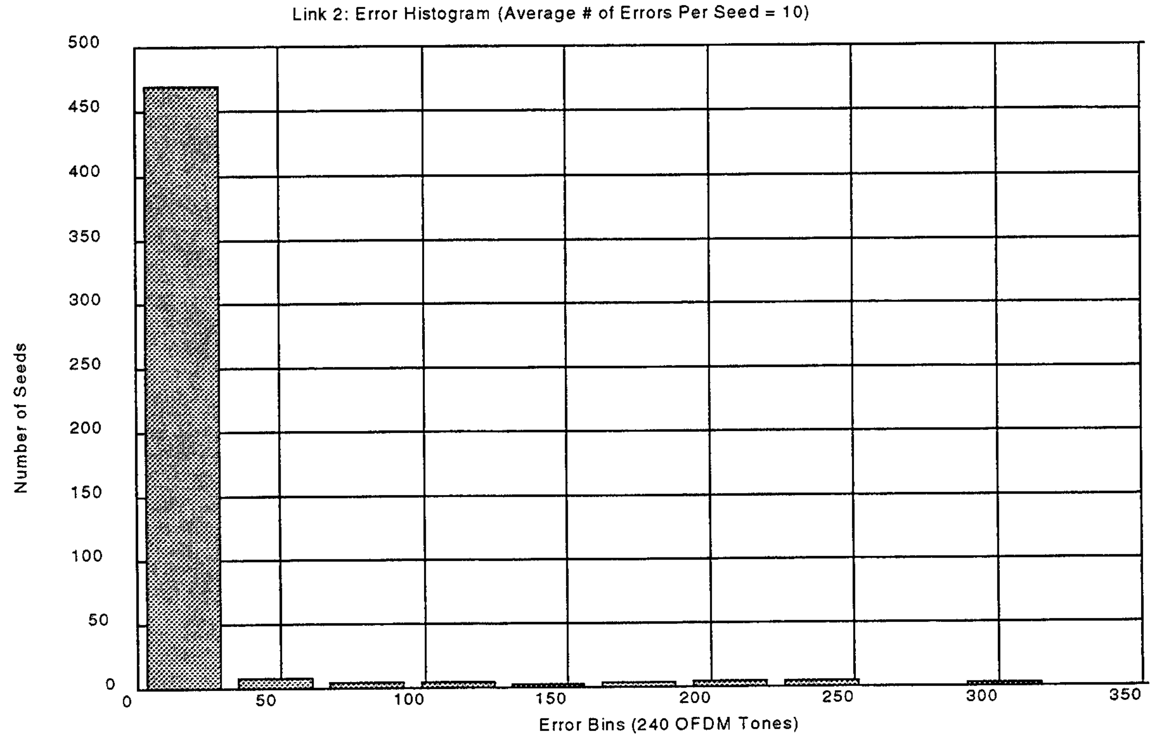


Fig. 65. Link 2 Error Histogram for 240 Tones and Frequency Differential Encoding

E. TEST PHASE 4 - SYSTEM MODEL 2 SIMULATIONS

With 60 and 240 OFDM tones as the preferred system configuration choice, the simulation test plan progresses to phase 4. The objective of this test phase is to simulate the system transmitting symbols through the multipath channel exclusively to reveal the burst error patterns associated with worst case channel seeds. It is instructive to observe general burst noise error patterns within the message array without interleaving to recognize familiar fading behaviors so as to later identify during test phase 5 optimal interleaver configurations. The link 3 simulations performed during this test phase use the worst case seeds, 184, corresponding to 60 OFDM tones and 279, corresponding to 240 OFDM tones. Using these "bad" seeds ensures that worst case channel conditions exist during simulations and generate the most errors. Phase 4 simulations are conducted using batch file **cofdmsim.m** configured for system model 2 testing as exemplified in Table 22. Links 1 and 2 are also simulated later during this test phase to identify their corresponding error patterns.

```
» cofdmsim  % Perform a system model 2 simulation with 60 OFDM tones and 300 symbols. (16-PSK)
________________________________________________________________________________

This batch m-file runs COFDM simulations using different channel models.
To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or to run both enter 2
(two): 0
Enter the # of OFDM frequencies (Note: Must be even): 60
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 64
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter 0,1,2
or 3): 2
Channel model 2 simulation performed.
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 3
Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): 0
Enter specific interleaver case numbers from (0 to 8) (Ex. [0 4 5 8]): 0
Enter the total minimum number of symbols to simulate (Ex. 10000): 280
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 300
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 300], or [300 1], offers no
interleaving functionality): [300 1]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Enter the guard interval length (Number of sample points): 6
Do you want to include error correction coding? (1 = yes, 0 = no): 0
Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 184
Do you want signal plots? (1 = yes, 0 = no): 1
How many seconds of delay between pictures? 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 22. System Model 2 Simulation With 60 OFDM Tones

While performing **cofdmsim.m** system model 2 simulations, various output plots depicting various forms of the signal data at strategic stages in the signal path are possible if desired and configured by the user. As an example of the types of plots generated during simulations, Fig. 66 through Fig. 71 depict corresponding information generated by batch m-file **cofdmsim.m.** configured as in Table 22. This example uses only 600 OFDM symbols during system model 2 simulations to reduce the number of data points for plotting demonstration purposes primarily. Actual meaningful system simulations oriented around 240 and 60 OFDM tones use the full complement of approximately 10,000 OFDM symbols for source message transmissions; however, corresponding high detail plots are not displayed but summarized instead.

Fig. 66 depicts the constellation plot characteristic of 16-PSK type modulation. As expected, 16 individual phase points are generated resulting from symbol mapping of 4-bit words into complex modulation values with unit magnitude and one of 16 possible phases. The constellation points, denoted by an asterisks, are positioned symmetrically spaced on the unit circle, partitioning the circle into 16 equally sized sector formations.

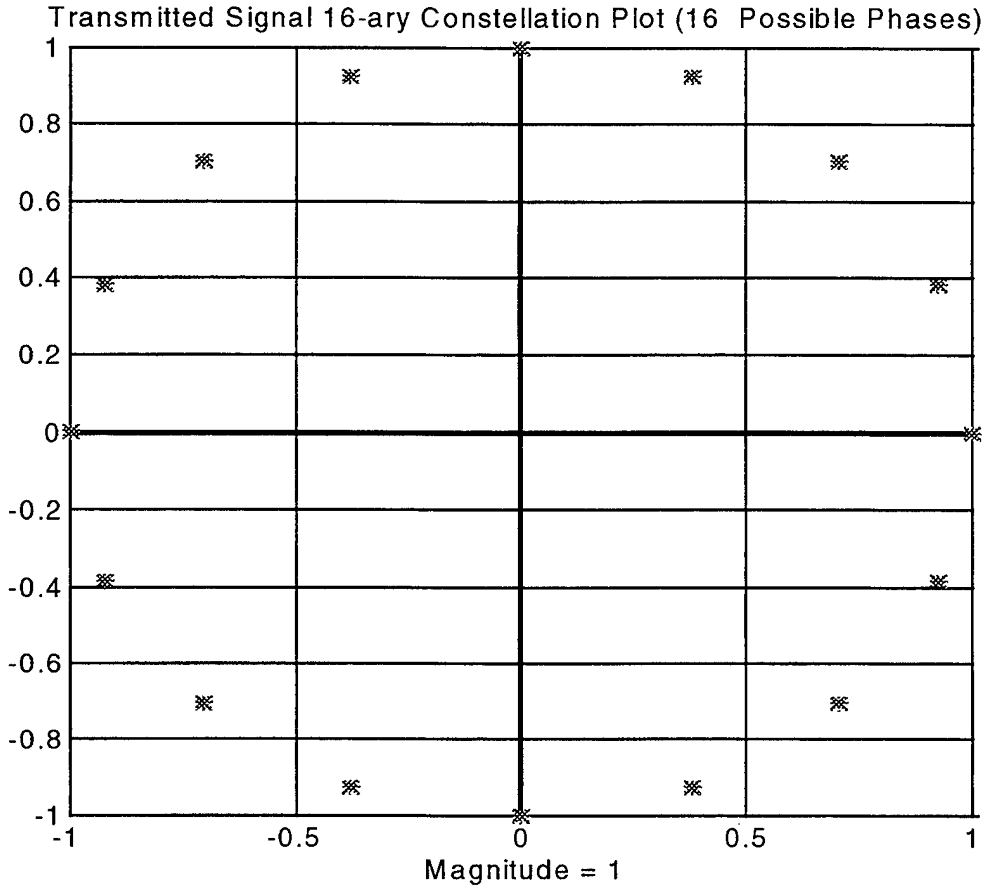


Fig. 66. Ideal Signal Constellation Plot of Transmitted 16-PSK Signal

The corresponding message array of reformatted 4-bit PSK symbols with unit magnitude are depicted in Fig. 67 and are transmitted through the channel. Notice the flat planar magnitude representation of the symbols prior to transmission. In this example, there are 60 OFDM tones (columns) and 10 symbol rows for a total of 600 transmitted PSK symbols. The increase in symbol number from an initial specified 300 OFDM symbols (Table 22) to the actual transmitted 600 PSK symbol quantity is the result of symbol reformatting from 8-bit words to 4-bit word format. Recall that once a simulation is configured for a specified number of OFDM frequency tones, the number of tones remain fixed throughout the simulation duration. Consequently, additional symbols may be generated as a result of symbol word reformatting, increasing the original message array size in the time dimension (added symbol rows). For this example, the total number of symbols doubles from 300 to 600 since the symbol word lengths halve from 8-bits to 4-bits.

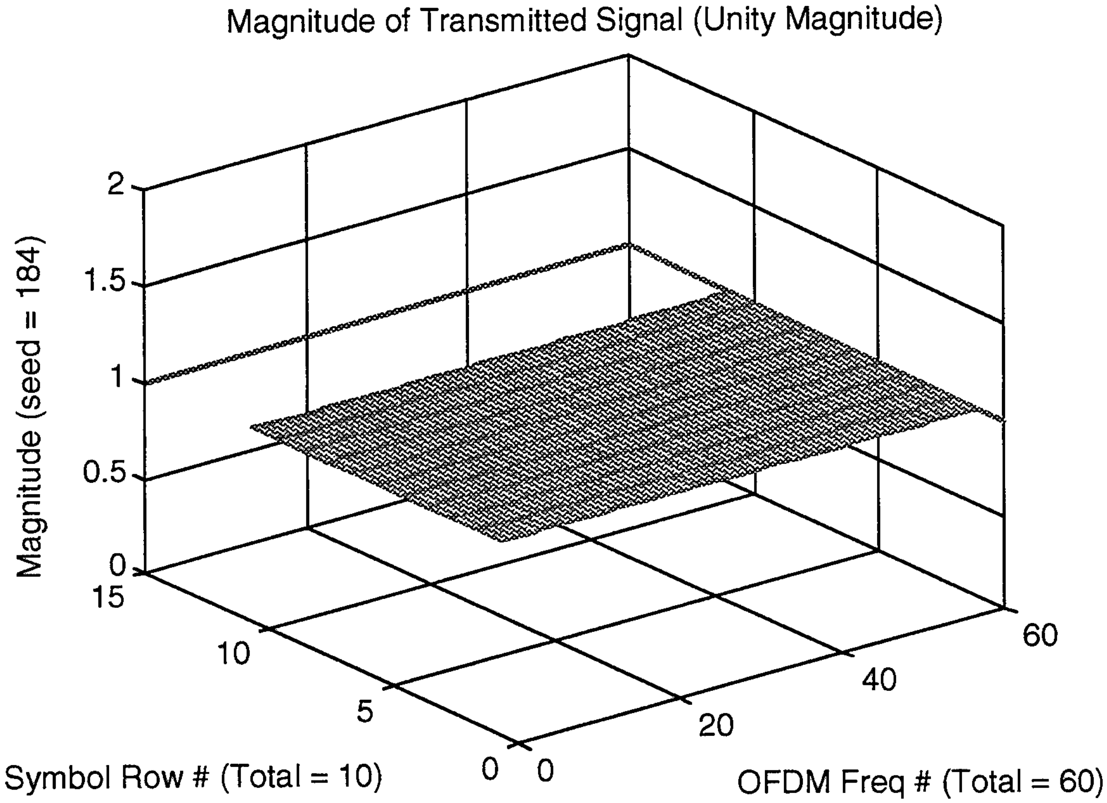


Fig. 67. Constant Unity Magnitude Plot of Transmitted Signal

The corresponding received signal constellation plot is shown in Fig. 68. As a consequence of multipath distortions within the channel causing constructive and destructive signal interference, the received constellation points are scattered from their normal pre-transmitted positions (Fig. 66). The figure also suggests that without additional signal conditioning, a majority of the received symbols would be decoded in error since many points cross sector borders into adjacent phase sectors. However, with the inclusion of time differential encoding as demonstrated in Fig. 69, the constellation points realign within their respective sector spaces forming a distinct star like structure.

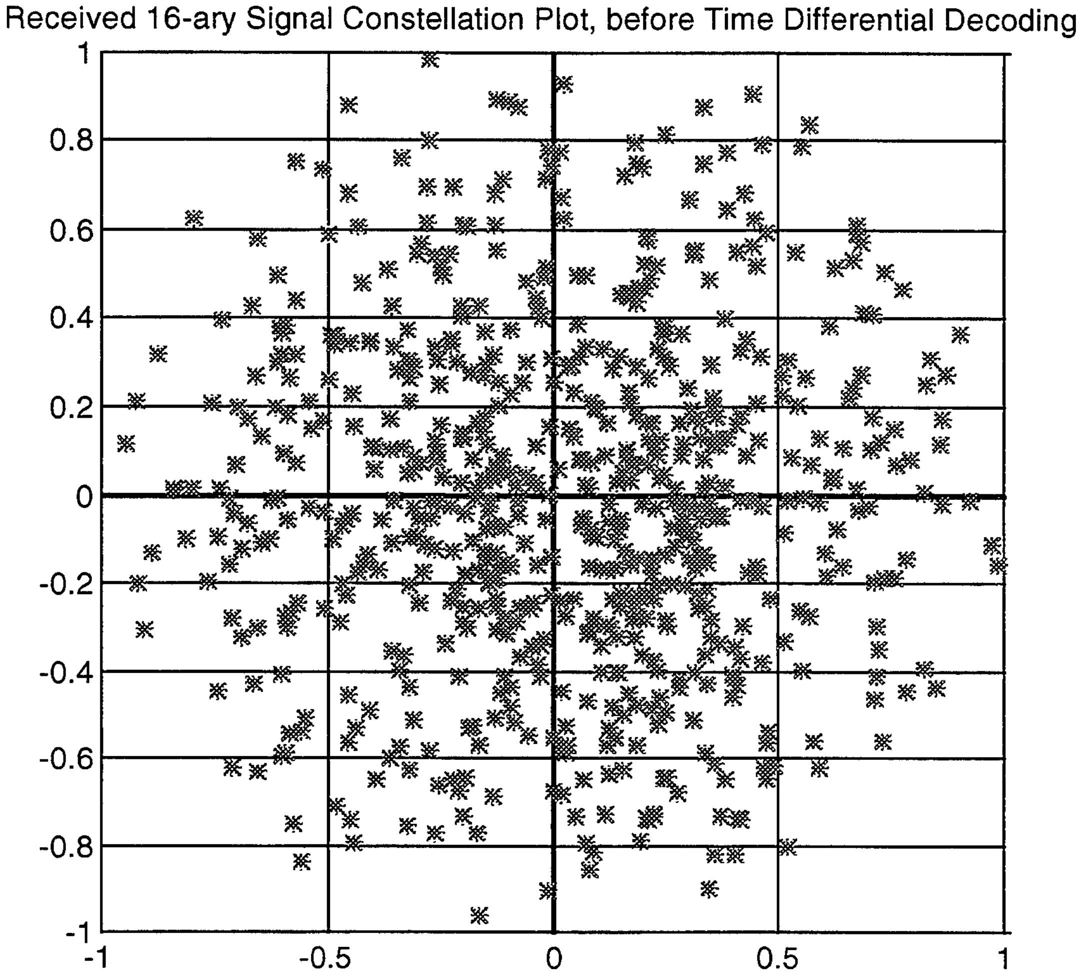


Fig. 68. Constellation Plot of Received Signal Showing Multipath Distortion

Additionally, there is a resulting signal energy loss as is evident by some constellation points converging upon the origin from their normal unit circle positions. Hence, the benefit of multipath distortion error reduction through differential encoding/decoding is gained at the expense of partial signal energy loss. In general, for a fixed symbol error rate, differential encoding/decoding can require up to an additional 3 dB of signal energy as an identical system without differential encoding/decoding; how^ever, as is evident by reduced error total improvements, differential encoding/decoding is well worth the signal energy loss expense [8; p. 147].

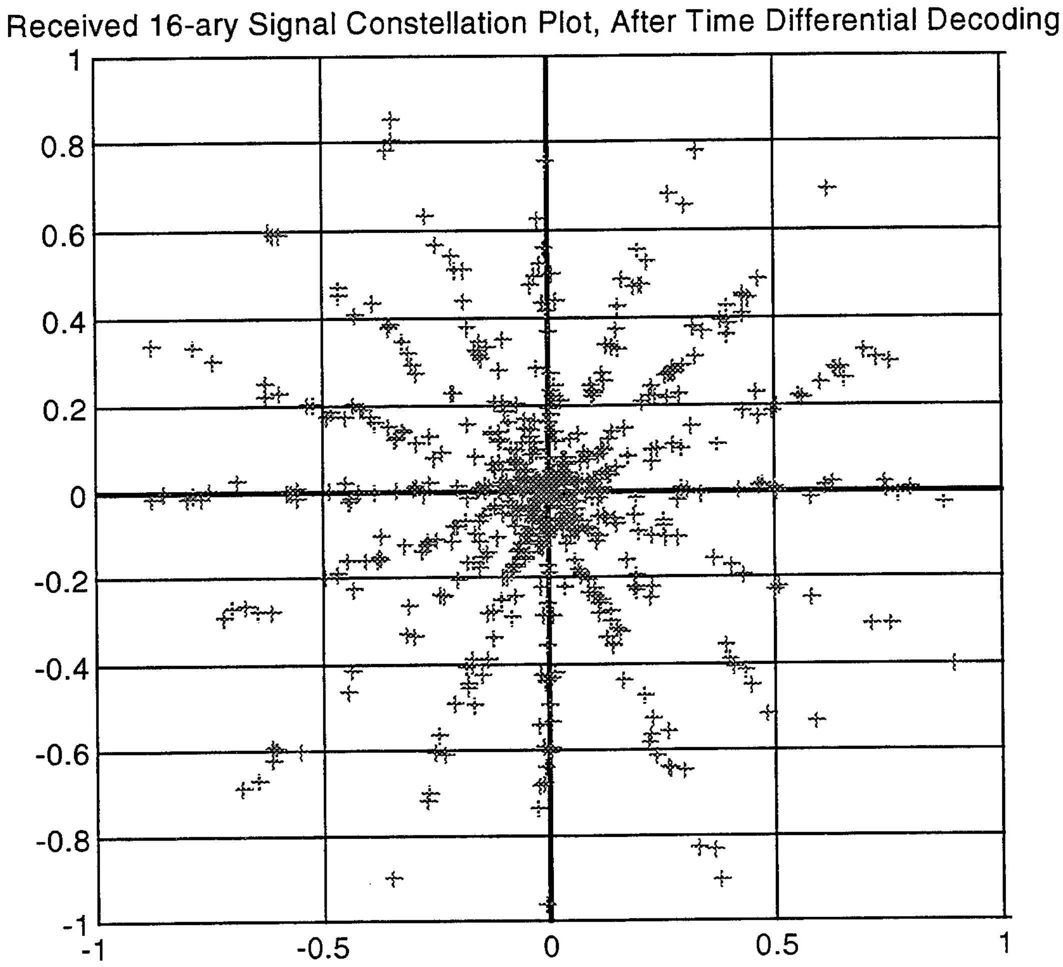


Fig. 69. Constellation Plot of Received Signal After Time Differential Decoding

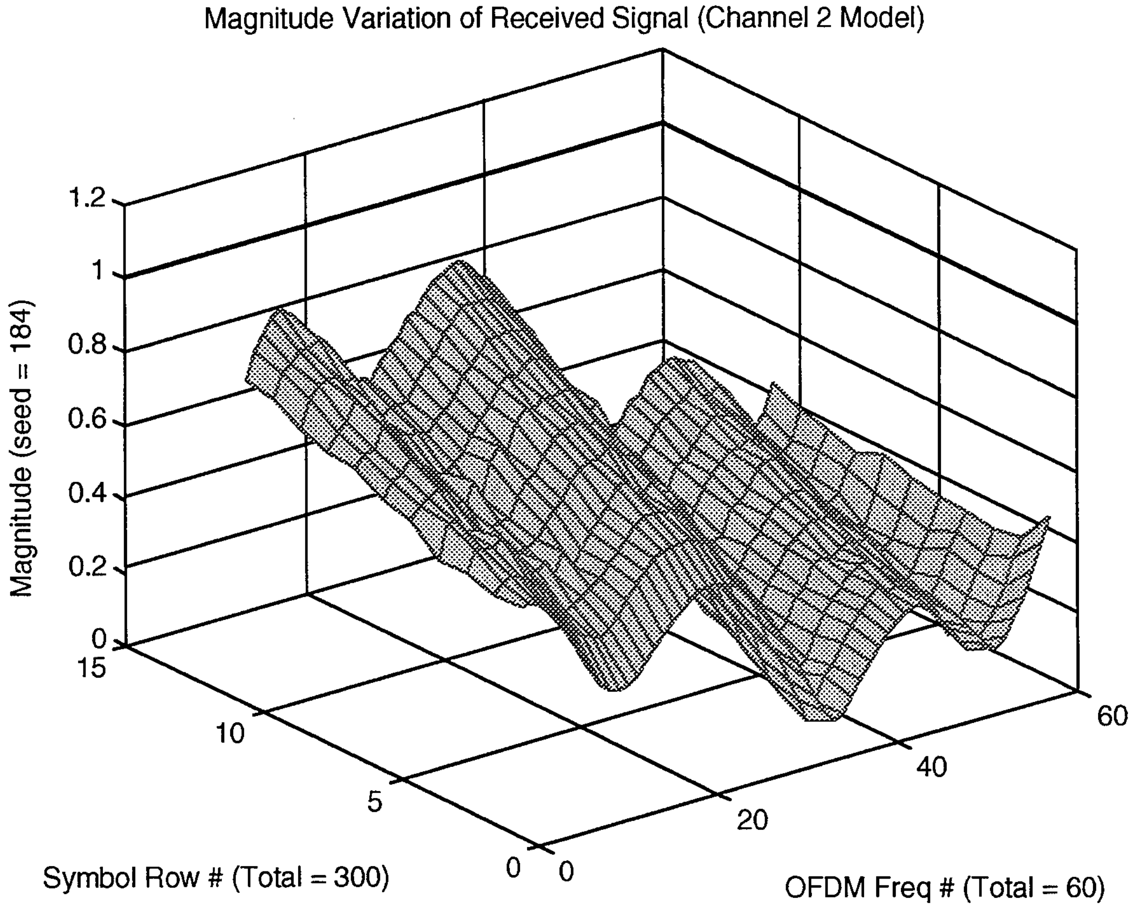


Fig. 70. Magnitude Plot of Received Signal Demonstrating Power Fading

A corresponding received signal magnitude plot is depicted in Fig. 70 with noticeable variations in the RSL, indicative of power fading. In stark contrast to the pre-transmitted magnitude plot (Fig. 67), the noticeable peaks and valleys in the received magnitude plot demonstrate the consequences of multipath distortion influences on the transmitted signal through constructive and destructive signal interference by altering the message symbol magnitudes from their pre-transmitted unity levels. It is apparent that for this link 3 system model 2 simulation using a worst case channel seed, frequency selective fading occurs causing the frequency dependent peaks and valleys of the RSL.

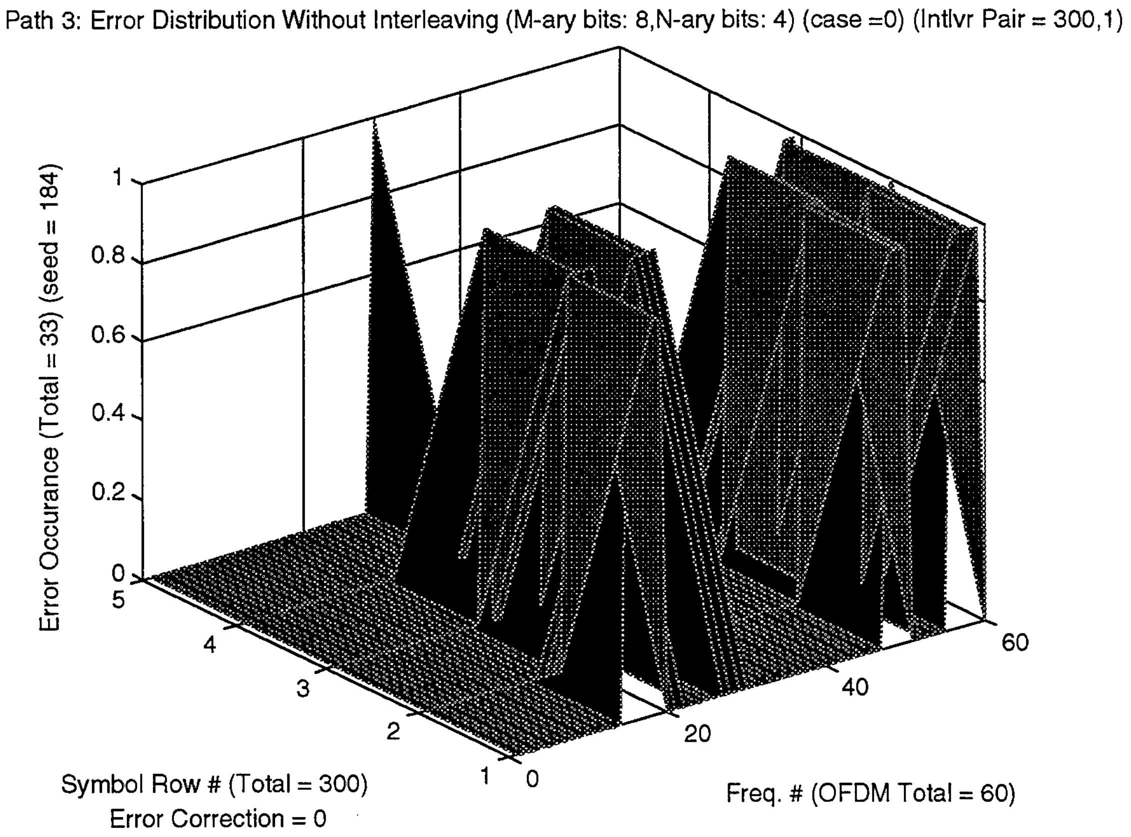


Fig. 71. Corresponding Error Matrix Identifying Symbol Error Locations

While some RSL variations may cause signal reinforcement, allowing symbols to be decoded correctly without errors, Fig. 71 demonstrates how incorrect message symbol decoding results in the formation of an error matrix, indicating symbol error locations within the sink message array. In this example, 33 errors exist in the decoded sink message block out of 300 total OFDM symbols, or a 11% OFDM symbol error rate. Without further FEC, these errors remain corrupting the message, and the 99% reliability criteria stated in the BAA is not met once AWGN is included in the channel. As evident from Fig. 70 and Fig. 71, indicating frequency selective fading and power loss, formations of isolated error groups occur not only along the OFDM frequency dimension, but also across symbol rows in the time dimension. It appears that by using effective interleaving, these correlated error groups could be redistributed through the rest of the sink message array where no errors currently exist, allowing for more effective FEC with minimal code strength.

Throughout this test phase, multiple system model 2 simulations are performed using 60 and 240 OFDM frequency tones with a sample message size of approximately 10,000 total OFDM symbols. While it is redundant to display all of the simulation output plots previously presented by example in Fig. 66 through Fig. 71 using 60 OFDM tones, it is instructive to observe the received sink message error matrix demonstrating unique link 3 symbol error distributions for the configurations of interest.

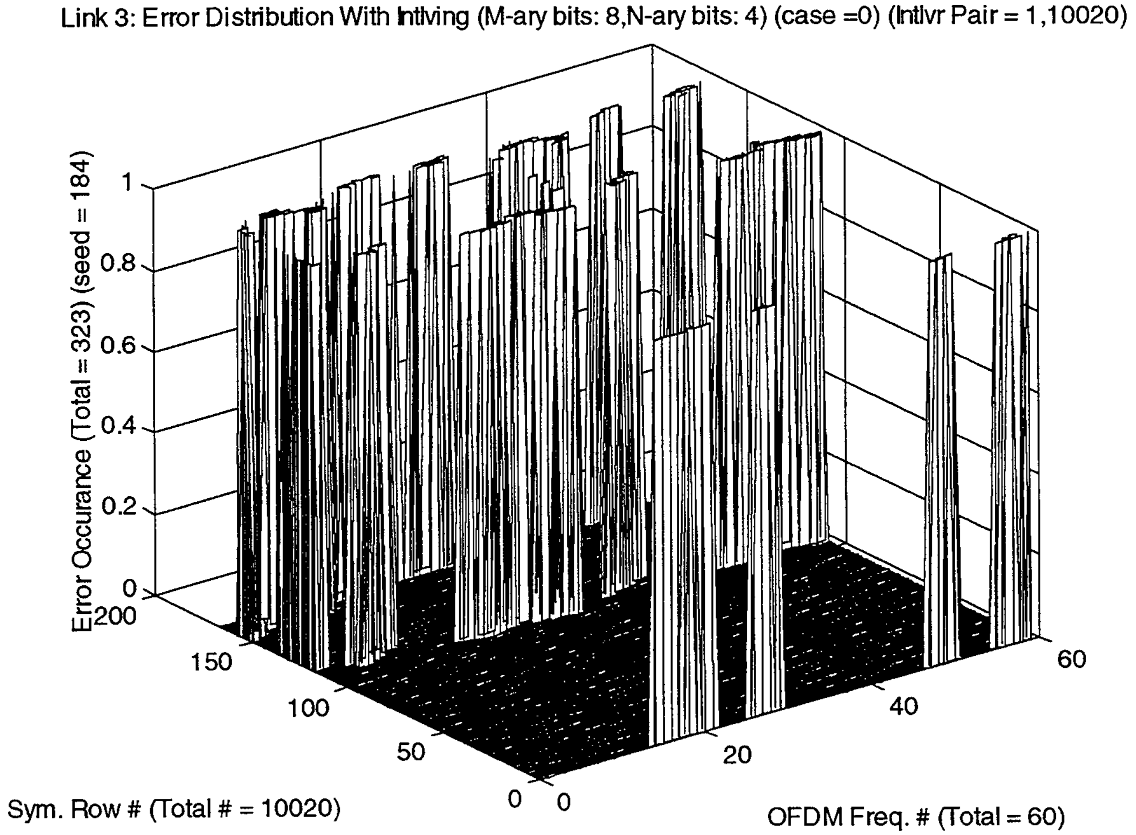


Fig. 72. Link 3 Error Matrix For 60 OFDM Tones Using 10,020 Symbols

The link 3 system model 2 error matrix corresponding to 60 OFDM tones using time differential encoding/decoding and no effective interleaving is shown in Fig. 72. From the figure it is again apparent that frequency selective fading occurs within the channel as is evident by error free gaps existing between isolated symbol error groups along the OFDM frequency dimension. Out of 10,020 simulated OFDM symbols, 323 are in error, 3.2% of the entire message block. This simulation uses a worst case channel seed of 184 and the most challenging link 3. Corresponding error matrices for link 1 and link 2 are shown in Fig. 73 and Fig. 74, respectively, both using 60 OFDM tones, 16-PSK and 10,020 symbol message block size along with their worst case seeds.

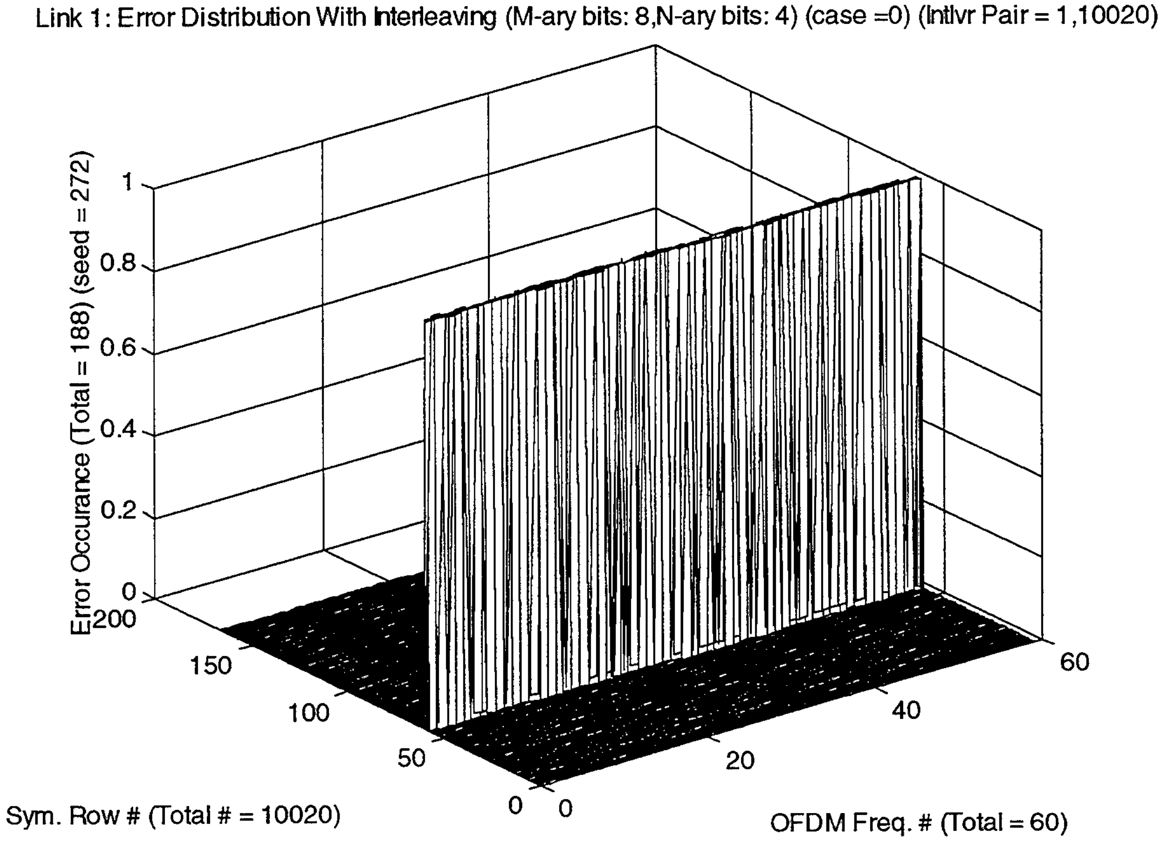


Fig. 73. Link 1 Error Matrix For 60 OFDM Tones Using 10,020 Symbols

Referring to Fig. 73, link 1 and the corresponding worst case seed, 272, we observe that flat fading occurs within the channel affecting all OFDM tones equally as identified by the complete row of symbol errors. In Fig. 74 while using a link 2 worst case seed of 148, flat fading again occurs affecting roughly half of the OFDM tones for one row, and the other half of the OFDM tones in another row. Based upon observations of both these simulation outputs, interleaving should be helpful in breaking-up the concentrated error bursts and re-dispersing them to other symbol rows where there are an absence of errors.

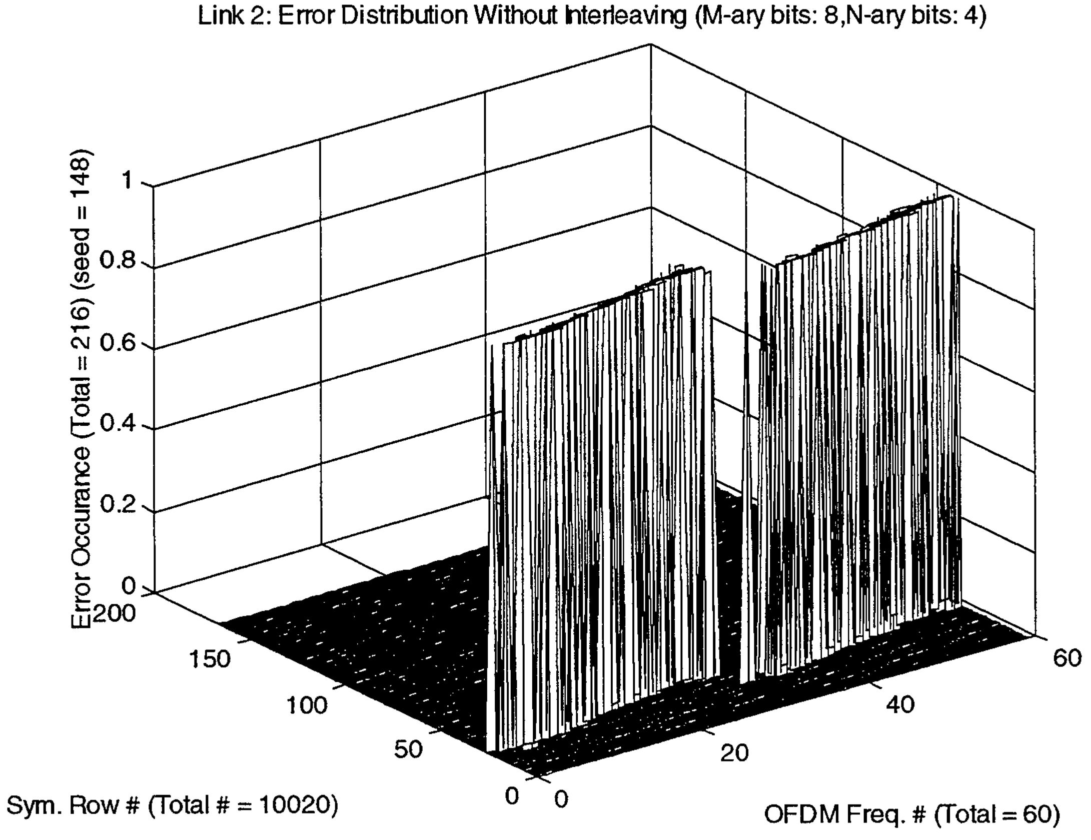


Fig. 74. Link 2 Error Matrix For 60 OFDM Tones Using 10,020 Symbols

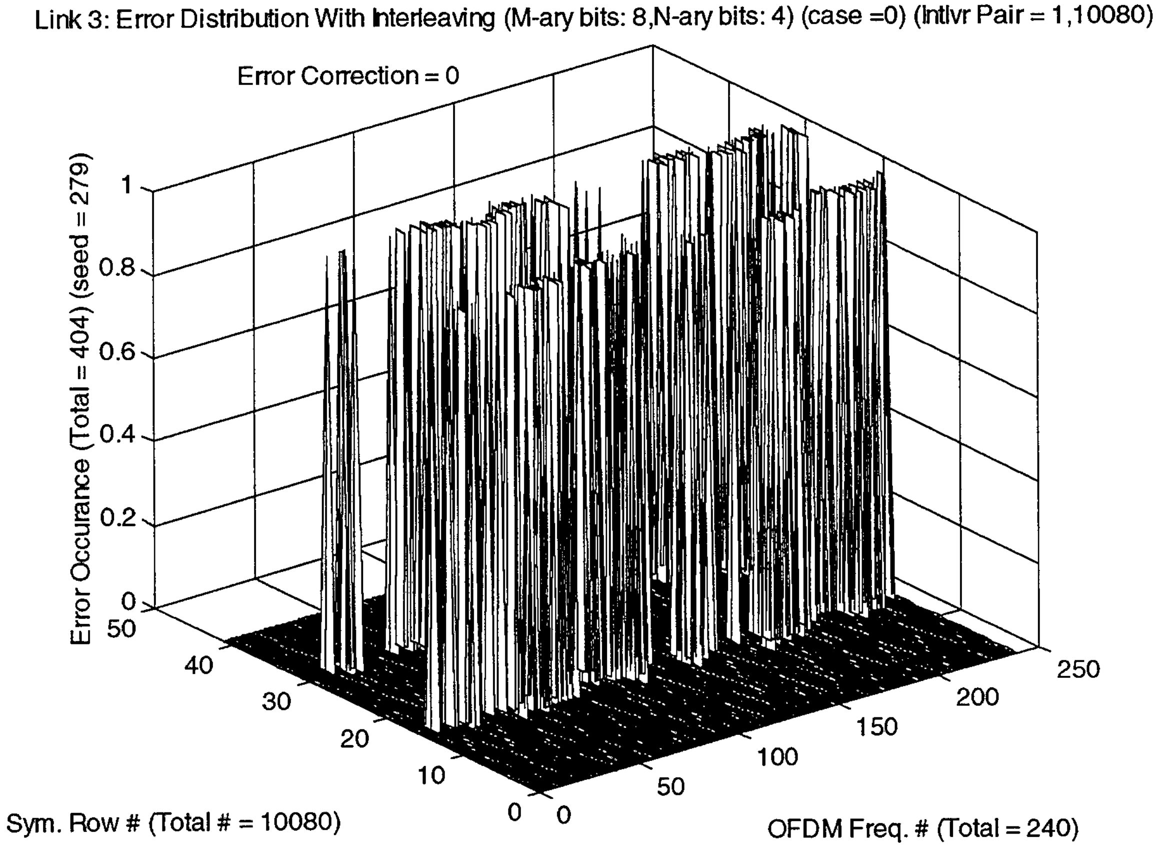


Fig. 75. Link 3 Error Matrix For 240 OFDM Tones Using 10,080 Symbols

In Fig. 75 a resulting simulation message error matrix plot corresponding to 240 OFDM tones, frequency differential encoding/decoding and 10,080 simulated OFDM symbols without effective interleaving is presented. From the figure it is again apparent that flat fading occurs for a few rows of symbols while frequency selective fading occurs during other symbol rows while using the worst case link 3 channel 2 seed of 279. Fig. 76 and Fig. 77 present similar sink message error matrices for identical configurations using links 2 and 1 with corresponding worst case channel model 2 seeds. Fig. 76 demonstrates link 2 flat fading for about half the frequencies along a group of rows and flat fading for the other half of frequencies along an adjacent symbol row group. Fig. 77 once again depicts flat fading across all frequencies affecting a few rows during a link 1 simulation.

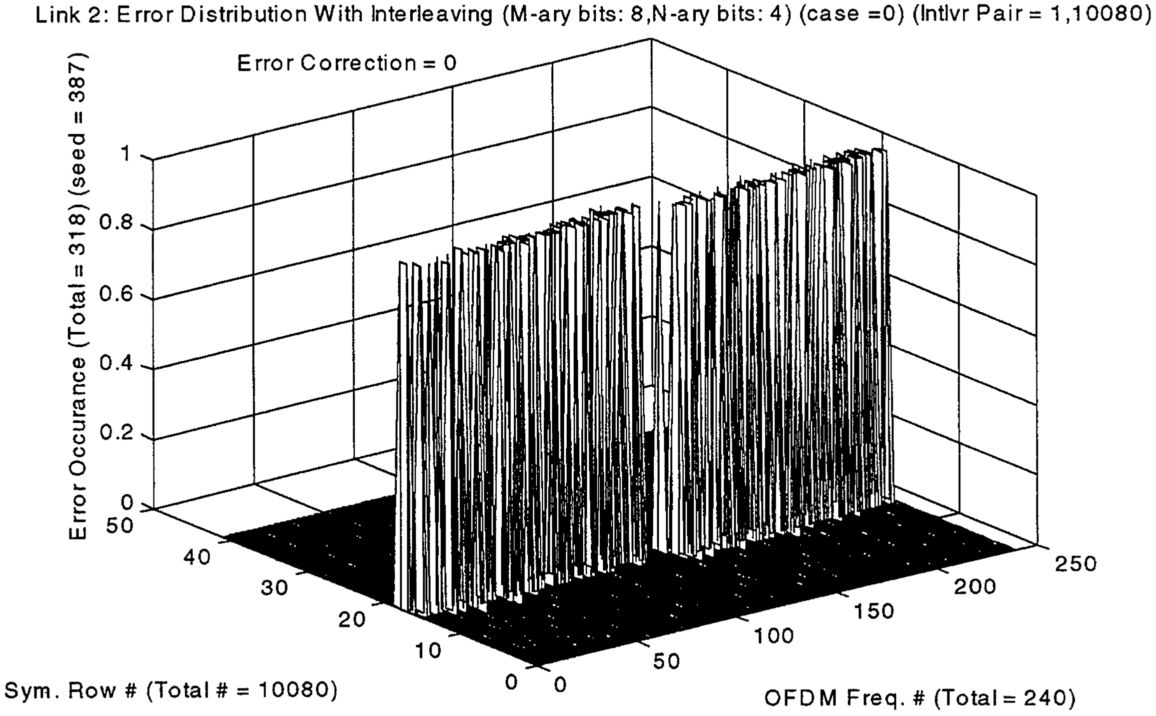


Fig. 76. Link 2 Error Matrix For 240 OFDM Tones Using 10,080 Symbols

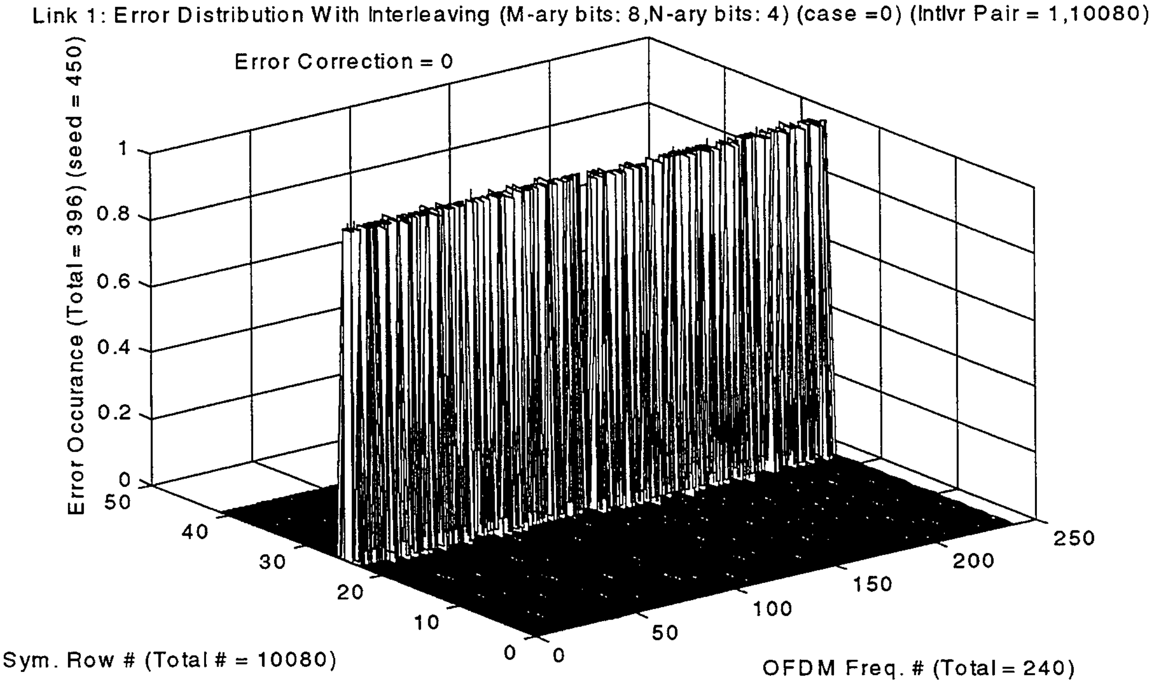


Fig. 77. Link 1 Error Matrix For 240 OFDM Tones Using 10,080 Symbols

F. TEST PHASE 5 - IDENTIFYING OPTIMAL INTERLEAVER CASES

From the previous figures displayed as part of test phase 4, it is apparent that multipath channel 2 burst errors occur in correlated groups along symbol rows during transmissions using the 60 OFDM and 240 OFDM tone configurations. Consequently, multiple errors appear across multiple OFDM frequency tones and corrupt many adjacent symbols. Without FEC, these burst errors would remain in the final sink message array, degrading performance and diminishing communication reliability with transmission outages.

Obviously a suitable FEC code of sufficient strength must be included to correct symbol errors and recover lost information. However, it is also apparent from the figures that most other sink error matrix rows are devoid of any symbol errors. In this situation interleaving could be effective in decorrelating the afflicted rows by redistributing errored symbols along rows where few or no errors exist. Consequently, the error concentrations across OFDM frequency tones (along rows) could be reduced, allowing for a weaker code to effectively correct all errors. If a weaker code can sufficiently correct all errors as well as a stronger code can, then the weaker code is preferable since the code rate improves and increases the information rate. Thus, it is advantageous to determine which CDL interleaver case is more effective in dispersing errors throughout the message array in such a way that minimal error totals across any given row are formed. In this way, a minimal strength code, which is designed to perform the error correction functionality across rows, can be more effective.

Recall from Fig. 72 the fading that occurs during transmission and the corresponding concentrated burst error events in the sink message array. To demonstrate the affects of message symbol interleaving, Fig. 78 presents an identical configuration as in Fig. 72, however, this time case 2 interleaving is included using a (60,167) dimension intermediate matrix. The interleaving effects are apparent as the concentrated error groups are sufficiently broken up and strewn throughout the message array.

Consequently, the scattered errors appear as randomly isolated events uncorrelated to specific OFDM frequencies. The advantages of interleaving can only be exploited if suitable FEC is also included, since the errors do not go away but are simply relocated somewhere else within the sink message array. It is intuitively apparent that certain interleaver cases will redistribute errors within identical sink message arrays differently; but which interleaver performs optimally when combined with R-S FEC needs to be investigated. Identification of an interleaver that minimizes total row errors proves useful in determining the required code strength.

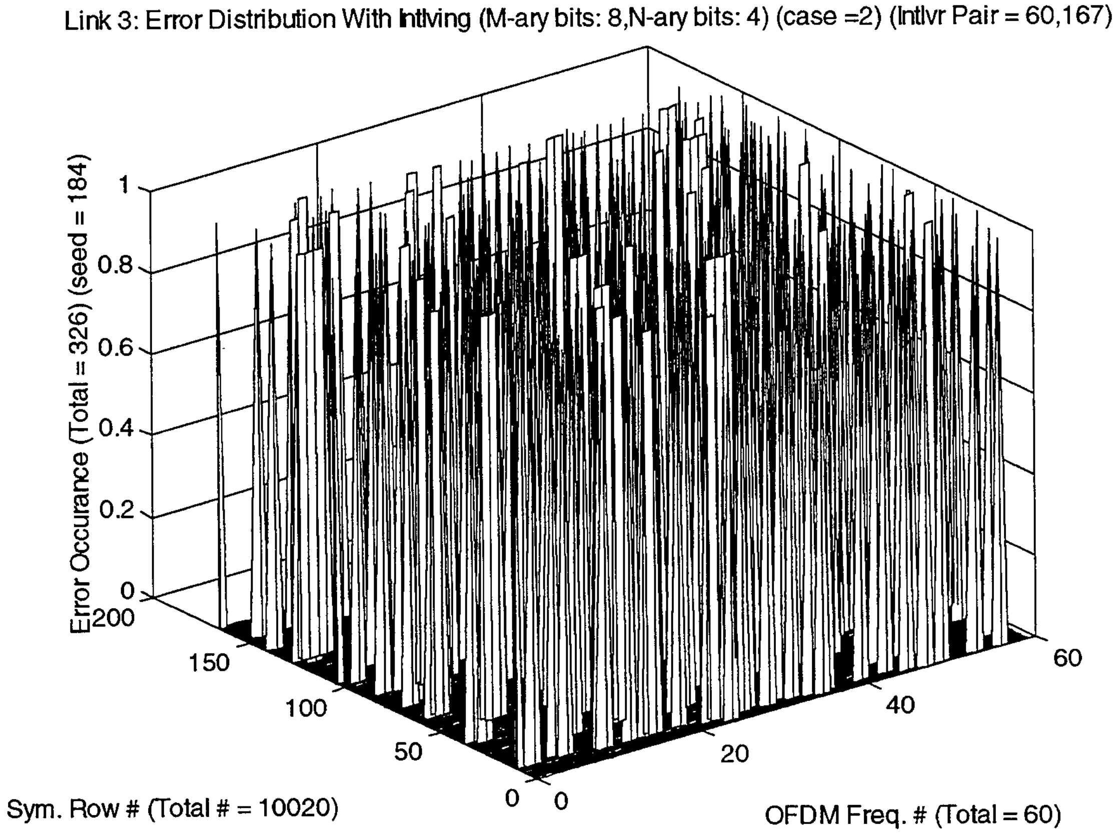


Fig. 78. Link 3 Interleaved Error Matrix (Case 2) For 60 OFDM Tones

The previous simulation demonstration depicts the interleaving operation and suggests through example its practical benefit; however, it is desirable to determine the most appropriate and optimal interleaver for different system configurations. This is the motivation for adopting test phase 5, to identify optimal CDL interleaver cases using different system simulation configurations.

As a way to identify optimal interleaver cases for arbitrary system configurations, batch m-file **chancase.m** calculates row errors within the sink message array and records the maximum total. This operation is performed on identical source message arrays for all interleaver cases in a repetitive looping fashion, allowing **chancase.m** to determine which case or cases generate minimal symbol error totals along any given row. In this way, knowledge of the optimal interleaver case generating the corresponding maximum row error total dictates the required R-S coding strength. Recall that R-S FEC is performed by m-file **check.m** which corrects errors within a specified code block length along message array rows. A sample **chancase.m** configuration using 60 OFDM tones is presented in Table 23.

