Design of a Direct Sequence Spread Spectrum System for Power Line Communication,

John D. Sun

Center for Communications and Signal Processing
Electrical and Computer Engineering Department
North Carolina State University

August 1987

CCSP-TR-87/11

ABSTRACT

SUN, JOHN D. Design of a Direct Sequence Spread Spectrum System for Power Line Communication. (Under the direction of Dr. J. B. O'Neal, Jr.)

A Direct Sequence spread spectrum system is designed for communicating over a power distribution line network. The use of spread spectrum provides a measure of immunity to narrowband interference and to notches in the transmission spectrum of the power line. Synchronization of the spread spectrum signal at the receiver is performed through the use of the zero crossings of the 60 Hz power signal.

Simulations of the system were performed using a general-purpose simulation environment known as BLOSIM. All the functional blocks of the system were simulated in discrete time and in a modular fashion. The results of the simulation indicate that Direct Sequence spread spectrum is a viable technique for use in power line communication systems; however, the use of the zero crossings of the 60 Hz power signal for synchronization places strict demands on the method in which the zero crossings are detected.

ACKNOWLEDGEMENTS

I would like to thank Dr. J. B. O'Neal, Dr. S. H. Ardalan, and Dr. N. J. Rose for serving on my advisary committee and for their technical and personal advice. I would like to give special thanks to Dr. M. Reha Civanlar for his valuable advice throughout the course of this project. Thanks are also in order to Ken Shuey for his interest and support. Finally, my wife deserves my deepest thanks for putting up with me while I struggled to complete this work.

TABLE OF CONTENTS

1.0 INTRODUCTION

The abundance of power distribution lines continues to spur interest in them as channels for communication systems. Of special interest to utilities are possible applications such as remote meter reading, load control, load survey, and feeder automation [7]. All of these applications could and have been controlled by radio or telephone [38]; however, in both cases, their use can cause interference and bring the user under the auspices of the FCC. On the other hand, transmission of signals over the power distribution line usually will not require the attention of the FCC [7].

While conceptually simple, the design of an effective Distribution Line Carrier (DLC) communication system faces a major problem in the form of the hostile environment that exists on power distribution lines. The noise on distribution lines can be broken down into four parts: 1) flat background noise, 2) harmonic noise at multiples of 60 Hz, 3) single periodic tones at frequencies unrelated to 60 Hz and 4) impulse noise. Another degradation that also occurs in distribution lines is notches in the transmission spectrum caused by standing wave patterns.

Existing DLC systems, combat the noise problem by using signals that are synchronous to 60 Hz and/or high transmission power [36]. These solutions though are still extremely susceptible to the effects of standing waves and

to single periodic tone noise.

The use of Spread Spectrum techniques has been proposed to alleviate some of the noise problems mentioned above [40,68]. Perhaps the most important feature of spread spectrum systems that make it ideal for DLC systems is theirinterference suppression properties [25] and immunity to notches in the received signals spectrum. Some other advantages that spread spectrum systems offer are: 1) low probability of detection [6], 2) asynchronous code-division multiple access (CDMA) [15,16], and 3) multipath rejection [67].

There are two basic types of spread spectrum systems: Direct Sequence (DS) and Frequency Hop (FH). For both systems, the use of pseudorandom (PR) or pseudonoise (PN) codes play a central role. In a DS system, the carrier is modulated by a high rate PN code which spreads the frequency spectrum of the original signal. In a FH system, the PN code controls the "hopping" of the carrier's frequency thus increasing the system's effective bandwidth. (General discussion and tutorials on spread spectrum systems can be found in: [10,20,44,57,62,66,70,75].)

One spread spectrum DLC system has already been proposed by Van Der Gracht and Donaldson [68]. In their system, Direct Sequence spread spectrum is employed along with standard BPSK. The unique feature of their system is the use of the 60 Hz zero crossing to synchronize the PN code and the data signal. This idea eliminates the need

for expensive and complicated tracking schemes which is one of the main disadvantages of spread spectrum. Unfortunately, the application and results presented in [68] are for communication over localized (secondary) power distribution networks (e.g. among offices in a building or between local buildings) and not for communication between substations.

In this thesis, a new DS spread spectrum DLC system is proposed for communication between substations along the power distribution line network. The proposed system is an extension of the system presented in [68] and retains the idea of using the 60 Hz zero crossings to synchronize the PN code and the data. A key feature of the proposed system (in addition to the properties of spread spectrum mentioned earlier) is the use of a modified duobinary signaling scheme which utilizes the spectrum between the harmonic noise components effectively. This scheme serves to increase the bit rate over that of existing DLC systems [38] with corresponding improvements in the bit error rate.

The remainder of this thesis is organized as follows. Section 2 describes the distribution line channel in which the proposed DLC systems is to operate. Sections 3 and 4 discuss the design of the transmitter and receiver. Section 5 outlines the design of the simulations and evaluates the performance of the proposed system as predicted by the simulations. Finally, section 6 presents conclusions and ideas for further research.

2.0 THE POWER DISTRIBUTION LINE AS A CHANNEL

The primary use of the network of distribution lines is to deliver electric power and, consequently, distribution lines are a harsh environment in which to communicate. The high noise level on the lines and/or propagation effects caused by the network topology can potentially cripple standard communication systems. Both the noise and propagation effects on distribution lines are reviewed in this section. (Detailed discussions on power line noise can be found in [19,37,39,61,65,69] and much of the discussion in this section is drawn from those references.)

The noise on distribution lines can be broken down into 4 types [39]. They are:

1. Harmonic noise at multiples of 60 Hz
2. Flat background noise
3. Single tone noise at frequencies unrelated to 60 Hz
4. Impulse noise.

Harmonic noise is the predominant form of noise on the distribution line [69]. This noise is found on every distribution line and is synchronous to the 60 Hz power signal. It exists at multiples of 60 Hz and each harmonic is usually 3 to 5 Hz wide. The source of harmonic noise stems directly from the 60 Hz power signal and synchronous switching devices on the line.

Flat background noise is that component of the line

noise whose power spectral density is flat. In reference to harmonic noise, this noise is the part of the noise spectrum that lies between the harmonic noise components (see Figure 2.1). While the noise is not exactly flat across the entire spectrum of the distribution line channel, it is relatively flat in the band in which most DLC systems operate, 0 - 25 KHz, and is treated as typical flat noise in this region. The source of this noise originates from equipment connected to the secondary of distribution transformers [69].

Single tone noise are noise spikes that occur at frequencies which are unrelated to 60 Hz. A common source of these noise spikes are broadcast systems such as television, radio, CB's, and mobile radio signals [39].

Impulse noise on the distribution line is usually intermittent noise. The source of impulse noise is usually due to switching transients. For example, turning on vacuum cleaners, thermostats, and washing machines all cause impulse noise. Lightning and capacitor bank switching are other sources of impulse noise [39].

Figure 2.1 shows the power spectrum of a typical distribution line. This figure was taken from a report on the measurement of substation noise on the distribution line in the 6-15 KHz range [39]. Some of the results of this report are summarized below.

1. The background noise level varied between -5 and 28 dBmV depending on the substation location.

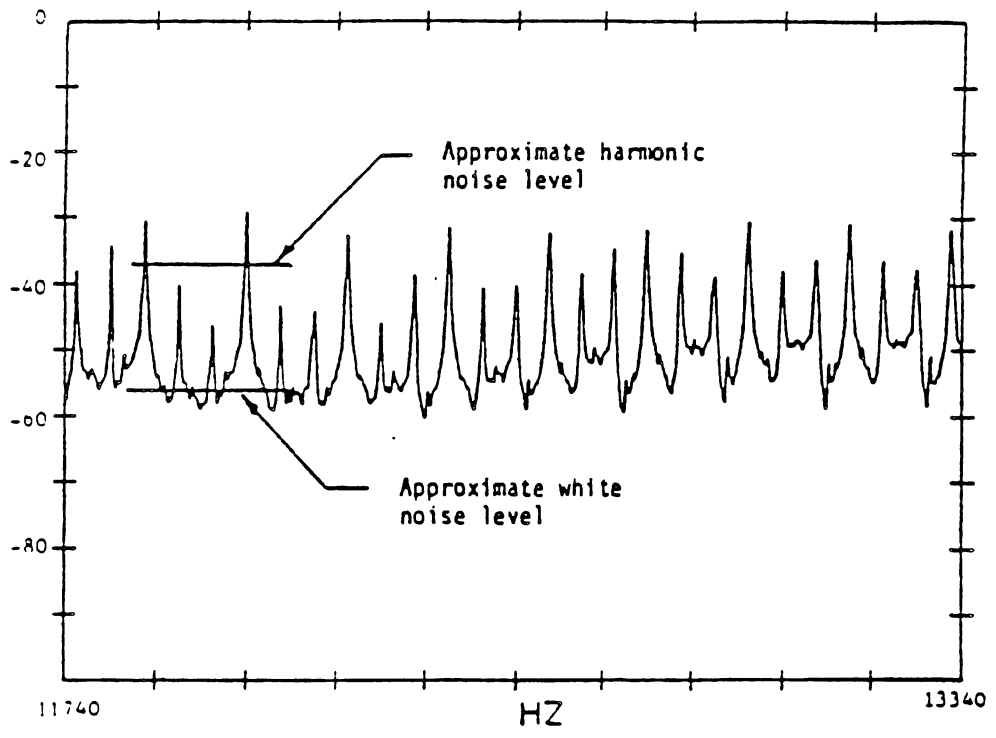


Figure 2.1 - Sample Power Spectrum of the Noise on Power Distribution Lines (from [39])

2. On the average, the power spectrum of the harmonic noise is approximately 22 dB greater than that of the background noise.
3. Histograms of the sampled noise show that the noise can be reasonably approximated by a gaussian probability density function.

Propagation effects on power distribution lines deal primarily with standing waves. Standing waves are caused by the network topology and terminals that are not terminated properly. Various topologies consisting of both terminated and unterminated terminals can cause reflections of a transmitted signal which create standing wave patterns. These standing waves can cause notches to appear in the spectrum of the distribution line and cause fading to occur in the received signal of a DLC system. The standing wave patterns are also time varying because of load changes during the day [39]. (Progress in predicting standing wave patterns has been made using computer programs such as CAPNET [1].)

3.0 DESIGN OF THE TRANSMITTER

3.1 System Design

One major requirement in designing the transmitter is that the transmitted signal's spectrum have nulls at the harmonic noise frequencies. This requirement makes it possible to minimize the distortion caused by the harmonic noise at the receiver. Another important requirement is that the PN code used in the spreading process be synchronized to the zero crossings of the 60 Hz power signal. This requirement greatly simplifies the tracking and acquisition of the PN code phase in the receiver. These two requirements when considered together imply that the system should fully exploit the benefits of synchronous operation provided for by the 60 Hz power signal.

The design of the transmitter can be broken down into two parts: the design of the message signal (i.e. the signal to be spread) and the design of the spreading system. A block diagram of the transmitter design is shown in Figure 3.1 and is described as follows. The input data sequence $\{d_k\}$ is a random binary sequence of 1's and 0's. This data sequence is first precoded and converted into impulses. (See section 3.2 for the reason for the precoding.) After appropriate gain, these impulses are then filtered by $H_T(f)$ to generate the message signal $m_o(t)$. Instead of modulating at this point, as is done in existing systems [36], the message signal is first spread

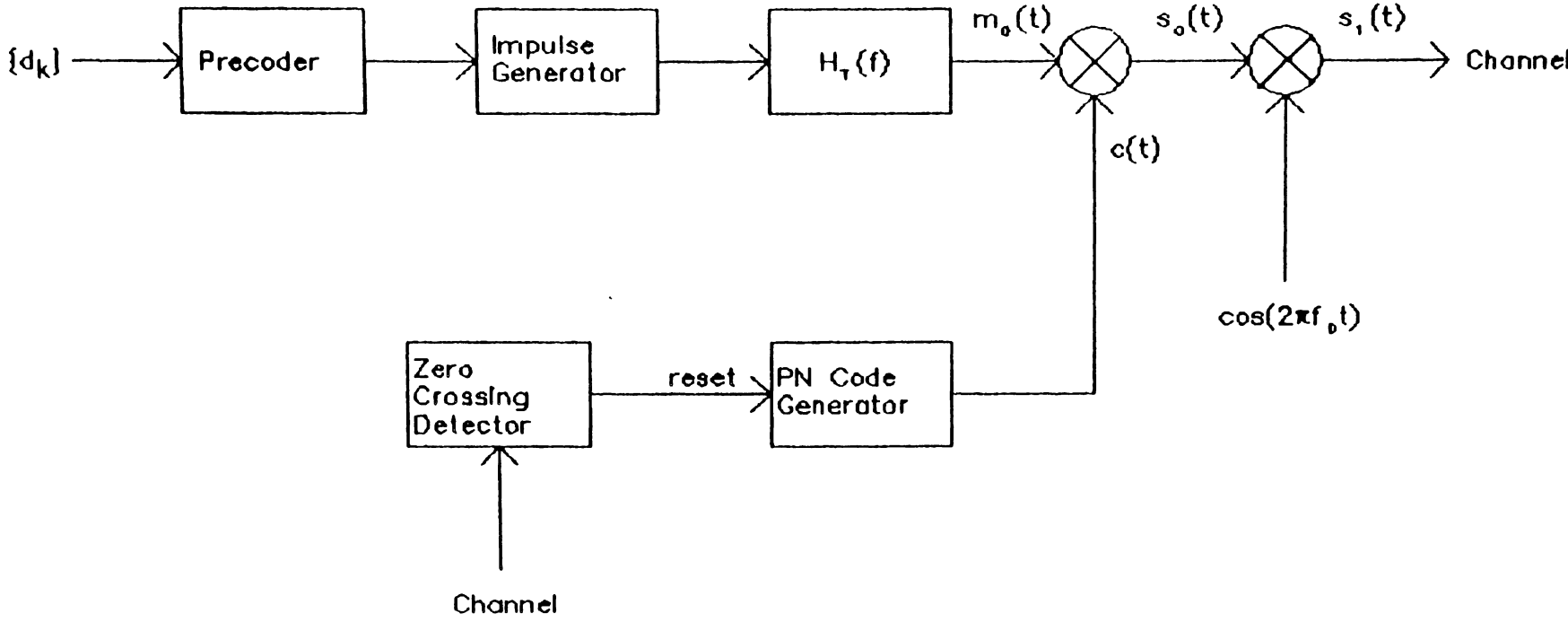


Figure 3.1 – Transmitter Design

using Direct Sequence (DS) spread spectrum. The spread signal $s_o(t)$ is then modulated to form the transmitted signal $s_1(t)$.

