Adaptive Systems in Digital Communication Designs

by

Xinben Garrison Qian

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science
and
Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Author

Department of Electrical Engineering and Computer Science

Certified by

Professor Richard D. Thornton
Thesis Supervisor

Certified by

Dr. Ganesh Rajan
Tektronix Supervisor

Accepted by

F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Adaptive Systems in Digital Communication Designs

by

Xinben Garrison Qian

Submitted to the Department of Electrical Engineering and Computer Science on May 10, 1996, in partial fulfillment of the requirements for the degrees of Bachelor of Science in Electrical Engineering and Computer Science and Master of Science in Electrical Engineering and Computer Science.

Abstract

The desirable features of an adaptive system, namely, the ability to operate satisfactorily in an unknown environment and also track time variations of input statistics, have make it a powerful device to provide the robustness that is desired for all digital communication systems. An adaptive filter and a closed tracking loop are forms of an general adaptive system. Applications of both forms are considered in this thesis. As an adaptive filtering application, a linear adaptive transversal filter can be used to remove undesirable distortion caused by an analog front end that is present in most digital communication systems. As an closed tracking loop application, a delay-locked loop (DLL) can be used to perform pseudo-noise (PN) code tracking, an essential task in direct-sequence spread-spectrum communication systems.

Thesis Supervisor: Richard D. Thornton
Title: Professor of Electrical Engineering

Thesis Supervisor: Ganesh Rajan
Title: Manager, DSP Group

Acknowledgments

I would like to first thank Ganesh Rajan for his encouragement and support without which this thesis would not be possible. And also I thank my good friend James Yang and Scott Velazquez for their great effort in proofreading this paper. James, Scott, beers will be on me next time. Of course, I have not forgot my fellow co-opers at Tektronix, Kathy Chiu and Johnson Tan. With you guys around, life, no matter how unfair sometimes, will always be fun. At last I would like to thank my dear aunt Binky, and my parents for the love they have given me through the years. I love you all.

Contents

List of Figures

List of Tables

Chapter 1 : Introduction

Digital Communication systems are often required to operate under statistically unknown, or time varying environment. The desirable features of an adaptive system, namely, the ability to operate satisfactorily in an unknown environment and also track time variations of input statistics, have make it a powerful device to provide the robustness that is desired for all communication systems. Perhaps one of the most well known applications of adaptive system in digital communications is that of channel equalization, where an adaptive filter is used to remove inter-symbol interference (ISI) caused by dispersion in the transmission channel. Of course not just limited to filters, adaptive systems also include closed-loop tracking devices, such as phase-locked loops (PLLs) which play an important role in coherent communications. Although adaptive filters and closed tracking loops are quite different in appearance, they have one basic common feature that is shared by all adaptive systems: an input vector and a desired response are used to compute an estimation error, which is in turn used to control the values of a set of adjustable coefficients. The adjustable coefficients may take various forms, such as tap weights, reflection coefficients, or frequency parameters, depending on the system structure employed. But the essential difference between the various adaptive system applications arises in the manner in which the desired response is extracted.

In this thesis, we are going to present two specific adaptive system applications that are used in modern digital communications systems: adaptive receiver front end correction, and pseudo-noise (PN) code tracking in direct-sequence spread-spectrum (DS/SS) systems using delay-locked loops (DLL). Adaptive front end correction is an example of adaptive filtering, while code tracking using DLL, which is closely related to PLL, is an example of a closed-loop tracking device.

1.1 Receiver Front End Correction Problem

In digital communications an attempt is made to determine which signal from a discrete signal has been transmitted. The transmitter codes and modulates a digital information sequence in a manner suitable for the transmission channel in question. This signal is then transmitted through the channel which will introduce both time dispersion and additive noise. If the communication channel is a cable then this dispersion will have a continuous impulse response which may spread over many intervals, thus causing inter-symbol interference (ISI).[1] In the case of a radio channel the dispersion is more likely to be discrete in nature and caused by multipath effects. If the multipath spread exceeds the symbol duration then ISI is once more introduced [21].

After the transmission channel, the signal may encounter another source of distortion at the receiver end. Before this radio frequency (RF) signal can be demodulated, it must be down converted to an appropriate intermediate frequency (IF) through some process such as superheterodyning. The section of the receiver that controls the process of frequency down-conversion is called the receiver front end. Since the down-conversion to IF is normally done through analog means, the front end may introduce some additional distortion to the signal [34]. It is therefore essential for the receiver designer to know whether this additional distortion is significant. First the engineer can formulate a mathematical model for the particular front end to be studied. Often front end can be modeled as a non-ideal band-pass filter similar to a transmission channel model [21], afterward one can simulate a receiver that incorporates this model to study its front end distortion effect. If the distortion were to be found significant, then a correction filter must be applied to remove the excessive distortion in order to keep the overall receiver performance within specifications. One complication arises in practice is that the front

---

[1] ISI is not always undesirable. In the case of Gaussian minimum-shift keying, ISI is allowed as a trade-off for more efficient bandwidth [10].

end characteristic is time varying, due to variations in its down-conversion rate. Accordingly, the use of a single fixed correction filter, which is designed on the basis of average front end characteristics, may not adequately reduce the distortion. This suggests the need for an adaptive equalizer that provides precise control over the time response of the front end. As Chapter 2 of this thesis will show, a simple linear adaptive filter can be used to achieve this correction.

1.2 Code Tracking in Direct-Sequence Spread-Spectrum Systems

Spread-spectrum communication technology has been used in military communications for over half a century, primarily for two purposes: to overcome the effects of strong intentional interference (jamming), and to hide the signal from the eavesdropper (covertness). Both goals can be achieved by spreading the signal's spectrum to make it virtually indistinguishable from background noise [22]. Recently, direct-sequence, a type of spread-spectrum system, has received considerable amount of attention for its possible civilian applications in wireless mobile and personal communications. The key desirable feature in a direct-sequence code division multiple access system (CDMA) is universal frequency reuse, the fact that all users, whether communicating within a neighborhood, a metropolitan area, or even a nation, occupy a common frequency spectrum allocation [12] - [14].