```
» chancase % Find all optimal interleaver cases for 60 OFDM tones using system model 2 link 3.

To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time version: 0
Enter the # of OFDM frequencies (Note: Must be even): 60
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 64
Enter specific integer seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 184
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter
0,1,2 or 3): 2
Channel model 2 simulation performed.
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 3
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10020
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
```

```
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10020], or [10020 1], offers no
interleaving functionality): [60 167]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Test all interleaver cases(yes) or specific ones(no)? (1 = yes, 0 = no): 1
All cases, 0 through 8, will be tested.
Enter the guard interval length (Number of sample points): 6
Do you want pictures? (1 = yes, 0 = no): 1
How many seconds of delay between pictures? 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 23. Configuring Batch M-file Chancase.m For Optimal Interleaver Cases

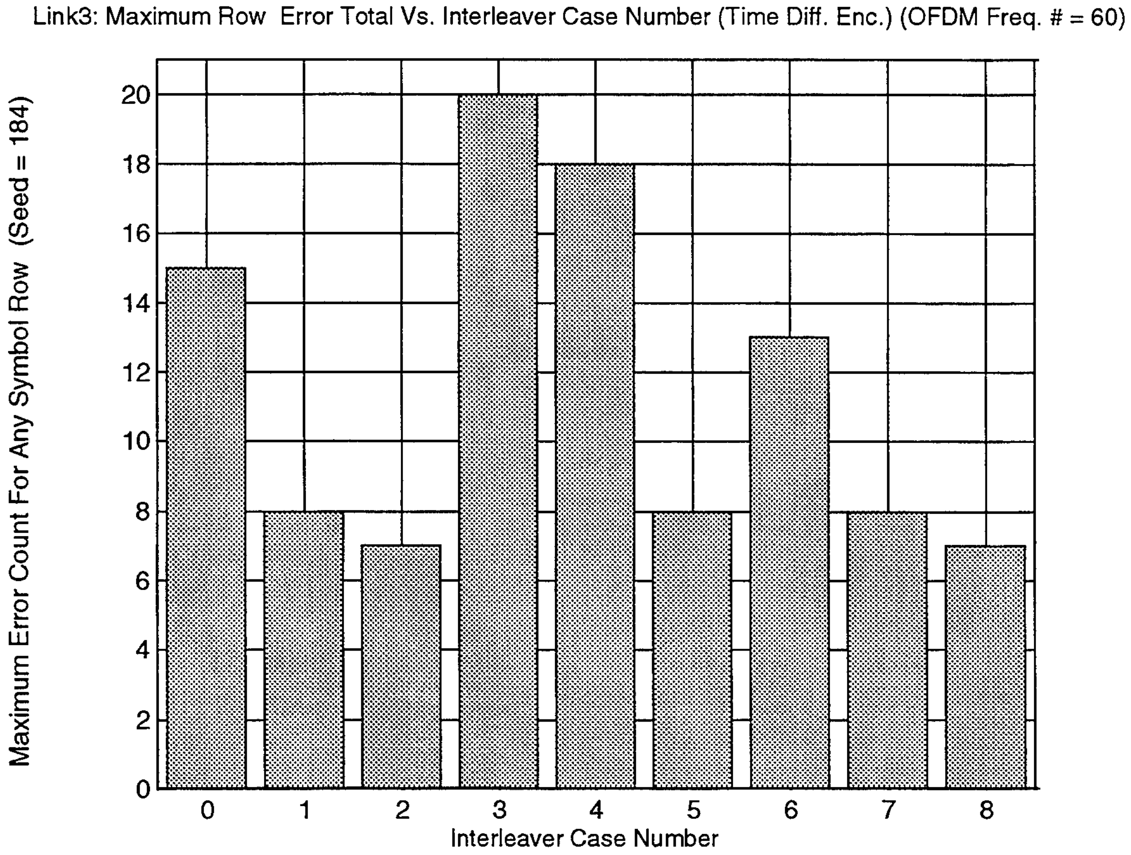


Fig. 79. Link 3 Row Error Totals Verse Interleaver Case, 60 OFDM Tones

The corresponding output plot from Table 23 system configuration is displayed in Fig. 79. The graphical summary of error totals indicate that the optimal interleaver case offering fewest number of errors in any given row are interleaver cases 2 and 8. Consequently, based upon this result case 2 or case 8 are included in later system model 3 simulations during interleaving configuration. From Fig. 79, it is also apparent that cases 3 and 4 offer the worst error total results; thus, they are avoided during subsequent system simulations.

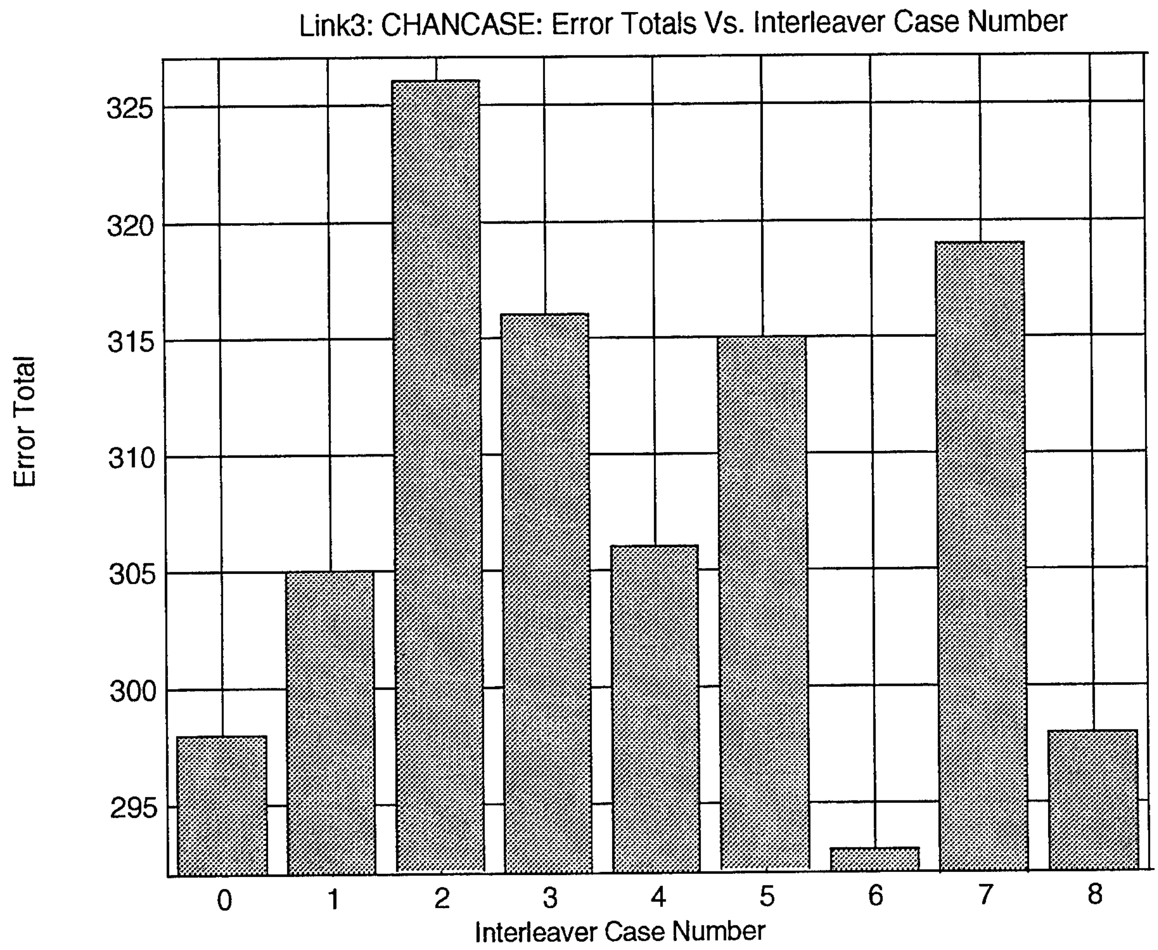


Fig. 80. Link 3 Total Message Errors Verse Interleaver Case, 60 OFDM Tones

Fig. 80 is interesting because it suggests that different interleaver configurations affect the message array error totals differently while using identical source messages. It might initially be hypothesized that a design error is present as error totals are expected to be the same for all tested interleaver cases, since system configurations are identical for all simulation loops aside from the particular interleaver case being tested. However, error total variations are understandable and expected when considering that the ordering of the transmitted message symbol sequence differ among the interleaver cases even though the identical message source is used (recall from Fig. 12 that multiple unique sequences are possible as a function of different interleaver parameters given the same source message array). With numerous possible transmit sequences a function of the interleaver case, the corresponding phases of the channel encoded symbols appropriately differ among symbols. In reaction to each transmitted symbol, the channel randomly alters symbol magnitudes and phases according to the simulated link multipath parameters; thus, certain symbols may be more susceptible to channel influences than others and may be decoded as errors, when previously the same symbols were located in a different part of a transmitted sequence and may not have been affected by the channel at all. Thus, depending on their transmit sequence location, certain symbols are more prone to multipath burst error corruption than others and this accounts for the variation.

Based upon this understanding, it is interesting to note that certain interleaver cases may generate more total message errors, as case 7 does in Fig. 80. However, due to the manner in which case 7 interleaving is performed, it also offers one of the better interleaving choices in terms of minimizing row errors (Fig. 79).

Fig. 81 demonstrates similar **chancase.m** link 3 simulation results configured for 240 OFDM tones, frequency differential encoding/decoding and a source message size of 10,080 total OFDM symbols, while Fig. 82 indicate message errors totals verse interleaver cases. From Fig. 81 It is apparent that interleaver case 7 minimizes message row error totals, with cases 1 and 8 a close second.

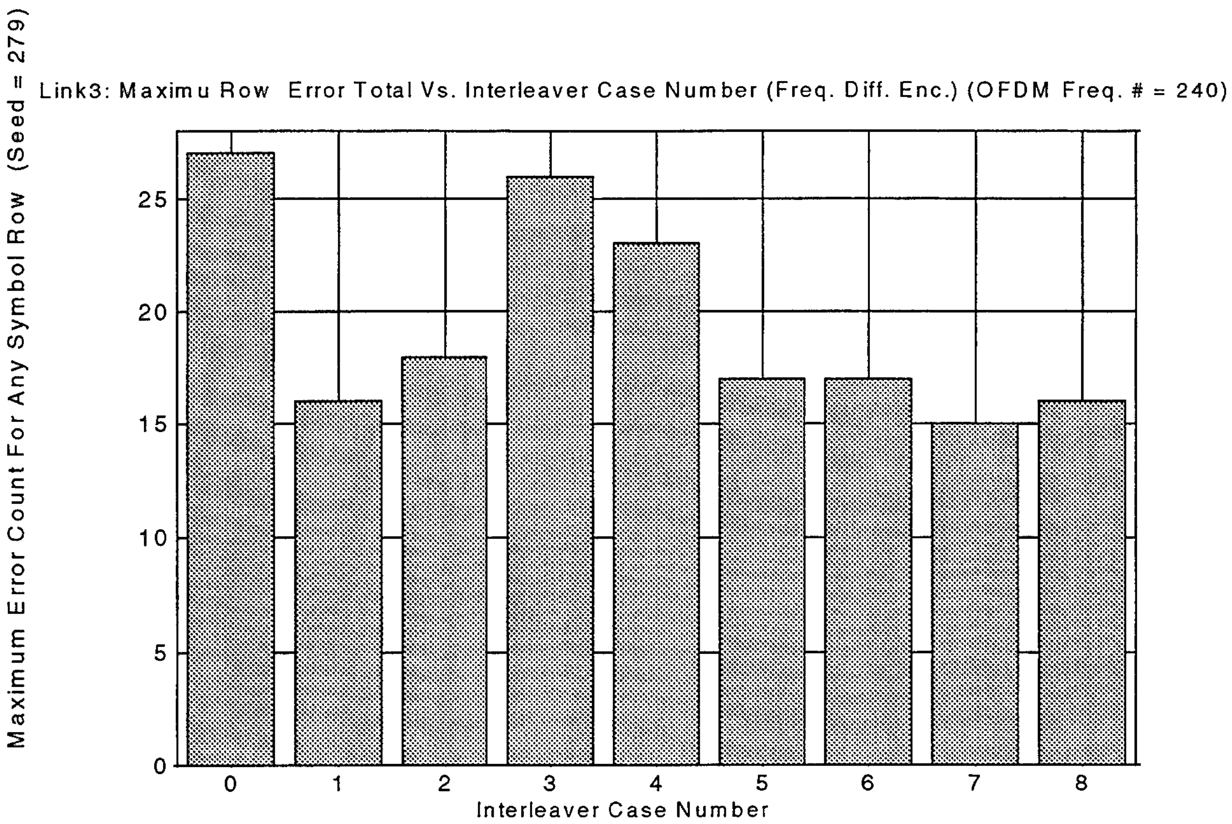


Fig. 81. Link 3 Row Error Totals Verse Interleaver Case, 240 OFDM Tones

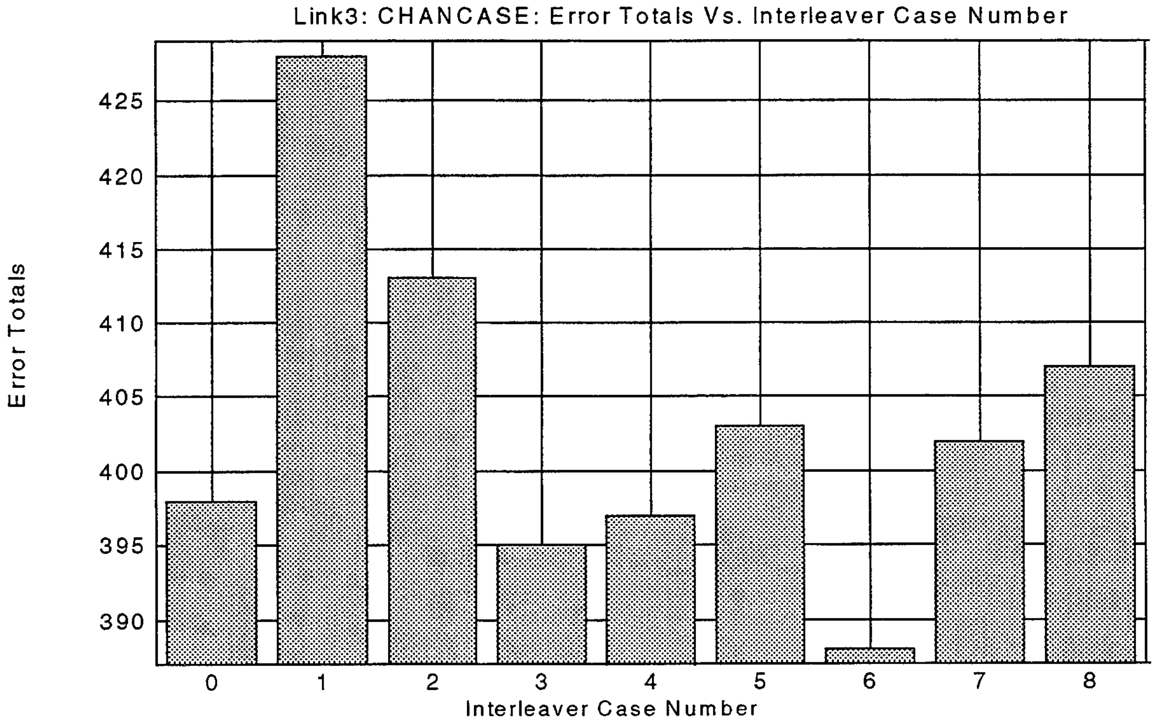


Fig. 82. Total Errors Verse Interleaver Case Number, 240 OFDM Tones, Link 3

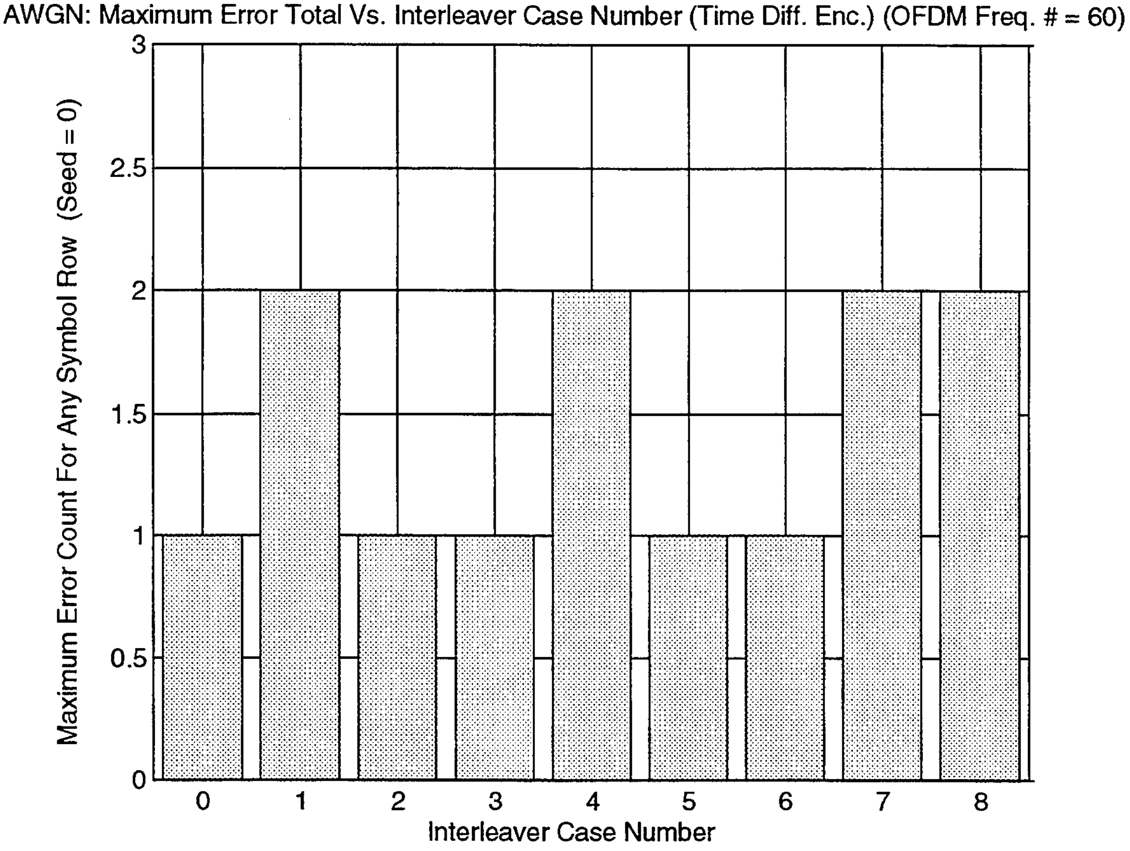


Fig. 83. AWGN: Row Error Totals Verse Interleaver Case, 60 OFDM Tones

Test phase 5 is oriented around system model 2 simulations only since interleaving is most effective on multipath induced burst noise and not as effective with the AWGN channel having a more random error distribution. From the example graph depicted in Fig. 83 demonstrating a **chancase.m** simulation result using system model 1 only, it is apparent that symbol interleaving using different cases has negligible benefit with AWGN as expected since there is little variation in the row error totals. From the figure, the difference in row error totals among the various interleaver cases is merely one error.

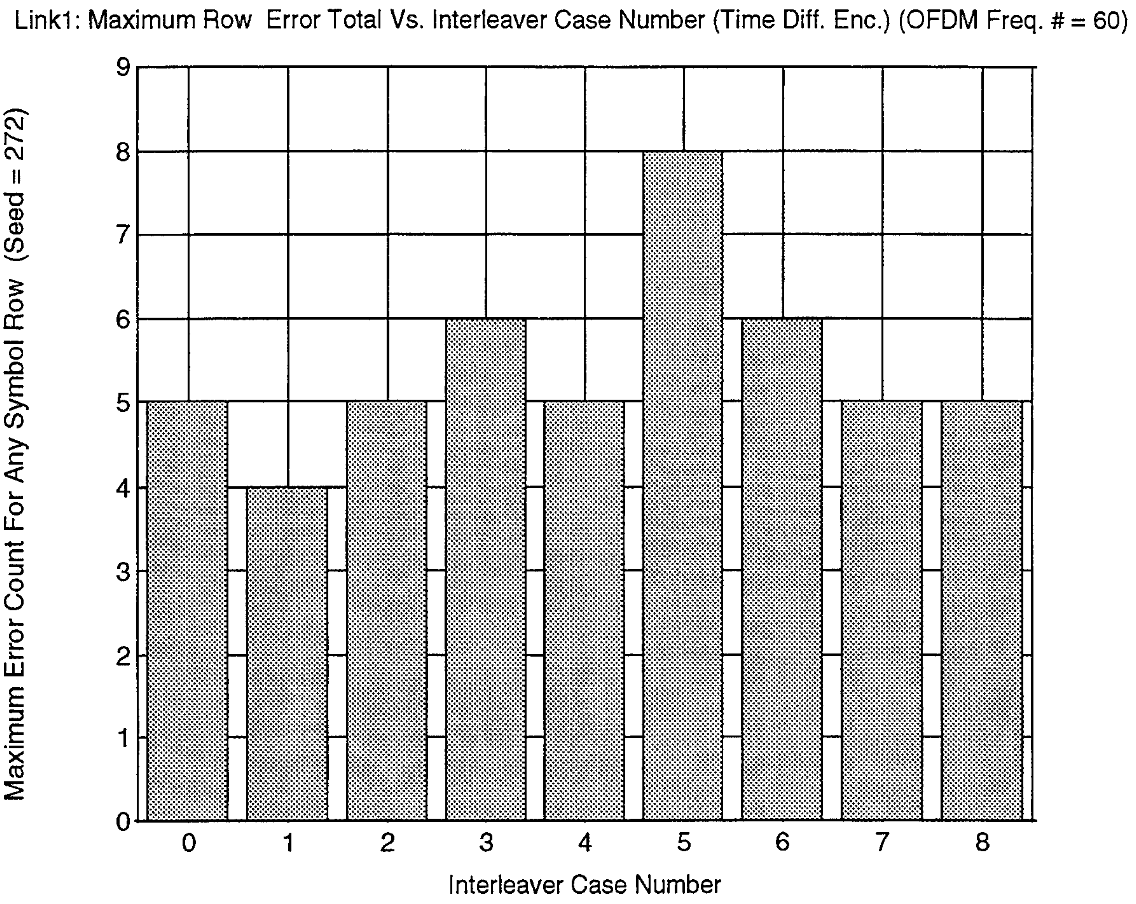


Fig. 84. Link 1 Row Error Totals Verse Interleaver Case, 60 OFDM Tones

With identification of optimal interleaver cases for link 3 using 60 and 240 OFDM tones, it is desirable to perform similar **chancase.m** batch file simulations for links 1 and 2 while using corresponding worst case channel seeds. From Fig. 84, interleaver case 1 performs the best using link 1 and 60 OFDM tones. Likewise from Fig. 85, interleaver case 4 performs the best for link 2 and 60 OFDM tones. Similar simulations are again performed on links 1 and 2 using 240 OFDM tones with corresponding graphical results presented in Fig. 86 and Fig. 87. For link 1, cases 0,1 and 2 perform equally as well. For link 2, case 0 (conventional block interleaver) performs the best with (240,42) intermediate matrix dimensions.

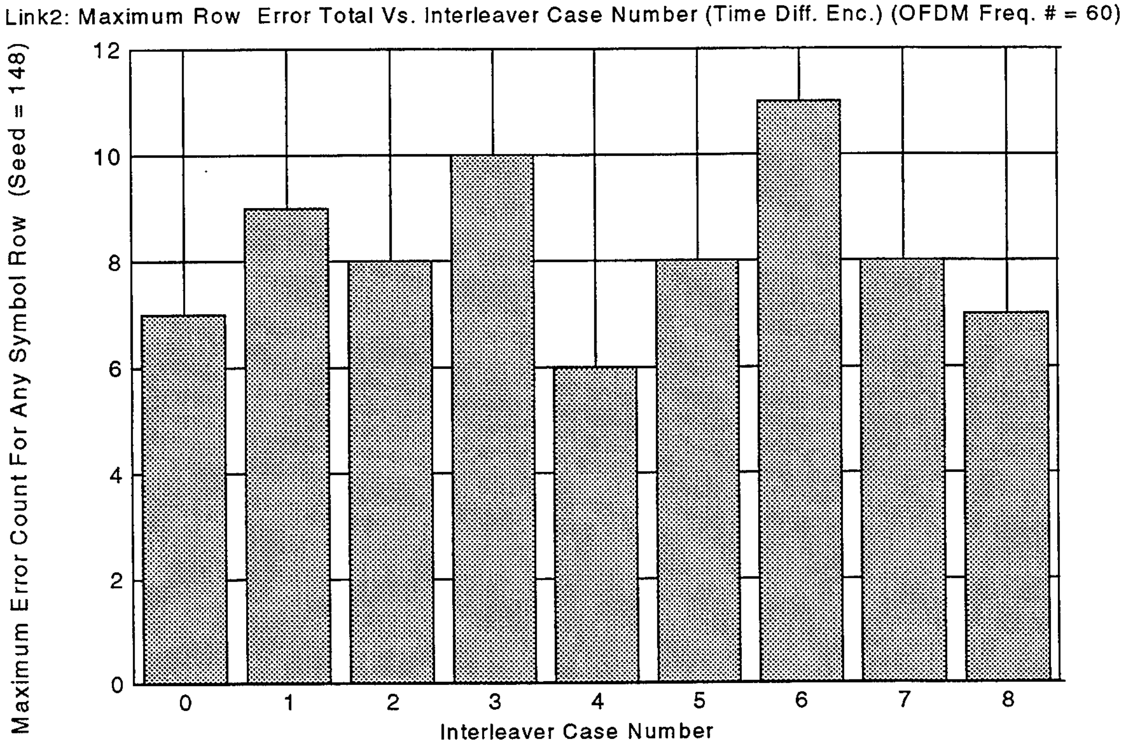


Fig. 85. Link 2 Row Error Totals Verse Interleaver Case, 60 OFDM Tones

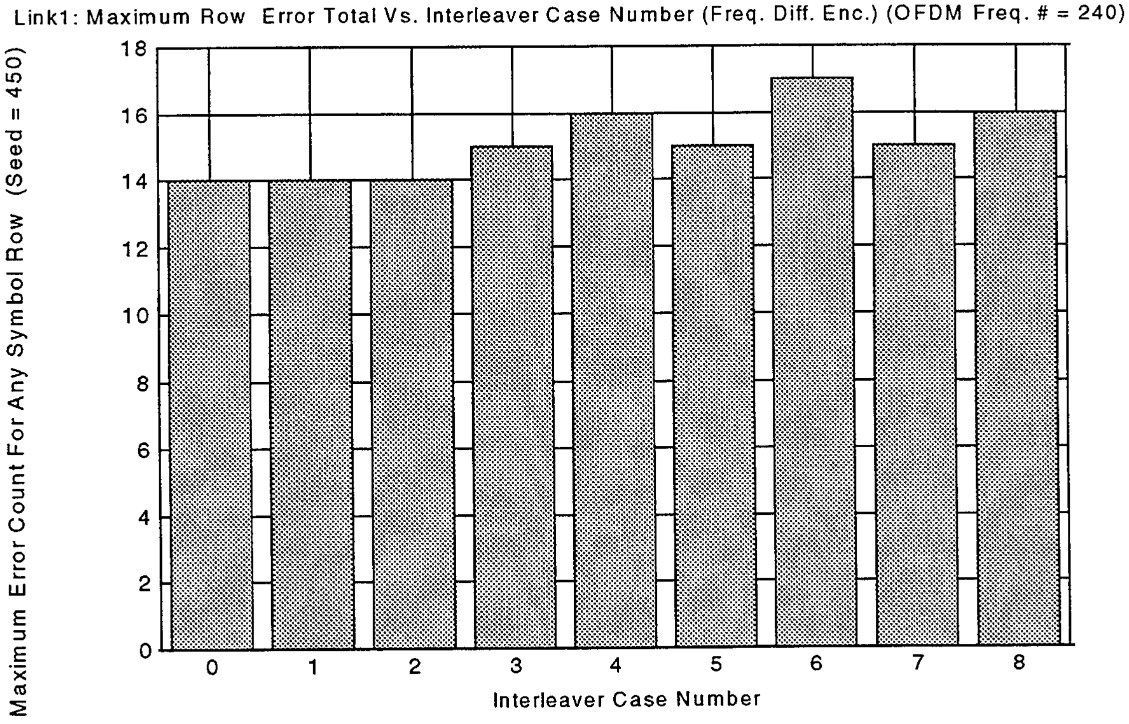


Fig. 86. Link 1 Row Error Totals Verse Interleaver Case, 240 OFDM Tones

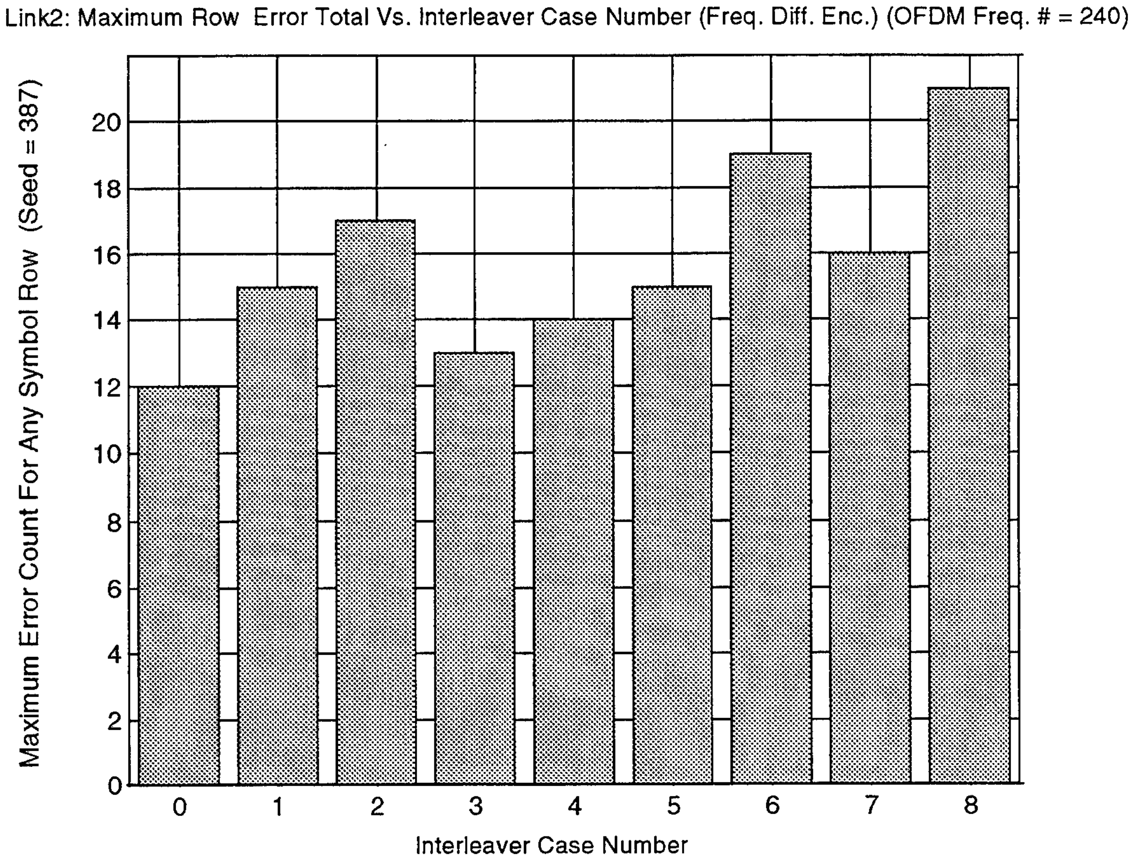


Fig. 87. Link 2 Row Error Totals Verse Interleaver Case, 240 OFDM Tones

Conclusion of test phase 5 simulations successfully revealed preferable interleaver cases for inclusion in subsequent system trials. System testing now progresses to phase 6, the final test step, allowing complete system model 3 simulations using optimal input configurations. The test phases conducted up to this point are included to enhance the overall COFDM system for best performance during system model 3 testing while using worst case channel 2 models as determined by the seed selection.

G. TEST PHASE 6 - FINAL SYSTEM MODEL 3 SIMULATIONS

The culmination of system design and emulation, m-file macro development, batch m-file creation and test phases 1 through 5 conclude with the final test phase 6 simulation trials. During this comprehensive test step, complete system model 3 simulations are performed using the channel 3 model (AWGN and multipath) with appropriate R-S coding, generating corresponding system performance curves. Judicious selection of practical R-S coding is necessary to ensure reasonable performance curves comparable to the theoretical uncoded AWGN curves of Fig. 40.

Up to this point, the ideal system is configured for either 60 OFDM tones using time differential encoding/decoding or 240 OFDM tones using frequency differential encoding/decoding. Total source message OFDM symbol sizes are either 10,020 for 60 tones or 10,080 for 240 tones and are within the 10,000 +1% symbol quantity range. For 60 OFDM tones using link 3, optimal interleaver cases 2 and 8 are preferred, since they demonstrate superior performance during the last testing phase. For 240 OFDM tones using link 3, optimal interleaver case 7 is preferred. All system model 3 simulations use 16-PSK modulation scheme.

A sample system model 3 simulation configuration using batch m-file **cofdmsim.m** is presented in Table 24 with 60 OFDM tones. For this example the most challenging link 3 is included, along with a noise sigma parameter range of from 0 to 0.02 (recall that the sigma parameter sets the desired AWGN power, N_o). The batch file outputs are performance curves similar to the ones presented during test phase 1; however, typically the performance is greatly degraded from AWGN theoretical curves due to the added multipath influences. Appropriately, R-S FEC is included to improve overall performance within acceptable code rate constraints. From the example in Table 24, the R-S code is capable of correcting any 12 symbol errors appearing in a code block length of 240 symbols. With the n and k parameters chosen as such, the code rate is calculated to be 0.90. Furthermore, an additional overhead loss of 9.3% from the inclusion of a 6 sample point guard interval precursor with 64 FFT points (6/64 = 0.093) reduces the effective information rate to approximately 0.80 or 80%. Recall from earlier calculations that a code rate of 0.80 applied to the system reduces the information bit rate from the robustly designed 1.92 Mbps to the stated objective of 1.536 Mbps.

```
» cofdmsim

This batch m-file runs COFDM simulations using different channel models.
To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or to run both
enter 2 (two): 0
Enter the # of OFDM frequencies (Note: Must be even): 60
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 64
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter
0,1,2 or 3): 3
Channel model 3 simulation performed.
Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003):
[linspace(0,0.015,20),linspace(0.015,0.02,20)]
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 3
Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): 0
Enter specific interleaver case numbers from (0 to 8) (Ex. [0 4 5 8]): 2
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10020
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 =
yes, 0 = no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10020], or [10020 1],
offers no interleaving functionality): [60 167]
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Enter the guard interval length (Number of sample points): 6
Do you want to include error correction coding? (1 = yes, 0 = no): 1
Enter n,k and error correction block length (Ex. [240 200 240]): [240 216 240]
Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 184
Do you want signal plots? (1 = yes, 0 = no): 1
How many seconds of delay between pictures? 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 24. System Model 3 Simulation Using Batch M-file Cofdmsim.m, 60 Tones

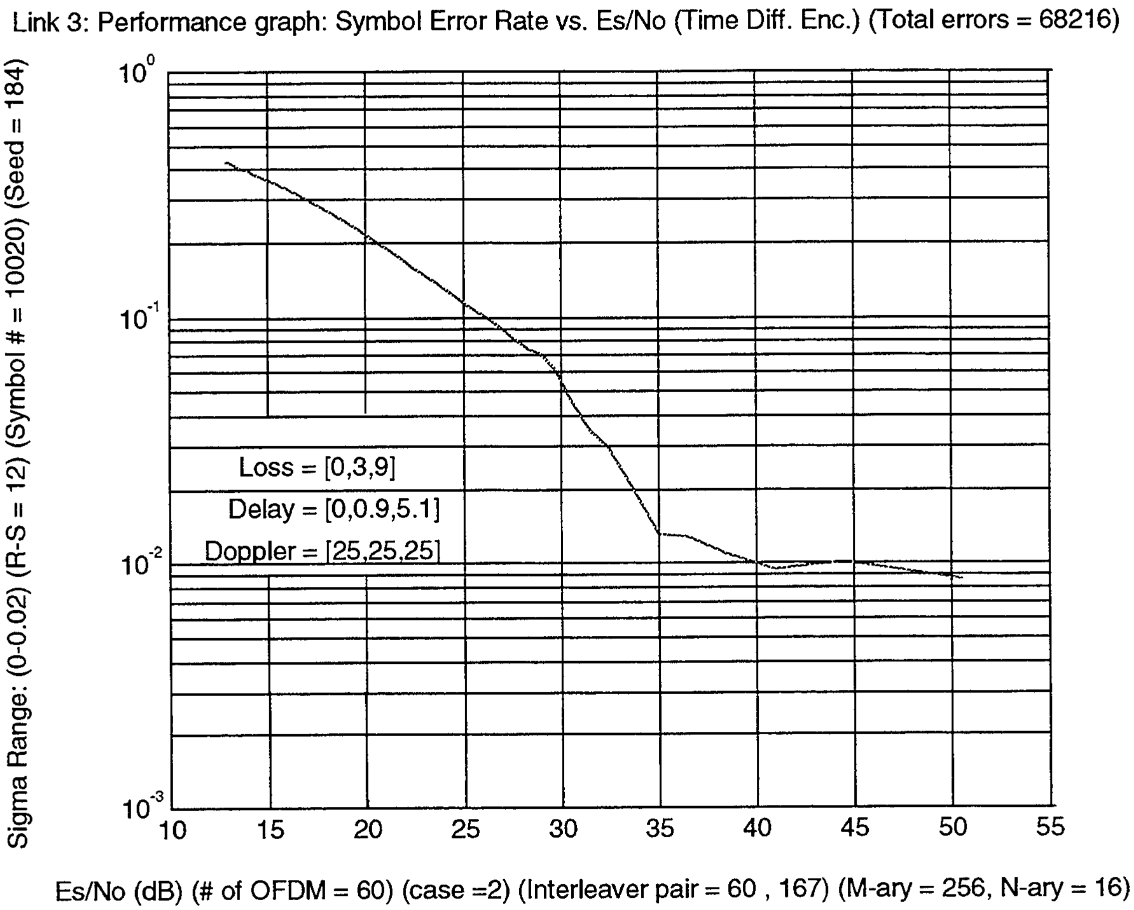


Fig. 88. Link 3 System Model 3 Simulation Using 60 OFDM Tones, 0.9 Code Rate

The corresponding performance curve of the configured simulation in Table 24 is presented in Fig. 88. From the plot and in comparison to Fig. 40, it is apparent that using a worst case multipath channel as part of the simulation (seed 184) causes numerous message symbol errors greatly affecting performance. The dismal performance generated by the configured simulation is unacceptable since multipath induced errors occur even with zero AWGN. Since this simulation result is considered a failure, the system configuration using 60 OFDM tones must be modified to strengthen and improve performance.

A second system model 3 simulation is conducted similar to Table 24; however, 240 OFDM tones, frequency differential encoding/decoding and 10,080 OFDM symbols are used instead. The corresponding worst case channel 2 link 3 seed is 279. The exact simulation configuration is presented in Table 25. During this trial, case 7 interleaving is performed since it is optimal in terms of phase 5 test results. A 6 sample point precursor is once again used using 256 FFT points, causing an additional 2.3% of overhead (6/256 = 0.023), which is substantially less guard interval overhead than the 60 tone simulation. In consideration of the reduced guard interval overhead, a R-S code rate of 0.825 is now possible, allowing for an acceptable total overhead of approximately 0.20. Once again, this 0.80 efficiency permits a 1.536 Mbps information bit rate.