3.2 The Modified Duobinary Signal

The modified duobinary signal shape is one that is suitable for use as the impulse response of the transmission filter $H_T(f)$ in Figure 3.1. The reasons can be found by examining the frequency spectrum of the modified duobinary signal. In the time domain, the modified duobinary signal is specified by

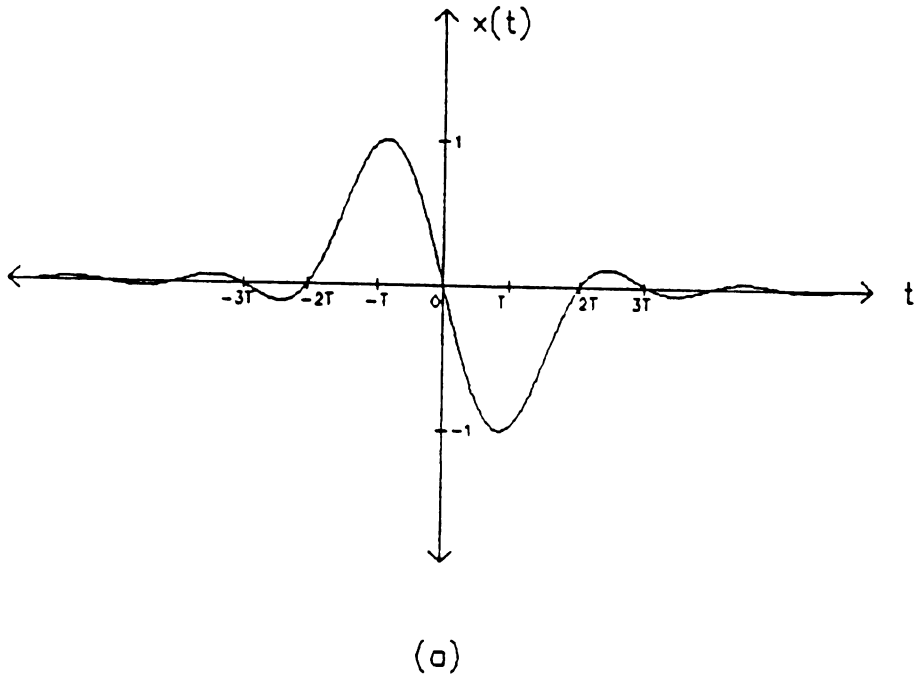
$$x(t) = \frac{\sin 2\Pi W(t + \frac{1}{2W})}{2\Pi W(t + \frac{1}{2W})} - \frac{\sin 2\Pi W(t - \frac{1}{2W})}{2\Pi W(t - \frac{1}{2W})} \qquad (3.1)$$

and it's frequency spectrum is expressed as

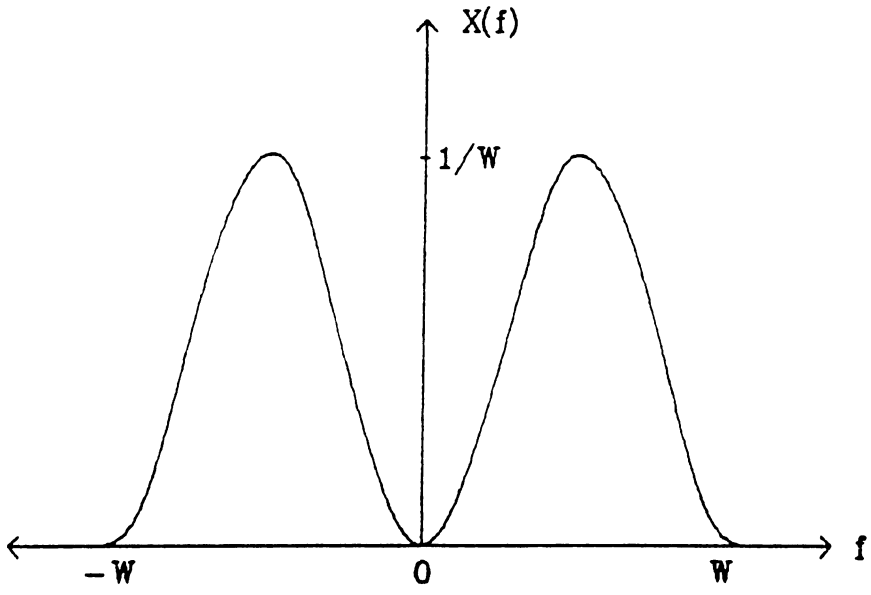
$$X(f) = \frac{j}{W} \sin(\frac{\Pi f}{W}) \, \mathrm{rect}(\frac{f}{W}) \qquad (3.2)$$

Figure 3.2 shows the time and frequency domain characteristics of the modified duobinary signal. (The modified duobinary signal is discussed in complete detail in [11,12,27,32].)

The main feature of the modified duobinary signal that makes it suitable for DLC systems is the fact that it's frequency spectrum is zero at DC as is shown in Figure 3.2b. The reason that a zero DC value is important is



(a)



(b)

Figure 3.2 - Time (a) and Frequency (b) Characteristics of the Modified Duobinary Signal

because when the signal is modulated, a noise harmonic can exist in the center of the signal's spectrum without distorting the signal. This fact allows for undistorted transmission of a modified duobinary waveform whose modulated bandwidth is 120 Hz which corresponds to a bit rate of 120 bits per second (bps). In contrast, modulating a 120 bps raised cosine waveform would place at least one harmonic noise component on top of the signal spectrum.

Using a modified duobinary signal is not without disadvantages though. The modified duobinary signal belongs to a class of signals known as *partial response signals* [13,23,28,30,31] which means that intersymbol interference (ISI) is present. Fortunately, the ISI that does exist is deliberately introduced and can be removed through a precoding operation [29,47]. This precoding operation is similar to that done for Differential PSK and is of the form

$$p_k = d_k \oplus d_{k-2} \tag{3.3}$$

where $\{d_k\}$ represents the random binary input data sequence, $\{p_k\}$ represents the precoded data sequence, and $\oplus$ represents modulo-2 addition. A simple decision strategy can then be used to recover the original data sequence. This strategy will be discussed in section 4.

3.3 The Application of Direct Sequence Spread Spectrum

The process of spreading the modified duobinary message signal using DS spread spectrum is fairly straight forward. As is shown in Figure 3.1 all that is involved is multiplication of the message signal m(t) by a high rate spreading sequence c(t) also known as a pseudonoise (PN) code.

The major design issue here is in the design of the PN code. The parameters of the PN code that must be specified are the processing gain, the period of the code, and the type of code [54]. All of these parameters are interrelated with the requirement that the PN code be synchronized with the zero crossings of the 60 Hz power signal. The first parameter to consider though is the processing gain since this parameter ultimately determines the spectral usage.

The processing gain G is simply a parameter that specifies the amount of spreading that occurs relative to the bandwidth of the original signal and is defined by $G = B_c/B_d$ where B_c denotes the bandwidth of the spread signal and B_d denotes the bandwidth of the original signal [56]. High processing gains are desired and indicate that the power in the original signal is spread over a wide frequency band making the transmitted signal difficult to detect or jam. In the case of DLC systems, the channel's limited bandwidth will also limit the processing gain. Typically, DLC systems use the 0-25 KHz band for

transmission because of degradations caused by the distribution transformers at frequencies above 25 KHz [61]. With this restriction in mind, the proposed DLC system's transmitted signal has a bandwidth of approximately 15 KHz centered at 12.5 KHz. Since the message signal is the modified duobinary signal operating at 120 bps discussed in section 3.2, the processing gain is approximately 125.

The period of the PN code can now be determined by considering the requirement that the code be aligned with the zero crossings of the 60 Hz power signals. Since there are two zero crossing per period of the 60 Hz power signal, if exactly one period of the PN code is fit between the zero crossings, then the code will be periodic with frequency 120 Hz. The number of chips per period then is determined by dividing the chip rate by the frequency of the PN code. In this case, the chip rate R_c is 7.5 KHz and the number of chips per period is 62.5. This number can be rounded to 63 if the bandwidth of the spread signal is allowed to increase slightly to 15120 Hz and the chip rate to 7560 Hz. The processing gain then becomes 126.

The last parameter that must be specified is the type of PN code to be used. The two most common PN codes used in current spread spectrum systems are maximal length and Gold codes. More information can be found on these and other codes in [17,49,55,63]. Because of the ease in which it is generated, a maximal length code is used as the PN code to spread the modified duobinary message signal.

With the above parameters specified, the spectrum of the spread signal $s_o(t)$ can be determined by examining the spreading process. When the message signal m(t) is multiplied by the PN code c(t) in the time domain, the spectrum of the message signal is convolved with the spectrum of the PN code in the frequency domain. Since the PN code is periodic with frequency 120 Hz, it's spectrum is discrete with impulses at multiples of 120 Hz and the envelope of these impulses has the familiar sinc shape with the first null at 7560 Hz [58]. Convolution of the modified duobinary signal's spectrum with this spectrum of impulses results in replicas of the modified duobinary signal's spectrum spaced every 120 Hz. The envelope of the resulting signal though still has a sinc shape and a null at 7650 Hz. Figure 3.3 shows the theoretical spectrum of the PN modulated modified duobinary signal.

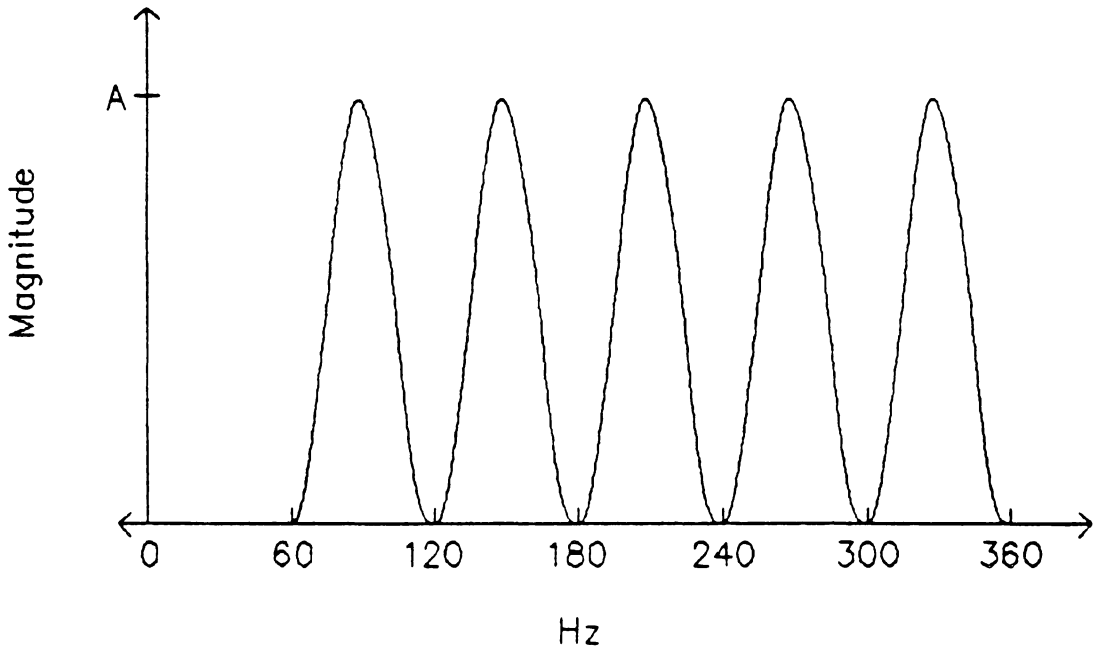


Figure 3.3 - Spectrum of the PN Modulated Modified Duobinary Signal

4.0 DESIGN OF THE RECEIVER

4.1 System Design

The design of the receiver is such that the process of converting the received signal back to the original data sequence consists of three steps. As shown in Figure 4.1, they are: 1) downconverting the received signal to baseband, 2) despreading the baseband received signal by synchronizing the receiver's reference PN code phase to the received signal's code phase, and 3) obtaining the original data sequence from the despread modified duobinary signal. (In Figure 4.1 filtering occurs at several points in the receiver design; however, for ease of analysis, they are assumed to be ideal filters.)

The downconversion process simply involves multiplying the received signal, $r_1(t)$, by a reference carrier and filtering out the upconverted signal. This operation is a common operation in most communication systems and is assumed to be performed synchronously here.