In a direct-sequence spread-spectrum systems, frequency spreading of an information-bearing (data) sequence is achieved through modulation of a wide-band pseudo-noise (PN) sequence. In order to successfully recover the original data sequence, a synchronized replica of the original transmitting PN sequence must be supplied to the receiver. A solution to the code synchronization problem consists of two parts: code acquisition and code tracking. During code acquisition, the two PN sequences are aligned to within a fraction of a chip in as short a time as possible. Once acquisition is completed, code tracking, or fine synchronization, takes place. The goal of the tracking step is to achieve perfect alignment of the two PN sequences. In this thesis,

we will focus on the code tracking step of the synchronization. Because of the distortion caused by the communication channel, it is unrealistic to expect the delay between the transmitter and receiver to be constant. Therefore, one must solve this problem through the use of an adaptive system. Similar to a phase-locked loop, a delay-locked loop (DLL) can be used to provide code tracking.

1.3 Organization

In the next two chapters, we will examine the front end correction problem and delay-locked code tracking in great detail. The front end correction problem will be covered in Chapter 2. First half of the chapter presents a study on front end distortion to both GSM and CDMA signals. Then the second half present a front end correction scheme that uses a linear adaptive filter together with computer simulation results. Chapter 3 addresses the application of delay-locked loops for PN code tracking. Simulation results of a first-order delay-order loop is presented. Chapter 4 provides a summary for the thesis by introducing a general adaptive system model that encompasses both applications being discussed so far. It also explores future research possibilities in the area of adaptive system applications.

Chapter 2 : Front End Correction Problem

The remainder of the thesis describes several applications of adaptive processing in digital communications. This chapter will cover one of the applications - adaptive front end equalization. The description of the front end correction problem is in two parts. The first part (2.1) will present a study of the effect of distortion in a front end used in a high precision telecommunication measurement device. This study shows that the distortion introduced by the front end is excessive and must be removed using a correction scheme. Then in part two (2.2), a front end correction scheme that uses least-mean-square (LMS) algorithm driven adaptive transversal filter is presented. The results will show that by using appropriate design parameters, distortion can be corrected with reasonable cost.

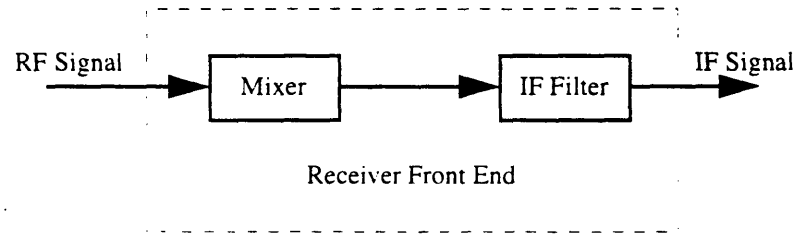

Figure 2.1 A general front end structure

As depicted in Figure 2.1, a front end is the section of a receiver that controls the process of down-conversion of radio frequency (RF) signal to an appropriate intermediate frequency (IF) through a process such as superheterodyning. To illustrate, Figure 2.2 shows a superheterodyne receiver commonly used in AM radio broadcasting. In this receiver, every AM radio signal is converted to a common IF frequency, say f_{IF}. This conversion allows the use of a single tuned IF amplifier for signal from any radio station in the frequency band. The IF amplifier is designed to have a certain bandwidth centered around f_{IF}, a bandwidth which matches that of the transmitted

signal. So in this case the IF amplifier can be seen as the cascade of an IF filter and an amplifier. Since down-conversion to an IF signal is normally done through analog means, the front end inevitably introduces distortion to the signal. This distortion may not have much effect in an AM radio receiver, but it may adversely affect a high resolution, high fidelity telecommunication measurement device. If so, then an error correction scheme must be applied. Through prior work done on the subject, a model for the front end has already been formulated. It was modeled as a six-pole Chebychev type I band-pass filter. This thesis examines use of this model to (1) characterize the distortion by applying simulated GSM and CDMA data signals to this model, and (2) recommend a correction scheme if the distortion was found to be excessive according to the product specifications.

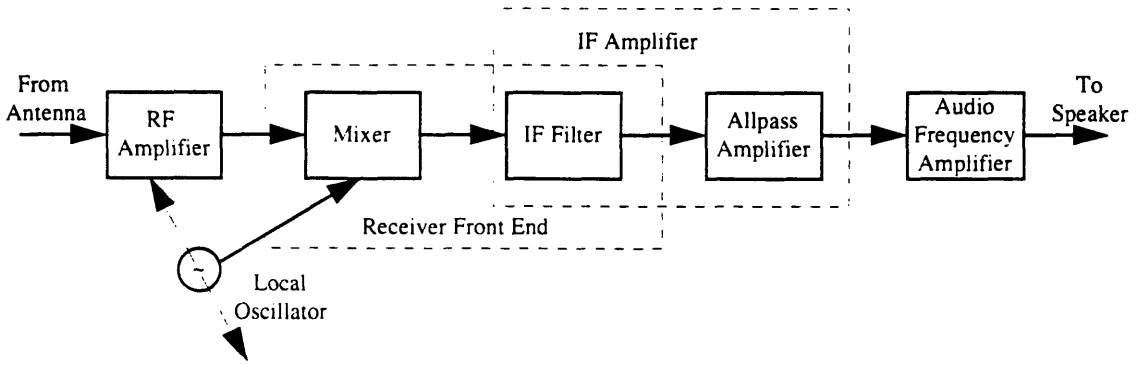

Figure 2.2 A superheterodyne receiver commonly used for AM radios

2.1 Front End Distortion Study

Both GSM and CDMA simulations are based on a front end that is modeled as a Chebychev Type I band pass filter with approximately 6 MHz bandwidth. Its prototype analog low pass filter has six specified poles:

$$\begin{bmatrix} -0.092 \pm 0.966j \\ -0.252 \pm 0.707j \\ -0.344 \pm 0.259j \end{bmatrix}$$

With specified sampling frequency, and above pole values, a MATLAB M-file was written to implement the corresponding digital band pass filter using bilinear transformation. Figure 2.3 shows the frequency response of this digital band pass filter based on these pole values and a sampling rate of 27 MHz.[2] As shown in Figure 2.3, two potential sources of distortion exists: One comes from the ripple in the pass band of the model, and the other stems from the non-constant group delay response visible in the pass band. As shown in the results and analysis section (2.1.1.4 and 2.1.2.4), the ripple in the pass band dominates the distortion.