```
» cofdmsim
_________________________________________________________________________
This batch m-file runs COFDM simulations using different channel models.
To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or to run both enter 2
(two): 1
Enter the # of OFDM frequencies (Note: Must be even): 240
Enter the number of FFT points (Note: This number must be larger than # of OFDM frequencies): 256
Do you want to run channel model 0, channel model 1, channel model 2 or channel model 3? (Enter 0,1,2
or 3): 3
Channel model 3 simulation performed.
Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003): [
linspace(0,0.005,20), linspace(0.005,0.02,20)]
Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): 3
Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): 0
Enter specific interleaver case numbers from (0 to 8) (Ex. [0 4 5 8]): 7
Enter the total minimum number of symbols to simulate (Ex. 10000): 10000
Note: Based on the parameters thus far, the actual total number of symbols to be simulated will be: 10080
For the  interleaver, do you want to calculate all possible intermediate matrix dimension pairs? (1 = yes, 0
= no): 0
Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering [1 10080], or [10080 1], offers no
interleaving functionality): [240 42]
```

```
Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): 8
Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): 4
Enter the guard interval length (Number of sample points): 6
Do you want to include error correction coding? (1 = yes, 0 = no): 1
Enter n,k and error correction block length (Ex. [240 200 240]): [240 198 240]
Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): 279
Do you want signal plots? (1 = yes, 0 = no): 0
Do you want print outs? (1 = yes, 0 = no): 0
```

Table 25. System Model 3 Simulation Using Batch M-file Cofdmsim.m, 240 Tones

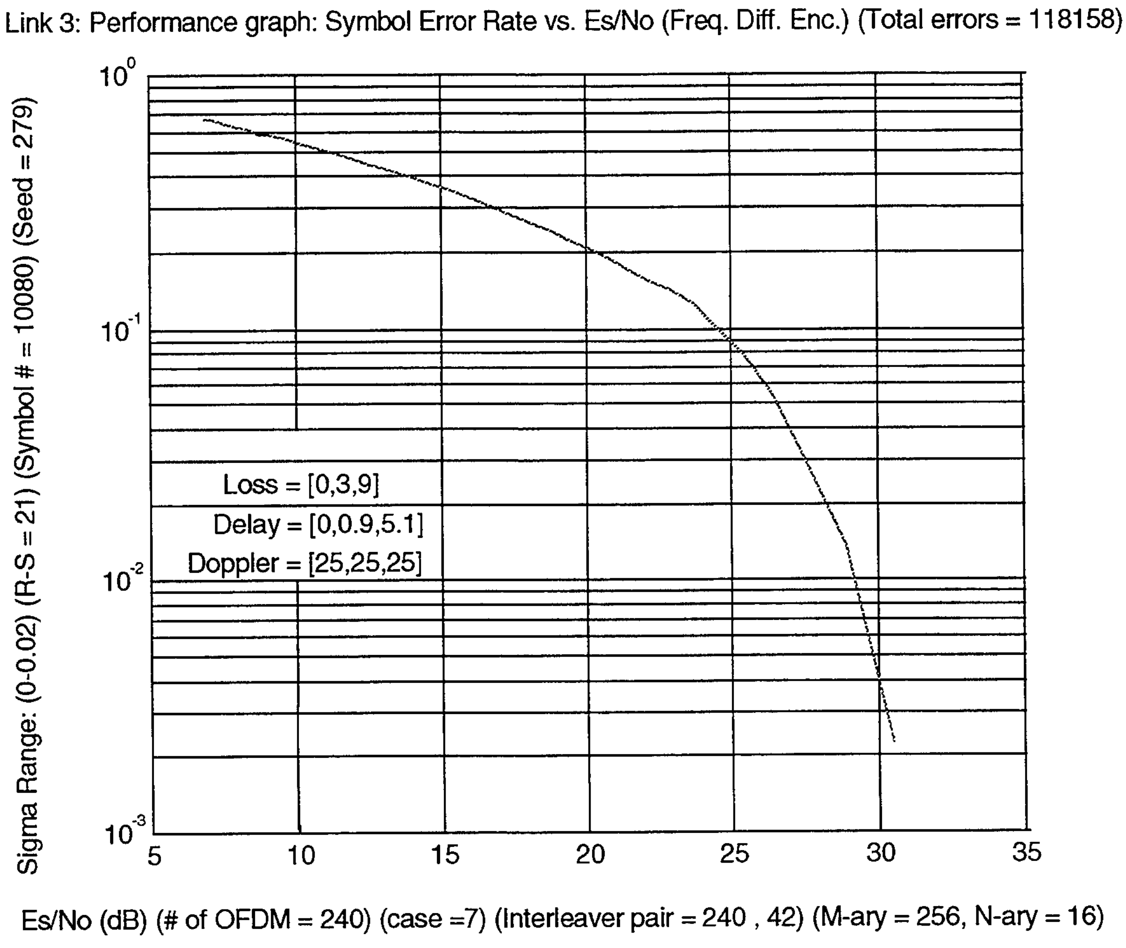


Fig. 89. Link 3 System Model 3 Simulation Using 240 OFDM Tones, 0.825 Code Rate

The corresponding Table 25 simulation performance results are displayed in Fig. 89, demonstrating substantial noticeable improvement over Fig. 88 and 60 tones. As expected, the increased R-S coding strength improves the SER greatly over the previous simulation. Observation of the ${}^{Es}/_{No}$ at a SER of 10^{-2} indicates a 29 dB reading. This is approximately 5.5 dB worse than theoretical for AWGN alone (Fig. 40). It seems that stronger R-S coding is required to improve performance further; however, since we are now at the threshold of acceptable overhead percentages, additional R-S coding causes a reduction of the 1.536 information bit rate objective.

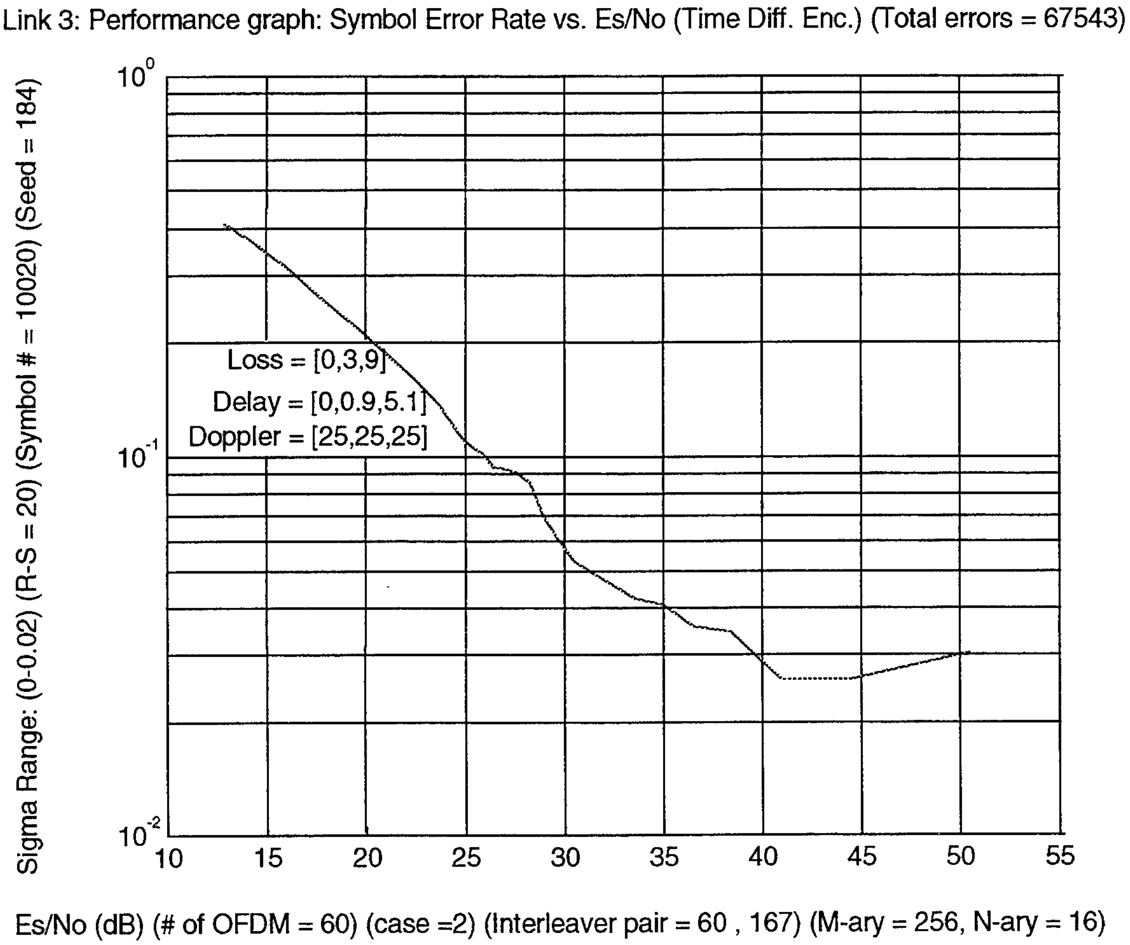


Fig. 90. Link 3 System Model 3 Simulation Using 60 OFDM Tones, 0.83 Code Rate

Another 60 OFDM tone simulation is considered; however, a reduced guard interval of 2 sample points is included producing 3.1% guard interval overhead when using 64 FFT points (2/64 = 0.031). This modification allows inclusion of a stronger R-S code with a 0.83 code rate, permitting 20 symbol error correction within a code block length of 240 symbols. The results of the modified simulation configuration is presented in Fig. 90. Again the simulation result is a failure as there are numerous uncorrected symbol errors occurring even without AWGN. This simulation, when compared to Fig. 88 demonstrates worse performance with a shortened guard interval and stronger R-S code.

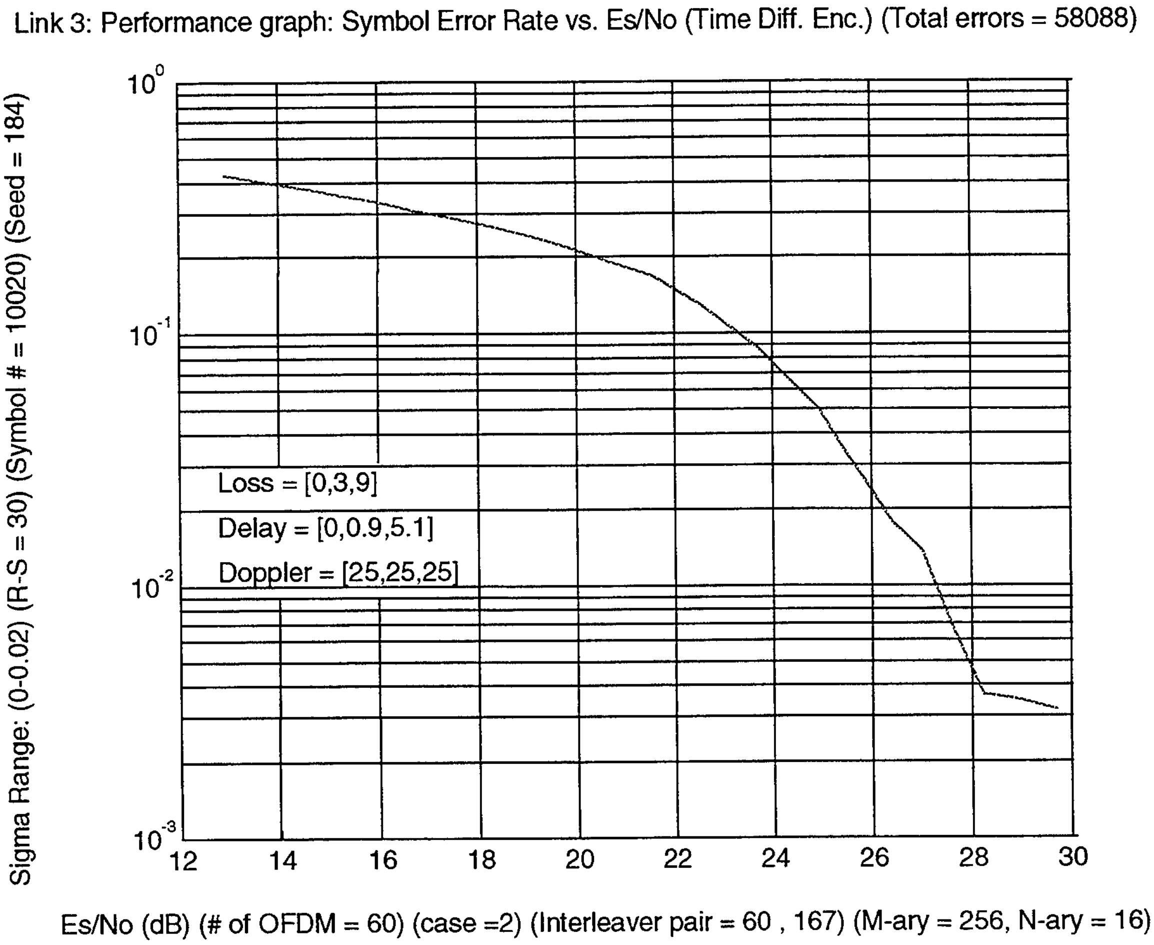


Fig. 91. System Model 3 Simulation Using 60 OFDM Tones, Extra R-S Coding

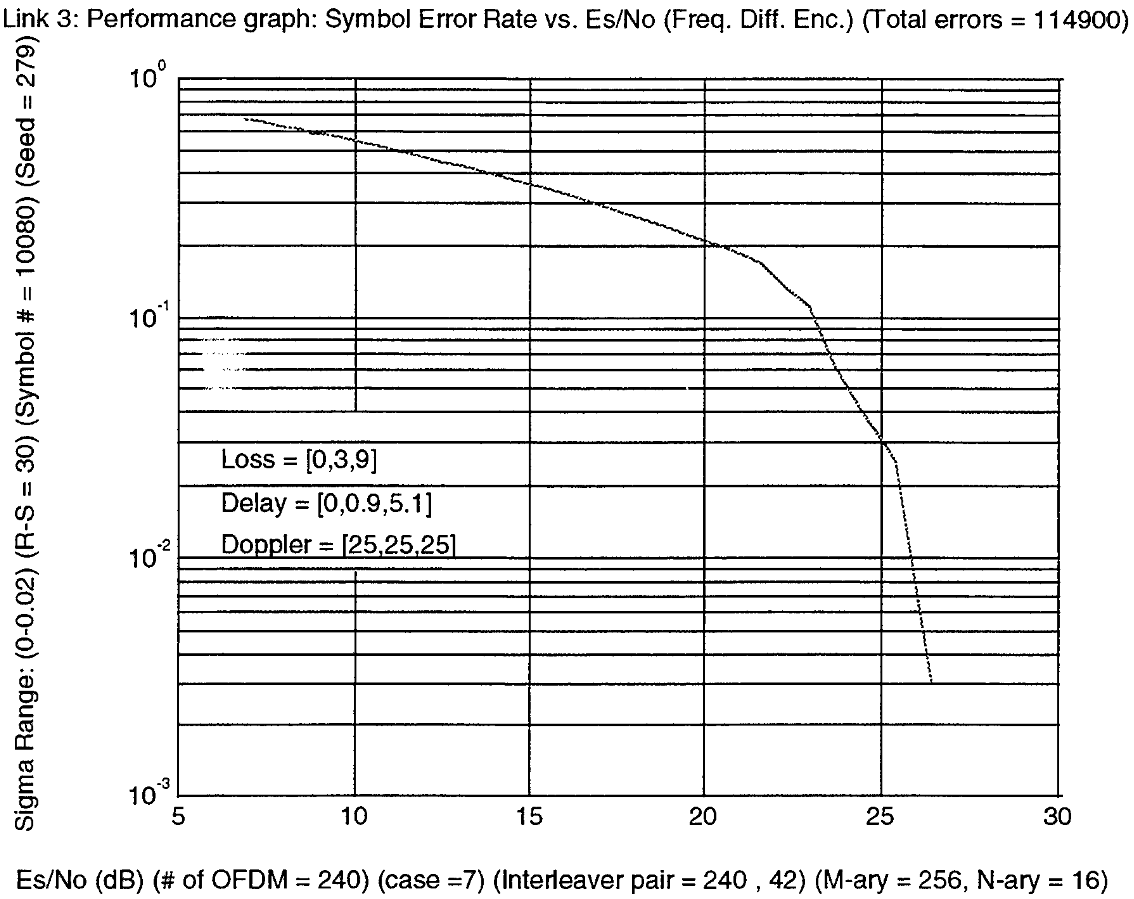


Fig. 92. Link 3 System Model 3 Simulation Using 240 OFDM Tones, 0.75 Code Rate

Initial indications suggest that 240 OFDM tones perform better than 60 OFDM tones with all other considerations equivalent and while transmitting at the 1.536 Mbps information rate. It is instructive to learn how much added robustness must be configured into the system to improve overall performance for both 240 and 60 tones to reach acceptable levels close to theoretical AWGN. With this in mind, additional simulations are conducted increasing the R-S coding until reasonable performance is attained. Fig. 91 and Fig. 92 demonstrate the results for 60 OFDM tones and 240 OFDM tones, respectively. Clearly, 240 OFDM tones perform better than 60 OFDM tones by more than 1dB at a SER of 10^{-2} with identical code rates of 0.75 and guard interval lengths of 6 sample points, although both are about 3dB worse than theoretical in AWGN even with additional FEC.

Computation of the effective information rates for 240 tones and 60 tones using these code strengths results in 1.38 Mbps and 1.26 Mbps respectively. Thus, improved performance is gained at the expense of reduced information bit rates.

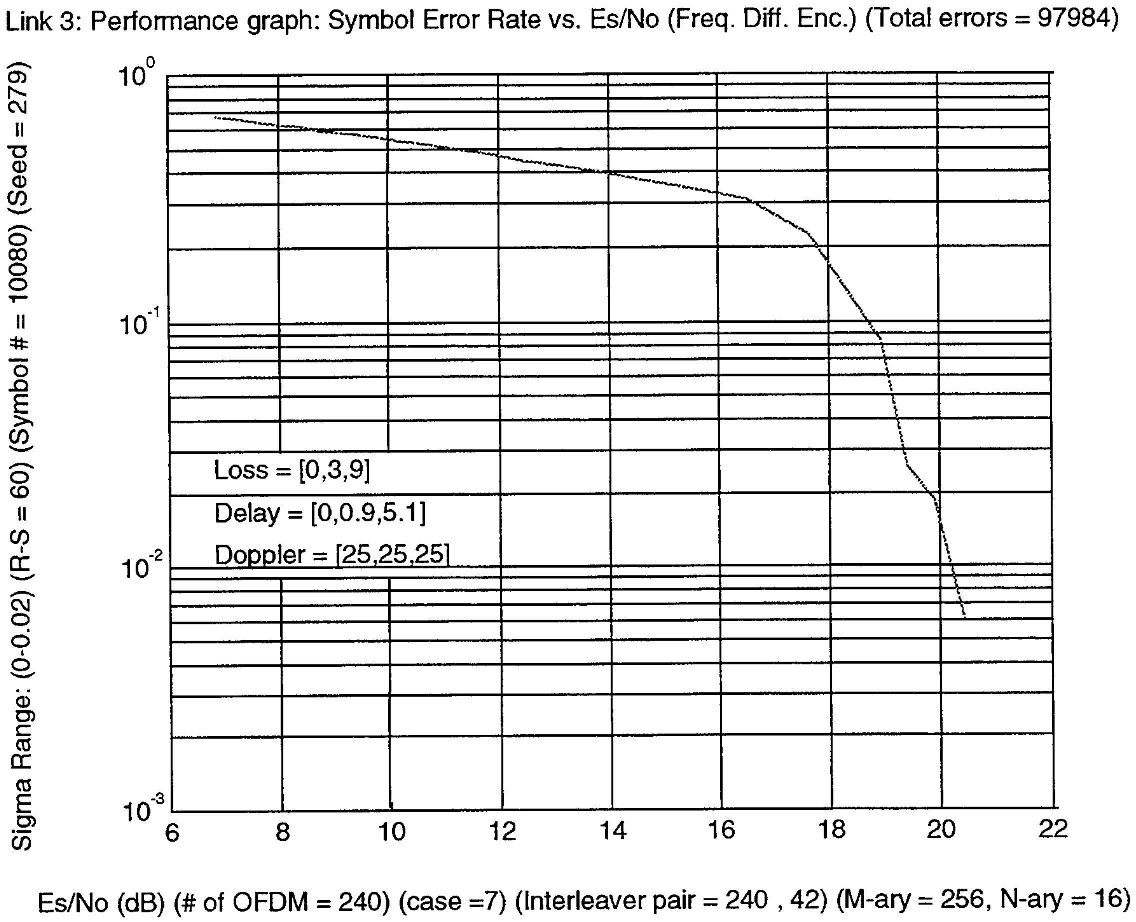


Fig. 93. Link 3 System Model 3 Simulation Using 240 OFDM Tones, 0.5 Code Rate

Fig. 93 and Fig. 94 display the results of additional simulation trials using a 0.5 code rate, including both 240 and 60 OFDM tones. Once again 240 OFDM tones performs better than 60 OFDM tones with remaining configured inputs identical. With the heavy R-S coding, Fig. 93 indicates performance almost 4 dB better than theoretical AWGN. The 60 OFDM tone performance curve also indicates improvement over theoretical by about 2 dB; however, considering the additional guard interval overhead required while using 60 tones, the 240 tone configuration is superior with respect to performance and information bit rate.

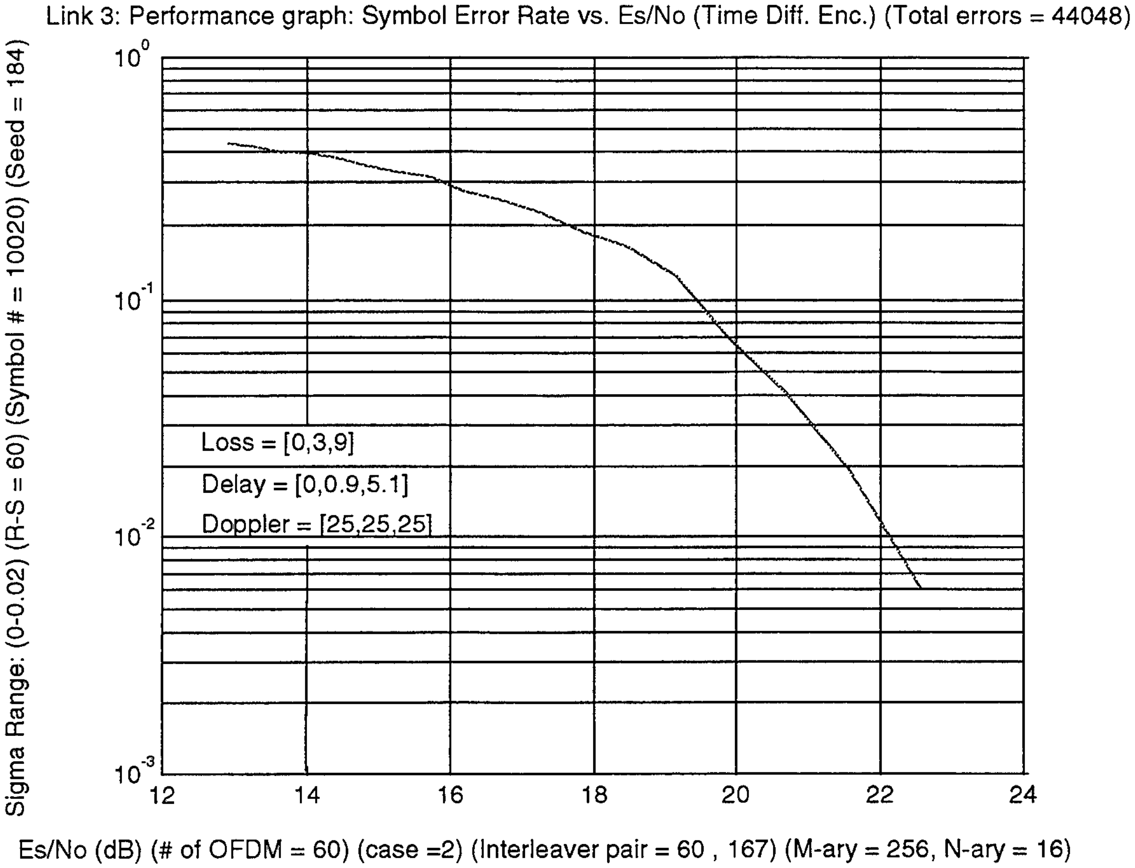


Fig. 94. Link 3 System Model 3 Simulation Using 60 OFDM Tones, 0.5 Code Rate

Calculations of overall information bit rate for 240 tones accounting for guard interval and coding overhead result in a 0.915 Mbps bit rate. For 60 OFDM tones the effective information rate reduces substantially to 0.78 Mbps. While there is dramatic improvement of performance using a code rate of 0.5, the desired information bit rate of 1.536 Mbps is no longer attainable. Furthermore, using a 0.5 code rate is similar to a system configured for QPSK (4-ary) without any coding, with respect to equivalent information bit rates.

As previously mentioned and further supported by the link 3 simulation output data, the COFDM based communication system oriented around 240 OFDM frequency tones offers superior performance over the alternate configuration using 60 OFDM tones. Having isolated 240 tones as the preferred choice, link2 and link1 system model 3 simulations are also performed each using the respective worst case channel model 2 seeds, corresponding optimal interleaver cases derived from test phase 3 and phase 5 and preferred R-S code rates.

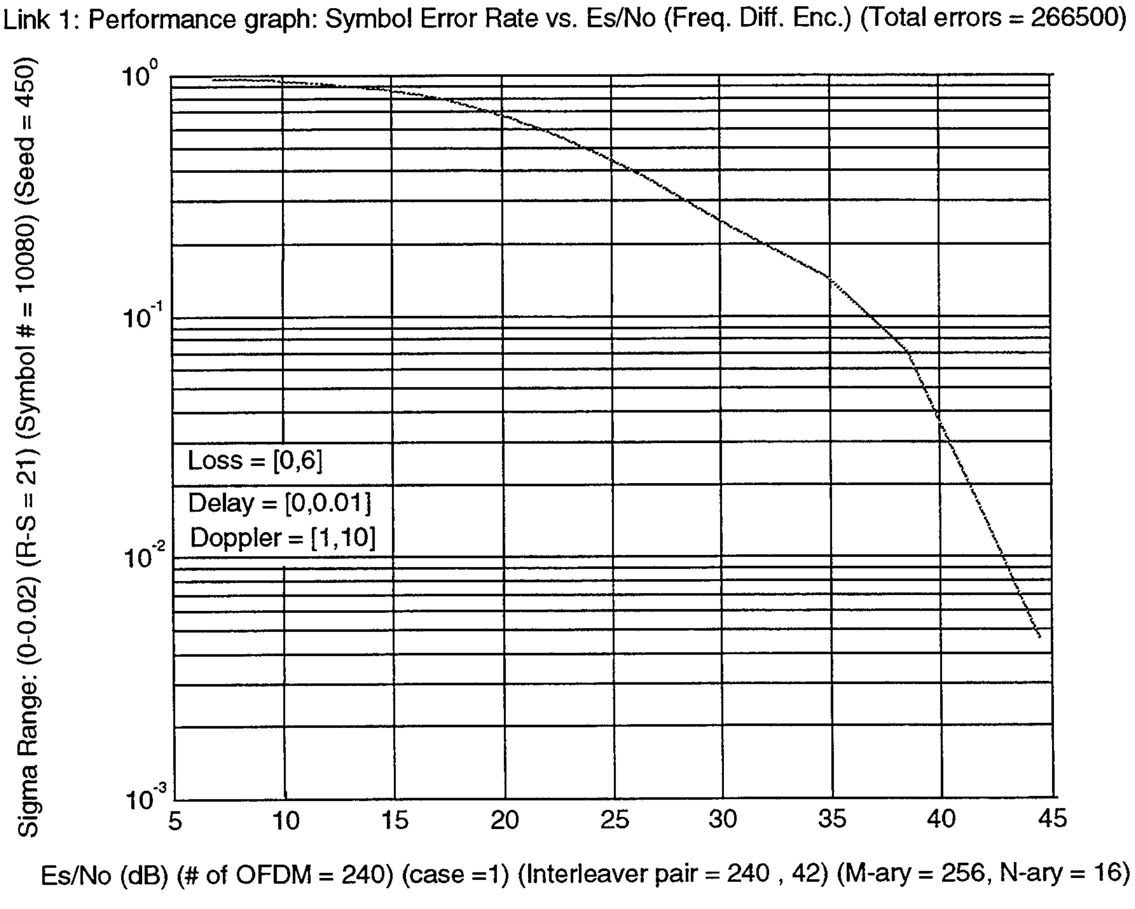


Fig. 95. Link 1 System Model 3 Simulation Using 240 OFDM Tones, 0.825 Code Rate

While Fig. 95 presents a link 1 performance result using 240 tones and Fig. 96 presents a link 2 result with 240 tones, numerous other link 1 and 2 simulation trials were performed also using 60 OFDM tones; however, as expected those performance results were consistently worse than the equivalent 240 tone results. Consequently, only the useful 240 tone performance curves are included in this presentation.

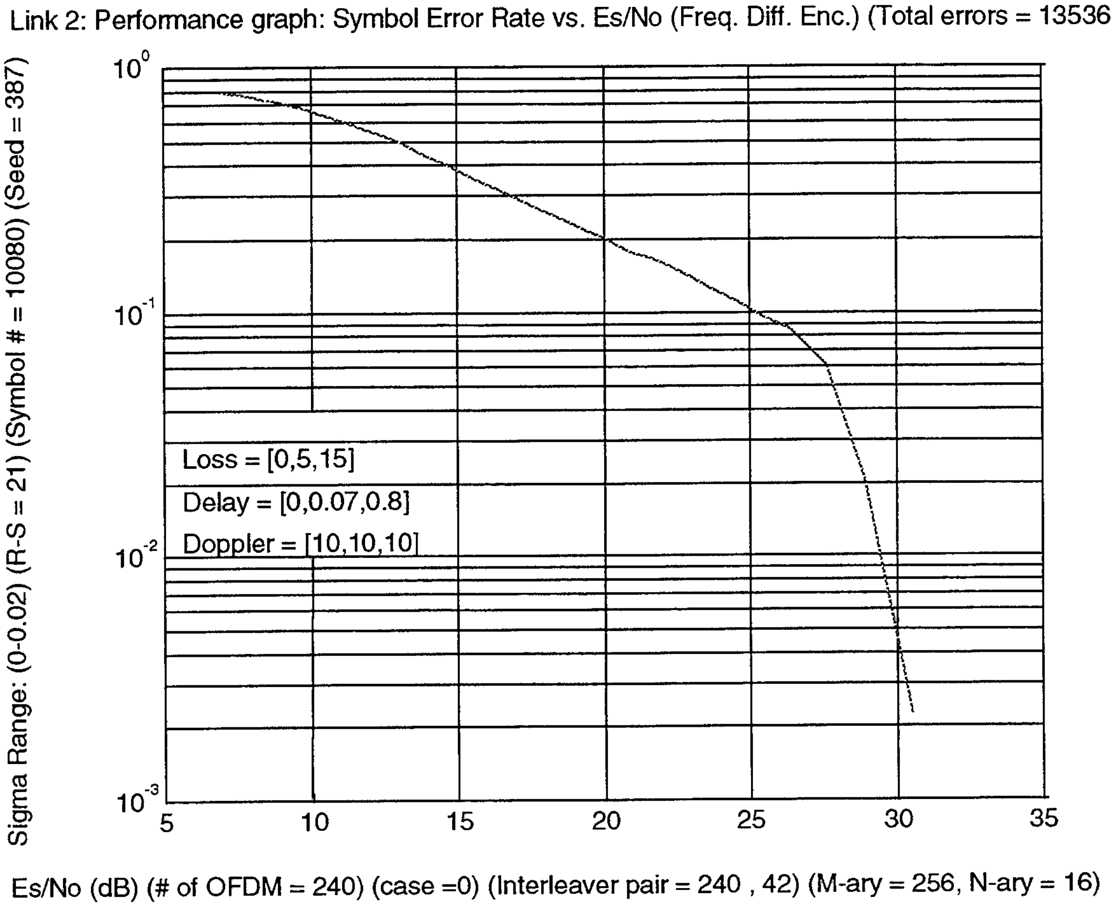


Fig. 96. Link 2 System Model 3 Simulation Using 240 OFDM Tones, 0.825 Code Rate

From Fig. 95, link 1 performance is very poor. The $^{Es}/_{No}$ reading of 42 dB at a SER of 10^{-2} is almost 20 dB worse than theoretical AWGN! (Fig. 40). This configuration using the worst case channel model 2 link 1 seed of 450 is unsatisfactory, requiring considerably extra margin. Again, an overall 0.80 transmission efficiency is incorporated into the simulation including code rate overhead and guard interval overhead, allowing for the 1.536 Mbps required bit rate. Clearly, stronger R-S coding is necessary to improve performance, adversely affecting the final information bit rate. Link 2 performance (Fig.96) demonstrates substantial improvement over link 1 (Fig 95) and comparable performance with link 3 (Fig. 89) at a SER of 10^{-2} with all other relevant input configurations equal. Apparently link 1 is the most challenging for a 240 OFDM tone based COFDM communication system with worst case channel conditions.

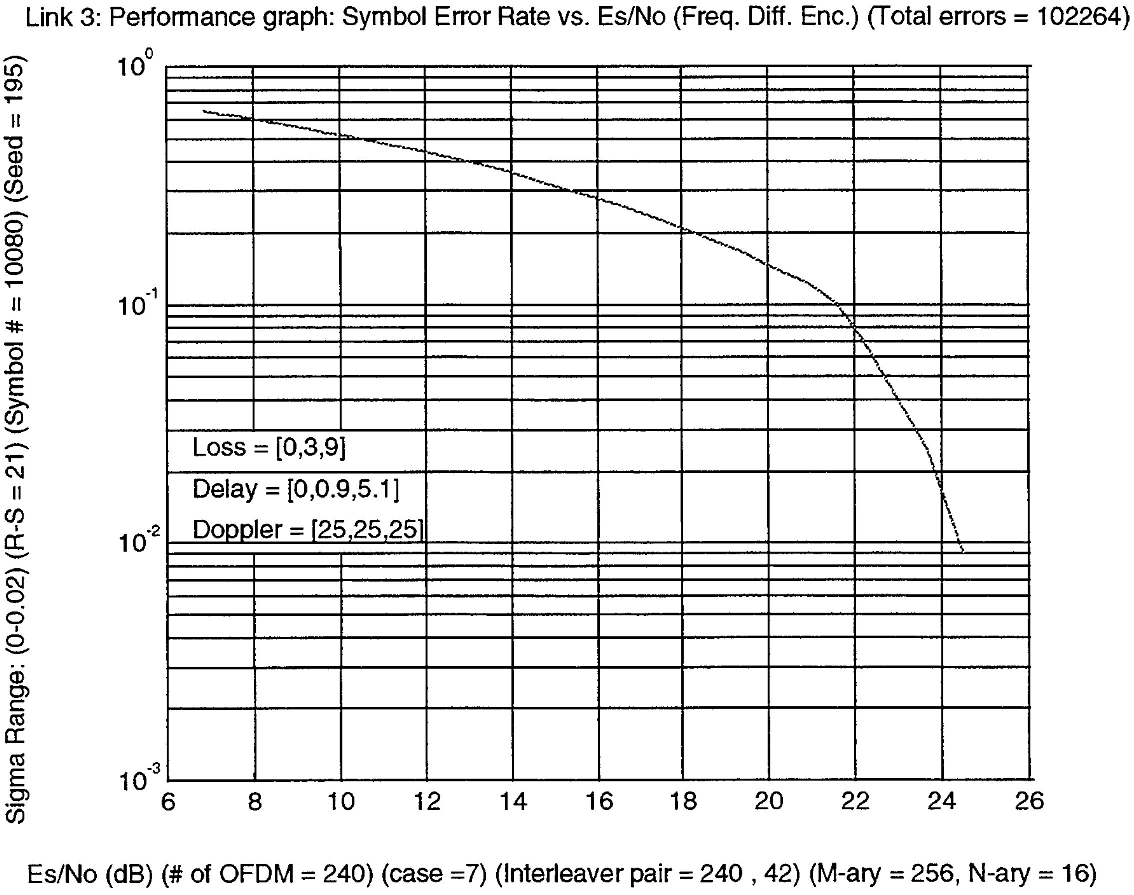


Fig. 97. Link 3 System Model 3 Simulation Using 240 OFDM Tones, Average Seed

It is important to be mindful of the fact that all test phase 6 performance simulations conducted up to this point include the worst case channel conditions with respect to multipath distortions as a result of using worst case channel 2 model seeds. For these multiple link simulations, use of the worst seed correlates with maximum sink message error totals resulting from severe channel distortions occurring less than 1% of the time. This percentage is based upon the discussions presented during test phase 3 and the fact that a single "bad" seed out of a total of 500 tested seeds is included in the simulation. Consequently, these simulation results reflect extreme operating conditions and do not necessarily indicate the typical expected channel performance. Thus, additional system model 3 simulations are performed using links 1 through 3 with 240 OFDM tones; however, this time seed values corresponding to average sink message error rates for channel model 2 conditions are included instead of the worst channel seeds.

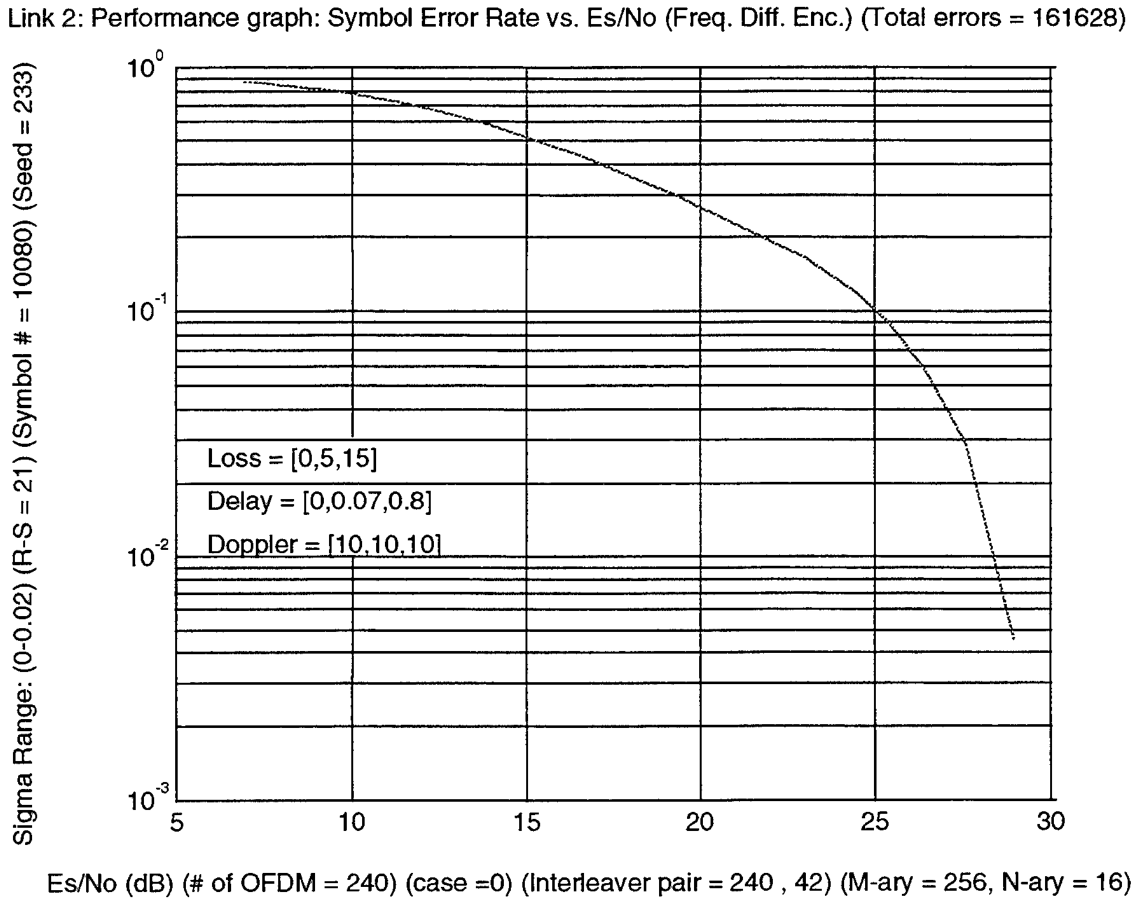


Fig. 98. Link 2 System Model 3 Simulation Using 240 OFDM Tones, Average Seed

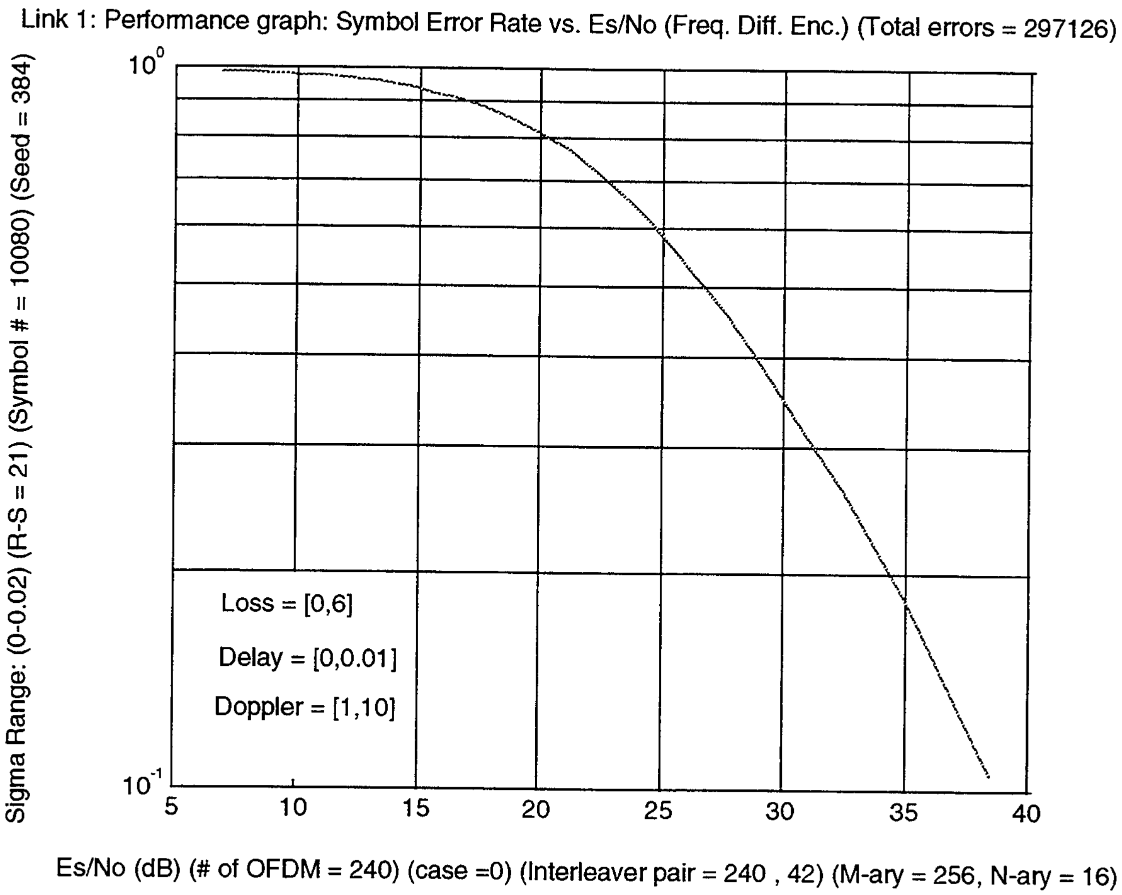


Fig. 99. Link 1 System Model 3 Simulation Using 240 OFDM Tones, Average Seed

Using the seed 195, which generates an average total message symbol error rate of 85 errors we get Fig. 97 which presents the corresponding link 3 performance curve with remaining input configurations consistent with similar worst case seed trials. Observation of $^{Es}/_{No}$ at a SER of 10^{-2}, indicates performance within 1 dB of uncoded theoretical AWGN. Link 2 however, displays performance at the same SER within 4 dB of theoretical (Fig. 98). Again link 1 demonstrates sub-standard performance being about 20 dB worse than theoretical AWGN even though the average error seed is used (Fig. 99).

These last performance figures represent the final simulation results of test phase 6. Having identified the optimal system OFDM tone number as 240, links 1 through 3 are simulated and their corresponding SER available for comparisons to theoretical. For purposes of this thesis based upon the work performed and the results obtained, the stated objectives have been accomplished, although performance results for some configurations are less than hoped for. The next chapter presents closing remarks about the results obtained including simulation data interpretations, general research comments and presentation of ideas for follow-on work.

VII. CONCLUSIONS

A. DISCUSSION OF SIMULATION RESULTS

The research work and results derived from this thesis are considered an overall success. From inception the objective has been to demonstrate a feasible COFDM modem system capable of maritime environment communications in the presence of known multipath and noise conditions. In general, these MATLAB based emulations and corresponding performance simulations support the COFDM concept based upon the results and information obtained. Further discussions relating to specific test phases are presented below.

1. Test Phase 1 and Test Phase 2 Discussions

Demonstration of an adequately working system emulation conducive to the overall success of the research was conducted and substantiated during test phase 0 and test phase 1. Test phase 0 validated a functionally correct model, as there were an absence of errors in the sink message with no channel included. This indicated that at least functionally all system sub-blocks within the transmitter and receiver were operating correctly according to design, and no obvious design flaws existed due to inaccurate m-file construction. Test phase 1 carries the functional verification one step further by also including complete system model 1 simulations. Conceptually, including this test permits performance curve comparisons to theoretical AWGN curves for M-DPSK as given by (39) to further verify correct emulation. Results of multiple system simulations indicated an approximate 1 dB discrepancy from theoretical AWGN. The hypothesis for the discrepancy was the symbol reformatting from 8-bit OFDM symbols to 4-bit PSK symbols. After running a similar trial using 4-bit OFDM symbol words sizes and 4-bit PSK symbol word sizes, the results indicate a near match to the theoretical curve approximation for 16-PSK, substantiating the opinion that the performance curve shift is due to symbol word reformatting.

As previously indicated, the reformatting of 8-bit OFDM symbols to 4-bit PSK symbols in preparation for 16-PSK transmission followed by inverse reformatting in the receiver adversely affects the R-S symbol correction sub-block. The R-S FEC function is located at the end of the receiver signal path after the deinterleaver and operates on 8-bit OFDM symbols attempting to correct symbol errors. However, it is apparent that channel error events affect 4-bit PSK symbols. Thus, it is possible for a 4-bit symbol error to become mapped into an 8-bit OFDM symbol error after reformatting, effectively extending the length of the error.

COFDM system design constraints require specific ordering of functional sub-block location within the transmitter and receiver. As a consequence, symbol error correction must be performed after deinterleaving; thus, 8-bits FEC is used since the deinterleaver operates on OFDM symbols. Interleaving and deinterleaving using 4-bit PSK symbols is a consideration, necessitating relocation of the interleaving/deinterleaving functional sub-blocks elsewhere in the signal path. However, as previously mentioned, R-S coding using 8-bit symbols permits a possible code block length of 255 symbols as compared to 4-bit symbols which would allow only 15 symbol code block lengths. Longer block lengths were considered more desirable, consequently for this thesis, the original design using 8-bit OFDM symbol interleaving prevailed.

2. Test Phase 3 Discussions

After construction of the software model was completed, testing and system simulations immediately began. Initially arbitrary seeds were chosen at random and used to configure the appropriate simulation for source message array content and channel properties. However, after running a few system simulation trials, large variations in performance results were observed using different seeds with remaining system configurations identical. With the chosen seed values used by both channel models, it was perceived that the performance variations correlated to the random channels created during simulations. Arguably, the particular seed value should have more of an impact on the multipath channel 2 model than the AWGN channel 1 model since AWGN generates random noise with iid characteristics in contrast to the multipath channel which tends to generate burst noise highly correlated events. Some preliminary system model 1 simulations verified this speculation, as total message symbol errors events as a function of the seed indicated little variance from the error totals for AWGN alone. Consequently, it was decided to perform system model 2 simulations to test a range of seeds and observe the total number of symbol errors appearing in the sink message array.

Within the inherent bounds of computer binary arithmetic, there are nearly an infinite number of conceivable integer seed values. However, it is possible to determine a smaller suitable finite subset as an appropriate representation of the nearly infinite set within a certain degree of confidence. As indicated during test phase 3, it was decided that a subset consisting of 500 integer seeds would be simulated through the system for each of the three links and corresponding sink error matrix symbol error totals recorded. Based upon this information, the worst case seed for each link would be included in configurations during subsequent system simulations to create a worst case channel 2 model condition. In this way, the corresponding performance plots emulate worst case communication links with respect to multipath distortions. Using the worst case channel 2 model seed out of a possible 500 seeds, we have 93.76% confidence that the particular seed is in the top 1% of worst seeds. Additionally, a record of the average number of symbol error events as a function of total test seeds would also be kept for later simulations using the average seed case. In this way, the corresponding simulation curves using the average error seed would reflect performance results for "average" communication link conditions. These simulation statistics were generated by batch m-file **uhfseeds.m**

Test phase 3 was also useful is isolating the optimal system configuration in terms of the number of OFDM tones. After initially considering a COFDM system based on 240 frequency tones, that number was reduced to also include 60 tones based upon the seed report results derived during test phase 3. Consequently, at that stage of development and testing, 240 tones and 60 tones were considered as likely candidates for optimal system performance in the presence of multipath distortions. It was also observed during the same test phase that frequency differential encoding worked better with 240 tones, and conversely time differential encoding worked better with 60 tones in terms of performance. Recall that differential encoding is a necessary functional component of the overall system, since without it the dynamics of the multipath channel distortions cause the received signal constellation plots to rotate their positions out of their expected sector spaces. It was verified through observations of received signal constellation plots that the constellations reoriented their positions in their proper sectors because of differential encoding at the expense of signal energy loss. There is a theoretical loss of 3 dB due to differential encoding as further evidenced by simulation received signal constellation output plots converging upon the origin; however, the benefit is increased error-free decoding of the received symbols and improved SER.

Recall that time differential encoding associated with the 60 OFDM tone configuration represents a cumulative symbol summation encoding technique applied to symbols along source message array rows. As previously mentioned, the source message symbol array consists of rows of symbols representing the time dimension as well as symbol columns representing the frequency dimension (OFDM frequency tones). Frequency differential encoding performs similar cumulative summations; however, the technique is applied to message array symbol columns and not rows. Hence, the frequency differential encoding technique is applied to the frequency dimension and performs well with 240 OFDM tones.

It is suggested by the data collected from the various simulations centered around 240 OFDM tones that frequency differential encoding works well with 240 tones because the frequency spacing between 240 tones is less than that for 60 tones. Remember that the same channel bandwidth of 480 kHz is available for both tone configurations. Since the frequency spacing for 60 tones is larger, there is more "room" between symbol carriers for the channel's multipath distortions to cause interference in the frequency dimension. Conversely, for 240 tones the spacing is narrower between orthogonal tones; thus, there is less "room" for multipath distortion interference to occur between symbol sub-carriers, allowing frequency differential coding to work better with 240 tones. This phenomenon is important since differential encoding effectively transmits information in terms of the differences between symbol constellation points, not the points themselves. In the time dimension, since the symbol interval lengths are the inverse of the frequency spacing according to OFDM theory, then 60 tones offer shorter interval lengths than 240 tones. Consequently, time differential encoding accommodates 60 tones better than 240 tones.

3. Test Phase 4 Discussions

Identifying 60 and 240 OFDM tones as the preferred choice during test phase 3, it was instructive to run system model 2 simulations during test phase 4 to observe general burst error pattern behavior. It was speculated prior to testing that the multipath channel causes signal distortions conducive to incorrect symbol decoding due to the RSL power loss, multipath delay and Doppler shifting parameters imposed by the channel upon the transmitted signal. It was also expected that time varying statistical burst noise events would adversely affect certain groups of adjacent symbols as they were transmitted through the channel. It was hoped that application of the COFDM technique would be effective in combating frequency selective fading. Hypothetically, the frequency selective fades would affect certain symbols within select portions of the overall channel spectral bandwidth, leaving other symbols corresponding to specific OFDM sub-carriers insufficiently affected and decoded correctly. Later interleaving combined with R-S FEC could correct the errored symbols. Test phase 4 simulations using the channel 2 model (multipath channel) exclusively demonstrated the effects of multipath on the received signal and the corresponding sink message array error event manifestations.

As expected for link 3, frequency selective fading occurred as well as partial flat fading. Links 1 and 2 indicated primarily flat fading. This test phase was also useful in depicting the behaviors of the received signal magnitudes and phases as seen by the constellation and magnitude plots. As anticipated, these plots demonstrated constructive and destructive interference due to channel multipath distortions, as evident by the distinguishing peaks and valleys apparent in the received signal magnitude plots. The received constellation plots demonstrated the manner in which individual symbol signal points were shifted in phase from their characteristic pre-transmitted positions. Also evident was the powerful effects of differential encoding/decoding on the received signal as many constellation points became realigned into their proper sector spaces, avoiding possible erroneous decoding but at the expense of signal energy. The observation of channel 2 model burst error patterns for specific link examples using 60 and 240 OFDM tones provided general knowledge of the expected error patterns for later system model 3 simulations and suggested that certain interleaver cases could be more effective in redistributing and decorrelating the symbol error groups.

4. Test Phase 5 Discussions

In addition to determining the preferable number of OFDM tones conducive to an optimal system, investigation and identification of optimal interleaver cases is also useful for optimizing system performance. Recall that interleaving can be effective in redistributing concentrated symbol error bursts to other locations within the message array. Under the assumption that the message array is of sufficient size so that multipath induced error bursts do not overwhelm the entire message block, then conceptually there should be enough error-free vacancy locations where errored symbols can be mapped to. Of course effective interleaving is primarily a function of the intermediate matrix dimensions. Larger matrices applied to the same multipath channel usually work better; however, large matrices increase latencies since all message symbols must be present in the intermediate matrix before CDL interleaving can be performed. Message array sizes of