After downconversion, despreading is done by multiplying the baseband received signal, $r_o(t)$, by the receiver's reference PN code, c(t); however, the process of synchronizing the reference PN code to the received signal's code phase is perhaps the most complex part of a spread spectrum system [53]. Fortunately, in this design, the synchronization problem is greatly simplified by the use of the zero crossings of the 60 Hz power signal to

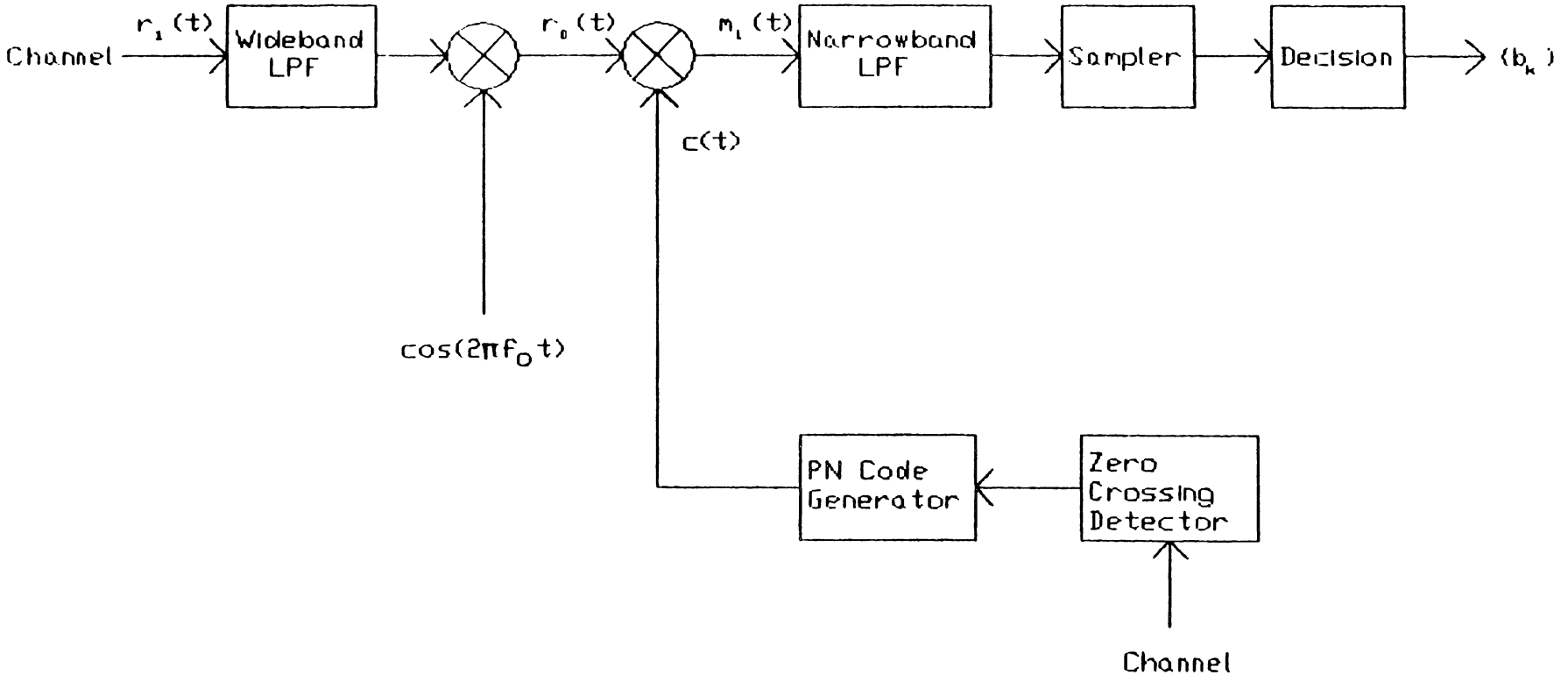


Figure 4.1 - Receiver Design

synchronize the code phases. Synchronization aspects are discussed in section 4.3.

After the received signal has been downconverted and despread, the last step is obtaining the original data sequence from the recovered modified duobinary signal, $m_1(t)$. This process involves first sampling the despread signal and then applying decision thresholds to the sampled values to recover the original binary data sequence. The decision process for the modified duobinary signal is discussed in section 4.2.

4.2 The Decision Process for Modified Duobinary Signaling

As mentioned in section 3.2, because controlled amounts of ISI are added to the modified duobinary signal, the original data sequence is precoded so that the ISI can be removed later. Where the ISI is effectively removed though is in the decision scheme used at the receiver.

Instead of the common single decision threshold level used in standard binary systems, bi-level decision thresholds are used. (The bi-level decision scheme is derived in detail in [43,47].) In this decision scheme, if the peak amplitude of the transmitted modified duobinary pulse is A, then the two decision thresholds are A and -A. The decision process is then described by the following rule

$$i_n = \begin{cases} 1 & \text{if } -A < y(nT) + v_n < A \\ 0 & \text{if } |y(nT) + v_n| \geq A \end{cases}$$

where $\{i_n\}$ represents the receivers decoded binary data sequence, $\{y(nT)\}$ represents the sampled values, and $\{v_n\}$ represents the noise. When no noise is present and ideal sampling is assumed, then the sampled values, y(nT), are either 2A, 0, or -2A.

4.3 Synchronization Aspects

In most spread spectrum systems, the code synchronization process involves first acquiring the received signal's code phase and then tracking this phase. These two steps are known as the acquisition and tracking process and are extremely important in determining the performance of the receiver. (See [20,21,26,53,62,75] for thorough discussions on the various acquisition strategies and tracking techniques.)

Since the zero crossings of the 60 Hz power signal are to be used in synchronizing the reference PN code to the received signal's code phase, the need for complex acquisition and tracking circuitry is reduced. Synchronization is now dependent on detecting the zero crossings accurately. When the zero crossings are detected accurately, the received signal is properly despread; however, when there is jitter in the detection of the zero crossings, the reference PN code and the received signal's

code phase will be out of alignment. Exactly how much jitter can be tolerated before the misalignment in the codes' phases causes the receiver to no longer despread the received signal will be examined through simulations and discussed in section 4.0.

Not only is the PN code synchronized to the zero crossings of the 60 Hz power signal, the modified duobinary signal is also synchronized to the same zero crossings. Jitter in the detection of the zero crossings will then change the sample times which can cause errors in the decoding of the original data sequence. How much jitter can be tolerated will also be examined by simulation and discussed in section 5.1.

5.0 SIMULATING THE DUOBINARY/SPREAD SPECTRUM DLC SYSTEM

To simulate the proposed DLC system, a simulation program known as BLOSIM was used [35]. BLOSIM which stands for "Block Simulator" is an extremely flexible simulation program especially suited for simulating systems which can be modeled as sampled data or discrete time systems. The flexibility of BLOSIM comes from the manner in which simulations are constructed. Like many other simulation programs, BLOSIM is a hierarchal program which allows the user to construct simulations or other modules by linking existing BLOSIM modules (called STARS and GALAXIES) together; however, unlike many simulation programs, BLOSIM also allows the user to construct primitive modules through direct programming in C. This ability to program primitive modules allows the user enormous power and flexibility in constructing efficient simulations.

5.1 Simulation Description

The proposed DLC system was simulated in BLOSIM by creating many STARs and GALAXIES to simulate the various elements in the proposed system. The overall simulation itself can be broken down into three functional simulations: the transmitter, the channel, and the receiver. Each simulation consists of several STARS which are described in detail in Appendix 8.2. The following describes the setup of the simulations. (The topology

files for the simulations are shown in Appendix 8.3.)

Figure 5.1 shows the block diagram of the transmitter simulation. Each block represents a STAR and the lines between the blocks represent the actual connections. The random binary data sequence is generated by the DATAGEN block. The PRECODER block then precodes the data sequence as described in section 3.2. The data sequence is also written to file by the block INT_PRINT. The next block, BITADJUST, converts the binary digits into impulses with a user-specified number of zeroes between the impulses. The specification of the number of zeroes between impulses in effect sets the sampling rate. The DUOBINARY block then convolves the impulses with a discrete modified duobinary impulse response to generate the modified duobinary signal. Finally, this signal is multiplied by the PN code which is generated by the PNGEN block. The resulting signal is the transmitted signal which is the input to the channel simulation.

There are two important points to notice in the simulation of the transmitter. The first point is that the transmitted signal is not modulated to 12.5 KHz as specified in the design of the transmitter (section 3.2). The reason that the transmitted signal is not modulated is because of the corresponding requirement for a higher sampling rate. The higher sampling rates cause the run times for the simulation to be extremely large which is not practical [2,3]. The second point is that there is no STAR

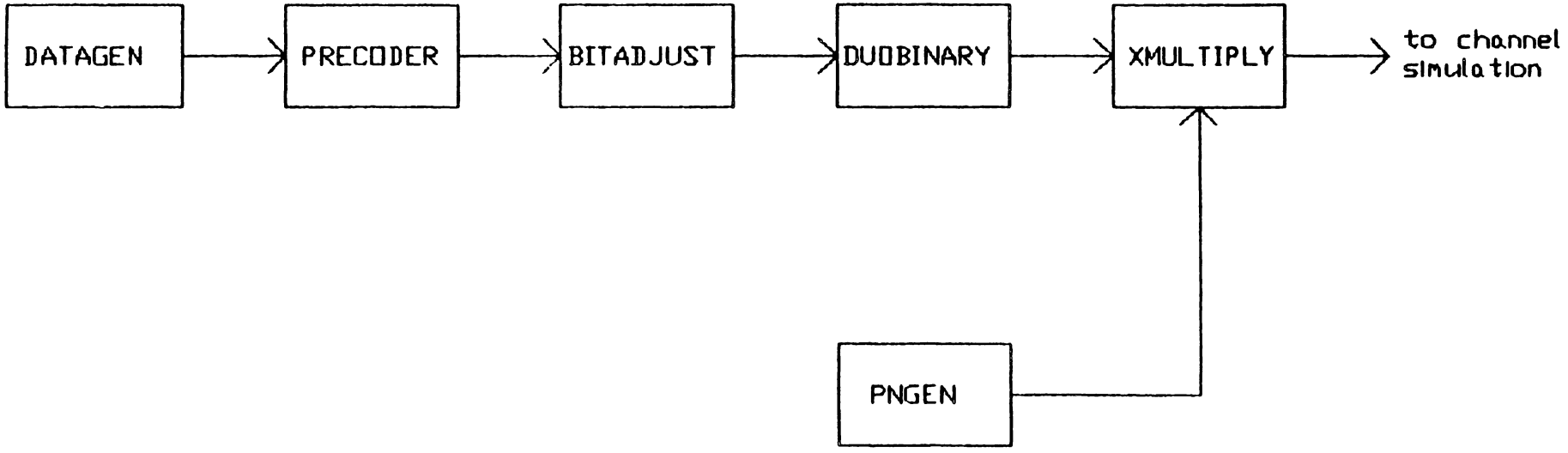


Figure 5.1 - Block Diagram of Transmitter Simulation

to detect the zero crossings. Instead, the period of the PN code is chosen such that it "resets" with frequency 120 Hz which in essence is the same as if zero crossings reset the PN code generator.

The simulation of the channel involves only three blocks. Figure 5.2 shows the block diagram of the channel simulation. The first block, LCONV, convolves the input signal with a specified impulse response. The impulse response that is used can be the response generated by CAPNET [1] which is a program that characterizes the behavior of a distribution line channel given a particular network topology. In this manner degradation and standing wave effects caused by the network topology can be incorporated into the simulation. It is important to note that since the transmitted signal in the simulation is a baseband signal, the channel's frequency response must be downconverted to baseband. The other two blocks simply add noise to the transmitted signal. The ADDNOISE block adds background noise while the ADDHARM block adds harmonic noise. For both blocks, the amount of noise that is added is user specified.

Finally, the block diagram of the receiver simulation is shown in Figure 5.3. The first block, FIRLPF, is a general FIR filter which serves as the wideband front-end filter for the receiver. Next, the signal is despread using the reference PN code generated by the block PNGEN. The signal generated by the block PNGEN can be delayed a

Figure 5.2 - Block Diagram of Channel Simulation

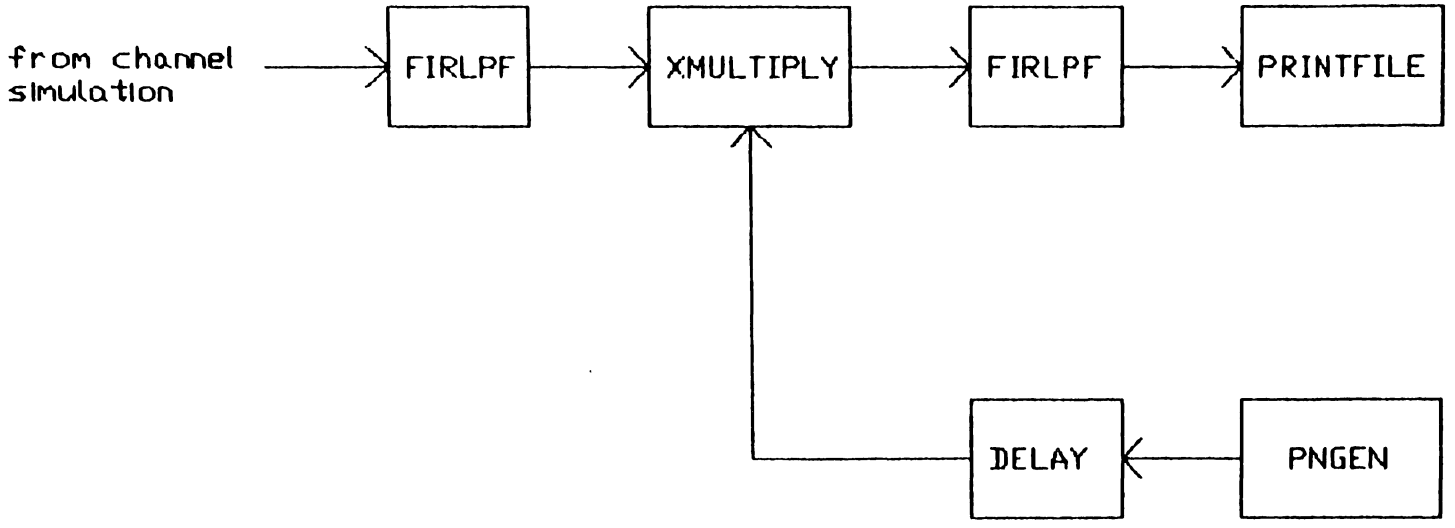


Figure 5.3 - Block Diagram of Receiver Simulation

specified number of samples by the block DELAY. The signal is then filtered by the block FIRLPF which serves as the narrowband filter this time. Lastly, the decoded data sequence is written to a file by the block PRINTFILE.

To test the validity of the overall simulation, plots of the spectrum and of the time waveform were examined at various points in the simulation. Only the spectrums are presented here. (Each block or STAR was also tested individually before inclusion in the overall simulation.) In all the cases, the spectrums that were obtained were as expected. The following is a list of the signal spectrums used and the appropriate Figures.