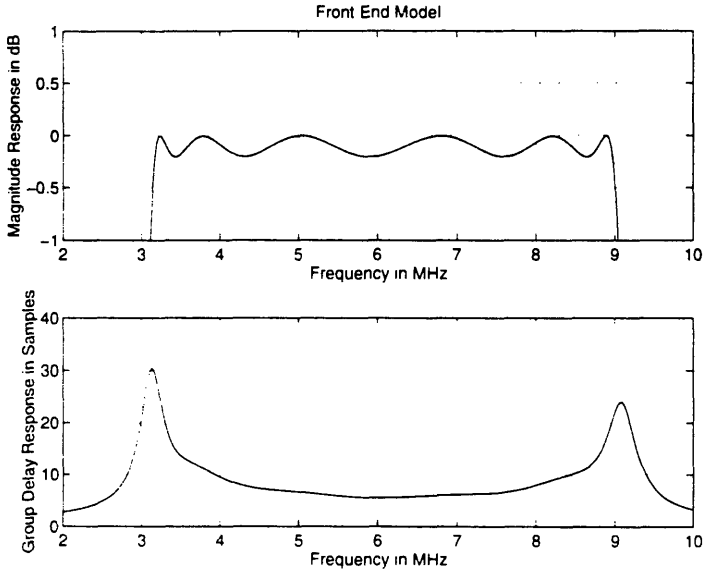

Figure 2.3 Front end modeled as a band pass filter with 6 MHz center frequency, 6 MHz bandwidth, and 0.2 dB ripple in the pass band: (top) magnitude response, (bottom) group delay response

[2] This value should become clear after discussion of the simulation parameters used for GSM and CDMA in the next two sections

To study the effect of this front end model on GSM and CDMA data signals, simulation programs were written to implement the transmitter-receiver structures that were representative of GSM and CDMA standards. Of course, incorporating every existing feature of these standards into our simulations would be both time consuming and unnecessary. Instead, we will only concern ourselves with the features that are pertinent to the study at hand. To elaborate, we will consider the bandwidth of the signal to be an important factor in deciding the extent of the distortion it can tolerate. For a narrow band signal, depending on the frequency location of the signal with respect to the front end, different distortion may result. On the other hand, for a wide band signal, the distortion would be approximately an average of distortions experienced by all the narrow band signals within the wide band range. It would not be unreasonable to assume that narrower bandwidth signals may escape the distortion effects of the front end better than the wider ones. In cases of GSM and CDMA, the former would be relatively narrower than the latter. The code spreading process is the major factor of the wider band appearance of the CDMA signal. Different modulation schemes used for GSM and CDMA further contributes to this bandwidth discrepancy. As we shall see later, Gossip minimum shift keying (GMSK), the modulation scheme used by GSM, is very spectrally efficient. In contrast, the modulation scheme used by CDMA is not as efficient. Features such as these must, therefore, be incorporated into the simulations, while the rest can be safely ignored without affecting the results. In the next two sections we will present the simulations done for GSM and CDMA in detail.

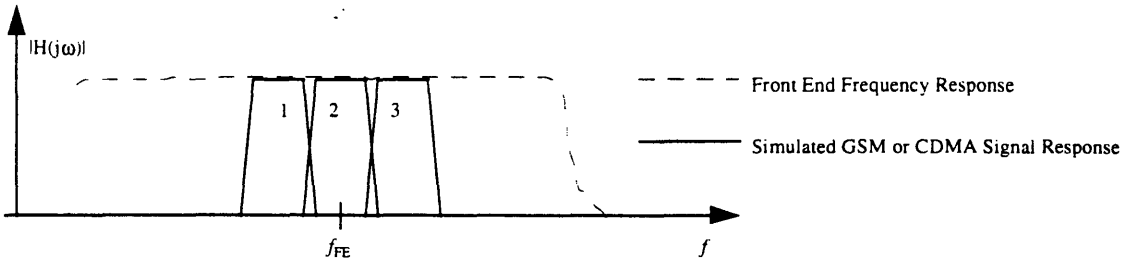

Figure 2.4 Using different carrier frequencies, signal can be placed over various frequency bands of the front end in order to study its behavior

After the transmitter-receiver structures are put into place, simulated data signals for both GSM and CDMA can be easily generated. The features of the instrument, whose front end is being studied here, require the placement of data signals over various frequency regions near the center of the front end. In order to measure distortion in these frequency regions, three sets of data signals will be generated for GSM and CDMA simulation as shown in Figure 2.4 These signals, labeled "1", "2" and "3", differ only in their carrier frequencies, which in turn determine their corresponding placements in the frequency domain. The random data bits used to generate them are identical for all three cases, a fact that will aid in better analysis of the relationship between distortion and frequency placements. For both simulations, the three carrier frequencies are chosen such that the lowest and the highest would correspond to, respectively, -1MHz and +1MHz offset from the center of the front end pass band (labeled f_{FE} in Figure 2.4).

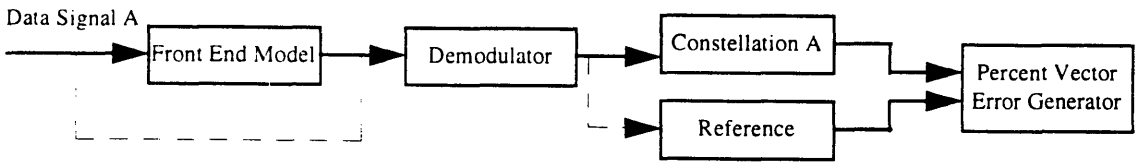

Figure 2.5 Basic experiment setup

In order to generate a measure of the distortion for each of the three data signals shown in Figure 2.4, two different receiver outputs will be generated depending on the data path the input

takes (illustrated in Figure 2.5). The output generated using the dotted line path is considered to be the reference output, or the distortion-free output, since the input data reaches the demodulation stage without passing through the front end model. The other output is obtained by passing the input data along the solid line, which goes through the front end model. This output will reflect the front end distortion effects, and is appropriately labeled as the corresponding distorted output. By comparing this output with its reference output, we can obtain the magnitude of the distortion.