10,000 OFDM symbols for simulations offer an acceptable 21 msec system latency as calculated by (38).

While the configured number of OFDM symbols and OFDM tones constrain the overall size of the message array and, hence, the size of the interleaver intermediate matrix, multiple intermediate matrix dimensions are still possible. This variability combined with nine different permitted CDL interleaver cases offer multiple conceivable different interleaving configurations. The purpose of test phase 5 was to perform repeated system model 2 simulations using all possible interleaver cases to identify the optimal ones for a particular link using corresponding worst case seeds. An optimal interleaver is defined as one that redistributes symbols errors in such a way as to cause the summation of errors across any single message array row to be minimized with respect to other interleaver cases operating on the identical source message array. Row error totals which are minimized promote the inclusion of reduced strength R-S FEC operating on symbols along message array rows and, hence, allows higher code rates. Higher but equally effective code rates are desirable in reducing overhead and increasing transmission efficiency since some non-information bearing symbols represent parity check symbols reducing the effective transmission information rate.

Test phase 5 was successful in identifying which interleaver cases demonstrated minimal row error totals corresponding to particular system configuration. It was also observed from multiple phase 5 simulation results that there was a variation in the number of total message errors as a function of the interleaver case tested. This phenomenon is unavoidable and is related to the specific transmission symbol sequence generated by the particular interleaver case. Certain symbols within a transmission sequence are individually affected differently by the channel depending on their location in the sequence, thus, a particular symbol may be decoded correctly for one interleaver case transmission sequence but also more prone to erroneous decoding for another interleaver case transmission sequence. The susceptibility of erroneous symbol decoding is attributed to correlations between the symbol's resulting channel encoded magnitude and phase values and the channel's statistical burst noise behavior acting on the transmitted complex modulation values. It is also apparent during test phase 5 that certain interleaver cases cause a preponderance of total message errors as compared to the other cases; however, in terms of message array row error totals the same interleaver case may minimize the error quantities. This is demonstrated for some of the links using 60 and 240 OFDM tones.

5. Test Phase 6 Discussions

As expected, test phase 6 presents the most important simulation data in terms of overall system performance results. This test phase initiated with 240 OFDM and 60 OFDM tones as leading configuration contenders. However, it consistently became evident after numerous trials using all three links that 60 OFDM tones is inferior to 240 tones in terms of performance. Consequently, after a few initial simulations the 60 OFDM tone case was no longer included in subsequent trials, while 240 OFDM tones, being the superior performer, continued to be used in later more refined simulation configurations. Contrary to the results obtained during test phase 3 suggesting the possibility that 60 OFDM tones should perform overall as well as or perhaps better than 240 tones, test phase 6 system model 3 simulation results suggested otherwise. Furthermore, when using seeds that generate average error totals, 240 OFDM tones consistently performed better for all the simulated links. It is not entirely understood why this is so, except that the simulation results demonstrated the facts.

With 60 OFDM tones no longer a system configuration candidate, simulations continued oriented around links 1 and 2 using the preferred 240 OFDM tones. Also, additional simulations using seed values generating average error totals were included. Link 2 performed the best when using R-S coding capable of correcting 21 errors, indicating performance better than theoretical. A better than theoretical uncoded AWGN performance is expected when applying R-S FEC using a 0.825 code rate. Link 3 also performed reasonably well being only slightly worse than theoretical AWGN performance while using an average error seed but not as well when using a worst case seed. Considering that link 3 is the most challenging communication link in terms of power loss, time delays and Doppler shifting with link 2 the next most challenging, the performance results of these two links using average error seeds is highly encouraging.

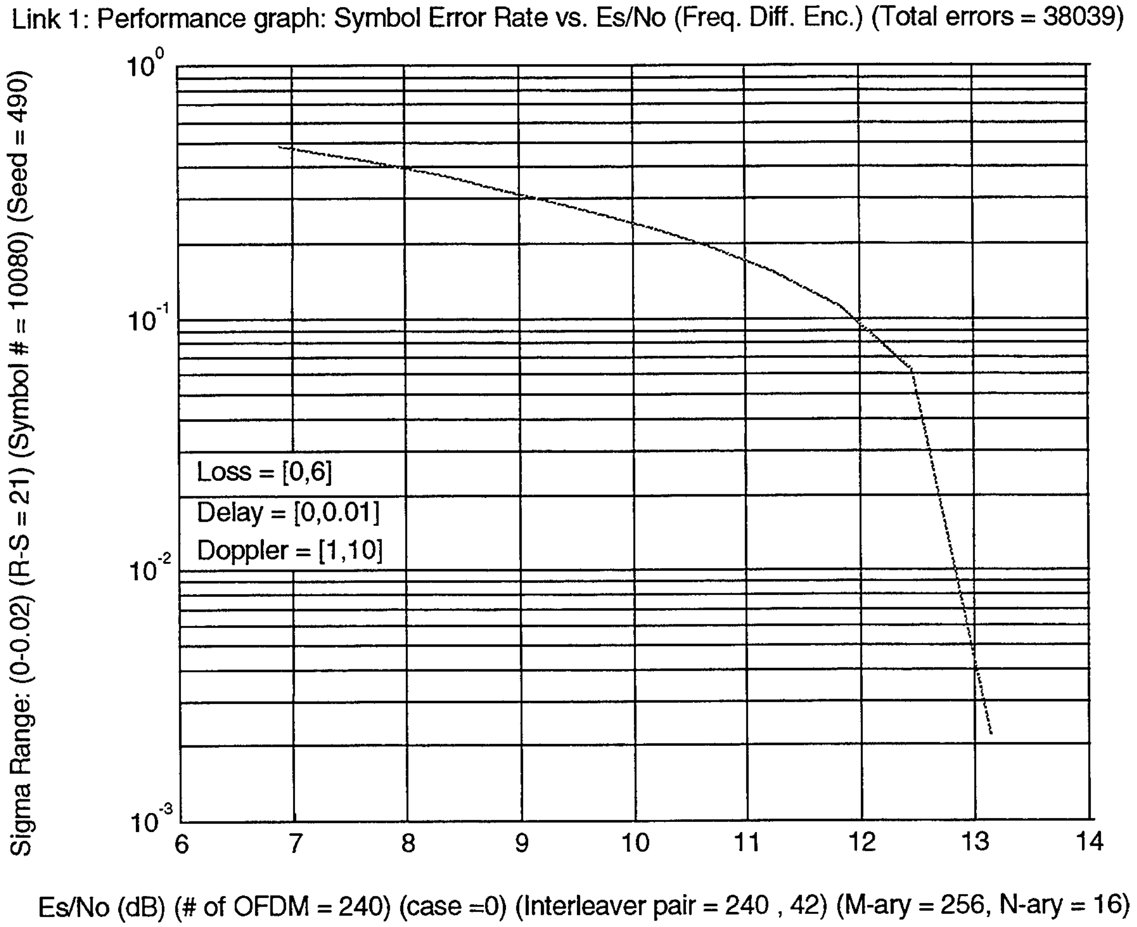


Fig. 100. Link 1 Simulation Using 240 OFDM Tones and Best Seed

Link 1, the supposed easiest link generates the poorest performance when a worst case or average seed is used. In fact, based upon the complete suite of link 1 simulation results thus far, the link would have to cut the effective information bit rate in half in order to allow any possibility of reliable communications. Consistently, link 1 performed inadequately apparently due to slow fading occurring within the channel when using the worst case and average seeds. However, according to test phase 3 results, link 1 only has 9 seeds out of 500 that generate any channel model 2 errors, 491 seeds generate zero errors; consequently, 98.2% of the time no multipath induced errors occur in the sink message. With this in mind, a link 1 simulation is presented in Fig. 100 using one of the best seeds, 490, that generates no channel model 2 symbol errors. After simulation of 10080 OFDM symbols with 240 tones and a 0.825 code rate, the results show dramatic performance improvement by better than 11 dB over theoretical uncoded AWGN performance due to the R-S FEC. Thus, while link 1 performance suffers most when worst case or average seeds are used, representing severe multipath distortion conditions characterized by slow fading, for 98% of the time reliable link 1 communications is possible with substantially improved performance when R-S coding is included. Perhaps possible methods to overcome the 2% unreliable condition are to dramatically decrease the information bit rate during periods of extreme link 1 multipath conditions or to include an alternate communication method such as a design using spatial diversity for transmission redundancy.

Link 2 performance demonstrates more reasonable performance than link 1 while using the worst case seed, although worse than 5 dB with respect to theoretical uncoded AWGN (Fig. 40). Link 3, comparable to link 2 performance, faired not as well as expected while using the worst case seed, and requires a 5.5 dB margin. With a 0.5 R-S code rate (Fig. 93), the performance improves considerably, better than uncoded theoretical AWGN, yet the effective information bit rate reduces by more than half when guard interval overhead is also taken into account.

It is informative to investigate the consequence of using QPSK with 240 tones and link 3 instead of 16-PSK and a 0.5 code rate, which offers a similar information rate. Configuring another simulation for evaluation purposes, we use 8-bit OFDM and 2-bit

PSK symbol lengths. A source message size of 5040 OFDM symbols are configured to generate a total of 20,160 QPSK symbols for channel transmissions, allowing compatibility with prior simulations. Initially, no R-S coding is included; and as evident from the simulation result shown in Fig. 101, the simulation fails beginning with a SER of 8x10^{-2} since the curve asymptotically approaches horizontal at that point. In Fig. 102, a similar simulation is configured with R-S FEC included using a 0.917 code rate capable of correcting 10 symbol errors in a code block length of 240 symbols. The simulation result indicates worse performance than theoretical uncoded AWGN, which is 13 dB at a SER of 10^{-3} and 11 dB at a SER of 10^{-2}. However, as expected, performance is better using QPSK than using 16-PSK at the substantially reduced information bit rate. In comparing QPSK and 16-PSK link 3 results, 16-PSK is preferable since a higher bit rate can be maintained

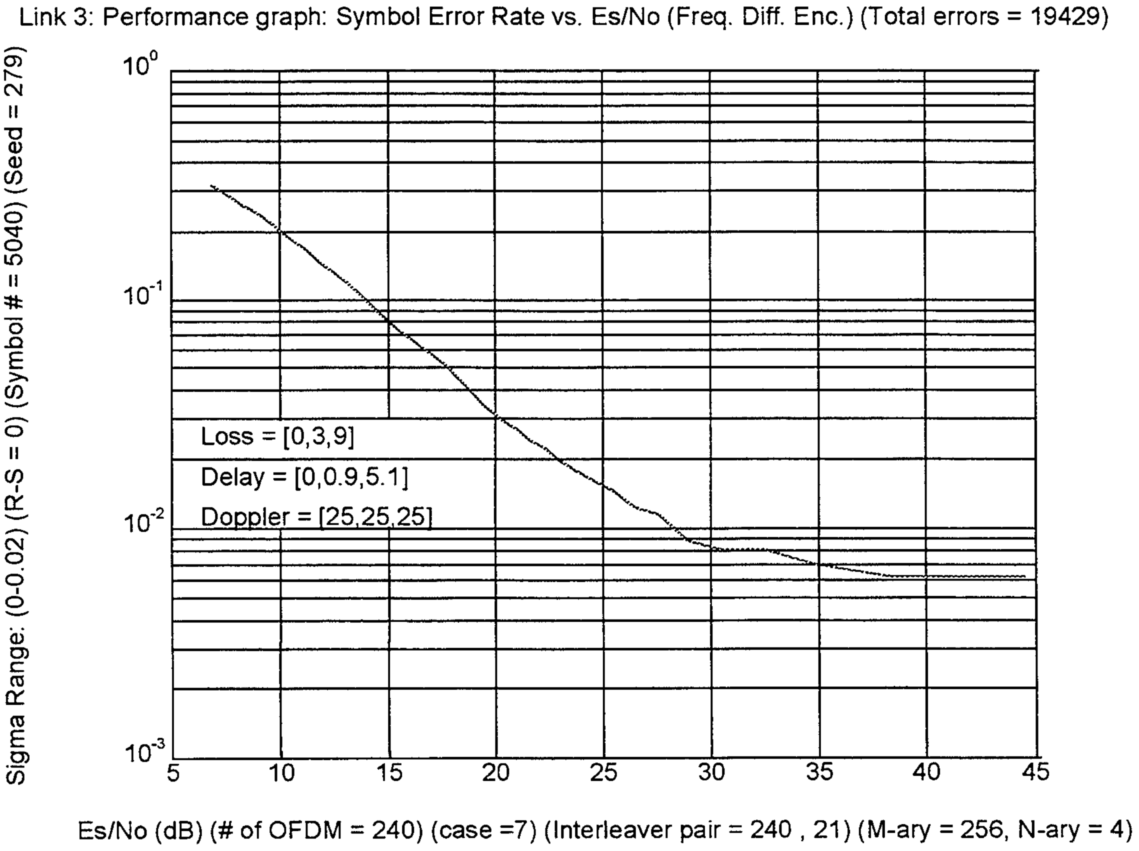


Fig. 101. Link 3 Simulation Using 240 OFDM Tones and QPSK, No R-S Coding

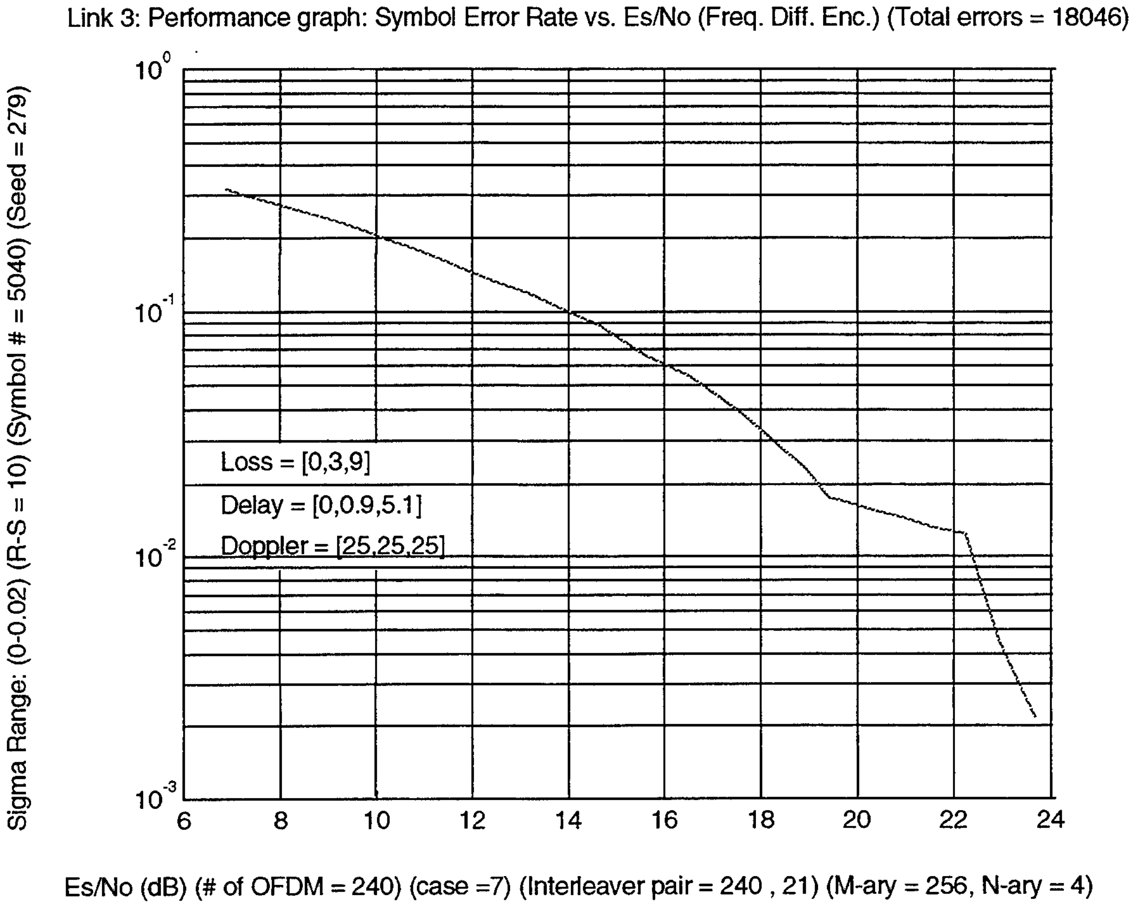


Fig. 102. Link 3 Simulation Using 240 OFDM Tones and QPSK, 0.917 Code Rate

The COFDM baseline design and corresponding simulation trials conducted throughout this thesis use the link specifications introduced in the BAA and are based upon the experimental multipath parameters measured and described in reference [7]. The Doppler frequency shifting, multipath time delays and received signal power loss parameters represent typical maritime link conditions as determined by prior controlled communication experiments; however, it is informative to alter the magnitudes of these parameters beyond the typical values to reflect a more harsh communication maritime environment with more severe multipath conditions. The purpose of this trial is to stress the system and determine how much added design tolerance is incorporated into the current COFDM baseline model. The simulation result presented in Fig. 103 reflects a link 3 custom simulation configuration with added Doppler shift of 10 Hz, while Fig. 104 demonstrates the effects of increased multipath time delays applied to link 3.

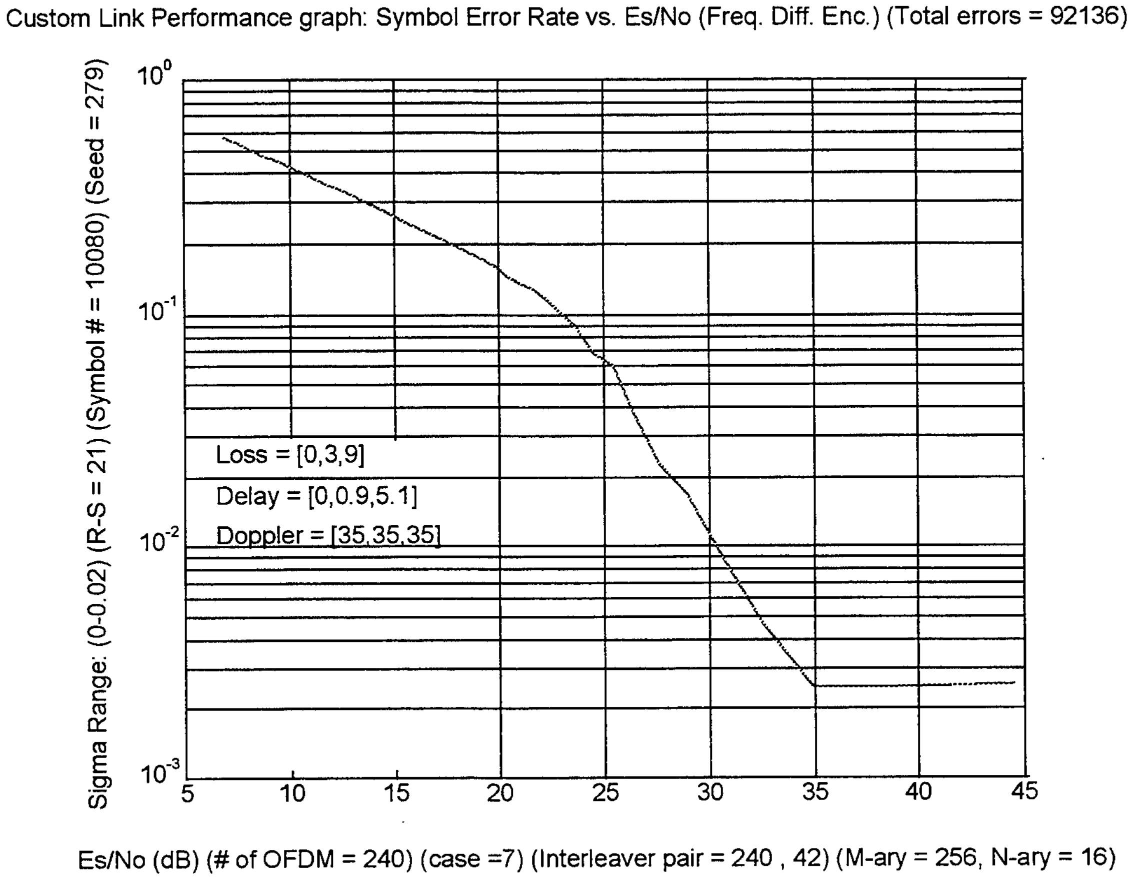


Fig. 103. Custom Simulation Using 240 OFDM Tones, F_d = 35 Hz

From Fig. 103 it is evident that the increase of Doppler shift to 35 Hz while maintaining the usual 0.825 code rate causes a breakdown of the formerly reliable system as errors appear in sink message arrays even without AWGN initially included. Evidently, there is little tolerance and immunity to added Doppler shift for a 240 OFDM tone based system. Of course, once again the worst case link 3 channel model 2 seed is used. The degraded system performance relative to increased Doppler is somewhat expected for a system using a large number of OFDM tones as there is less frequency spacing between tones. Consequently, additional Doppler shifting causes symbol spectra and their respective sub-carriers to shift their frequency location into adjacent symbol areas causing spectral overlap in addition to sub-carrier orthogonality loss. A system using less OFDM tones such as 60 tones offers better Doppler immunity since the frequency spacing is larger; however, as previously determined 60 tones generally does not perform as well as 240 tones and is not used. Correspondingly, in the time domain, added multipath delays should have a lesser affect with 240 OFDM tones as the symbol intervals and corresponding guard intervals are longer than 60 tones for example and should offer improved multipath delay immunity. Recall that the symbol intervals are the reciprocal of the tone spacing for OFDM; thus, a configuration with a tone spacing offering superior Doppler immunity would not necessarily offer equally good multipath delay immunity and vice versa.

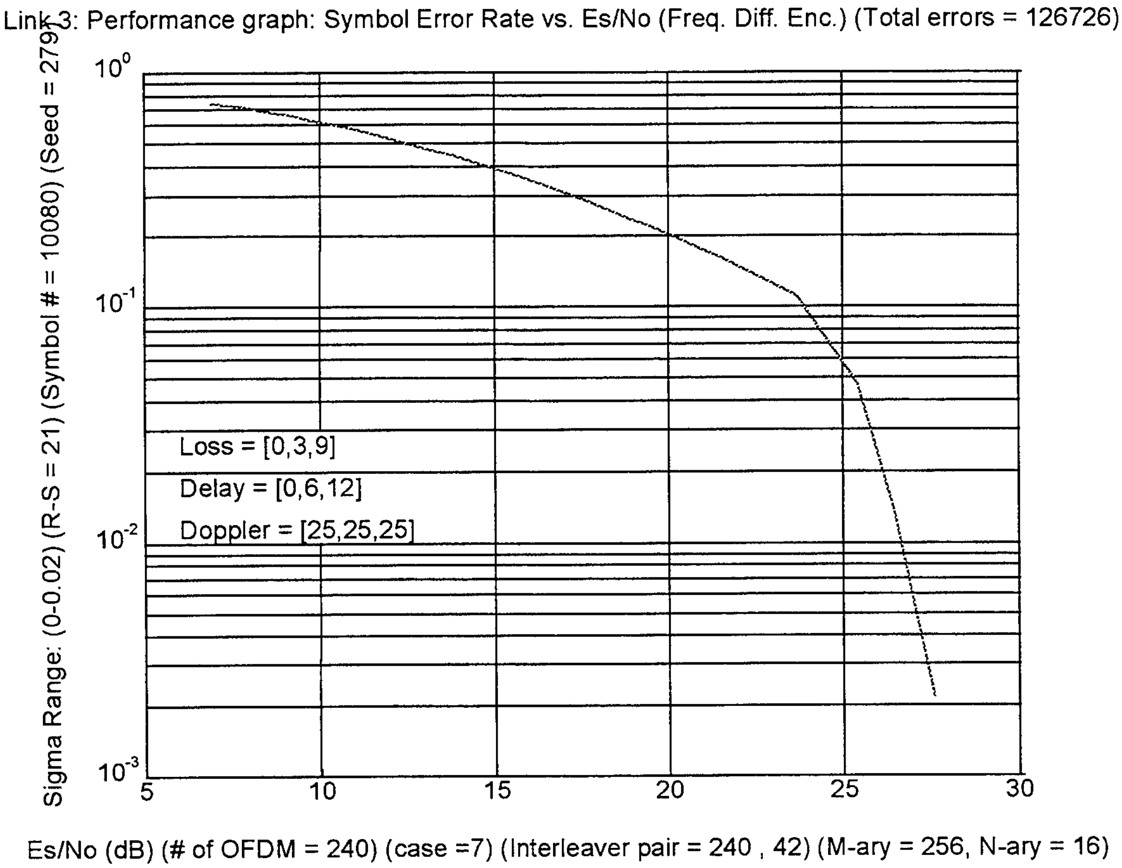


Fig. 104. Custom Simulation Using 240 OFDM Tones, $T_{1\text{-}P}$ = [0,6,12]

Performing another system model 3 custom simulation using increased multipath delays for the three separate transmission paths, we get the result depicted by the Fig. 104 corresponding performance curve. Ironically, the performance improves from the standard link 3 curve depicted in Fig. 92 by about 2.5 dB at a SER of 10^{-2}. However, after adding even more multipath delay, the performance gradually degrades as seen in Fig. 105 until complete system breakdown occurs during the simulation results seen in Fig. 106 suggesting unreliable communication.

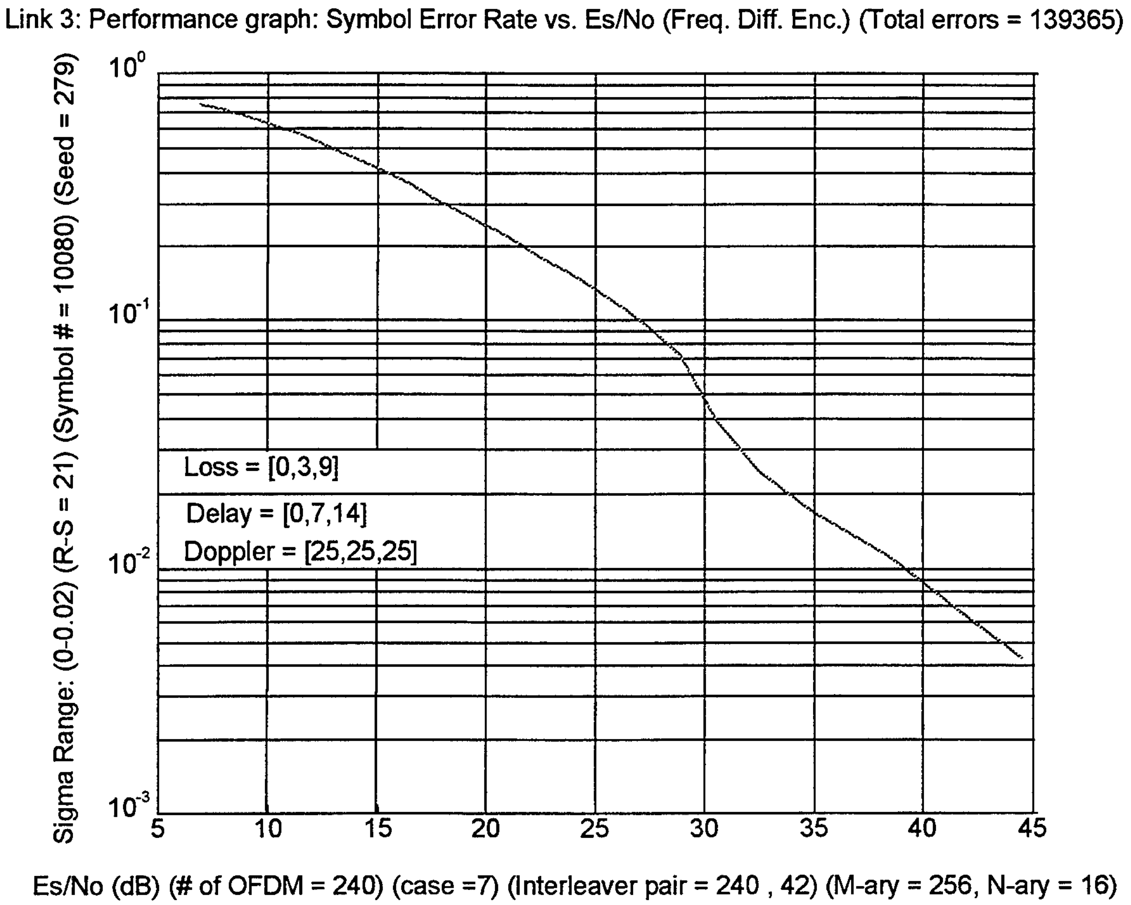


Fig. 105. Custom Simulation Using 240 OFDM Tones, $T_{1\text{-}P}$= [0,7,14]

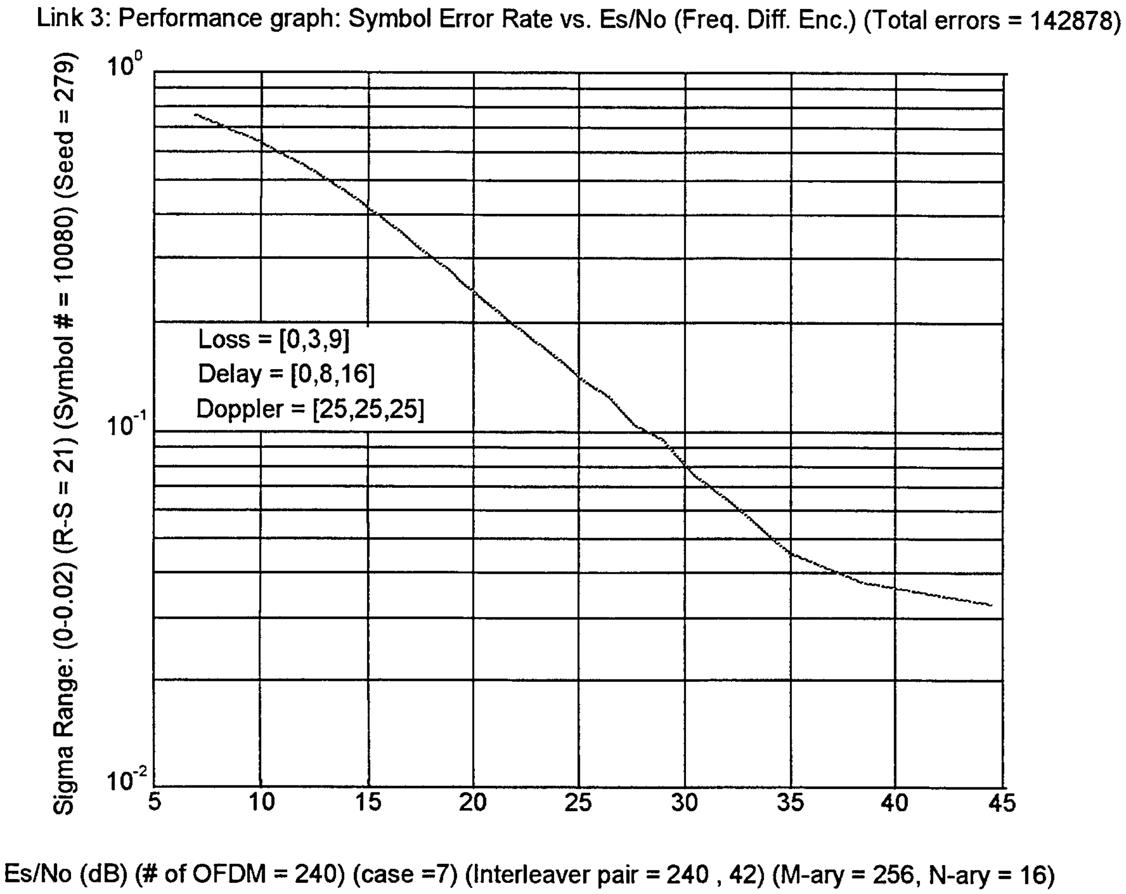


Fig. 106. Custom Simulation Using 240 OFDM Tones, $T_{1\text{-}P}$ = [0,8,16]

As previously explained, simulations oriented around 8-bit OFDM symbol word lengths followed by format conversion to 4-bit PSK symbols introduces additional artificial implementation errors adversely affecting performance results. The baseline model developed for this thesis includes the 8-bit OFDM symbol design approach to partially accommodate commercially available R-S FEC IC hardware as well as to permit formations of longer R-S code blocks within the message array. It is instructive to perform additional system model 3 simulations using 4-bit OFDM and 4-bit PSK symbols and observe the degree of expected performance improvement for sake of comparisons to the baseline model results. Consequently, Fig. 107 through 108 depict 4-bit OFDM message symbol simulations configured with 240 tones and a 0.825 code rate. Also, for consistency reasons, a total of 20,160 4-bit COFDM symbols equivalent to the 4-bit PSK symbols are simulated. While a 240 symbol code block length is configured for these simulations to remain consistent with prior COFDM system model 3 performance results, in actuality, another type of FEC code other than a R-S code would have to be incorporated into the model since R-S FEC using 4-bit symbols is limited to a 15 symbol code block length.

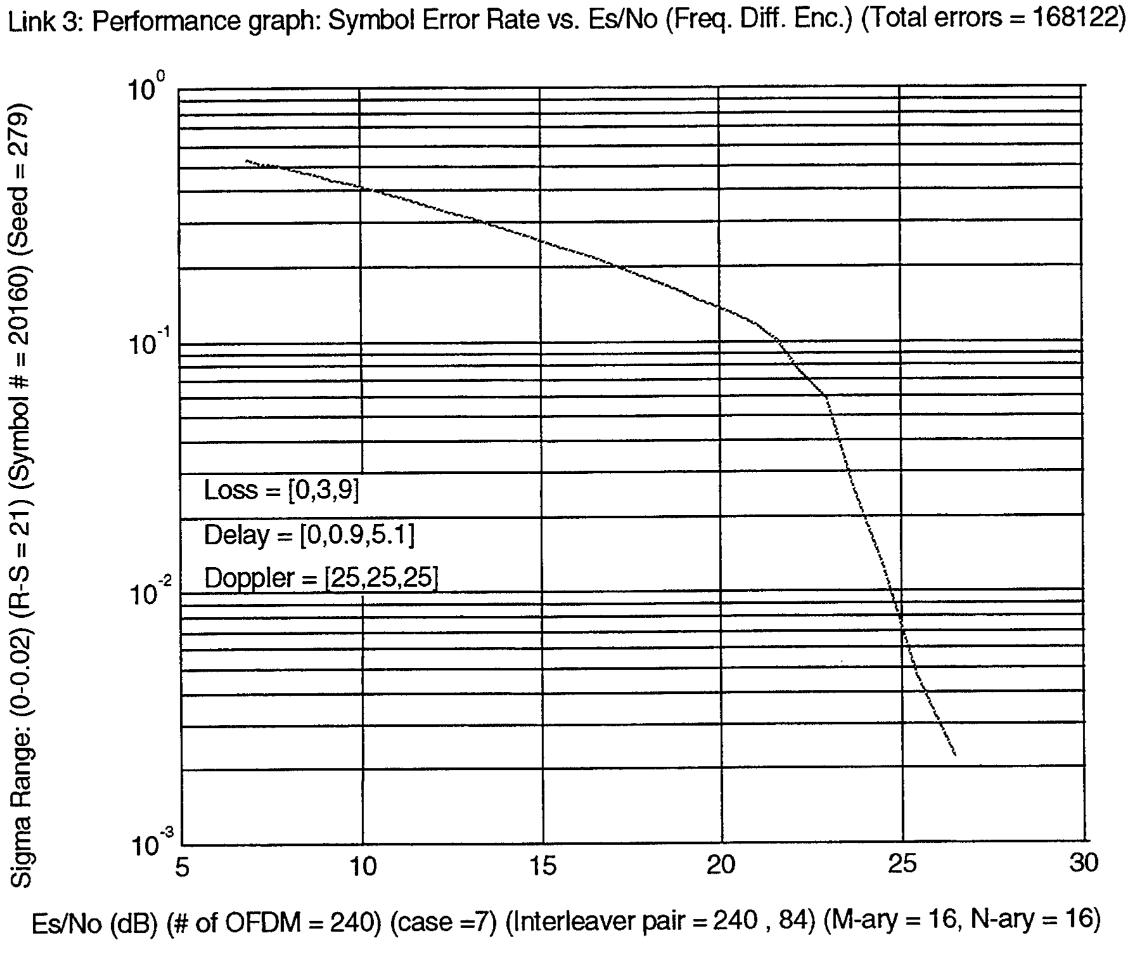


Fig. 107. Link 3 Simulation Using 240 OFDM Tones, 4-bit OFDM Symbols

The corresponding simulation result is presented in Fig. 107. From the figure there is approximately a 4 dB improvement at a SER of 10^{-2} in comparison to a similar simulation using 8-bit OFDM symbol word lengths depicted in Fig. 89. Furthermore, this result is within 3 dB of approximated theoretical uncoded AWGN performance (Fig. 40) using 16-PSK signaling. Evidently, the use of 4-bit OFDM symbols instead of 8-bit symbols suggests a preferred implementation choice if based upon performance alone, also assuming that a suitable FEC code producing a similar 0.825 code rate is available and applicable to the COFDM system.

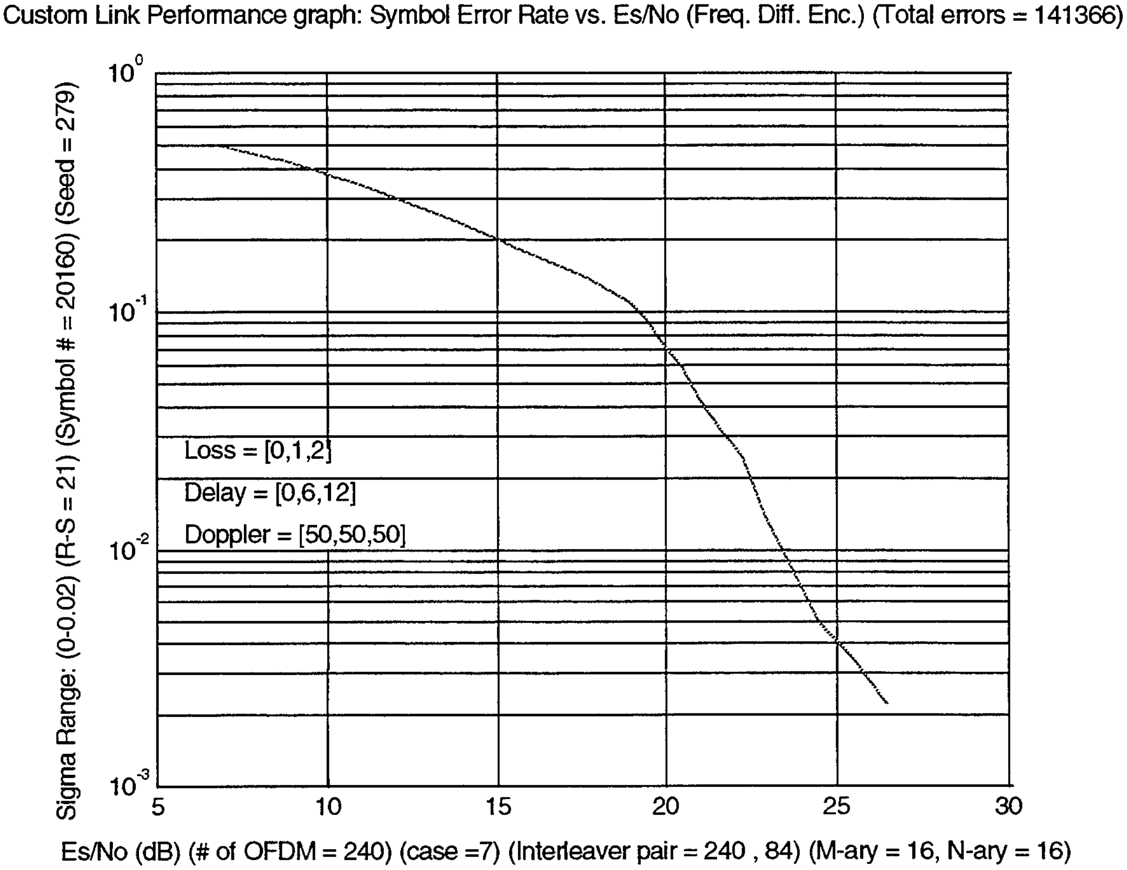


Fig. 108. Custom Simulation Using 240 OFDM Tones, 4-bit OFDM Symbols

Figure 108 depicts a similar simulation, however, added Doppler shifting and increased multipath delays with reduced RSL losses are include to determine the degree of added tolerance associated with the modified baseline model 4-bit OFDM symbol configuration. From the figure it is evident that this configuration is more immune to the added multipath distortion effects than the standard link 3 simulation using 8-bit OFDM symbols, as performance curve results are within 1 dB of the curves depicted in Fig. 107. In comparison to the 8-bit OFDM symbol design, this new configuration offers more robustness, lending further support for a possible future implementation.

6. General COFDM Emulation Discussions

With the presentation of a design variation to the thesis emulation baseline model using 4-bit OFDM symbols and offering potentially improved performance under a specific design constraint (suitable FEC applied), the research work and simulations conclude at this point demonstrating reasonable success overall in meeting the objectives. Specifically, using the baseline system design, link 2 offers overall performance results conducive to a practical implementation using the 1.536 Mbps bit rate specification for all types of channel conditions (seed values); however, link 1 demonstrates performance requiring much additional margin for about 2% of the time when the channel exhibits slow fading (worst case and average seeds) unless the information bit rate is dramatically reduced below the minimum stated BAA objective. For the other 98% of the time, no multipath errors occur and link 1 performs satisfactorily. Link 3 also suggests a feasible OFDM based system, but a lower information bit rate or higher margin is required when worst case channel conditions exist. On average, however, link 3 performs adequately when the average error channel seed is used. Additional simulation performance results for various link configurations including custom links are provided in Appendix B for further informative analysis by the reader. Next, comments are included relating to the research system model design and emulation challenges encountered along the way as well as suggestions for possible related future work.

B. RESEARCH AND EMULATION CHALLENGES

This thesis research oriented around a COFDM based communication system emulating a full-duplex modem was well worth the effort since in general the simulation results are successful in their intent to support the feasibility of a future COFDM implementation meeting BAA specifications and to provide reliable communications in the maritime environment. However, success of this research was not without the endurance and conquering of multiple challenges encountered along the way during system functional sub-block development, integration and system simulations.

Most notable were the difficulties involved with the construction and operational understanding of the interleaver and deinterleaver. Initially there were numerous batch m-file verification simulation failures as symbols within the message array were reordered incorrectly. However, after persistent debug and numerical paper calculations using arbitrary arrays and intermediate matrices, a correct interleaver and deinterleaver sub-block were eventually integrated into the system and functionally verified.

Other challenges were oriented around system simulations. Initially random simulations were configured using arbitrary seed values. With this approach, large variations in performance results were evident. At first a configured link 3 simulation would perform extremely well immediately indicating a successfully designed system model; later another simulation using a different seed generated much poorer results, suggesting an inadequately working model. Ultimately it was discovered that these performance variations were closely related to seed selection. This realization prompted the seed evaluation trials conducted during test phase 3 using batch m-file **uhfseeds.m** and the compilation of the subsequent seed error report data. As expected and previously discussed, different seeds caused the channel 2 model to behave differently with some seeds promoting far worse performance than others. Identifying the worst seed in terms of the number of message symbol errors produced during a simulation ensured that worst case channel conditions existed during later simulations. In that way various configurations could be included in multiple simulations by changing other pertinent parameters while still using a worst case channel 2 model. The channel 1 model in general does not have a worst case seed since AWGN ordinarily generates random error distributions and not the type of burst errors patterns typically associated with channel 2 multipath distortions.

Another simulation challenge occurred during test phase 5 with the identification of optimal interleaver cases. When using identical source message arrays for multiple independent simulations but including different interleaver cases, dissimilar transmission symbol sequences were produced. However, it was observed that the total message errors corresponding to the different cases varied. At first this was considered not possible since the identical channel 2 model was used for each trial as set by seed selection. This phenomenon seemed to suggest a software design implementation flaw. After much scrutiny it was determined that this effect was expected and was explained by the interleaving process combined with the channel model 2 stochastic distortion effects.

Some difficulties were also encountered with the PC platform used for the emulation and simulations. Initially a Pentium based 133 MHz PC with 32 Mbytes of RAM was used for simulation trials using 10,000 OFDM symbols. It quickly became apparent that a more powerful machine would be required as PC hardware limitations constantly thrashed the hard drive and drastically slowed simulation times. A requisition was eventually made to purchase a more powerful PC machine consisting of a Pentium based processor operating at 200 MHz with 64 Mbytes of RAM. This platform solved the simulation thrashing problems and considerably reduced simulation time by about one third.

C. FUTURE WORK

The research presented in this thesis represents a fundamental software baseline model successfully demonstrating operation of a COFDM based communications modem operating at baseband frequencies in the presence of known noise and signal fading conditions. The rudimentary but necessary sub-block components comprising the model such as symbol interleaving and R-S FEC are adequate in generating meaningful simulation data and are included in the emulation to maximize the COFDM technique conducive toward optimal performance. However, extension of this basic model to include, for example, up/down conversion and DAC/ADC is also possible. Associated with these functions would be filtering and perhaps additional time domain processing in the form of windowing such as Hamming or Hanning. Through these modifications, communication link simulations would be possible, more closely emulating physical hardware implementations. It may also be possible to improve the performance of the system with these changes.

Additionally, application of a more suitable FEC code could promote the removal of the symbol format conversion sub-blocks from the model so that 4-bit OFDM symbols could be maintained throughout the simulations for the entire signal path. As seen from the latest simulation performance graphs using 4-bit symbols, the resulting superior performance warrants investigation. Possible FEC methods include concatenated coding using convolutional codes and/or turbo codes. Also, incorporating FEC parity symbol encoding in the transmitter along with decoding in the receiver would augment emulation authenticity with respect to hardware implementation by adding the complexity and latency normally associated with the FEC operation.

It was observed during phase 6 testing that link 1 occasionally generated substandard performance requiring extra margin. Despite and overabundance of FEC, the link consistently performed poorly when worst case channel 2 conditions existed unless the information bit rate was reduced substantially. One possible solution to the link 1 problem is to modify the current system model to included spatial diversity. In this way redundant transmission links would be available promoting more reliable communication. The assumption is that the multipath fading channel and distortions generating signal interference may possibly catastrophically affect one transmission link but also may be less correlated with the redundant link inhibiting communications to a lesser degree or not at all.

Finally, it is hinted at during system test phase 5 that there may exist an optimal interleaver superior to the current CDL interleaver incorporated into the design It is possible that this unknown interleaver could effectively redistribute errored symbols in the best possible way for all known multipath channel conditions within certain constraints. This in combination with the most effective FEC algorithm could allow the COFDM system to perform the best for all predicted communication environments with respect to the multipath channel. At present the author in not aware of any such optimal interleaving method. Consequently, the thesis research ends here, and future modifications and improvements are left for another day, perhaps for a Ph.D. dissertation.

APPENDIX A. COFDM SYSTEM MODEL MATLAB MACRO CODE

1. Function: awgn.m