Figure 5.4 - Power Spectrum of the Modified Duobinary Signal

Figure 5.5 - Power Spectrum of the Transmitted Signal (0 - 15120 Hz)

Figure 5.6 - Power Spectrum of the Transmitted Signal (0 - 360 Hz)

Figure 5.7 - Power Spectrum of the Received Signal

Figure 5.8 - Power Spectrum of the Received Signal After Despreading

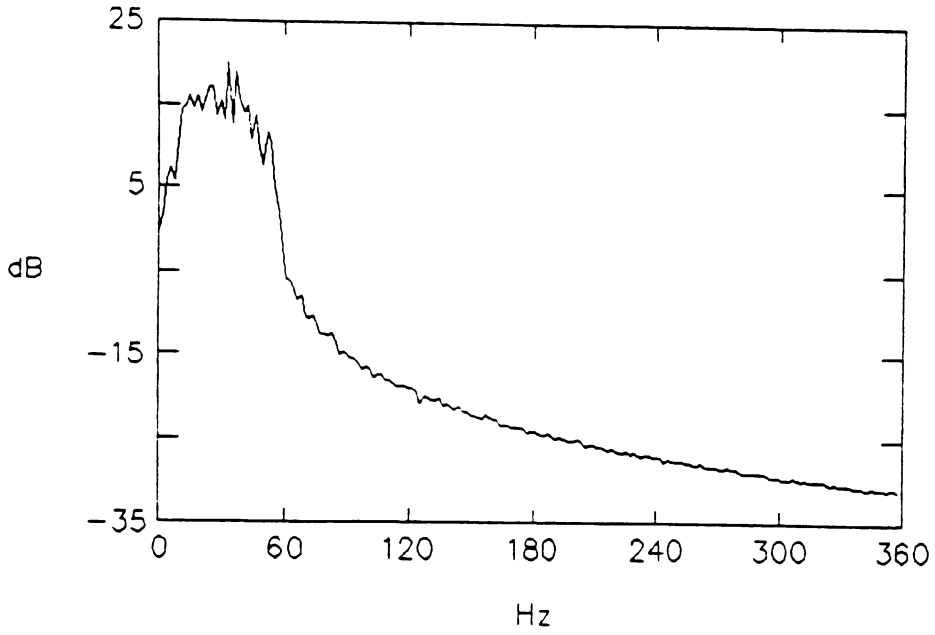


Figure 5.4 - Power Spectrum of the Modified Duobinary Signal

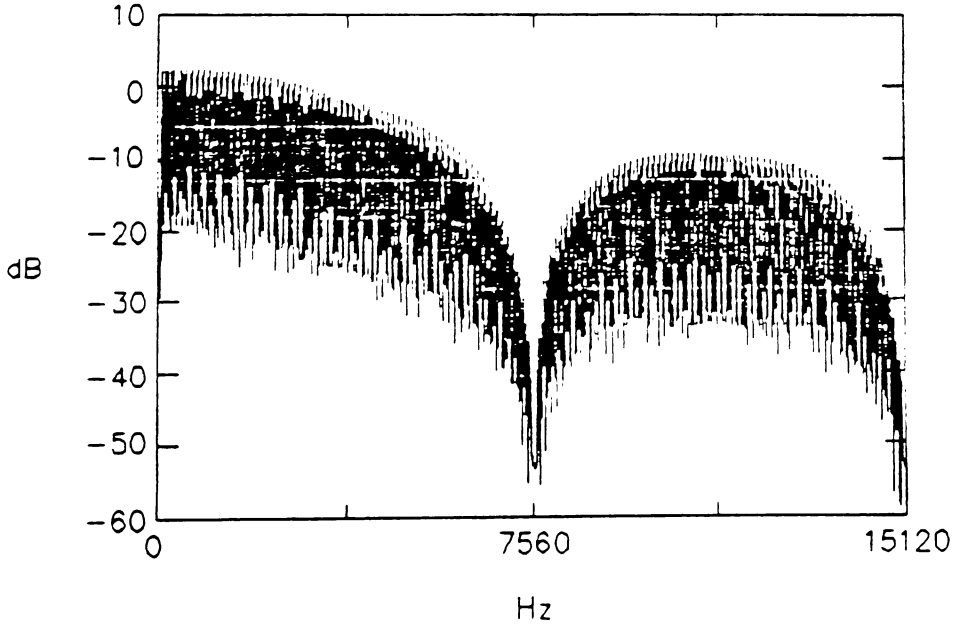


Figure 5.5 - Power Spectrum of the Transmitted Signal (0 - 15120 Hz)

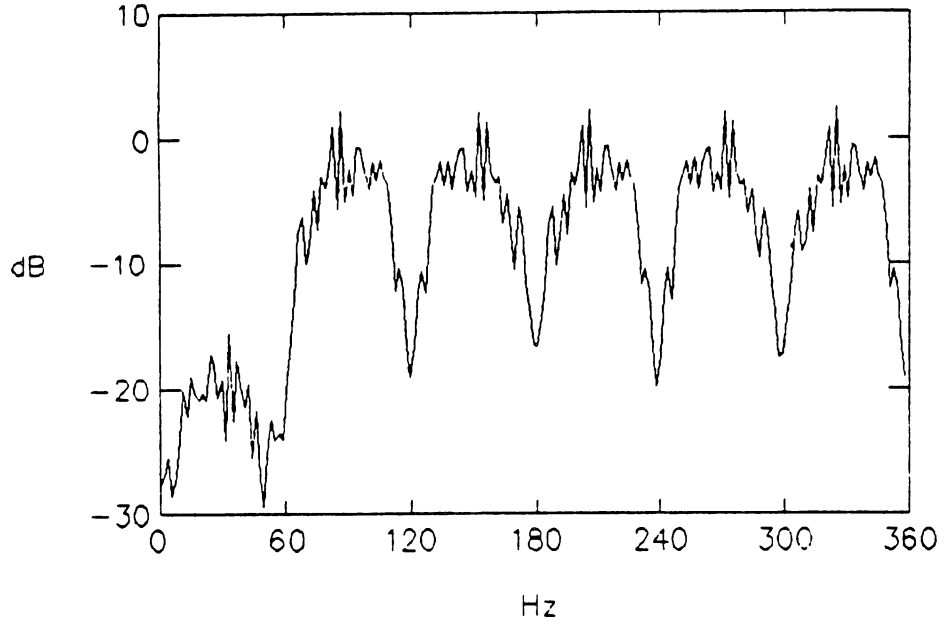


Figure 5.6 - Power Spectrum of the Transmitted Signal (0 - 360 Hz)

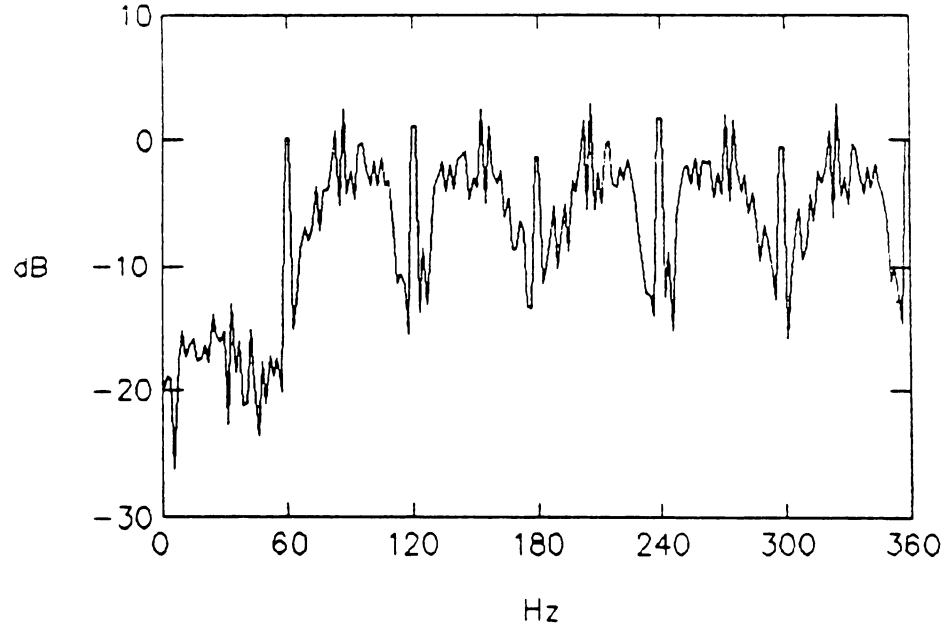


Figure 5.7 - Power Spectrum of the Received Signal

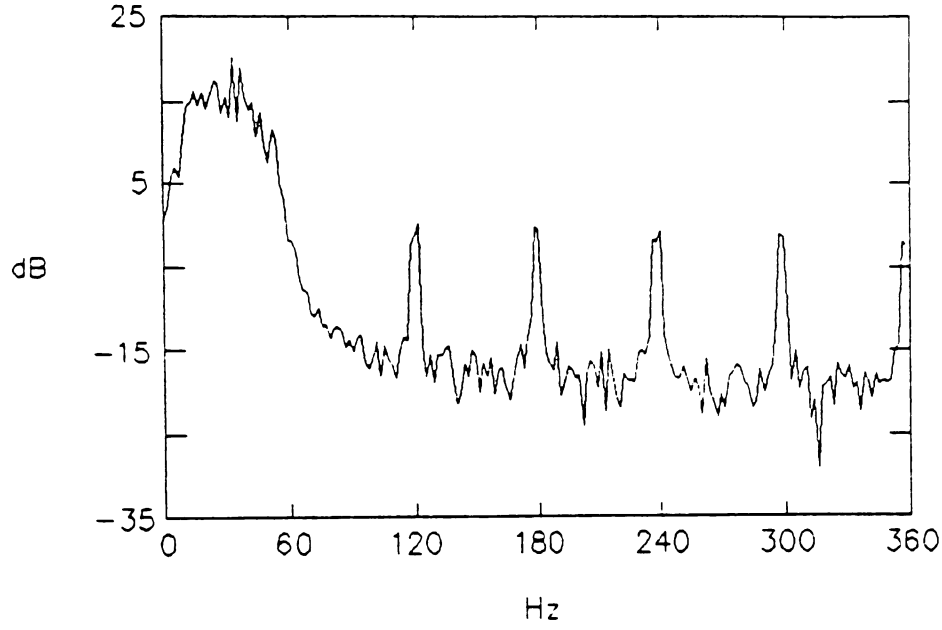


Figure 5.8 - Power Spectrum of the Received Signal After Despreading

5.2 Simulation Results

Three issues were tested using the simulation. They were: 1) the probability of error or the bit error rate of the proposed DLC system, 2) the effect of notches in the channel's spectrum, and 3) the effect of jitter in the detection of the zero crossings of the 60 Hz power signal.

In defining the performance of communication systems, the bit error rate (BER) is commonly used. In simulations, the typical method of obtaining the BER is through Monte Carlo trials; however, an undesirable characteristic of Monte Carlo trials is prohibitively long run times. An alternate method of determining the BER through simulations is by using a quasi-analytical approach [22].

The quasi-analytical approach involves simulating the system in a noiseless environment and then representing the noise analytically. First, simulations of the system are run to generate eye diagrams of the received despread signal. The mean eye opening can then be used to determine how much degradation is caused by elements in the receiver. Next, the degradation is incorporated into the derived expression for the probability of error of the system.

The probability of error of the proposed DLC system is derived in Appendix 8.1 and plots of the BER are shown in Figure 5.9. In the figure, E_b is the energy per bit and N_0 is the single-sided power spectral density of the background noise. Plot A is for the case where only background noise is present and plot B is for the case

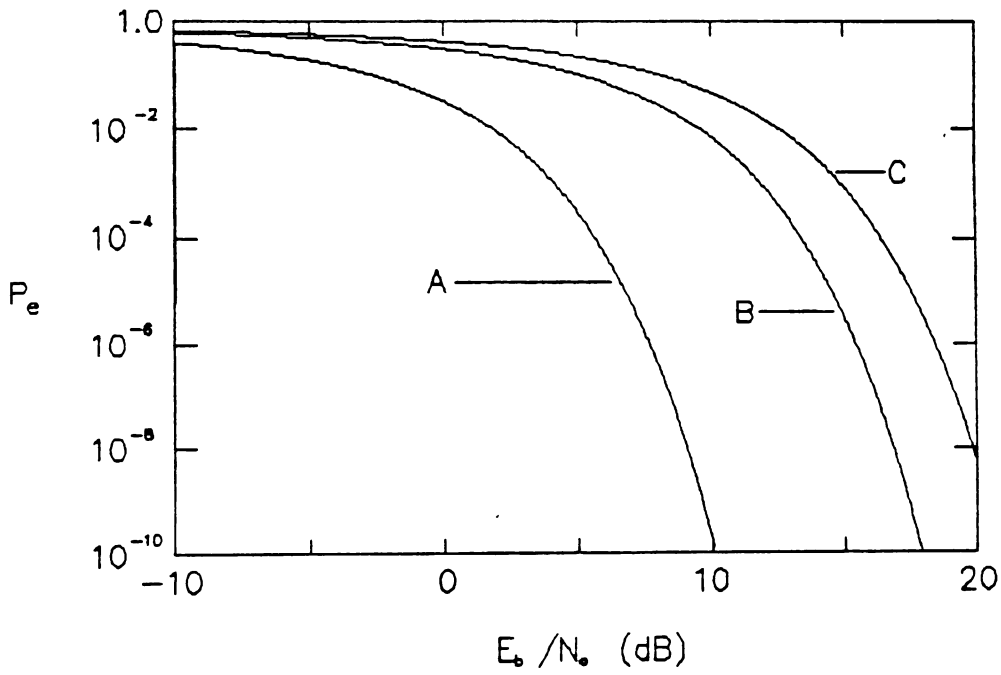


Figure 5.9 - BER Plots

where both background and harmonic noise is present (The power spectrum of the harmonic noise components is assumed to be 20 dB above the background noise and to be 3 Hz wide.). Also plotted in Figure 5.9 is the probability of error of a standard NRZ-L binary signaling scheme in the presence of both background and harmonic noise (Plot C).