To be consistent with the distortion measurement adopted by the instrument specifications, we are going to use percentage mean-square-root vector error (PMSRVE) to measure the extent of the distortion. Our simulated receiver will produce both an in-phase (I) and a quadrature-phase (Q) component. By plotting the correspond I and Q, we can create what is called a constellation diagram. Designating I and Q as y- and x-axis, respectively, we can treat any point in the constellation as a two dimensional vector. Comparing outputs thus becomes equivalent to comparing corresponding vectors in the two constellation plots created for the two inputs. PMSRVE is thus defined by the following:

$$\text{PMSRVE} = \frac{\sum_{i=1}^{N} \sqrt{\frac{(I_i - I_{i,ref})^2 + (Q_i - Q_{i,ref})^2}{I_{i,ref}^2 + Q_{i,ref}^2}}}{N} \times 100 \qquad \text{(Equation 2.1)}$$

where N is the total number of vectors to be considered for the calculation of PMSRVE, I_i and Q_i are the components for the i-th distorted vector. and $I_{i,ref}$ and $Q_{i,ref}$ are the components for the i-th reference vector. Figure 2.6 illustrates the error vector, the dotted line, between vector (I_i, Q_i) and vector ($I_{i,ref}$, $Q_{i,ref}$).

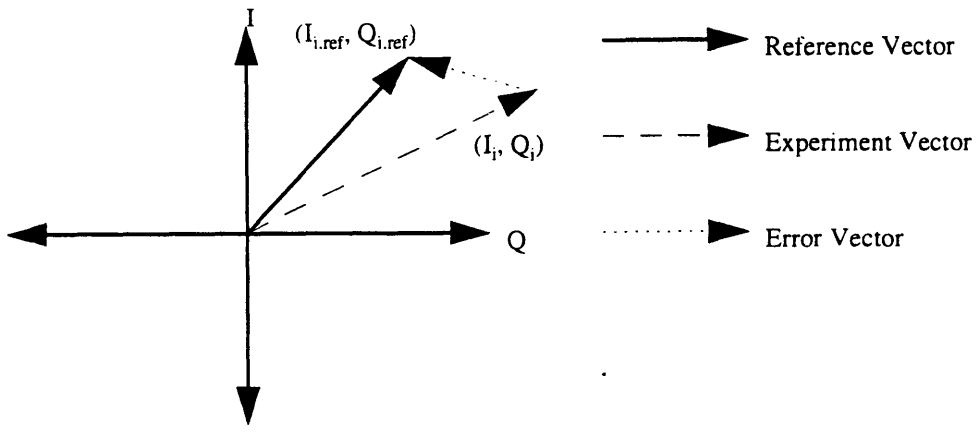

Figure 2.6 Looking at vector error in terms of a constellation diagram. A vector is made up of a pair of symbols from I and Q, respectively.

Using PMSRVE values, we are able to get a sense of the extent of the distortion introduced by our front end model. When values exceed those given in the specification sheet, the distortion will be considered excessive, and possible correction schemes must be addressed next. In the next two sections we are going to cover the GSM and CDMA simulation details. As the results of both simulations will show, the front end distortion indeed causes performance degradation that exceeds requirement, leading to the introduction of a correction scheme using least-mean-square (LMS) adaptive filtering. The results of using such a scheme will then be presented to show that FIR adaptive filtering does offer, with a reasonable cost, a robust error correction solution to the problem at hand.

2.1.1 GSM Simulation

While the last section gives an overall description of the approach we take to study the front end distortion, this section will present in detail the simulation setup for the GSM case. Specifically we are going to present the parallel GMSK transmitter that is used to produce the required GSM data signal, along with the parallel GSMK receiver that is used to produce the constellation plots

later used during PMSRVE calculations. But prior to that, we will give an overview of continuous phase modulation (CPM) of which GMSK is a subtype, which will help us to better understand the transmitter-receiver. And finally, we will present the simulation results that will show, as in the CDMA case, that the magnitude response of the front end introduces most of the error, and not its group delay response (non-constant group delay corresponds to nonlinear phase response). And the extent of the distortion depends heavily on the frequency location of the signal with respect to the front end response. More importantly, without any correction scheme, the front end model clearly introduces excess distortion.

2.1.1.1 Continuous Phase Modulation

While digital communication has grown as a field, one digital communication method known as continuous phase modulation (CPM) has become a popular method of transmitting digital information in a mobile radio environment. In applications such as mobile radio and satellite transmission, where power must be conserved, amplifiers need to operate in a saturated and nonlinear region for maximum efficiency [1]. Because of its constant envelop, CPM is less affected by nonlinear RF amplifiers than the non-constant envelop schemes such as quadrature amplitude modulation (QAM) [2]. Non-constant envelop modulation schemes are also more susceptible to the effects of fading than constant envelop schemes [3]. The continuity of the phase inherent in a CPM signal also reduces bandwidth requirements in comparison to discontinuous phase modulation such as quadrature phase shift keying (QPSK) [9] [5]. While QPSK can be bandlimited before transmission to reduce its out-of-band power, the resulting time varying envelope may cause high error rates for certain symbol transitions [3]. As with most engineering choices, CPM is chosen as a compromise for the mobile radio environment which requires digital transmission, efficient amplifiers, and limited bandwidth.