```
% function [Y] = awgn(X,s,N,sigma)
%______________________________________________________________________
%
%       Title:          ADDITIVE WHITE GAUSSIAN NOISE (CHANNEL MODEL 1)
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  2/7/97
%______________________________________________________________________
%       INPUTS:
%               X     - Input array of time domain complex modulation values
%               s     - Seed parameter for random number generator
%               N     - Number of OFDM frequencies (FFT size), includes zero pad
%               sigma - Noise parameter for calculating Eb/No (function of the noise variance)
%
%       OUTPUTS:
%               Y    - Output signal plus noise, array of time domain complex numbers
%
% AWGN:  Awgn is an m-file that adds AWGN noise to the matrix X, consisting of time
% domain complex numbers.  If X has complex elements with magnitude one and constant
% phase, then Ec/No = 1/(2*sigma^2) is the carrier energy to noise power spectral density ratio
% of y(n) = Re(Yexp(j*2*pi*k*n/N)).
%_____________________________________________________________________________
% USAGE: function [Y] = awgn(X,s,N,sigma)
%_____________________________________________________________________________
function Y = awgn(X,s,N,sigma)
%
% Find the dimensions of the input array.
%
[rr,cc]=size(X);
%
% Various seed configurations to set the random # generator seed.
%
%randn('seed',sum(100*clock))
%randn('seed',0)
randn('seed',s+30);
%
% Generate a random real part.
%
wreal =  randn(rr,cc);
```

```
%randn('seed',sum(100*clock))
%randn('seed',0)
%
% Generate a random imaginary part.
%
randn('seed',s+40);
wimg = i*randn(rr,cc);
%
% An array of random complex entries chosen from a normal distribution with mean 0.0
% and variance 1.0. Array dimensions is the same as X.
%
W = wreal + wimg;
%
% Random noise multiplied by the sigma factor and added to the signal.
%
Y = X + (sigma .* W);
%______________________________________________________________________
```

2. Function: bm.m

```
% function [m] = bm(q,v)
%________________________________________________________________________
%
%       Title:          BINARY TO M-ARY CONVERTER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  12/4/96
%________________________________________________________________________
%       INPUTS:
%               q - Base two exponent for M-ary symbol generation
%               v - Binary data vector
%
%       OUTPUTS:
%               m - M-ary output vector in decimal notation
%
%       NOTE: This m-file performs the inverse function of m-file mb.m
%
% BM: This m-file implements a binary to M-ary converter by accepting two input
% arguments, q and v, and returning the output, m. The input argument, v, is a
% vector of binary bits of any length. The input argument, q, represents
% the base two exponent for the M-ary conversion. Zeros are added to the
% end of v if necessary to ensure an even multiple of q with no remainder (even
% modulo q) during the generation of the final M-ary symbol. The bits, v, are stripped
% q at a time and are mapped to a symbol vector m with integer values 0 to 2^q-1.
% The least significant bit is taken to be on the left for each q bit symbol.
%________________________________________________________________________
% USAGE: function [m] = bm(q,v)
%________________________________________________________________________
function m = bm(q,v)
%
% Find the length of input vector, v, and determine if there is a remainder after
% dividing by q.
%
n = length(v);
r = rem(n,q);
%
% If there is no remainder, don't pad v input vector. Otherwise add the appropriate
% number of zeros to generate a code word with an exact multiple of q bits.
%
if r == 0
v = v;
```

```
else
v = [v zeros(1,q-r)];
end
%
% Place least significant bit of the symbol on the left end.
%
map = 1;
for j =1:q-1
map =[map 2^j];
end
%
% Remove q bits at a time from v to generate m-ary symbol values.
%
n = length(v);
p = round(n/q);
A = zeros(q,p);
A(:) = v;
m = map*A;
m_ary_msg = m;
%____________________________________________________________________
```

3. Function: cdldlv.m

```
% function s = cdldlv(l,k,case,si,SYNC)
%_________________________________________________________________________
%
%       Title:          CDL BLOCK DEINTERLEAVER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by      Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  3/24/97
%_________________________________________________________________________
%       INPUTS:
%               l    - Number of rows in intermediate matrix
%               k    - Number of columns in intermediate matrix
%               case - Variable indicating the deinterleaving method to be used (9 different
                       cases)
%               si   - Input message string to be deinterleaved
%               sinc - Frame syncronization bits. (Not required for COFDM simulations)
%
%       OUTPUTS:
%               s - Interleaved output string
%
%       SUBROUTINES USED:
%               rotm.m
%
%       NOTE: This m-file performs the inverse function of m-file cdlilv.m
%
% CDLDLV:   This m-file is a block deinterleaver of the type used in CDL.
% It is used to de-interleave a vector si, that has been interleaved using
% cdlilv.m, into the vector s. (See cdlilv.m for definitions of l,k and case.)
%_________________________________________________________________________
% USAGE: function s = cdldlv(l,k,case,si,SYNC)
%_________________________________________________________________________
function s = cdldlv(l,k,case,si,SYNC)
si(length(si)+1-length(SYNC):length(si))=zeros(1,length(SYNC));
N=length(si);
if l*k==N
x=zeros(l,k);
x(:)=si;
K=(1:k)-1;
CR=K.*(K+1)/2;
%CR=rem(CR,l);
L=(1:l)-1;
RR=L.*(L+1)/2;
```

```
%RR=rem(RR,k);
%
% Case types (Uses m-file rotm.m)
%
if case==1  % Column negative
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
elseif case==2  %Column positive
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
elseif case==3  %Row negative
for kk=1:l
x(kk,:)=rotm(x(kk,:),RR(kk));
end
elseif case==4  %Row positive
for kk=1:l
[z,x(kk,:)]=rotm(x(kk,:),RR(kk));
end
elseif case==5  %Row negative, column negative
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
for ll=1:l
x(ll,:)=rotm(x(ll,:),RR(ll));
end
elseif case==6 %Row negative, column positive
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
for ll=1:l
x(ll,:)=rotm(x(ll,:),RR(ll));
end
elseif case==7  %Row positive, column negative
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
for ll=1:l
[z,x(ll,:)]=rotm(x(ll,:),RR(ll));
end
elseif case==8  %Row positive, column positive
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
for ll=1:l
```

```
[z,x(ll,:)]=rotm(x(ll,:),RR(ll));
end
end
x=x';
s=x(:);
s=s';
end
%_______________________________________________________________________
```

4. Function: cdlilv.m

```
% function si = cdlilv(l,k,case,s,SYNC)
%______________________________________________________________________________
%
%       Title:          CDL BLOCK INTERLEAVER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by      Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  3/24/97
%______________________________________________________________________________
%       INPUTS:
%               l    - Number of rows in intermediate matrix
%               k    - Number of columns in intermediate matrix
%               case - Variable indicating the deinterleaving method to be used (9 different
                       cases)
%               s    - Input message string to be deinterleaved
%               SYNC - Frame syncronization bits. (Not required for COFDM simulations)
%
%       OUTPUTS:
%               si - Interleaved output string
%
%       SUBROUTINES USED:
%               rotm.m
%
%       NOTE: This m-file performs the inverse function of m-file cdldlv.m
%
% CDLILV:    This m-file is a block interleaver of the type used in CDL.
% The vector s is read into an (l,k) matrix by rows. The rows and columns
% are rotated (cyclically shifted) positively or negatively using the algorthim
% given in the Ref Unisys Doc. Spec 7690698. The variable "case" is set to
% the appropriate case number (1-8) for the eight combinations of row and
% columns interleaving given in the Spec.
%        After the row and columns interleaving, the matrix is read
% out by columns into the vector si. If "case" is set to 0, no rotations are
% used, the vector s is simply read into the matrix by rows and read out by
% columns as in an ordinary block interleaver. The bit sequence specified by
% SYNC overwrites the last bits of si. For CDL this is a 16 bit sequence.
%______________________________________________________________________________
% USAGE: function si = cdlilv(l,k,case,s,SYNC)
%______________________________________________________________________________
function si = cdlilv(l,k,case,s,SYNC)
N=length(s);
if l*k==N
```

```
x=zeros(l,k);
x=x';
x(:)=s;
x=x';
Intermediate_mx = x;
K=(1:k)-1;
CR=K.*(K+1)/2;
%CR=rem(CR,l);
L=(1:l)-1;
RR=L.*(L+1)/2;
%RR=rem(RR,k);
%
% Case types (Uses m-file rotm.m)
%
if case==1  %Column negative
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
elseif case==2 %Column positive
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
elseif case==3  %Row negative
for kk=1:l
[z,x(kk,:)]=rotm(x(kk,:),RR(kk));
end
elseif case==4  %Row positive
for kk=1:l
x(kk,:)=rotm(x(kk,:),RR(kk));
end
elseif case==5  %Row negative, column negative
for ll=1:l
[z,x(ll,:)]=rotm(x(ll,:),RR(ll));
end
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
elseif case==6  %Row negative, column positive
for ll=1:l
[z,x(ll,:)]=rotm(x(ll,:),RR(ll));
end
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
elseif case==7 %Row positive, column negative
for ll=1:l
```

```
x(ll,:)=rotm(x(ll,:),RR(ll));
end
for kk=1:k
[z,x(:,kk)]=rotm(x(:,kk),CR(kk));
end
elseif case==8  %Row positive, column positive
for ll=1:l
x(ll,:)=rotm(x(ll,:),RR(ll));
end
for kk=1:k
x(:,kk)=rotm(x(:,kk),CR(kk));
end
end
si=x(:);
si=si';
end
si(length(si)-length(SYNC)+1:length(si))=SYNC;
%_____________________________________________________________________________
```

5. Function: cdrcdlft.m

```
% function [Fa,MD,B,nsymno] = cdrcdlft(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,fort)
%_____________________________________________________________________________
%
%       Title:          COFDM ENCODER WITH CDL INTERLEAVING
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/10/97
%_____________________________________________________________________________
%       INPUTS:
%               picy_n - Switch variable to allow or disallow the generation of figures
%               pic    - Argument passed by another calling m-file to indicate the loop number
%               s      - Seed parameter for random number generator
%               freqno - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv - Interleaver parameter for intermediate matrix row #
%               cintlv - Interleaver parameter for intermediate matrix column #
%               N      - Number of FFT frequency sample points, must be larger than freqno
%               mary   - Initial M-ary symbol format (OFDM symbol bit length)
%               nary   - Final N-ary symbol format (PSK symbol bit length)
%               fort   - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               Fa     - Frequency array of prearranged modulation values
%               MD     - Matrix of differentially encoded complex values (unit magnitude)
%                        and one of N-ary possible phases (N-PSK)
%               B      - Matrix of 8-ary symbols
%               nsymno - Number of N-ary generated symbols
%
%       SUBROUTINES USED:
%               marymsg.m, cdlilv.m, mb.m, bm.m, difcdrft.m, cmv2fa.m
%
%       NOTE: This m-file performs the inverse function of m-file decdrcdl.m
%
% CDRCDLFT: This m-file represents the COFDM transmitter symbol and channel encoder.
% This m-file generates a random array of M-ary message symbols as a function
% of the inputs rintlv and freqno, and returns a matrix of equivalent differentially
% encoded complex numbers, MD, with unit magnitude and one of N possible phases (N-PSK).
% The message vector is initially formatted as M-ary symbols and reshaped into a
% matrix with values between 0 and (2^M-1). The matrix is CDL interleaved, reformatted as
% N-ary symbols with values between 0 and (2^N-1) and is frequency or time differentially
% encoded before finally being converted to complex values. The matrix of M-ary symbols
% is also returned as output matrix, B.
```

```
%_____________________________________________________________________________
% USAGE: function [Fa,MD,B,nsymno] =
cdrcdlft(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,fort);
%_____________________________________________________________________________
function [Fa,MD,B,nsymno] = cdrcdlft(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,fort);
%
% Determine if the number of OFDM frequencies are even (# of matrix columns), indicated
% by the "freqno" parameter.  If odd go to error message.  Odd frequencies are not
% allowed since the formation of the frequency array is symmetrical and even.
%
        if rem(freqno,2) ~= 0
        disp('ERROR: The number of matrix columns, freqno, representing OFDM frequencies
must be an even number!')
        elseif rem(freqno,2) == 0
%
% Determne if the row and column interleave parameters are greater than freqno when
% multiplied together.  If not, then display error message and stop.
%
        if (rintlv*cintlv) < (freqno)
        disp(' ')
        disp('ERROR: The row and column interleave parameters are not compatible with # of
OFDM frequencies!')
        disp(' ')
else
%
% Calculate the row symbol number.
%
symno = rintlv*cintlv/freqno;
%
% Display error message if symno and freqno not compatible with rintlv and cintlv and
% stop.  If not compatible, the interleaver function does not work correctly.
%
        if rem(symno,1) ~= 0
        disp(' ')
        disp('ERROR: The row and column interleave parameters are not compatible with # of
OFDM frequencies!')
        disp('    For the entered rintlv, cintlv, and freqno parameters, the calculated symno is: ')
        disp(symno)
        multiesall = mltpl(rintlv,cintlv);
        multies = multiesall(1,(2:length(multiesall)-1));
        disp('    Possible choices for freqno based upon rintlv and cintlv are:')
        disp(' ')
        disp(multies)
        elseif  rem(symno,1) == 0
if freqno >= N;
        disp(' ')
        disp('ERROR: The number of frequency points, N, needs to be increased!')
        disp('N must be larger than:')
        disp(' ')
        disp(freqno)
        disp(' ')
elseif freqno < N;

Nmbr_of_symbols = symno * freqno;
%
% Generate a random message matrix of m-ary symbols, based upon parameter, mary.
% (Uses macro: marymsg.m).
%
B=marymsg(mary,s,symno,freqno);
Rndm_m_ary_msg=B;
%
% Perform a CDL block interleaving function on the matrix, B, with rintlv rows
% and cintlv columns.  (Uses macro cdlilv.m).
%
SYNC = [];
[Br Bc] = size(B);
Bt=B';
Bvect = Bt(:)';
si = cdlilv(rintlv,cintlv,case,Bvect,SYNC);
Bi = reshape(si,Bc,Br)';
Intrlvd_array = Bi;
%
% Reconstruct the matrix block of m-ary symbols into an equivalent
% information block using n-ary symbols.  For the case when m=256 (256-ary) and n=16 (16-ary)
% the reshaped matrix will be twice the size of the initial matrix.  Padding of zeros
% may be necessary for certain m-ary and n-ary combinations.  The expansion of the
% original m-ary message block is along the row dimension after conversion to n-ary
% symbols and for the case when (m>n).  (Uses macros: mb.m and bm.m).
%
m1=bm(nary,mb(mary,Bi));
lengthm1 = length(m1);
nsymno = lengthm1;
remm1 = rem(lengthm1,freqno);
        if remm1 == 0;
        m1 = m1;
        else
        zero = zeros(freqno - remm1);
        m1 =[m1 zero(1,:)];
end
length2m1 = length(m1);
m = (reshape(m1,freqno,length2m1/freqno))';
N_ary_msg=m;
```

```
%
% Generate a differentially encoded matrix of complex values with unit magnitude and
% one of (2^n) equal phases. (Uses macro: difcdrft.m).
%
MDD = difcdrft(nary,m,fort);
[MDm MDn] = size(MDD);
MD = MDD;
Cmplx_mod_array = MDD;
%
% Form the frequency array of modulation values that include guard interval.
% (Uses macro: cmv2fa.m)
%
Fa = cmv2fa(N,MD);
Freq_array = Fa;
end
end
end
end
%________________________________________________________________________
```

6. Function: chancase.m

```
%______________________________________________________________________________
%
%       Title:          OPTIMAL INTERLEAVER CASES
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/17/97
%______________________________________________________________________________
%       INPUTS:
%               None
%
%       OUTPUTS:
%               None
%
%       SUBROUTINES USED:
%               chancdl.m
%
% CHANCASE:  This batch m-file performs numerous OFDM simulations using a channel three
% model (awgn.m plus chuhf.m) with various CDL interleaver cases.  This program helps to
% isolate which case offers the best reduction of row errror occurrences (optimal inteleaver case).
%______________________________________________________________________________
disp('____________________________________________________________________________');
fort = input('To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time
version: ');
freqno = input('Enter the # of OFDM frequencies (Note: Must be even): ');
N = input('Enter the number of FFT points (Note: This number must be larger than # of OFDM
frequencies): ');
svals = input('Enter specific integer seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]):
');
svals = round(svals);
chnmdl = input('Do you want to run channel model 0, channel model 1, channel model 2 or channel
model 3? (Enter 0,1,2 or 3): ');
        if chnmdl == 0
disp('Channel model 0 simulation performed.');
sigs = 0;
loss = 0;
dop = 0;
dly = 0;
        elseif chnmdl == 1
disp('Channel model 1 simulation performed.');
sigs = input('Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or
.003): ');
```

```
loss = 0;
dop = 0;
dly = 0;
          elseif chnmdl == 2
disp('Channel model 2 simulation performed.');
sigs = 0;
pthno = input('Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for
custom): ');
%
% Link parameters
%
                    if pthno == 3
% Link 3
loss = [0,3,9];
dop = [25,25,25];
dly = [0,.9,5.1];
                    elseif pthno == 2
% Link 2
loss = [0,5,15];
dop = [10,10,10];
dly = [0,.07,.8];
                    elseif pthno == 1
% Link 1
loss = [0,6];
dop = [1,10 0];
dly = [0,.01];
                    elseif pthno == 4
disp('Custom link simulation...')
loss = input('Enter the path loss in dB (Ex. [0 4 7]): ');
dop = input('Enter the doppler frequency in Hertz (Ex. [30 20 15]): ');
dly = input('Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): ');
                    end
          elseif chnmdl == 3
disp('Channel model 3 simulation performed.');
sigs = input('Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or
.003): ');
pthno = input('Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for
custom): ');
%
% Link parameters
%
                    if pthno == 3
% Link 3
loss = [0,3,9];
dop = [25,25,25];
dly = [0,.9,5.1];
```

```
                    elseif pthno == 2
% Link 2
loss = [0,5,15];
dop = [10,10,10];
dly = [0,.07,.8];
                    elseif pthno == 1
% Link 1
loss = [0,6];
dop = [1,10 0];
dly = [0,.01];
                    elseif pthno == 4
disp('Custom link simulation...')
loss = input('Enter the path loss in dB (Ex. [0 4 7]): ');
dop = input('Enter the doppler frequency in Hertz (Ex. [30 20 15]): ');
dly = input('Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): ');
                    end
end
totsym = input('Enter the total minimum number of symbols to simulate (Ex. 10000): ');
rowno = ceil(totsym/freqno);
          if totsym ~= (rowno*freqno)
disp(['Note: Based on the parameters thus far, the actual total number of symbols to be simulated
will be: ',int2str(rowno*freqno)]);
          end
pry_n = input('For the  interleaver, do you want to calculate all possible intermediate matrix
dimension pairs? (1 = yes, 0 = no): ');
pair1 = 1;
pair2 = rowno*freqno;
          if pry_n == 1
%
% Find all multiples of the data matrix based upon the number of rows (symbol #) and
% the number of columns (OFDM frequency number).  From the calculated list of multiples
% calculate all acceptable interleaver pairs   (Uses macro: intlvprs.m)
          Intrlvr_pairs = intlvprs(rowno,freqno);
          intlvrprs = Intrlvr_pairs;
          disp(' ')
          disp('For these input parameters, all possible acceptable interleaver dimension pairs are:')
          disp(Intrlvr_pairs)
          end
pairs = input(['Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering
[',int2str(pair1),' ',int2str(pair2),'], or [',int2str(pair2),' ',int2str(pair1),'], offers no interleaving
functionality): ']);
rintlv = pairs(1);
cintlv = pairs(2);
mary = input('Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): ');
nary = input('Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): ');
allcase = input('Test all interleaver cases(yes) or specific ones(no)? (1 = yes, 0 = no): ');
```

```
        if allcase == 1
        disp('All cases, 0 through 8, will be tested.');
        case = [0:8];
        elseif allcase == 0
        case = input('Enter specific case numbers from (0 to 8) (Ex. [0 4 5 8]): ');
        end
freqspace = round(480000/freqno);
Ng = input('Enter the guard interval length (Number of sample points): ');
picy_n = input('Do you want pictures? (1 = yes, 0 = no): ');
        if picy_n == 1
wait = input('How many seconds of delay between pictures? ');
wait = round(wait);
        elseif picy_n == 0
wait = 0;
        end
prnty_n = input('Do you want print outs? (1 = yes, 0 = no): ');
n = freqno;
k =  freqno;
blklgth = freqno;
pic = 0;
        for slp = 1:length(svals);
disp('_____________________________________________________________________________
_');
        Trial_nmbr = slp
        s = svals(slp)
        errcase = [];
        errtot = [];
                for lp = 1:length(case);
%
% If fort equals one, run the frequency simulation version; if fort equals zero, run
% the time version; else if fort equals two, run both versions.
%
%                    function                    [errmax,errors,freqerrs]                    =
chancdl(chnmdl,wait,prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,sigs,los
s,dly,dop,freqspace,fort);
[errmax,errors,freqerrs]                                                                     =
chancdl(chnmdl,wait,prnty_n,picy_n,pic,case(lp),s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,
sigs,loss,dly,dop,freqspace,fort);
                errtot = [errtot sum(errors)];
                errvect = [errvect,errtot];
                errcase = [errcase sum(errmax)];
                end
        pic = pic+1;
        end
casearry = [case;errcase]
%
```

```
%       *******************************************************
%       *                    Plots                            *
%       *******************************************************
%
figure(pic+13)
bar(case,errcase)
grid
orient tall
        if fort == 1
title([int2str(pic),': Maximum Error Total Vs. Interleaver Case Number (Freq. Diff. Enc.) (OFDM
Freq. # = ',int2str(freqno),')'])
        elseif fort == 0
title([int2str(pic),': Maximum Error Total Vs. Interleaver Case Number (Time Diff. Enc.) (OFDM
Freq. # = ',int2str(freqno),')'])
        end
xlabel(['CDL Interleaver Case Number'])
ylabel(['Maximum Error Count For Any Symbol Row  (Seed = ',int2str(s),')'])
axis([-.5 8.5 0 (max(errcase)+1)])
        if prnty_n == 1;
        print
        pause(10)
        end
pause(wait);
%
figure(pic+14)
bar(case,errtot)
grid
orient tall
title([int2str(pic),': CHN3CASE: Error Totals Vs. Interleaver Case Number'])
xlabel(['CDL Interleaver Case Number'])
ylabel(['Sigma: (',num2str(min(sigs)),'-',num2str(max(sigs)),') Error Total'])
axis([-.5 8.5 (min(errtot)-1) (max(errtot)+1)])
        if prnty_n == 1;
        print
        pause(10)
        end
pause(wait);
        end
disp('**********************************************************************')
disp(' ')
disp('Case batch run is finished! ')
%_____________________________________________________________________________
```

7. Function: chancdl.m

```
% function [errmax,errors,freqerrs] =
chancdl(chnmdl,wait,prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,N
g,sigs,loss,dly,dop,freqspace,fort)
%____________________________________________________________________________
%
%       Title:          SYSTEM MODEL0 - 3 SIMULATION (AWGN & MULTIPATH
                        FADING CHANNEL)
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/29/97
%____________________________________________________________________________
%       INPUTS:
%               chnmdl  - Selects the desired channel model (model 0 - 3)
%               wait    - Delay in seconds to pause after displaying a plot
%               prnt    - Allows print outs of plots if true
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               case    - Variable indicating the deinterleaving method to be used (9 different
                          cases)
%               s       - Seed parameter for random number generator
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               N       - Number of FFT frequency sample points, must be larger than freqno
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%               n       - Integer number representing code word symbol length
%               k       - Integer number representing information word symbol length
%                         (Typically: n>k)
%               blklgth - Block number indicating number of symbols over which the
%                         Reed-Soloman code can perform error detection and correction
%               Ng      - Number of time domain samples for the addition of guard interval
%               sigs    - Noise parameter for calculating Eb/No (square root of noise variance)
%               loss    - Multipath free space loss in dB (vectors accepted)
%               dly     - Multipath delay in microseconds (vectors accepted)
%               dop     - Doppler frequency in Hertz (vectors accepted)
%               freqspace - Frequency spacing between individual OFDM carriers in Hz
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               errmax  - Maximum total error count in any given sink message symbol row
%               errors    - Number of sink message symbol errors, if any
%               freqerrs  - Number of sink message symbol errors vs. OFDM frequency number
%
%       SUBROUTINES USED:
%               cdrcdlft.m, tda.m, awgn.m, chuhf.m, itda.m, decdrcdl.m, check.m
%
% CHANCDL:  This m-file performs an OFDM simulation using multiple channel models (0-3)
% (awgn.m & chuhf.m).  A check is performed comparing the source message with the
% sink message to determine if any symbol errors occurred as a result of channel noise
% corruption.  R-S symbol error correction is possible.  The m-file initiates by querying the user
% for input configurations.
%______________________________________________________________________________
% USAGE: function [errmax,errors,freqerrs] =
chancdl(chnmdl,wait,prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,sigs,l
oss,dly,dop,freqspace,fort)
%______________________________________________________________________________
function [errmax,errors,freqerrs] =
chancdl(chnmdl,wait,prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,sigs,l
oss,dly,dop,freqspace,fort)
sigvect = sigs;
klgth = length(k);
chklp = 1;
errvect = [];
freqerrmx = [];
errsperpr = [];
Es_No = [];
sermx = [];
rowerrmx = [];
symno = rintlv*cintlv/freqno;
for lp = 1:length(sigvect);
%
% Randomly generate a source message, encode as an OFDM frequency array.  Perform CDL
% inteleaving to overcome channel induced noise errors.  (Uses macro: cdrcdlft.m)
%
[xmt,modvals,B,nsymno] = cdrcdlft(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,fort);
%
% Transform into the time domain by performing the IFFT and add guard interval.
% Generates the OFDM sub-carriers.  (Uses macro: tda.m)
%
xmtifft = tda(Ng,xmt);
xmtpts = 1:length(xmtifft);
        if chnmdl == 0
%
% Transmit the encoded message signal through the perfect, noise-free channel.
% Channel model 0.
%
```

```
sandn = xmtifft;
        elseif chnmdl == 1
%
% Transmit the encoded message signal through AWGN channel only.
% Channel model 1.  (Uses macro: awgn.m)
%
disp(['Sigma = ',num2str(sigvect(lp))]);
sandn = awgn(xmtifft,s,N,sigvect(lp));
        elseif chnmdl == 2
%
% Transmit the encoded message signal through multipath channel only.
% Channel model 2.  (Uses macro: chuhf.m)
%
sandn = chuhf(s+1,xmtifft,loss,dly,dop,N,freqspace);
        elseif chnmdl == 3
%
% Transmit the encoded message signal through multipath channel and AWGN channel.
% Channel model 3.  (Uses macros: chuhf.m and awgn.m)
%
sandmltpth = chuhf(s+1,xmtifft,loss,dly,dop,N,freqspace);
disp(['Sigma = ',num2str(sigvect(lp))]);
sandn = awgn(sandmltpth,s,N,sigvect(lp));
        end
%
% Remove guard interval and perform FFT to put back into frequency domain.
% (Uses macro: itda.m)
%
sandnfft = itda(Ng,sandn);
%
% Decode the received message signal, deinterleave and recover the sink message.
% (Uses macro:  decdrcdl.m)
%
K = (length(modvals(1,:)))/2;
[rcvd,M,MM] = decdrcdl(picy_n,pic,case,K,sandnfft,nsymno,freqno,rintlv,cintlv,mary,nary,fort);
Transmitted_msg = B;
Received_msg = rcvd;
%
% Compare the source message against the sink message and check for errors.
% Returns error report.  (Uses macro:  check.m)
%
[errors,freqerrs,errmx,rowerrs] = check(pic,B,rcvd,n,k(chklp),blklgth);
errvect = [errvect,errors];
freqerrmx = [freqerrmx;freqerrs];
rowerrmx = [rowerrmx;rowerrs];
crntEs_No = 1/(2 * N * (sigvect(lp)^2));
Es_No = [Es_No,crntEs_No];
```

```
Es_Nodb = 10*log10(Es_No);
end
ser = errvect/(symno*freqno);
sermx = [sermx;ser];
errsum = sum(errvect);
errsperpr = [errsperpr,errsum];
errmax = max(rowerrmx');
%
%       ********************************************************
%       *                    Plots                             *
%       ********************************************************
%
%
% Generate a constellation plot of complex modulation values (Ideal, pre-transmitted),
% only if picy_n is true.
%
if picy_n == 1
figure(pic+1)
plot(modvals,'*')
hold on;
plot(0,0,'+')
hold off;
title(['Transmitted Signal ',int2str(2^nary),'-ary Constellation Plot'])
xlabel(['Magnitude = 1'])
axis('square');
orient tall
grid
if prnt == 1;
print
pause(10);
end
pause(wait);
end
%
% Plot the transmitter frequency array.
%
if picy_n == 1
figure(pic+2)
plot([0:N-1],abs(xmt),'*')
title(['Frequency Array Plot (number of FFT frequency points are ',int2str(N),')'])
xlabel(['Guard interval length is ',int2str(N-freqno)])
axis('square');
orient tall
grid
if prnt == 1;
print
```

```
pause(10)
end
pause(wait);
end
%
% Plot the Magnitudes of the Transmitted Message Array (unit Magnitude),
% only if picy_n is true.
%
if picy_n == 1
figure(pic+3)
surf(abs(modvals));
shading interp
grid
orient tall
title(['Magnitude of Transmitted Signal (Unity Magnitude)'])
xlabel('OFDM Freq #')
ylabel('Symbol Row Number')
zlabel(['Magnitude (seed = ',int2str(s),')'])
if prnt == 1;
print
pause(10)
end
pause(wait);
end
%
% Plot the received signal constellation plot before differential decoding,
% only if picy_n is true.
%
if picy_n == 1
figure(pic+6)
plot(M,'*')
hold on;
plot(0,0,'+')
hold off;
title(['Received ',int2str(2^nary),'-ary Signal Constellation Plot, before Differential Decoding'])
orient tall
axis('square');
grid
if prnt == 1;
print
pause(10)
end
pause(wait);
end
%
% Generate a constellation plot of received signal complex modulation values after
% differential decoding, only if picy_n is true.
%
if picy_n == 1
figure(pic+7)
plot(MM,'+')
hold on;
plot(0,0,'+')
hold off;
title(['Received ',int2str(2^nary),'-ary Signal Constellation Plot, After Differential Decoding'])
orient tall
axis('square');
grid
if prnt == 1;
print
pause(10)
end
pause(wait);
end
%
% Plot Magnitudes of Received Message Array, if picy_n is true.
%
if picy_n == 1
roty_n = input('Do you want to rotate 3-D plot? (yes = 1, no = 0): ');
figure(pic+8)
surf(abs(M));
shading interp
grid
orient tall
title(['Magnitude Variation of Received Signal (Sigma = ',num2str(sigvect(lp)),')'])
xlabel('OFDM Freq #')
ylabel('Symbol Row Number')
zlabel(['Magnitude (seed = ',int2str(s),')'])
if roty_n == 1
% Rotate the 3D plot
for k = 1:5
view(-70+10*k,15+10*k)
disp(' ');
disp('Press "enter" to rotate plot...');
pause
end
end
if prnt == 1;
print
pause(10)
end
pause(wait);
```

```
end
%
% 3-D Error Distribution Plot With Interleaving.
%
figure(pic+9)
meshz(errmx)
%shading interp
            if dop == [25,25,25]
title(['Link 3: Error Distribution With Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),') (case =',int2str(case),') (Intlvr Pair = ',int2str(rintlv),',',int2str(cintlv),')'])
    elseif dop == [10,10,10]
title(['Link 2: Error Distribution With Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),') (case =',int2str(case),') (Intlvr Pair = ',int2str(rintlv),',',int2str(cintlv),')'])
    elseif dop == [1,10,0]
title(['Link 1: Error Distribution With Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),') (case =',int2str(case),') (Intlvr Pair = ',int2str(rintlv),',',int2str(cintlv),')'])
    else
title(['Custom Link: Error Distribution With Intlving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),') (case =',int2str(case),') (Intlvr Pair = ',int2str(rintlv),',',int2str(cintlv),')'])
            end
%axis([0 freqno 0 symno 0 max(max(errmx))+1])
xlabel(['OFDM Freq. # (Total = ',int2str(freqno),')'])
ylabel(['Sym. Row # (Total # = ',int2str(symno*freqno),')'])
zlabel(['Error Occurance (Total = ',int2str(errsum),') (seed = ',num2str(s),')'])
text(-150,0,1.95,['Error Correction = ',int2str(floor((n-k)/2))])
grid
orient tall
        if prnt == 1;
        %print
        %pause(10)
        end
pause(wait);
if length(sigs) > 1
    if picy_n == 1
%
% 3-D Error Distribution Plot Vs. Row #.
%
figure(pic+10)
surf((1:symno),sigvect,rowerrmx)
shading interp
    if dop == [1,10,0]
title(['Link 1: Error Distribution Vs. Row Number and Sigma (case =',int2str(case),') (Interleaver
Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
    elseif dop == [10,10,10]
title(['Link 2: Error Distribution Vs. Row Number and Sigma (case =',int2str(case),') (Interleaver
Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
```

```
        elseif dop == [25,25,25]
title(['Link 3: Error Distribution Vs. Row Number and Sigma (case =',int2str(case),') (Interleaver
Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
        end
xlabel(['OFDM Row # (R-S = ',int2str(floor((n-k)/2)),')'])
ylabel(['Sigma Values (Seed = ',num2str(s),')'])
zlabel(['Error Number (Total errors = ',int2str(sum(errvect)),')'])
grid
orient tall
                if prnt == 1;
                print
                pause(10)
                end
pause(wait);
%
% 3-D Error Distribution Plot Vs. OFDM Frequency.
%
figure(pic+11)
surf((1:freqno),sigvect,freqerrmx)
shading interp
        if dop == [1,10,0]
title(['Link 1: Error Distribution Vs. OFDM Frequencies & Sigma (case =',int2str(case),')
(Interleaver Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
        elseif dop == [10,10,10]
title(['Link 2: Error Distribution Vs. OFDM Frequencies & Sigma (case =',int2str(case),')
(Interleaver Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
        elseif dop == [25,25,25]
title(['Link 3: Error Distribution Vs. OFDM Frequencies & Sigma (case =',int2str(case),')
(Interleaver Pair = ',int2str(rintlv),' , ',int2str(cintlv),')']);
        else
title(['Custom Link: Error Distribution With Intlving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),') (case =',int2str(case),') (Intlvr Pair = ',int2str(rintlv),',',int2str(cintlv),')'])
        end
xlabel(['OFDM Freq # (R-S = ',int2str(floor((n-k)/2)),')'])
ylabel(['Sigma Values (Seed = ',num2str(s),')'])
zlabel(['Error Number (Total errors = ',int2str(errsum),')'])
grid
orient tall
                if prnt == 1;
                %print
                %pause(10)
                end
pause(wait);
        end
if errsum ~= 0
%
```