As can be seen from Figure 5.9, the BER curve for the proposed DLC system shows improvement over that of the NRZ-L system. Notice though that the plots are for the case of no degradations in the transmitted signal; however, degradations, whether it's caused by the channel or by components in the receiver, will simply result in translation of the plots by the amount of degradation present. For example, if the total degradation is 30 dB, then the plots in Figure 5.9 would begin at 20 dB instead of -10 dB.

A possible source of degradation in the received signal are notches in the spectrum of the channel. As mentioned in section 2, notches are the result of standing wave patterns on the distribution line and can cause fading in the received signal. In narrowband systems, the effects of notches can severely hamper the performance of the system; however, in a spread spectrum system, if the total bandwidth of the notchs is small relative to the bandwidth of the transmitted signal, the energy that is attenuated by the notches is relatively small compared to the transmitted signal's total energy. To test this idea, the simulation

was run with a notch filter present. The spectrum of the received signal is shown in Figure 5.10. The effect of this notch, though, was minimal--the total degradation was less than .1 dB.

Another source of degradation is jitter in the detection of the zero crossing of the 60 Hz power signal. This jitter will cause the phase of the receiver's reference PN code to be misaligned with that of the received signal and, consequently, the received signal will not be properly despread. The jitter will also cause the decision portion of the receiver to sample the received signal at incorrect times.

To find out how much jitter the system can stand, jitter was simulated by inserting constant delays in the receiver's reference PN code. This method is a worst case simulation since the jitter will actually vary from one zero crossing to the next. A plot of the receiver's normalized mean eye opening versus the amount of constant jitter is shown in Figure 5.11. As can be seen, the system is extremely susceptible to jitter in the detection of the zero crossings. A jitter equal to $\frac{1}{2}T_c$ causes the mean eye opening to degrade by a half and jitter greater than or equal to T_c causes the eye opening to close entirely.

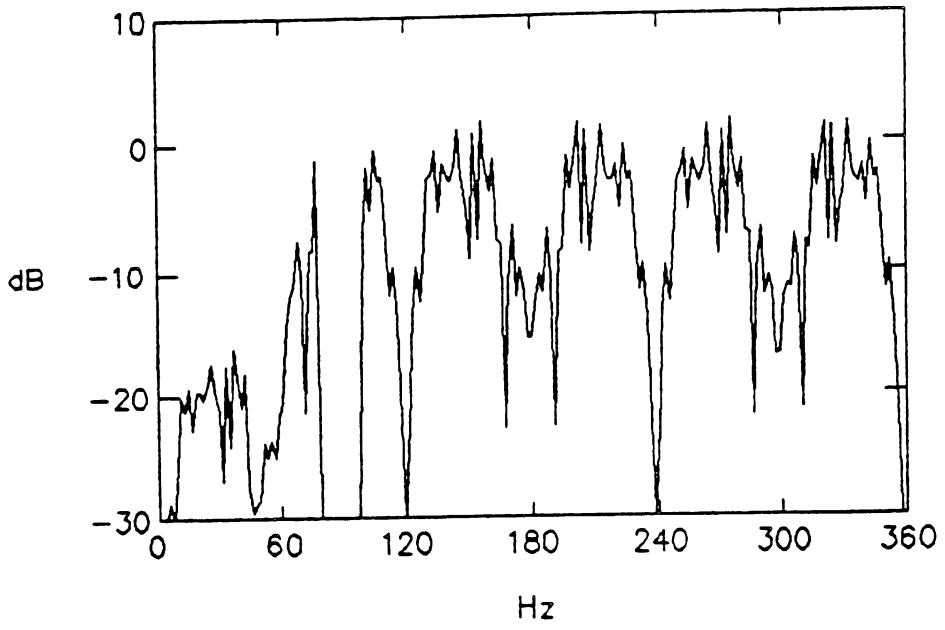


Figure 5.10 - Power Spectrum of the Received Signal With A Notch Filter Present

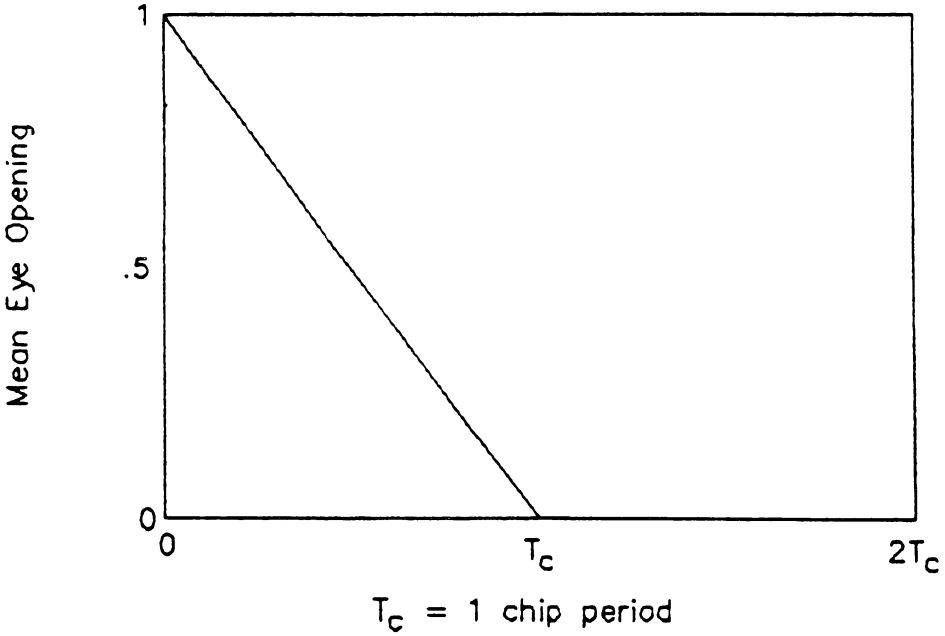


Figure 5.11 - Mean Eye Opening of the Received Signal With Jitter in the Detection of the Zero Crossings

6.0 CONCLUSION

6.1 Conclusions

From the design of the DLC system and the results of the simulation, the following conclusions can be drawn.

1) Using the proposed duobinary/spread spectrum DLC system allows for communication at 120 bps with an improvement in the BER over that of a NRZ-L system operating at 120 bps.
2) The use of spread spectrum in the proposed DLC system provides a measure of immunity to notches in the transmission spectrum caused by standing wave patterns.
3) The use of the zero crossings of the 60 Hz power signal can allow for simple PN code synchronization; however, jitter in the detection of the zero crossings can severely degrade the system's performance.

6.2 Recommendations for Further Research

The following recommendations suggest further research in the application of spread spectrum to power line communications.

1) Because of the proposed DLC system's susceptibility to jitter in the detection of the zero crossing of the 60 Hz power signal, tracking and acquisition schemes could be employed to perform the PN code

synchronization independent of the 60 Hz power signal [5,45,46,60,64,72,74,75].

2) The possibility of using Frequency Hop or a hybrid spread spectrum scheme instead of Direct Sequence should be investigated [20,62,66,70,75].

3) The use of spread spectrum's capability for code division multiple accessing (CDMA) has not been addressed in this report and should be explored further [20,48,49,50,62,73,75].

4) The use of adaptive filters and interference suppression filters could aid the performance of DLC systems [34,51,71].

5) In simulating DLC systems, importance sampling could be used to obtain more accurate estimates of the BER [3,9,18,22].

7.0 REFERENCES

[1] S. Alexander and S. Ardalan, "Computer Modeling and Analysis of Transmission Line Networks," a report of the Center for Communications and Signal Processing Technical, CCSP-TR-87/4, Raleigh, NC, March 1987.

[2] P. Balaban and K.S. Shanmugan, "Computer-Aided Modeling, Analysis and Design of Communication Systems: Introduction and Issue Overview," *IEEE Journal on Selected Areas of Communication*, vol. SAC-2, no.1, January 1984, pp.1-7.

[3] P.Balaban and K.S. Shanmugan, "Modified Monte-Carlo Simulation Techniques for the Evaluation of Error Rate in Digital Communication Systems," *IEEE Trans. Commun.*, vol. COM-28, no.11, November 1980, pp. 1916-1924.

[4] P.O. Brjesson and C.W. Simple, "Approximation of the Error Function Q(x) for Communication Applications," *IEEE Trans. Commun.*, vol. COM-27, March 1979, pp. 639-643.

[5] C.R. Cahn, D.K. Leimer, C.L. Marsh, F.J. Huntowski, and G.D. Larue, "Software Inplementation of a PN Spread Spectrum Receiver to Accommodate Dynamics," *IEEE Trans. Commun.*, vol.COM-25, no.8, August 1977, pp. 832-840.

[6] E.W. Chandler and G.R. Cooper, "Low Probability of Intercept Performance Bounds for Spread-Spectrum Systems," *IEEE Journal on Selected Areas of Communication*, vol. SAC-3, no.5, September 1985, pp. 706-713.

[7] K.N. Clinard, "Utility Experience and Desires for Distribution Power Line Carrier Communications," *IEEE Global Telecommunications Conference*, Nov. 28 - Dec. 1, 1983, pp. 487-491.

[8] L. Couch, "Performance of DS Spread Spectrum Systems," *Proceedings of the IEEE*, vol. 68, no.2, February 1980, pp. 298-300.

[9] B.R. Davis, "An Improved Importance Sampling Method for Digital Communication System Simulations," *IEEE Trans. Commun.*, vol. COM-34, no.7, July 1986, pp. 715-719.

[10] R.C. Dixon, *Spread Spectrum Systems*, New York: Wiley, 1984.

[11] M.D. Eggers and J.H. Painter, "Optimal Symbol-by-Symbol Detection for Duobinary Signaling," *IEEE Trans. Commun.*, vol. COM-31, no.9, September 1983, pp. 1077-1085.

[12] K. Feher, *Digital Communications - Microwave Applications*, Englewood, NJ: Prentice-Hall, 1981.

[13] G.D. Forney, Jr., "Maximum-Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Trans. Inform. Theory*, vol. IT-18, no.3, May 1972, pp.363-378.

[14] F.M. Gardner, "A Transformation for Digital Simulation of Analog Filters," *IEEE Trans. Commun.*, vol. COM-34, no.7, July 1986, pp. 676-680.

[15] E.A. Geraniotis, "Performance of Noncoherent Direct-Sequence Spread-Spectrum Multiple-Access Communications," *IEEE Journal on Selected Areas of Communication*, vol. SAC-3, no.5, Sept. 1985.

[16] E.A. Geraniotis and M.B. Pursley, "Error Probability for Direct-Sequence Spread-Spectrum Multiple-Access Communication- Part II: Approximations," *IEEE Trans. Commun.*, vol. COM-30, May 1982, pp. 985-995.

[17] R. Gold, "Optimal Binary Sequences for Spread Spectrum Multiplexing," *IEEE Trans. Inform. Theory*, vol. IT-13, October 1967, pp. 619-621.

[18] P.M. Hahn and M.C. Jeruchim, "Developements in the Theory and Application of Importance Sampling," *IEEE Trans. Commun.*, vol. COM-35, no.7, July 1987, pp. 706-714.

[19] R.C. Hemminger, "The Effect of Distribution Transformers on Carrier Signal Propagation at Power Line Carrier Frequencies," a report of the Center for Communications and Signal Processing, CCSP-TR-85/9, Raleigh, NC, 1985.

[20] J.K. Holmes, *Coherent Spread Spectrum Systems*, New York: Wiley, 1982.

[21] P.M. Hopkins, "A Unified Analysis of Pseudonoise Synchronization by Envelope Correlation," *IEEE Trans. Commun.*, vol.25, no.8, August 1977, pp. 770-778.

[22] M.C. Jeruchim, "Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communicaton Systems," *IEEE Journal on Selected Areas of Communication*, vol. SAC-2, no.1, January 1984, pp. 153-170.

[23] P. Kabal and S. Pasupathy, "Partial-Response Signaling," *IEEE Trans. Commun.*, vol. COM-23, no.9, September 1975.

[24] M. Kavehrad and P.J. McLane, "Performance of Direct Sequence Spread Spectrum for Indoor Wireless Digital Communication," *IEEE GLOBECOM*, New Orleans, Louisiana, December 2-5, 1985, pp. 974-979.

[25] M. Kavehrad and P.J. McLane, "Spread Spectrum for Indoor Digital Radio," *IEEE Communications Magazine*, vol. 25, no.6, June 1987, pp. 32-40.

[26] N.F. Krasner, "Efficient Search Methods Using Energy Detector- Maximum Probability of Detection," *IEEE Journal on Selected Areas of Communication*,vol. SAC-4, no.2, March 1986, pp. 273- 279.

[27] E.R. Kretzmer, "An Efficient Binary Data Transmission System," *IEEE Trans. Commun.* Syst., vol. CS-12, no.2, June 1964, pp. 250-251.

[28] E.R. Kretzmer, "Generalization of a Technique For Binary Data Communication," *IEEE Trans. Commun. Technol.*, vol. COM-14, no.1, February 1966, pp. 67-68.

[29] H.Kobayashi, "Correlative Level Coding And Maximum-Likelihood Decoding," *IEEE Trans. Inform. Theory*, vol. IT-17, no.5, September 1971, pp. 586-594.

[30] A. Lender, "Correlative Digital Communication Techniques," *IEEE Trans. Commun. Technol.*, vol. COM-12, December 1964, pp. 128-135.

[31] A. Lender, "Correlative Level Coding for Binary-Data Transmission," *IEEE Spectrum*, vol. 3, no.2, February 1966, pp. 104-115.

[32] A. Lender, "The Duobinary Technique for High-Speed Data Transmission," *IEEE Trans. Commun. and Electronics*, vol. 82, May 1963, pp. 214-218.

[33] B.K. Levitt, "Effect of Modulation Format and Jamming Spectrum on Performance of Direct Sequence Spread Spectrum Systems," *IEEE National Telecommunications Conference*, vol. 1, Houston, Texas, 1980, pp. 3.4.1-3.4.5.