CPM can be sub-divided into a number of specific types, providing choices between bandwidth, receiver complexity, and power [5] [6]. The general form of a transmitted CPM signal is

$$s(t,\overline{\alpha}) = \sqrt{\frac{2E}{T}}\cos(2\pi f_c t + \phi(t,\overline{\alpha}) + \phi_0) \qquad \text{(Equation 2.2)}$$

$$\phi(t,\overline{\alpha}) = 2\pi h\sum_{i=0}^{\infty}\alpha_i q(t-iT) \qquad \text{(Equation 2.3)}$$

where f_c is the carrier frequency, T is the symbol duration, ϕ_0 is the arbitrary phase, and $\phi(t,\overline{\alpha})$ is the phase information. The phase information is determined by a sequence of M-ary data symbols, $\overline{\alpha} = \alpha_0,\alpha_1,\alpha_2,\ldots$ with $\alpha_i = \pm 1,\pm 3,\ldots,\pm(M-1)$. The parameter h is known as the modulation index and plays a role similar to the frequency deviation ratio in analog frequency modulation by scaling the phase signal representing the transmitted information.

CPM systems are partly classified by the form of q(t), the phase response, which is denoted as

$$q(t) = \int_0^t g(t)dt \qquad \text{(Equation 2.4)}$$

where g(t) is causal and known as the pulse shape of a CPM system. The final value of q(t) is always set to 1/2.

The pulse shape, g(t), has a direct effect on the bandwidth, receiver complexity, and the likelihood of correctly detecting the transmitted data. Each g(t) is primarily characterized by its shape and duration. Longer and smoother pulses reduce the bandwidth of the CPM signal while making symbol detection more difficult. As the duration of g(t) increases past one symbol period, controlled intersymbol interference (ISI) is introduced into the signal. This ISI blurs the phase response of each symbol, making signal detection more difficult. A prefix or suffix, "L", may be

added to the name of a modulation. When present, it will indicate that the corresponding g(t) is truncated to [0, LT]. When L = 1, the CPM system is known as a full response system. When L >1, the system is known as a partial response system. This distinction is important because only partial response systems can create controlled ISI, although they usually have better bandwidth performance. Figure 2.7 illustrates an equivalent representation of the CPM signal as a frequency modulation (FM) system where input symbols are convolved with a filter having the pulse shape, g(t), as its impulse response [6].

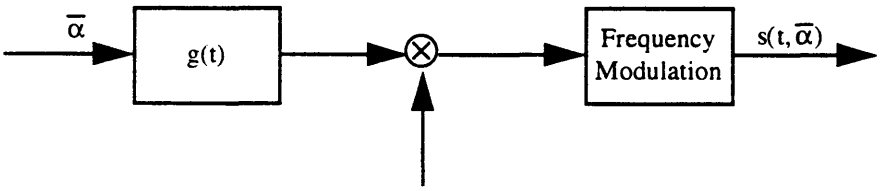

Figure 2.7 An alternative representation of CPM using frequency modulation

2.1.1.2 Gossip Minimum Shift Keying

The modulation scheme adopted by GSM is Gossip minimum shift keying (GMSK), which is a type of CPM. In the GMSK that we are going to consider for the implementation of GSM transmitter/receiver structure, g(t) is a Gossip pulse convolved with a rectangular pulse of unit area and one symbol duration. Specifically, as given in [7], the modulating data values α_l, as represented by Dirac pulses, excite a linear filter with impulse response defined by:

$$g(t) = h(t) * \mathrm{rect}(t / T) \qquad \text{(Equation 2.5)}$$

where $*$ means convolution, T is the symbol duration, the function rect(t) is defined by:

$$\mathrm{rect}(t) = \begin{cases} \frac{1}{T} & |t| < \frac{t}{2T} \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Equation 2.6)}$$

and h(t) is defined by:

$$h(t) = \frac{1}{\sqrt{2\pi}\sigma T} e^{-t^2/2\sigma^2 T^2} \qquad \text{(Equation 2.7)}$$

where $\sigma = \frac{\sqrt{\ln 2}}{2\pi BT}$, and B is the 3 dB bandwidth of the filter with impulse response h(t). The shape of g(t) is parameterized by the time-bandwidth product, BT, where smaller BT values indicate more spreading in the pulse shape (see Figure 2.8). [7] further sets the value of BT at 0.3. The simulation will be based on the values given in [7]. Notice that the response for the ideal Gossip filter is infinite in duration, and thus the GMSK in our simulation must be a truncated signal. Fortunately, the amplitude of GMSK with BT = 0.3 at time instants more than two symbol periods away from its center will be very small, resulting in little effect from the truncation. For smaller BT values, however, such as BT = 0.1, the truncated pulse duration may need to be extended to perhaps six or seven symbol periods before the amplitude of g(t) becomes efficiently small. In any case, the inter-symbol-interference (ISI) will be present.

In the transmitter-receiver structure we are going to simulate, we only considers binary modulation, i.e., M = 2, thus making the area of α_i equal to ± 1. Larger values of M significantly increase the complexity of the receiver. Further more, we will the modulation index, h, to the value of 1/2 (see Equation 2.3) which allows for highly simplified receivers [8]. In the case of full response systems, the value of h = 1/2 is the minimum possible value for the two signals produced by 1 and -1 to be orthogonal-hence the name minimum shift keying [9].

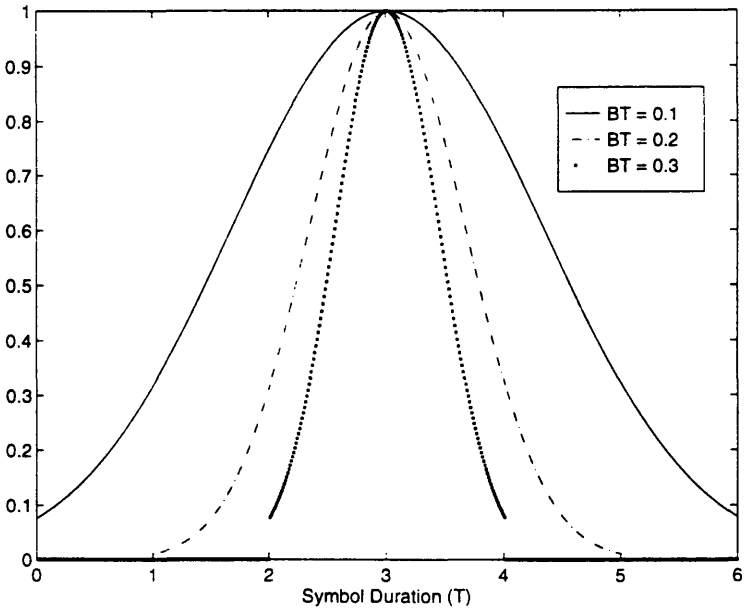

Figure 2.8 Impulse response of g(t) for GMSK with different BT values

2.1.1.3 GMSK Transmitter-Receiver

The simulation program for the transmitter-receiver structure incorporates several assumptions. We assume that the transmission channel is a perfect all pass system without any additive noise terms, and the receiver uses a coherent demodulation method that is capable of both perfect phase recovery and exact timing synchronization.