```
% 2-D Error Performance Curve showing SER vs. Es/No.
%
figure(pic+12)
semilogy(Es_Nodb,ser)
grid
if fort == 1
                        if dop == [1,10,0]
title(['Link 1: Performance graph: Symbol Error Rate vs. Es/No (Freq. Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        elseif dop == [10,10,10]
title(['Link 2: Performance graph: Symbol Error Rate vs. Es/No (Freq. Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        elseif dop == [25,25,25]
title(['Link 3: Performance graph: Symbol Error Rate vs. Es/No (Freq. Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        else
title(['Custom Link Performance graph: Symbol Error Rate vs. Es/No (Freq. Diff. Enc.) (Total
errors = ',int2str(sum(errvect)),')'])
                        end
elseif fort == 0
                        if dop == [1,10,0]
title(['Link 1: Performance graph: Symbol Error Rate vs. Es/No (Time Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        elseif dop == [10,10,10]
title(['Link 2: Performance graph: Symbol Error Rate vs. Es/No (Time Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        elseif dop == [25,25,25]
title(['Link 3: Performance graph: Symbol Error Rate vs. Es/No (Time Diff. Enc.) (Total errors =
',int2str(sum(errvect)),')'])
                        else
title(['Custom Link Performance graph: Symbol Error Rate vs. Es/No (Time Diff. Enc.) (Total
errors = ',int2str(sum(errvect)),')'])
                        end
end
text(min(ceil(Es_Nodb)),.18,['Loss = [',num2str(loss),']']);
text(min(ceil(Es_Nodb)),.12,['Delay = [',num2str(dly),']']);
text(min(ceil(Es_Nodb)),.08,['Doppler = [',num2str(dop),']']);
xlabel(['Es/No (dB) (# of OFDM = ',int2str(freqno),') (case =',int2str(case),') (Interleaver pair =
',int2str(rintlv),' , ',int2str(cintlv),') M-ary = ',int2str(2^mary),', N-ary = ',int2str(2^nary)]);
ylabel(['Sigma Range: (',num2str(min(sigs)),'-',num2str(max(sigs)),') (R-S = ',int2str(floor((n-
k)/2)),') (Symbol # = ',int2str(symno*freqno),') (Seed = ',num2str(s),')']);
orient tall
end
        if prnt == 1;
        print
        pause(10)
        end
pause(wait);
end
%_____________________________________________________________
```

8. Function: check.m

```
% function [error_no,freqerrs,errmx,rowerrs] = check(pic,x,y,n,k,blklgth)
%____________________________________________________________________________
%
%       Title:          SOURCE AND SINK MESSAGE CHECKER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/23/96
%____________________________________________________________________________
%       INPUTS:
%               pic - Argument passed by another m-file to indicate the loop number
%               x  - Variable sized matrix (source message array)
%               y  - Variable sized matrix (sink message array , same size as x)
%               n  - Integer number representing code word bit length
%               k  - Integer number representing information word bit length
%                   (Typically: n>k)
%               blklgth - Block number indicating number of symbols over which the
%                       Reed-Solomon code can perform error detection and correction
%
%       OUTPUTS:
%               error_no - Number of symbol errors that occur after checking
%               freqerrs  - Number of message array symbol errors vs. OFDM freq. #
%                errmx    - Matrix of message array symbol error locations (indicated by a "1")
%                rowerrs  - Number of message array symbol errors vs. row #
%
% CHECK: This m-file locates the positions in input matrices x and y that do
% not agree (used for comparing source and sink message arrays).
% It generates a one in matrix, "errors" if they do not agree and a zero
% if they do agree. A calculation of the total number of existing errors along with
% the error positions in the "errors" matrix is also performed. Both input matrices
% must be the same size.  A Reed-Solomon correction scheme is also emulated by allowing
% for (n-k)/2 symbols within the "errors" block, blklgth, to be corrected.  For errored
% symbols numbering in excess of (n-k)/2, they remain in error and are indicated as
% such.  The error corrections occur across the matrix columns (OFDM frequencies).
%____________________________________________________________________________
% USAGE: function [error_no,freqerrs,errmx,rowerrs] = check(pic,x,y,n,k,blklgth)
%____________________________________________________________________________
function [error_no,freqerrs,errmx,rowerrs] = check(pic,x,y,n,k,blklgth)
        if blklgth > n
        disp(' ')
        disp('ERROR! The block length, blklgth, must be equal or less than the code word length,
n.')
```

```
        disp('Please enter a smaller value for blklgth, or change n.')
        disp(' ')
        elseif blklgth <= n
                if n < k
                disp(' ')
                disp('ERROR! The code word length, n, must be equal or larger than the
information length, k.')
                disp('Please enter a larger value for n, or change k to a smaller number.')
                disp(' ')
                elseif n >= k
First_matrix = x;
Second_matrix = y;
[rx cx] = size(x);
%
% Compare inputs x and y and generate error matrix, "errors".
%
        errors = (x~=y);
%
% Find the error distribution vs. OFDM frequencies.
%
        freqerrs = sum(errors);
%
% Find the error locations in "errors" where elements in x and y differ.
%
        Error_locations = (find(errors))';
%
% Calculate the total number of errors occurring in "errors".
%
        Error_number = sum(sum(errors));
%
% Find how many correct symbols there are.
%
        Correct_smbl_num = (size(y,1)*size(y,2)) - Error_number;
%
% Reed-Solomon 8-bit symbol correction for (n-k)/2 symbols.
%
symcorr = floor((n-k)/2);
        if blklgth <= (n-k)
disp('Error!!! The block length is too short for the given n and k values.')
disp(' ')
        elseif blklgth > (n-k)
errtrans = errors';
%
% Reshape the error matrix as a vector of errors.
%
errvect = errtrans(:)';
```

```
%
% Pad the error vector with zeros if not an even multiple of the
% block length.
%
blkrem = rem(length(errvect),blklgth);
        if blkrem ~= 0;
        zeropad = zeros(blklgth - blkrem);
        errvectpad = [errvect zeropad(1,:)];
        elseif blkrem == 0;
        errvectpad = errvect;
        end
%
% Calculated the number of blocks in the message
%
blknos = length(errvectpad)/blklgth;
%
% Initialize empty vectors.
%
errcorct = [];
errblksum = [];
%
% For each block determine the # of errors contained.  If less than or
% equal to (n-k)/2 then correct.  If greater, do nothing and check the
% next block.  Continue this for all blocks in the message.
%
        for lp = 1:blknos;
        errblk = errvectpad(((blklgth*(lp-1))+1):(blklgth*lp));
        errblklgth = length(errblk);
                if sum(errblk) <= symcorr;
                noerr = zeros(errblklgth);
                errblk = noerr(1,:);
                elseif sum(errblk) > symcorr;
                errblk = errblk;
                end
        errcorct = [errcorct errblk];
        errblksum = [errblksum sum(errblk)];
        end
%
% A new error vector is formed that contains corrected symbols and
% uncorrected (errored) symbols after R-S decoding.
%
newerrvect = errcorct(1:length(errvect));
%
% Find the total number of errors in the corrected message.
%
errtot = sum(newerrvect);
```

```
RSerrs = (reshape(newerrvect,size(errors,2),size(errors,1)))';
%
% Find the error distribution vs. OFDM frequencies.
%
freqerrs = sum(RSerrs);
errindex = (find(RSerrs))';
RSerrtot = sum(errblksum);
RSerrdif = Error_number - RSerrtot;
errperblk = [(1:blknos);errblksum];
%
% Check to see if x and y are the same.  If not, display error message.
%
if  x == y;
disp('GREAT!!! there are no errors.')
error_no = 0;
errmx = errors;
rowerrs = sum(errors');
else
disp('WARNING! Errors were detected!')
disp(' ')
        if n ==k
        disp('WARNING!: Since n = k, there is no R-S error correcting possible.')
        disp(' ')
        end
disp(['For the given input parameters: n = ',int2str(n),' and k = ',int2str(k),', the Reed-Solomon code
is capable'])
disp(['of correcting ',int2str(symcorr),' errors.'])
disp(' ')
%
% RS code was able to correct all errors
%
        if errtot == 0
                Pre_RS_error_matrix = errors;
                disp('EXCELLENT: The Reed-Solomon code corrected all detected errors!')
                disp(['Originally the error total was: ',int2str(Error_number)])
                disp(' ')
                error_no = 0;
                errmx = zeros(rx,cx);
                rowerrs = sum(errmx');
%
% RS code was able to correct some errors but not all of them
%
        elseif  errtot < Error_number
                Pre_RS_error_matrix = errors;
                Post_RS_error_matrx = RSerrs;
                errmx = RSerrs;
                rowerrs = sum(errmx');
                disp('OOOPS: The Reed-Solomon code corrected some detected errors, but not
all.')
                disp(['Originally the error total was: ',int2str(Error_number)])
                disp(' ')
                disp(['After R-S decoding, the error number was reduced to: ',int2str(RSerrtot)])
                disp(' ')
                error_no = RSerrtot;
                disp(['The total number of correct symbols are: ',int2str((size(y,1)*size(y,2)) -
RSerrtot)])
                disp(' ')
                disp('The error number distribution per block number is:')
                disp(errperblk)
                %figure(pic+3)
                %bar((1:blknos),errblksum)
                %axis([0.5 (blknos+.5) 0 (max(errblksum)+1)])
                %title(['Simulation # ',int2str(pic),': Error Distribution Per Message Block (Error
count = ',int2str(error_no),')'])
                %xlabel(['Message Block Number (block size: ',int2str(blklgth),' symbols)'])
% RS code did not correct any errors
%
        elseif  errtot == Error_number
                Error_matrix = errors;
                errmx = errors;
                rowerrs = sum(errors');
                disp('OOOPS!: The Reed-Solomon code did not correct any errors.')
                disp('Perhaps a more powerful R-S code is required.')
                disp(' ')
                disp(['The total number of error occurrences is: ',int2str(Error_number)])
                disp(' ')
                error_no = errtot;
                disp('The error number distribution per block number is:')
                disp(errperblk)
                %figure(pic+4)
                %bar((1:blknos),errblksum)
                %axis([0.5 (blknos+.5) 0 (max(errblksum)+1)])
                %title([' Simulation # ',int2str(pic),': Error Distribution Per Message Block. (Error
count = ',int2str(error_no),')'])
                %xlabel(['Message Block Number (block size: ',int2str(blklgth),' symbols)'])
                end
        end
end
end
end
disp('____________________________________________');
%_____________________________________________________________________________________________
```

9. Function: chn0cdl.m

```
% function [errmax,errors,freqerrs] =
chn0cdl(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,fort);
%__________________________________________________________________________
%
%       Title:          MODEL ZERO (NOISE FREE) SIMULATION
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/22/97
%__________________________________________________________________________
%       INPUTS:
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               case    - Variable indicating the deinterleaving method to be used (9 different
%                         cases)
%               s       - Seed parameter for random number generator
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               N       - Number of FFT frequency sample points, must be larger than freqno
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%               n       - Integer number representing code word symbol length
%               k       - Integer number representing information word symbol length
%                         (Typically: n>k)
%               blklgth - Block number indicating number of symbols over which the
%                         Reed-Soloman code can perform error detection and correction
%               Ng      - Number of time domain samples for the addition of guard interval
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               errmax   - Maximum total error count in any given sink message symbol row
%               errors   - Number of sink message symbol errors, if any
%               freqerrs - Number of sink message symbol errors vs. OFDM frequency number
%
%       SUBROUTINES USED:
%               cdrcdlft.m, tda.m, itda.m, dcdrcdlf.m, check.m
%
% CHN0CDL: This m-file performs an OFDM simulation using a channel zero model.
% This function verifies correct operation of the OFDM transmitter and OFDM receiver.
% A check is performed comparing the source message with the sink message to
% determine if any errors occurred.
```

```
%______________________________________________________________________________
% USAGE: function [errmax,errors,freqerrs] =
chn0cdl(prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,fort)
%______________________________________________________________________________
function [errmax,errors,freqerrs] =
chn0cdl(prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,fort)
disp(' ');
klgth = length(k);
chklp = 1;
errvect = [];
freqerrmx = [];
errsperpr = [];
Es_No = [];
sermx = [];
rowerrmx = [];
symno = rintlv*cintlv/freqno;
%
% Randomly generate a block message and encode as a OFDM frequency array.  Perform
% frequency/time differential encoding and interleaving to overcome channel induced
% noise errors.  (Uses macro: cdrcdlft.m)
%
% function [Fa,MD,B,nsymno] = cdrcdlft(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,fort);
[xmt,modvals,B,nsymno] = cdrcdlft(picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,fort);
Random_Source_Msg = B
%
% Put into the time domain by performing the IFFT and add periodic precursor.
% (Uses macro: tda.m)
%
xmtifft = tda(Ng,xmt);
xmtpts = 1:length(xmtifft);

%
% Remove precursor and take FFT to put back into frequency domain.
% (Uses macro: itda.m)
%
sandnfft = itda(Ng,xmtifft);
%
% Decode the received message signal and generate the sink message.
% (Uses macro:  decdrcdl.m)
%
K = (length(modvals(1,:)))/2;
[rcvd,M] = decdrcdl(picy_n,pic,case,K,sandnfft,nsymno,freqno,rintlv,cintlv,mary,nary,fort);
Transmitted_msg = B;
Sink_msg = rcvd
%
% Check the source message against the sink message for errors.  Returns error report.
% (Uses macro:  check.m)
%
[errors,freqerrs,errmx,rowerrs] = check(pic,B,rcvd,n,k(chklp),blklgth);
errvect = [errvect,errors];
freqerrmx = [freqerrmx;freqerrs];
rowerrmx = [rowerrmx;rowerrs];
end
ser = errvect/(symno*freqno);
sermx = [sermx;ser];
errsum = sum(errvect);
errsperpr = [errsperpr,errsum];
errmax = max(rowerrmx');
        if errsum == 0;
disp('Test Passed!!!')
disp(' ')
        elseif errsum ~= 0;
disp('WARNING! Test Failed!')
disp(' ')
        end
%______________________________________________________________________________
```

10. Function: chuhf.m

```
% function [y] = chuhf(s,x,loss,dly,dop,N,freqspace)
%_____________________________________________________________________________
%
%       Title:          UHF CHANNEL MODEL (MULTIPATH CHANNEL MODEL 2)
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  3/28/97
%_____________________________________________________________________________
%       INPUTS:
%               s         - Seed parameter for random number generator
%               x         - input time domain matrix of samples
%               loss      - vector of multipath weights in dB.
%               dly       - vector of multipath delays (in microseconds), same length
%                           as c
%               dop       - vector of fading bandwidths (Hertz)
%               N         - Number of frequncy points for FFT
%               freqspace - OFDM tone spacing in Hertz
%
%       OUTPUTS:
%               y - output time domain matrix same size as x with multipath distortions included
%
%       SUBROUTINES USED:
%               dline.m, ofst.m, ray_dop.m
%
% CHUHF: This m-file is the uhf channel model characterized by multipath, power loss of
% received signal levels (RSL) and Doppler frequency shifting. The input is an array of COFDM
% transmitted time domain samples. The output is a time domain sample array with identical
% dimensions with multipath distortions included. This m-file rperesents channel model 2.
%_____________________________________________________________________________
% USAGE: function [y] = chuhf(s,x,loss,dly,dop,N,freqspace)
%_____________________________________________________________________________
function y = chuhf(s,x,loss,dly,dop,N,freqspace)

c = 10 .^ (-loss ./ 20);
deltat = 1 / (N * freqspace);
d = (dly .* .000001) ./ deltat;
e = dop ./ freqspace;
[L,Nt] = size(x);
D=length(d);

        x = x.';
        x = x(:).';
%
% D paths with delays from d. (Uses macro dline.m)
%
        xd = dline(x,d);
[rr,cc] = size(xd);
        x = xd(1,:);
%
% Offsets direct path by .7 of max doppler freq. (Uses macro ofst.m)
%
        xo = ofst(.7*e(1),N,x);
%
% First path with no fading. (Uses macro ray_dop.m)
%
                for l = 1:D
                a = ray_dop(s,cc,N,e(l));
                xd(l,:) = a.*xd(l,:);
                end
%
% Sums the fading paths
%
        y = c*xd;
%
% Adds in the First path without fading for the GSM-Ricean.
%
        y = y+xo;

y = y(1:L*Nt);

y = reshape(y,Nt,L).';
%______________________________________________________________________
```

11. Function: cmv2fa.m

```
% function [X] = cmv2fa(N,M)
%________________________________________________________________________________
%
%       Title:          COMPLEX FREQUENCY ARRAY GENERATOR
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/7/97
%________________________________________________________________________________
%       INPUTS:
%               N - Number of FFT points.
%               M - Array of complex modulation values (frequency domain).
%
%       OUTPUTS:
%               X - Frequency array of complex values with center zero pad.
%
%       NOTE: This m-file performs the inverse function of m-file fa2cmv.m
%
% CMV2FA: This M-file accepts complex modulation values in the array M and puts them
% into frequency array, X. Array, X, is complex of length N and includes a pad of zeros
% in the center representing the guard band.
%________________________________________________________________________________
% USAGE: function [X] = cmv2fa(N,M)
%________________________________________________________________________________
function X = cmv2fa(N,M)
[m n] =size(M);
%
% Determine if there are an even number of columns, and keep M the same if even.
%
if rem(n,2) ==0;
M=M;
else
%
% If there are an odd number of columns, insert a column of zeros at the beginning.
%
M =[zeros(m,1) M];
end
[m n]=size(M);
K=round(n/2);
%
% Generate a matrix of zeros with m rows and N columns.
%
```

```
X=zeros(m,N);
%
% Interchange the array of complex modulation value elements to form the appropriate
% frequency array and include an interval of zeros in the middle for the guard interval
%
X(:,1:K)=M(:,K+1:2*K);
X(:,N-K+1:N)=M(:,1:K);
%_______________________________________________________________________
```

12. Function: cmvdifck.m

```
% function cmvdifck(s,symno,freqno,N,mary,nary)
%_______________________________________________________________________
%
%       Title:          FREQUENCY ARRAY & DIFFERENTIAL ENCODER/DECODER
                        VERIFIER
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  5/2/97
%_______________________________________________________________________
%       INPUTS:
%               s       - Seed parameter for random number generator
%               symno   - Number of symbol rows in message block
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               N       - Number of frequency points for frequency array generation
%               mary    - Initial M-ary OFDM symbol format
%               nary    - Final N-ary PSK symbol format
%
%       OUTPUTS:
%               None
%
% CMVDIFCK:  This batch m-file verifys correct functionality of the differential encoder/decoder
% & the frequency array arranger/unarranger.
%_______________________________________________________________________
% USAGE: function cmvdifck(s,symno,freqno,N,mary,nary)
%_______________________________________________________________________
function cmvdifck(s,symno,freqno,N,mary,nary)
disp('____________________________________________________________________')
disp('This m-file checks the correctness of the differential encoder/decoder & the frequency
arrangers.')
fort = input('To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time
version: ');
%
% Generate random m-ary message array.
%
B=marymsg(mary,s,symno,freqno);
Source_msg=B
[Br Bc] = size(B);
Bt=B';
Bvect = Bt(:)';
si = Bvect;
Bi = reshape(si,Bc,Br)';
```

```
%
% Reconstruct the matrix block of m-ary symbols into an equivalent
% information block using n-ary symbols.  For the case when m=256 (256-ary) and n=16 (16-ary)
% the reshaped matrix will be twice the size of the initial matrix.  Padding of zeros
% may be necessary for certain m-ary and n-ary combinations.  The expansion of the
% original m-ary message block is along the row dimension after conversion to n-ary
% symbols and for the case when (m>n).  (Uses macros: mb.m and bm.m).
%
m1=bm(nary,mb(mary,Bi));
lengthm1 = length(m1);
nsymno = lengthm1;
remm1 = rem(lengthm1,freqno);
        if remm1 == 0;
        m1 = m1;
        else
        zero = zeros(freqno - remm1);
        m1 =[m1 zero(1,:)];
end
length2m1 = length(m1);
m = (reshape(m1,freqno,length2m1/freqno))';
N_ary_msg=m;
%
% Generate a differentially encoded matrix of complex values with unit magnitude and
% one of (2^n) equal phases. (Uses macro: difcdrft.m).
%
MDD = difcdrft(nary,m,fort);
[MDm MDn] = size(MDD);
MD = MDD;
Cmplx_mod_array = MDD;
%
% Form the frequency array of modulation values that include guard interval.
% (Uses macro: cmv2fa.m)
%
Fa = cmv2fa(N,MD);
Freq_array = Fa;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Generate the corresponding complex modulation values from the received frequency
% array. (Uses macro: fa2ma.m)
%
K = (length(MD(1,:)))/2;
M =fa2cma(K,Fa);
Cmplx_mod_vals = M;
%
% Perform differential decoding. (Uses macro: dfdcdrft.m)
%
```

```
naryp = nary;
[s,MM] = dfdcdrft(naryp,nary,M,fort);
[L,cc] = size(s);
strans = s';
svect = strans(:)';
corrs = svect(1:nsymno);
%
% Convert from N-ary symbols to the final message format of M-ary symbols
% (Uses macros: mb.m and bm.m)
%
nsymno;
Br = bm(mary,mb(nary,corrs));
lengthBr = length(Br);
rmndr = rem(length(Br),freqno);
          if rmndr == 0;
          Br = Br;
          elseif rmndr ~=0;
          Br = Br(1:(lengthBr-rmndr));
end
rcvd = (reshape(Br,freqno,length(Br)/freqno))';
[Br Bc] = size(rcvd);
SYNC = [];
sr = rcvd';
si = sr(:)';
sd = si;
outmsg = reshape(sd,Bc,Br)';
Sink_Msg = outmsg
%
%
% Check results for correctness. (Uses m-file check.m).
[error_no,freqerrs,errmx,rowerrs] = check(0,B,rcvd,freqno,freqno,freqno);
          if sum(rowerrs) == 0
disp('TEST PASSED!!!');
          elseif sum(rowerrs) ~= 0
disp('OOOPS - TEST FAILED!')
          end
disp('________________________________________________________________')
%________________________________________________________________
```

13. Function: coderift.m

```
% function [Fa,MD,B,nsymno] = coderift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,fort)
%_______________________________________________________________________________
%
%       Title:          COFDM ENCODER WITHOUT INTERLEAVING
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/3/97
%_______________________________________________________________________________
%       INPUTS:
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               s       - Seed parameter for random number generator
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               N       - Number of FFT frequency sample points, must be larger than freqno
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               Fa   - Frequency array of arranged modulation values
%               MD   - Matrix of differentially encoded complex values (unit magnitude)
%                      and one of N-ary possible phases (N-PSK)
%               B    - Message matrix of M-ary symbols
%               nsymno - Number of N-ary generated symbols
%
%       SUBROUTINES USED:
%               marymsg.m, mb.m, bm.m, difcdrft.m, cmv2fa.m
%
%       NOTE: This m-file performs the inverse function of m-file decdrift.m
%
% CODERIF: This m-file generates a random array of M-ary message symbols as a function
% of the inputs rintlv and freqno, and returns a matrix of equivalent differentially
% encoded complex numbers, MD, with unit magnitude and one of N possible phases (N-ary).
% The message vector is initially formatted as M-ary OFDM symbols and reshaped into a
% matrix with values between 0 and 2^M.  The matrix is  reformatted as N-ary PSK symbols
% with values between 0 and 2^N and depending on fort, is either frequency or time
% differentially encoded before finally being converted to complex values.  The matrix
% of M-ary symbols is also returned as output matrix, B.
% (NOTE: No interleaving is performed)
```

```
%________________________________________________________________________
% USAGE: function [Fa,MD,B,nsymno] =
coderift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,fort)
%________________________________________________________________________
function [Fa,MD,B,nsymno] = coderift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,fort);
%
% Determine if the number of OFDM frequencies are even (# of matrix columns), indicated
% by the "freqno" parameter.  If odd go to error message.  Odd frequencies are not
% allowed since the formation of the frequency array is symmetrical.
%
        if rem(freqno,2) ~= 0
        disp('ERROR: The number of matrix columns, freqno, representing OFDM frequencies
must be an even number!')
        elseif rem(freqno,2) == 0
%
% Determne if the row and column interleave parameters are greater than freqno when
% multiplied together.  If not, then display error message and stop.
%
        if (rintlv*cintlv) < (freqno)
        disp(' ')
        disp('ERROR: The row and column interleave parameters are not compatible with # of
OFDM frequencies!')
        disp(' ')
else
%
% Calculate the row symbol number.
%
symno = rintlv*cintlv/freqno;
%
% Display error message if symno and freqno not compatible with rintlv and cintlv and
% stop.  If not compatible, the interleaver function does not work correctly.
%
if freqno >= N;
        disp(' ')
        disp('ERROR: The number of frequency points, N, needs to be increased!')
        disp('N must be larger than:')
        disp(' ')
        disp(freqno)
        disp(' ')
elseif freqno < N;
Nmbr_of_symbols = symno * freqno;
%
% Generate a random message matrix of M-ary symbols, based upon parameter, mary.
% (Uses macro: marymsg.m).
%
B=marymsg(mary,s,symno,freqno);
```

```
Rndm_m_ary_msg=B;
%
% Reconstruct the matrix block of m-ary symbols into an equivalent information block
% using n-ary symbols.  For the case when M=256 (256-ary) and N=16 (16-ary)
% the reshaped matrix will be twice the size of the initial matrix.  Padding of zeros
% may be necessary for certain m-ary and n-ary combinations.  The expansion of the
% original m-ary message block is along the row dimension after conversion to n-ary
% symbols and for the case when (m>n).  (Uses macros: mb.m and bm.m).
%
m1=bm(nary,mb(mary,B));
lengthm1 = length(m1);
nsymno = lengthm1;
remm1 = rem(lengthm1,freqno);
        if remm1 == 0;
        m1 = m1;
        else
        zero = zeros(freqno - remm1);
        m1 =[m1 zero(1,:)];
end
length2m1 = length(m1);
m = (reshape(m1,freqno,length2m1/freqno))';
N_ary_msg=m;
%
% Generate a differentially encoded matrix of complex values with unit
% magnitude and one of (2^n) equal phases. (Uses macro: difcdrft.m).
%
MDD = difcdrft(nary,m,fort);
[MDm MDn] = size(MDD);
MD = MDD;
Cmplx_mod_array = MDD;
%
% Form the frequency array of modulation values that include guard interval.
% (Uses macro: cmv2fa.m)
%
Fa = cmv2fa(N,MD);
Freq_array = Fa;
%
% Generate a constellation plot of complex modulation values.
%
if picy_n == 1
if pic == 1
figure(pic)
plot(MD,'*')
hold on;
plot(0,0,'+')
hold off;
```

```
title(['Transmitted Signal ',int2str(nary),'-ary Constellation Plot'])
axis('square');
orient tall
grid
%
% Plot the frequency array
%
figure(pic+1)
%
% Create x-axis vector
%
xaxis = [0:N-1];
plot(xaxis,abs(Fa),'*')
title(['Frequency Array Plot (number of frequency points are ',int2str(N),')'])
xlabel(['Guard interval length is ',int2str(N-freqno)])
axis('square');
orient tall
grid
end
end
end
end
end
end
end
end
%____________________________________________________________
```

14. Function: cofdmsim.m

```
%____________________________________________________________________________
%
%       Title:          BATCH SIMULATION OF COFDM MODEL
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  5/1/97
%____________________________________________________________________________
%       INPUTS:
%               None
%
%       OUTPUTS:
%               None
%
%       SUBROUTINES USED:
%               chancdl.m, intlvprs.m
%
% COFDMSIM: This batch m-file emulates system models 0 through three using a channel one
% model and channel two model (awgn.m + chuhf.m) with various seed input values to generate
% an error report. This program performs simulations which generate performance SER curves
% which may be compared to the theoretical curves. Different transmission multipaths may be
% selected as well as entering the AWGN noise sigma parameter.
%____________________________________________________________________________
disp('___________________________________________________________________________
________');
disp('This batch m-file runs COFDM simulations using different channel models.')
fort = input('To run the frequency version, enter 1 (one), To run the time version, enter 0 (zero), or
to run both enter 2 (two): ');
freqno = input('Enter the # of OFDM frequencies (Note: Must be even): ');
N = input('Enter the number of FFT points (Note: This number must be larger than # of OFDM
frequencies): ');
chnmdl = input('Do you want to run channel model 0, channel model 1, channel model 2 or channel
model 3? (Enter 0,1,2 or 3): ');
        if chnmdl == 0
disp('Channel model 0 simulation performed.');
sigs = 0;
loss = 0;
dop = 0;
dly = 0;
        elseif chnmdl == 1
disp('Channel model 1 simulation performed.');
```

```
sigs = input('Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003): ');
loss = 0;
dop = 0;
dly = 0;
          elseif chnmdl == 2
disp('Channel model 2 simulation performed.');
sigs = 0;
pthno = input('Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): ');
%
% Link parameters
%
                    if pthno == 3
% Link 3
loss = [0,3,9];
dop = [25,25,25];
dly = [0,.9,5.1];
                    elseif pthno == 2
% Link 2
loss = [0,5,15];
dop = [10,10,10];
dly = [0,.07,.8];
                    elseif pthno == 1
% Link 1
loss = [0,6];
dop = [1,10 0];
dly = [0,.01];
                    elseif pthno == 4
disp('Custom link simulation...')
loss = input('Enter the path loss in dB (Ex. [0 4 7]): ');
dop = input('Enter the doppler frequency in Hertz (Ex. [30 20 15]): ');
dly = input('Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): ');
                    end
          elseif chnmdl == 3
disp('Channel model 3 simulation performed.');
sigs = input('Enter the sigma noise parameter range or single value. (Ex. linspace(0,0.02,20) or .003): ');
pthno = input('Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for custom): ');
%
% Link parameters
%
                    if pthno == 3
% Link 3
loss = [0,3,9];
```

```
dop = [25,25,25];
dly = [0,.9,5.1];
                elseif pthno == 2
% Link 2
loss = [0,5,15];
dop = [10,10,10];
dly = [0,.07,.8];
                elseif pthno == 1
% Link 1
loss = [0,6];
dop = [1,10 0];
dly = [0,.01];
                elseif pthno == 4
disp('Custom link simulation...')
loss = input('Enter the path loss in dB (Ex. [0 4 7]): ');
dop = input('Enter the doppler frequency in Hertz (Ex. [30 20 15]): ');
dly = input('Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): ');
                end
end
allcase = input('Simulate all interleaver cases (yes) or specific ones (no)? (1 = yes, 0 = no): ');
        if allcase == 1
        disp('All cases, (0-8), will be tested.');
        cases = [0:8];
        elseif allcase == 0
        cases = input('Enter specific case numbers from (0 to 8) (Ex. [0 4 5 8]): ');
        end
        if fort ~= 2
if length(cases) ~= 1
casey_n = input('Do you want to find optimal interleaver case(s)? (1 = yes, 0 = no): ');
end
        end
totsym = input('Enter the total minimum number of symbols to simulate (Ex. 10000): ');
rowno = ceil(totsym/freqno);
        if totsym ~= (rowno*freqno)
disp(['Note: Based on the parameters thus far, the actual total number of symbols to be simulated
will be: ',int2str(rowno*freqno)]);
        end
pry_n = input('For the  interleaver, do you want to calculate all possible intermediate matrix
dimension pairs? (1 = yes, 0 = no): ');
pair1 = 1;
pair2 = rowno*freqno;
        if pry_n == 1
%
% Find all multiples of the data matrix based upon the number of rows (symbol #) and
% the number of columns (OFDM frequency number).  From the calculated list of multiples
% calculate all acceptable interleaver pairs   (Uses macro: intlvprs.m)
```

```
        Intrlvr_pairs = intlvprs(rowno,freqno);
        intlvrprs = Intrlvr_pairs;
        disp(' ')
        disp('For these input parameters, all possible acceptable interleaver dimension pairs are:')
        disp(Intrlvr_pairs)
        end
pairs = input(['Desired interleaver pair? (Ex. [row # col #] = [20 50]) (Note: entering
[',int2str(pair1),' ',int2str(pair2),'], or [',int2str(pair2),' ',int2str(pair1),'], offers no interleaving
functionality): ']);
rintlv = pairs(1);
cintlv = pairs(2);
mary = input('Enter the number of M-ary bits, q (i.e. for 256-ary, q = 8): ');
nary = input('Enter the number of N-ary bits, q (i.e. for 16-ary, q = 4): ');
freqspace = round(480000/freqno);
Ng = input('Enter the guard interval length (Number of sample points): ');
ecc = input('Do you want to include error correction coding? (1 = yes, 0 = no): ');
        if  ecc == 1
        code = input('Enter n,k and error correction block length (Ex. [240 200 240]): ');
        n = code(1);
        k = code(2);
        blklgth = code(3);
        elseif  ecc == 0
        n = freqno;
        k =  freqno;
        blklgth = freqno;
        end
svals = input('Enter specific seed values, or 0 for a random seed. (Ex. [ 103 22, 60] or [0]): ');
picy_n = input('Do you want signal plots? (1 = yes, 0 = no): ');
        if picy_n == 1
wait = input('How many seconds of delay between pictures? ');
wait = round(wait);
        elseif picy_n == 0
wait = 0;
        end
prnty_n = input('Do you want print outs? (1 = yes, 0 = no): ');
pic = 0;
svect = [];
for run = 1:length(svals);
        errcase = [];
        errtot = [];
        if min(svals) == 0
rand('seed',sum(100*clock));
s = round(abs(rand(1)*pi*10*(pic+1)*run));
        elseif min(svals) ~= 0
s = svals(run);
        end
```

```
svect = [svect,s];
        for l = 1:length(cases);
disp('_______________________________________________________________________')
disp(['Run #: ',int2str(run)]);
disp(['Seed = ',int2str(s)]);
disp(['Interleaver case = ',int2str(cases(l))]);
%
% If fort equals one, run the frequency simulation version; if fort equals zero, run
% the time version; else if fort equals two, run both versions.  (Uses m-file: chancdl.m)
%
                if fort <= 1
%                    function                    [errmax,errors,freqerrs]                    =
chancdl(chnmdl,wait,prnt,picy_n,pic,case,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,sigs,los
s,dly,dop,freqspace,fort);
[errmax,errors,freqerrs]                                                                        =
chancdl(chnmdl,wait,prnty_n,picy_n,pic,cases(l),s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,
sigs,loss,dly,dop,freqspace,fort);
                elseif fort == 2
disp('Frequency differential encoding/decoding simulation ... ')
disp('_______________________________________________________________________')
[errmax,errors,freqerrs]                                                                        =
chancdl(chnmdl,wait,prnty_n,picy_n,pic,case(l),s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,si
gs,loss,dly,dop,freqspace,1);
disp('*****************************************************************')
disp('Time differential encoding/decoding simulation ... ')
disp('_______________________________________________________________________')
[errmax,errors,freqerrs]                                                                        =
chancdl(chnmdl,wait,prnty_n,picy_n,pic+12,case(l),s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,
Ng,sigs,loss,dly,dop,freqspace,0);
                end
        errtot = [errtot sum(errors)];
        errvect = [errvect,errtot];
        errcase = [errcase sum(errmax)];
        end
        if fort ~= 2
casearry = [cases;errcase];
%       *********************************************************
%       *                       Plots                           *
%       *********************************************************
%
        if casey_n == 1
figure(pic+13)
bar(cases,errcase)
grid
orient tall
        if fort == 1
```

```
title([int2str(pic),': Maximum Error Total Vs. Interleaver Case Number (Freq. Diff. Enc.) (OFDM
Freq. # = ',int2str(freqno),')'])
        elseif fort == 0
title([int2str(pic),': Maximum Error Total Vs. Interleaver Case Number (Time Diff. Enc.) (OFDM
Freq. # = ',int2str(freqno),')'])
        end
xlabel(['CDL Interleaver Case Number'])
ylabel(['Maximum Error Count For Any Symbol Row  (Seed = ',int2str(s),')'])
axis([-.5 8.5 0 (max(errcase)+1)])
        if prnty_n == 1;
        print
        pause(10)
        end
pause(wait);
%
figure(pic+14)
bar(cases,errtot)
grid
orient tall
title([int2str(pic),': Error Totals Vs. Interleaver Case Number'])
xlabel(['CDL Interleaver Case Number'])
ylabel(['Sigma: (',num2str(min(sigs)),'-',num2str(max(sigs)),') Error Total'])
axis([-.5 8.5 (min(errtot)-1) (max(errtot)+1)])
        if prnty_n == 1;
        print
        pause(10)
        end
pause(wait);
        end
pic = pic+1;
        end
end
disp('******************************************************************************')
disp(' ')
disp('Channel model batch run is finished! ')
Seed = svect
%______________________________________________________________________________
```

15. Function: cvdd.m

```
% function  [y] = cvdd(x,alpha)
%=================================================================
% CVDD: This m-file implements the "continuously variable digital delay
%     element" [1].  The input signal 'x' is filtered by an 8tap FIR
%     filter whose tap coefficients are a function of the desired delay,
%     delay = alpha/fsample.  It is implemented as four 8tap FIR filters
%     with fixed coefficients whose outputs are then multipled by alpha
%     (see below).  The particular coefficients used in this program result
%     in a LPF with a passband of 0.0-0.328 norm freq.
%
% INPUTS:
%  x      - Tx1 input data vector to be interpolated
%  alpha  - Tx1 vector of delay values normalized to sample rate (-0.5 < 0.5)
%
% OUTPUTS:
%  y      - Tx1 interpolated and filtered output data vector
%
% ALGORITHM:
%  3rd ordered polynomial Ntap FIR:
%      y(n) = SUM {i=0}^N-1  x(n-i)*C_i
%      where: C_i = alpha^3*C_i,3 + alpha^2*C_i,2 + alpha*C_i,1 + C_i,0
%
%  is implemented as:
%      y0(n) = SUM {i=0}^N-1  x(n-i)*C_i,0
%      y1(n) = SUM {i=0}^N-1  x(n-i)*C_i,1
%      y2(n) = SUM {i=0}^N-1  x(n-i)*C_i,2
%      y3(n) = SUM {i=0}^N-1  x(n-i)*C_i,3
%      y(n) = alpha^3*y3(n) + alpha^2*y2(n) + alpha*y1(n) + y0(n)
%
% NOTE:
%  1. Since linear phase 8tap filters are used, there is an inherent
%     3.5 sample group delay between the output and the input.
%     (therefore y(n)=x(n-(3.5+alpha)));
%  2. mex file allows for any value of alpha, .m file requires |alpha| < -/5
%
% REFERENCE:
%  [1] C.W. Farrow, "A Continuously Variable Digital Delay Element",
%      IEEE International Symposium on Circuits and Systems, pp. 2641-2645,
%      1988.
%
% WRITTEN: R.North/NRaD 1-24-94
% LAST UPDATE:
% USAGE:  y = cvdd(x,alpha);
%=================================================================
function  [y] = cvdd(x,alpha)
if ((nargin ~= 2) | (nargout ~= 1))
  error('ERROR: usage: y = y = cvdd(x,alpha);');
  return;
end
if (size(x) ~= size(alpha))
  error('ERROR: x and alpha must be the same size');
  return;
end
if (abs(alpha) > 0.5)
  error('ERROR: alpha must be within -0.5 and 0.5 ');
  return;
end

%-------------------------------------------------------------------------
% Initialization
%-------------------------------------------------------------------------

% initialize FIR filter coefficients as in [1] (0,0.328 pass band)
C0 = [-0.013824  0.054062 -0.157959  0.616394  0.616394 -0.157959  0.054062 -0.013824];
C1 = [ 0.003143 -0.019287  0.1008   -1.226364  1.226364 -0.1008    0.019287 -0.003143];
C2 = [ 0.055298 -0.216248  0.631836 -0.465576 -0.465576  0.631836 -0.216248  0.055298];
C3 = [-0.012573  0.077148 -0.403198  0.905457 -0.905457  0.403198 -0.077148  0.012573];

%-------------------------------------------------------------------------
% 4 parallel FIRs and add together based on [1]
%-------------------------------------------------------------------------
y0 = filter(C0,[1],x);
y1 = filter(C1,[1],x);
y2 = filter(C2,[1],x);
y3 = filter(C3,[1],x);

y = alpha.*y3;
y = alpha .* (y + y2);
y = alpha .* (y + y1);
y = y + y0;
%-------------------------------------------------------------------------
```

16. Function: decdrcdl.m

```
% function [outmsg,M,MM] =
decdrcdl(picy_n,pic,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%___________________________________________________________________
%
%       Title:          COFDM DECODER WITH CDL DEINTERLEAVING
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/20/97
%___________________________________________________________________
%       INPUTS:
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               K       - Number of OFDM frequencies, equal to the number of columns
%                         in array, M
%               Fa      - Frequency array of complex values
%               nsymno  - Number of n-ary generated symbols
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%                         Reed-Soloman code can perform error detection and correction
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               outmsg  - Matrix of differentially encoded complex values
%                         (unit magnitude)
%               M       - Received complex modulation values
%               MM      - Modulation values after differential decoding
%
%       SUBROUTINES USED:
%               fa2ma.m, dfdcdrft.m, mb.m, bm.m, cdldlv.m
%
%       NOTE: This m-file performs the inverse function of m-file cdrcdlft.m
%
% DECDRCDL: This m-file performs a decoding of the received frequency array of
% complex modulation values to extract the message information. CDL deinterleaving and
% differential decoding is performed. The sink message should be identical to the
% source message (assuming no noise corruption).
%___________________________________________________________________
% USAGE: function [outmsg,M,MM] =
decdrcdl(picy_n,pic,case,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%____________________________________________________________________________
function [outmsg,M,MM] =
decdrcdl(picy_n,pic,case,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%
% Generate the corresponding complex modulation values from the received frequency
% array. (Uses macro: fa2ma.m)
%
M =fa2cma(K,Fa);
Cmplx_mod_vals = M;
%
% Perform differential decoding. (Uses macro: dfdcdrft.m)
%
naryp = nary;
[s,MM] = dfdcdrft(naryp,nary,M,fort);
[L,cc] = size(s);
strans = s';
svect = strans(:)';
corrs = svect(1:nsymno);
%
% Convert from N-ary symbols to the final message format of M-ary symbols
% (Uses macros: mb.m and bm.m)
%
nsymno;
Br = bm(mary,mb(nary,corrs));
lengthBr = length(Br);
rmndr = rem(length(Br),freqno);
        if rmndr == 0;
        Br = Br;
        elseif rmndr ~=0;
        Br = Br(1:(lengthBr-rmndr));
end
rcvd = (reshape(Br,freqno,length(Br)/freqno))';
Rcvd_Intlv_Ary = rcvd;
%
% Performs the CDL deinterleaving function with the same  parameters
% used during the encoding and interleaving process.  (Uses macro: cdldlv.m)
%
[Br Bc] = size(rcvd);
SYNC = [];
sr = rcvd';
si = sr(:)';
sd = cdldlv(rdintlv,cdintlv,case,si,SYNC);
outmsg = reshape(sd,Bc,Br)';
Sink_Msg = outmsg;
%____________________________________________________________________________
```

17. Function: decdrift.m

```
% function [outmsg] = decdrift(picy_n,pic,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%______________________________________________________________________________
%
%       Title:          COFDM DECODER WITHOUT DEINTERLEAVING
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/3/97
%______________________________________________________________________________
%       INPUTS:
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               K       - Number of OFDM frequencies, equal to the number of columns
%                         in array, M
%               Fa      - Frequency array of complex values
%               nsymno  - Number of N-ary generated symbols
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%                         Reed-Soloman code can perform error detection and correction
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               outmsg  - Matrix of differentially encoded complex values  (unit magnitude)
%
%       SUBROUTINES USED:
%               fa2ma.m, dfdcdrft.m, mb.m, bm.m
%
%       NOTE: This m-file performs the inverse function of m-file coderift.m
%
% DECDRIFT: This m-file performs a decoding of the received frequency array of
% complex modulation values to extract the message information.  The sink message
% should be identical to the source message.  Uses either frequency or time
% differential decoding depending on fort.
%______________________________________________________________________________
% USAGE: function [outmsg] =
decdrift(picy_n,pic,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%______________________________________________________________________________
function [outmsg] = decdrift(picy_n,pic,K,Fa,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort)
%
```

```
% Generate the complex modulation values from the received frequency array.
% (Uses macro: fa2ma.m)
%
M =fa2cma(K,Fa);
Cmplx_mod_vals = M;
%
% Plot received signal before differential decoding, only if picy_n is true.
%
if picy_n == 1
figure(pic+5)
plot(M,'*')
hold on;
plot(0,0,'+')
hold off;
title(['Received ',int2str(nary),'-ary Signal Constellation Plot, before Differential Decoding'])
orient tall
axis('square');
grid
end
%
% Perform the time of frequency differential decoding necessary for synchronization.
% (Uses macro: dfdcdrft.m)
%
naryp = nary;
[s,MM] = dfdcdrft(naryp,nary,M,fort);
[L,cc]=size(s);
strans = s';
svect = strans(:)';
corrs = svect(1:nsymno);
%
% Generate a constellation plot of received signal complex modulation values after
% differential decoding, only if picy_n is true.
%
if picy_n == 1
figure(pic+6)
plot(MM,'+')
hold on;
plot(0,0,'+')
hold off;
title(['Received ',int2str(nary),'-ary Signal Constellation Plot, After Differential Decoding'])
orient tall
axis('square');
grid
end
%
% Convert from n-ary symbols to the final message format of m-ary symbols
% (Uses macros: mb.m and bm.m)
%
nsymno;
Br = bm(mary,mb(nary,corrs));
lengthBr = length(Br);
rmndr = rem(length(Br),freqno);
        if rmndr == 0;
        Br = Br;
        elseif rmndr ~=0;
        Br = Br(1:(lengthBr-rmndr));
end
rcvd = (reshape(Br,freqno,length(Br)/freqno))';
M_ary_rcvd = rcvd;
outmsg = rcvd;
%______________________________________________________________
```

18. Function: dfdcdrft.m