[34] E. Masry and L.B. Milstein, "Performance of DS Spread-Spectrum Receiver Employing Interference-Suppression Filters Under a Worst-Case Jamming Condition," *IEEE Trans. Commun.*, vol. COM-34, no.1, January 1986, pp. 13-21.

[35] D.G. Messerschmitt, "A Tool for Structured Functional Simulation," *IEEE Journal on Selected Areas of Communication*, vol. SAC-2, no.1, January 1984, pp. 137-147.

[36] J.B. O'Neal, Jr., "Analysis of Some Synchronous and Non- Synchronous Data Transmission Systems in Harmonic Noise," Raleigh, NC.

[37] J.B. O'Neal, Jr., "Modeling the Noise on the Substation Power Distribution Bus at Frequencies from 1-20 kHz," a report of the Center for Communications and Signal Processing, CCSP-WP-2, Raleigh, NC, 1984.

[38] J.B. O'Neal, Jr., "Overview of Power Distribution Line Carrier Communications Systems," *IEEE Global Telecommunications Conference*, Nov. 28 - Dec. 1, 1983, pp. 464-467.

[39] J.B. O'Neal, Jr., "Substation Noise at Distribution Line Communication Frequencies," a report of the Center for Communications and Signal Processing, CCSP-WP-6, Raleigh, NC, July 1986.

[40] J.B. O'Neal, Jr., "The Transmission Medium, Impairments and Testing of Residential Power Line Carrier Systems," a report for the Electronic Industries Association Consumer Electronics Bus Committee, January 1982.

[41] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.

[42] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, 1984.

[43] P.Z. Peeples, Jr., *Digital Communication Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

[44] R.L. Pickholtz, D.L. Schilling and L.B. Milstein, "Theory of Spread-Spectrum Communications--A Tutorial," *IEEE Trans. Commun.*, vol. COM-30, no.5, May 1982, pp. 855-884.

[45] A. Polydoros and C.L. Weber, "A Unified Approach to Serial Search Spread-Spectrum Code Acquisition--Part I: General Theory, " *IEEE Trans. Commun.*, vol. COM-32, no.5, May 1984, pp. 542-549.

[46] A. Polydoros and C.L. Weber, "A Unified Approach to Serial Search Spread-Spectrum Code Acquisition--Part II: A Matched- Filter Receiver," *IEEE Trans. Commun.*, vol. COM-32, no.5, May 1984, pp. 550-560.

[47] J.G. Proakis, *Digital Communications*, New York: McGraw-Hill, 1983.

[48] M.B. Pursley, "Performance Evaluation for Phase-Coded Spread- Spectrum Multiple-Access Communication--Part I: System Anaylsis," *IEEE Trans. Commun.*, vol. COM-25, no.8, August 1977, pp.795-799.

[49] M.B. Pursley and D.V. Sarwate,"Performance Evaluation for Phase-Coded Spread-Spectrum Multiple-Access Communication-- Part II: Code Sequence Anaylsis," *IEEE Trans. Commun.*, vol. COM-25, no.8, August 1977, pp.800-803.

[50] M.B. Pursley, D.V.Sarwate and W.E. Stark, "Error Probability for Direct-Sequence Spread-Spectrum Multiple-Access Communications--Part I: Upper and Lower Bounds," *IEEE Trans. Commun.*, vol. 30, May 1982, pp. 975-984.

[51] S. Qureshi, "Adaptive Equalization," *IEEE Communications Magazine*, March 1982, pp. 9-16.

[52] L.R.Rabiner and B. Gold, *Theory and Applications of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.

[53] S.S. Rappaport and D.M. Grieco, "Spread-Spectrum Signal Acquisitions: Methods and Technology," *IEEE Communications Magazine*, vol. 22, no.6, June 1984, pp. 6-21.

[54] M.P.Ristenbatt and J.L. Daws, Jr., "Performance Criteria for Spread Spectrum Communications," *IEEE Trans. Commun.*, vol. COM-25, no.8, August 1977, pp. 756-762.

[55] H.E. Rowe, "Bounds on the Number of Signals with Restricted Cross correlation," *IEEE Trans. Commun.*, vol. COM-30, May 982, pp. 966-974.

[56] D.L. Schilling, L.B. Milstein, R.L. Pickholtz, and R.W. Brown, "Optimization of the Processing Gain of an M-ary Direct Sequence Spread Spectrum Communication System," *IEEE Trans. Commun.*, vol COM-28, no.8, August 1980, pp.1389-1398.

[57] R.A. Scholtz, "The Spread Spectrum Concept," *IEEE Trans. Commun.*, vol. COM-25, no.8, August 1977, pp. 748-755.

[58] K.S. Shanmugan, V.S. Frost and P.K. Leong, "Simulation of Spread-Spectrum Systems Using ICSSM," *IEEE Global Telcommunications Conference*, Nov. 28 - Dec. 1, 1983.

[59] D.A. Shnidman, "Evaluation of the Q Function", *IEEE Trans. Commun.*, vol. COM-22, no.3, March 1974, pp. 342-346.

[60] E.W. Seiss and C.L.Weber, "Acquisition of Direct Sequence Signals with Modulation and Jamming," *IEEE Journal on Selected Areas of Communication*, vol. SAC-4, no.2, March 1986, pp. 254-272.

[61] K.C. Shuey, "Distribution Transformers at Power-Line Carrier Frequencies" *IEEE Global Telecommunication Conference*, Nov. 28 - Dec. 1, 1983, pp. 483-486.

[62] M.K.Simon, J.K. Omura, R.A. Scholtz, and B.K. Levitt, *Spread Spectrum Communications*, Vols. I-III, Rockville, Maryland: Computer Science Press, 1985.

[63] R. Singh, "Performance of a Direct Sequence Spread Spectrum System with Long Period and Short Period Code Sequences," *International Conference on Communication*," Denver, Colorado, vol.3, 1981, pp.45.2.1-45.2.5.

[64] J.J. Spilker, Jr., and D.T. Magill, "The Delay-Lock Discriminatior--An Optimum Tracking Device," *Procedures of the I.R.E.*, vol. 49, September 1961, pp. 1403-1416.

[65] T.E. Spotts, "The Measurement and Analysis of High Frequency Noise on Distribution Lines," Master's thesis, North Carolina State University, Raleigh, NC, 1982.

[66] D.J. Torrieri, *Principles of Military Communication Systems*, Dedham, Massachusetts: Artech House, 1982.

[67] G.L. Turin, "Introduction to Spread-Spectrum Anti-Multipath Techniques and Their Application to Urban Digital Radio," *Proceedings of the IEEE*, vol. 68, no.3, March 1980, pp. 328- 353.

[68] P.K. van der Gracht and R.W. Donaldson, "Communication Using Pseudonoise Modulation on Electric Power Distribution Circuits," *IEEE Trans. Commun.*, vol. COM-33, no.9, September 1985, pp. 964-974.

[69] R.J. Vines, "The Characterization of Residential Impedances and Noise Sources for Power Line Carrier Communications," Master's Thesis, North Carolina State University University, Raleigh, NC, 1983.

[70] A.J. Viterbi, "Spread Spectrum Communication -- Myths and Realities," *IEEE Communications Magazine*, vol. 17, no.3, May 1979, pp. 11-18.

[71] J.D. Wang, "Analysis of Adaptive Filter Algorithms with an Application to Harmonic Noise Cancellation in Distribution Power Line Communications," a report of the Center for Communications and Signal Processing, CCSP-TR-85/15, Raleigh, NC, 1985.

[72] R.B. Ward, "Acquisiton of Pseudonoise Signals by Sequential Estimation," *IEEE Trans. Commun.*, vol. COM-13, no.4, December 1965, pp. 475-483.

[73] K. Yao, "Error Probability of Asynchronous Spread Spectrum Multiple Access Communication Systems," *IEEE Trans. Commun.*, vol. COM-25, no.8, August 1977, pp. 803-809.

[74] R.A. Yost and R.W. Boyd, "A Modified PN Code Tracking Loop: Its Performance Anaylsis and Comparative Evaluation," *IEEE Trans. Commun.*, vol. COM-30, no.5, May 1982, pp. 1027-1036.

[75] R.E. Ziemer and R.L. Peterson, *Digital Communications and Spread Spectrum Systems*, New York: Macmillan, 1985.

8.0 APPENDICES

8.1 Probability of Error for the Proposed DLC System

The probability of error for the proposed DLC system is found by considering the noise characteristic at the sampler and decision blocks in Figure 4.1. If ideal filtering is assumed and the received signal is properly despread, the input signal to the sampler is a modified duobinary signal and the decision process described in section 4.2 applies. For the case of additive white gaussian noise (AWGN) [32,43,75], the probability of error is then given by:

$$P_e = \frac{3}{2} Q(\frac{A}{\sigma}) \qquad (8.1)$$

where A is the amplitude of the modified duobinary signal, σ is the standard deviation of the noise and $Q(\cdot)$ is the standard Q-function [4,41,59]. For the proposed DLC system, the noise at the sampler consists of flat background noise and harmonic noise. From [39] it was found that the noise could be reasonably approximated by a gaussian density function; therefore, equation (8.1) can be used.

To evaluate equation (8.1), the total noise power, σ^2, is needed. Before the noise power can be found though, the flat power spectral densities (PSD) of the background and harmonic noise must be found. From section 2, the power

spectrum of the harmonic noise components is typically 20 dBs above that of the background noise and 3 Hz wide. With these observations and the fact that the harmonic noise components occur every 60 Hz, the flat single-sided PSD of the harmonic noise, N_H, is found to be

$$N_H = 3 \cdot N_o(10^{20/10}) \; / \; 60 = 5N_o$$

where N_o is the flat single-sided PSD of the background noise.

The total noise power, σ^2, can then be found by

$$\sigma^2 = 2 \left(\frac{N_o + N_H}{2}\right) B_n = 6N_oB_n \qquad (8.2)$$

where B_n is the equivalent noise bandwidth of the narrowband filter in Figure 4.1.

The probability of error is frequently expressed in terms of E_b/N_o where E_b is the energy per bit. The energy per bit is found by dividing the average signal power, S_T, by the bit rate R. In [30,31,32] the average signal power is found to be $A^2/2$ and so E_b can be written as