Now we are going to consider some of the parameters that are used in the simulation. The first issue is the sampling rate. A low sampling rate is desirable to reduce the amount of processing, but a high sampling rate reduces the effects of aliasing. Based on the power spectrum given in Figure 2 of [10], and a symbol rate of 271 kbits/sec specified for GSM [7], the GMSK signal will be attenuated by 60 dB at approximately 340 kHz from its center frequency. Therefore, we will assume that the signal has a bandwidth of 680 kHz (this is a lot smaller than the bandwidth of the signal to be considered for the CDMA case, which we will present in the next section). This dictates the lowest permissible sampling rate of 1360 kHz. For our

simulation, we are going to use an odd integral number of samples per symbol, so that at the receiver end we will have an unambiguous choice for symbol timing. As stated in the previous section, we are going to use a band pass filter of 6 MHz bandwidth, thus the sampling rate has to be much higher. We pick, rather arbitrarily, the number of samples per symbol to be 99. This gives a sampling rate around 27 MHz. The symbols used in the simulation are binary random ± 1 's.

Since GSM uses partial response system [7], i.e., $L > 1$ (see discussion in the CPM section), there exists finite controlled inter-symbol-interference (ISI). For the simulation, we are going to set $L = 2$. It is a compromise between bandwidth performance and controlled ISI. In other words, we are going to use g(t) with duration of 2 symbol periods (T). The time-bandwidth product, BT, is always set to be 0.3.

Incorporating all these parameters, Figure 2.9 shows the block diagram of the parallel GMSK transmitter used for the simulation. Here we take the approach outlined in [9]. The only difference between MSK and GMSK is the addition of a Gossip pulse shaping filter g(t) used in the latter. As proved in [9], a MSK signal can be thought of as a special case of offset quadrature phase shift keying (OQPSK) (We are going to talk more about QPSK and OQPSK in our CDMA simulation). The only difference is that instead of using rectangular pulse shapes as in OQPSK, MSK uses sinusoidal pulses, which is the reason for the appearance of $\cos(\frac{\pi t}{2T})$ and $\sin(\frac{\pi t}{2T})$. Both $a_I(t)$ and $a_Q(t)$ are rectangular pulse stream corresponding to the even and odd bits in the input symbol. All the pulses has duration of 2T. Also note that I channel leads Q channel by one symbol duration, T.

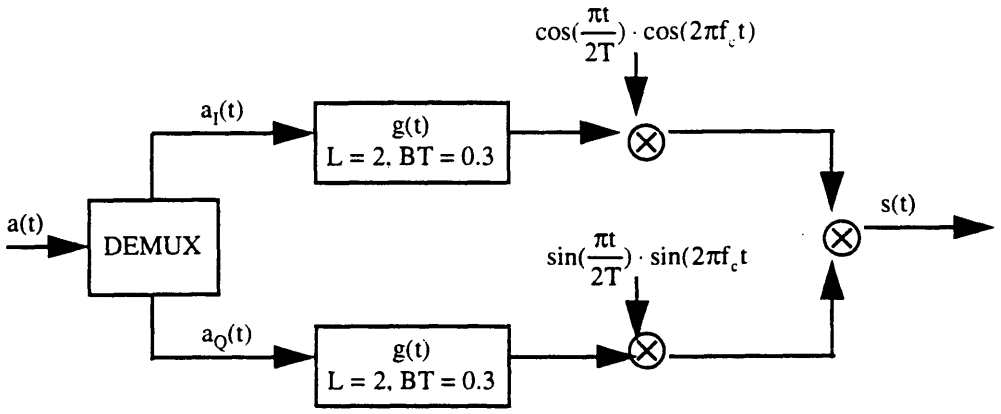

Figure 2.9 Block diagram for the GMSK parallel transmitter: BT = 0.3, sampling frequency = 27 MHz

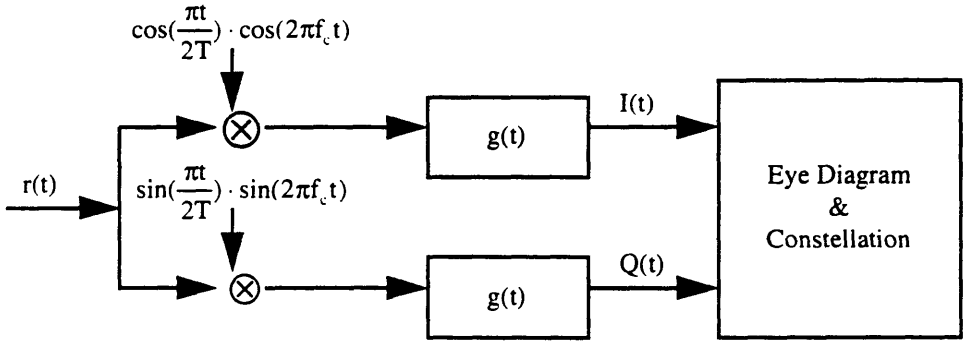

Figure 2.10 Block diagram for the GMSK parallel receiver: BT = 0.3, sampling rate = 27 MHz

For the receiver, we assume perfect recovery of the phase, thus committing the use of a phase-locked loop. The parallel receiver shown in Figure 2.10 downconverts the received signal r(t) into an inphase (I(t)) and quadrature (Q(t)) branch. While a low-pass filter could be used to remove the high frequency components, the predetection filter sufficiently attenuates the high frequency components to avoid the use of this extra filter. The time domain shape of the predetection filter is equivalent to a Gossip pulse convolved with a rectangular pulse, which is like g(t) itself. Since BT = 0.3 is very close to the level found in [10] for optimum bit error rate (BER) performance, the same g(t) is also used in the receiver. One should note, however, that the

value of BT = 0.3 for the predetection filter is almost independent of the BT value used in the transmitter.