```
% function [s,M] = dfdcdrft(qp,q,MD,fort)
%______________________________________________________________________
%
%       Title:          COMPLEX NUMBER DEMODULATOR AND FREQUENCY/TIME
%                       DIFFERENTIAL DECODER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/13/97
%______________________________________________________________________
%       INPUTS:
%               qp    - Base two exponent to generate equal phase sectors (soft decoding)
%               q     - Base two exponent of constellation phase sectors (N = 2^q)
%               MD  - Array of complex modulation values (frequency domain)
%               fort  - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               s - Phase sector number (equivalent decoded N-ary symbols in decimal notation)
%               M - Differentially decoded in time modulation array
%
%       NOTE: This m-file performs the inverse function of m-file difcdrft.m
%
% DFDCDRFT: This m-file differentially demodulates complex modulation values in MD
% into 2^qp equal phase sectors from constellations of 2^q phase sectors.  If fort
% equals zero, the output is [s M] where s is the phase sector number and M is the
% time differentially decoded modulation values.  If fort equals one, The output
% is [s M] where s is the phase sector number and M is the frequency differentially
% decoded modulation values.
%______________________________________________________________________
% USAGE: function [s,M] = dfdcdrft(qp,q,MD,fort)
%______________________________________________________________________
function [s,M] = dfdcdrft(qp,q,MD,fort)
        if fort == 0 % Time differential decoding
%
% Transpose the modulation array, and find the dimensions.
%
MD=MD';
[m n]=size(MD);
%
% Perform a looping routine to find the phase differences between adjacent values in
% the array, MD, and put these calculated values into array, M.
%
```

```
for l=1:m
for j=1:n-1
M(l,j)=MD(l,j+1)*conj(MD(l,j));
end
end
%
% Transpose the array back to its original form.
%
M=M';
%
% Calculate the number of M-ary symbols based upon the exponent qp, then use this
% number to find the number of equally spaced phases in a unit circle.
%
N=2^qp;
dph=2*pi/N;
%
% Divide the phase arguments of elements in M, by the equal phases generated by dph.
%
phn=angle(M)./dph;
%
% Calculate the phase sector number by finding the remainders.
%
s=rem(round(phn)+N,N);
        elseif fort == 1 % Frequency differential decoding
%
% Transpose the modulation array, and find the dimensions.
%
[m,n]=size(MD);
MD=MD(:,2:n);
[m n]=size(MD);
%
% Perform a looping routine to find the phase differences between adjacent values in
% the array, MD, and put these calculated values into array, M.
%
for l=1:m
for j=1:n-1
M(l,j)=MD(l,j+1)*conj(MD(l,j));
end
end
%
% Transpose the array back to its original form.
%
%M=M';
%
% Calculate the number of m-ary symbols based upon the exponent qp, then use this
% number to find the number of equally spaced phases in a unit circle.
```

```
%
N=2^qp;
dph=2*pi/N;
%
% Divide the phase arguments of elements in M, by the equal phases generated by dph.
%
phn=angle(M)./dph;
%
% Calculate the phase sector number by finding the remainders.
%
s=rem(round(phn)+N,N);
        end
%_________________________________________________________________
```

19. Function: difcdrft.m

```
% function [MD] = difcdrft(q,m,fort)
%____________________________________________________________________________
%
%       Title:          COMPLEX NUMBER MODULATOR AND FREQUENCY/TIME
%                       DIFFERENTIAL ENCODER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/10/97
%____________________________________________________________________________
%       INPUTS:
%               q     - Base two exponent of constellation phase sectors (N = 2^q)
%               m     - Matrix of M-ary symbols to be transformed to complex numbers and
%                       differentially encoded
%               fort  - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               MD - Matrix of differentially encoded complex values (unit magnitude)
%
%       NOTE: This m-file performs the inverse function of m-file dfdcdrft.m
%
% DIFCDRFT: This m-file creates complex values MD with amplitude one and one of
% 2^q possible equal phase values. If fort is zero, The first row is one (zero phase)
% and represents the synchronization reference. The remaining rows are time
% differentially coded in phase. If fort is one, the first two columns are ones and
% represent the synchronization reference (two reference columns are used to maintain
% an even number of total columns). The remaining columns are frequency differentially
% coded in phase. For both frequency and time cases, the differential encoding is
% executed by performing a cumulative summation either down columns or across rows.
% The input symbols are provided by the m-ary input matrix, m.
%____________________________________________________________________________
% USAGE: function [MD] = difcdrft(q,m,fort)
%____________________________________________________________________________
function MD = difcdrft(q,m,fort)
        if fort == 0 %Time differential encoding
%
% M-ary alphabet size.
%
N=2^q;
%
% Determine the number of equal phases based upon the m-ary symbol length.
%
```

```
dph=2*pi/N;
%
% Find the size of the input symbol matrix (# of rows and # of columns).
%
[rr n]=size(m);
%
% Perform the time differential encoding of phase values by cumulative summing matrix,
% m, down one column at a time across the entire matrix.  This function generates a
% matrix.
%
for k=1:n
md=cumsum(m(:,k));
%
% Generate the complex numbers with corresponding phase values.
%
MD(:,k) = exp(i*dph.*md);
end
%
% Inject the reference row of ones (zero phase) at top of output matrix for
% differential encoding synchronization.
%
MD=[ones(1,n); MD];
        elseif fort == 1 %Frequency differential encoding
%
% M-ary alphabet size.
%
N=2^q;
%
% Determine the number of equal phases based upon the m-ary symbol length.
%
dph=2*pi/N;
%
% Find the size of the input symbol matrix (# of rows and # of columns).
%
[rr n]=size(m);
%
% Perform the frequency differential encoding of phase values by cumulative summing
% matrix, m, across one row at a time down the entire matrix.  This function generates
% a matrix.
%
md=cumsum(m');
md=md';
%
% Generate the complex numbers with corresponding phase values.
%
MD = exp(i*dph.*md);
```

```
%
% Inject the reference row of ones (zero phase) at top of output matrix for
% differential encoding synchronization.
%
MD=[ones(rr,2) MD];
        end
%________________________________________________________________
```

20. Function: diffchkr.m

```
% function diffchkr(s,symno,freqno,mary,nary)
%_________________________________________________________________________
%
%       Title:          DIFFERENTIAL ENCODER/DECODER CHECKER
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  5/2/97
%_________________________________________________________________________
%       INPUTS:
%               s       - Seed parameter for random number generator
%               symno   - Number of symbol rows in message block
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%
%       OUTPUTS:
%               None
%
% DIFFCHKR:  This m-file verifys correct functionality of the differential encoder/decoder.
%_________________________________________________________________________
% USAGE: function diffchkr(s,symno,freqno,mary,nary)
%_________________________________________________________________________
function diffchkr(s,symno,freqno,mary,nary)
fort = input('To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time
version: ');
%
% Generate random m-ary message array.
%
B=marymsg(mary,s,symno,freqno);
Rndm_m_ary_msg = B;
%
% Reconstruct the matrix block of m-ary symbols into an equivalent
% information block using n-ary symbols.  For the case when m=8 (8-ary) and n=4 (4-ary)
% the reshaped matrix will be twice the size of the initial matrix.
% (Uses macros: mb.m and bm.m).
%
m1=bm(nary,mb(mary,B));
lengthm1 = length(m1);
m=(reshape(m1,lengthm1/symno,symno))';
N_ary_msg=m;
%
```

```
% Perform differential encoding on messsage array.
%
%
% If fort equals one, run the frequency simulation version; otherwise, run the time version.
%
        if fort == 1
disp(' ')
disp('Frequency Differential Encode/Decode version')
%
% Freq. Diff. Enc.
%
MDD = difcdrf(mary,m);
        elseif fort ~= 1
disp(' ')
disp('Time Differential Encode/Decode version')
%
% Time. Diff. Enc.
%
MDD = difcdrt(mary,m);
        end
%
% Perform differential decoding.
%
maryq = mary;
        if fort == 1
%
% Freq. Diff. Enc.
%
[s M] = difdcdrf(maryq,mary,MDD);
        elseif fort ~= 1
%
% Time. Diff. Enc.
%
[s M] = difdcdrt(maryq,mary,MDD);
        end
%
% Check results for correctness. (Uses m-file check.m).
%
[error_no,freqerrs,errmx,rowerrs] = check(0,m,s,freqno,freqno,freqno);
%______________________________________________________________________
```

21. Function: dline.m

```
% function xd=dline(x,d)
%_______________________________________________________________________
%
%       Title:          UHF CHANNEL DELAY LINE GENERATOR
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/19/97
%_______________________________________________________________________
%       INPUTS:
%               x  - input time domain array representing the transmitted signal
%               d  - is a vector of delays
%
%       OUTPUTS:
%               xd - output time domain signal array with delays
%
%       SUBROUTINES USED:
%               cvdd.m
%
% DLINE:  This m-file is used by the the uhf channel model (chuhf.m) to add the multipath delays
% to the direct signal.
%_______________________________________________________________________
% USAGE: function xd = dline(x,d)
%_______________________________________________________________________
function xd = dline(x,d)
x = x.';
dmax = max(d);
dmin = min(d);
nmin = floor(dmin);
nmax = ceil(dmax);
x = [x;zeros(nmax+3,1)];
N = length(x);
Nd = length(d);

for n = 1:Nd;
        di=d(n);
        D=floor(di);
        deld=di-D;
        xd(:,n)=cvdd(x,deld-.5);
        xd(:,n)=[zeros(D,1);xd(1:N-D,n)];
end
xd=xd.';
```

```
[rr,cc]=size(xd);
xd=xd(:,4+nmin:cc);
%______________________________________________________________
```

22. Function: fa2cma.m

```
% function [Mm] = fa2cma(K,X)
%_____________________________________________________________________________
%
%       Title:          FREQUENCY ARRAY TO COMPLEX MODULATION ARRAY
                        CONVERTER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/21/97
%_____________________________________________________________________________
%       INPUTS:
%               K - Number of OFDM frequencies equal to the number of columns in array, M.
%               X - Frequency array of complex values
%
%       OUTPUTS:
%               Mm - Complex modulation value array
%
%       NOTE: This m-file performs the inverse function of m-file cmv2fa.m
%
% FA2CMA: This m-file accepts complex values in the frequency array X and places them into
% complex modulation array M. M is complex of length 2K+1.
%_____________________________________________________________________________
% USAGE: function [Mm] = fa2cma(K,X)
%_____________________________________________________________________________
function Mm =fa2cma(K,X)
%
% Determine the dimensions of the input matrix
%
[m n] = size(X);
%
% Flip the array values around and remove the zero pad.
%
Mm(:,1:K) = X(:,n-K+1:n);
Mm(:,K+1:2*K) = X(:,1:K);
Cmplx_mod_vals = Mm;
%_____________________________________________________________________________
```

23. Function: intlvchk.m

```
% function intlvchk(s,symno,freqno,rintlv,cintlv,mary,case)
%______________________________________________________________________
%
%       Title:          INTERLEAVER/DEINTERLEAVER VERIFIER
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  5/2/97
%______________________________________________________________________
%       INPUTS:
%               s       - Seed parameter for random number generator
%               symno   - Number of symbol rows in message block
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               mary    - Initial M-ary symbol format (M = 2^q)
%               case    - Variable indicating the deinterleaving method to be used (9 different
%                         cases)
%
%       OUTPUTS:
%               None
%
% INTLVCHK:  This m-file verifys correct functionality of the CDL interleaver and deinterleaver.
%______________________________________________________________________
% USAGE: function intlvchk(s,symno,freqno,rintlv,cintlv,mary,case)
%______________________________________________________________________
function intlvchk(s,symno,freqno,rintlv,cintlv,mary,case)
%
% Find all multiples of the data matrix based upon the number of rows (symbol #) and
% the number of columns. (OFDM frequency #).
% (Uses macro: mltpl.m)
%
multiples = mltpl(symno,freqno);
Intrlvr_nbr_mltpls = multiples;
%
% Display error message if symno and freqno/2 not compatible with rintlv and cintlv and stop.
% If not compatible, the interleaver function does not work correctly.
%
if (symno*freqno) ~= (rintlv*cintlv)
        disp('ERROR: The interleaver parameters, rintlv and cintlv, are not compatible with the
message array size.')
        disp('          The acceptable choice of possible numbers are: ')
        disp(' ')
        disp(multiples)
        disp('Note: The selected pair of numbers must be divisible by the number of rows and
columns of the input matrix multiplied together.')
        disp('        In this case the number of rows times the number of columns is:')
        disp(' ')
        disp(symno*freqno)
elseif (symno*freqno)/(rintlv*cintlv) == 1
%
% Generate a random message matrix of m-ary symbols.  (Uses macro: marymsg.m).
%
B=marymsg(mary,s,symno,freqno);
Random_msg=B
%
% Perform a CDL block interleaving function on the matrix, B, with rintlv rows
% and cintlv columns.  (Uses macro cdlilv.m).
%
SYNC = [];
[Br Bc] = size(B);
Bt=B';
Bvect = Bt(:)';
si = cdlilv(rintlv,cintlv,case,Bvect,SYNC);
Bi = reshape(si,Bc,Br)';
Interleaved_array = Bi
%
% Performs the CDL deinterleaving function with the same  parameters
% used during the encoding and interleaving process.  (Uses macro: cdldlv.m)
%
[Br Bc] = size(Bi);
SYNC = [];
sr = Bi';
si = sr(:)';
sd = cdldlv(rintlv,cintlv,case,si,SYNC);
Bd = reshape(sd,Bc,Br)';
Deinterleaved_array = Bd
%
% Check the results of the initial message matrix with the interleaved and deinterleaved one
% for errors.  (Uses macro check.m).
%
% function [error_no,freqerrs,errmx,rowerrs] = check(pic,x,y,n,k,blklgth);
[error_no,freqerrs,errmx,rowerrs] = check(0,B,Bd,freqno,freqno,freqno);
end
%________________________________________________________________________
```

24. Function: intlvprs.m

```
% function pairs = intlvprs(n,m)
%_____________________________________________________________________
%
%       Title:          INTERMEDIATE MATRIX INTERLEAVER DIMENSION PAIRS
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Editor:         Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/23/96
%_____________________________________________________________________
%       INPUTS:
%
%               n  - Number of rows in input matrix
%               m - Number of columns in input matrix
%
%       OUTPUTS:
%               pairs - Output matrix of permissable interleaver matrix dimension pairs
%
% INTLVPRS: This m-file initially finds all positive whole number multiples of a (nxm)
% matrix (ie: multiples of (n*m), # of columns times # of rows). Based upon this
% result, the formation of permissable intermediate matrix dimension pairs are
% calculated. The output is a matrix of suitable intermediate matrix dimensions
% agreeable with the original message matrix dimension.
%_____________________________________________________________________
% USAGE: function pairs = intlvprs(n,m)
%_____________________________________________________________________
function pairs = intlvprs(n,m)
%
% Find the product of the # of rows and # of columns of the input matrix
%
prod = n*m;
%
% Initialize a vector with value one, since all numbers are divisible by one
%
multvect=[1];
%
% Looping sequence to find all whole number multiples of max. If remaider is zero,
% that divisor is appended to multvect indication a legitament whole number divisor.
% If the remainder is other than zero, the index is inreased by one and multvect is
% not changed. The looping process ends at max, since that is the largest number that
% can wholly divide into itself.
%
for i = 2:prod;
```

```
remdr = rem(prod,i);
if remdr == 0
        multvect = [multvect i];
        else
        multvect = multvect;
end
%
% Output matrix of pair results
%
mult = multvect;
end
lngth = length(mult);
nbr = mult(lngth);
result = [1 nbr];
for i = 2:lngth;
crntpr = [mult(i) nbr/mult(i)];
result = [result;crntpr];
end
pairs = result;
%_______________________________________________________________________
```

25. Function: itda.m

```
% function Y = itda(Ng,y)
%______________________________________________________________________
%
%      Title:          FREQUENCY DOMAIN SAMPLES WITHOUT GUARD INTERVAL
%      Author:         Dr. Paul H. Moose
%                      Naval Postgraduate School
%      Revised by:     Dave Roderick
%                      Naval Postgraduate School
%
%      Last revision:  1/27/97
%______________________________________________________________________
%      INPUTS:
%              Ng - Number of samples for the periodic precursor guard interval to remove
%              y  - Array of received time domain values
%
%      OUTPUTS:
%              Y - Frequency domain array of sample values after FFT is performed
%
%      NOTE: This m-file performs the inverse function of m-file tda.m
%
% ITDA: This M-file removes the Ng point precursor guard interval and takes FFT.
%______________________________________________________________________
% USAGE: function [Y] = itda(Ng,y)
%______________________________________________________________________
function Y = itda(Ng,y)
%
% Find the dimensions of input array, y.
%
[L Nt] = size(y);
%
% Remove the guard interval for channel compensation, Ng, precursor.
%
y = y(:,Ng+1:Nt);
%
% Take the FFT of array, y.
%
Y = fft(y.').';
%______________________________________________________________________
```

26. Function: marymsg.m

```
% function [vmary] = marymsg(q,s,n,m)
%______________________________________________________________________
%
%       Title:          M-ARY MESSAGE TEST PATERN GENERATOR
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/7/96
%______________________________________________________________________
%       INPUTS:
%               q - Base two exponent used for M-ary conversion of binary message (M=2^q)
%               s - Seed parameter for random number generation
%               n - Desired number of rows in random output message matrix
%               m - Desired number of columns in random output message matrix
%
%       OUTPUTS:
%               vmary - M-ary message symbol array
%
%       SUBROUTINES USED:
%               bm.m,msg.m
%
% MARYMSG:  This m-file generates a random test message matrix of M-ary symbol values.
% The number of output symbols generated for the random message is determined by n
% (rows) and m (columns).  The randomness aspect is determined by s, the seed parameter.
%______________________________________________________________________
% USAGE: function [vmary] = marymsg(q,s,n,m)
%______________________________________________________________________
function vmary = marymsg(q,s,n,m)
%
% Calls macros msg.m to generate a random binary test pattern and bm.m to convert the
% random test pattern to M-ary symbols.
%
vmary = (reshape(bm(q,msg(s,(q*n*m))),m,n))';
Random_msg = vmary;
%______________________________________________________________________
```

27. Function: mb.m

```
% function [b] = mb(q,m)
%________________________________________________________________________
%
%       Title:          M-ARY TO BINARY CONVERTER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  12/18/96
%________________________________________________________________________
%       INPUTS:
%               q  - Base two exponent representing M-ary symbol
%               m - M-ary data matrix or vector
%
%       OUTPUTS:
%               b - binary output vector (LSB for each symbol on left)
%
%       NOTE: This m-file performs the inverse function of m-file bm.m
%
% MB: This m-file implements a M-ary to binary converter by accepting two input
% arguments, q and m, and returning the output, b. The input argument, m, is a
% M-ary (2^q) matrix or vector. The input argument, q, represents
% the base two exponent representing the M-ary symbol and used during the binary
% conversion. The output b, is the binary vector mapping of the m-ary symbol.
% end of v if necessary to ensure an even multiple of q with no remainder (even
% modulo q) during the generation of the final M-ary symbol. The bits, v, are stripped
% q at a time and are mapped to a symbol vector m with integer values 0 to 2^q-1.
% The least significant bit is taken to be on the left.
%________________________________________________________________________
% USAGE: function [b] = mb(q,m)
%________________________________________________________________________
function [b] = mb(q,m)
%
% Find the number of rows and columns in input matrix, m.
%
row = size(m,1);
col = size(m,2);
%
% Reshape the transposed input M-ary matrix, m, into a vector, m.
%
m = reshape(m',1,(row*col));
%
% Calculate remainders by dividing m vector elements by 2.
```

```
%
b0=rem(m,2);
m=(m-b0)./2;
B=b0;
%
% Looping algorithum for finding remainders for each m element.
%
for j=1:q-1
bj= rem(m,2);
m=(m-bj)./2;
%
% Generate a column vector of remainders for each symbol with
% least significant bit in first row.
%
B=[B;bj];
end
%
% Transpose column vector with Least significant bit on the left.
%
b=B(:)';
binary=b;
%_______________________________________________________________________________
```

28. Function: mltpl.m

```
% function [mult] = mltpl(n,m)
%_________________________________________________________________________
%
%       Title:          COMMON MULTIPLES
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Editor:         Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/13/96
%_________________________________________________________________________
%       INPUTS:
%               n  - Number of rows in input matrix
%               m - Number of columns in input matrix
%
%       OUTPUTS:
%               mult - Output vector of whole number multiples of product n x m
%
%
% MLTPL: This m-file finds all positive whole number multiples of a (nxm) matrix.
% (ie: multiples of (n*m), # of columns times # of rows).  Useful for the interleaving
% function in order to acertain how many different row and column shifts are permitted
% and what values are allowed.
%_________________________________________________________________________
% USAGE: function [mult] = mltpl(n,m)
%_________________________________________________________________________
function [mult] = mltpl(n,m)
%
% Find the multiple of the # of rows and # of columns of the input matrix
%
max = n*m;
%
% Initialize a vector with value one, since all number are divisible by one
%
multvect=[1];
%
% Looping sequence to find all whole number multiples of max.  If remaider is zero,
% that divisor is appended to multvect indication a legitament whole number divisor.
% If the remainder is other than zero, the index is inreased by one and multvect is
% not changed.  The looping process ends at max, since that is the largest number that
% can wholly divide into itself.
%
for i = 2:max;
remdr = rem(max,i);
```

```
if remdr == 0
        multvect = [multvect i];
        else
        multvect = multvect;
end
%
% Output vector results
%
mult = multvect;
end
%___________________________________________________________________________
```

29. Function: msg.m

```
% function u = msg(s,k)
%______________________________________________________________________________
%
%       Title:          MESSAGE TEST PATERN GENERATOR
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  12/23/96
%______________________________________________________________________________
%       INPUTS:
%               s - Used during the random number generation and represents the seed
%               k - Number of bits in the formation of the output sequence
%
%       OUTPUTS:
%               u - Random binary message pattern
%
% MSG:  This m-file generates a random test message sequence of binary bits.  The
% number of output bits generated for the message as well as the randomness aspect is
% determined by, k, and, s, respectively.
%______________________________________________________________________________
% USAGE: function u = msg(s,k)
%______________________________________________________________________________
function u = msg(s,k)
%
% Set the random variable to a uniform distribution.
%
rand('uniform');
%
% Returns the current seed of the uniform generator.
%
temp = rand('seed');
%
% sets the uniform generator s to 'seed'.
%
rand('seed',s);
%
% Generate a random number vector of length, k, and round up or down to make binary.
%
u = round(rand(1,k));
%rand('seed',temp)
Binary_sequence = u;
%__________________________________________________________________________
```

30. Function: ofst.m

```
% function xo = ofst(e,N,x)
%________________________________________________________________________
%
%       Title:          CHANNEL OFFSET
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/18/97
%________________________________________________________________________
%       INPUTS:
%               e  - Frequency offset error amount
%               N  - Number of FFT frequency sample points
%               x  - input time domain values
%
%       OUTPUTS:
%               xo - Direct path offset output
%
% OFST: This m-file applies a frequency offset of e to the direct path of vector x (0.7 of max
% Doppler shift). The phase is 2*pi*e/N.
%________________________________________________________________________
% USAGE: function xo = ofst(e,N,x)
%________________________________________________________________________
function xo = ofst(e,N,x)
[m Nt]=size(x);
xo=x.';
x=x.';
x=x(:);
x=x.';
Nt=length(x);
l=1:Nt;
%
% Now create the offset frequency
%
ex=x.*exp(i*(2*pi/N)*e.*l);
xo(:)=x;
xo=xo.';
%________________________________________________________________________
```

31. Function: ray_dop.m

```
% function c = ray_dop(s,M,N,es)
%________________________________________________________________
%
%       Title:          RAYLEIGH DOPPLER
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School, Oct 10, 1996
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  2/9/97
%________________________________________________________________
%       INPUTS:
%               s  - Seed parameter for random number generator
%               M  - No. of points
%               N  - No. of sample points per symbol
%               es - maximum doppler shift frequency as a fraction of tone spacing.
%
%       OUTPUTS:
%               c - complex random sequence with iid guassian real and imag parts of length M
%                   points.
%
% RAY_DOP: This m-file generates a sequence of length L*N points
% (L bauds of N samples per baud) of complex numbers with zero
% mean, 0.5 variance real and imaginary parts. Envelope is Rayleigh,
% with mean square value of one. The power spectral density of the
% sequence is Sc(f)=1/(2*pi((es*delf)^2 - f^2)^.5) for abs(f)<es*delf
% and zero elsewhere. es*delf is maximum doppler fdmax and delf is
% OFDM tone spacing [1].
%
% REFS:
%       [1] Pommier and Wu; "Interleaving or spectrum spreading in
%       digital radio intended for vehicles," EBU Review - Technical No.
%       217, (June 1986).
%________________________________________________________________
% USAGE: function [c] = ray_dop(s,M,N,es)
%________________________________________________________________
function c = ray_dop(s,M,N,es)
m=0:M-1;
randn('seed',s+10);
prl = randn(1,20);
randn('seed',s+20);
pim = i*randn(1,20);
p = prl + pim;
p=p/(40^.5);
```

```
rand('seed',s+30);
e=rand(1,20);
e=es*cos(2*pi*(e-.5));
E=exp(i*2*pi*e'*m/N);
c=p*E;
%_______________________________________________________________
```

32. Function: rotm.m

```
% function [vp,vn] = rotm(v,m)
%______________________________________________________________________
%
%      Title:          ROTATE VECTOR
%      Author:         Dr. Paul H. Moose
%                      Naval Postgraduate School
%      Revised by      Dave Roderick
%                      Naval Postgraduate School
%
%      Last revision:  3/24/97
%______________________________________________________________________
%      INPUTS:
%              v  - Input matrix to be rotated
%              m  - Number of positions to be shifted
%
%      OUTPUTS:
%              vp - Matrix with positive rotation
%              vn - Matrix with negative rotation
%
% ROTM:  This m-file is used to rotate (cyclically shift) a vector v by m positions.
% vp is v rotated positively (shifted to the right or down), and vn is v rotated
% negatively (shifted to the left or up).
%______________________________________________________________________
% USAGE: function [vp,vn] = rotm(v,m)
%______________________________________________________________________
function [vp,vn] = rotm(v,m)
L=length(v);
m=rem(m,L);
ii=(1:L)-1;
isp=rem(ii-m+L,L)+1;
isn=rem(ii+m+L,L)+1;
vp=v(isp);
vn=v(isn);
%______________________________________________________________________
```

33. Function: tda.m

```
% function x = tda(Ng,X)
%____________________________________________________________________________
%
%       Title:          TIME DOMAIN SAMPLES WITH GUARD INTERVAL
                        PRECURSOR
%       Author:         Dr. Paul H. Moose
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  1/27/97
%____________________________________________________________________________
%       INPUTS:
%               Ng   - Number of samples for the addition of a periodic precursor guard interval
%               X    - Array of frequency values taken from the frequency array
%
%       OUTPUTS:
%               x    - Time domain array of sample values after IFFT is performed
%
%       NOTE: This m-file performs the inverse function of m-file itda.m
%
% TDA:  This M-file takes the inverse fft of array X and adds a periodic precursor guard interval
% of Ng samples for channel compensation. The result is the mfm xmit signal.
%____________________________________________________________________________
% USAGE: function [x] = tda(Ng,X)
%____________________________________________________________________________
function x = tda(Ng,X)
%
% Find the dimensions of the input frequency array
[m N] =size(X);
%
% Perform inverse FFT on frequency values in array, X.
%
x =ifft(X.');
% Add precursor of Ng samples to the beginning of the time domain array for channel
% compensation.
%
x=x.';
if Ng==0
x=x;
else
x =[x(:,N-Ng+1:N) x];
end
%____________________________________________________________________________
```

34. Function: uhfift.m

```
% function [errors,freqerrs] =
uhfift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,loss,dly,dop,freqspace,fort)
%_______________________________________________________________________________
%
%       Title:          CHANNEL TWO SIMULATION W/O INTERLEAVING
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/5/97
%_______________________________________________________________________________
%       INPUTS:
%               picy_n  - Switch variable to allow or disallow the generation of figures
%               pic     - Argument passed by a calling m-file to indicate the loop number
%               s       - Seed parameter for random number generator
%               freqno  - Number of OFDM frequencies (sub-carriers) used in each message array
%               rintlv  - Interleaver parameter for intermediate matrix row #
%               cintlv  - Interleaver parameter for intermediate matrix column #
%               N       - Number of FFT frequency sample points, must be larger than freqno
%               mary    - Initial M-ary symbol format (M = 2^q)
%               nary    - Final N-ary symbol format (N = 2^p)
%               n       - Integer number representing code word symbol length
%               k       - Integer number representing information word symbol length
%                         (Typically: n>k)
%               blklgth - Block number indicating number of symbols over which the
%                         Reed-Soloman code can perform error detection and correction
%               Ng      - Number of time domain samples for the addition of guard interval
%               loss    - Multipath free space loss in dB (vectors accepted)
%               dly     - Multipath delay in microseconds (vectors accepted)
%               dop     - Doppler frequency in Hertz (vectors accepted)
%               freqspace - Frequency spacing between individual OFDM carriers in Hz
%               fort    - Selects either frequency (fort = 1) or time (fort = 0) differential encoding
%
%       OUTPUTS:
%               errors   - Number of sink message errors found if any
%               freqerrs - Number of sink message errors vs. OFDM freq. #
%
%       SUBROUTINES USED:
%               coderift.m, tda.m, chuhf.m, itda.m, decdrift.m, check.m
%
% UHFIFT:  This m-file performs an OFDM simulation using a channel two model (chuhf.m)
% to observe the multipath fading effects. No interleaving is performed; however, time
% or frequency differential encoding is included depending on fort.  A check is performed
% comparing the source message with the sink message to determine where any errors
% occurred as a result of channel noise corruption.  Error plots show the error
% locations within the message array.
%______________________________________________________________________________
% USAGE: function [errors,freqerrs] =
uhfift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,loss,dly,dop,freqspace,fort)
%______________________________________________________________________________
function [errors,freqerrs] =
uhfift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,loss,dly,dop,freqspace,fort)
%
% Randomly generate a block message and encode as a OFDM frequency array.  Perform
% freq. differential encoding with no interleaving.  (Uses macro: coderift.m)
%
[xmt,modvals,B,nsymno] = coderift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,fort);
%
% Put into the time domain by performing the IFFT and add periodic precursor.
% (Uses macro: tda.m)
%
xmtifft = tda(Ng,xmt);
xmtpts = 1:length(xmtifft);
%
% Plot the time-domain transmitted signal if picy_n is true
%
if picy_n == 1
xmtpts = 1:length(xmtifft);
figure(3)
plot(xmtpts,xmtifft)
title('Transmitted Time Domain Signal')
axis('square');
orient tall
grid
end
%
% Transmit the message signal through UHF channel (multipath maritime channel).
% The output represents signal plus noise.  (Uses macro: chuhf.m)
%
sandn = chuhf(s+1,xmtifft,loss,dly,dop,N,freqspace);
%
% Plot the time-domain received signal if picy_n is true
%
if picy_n == 1
rcvdpts = 1:length(sandn);
figure(4)
plot(rcvdpts,sandn)
title('Received Time Domain Signal')
axis('square');
```

```
orient tall
grid
end
%
% Remove precursor and take FFT to put back into frequency domain.
% (Uses macro: itda.m)
%
sandnfft = itda(Ng,sandn);
%
% Decode the received message signal and generate the corresponding sink message.
% (Uses macro:  decdrift.m)
%
K = (length(modvals(1,:)))/2;
rdintlv = rintlv;
cdintlv = cintlv;
rcvd = decdrift(picy_n,pic,K,sandnfft,nsymno,freqno,rdintlv,cdintlv,mary,nary,fort);
Transmitted_msg = B;
Received_msg = rcvd;
%
% Check the source message against the sink message for errors.  Returns error report.
% (Uses macro:  check.m)
%

[errors,freqerrs,errmx,rowerrs] = check(pic,B,rcvd,n,k,blklgth);
errmx;
[rm cm] = size(errmx);
errsum = sum(errors);
if errsum ~= 0
symno = rintlv*cintlv/freqno;
freqno;
if picy_n == 1
        if dop == [25,25,25]
figure(2)
mesh(errmx)
title(['Link 3: Error Distribution Without Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),')'])
axis([0 freqno 0 symno 0 max(max(errmx))])
xlabel(['Freq. # (Total = ',int2str(freqno),')'])
ylabel(['Row # (Symbol # = ',int2str(symno*freqno),')'])
zlabel(['Error Occurance (Total = ',int2str(errsum),') (seed = ',num2str(s),')'])
text(-150,0,1.95,['Error Correction = ',int2str(floor((n-k)/2))])
grid
orient tall
%for i = 1:36
%view(i*10,30)
%pause(1)
```

```
%end
%print
%pause(10)
        elseif dop == [10,10,10]
figure(2)
mesh(errmx)
title(['Link 2: Error Distribution Without Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),')'])
axis([0 freqno 0 symno 0 max(max(errmx))])
xlabel(['Freq. # (Total = ',int2str(freqno),')'])
ylabel(['Row # (Symbol # = ',int2str(symno*freqno),')'])
zlabel(['Error Occurance (Total = ',int2str(errsum),') (seed = ',num2str(s),')'])
text(-150,0,1.95,['Error Correction = ',int2str(floor((n-k)/2))])
grid
orient tall
%for i = 1:36
%view(i*10,30)
%pause(1)
%end
%print
%pause(10)
        elseif dop == [1,10,0]
figure(2)
mesh(errmx)
title(['Link 1: Error Distribution Without Interleaving (M-ary bits: ',int2str(mary),',','N-ary bits:
',int2str(nary),')'])
axis([0 freqno 0 symno 0 max(max(errmx))])
xlabel(['Freq. # (Total = ',int2str(freqno),')'])
ylabel(['Row # (Symbol # = ',int2str(symno*freqno),')'])
zlabel(['Error Occurance (Total = ',int2str(errsum),') (seed = ',num2str(s),')'])
text(-150,0,1.95,['Error Correction = ',int2str(floor((n-k)/2))])
grid
orient tall
%for i = 1:36
%view(i*10,30)
%pause(1)
%end
%print
%pause(10)
end
end
else
disp(' ')
disp('GREAT!!! Test passed.')
end
if sum(rowerrs) ~= 0
```

```
figure(3)
cony = (max(rowerrs)+5)/60;
conx = symno/80;
errindx = 1:length(rowerrs);
bar(errindx,rowerrs)
title(['Error Count Per Symbol Row (Total Errors = ',int2str(sum(rowerrs)),')'])
xlabel('Row Number')
ylabel('Number of Errors')
axis([0.5,(symno+.5),0,(max(rowerrs)+(6*cony))])
if sum(rowerrs) ~= 0
for i = 1:length(rowerrs)
text(i-(1.5*conx),rowerrs(i)+(4*cony),int2str(rowerrs(i)))
end
end
orient tall
%print
%pause(10)
end
%______________________________________________________________________________
```

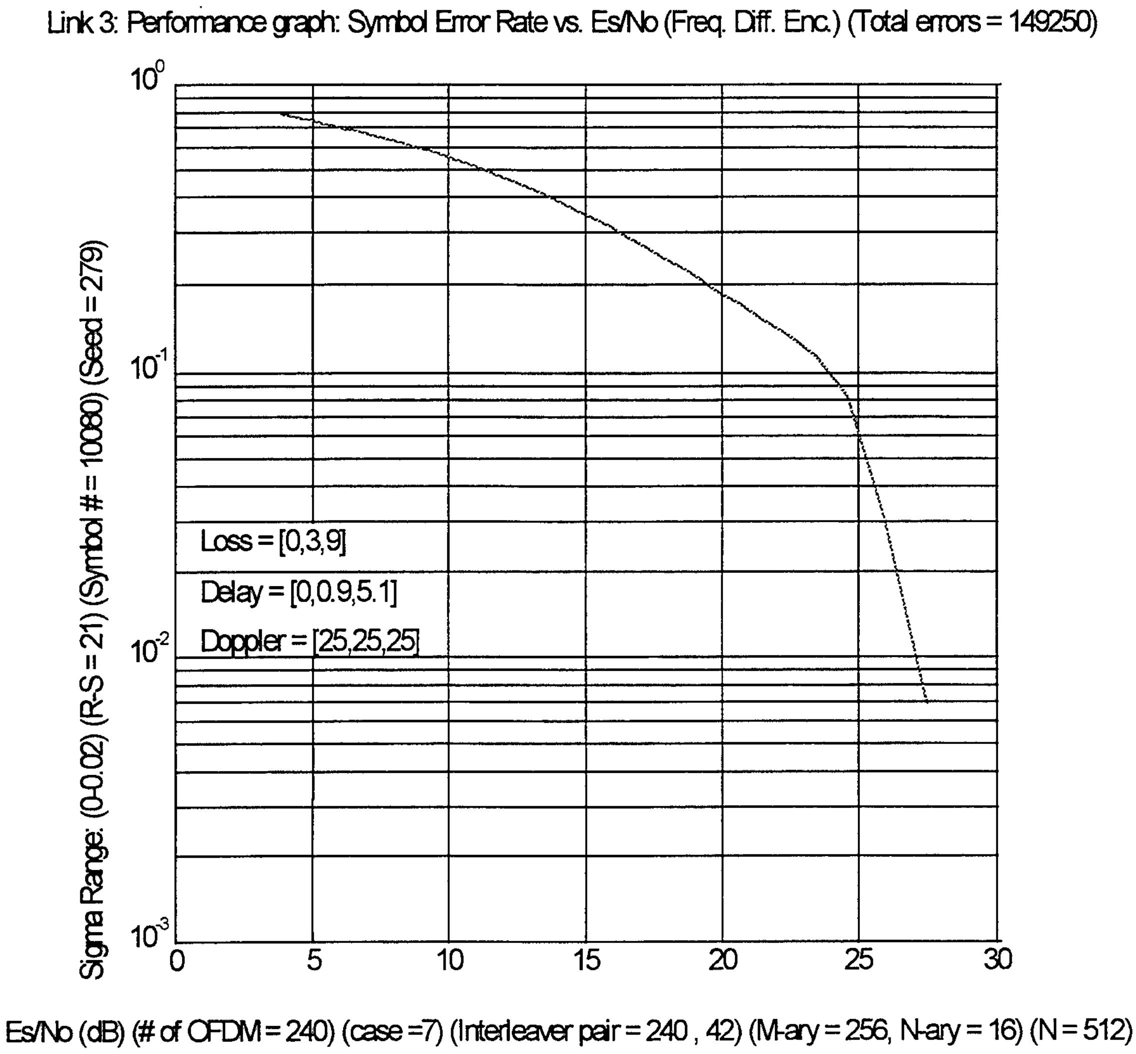
35. Function: uhfseeds.m

```
%____________________________________________________________________________
%
%       Title:          SEED ERROR REPORT
%       Author:         Dave Roderick
%                       Naval Postgraduate School
%       Revised by:     Dave Roderick
%                       Naval Postgraduate School
%
%       Last revision:  4/5/97
%____________________________________________________________________________
%       INPUTS:
%               None
%
%       OUTPUTS:
%               None
%
%       SUBROUTINES USED:
%               uhfift.m
%
% UHFSEEDS: This batch m-file performs numerous OFDM simulations using a channel two
% model (chuhf.m) with various seed input values to generate an error report. No interleaving
% or error correction is performed. This program helps to identify the worst seed values
% which cause the most errors within the channel. Different communication links may also be
% selected.
%____________________________________________________________________________
disp('__________________________________________________________________________');
fort = input('To run the frequency version, enter 1 (one); otherwise, enter 0 (zero) to run the time
version: ');
pthno = input('Do you want to run link1, link2, link3 or a custom link? (Enter 1,2, 3 or 4 for
custom): ');
%
% Link parameters
%
        if pthno == 3
% Link 3
loss = [0,3,9];
dop = [25,25,25];
dly = [0,.9,5.1];
        elseif pthno == 2
% Link 2
loss = [0,5,15];
dop = [10,10,10];
dly = [0,.07,.8];
        elseif pthno == 1
```

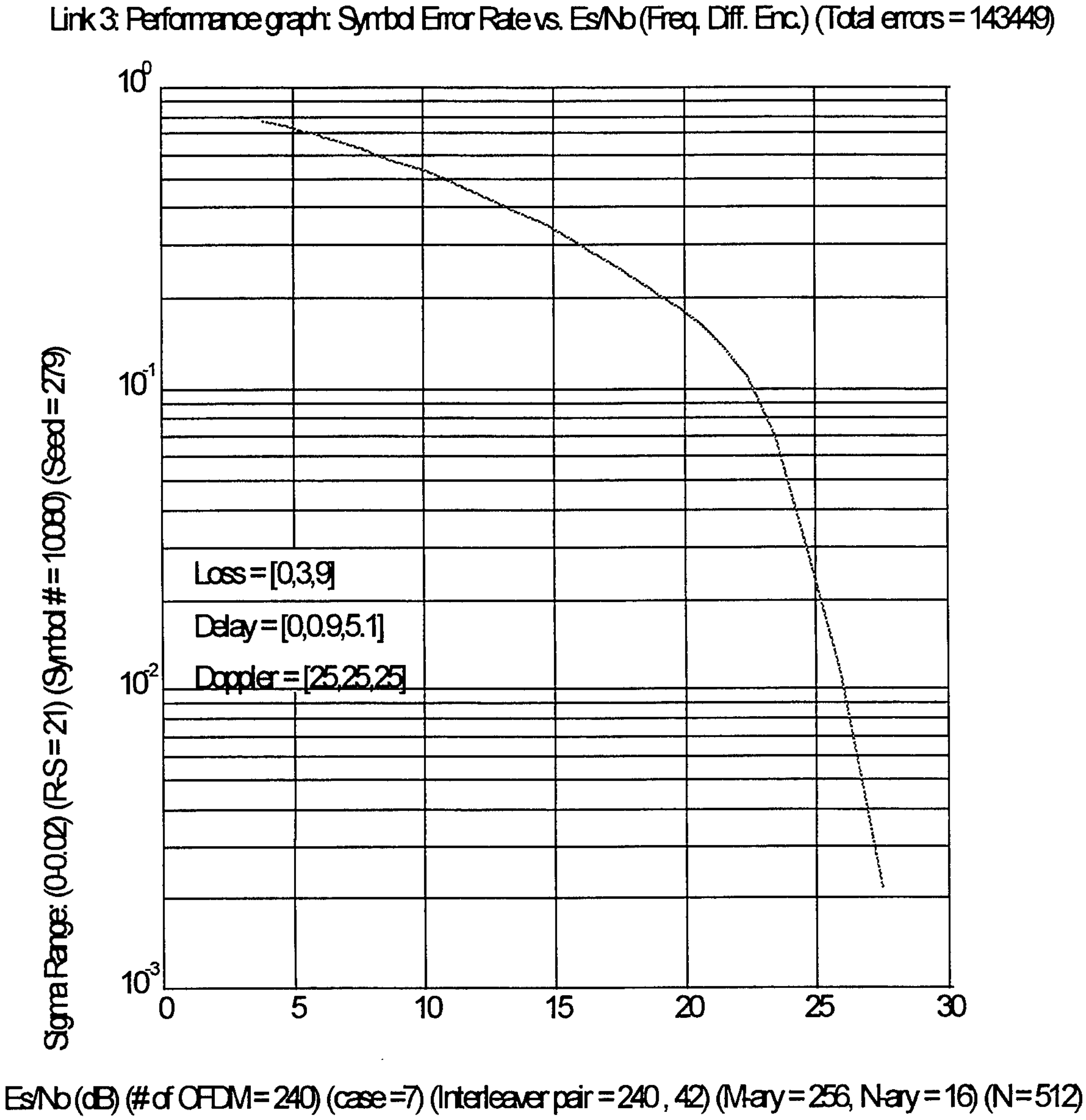
```
% Link 1
loss = [0,6];
dop = [1,10];
dly = [0,.01];
        elseif pthno == 4
disp('Custom link simulation...')
loss = input('Enter the path loss in dB (Ex. [0 4 7]): ');
dop = input('Enter the doppler frequency in Hertz (Ex. [30 20 15]): ');
dly = input('Enter the time delays of the multipaths in microsecs (Ex. [0 0.6 3.9]): ');
        end
prnty_n = input('Do you want print outs? (1 = yes, 0 = no): ');
%
% Simulation input parameters
%
symbols = input('Enter the minimum number of symbols to test: ');
freqno = input('Enter the number of OFDM frequencies (NOTE: Must be even): ');
N = input('Enter the number of FFT points (NOTE: Must be larger than # of OFDM frequencies):
');
smax = input('All tested seeds begin with one and end with a max number.  Enter Smax (Integer #):
');
disp(['Tested seed range is 1 - ',int2str(floor(smax)),' ...'])
mary = 8;
nary = 4;
symno = ceil(symbols/freqno);
freqspc = 480000/freqno;
errvect = [];
incvect = [];
topervect = [];
sindex = 1:floor(smax);
for s = sindex;
%
% If fort equals one, run the frequency simulation version; otherwise, run the time version.
% (Uses macro: uhfift.m)
%[errors,freqerrs]                                                                              =
uhfift(picy_n,pic,s,freqno,rintlv,cintlv,N,mary,nary,n,k,blklgth,Ng,loss,dly,dop,freqspace,fort)
[errors,freqerrs]                                                                               =
uhfift(0,0,s,freqno,freqno,symno,N,mary,nary,0,0,freqno,6,loss,dly,dop,freqspc,fort);
errtot = sum(errors);
errvect = [errvect,errtot];
end
totalerr = sum(errvect);
avgerr = ceil(totalerr/floor(smax));
[inc I] = sort(errvect);
errmx = [I;inc]
Error_Seeds = incvect
Total_Errors = totalerr
```