$$E_b = \frac{S_T}{R} = \frac{A^2}{2R} \qquad (8.3)$$

Finally, using (8.1) and (8.2) the probability of error can be written as

$$P_e = \frac{3}{2} Q\left[\sqrt{\frac{E_b R}{3 N_o B_n}} \right] \qquad (8.4)$$

8.2 BLOSIM Stars

The following pages list the source code for all the BLOSIM stars developed to simulate the proposed DLC system. The following references on digital signal processing were consulted extensively in constructing the STARs and the simulations: [14,41,52].

```
/* addharm.s */
/*************************************************************************

                              addharm()

*************************************************************************

This star adds harmonic noise to the input signal.

Programmer:  J.D. Sun
Date:  6/10/87
*/

#include <math.h>
#define  FFTEXP      14
#define  FFTSIZE     16384

parameters

     float srate;     /* sampling rate */
     float fundamen; /* fundamental frequency */
     float mean;      /* mean value of the fundamental in volts */
     float range;     /* +- minus range of uniform variation of mean */
     int seed;        /* seed for the random number generator */
end

input_buffer

     float x;
end

output_buffer

     float y;
end

states

     float* temp;    /* array for the fft */
     float norfun;   /* normalized fundamental frequency */
     int numharm;    /* number of harmonics */
     int numsamp;    /* number of unprocessed samples received */
end
```

```
declarations

     char *calloc();
     int i;
     float resolutn;     /* frequency resolution of FFT */
     float newmag;
     float mag;
     int nn;
     int nn1;
     float z;


end

initialization_code

     if ((temp = (float*)calloc(FFTSIZE,4)) == NULL) {
          fprintf(stderr,"addharm:  can't allocate work space\n");
          return(1);
     }
     numsamp = 0;
     resolutn = srate / FFTSIZE;
     norfun = fundamen / resolutn;
     numharm = (int) (srate / (2 * fundamen));
     srand(seed);
end

main_code

     while(it_in(0)) {
          temp[numsamp] = x(0);
          ++numsamp;
          if (numsamp == FFTSIZE) {
               lrfft(temp,FFTSIZE);
               for(i = 1; i < numharm; i++) {
                    z = ((rand()/2147483647.) - .5) * range;
                    nn = 2 * ((int) (i*norfun));
                    nn1 = nn + 1;
                    mag = sqrt(temp[nn]*temp[nn]+temp[nn1]*temp[nn1]);
                    newmag = ((mean + z) * FFTSIZE)/2.0;
                    if (mag < newmag) {
                         temp[nn] = newmag;
                         temp[nn1] = 0.0;
                    }
```

```
                    nn = nn + 2;
                    nn1 = nn + 1;
                    mag = sqrt(temp[nn]*temp[nn]+temp[nn1]*temp[nn1]);
                    if (mag < newmag ) {
                         temp[nn] = newmag;
                         temp[nn1] = 0.0;
                    }
               }
               lrfti(temp,FFTSIZE);
               for (i = 0; i < FFTSIZE; i++) {
                    it_out(0);
                    y(0) = temp[i];
               }
               numsamp = 0;
          }
          return(0);
     }
end

wrapup_code
        free((char*)temp);
end
```

```
/* addnoise.s */
/*************************************************************************

                        addnoise()

*************************************************************************

This star adds a gaussian random variable to the input signal.

              :input float x
              :output float y = x + random number

The mean and variance are parameters.

*/

#include <math.h>

parameters

     float mean;
     float variance;
     int seed;
end

input_buffer

     float x;
end

output_buffer

     float y;
end

declarations

     int i;
     float z;
end

initialization_code

     srand(seed);
```

```
end

main_code

    while(it_in(0)) {
         it_out(0);

         /* calculate a standard normal random number */
         z = 0;
         for (i=0;i<12;i++) z = z + rand();
         z = (z/2147483647. - 6.);

         /* transform to desired gaussain random number */
         z = sqrt(variance) * z + mean;

         /* compute the output value */
         y(0) =  x(0) + z;
    }
    return(0);
end
```

```
/* bitadjust.s */
/*****************************************************************************

                              bitadjust()

*****************************************************************************

This star takes an input bit stream and inserts a number of zeroes
between the bits.  The output is a +1, -1, or 0.  (In essence this
star generates an impulse train.

Programmer:  J.D. Sun
Date:  6/2/87
*/

input_buffer

     int x;
end

output_buffer

     float y;
end

parameters

     int nspb;  /* number of samples per bit */
end

declarations

     int i;
end

main_code

     while (it_in(0)) {
          it_out(0);

          /* output the bit */
          if (x(0) == 1)
               y(0) = 1.0;
          else
```

```
                y(0) = -1.0;

            /* send the zeroes */
            for (i = 0; i < nspb-1; ++i) {
                it_out(0);
                y(0) = 0.0;
            }
        }
end
```

```
/* datagen.s */
/*****************************************************************************

                              datagen()

*****************************************************************************

This star generates an integer pseudo random binary sequence using a
shift register generator.
The pseudo-random sequence generator uses the polynomial x**10+x**3+1.

Programmer:  J. D. Sun
Date:  5/19/87
*/

parameters

     int length;           /* number of bits to generate */
     int initialize = 12;  /* initialization of shift register */
end

output_buffer

     int y;
end

states

     int shift_reg = initialize;
     int samples_out = 0;
end

declarations

     int i;
end

main_code

  for(i = 0; i < NOSAMPLES; ++i) {

     if (samples_out < length) {
          /* run the shift register sequence to obtain one more bit */
          shift_reg = ((((shift_reg >> 10)&1) + ((shift_reg >> 3)&1)
```

```
                +1 ) % 2)+(shift_reg << 1);
            it_out(0);
            y(0) = shift_reg & 1;
            ++samples_out;
        }
    }
    return(0);
end
```

```
/* duobinary.s */
/*****************************************************************************

                              duobinary()

*****************************************************************************

This star is a filter with a modified duobinary impulse response.
See Proakis, Digitial Communications for further details.
There are 2 parameters. The first parameter, "smplbd", is the sampling
rate normalized with respect to the baud rate.  The second paramter is
the power of two FFT size (FFT length = 2^fftexp).  The output is
generated by convolving the input sequence with the impulse response.
Convolution is performed by the overlap-save method (described in
Oppenheim & Schafer, Digital Signal Processing, pp. 113).

*/

#include <math.h>
#define  PI             3.141592654
#define  IMPBAUD        12
#define  MAXFFTEXP      14

parameters

     int smplbd;
     int fftexp;
end

states

     float* temp;
     float* iresp;
     float* save;
     int pcount;
     int impl;
end

input_buffer

     float x;
end

output_buffer
```

```
     float y;
end

declarations

     char *calloc();
     int i;
     int fftl = 1 << fftexp;
     float t;
     float ts;
end

initialization_code

     /* determine normalized sampling period */
     ts = 1.0/((float) smplbd);

     /* check if fftexp too big */
     if (fftexp > MAXFFTEXP) {
          fprintf(stderr,"duobinary:  FFT size too big\n");
          return(1);
     }

     /* find impulse response length and compare with fft size */
     impl = IMPBAUD * smplbd;
     if(impl >= fftl) {
          fprintf(stderr,"duobinary:  FFT size too small\n");
          return(1);
     }

     /* allocate memory for arrays */
     if(((temp = (float*)calloc(fftl,4)) == NULL) ||
         ((iresp = (float*)calloc(fftl,4)) == NULL) ||
         ((save = (float*)calloc(impl,4)) == NULL)) {
            fprintf(stderr,"duobinary:  can't allocate work space\n");
            return(1);
     }

     /* generate the modified duobinary impulse response */
     t = (-1.0 * IMPBAUD)/2.0;
     for (i = 0; i <= impl; ++i) {
       if (t - 1 == 0.0)
         iresp[i] = -1.0;
```

```
        else if (t + 1 == 0.0)
          iresp[i] = 1.0;
        else
          iresp[i] = sin(PI*(t+1))/(PI*(t+1))-sin(PI*(t-1))/(PI*(t-1));
        t = ts + t;
        temp[i] = 0.0;
      }

      for (i = impl+1; i < fft1; ++i)
           iresp[i] = 0.0;

      /* convert to frequency domain using lrfft */
      lrfft(iresp, fft1);

      pcount = impl;
end

main_code

        while(it_in(0)) {
                temp[pcount++] = x(0);
                if(pcount == fft1) {
                        for(i=0; i<impl; i++)
                                save[i] = temp[fft1-impl+i];
                        lrfft(temp,fft1);
                        cmultfft(temp, iresp, fft1);
                        lrfti(temp,fft1);
                        for(i=impl; i<fft1; i++){
                                it_out(0);
                                y(0) = temp[i];
                        }
                        pcount = impl;
                        for(i=0; i<impl; i++)
                                temp[i] = save[i];
                }
        }
end

wrapup_code

        free((char*)temp);
        free((char*)iresp);
        free((char*)save);
end
```

```
/* duobindec.s */
/*************************************************************************

                              duobindec()

*************************************************************************

This star is the decision block of a modified duobinary signaling
scheme.  It assumes that the orignal data sequence has been precoded
using the star precoder.s.  See Proakis, DIGITAL COMMUNICATIONS, for
more details.

Programmer:  J.D. Sun
Date:  6/14/87
*/

parameters

     float thresh;     /* decision threshold */
end

input_buffer

     float x;
end

output_buffer

     int y;
end

states

     float lower;
end

initialization_code

     lower = -1.0 * thresh;
end

main_code

     while (it_in(0)) {
```

```
          it_out(0);

          if ((x(0) >= thresh) || (x(0) <= lower))
               y(0) = 1;
          else
               y(0) = 0;

          return(0);
     }
end
```

```
/* firlpf.s */
/*************************************************************************

                              firlpf()

*************************************************************************

This star simulates a FIR low pass filter using frequency sampling.
See Jackson, DIGITAL FILTERS AND SIGNAL PROCESSING, pp. 134-139, for
more details on the algorithm used in this star.

The impulse response is generated from the desired frequency response
and a difference equation is generated.

Programmer:  J.D. Sun
Date:  8/10/87
*/

#include <math.h>
#define    PI       3.141592654
#define    PI2      6.283185308

parameters

        int nfs;       /* number of frequency samples from 0 to fs/2 */
        int passband;  /* passband edge sample number */
end

states

        float* mresp;
        float* A;
        float* inseq;  /* input sequence */
        float* iresp;  /* impulse response */
        int impl;      /* impulse response length, equal to 2 * nfs */
end

input_buffer

        float x;
end

output_buffer
```

```
        float y;
end

declarations

        char *calloc();
        float angle;
        float output;
        float signofak;
        int i;
        int k;
end

initialization_code

        impl = 2 * nfs;
        if(((inseq = (float*)calloc(impl,4)) == NULL) ||
                ((A = (float*)calloc(nfs,4)) == NULL) ||
                ((mresp = (float*)calloc(nfs,4)) == NULL) ||
                ((iresp = (float*)calloc(impl,4)) == NULL)) {
                fprintf(stderr,"firlpf:  can't allocate work space\n");
                return(1);
        }

/*      generate the magnitude response of the filter */
        for (i = 0; i < nfs; i++)
                if (i <= passband)
                        mresp[i] = 1.0;
                else
                        mresp[i] = 0.0;

/*      insert a transition sample */
        mresp[passband+1] = .38;

/*      calculate the A[k] coefficients */
        signofak = -1;
        for (k = 0; k < nfs; k++) {
                signofak = -1 * signofak;
                A[k] = signofak * mresp[k]/impl;
        }

/*      generate the impulse response */
        for(i = 0; i < impl; i++) {
                output = 0.0;
```

```
                for (k = 1; k < nfs; k++) {
                        angle = PI2*((float) k)*(((float) i)+.5)/impl;
                        output = output + (2.0 * A[k] * cos(angle));
                }
                iresp[i] = output + A[0];
                inseq[i] = 0.0;
        }

        free((char*)A);
        free((char*)mresp);

end

main_code

        while(it_in(0)) {
                it_out(0);

/*              shift the sample into the input sequence array */
                for (i = impl-1; i >= 1; i--)
                        inseq[i] = inseq[i-1];
                inseq[0] = x(0);

/*              calculate the output of the difference equation */
                output = 0.0;
                for (i = 0; i < impl; i++)
                        output = output + inseq[i]*iresp[i];
                y(0) = output;

        }
end

wrapup_code

        free((char*)inseq);
        free((char*)iresp);
end
```

```
/* gain.s */
/*****************************************************************************

                              gain()

*****************************************************************************

This star acts as an ideal amplifier.  The only parameter is the gain.

Programmer:  J.D. Sun
Date:  6/8/87
*/

parameters

     float g;     /* the gain */
end

input_buffer

     float x;
end

output_buffer

     float y;
end

main_code

     while (it_in(0)) {
          it_out(0);

          /* output is the input multiplied by the gain */
          y(0) = g * x(0);
     }
     return(0);
end
```

```
/* int_fork.s */
/*****************************************************************************

                              int_fork()

*****************************************************************************

This star forks an integer signal into two integer signals.

Programmer:  J.D. Sun
Date:  6/14/87
*/

input_buffers

     int x;
end

output_buffers

     int y0;
     int y1;
end

main_code

     while (it_in(0)) {
          it_out(0);
          it_out(1);
          y0(0) = x(0);
          y1(0) = x(0);
     }
     return(0);
end
```

```
/* int_print.s */
/*****************************************************************************

                              int_print()

*****************************************************************************

Function prints samples from a single integer input to a file, the
name of which is passed as a parameter.  The file, when not specified,
defaults to standard output

Programmer:  J.D. Sun
Date:  6/2/87
*/

parameters

     file file_name = "stdout";
end

input_buffer

     int x;
end

states

        FILE* fp;
end

declarations

        FILE *fopen(); /* fopen opens file and retrieves pointer fp */
end

initialization_code

        if(strcmp(file_name,"stdout") == 0)
                fp = stdout;
        else
                if((fp = fopen(file_name,"w")) == NULL) {
                    fprintf(stderr,"int_print:  can't open file for write\n");
                    return(1);
                }
```

```
end

main_code

     while(it_in(0)) {
          fprintf(fp,"%d\n", x(0));
     }
     return(0);
end

wrapup_code

        if (fp != stdout)
             fclose(fp);
        return(0);
end
```

```
/* int_printx.s */
/*****************************************************************************

                              int_printx()

*****************************************************************************

Function prints samples from a single integer input to a file, the
name of which is passed as a parameter.  The file, when not specified,
defaults to standard output.  The input is also passed without change
to the output buffer.

Programmer:  J.D. Sun
Date:  6/2/87
*/

parameters

     file file_name = "stdout";
end

input_buffer

     int x;
end

output_buffer

     int y;
end

states

        FILE* fp;
end

declarations

        FILE *fopen(); /* fopen opens file and retrieves pointer fp */
end

initialization_code

        if(strcmp(file_name,"stdout") == 0)
```

```
                fp = stdout;
        else
                if((fp = fopen(file_name,"w")) == NULL) {
            fprintf(stderr,"int_print:  can't open file for write\n");
            return(1);
          }
end

main_code

     while(it_in(0)) {
          it_out(0);
          y(0) = x(0);
          fprintf(fp,"%d\n", x(0));
     }
     return(0);
end

wrapup_code

        if (fp != stdout)
             fclose(fp);
        return(0);
end
```

```
/* notch.s */
/*****************************************************************************

                              notch()

*****************************************************************************

This star notches out part of the input spectrum.

Programmer:  J.D. Sun
Date:  7/1/87
*/

#include <math.h>
#define  FFTEXP       14
#define  FFTSIZE      16384
#define  ATTENU       .0001

parameters

     float srate;    /* sampling rate */
     float cfreq;    /* notch center frequency */
     int band;       /* BW of notch in normalized freqeuncy units */
end

input_buffer

     float x;
end

output_buffer

     float y;
end

states

     float* temp;      /* array for the fft */
     int norcfreq;     /* normalized notch center frequency */
     int numsamp;
end

declarations
```

```
     char *calloc();
     int i;
     int nn;
     float resolutn;      /* frequency resolution of FFT */
     float mag;
end

initialization_code

     if ((temp = (float*)calloc(FFTSIZE,4)) == NULL) {
          fprintf(stderr,"notch:  can't allocate work space\n");
          return(1);
     }

     resolutn = srate / FFTSIZE;
     norcfreq = (int) cfreq/resolutn;
     norcfreq = 2 * norcfreq;
end

main_code

     while(it_in(0)) {
          temp[numsamp] = x(0);
          ++numsamp;
          if (numsamp == FFTSIZE) {
               lrfft(temp,FFTSIZE);

               /* notch out the specified frequency region */
               nn = 2 * (band/2);
               mag = 100.0 * ATTENU * temp[norcfreq + nn + 2];
               temp[norcfreq + nn + 2] = mag;
               mag = 100.0 * ATTENU * temp[norcfreq + nn + 1];
               temp[norcfreq + nn + 1] = mag;
               for (i = -1*nn; i <= nn+1; i++) {
                    mag = ATTENU * temp[norcfreq + i];
                    temp[norcfreq + i] = mag;
               }
               mag = 100.0 * ATTENU * temp[norcfreq - nn - 1];
               temp[norcfreq - nn - 1];
               mag = 100.0 * ATTENU * temp[norcfreq - nn - 2];
               temp[norcfreq - nn - 2];

               /* back to time domain */
               lrfti(temp,FFTSIZE);
```

```
                for (i = 0; i < FFTSIZE; i++) {
                     it_out(0);
                     y(0) = temp[i];
                }
                numsamp = 0;
           }
           return(0);
     }
end

wrapup_code

        free((char*)temp);
end
```

```
/* pngen.s */
/*************************************************************************

                        pngen()

*************************************************************************
This star generates the current output bit of a PN code generator.

Programmer:  J.D. Sun
Date:  12/19/86
*/

output_buffer

     float y;
end

parameters

     int length;       /* number of samples to send */
     int nspc;         /* number of samples per chip */
     int numstages;  /* number of stages in the SRG */
     int cp;         /* generator code polynomial (in octal) */
     int delay;      /* number of samples to delay before beginning */
end

state

     float shiftreg[20]; /* shift register array */
     int taploc[20];     /* tap locations array */
     float currentbit;
     int ns;             /* number of samples sent per chip */
     int samples_out;    /* number of total samples sent */
end

declarations

     int i;
     int sum;
     int taps;
     int codepoly;
     char buffer[20];
end
```

```
initialization_code

      /* change the input code polynomial to octal */
      sprintf(buffer,"%d", cp);
      sscanf(buffer,"%o", &codepoly);

      /* set up the taps on the SRG */
      taps = codepoly;
      for (i = 0 ; i <= numstages - 1 ; ++i) {
           taps = taps >> 1;
           taploc[i] = taps & 01;
           shiftreg[i] = 1.0;
      }
      shiftreg[numstages - 1] = 0.0;

      currentbit = shiftreg[numstages - 1];
      ns = 0;
      samples_out = 0;
end

main_code

     ++samples_out;
     if (samples_out <= length) {
          it_out(0);
          ++ns;
          /* if ns has exceeded the delay, then begin */
          if (ns > delay) {
               if (delay >= 0)
                    ns = 1;
               delay = -9999;

               /* test if need new bit */
               if (ns >= nspc) {

                    /* find new output bit */
                    /* find the result of the feedback connections */
                    sum = 0 ;
                    for (i = 0 ; i <= numstages - 1 ; ++i)
                         if (taploc[i] == 1)
                              sum = sum + shiftreg[i] ;
                    sum = sum & 01 ;

                    /* then, clock the registers */
```

```
                    for (i = (numstages) - 1 ; i >= 1 ; --i)
                         shiftreg[i] = shiftreg[i-1] ;
                    shiftreg[0] = sum ;

                    /* outbit bit is contents of last shift register */
                    currentbit = shiftreg[(numstages) - 1] ;

                    /* reset ns */
                    ns = 0;
               }

               /* output is the current bit */
               if (currentbit == 1.0)
                    y(0) = currentbit;
               else
                    y(0) = -1.0;
          }
          else {
               y(0) = 0.0;
          }
     }
     return(0);
end
```

```
/* power_spec.s */
/*****************************************************************************

                              power_spec()

*****************************************************************************

This star estimates the power spectrum of a signal using Bartlett's
method.

Programmer:  J.D. Sun (this star is a modification of star:  fft.s)
Date: December 20, 1986

*/

#include <math.h>
#define MAXFFTEXP     14

parameters

     int fftexp;      /* power of two FFT size */
     file filename;   /* file to which the power spectrum is written */
end

states

     int fftl;          /* fft length,  16K maximum */
     float* temp;       /* array for input samples and fft output */
     float* pwrest;     /* array for power spectrum estimate */
     int pcount = 0;    /* current number of samples received */
     int fftlo2;
     int blocknum = 0; /* blocknum + 1 = current block number */
     FILE* fp;
end

input_buffer

     float x;
end

output_buffer

     float yl;     /* the input sample passed without change */
end
```

```
declarations

     char *calloc();
     float mag;
     int i,k,bpl;
end

initialization_code

     /* calculate FFT length */
     if (fftexp > MAXFFTEXP) {
          fprintf(stderr,"power_spec:  FFT size too big\n");
          return(1);
     }
     fft1 = 1 << fftexp;

     if(((temp = (float*)calloc(fft1,4)) == NULL)){
          fprintf(stderr,"power_spec:  can't allocate work space\n");
          return(1);
     }
     fft1o2 = fft1 / 2;

     if(((pwrest = (float*)calloc(fft1,4)) == NULL)){
          fprintf(stderr,"power_spec:  can't allocate work space\n");
          return(1);
     }

     if(strcmp(filename,"stdout") == 0) {
          fp = stdout;
     }
     else {
          if((fp = fopen(filename,"w")) == NULL) {
            fprintf(stderr,"power_spec:  can't open file open\n ");
            return(1);
          }
     }
end

main_code

   while(it_in(0)){

      temp[pcount] = x(0);
```

```
        it_out(0);
        y1(0) = temp[pcount];
        pcount++;

        /* if have enough samples, perform fft  */
        if(pcount == fft1){
           lrfft(temp,fft1);

           /* perform spectrum estimation */
           bp1 = blocknum + 1;

           /* DC term */
           mag = sqrt(temp[0]*temp[0])/fft1;
           pwrest[0] = (pwrest[0] * blocknum + mag*mag) /((float) bp1);

           /* Nyquist frequency term */
           mag = sqrt(temp[1]*temp[1])/fft1;
           pwrest[fft1o2-1] = (pwrest[fft1o2-1]*blocknum
                               + mag*mag)/((float) bp1);

           for (i = 1; i <= fft1o2-1; i++) {
                k = 2 * i;
                mag = sqrt(temp[k]*temp[k] + temp[k+1]*temp[k+1])/fft1;
                pwrest[i] = (pwrest[i] * blocknum + mag*mag)
                            / ((float) bp1) ;
           }

           pcount = 0;
           blocknum = blocknum + 1;
           }
        }
end

wrapup_code

   /* write power spectrum estimate to filename */
   for (i = 0; i < fft1o2; i++)
        fprintf(fp,"%e\n", pwrest[i]);
   if (fp != stdout)
        fclose(fp);

   free((char*)temp);
   free((char*)pwrest);
   return(0);
```

```
/* precoder.s */
/*************************************************************************

                                precoder()

*************************************************************************

This star precodes an input bit stream so that when using modified
duobinary transmission, the errors do not propagate.  See Proakis,
DIGITAL COMMUNICATIONS, for more details.  The new bit stream p is
found from the input bit stream d by:  p[n] = d[n] xor p[n-2].

Programmer:  J.D. Sun
Date:  6/12/87
*/

input_buffer

     int d;
end

output_buffer

     int p;
end

states

     int pn1;
     int pn2;
end

declarations

     int temp;
end

initialization_code

     pn1 = 0;
     pn2 = 0;
end

main_code
```

```
        while (it_in(0)) {
                it_out(0);

                temp = d(0) ^ pn2;
                p(0) = temp && 01;
                pn2 = pn1;
                pn1 = p(0);
        }
        return(0);
end
```

```
/* sample.s */
/************************************************************************

                            sample()

************************************************************************

This star samples a discrete time signal at specified periodic
intervals.  It begins sampling after a specified delay.  This star
is useful for sampling the output of a matched filter.

Programmer:  J.D. Sun
Date:  6/12/87
*/

parameters

     int nspp;     /* number of sampler per period */
     int delay;    /* number of samples to delay before starting */
end

input_buffer

     float x;
end

output_buffer

     float y;
end

states

     int flg;
     int cnt;
end

initialization_code

     flg = 0;
     cnt = 0;
end

main_code
```

```
      while (it_in(0)) {
           ++cnt;

           if (flg == 0) {
                if (cnt == delay) {
                     flg = 777;
                     cnt = 0;
                }
                return(0);
           }

           if (cnt == 1) {
                it_out(0);
                y(0) = x(0);
                return(0);
           }
           if (cnt == nspp) {
                cnt = 0;
                return(0);
           }
           return(0);
      }
end
```

```
/* xmultiply.s */
/*************************************************************************

                        xmultiply()

*************************************************************************

Function multiplies all its input samples to yield an output sample;
the number of input buffers is arbitrary and determined at run time

Programmer:  J.D. Sun  (this star is a modification of star:  add.s)
Date:  1/9/87
*/

output_buffers

        float sample_out;
end

states

        int no_buffers;
end

declarations

        int buffer_no;
        int no_samples;
end

initialization_code

        /* determine and store as state the number of input buffers */
        if((no_buffers = no_input_buffers()) <= 0)
                return(2); /* no input buffers */
end

main_code

        /* read one sample from each input buffer and multiply them */
        /* note the minimum number of samples on the input buffers
           and iterate that many times */
        for(no_samples=min_avail();no_samples >0; --no_samples) {
```

```
                /* increment time on the output and set sample to one */
                it_out(0);
                sample_out(0) = 1.0;
                for(buffer_no=0; buffer_no<no_buffers; ++buffer_no) {
                        it_in(buffer_no);
                        sample_out(0) = inf(buffer_no,0) * sample_out(0);
                        }
                }

        return(0);      /* one input buffer empty */
end
```