2.1.1.4 Simulation Results and Analysis

GMSK transmitter/receiver is simulated using MATLAB with data bits generated by a random number generator. As we have stated, the simulated GSM signal would be generated using three different carrier frequencies so that we can study the relationship between the magnitude of the distortion and frequency. A single set of data bits is used throughout the whole simulation to ensure that we can analyze the results considering minimum number of variables. Since both the front end model and the data bits used are identical in all three cases, any difference in distortion behavior noticeable at the receiver end can be solely ascribed to the nonidealness of the front end response. The measure for the distortion is provided by PMSRVE values as described in Equation 2.1.

Since the center frequency of the front end is approximately around 6 MHz, with the assumption that most signals we are interested in will be concentrated around that frequency, we choose 5Mhz, 6 MHz, and 7 MHz to be the three carrier frequencies to be used in the simulation. Three percent error values are calculated based on Equation 2.1, and they are listed in Table 2-1. We can easily notice that the values are rather different depending on the carrier frequency being used: from 0.29% at 5MHz, to 4.5% at 6MHz. This means that the amount of distortion created by the front end depends heavily on the frequency location of the data signal. This frequency dependent behavior can be easily explained if we look closely at the frequency response of the front end model shown before in Figure 2.3, where the group delay response over most of the pass band (excluding where it is near the edges) can be characterized as flat, the magnitude response clearly shows ripples as large as 0.2 dB. As we showed before, the bandwidth of the GSM signal is around 680 kHz. With the particular carrier frequency of 5MHz, most of the data is

concentrated around the part of the ripple that is probably the closest to the ideal 0 dB line; this explains why the error value at that location is small. On the other hand, for the carrier frequency of 6MHz, the data will be sitting near where the magnitude response of the front end is at its worst, thus the resulting high PMSRVE value.

Carrier Frequency	5MHz	6MHz	7MHz
Percent Error (%)	0.2993	4.5057	0.6755

Table 2-1 Error values for various carrier frequencies

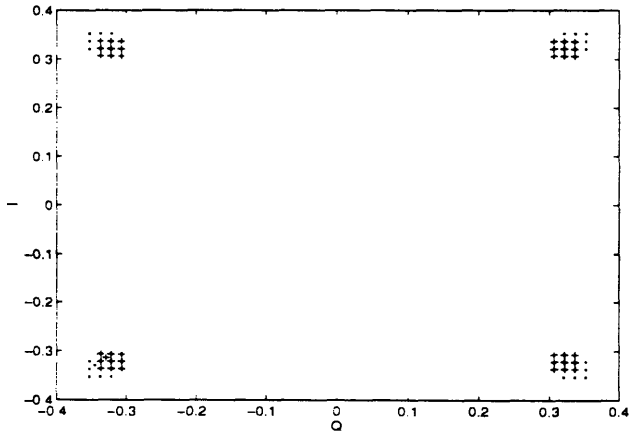

Figure 2.11 GMSK constellation diagram: dotted line represents undistorted data, while "+" represents the one using front end corrupted data.

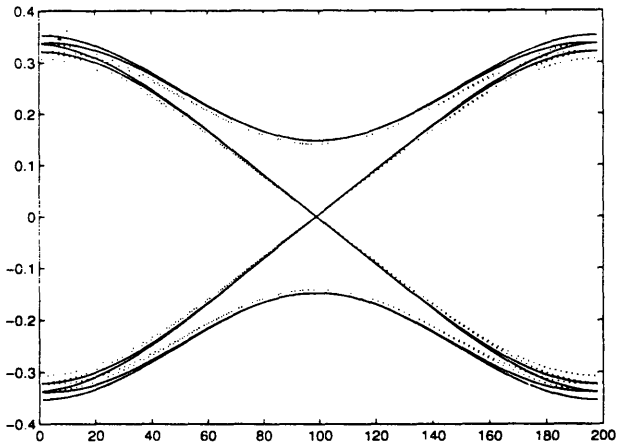

Figure 2.12 GMSK eye-diagram: solid line is the undistorted version, while the dotted line is from corrupted data. Number of samples per symbol is 99.

This distortion, if not removed, will cause significant loss in performance in the decision making stage of receiver. Specifically, we will see that this distortion can be translated into the shift in the constellation plot, or equivalently, the narrowing of the eye opening in the eye-diagram. Using I and Q data, we can easily plot the constellation. Figure 2.11 shows two superimposed plots. The dotted one is the undistorted version, while the one plotted using "+" is generated using data received after passing through the front end at 6 MHz. For the 6 MHz one, there are decreases in magnitude for both I and Q. Since we are dealing with binary ± 1 symbols, this brings the + side closer to the - side, thus distinguishing them correctly becomes harder. This effect can also be seen in an eye-diagram shown in Figure 2.12, where the solid line represents the undistorted data, and the dotted line the corrupted data. For a high resolution GSM measurement device, the largest percent error it can tolerate is less than 0.3%[3]. Thus some kind of correction scheme must be devised to remove this distortion. Later we are going to show that using a linear phase FIR adaptive filter, this kind of distortion can be easily removed.

---

[3] This number is simply an estimate of what is actually used to build the instrument.

2.1.2 CDMA Simulation

Similar front end distortion studies are performed for the simulated CDMA data signals. To generate CDMA signal, we have to first develop a sufficient model for the system. In contrast to GSM, which uses the conventional narrow band communication technology, CDMA is based on spread spectrum signaling, which is used extensively in military communications to overcome strong intentional interference (jamming), and to prevent eavesdropping. Both goals can be achieved by spreading the signal's spectrum to make it virtually indistinguishable from background noise. The particular CDMA signal we are going to simulate here is based on direct-sequence spread spectrum signaling, where spreading is achieved by modulating the conventional bandlimited signal with a long pseudo-random code. CDMA is a complicated system where it would be impractical to include every single feature in our model. Fortunately, as we will show later, we need not concern too much about the spread spectrum technical details for the purpose of this simulation. A conventional QPSK transmitter, provided with appropriate data rate, should be sufficient to generate a data signal that includes the necessary features adequate for the study of distortion effects of the front end model on CDMA. A more extensive discussion on spread spectrum system will be given in the next chapter.