```
Avg_Errors = avgerr
save unfhist errmx
disp('All Done!!!')
disp(' ')
          if sum(inc) == 0
disp('GREAT!!! Simulation passed with no errors.')
          elseif sum(inc) ~= 0
disp('OOOPS!!! Errors occured.')
          end%
%         *********************************************************
%         *                             Plots                             *
%         *********************************************************
%
figure(3)
bar(sindex,errvect)
grid
orient tall
xlabel(['UHFSEEDS: Seed Value (Symbol # = ',int2str(freqno*symno),')'])
ylabel(['Error Number (OFDM Freq. # = ',int2str(freqno),') (M-ary = ',int2str(2^mary),' ,N-ary =
',int2str(2^nary),')'])
          if fort == 1
                    if pthno == 1
title(['Link1: Error Dist. vs. Seed Values (Freq. Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 2
title(['Link2: Error Dist. vs. Seed Values (Freq. Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 3
title(['Link3: Error Dist. vs. Seed Values (Freq. Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 4
title(['Custom Link: Error Dist. vs. Seed Values (Freq. Diff. Enc.) (Loss = ',num2str(loss),') (Dop
= ',num2str(dop),') (Delay = ',num2str(dly),')']);
                    end
          elseif fort ~= 1
                    if pthno == 1
title(['Link1: Error Dist. vs. Seed Values (Time Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 2
title(['Link2: Error Dist. vs. Seed Values (Time Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 3
title(['Link3: Error Dist. vs. Seed Values (Time Diff. Enc.) (Loss = ',num2str(loss),') (Dop =
',num2str(dop),') (Delay = ',num2str(dly),')']);
                    elseif pthno == 4
title(['Custom Link: Error Dist. vs. Seed Values (Time Diff. Enc.) (Loss = ',num2str(loss),') (Dop
= ',num2str(dop),') (Delay = ',num2str(dly),')']);
                    end
          end
axis([.5 (max(s)+.5) 0 (max(errvect)+1)])
if prnty_n == 1
print
pause(10)
end
figure(4)
bar(inc)
grid
orient tall
xlabel('UHFSEEDS: Seed Value (out of order)')
ylabel('Error Number')
title('Ordered Error Dist. vs. Seed Values (Corresponding Seed Shown on Plot)')
axis([.5 (max(s)+.5) 0 (max(errvect)+1)])
for i = 1:length(errvect)
if inc(i) > (max(inc))*.8
incvect = [incvect,I(i)];
topervect = [topervect,inc(i)];
errlth = length(topervect);
yinc = (max(inc(i))-min(inc(i)))/2;
text(5,(inc(i)+1),int2str(I(i)))
end
end
if prnty_n == 1
print
pause(10)
end
figure(5)
hist(errvect)
title(['Error Histogram (Average # of Errors Per Seed = ',int2str(avgerr),')'])
xlabel('Error Bins')
ylabel('Number of Seeds')
grid
orient tall
if prnty_n == 1
print
end
%______________________________________________________________________________
```

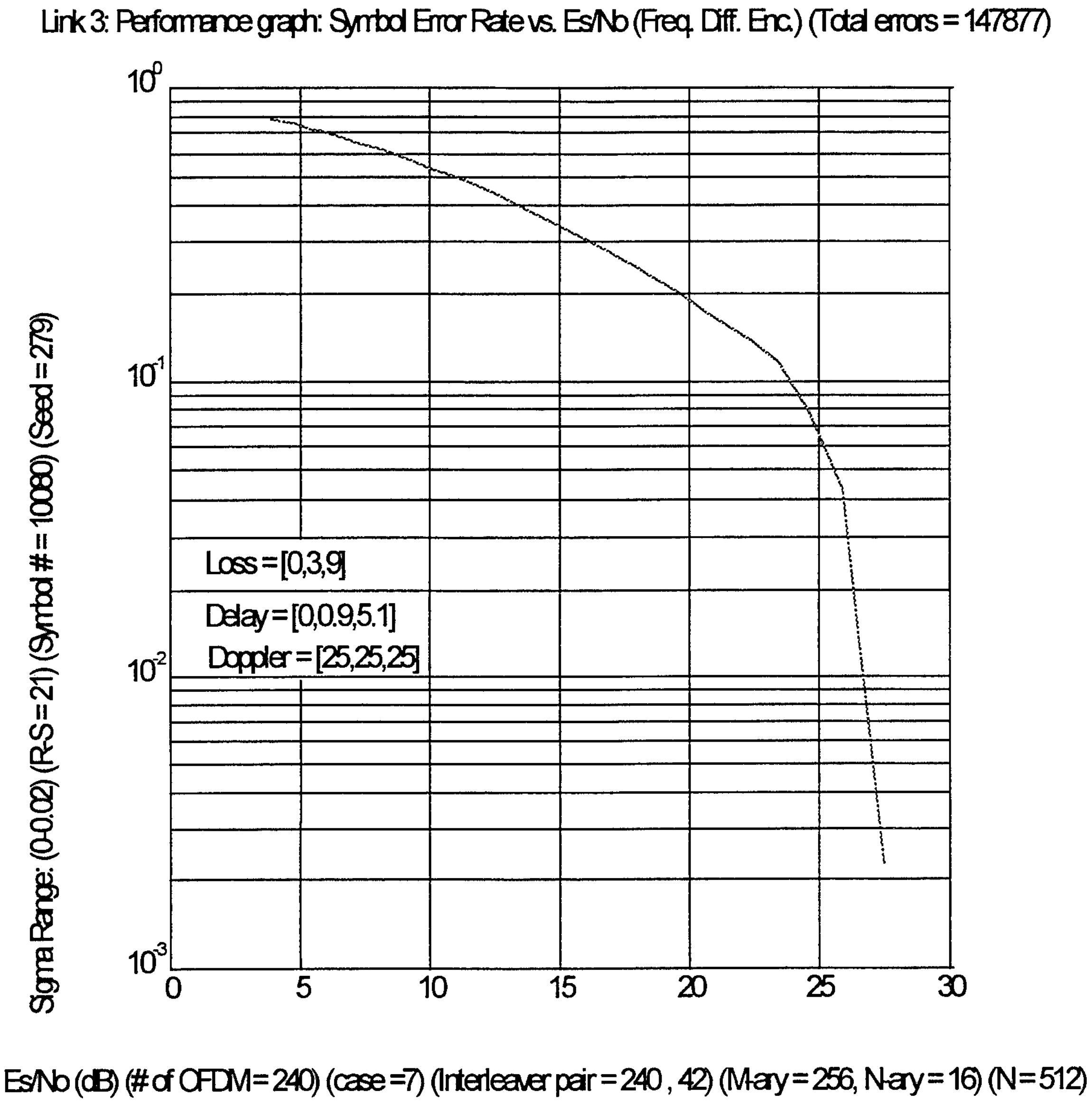
APPENDIX B. VARIOUS SYSTEM SIMULATION PERFORMANCE RESULTS



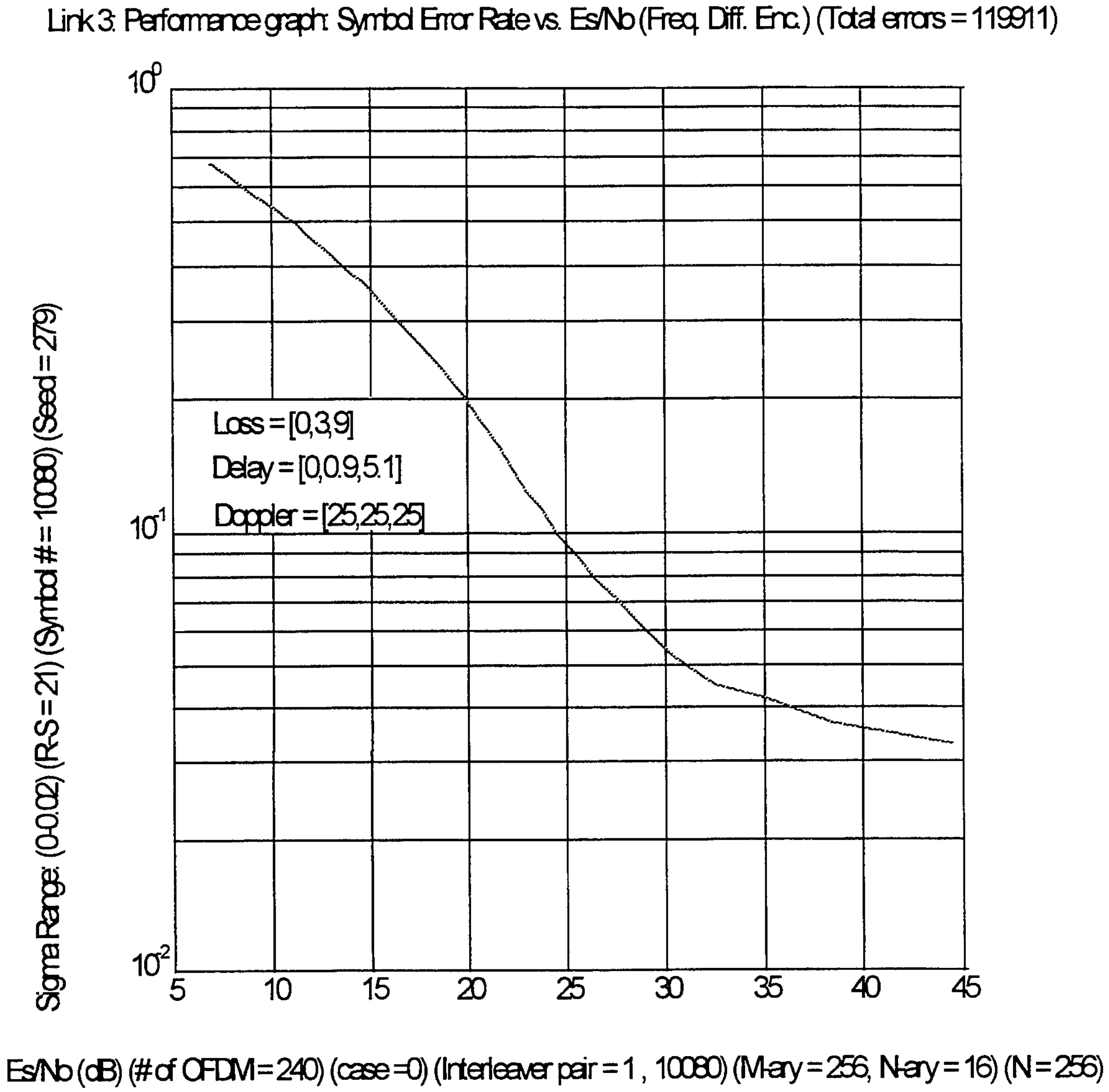
Link 3 System Model 3 Simulation (240 Tones, 512 FFT Points, 1.2% Guard, 8-Bit OFDM Symbols)



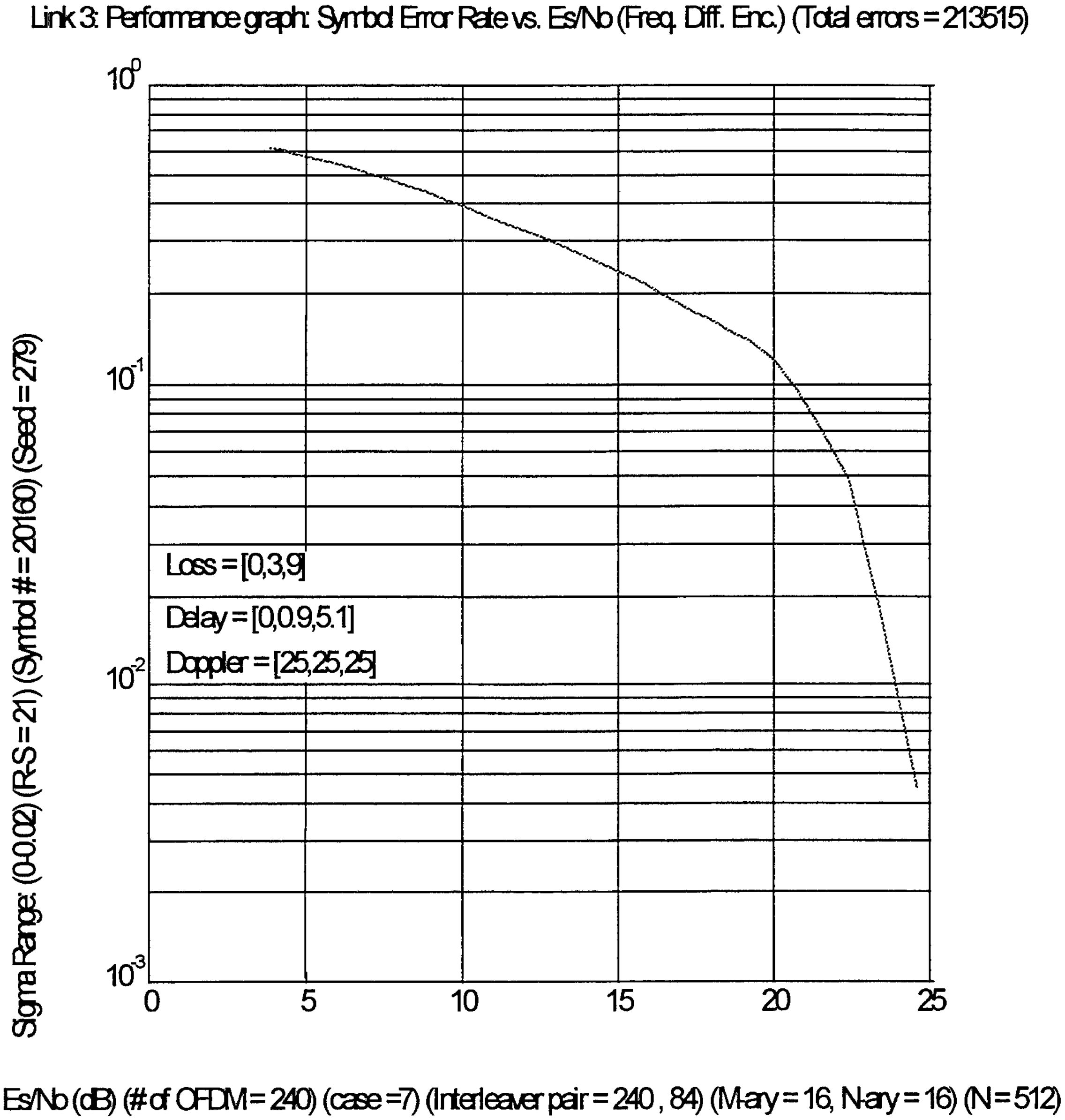
Link 3 System Model 3 Simulation (240 Tones, 512 FFT Points, 4.9% Guard, 8-Bit OFDM Symbols)



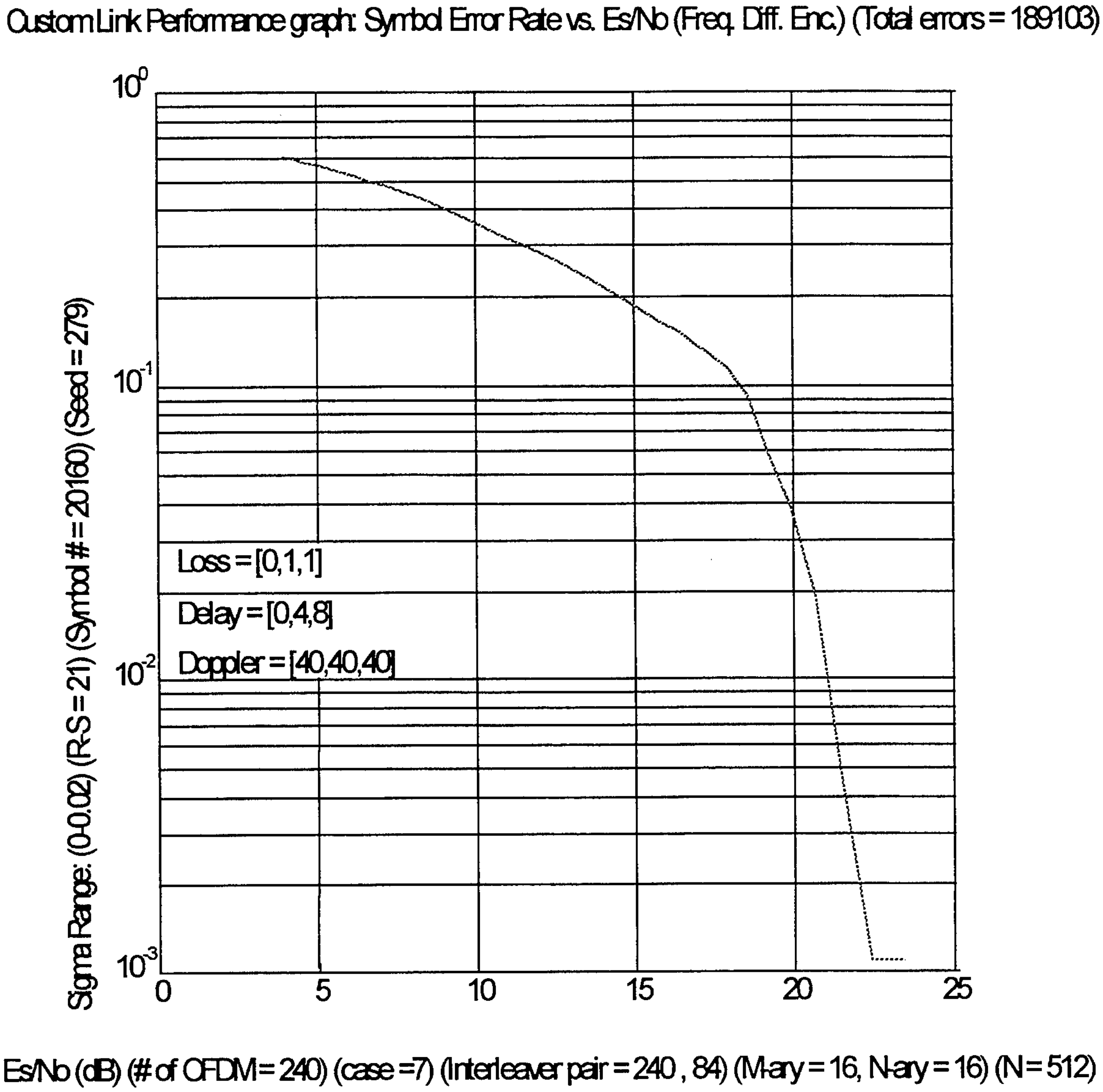
Link 3 System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 8-Bit OFDM Symbols)



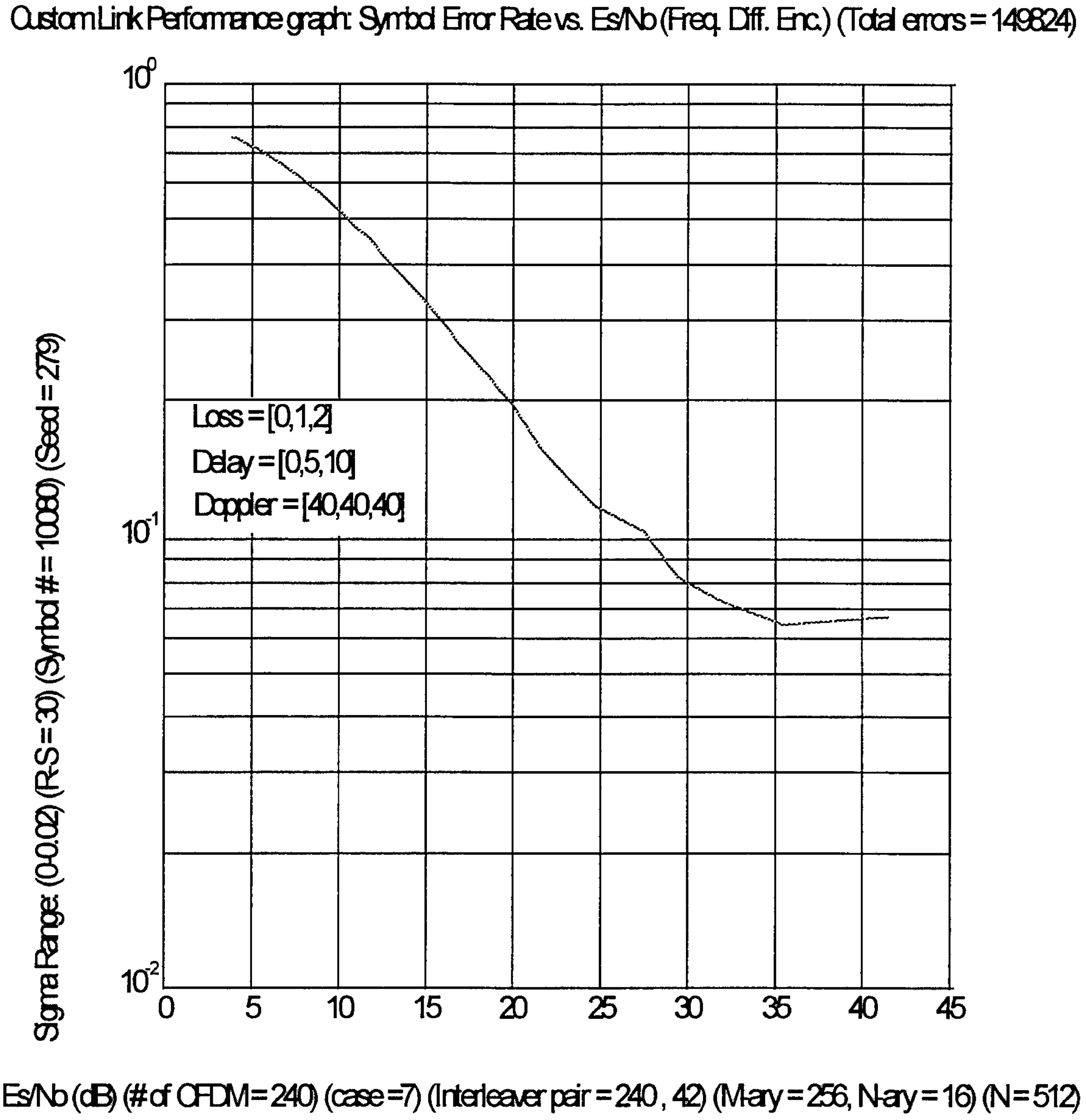
Link 3 System Model 3 Simulation (240 Tones, 256 FFT Points, 2.3% Guard, 8-Bit OFDM Symbols, No Interleaving)



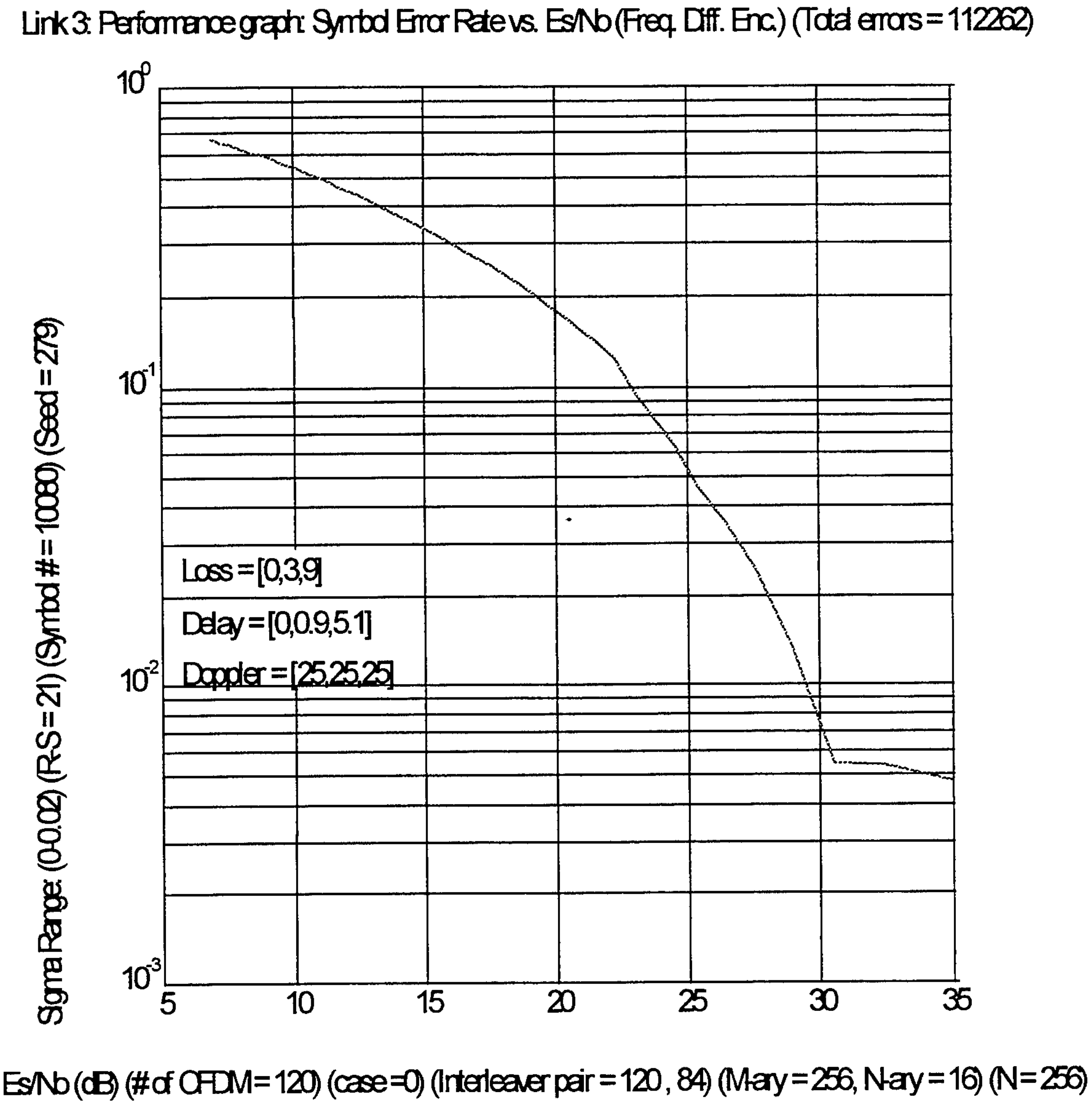
Link 3 System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 4-Bit OFDM Symbols)



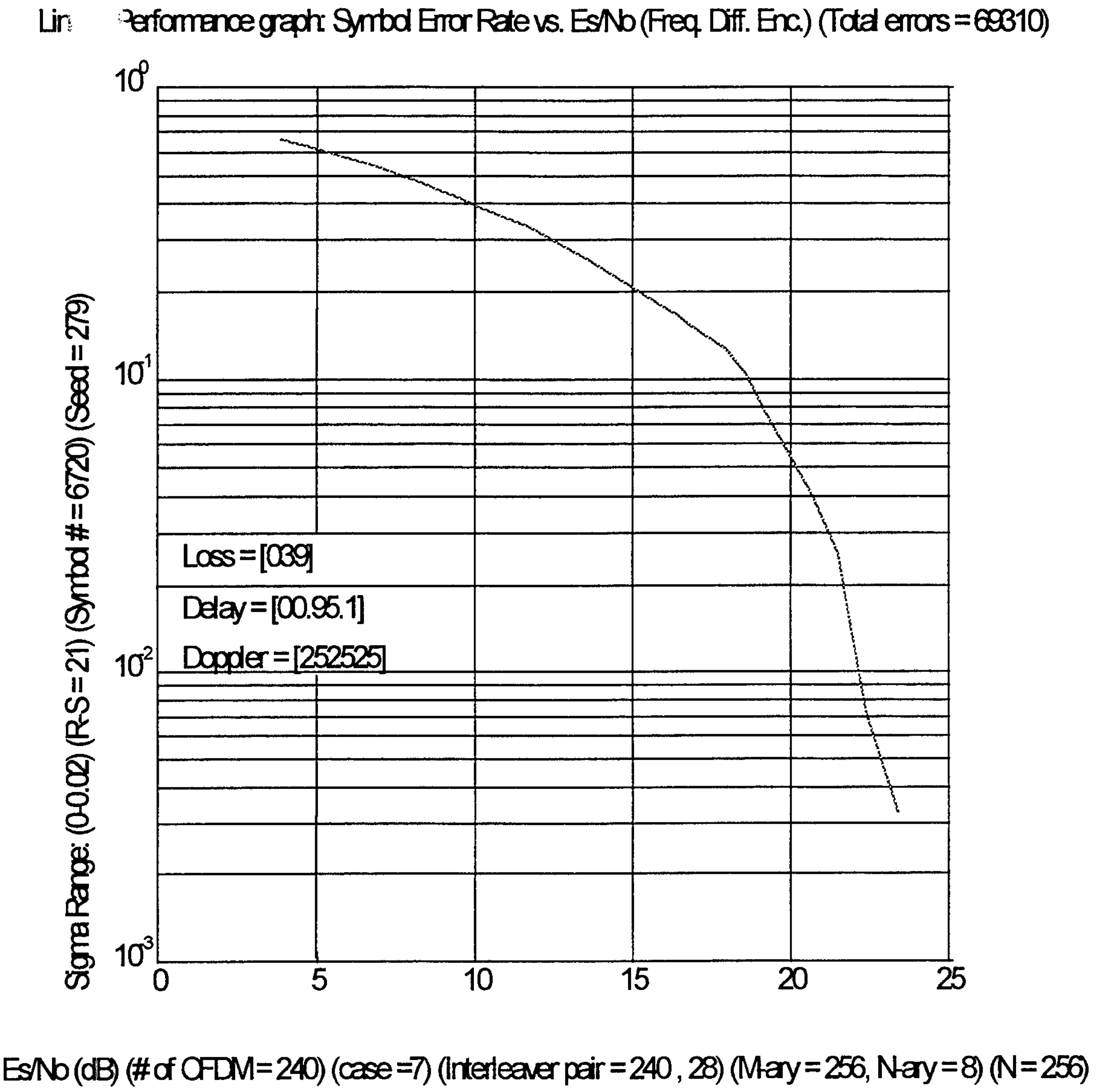
Custom System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 4-Bit OFDM Symbols)



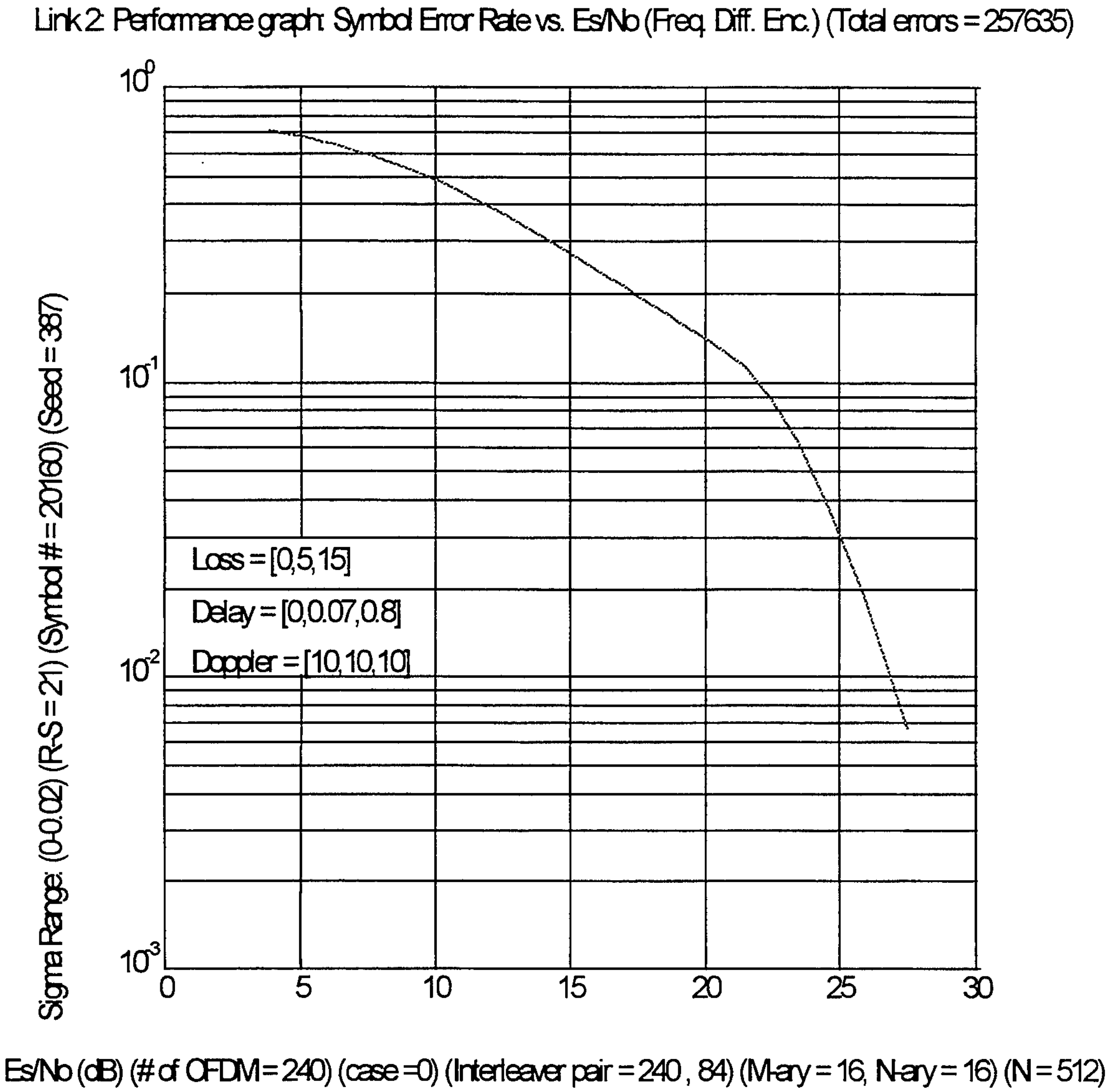
Custom System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 8-Bit OFDM Symbols)



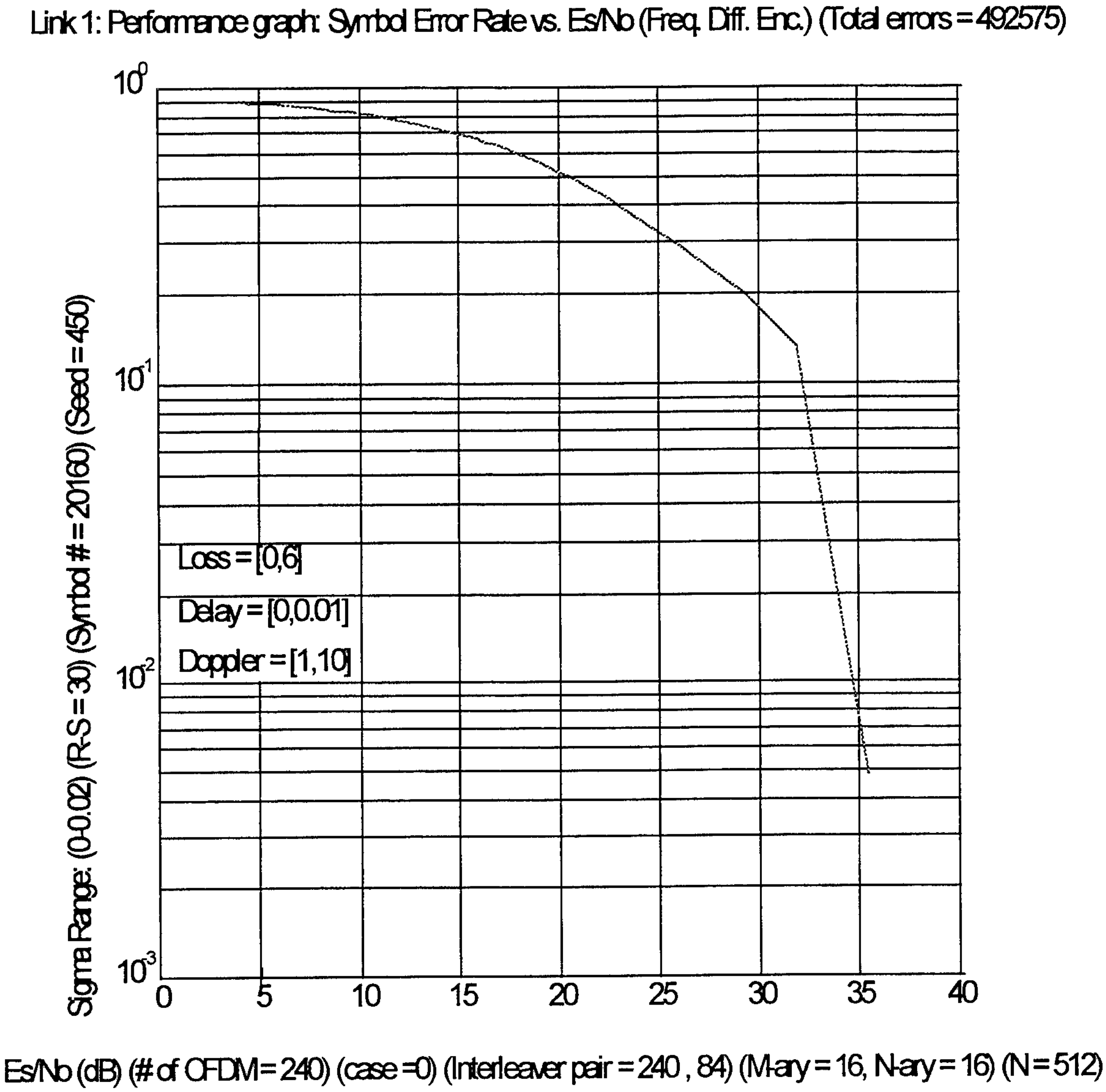
Link 3 System Model 3 Simulation (120 Tones, 256 FFT Points, 2.3% Guard, 8-Bit OFDM Symbols)



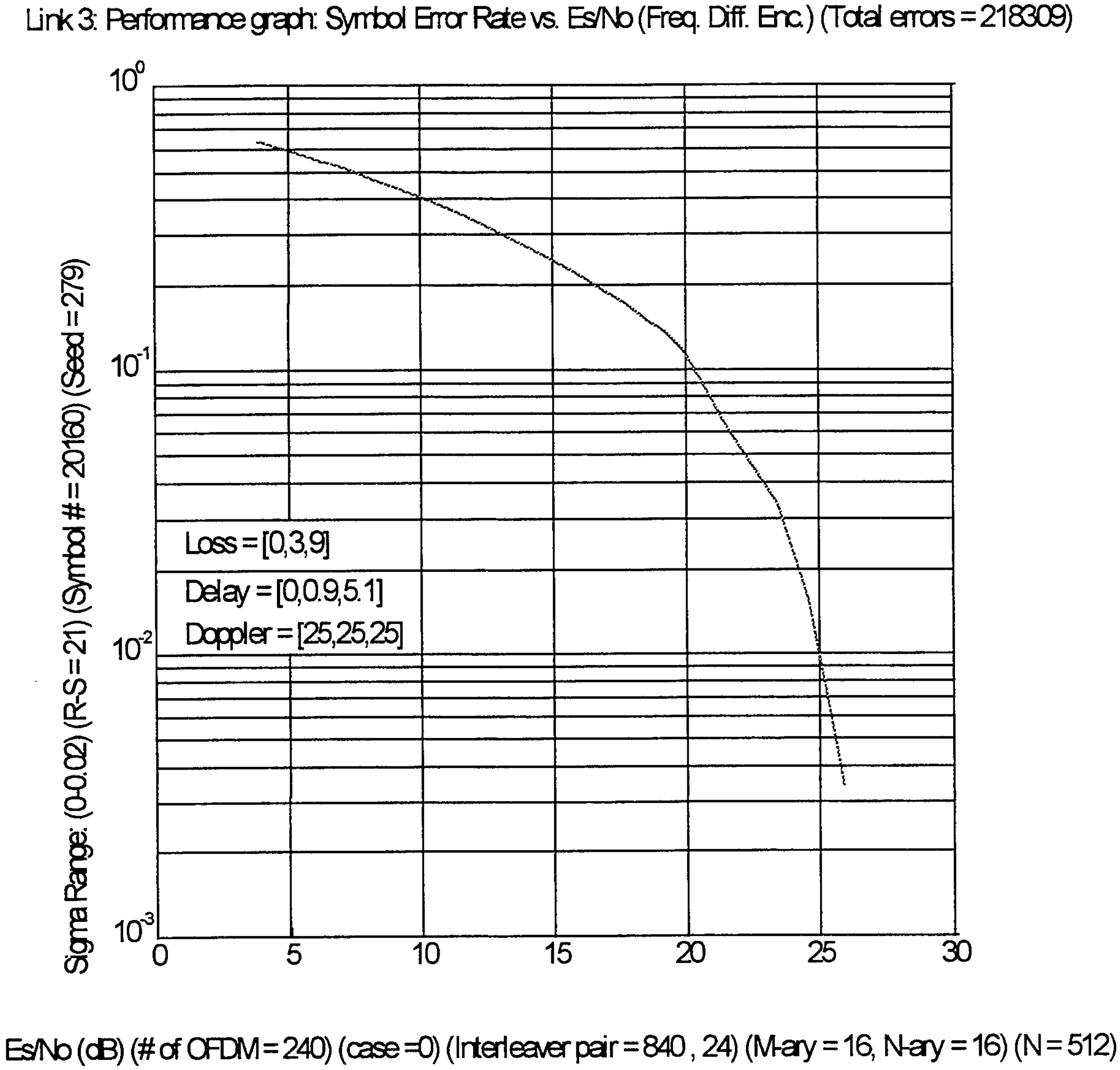
Link 3 System Model 3 Simulation (240 Tones, 256 FFT Points, 2.3% Guard, 8-Bit OFDM Symbols, 3-Bit PSK Symbols)



Link 2 System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 4-Bit OFDM Symbols)



Link 1 System Model 3 Simulation (240 Tones, 512 FFT Points, 2% Guard, 4-Bit OFDM Symbols)



Link 3 System Model 3 Simulation (240 Tones, 512 Fft Points, 2% Guard, 4-Bit Ofdm Symbols, Interleaver (840,24))

LIST OF REFERENCES

[1] Broad Agency Announcement, "High-Data-Rate, Line-of-Sight Digital Radio for Mobile Maritime Communications", N66001-96-X-6911, Naval Command, Control and Ocean Surveillance Center RDT&E Division, August 1996.

[2] William Y. Zou and Yiyan Wu, "COFDM An Overview", *IEEE Transactions on Broadcasting*, Vol. 41, No. 1, pp. 1-8, March 1995.

[3] Leonard J. Cimini, "Analysis and Simulation of a Digital Mobile Channel Using Orthogonal Frequency Division Multiplexing", *IEEE Transactions on Broadcasting*, Vol. Com-33, No. 7, pp. 665-675, July 1985.

[4] Roger L. Freeman, *Telecommunication Transmission Handbook*, John Wiley and Sons, Inc., 1991.

[5] John G. Proakis and Masoud Salehi, *Communication Systems Engineering*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[6] Paul H. Moose, "Simulation of the Common Data Link Using MATLAB", NPS-EC-95-010PR, Naval Postgraduate School, Monterey, CA, July 1995.

[7] R. C. North, W. D. Bryan, R. A. Axford, Jr., K. C. Owens, D. R. Butts, B. Watkins, P. D. Donich, "Use of the AN/WSC-3 External Modem Interface for High-Data-Rate UHF Digital Communication", Technical Report 1701, Naval Command, Control and Ocean Surveillance Center RDT&E Division, San Diego, CA, May 1995.

[8] S. B. Weinstein and Paul M. Ebert, "Data Transmission by Frequency-Division Multiplexing Using the Discrete Fourier Transform", *IEEE Transactions on Communication Technology*, Vol. Com-19, No. 5, pp. 628-634, October 1971.

[9] E. Oran Brigham, *The Fast Fourier Transform*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1974.

[10] Jens Engle, "Digital Data Detection and Synchronization in the Spectral Domain Using FFT", Master's Thesis, Rose-Hulman Institute of Technology, May 1991.

[11] Richard E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Co., Inc., 1983.

[12] Shu Lin and Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications,* Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[13] Bernard Sklar, *Digital Communications Fundamentals and Applications*, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[14] Unisys specification no. 7690698, pp. 24-25, Unisys proprietary.

[15] C.W. Farrow, "A Continuously Variable Digital Delay Element", IEEE International Symposium on Circuits and Systems, pp. 2641-2645, 1988.

[16] Carl W. Helstrom, *Probability and Stochastic Processes for Engineers,* Macmillan Publishing Company, New York, 1984.

BIBLIOGRAPHY

[1] Leon W. Couch II, *Digital and Analog Communication Systems*, Macmillan Publishing Company, New York, 1990.

[2] Gianfranco Cariolaro, “Introduction to Multicarrier Modulation Systems”, Seminars on OFDM Systems, Italtel BUTR, L’Aquila, May 1994.

[3] *MATLAB® High-Performance Numeric Computation and Visualization Software*, Reference Guide, The Math Works Inc., 1992.

[4] Bernard Le Floch, Michel Alard, and Claude Berrou, “Coded Orthogonal Frequency Division Multiplex”, *Proceedings of the IEEE,* Vol. 83, No. 6, pp. 982-996, June 1995.

[5] Pommier and Wu, "Interleaving or spectrum spreading in digital radio intended for vehicles," EBU Review - Technical No. 217, June 1986.

[6] John G. Proakis, *Digital Communications*, 2nd Edition, McGraw-Hill Book Company, 1989.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

3. Chairman, Code EC 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

4. Prof. Paul H. Moose, Code EC/Me 4
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

5. Prof. R. C. Robertson, Code EC/Rc 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

6. Prof. Jovan Lebaric, Code EC/Lb 1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121

7. Dr. Richard C. North, Code 855 1
NCCSOC RDT&E DIV (NRaD)
53560 Hull St.
San Diego, CA 92152-5001

8. Mr. Mike Geile 1
Nova Engineering
5 Circle Freeway Drive
Cincinnati, OH 45246-1201

9. Mr. David V. Roderick 2
PO Box 5891
Monterey, California 93944-0891