8.3 Topology Files

```
# system2.t
# This system models the duobinary transmitter ,the PN modulator,
# the channel, and the PN demodulator.

#*************************************************************************
# the Transmitter

param int 200
param int 379
star datagen datagen.o

param file data.tm
star data.tm int_printx.o

star precoder precoder.o

param int 252
star bitadjust bitadjust.o

param float 30.0
star datagain gain.o

param int 252
param int 14
star duobinary duobinary.o

param int 60000
param int 4
param int 6
param int 103
param int 0
star pngen pngen.o

param float 1.0
star pngain gain.o

star pnfork fork

star pnmod xmultiply.o

#*************************************************************************
```

```
# the Channel

param float 0
param float 163.84
param int 773
star addnoise addnoise.o

param float 30240.0
param float 60.0
param float 2
param float 1.5
param int 653
star addharm addharm.o

#*****************************************************************************
# the Receiver

star pndemod xmultiply.o

param int 2048
param int 9
star lpf firlpf.o

param file m2.tm
star m2.tm printfile

#*****************************************************************************
# the Connections

connect datagen 0 data.tm 0
connect data.tm 0 precoder 0
connect precoder 0 bitadjust 0
connect bitadjust 0 datagain 0
connect datagain 0 duobinary 0
connect duobinary 0 pnmod 0
connect pngen 0 pngain 0
connect pngain 0 pnfork 0
connect pnfork 0 pnmod 1
connect pnmod 0 addnoise 0
connect addnoise 0 addharm 0
connect addharm 0 pndemod 0
connect pnfork 1 pndemod 1
connect pndemod 0 lpf 0
connect lpf 0 m2.tm 0
```

```
# system3.t
# This system models the duobinary transmitter ,the PN modulator,
# and the notch filter

#*************************************************************************
# the Transmitter

param int 200
param int 379
star datagen datagen.o

param file data.tm
star data.tm int_printx.o

star precoder precoder.o

param int 252
star bitadjust bitadjust.o

param float 30.0
star datagain gain.o

param int 252
param int 14
star duobinary duobinary.o

param int 60000
param int 4
param int 6
param int 103
param int 0
star pngen pngen.o

param float 1.0
star pngain gain.o

star pnfork fork

star pnmod xmultiply.o

param float 30240.0
param float 90.0
param int 11
star notch notch.o
```

```
star pndemod xmultiply

param file notch.tm
star pfile printfile

#*************************************************************************
# the Connections

connect datagen 0 data.tm 0
connect data.tm 0 precoder 0
connect precoder 0 bitadjust 0
connect bitadjust 0 datagain 0
connect datagain 0 duobinary 0
connect duobinary 0 pnmod 0
connect pngen 0 pngain 0
connect pngain 0 pnfork 0
connect pnfork 0 pnmod 1
connect pnmod 0 notch 0
connect notch 0 pndemod 0
connect pnfork 1 pndemod 1
connect pndemod 0 pfile 0
```

```
# system4.t
# This system models the duobinary transmitter ,the PN modulator,
# the notch filter, the PN demodulator, and the narrowband filter.

#*************************************************************************
# the Transmitter

param int 200
param int 379
star datagen datagen.o

param file data.tm
star data.tm int_printx.o

star precoder precoder.o

param int 252
star bitadjust bitadjust.o

param float 30.0
star datagain gain.o

param int 252
param int 14
star duobinary duobinary.o

param int 60000
param int 4
param int 6
param int 103
param int 0
star pngen pngen.o

param float 1.0
star pngain gain.o

star pnfork fork

star pnmod xmultiply.o

param float 30240.0
param float 60.0
param int 11
star notch notch.o
```

```
star pndemod xmultiply.o

param int 2048
param int 9
star lpf firlpf.o

param file m2.tm
star m2.tm printfile

#*************************************************************************
# the Connections

connect datagen 0 data.tm 0
connect data.tm 0 precoder 0
connect precoder 0 bitadjust 0
connect bitadjust 0 datagain 0
connect datagain 0 duobinary 0
connect duobinary 0 pnmod 0
connect pngen 0 pngain 0
connect pngain 0 pnfork 0
connect pnfork 0 pnmod 1
connect pnmod 0 notch 0
connect notch 0 pndemod 0
connect pnmod 0 pndemod 0
connect pnfork 1 pndemod 1
connect pndemod 0 lpf 0
connect lpf 0 m2.tm 0
```

```
# jitter1.t
# This system models the duobinary transmitter ,the PN modulator,
# the channel, and the PN demodulator.
#It also tests the effects of jitter.

#*************************************************************************
# the Transmitter

param int 200
param int 379
star datagen datagen.o

param file data.tm
star data.tm int_printx.o

star precoder precoder.o

param int 252
star bitadjust bitadjust.o

param float 30.0
star datagain gain.o

param int 252
param int 14
star duobinary duobinary.o

param int 70000
param int 4
param int 6
param int 103
param int 0
star pngen pngen.o

param float 1.0
star pngain gain.o

star pnfork fork

star pnmod xmultiply.o

star pndemod xmultiply.o

param int 1
```

```
star delay delay

param int 2048
param int 9
star lpf firlpf.o

param file mljl.tm
star mlj.tm printfile

#*************************************************************************
# the Connections

connect datagen 0 data.tm 0
connect data.tm 0 precoder 0
connect precoder 0 bitadjust 0
connect bitadjust 0 datagain 0
connect datagain 0 duobinary 0
connect duobinary 0 pnmod 0
connect pngen 0 pngain 0
connect pngain 0 pnfork 0
connect pnfork 0 pnmod 1
connect pnmod 0 pndemod 0
connect pnfork 1 delay 0
connect delay 0 pndemod 1
connect pndemod 0 lpf 0
connect lpf 0 mlj.tm 0
```