The remainder of the section is structured as follows. First we are going to give a very brief description of a direct-sequence CDMA system. Based on that, we are going to select a small number of important features that are going to be included into our simulation model by making appropriate assumptions. The most important feature to be included in our model of CDMA is its much wider signal bandwidth compare to that of GSM. Second we are going to present a QPSK transmitter-receiver structure that is used to both generate the simulate CDMA signals and derive the their outputs to be used for the PMSRVE calculations. At this point it should become apparent that the main reason for the CDMA signals to experience more severe distortion stems from its wider bandwidth. Based on the results of the previous GSM study, we

can safely say that correction should be necessary to keep the receiver performance level within specification.

2.1.2.1 CDMA

As is well known, the two most common multiple access techniques are frequency division multiple access (FDMA) and time division multiple access (TDMA). In FDMA, all users transmit simultaneously, but use disjointed frequency bands. In TDMA, all users occupy the same radio frequency (RF) bandwidth, but transmit sequentially in time. When users are allowed to transmit simultaneously in time and occupy the same RF bandwidth as well, some other means of signal separation at the receiver must be available; CDMA, or code division multiple access, provides this necessary capability.

A simplified block diagram of a direct sequence CDMA system is shown in Figure 2.13. (Direct-sequence spread spectrum system will be discussed in the next chapter when we cover delay-locked loops. Interested readers can also consult several other references on the subject [11] - [13].) Here the message $d_i(t)$ of the i-th user, in addition to being modulated by the carrier $\cos(\omega_o t + \theta_i)$, where θ_i is the random phase of the i-th carrier, is also modulated by a spreading code waveform $p_i(t)$ that is specifically assigned to the i-th user, hence the name code division. This code modulation translates into spectrum spreading in the frequency domain because the code rate, or chip rate, is usually much higher than the underlying data rate. To see this, for instance before the spreading, the signal has a one-sided bandwidth of B, where B is the data rate. But after code modulation, the signal bandwidth becomes B_c, where B_c is the chip rate (which is much greater than B). Now this time, the signal that has to pass through our front end model has a much wider bandwidth than what we saw for the GSM case. So we would expect different levels of distortion at the receiver output. The spreading code waveform for the i-th user is a pseudo-random (PN) sequence waveform that is approximately uncorrelated with the waveforms

assigned to other users. This property enables the receiver to distinguish the different data sent by multiple users. There are a lot of technical issues involved in the design of CDMA systems [11]. For the present simulation, we are able to safely to simplify most of the details.

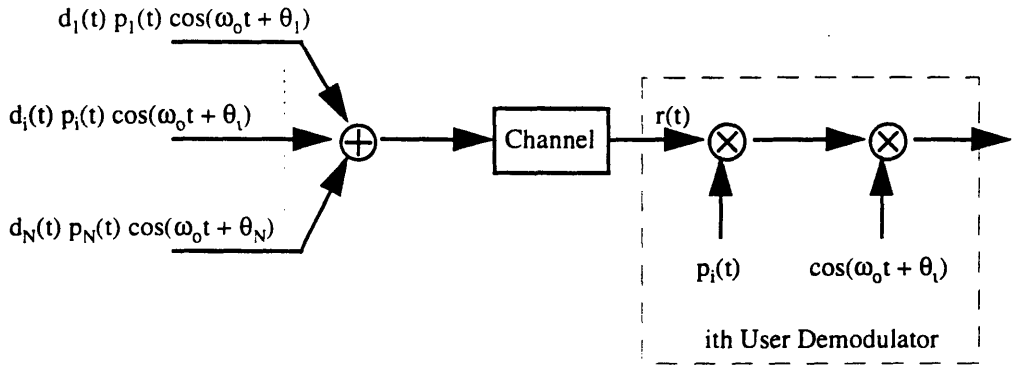

Figure 2.13 General direct-sequence CDMA system

2.1.2.2 Simulation Setup

Similar to what we have done for the GSM simulation, we are going to assume that the only source of distortion is the front end itself. This leads to the assumptions of an ideal channel and a receiver which performs perfect carrier recovery. In addition, since we are dealing with CDMA, which is based on a spread spectrum system, we are also going to assume perfect code synchronization at the receiver end (code synchronization is the topic of next chapter).

Based on the model illustrated in Figure 2.3, the amount of distortion that front end can inflict on a signal will be closely related to two parameters associated with the signal, namely its frequency location and bandwidth. We have already seen the close relationship between frequency location and distortion from the GSM simulation results. We expect the similar relationship holds for the CDMA signals. Since CDMA signals has a much wider bandwidth than that of GSM, we are going to use the CDMA simulation to help us better understand the relationship between distortion and signal bandwidth, which can not be observed just from the GSM simulations. The bandwidth difference is mainly caused by the additional code spreading

process used in CDMA systems. In addition, less spectrally efficient modulation scheme used by CDMA also widens this bandwidth discrepancy. For the purpose of our simulation, we can further simplify the simulation model from what is shown in Figure 2.13. We are only going to consider the single-user case, since its signal has the same bandwidth as that of a multi-user signal, which is also much more complex. The code spreading process is simulated using random data bits whose rate equals the code rate (also called chip rate) of 1.2288 MHz, which is often used for CDMA systems [14]. The transmitter-receiver structure used in the simulation is based on quadrature phase-shift-keying (QPSK), a common modulation scheme for CDMA [12]. Regarding sampling rate, we are going to choose the number of samples per chip such that the overall sampling rate is at par with that used in the GSM case. This would ease the comparison of their distortion results so that we can better understand the relationship between distortion and bandwidth.

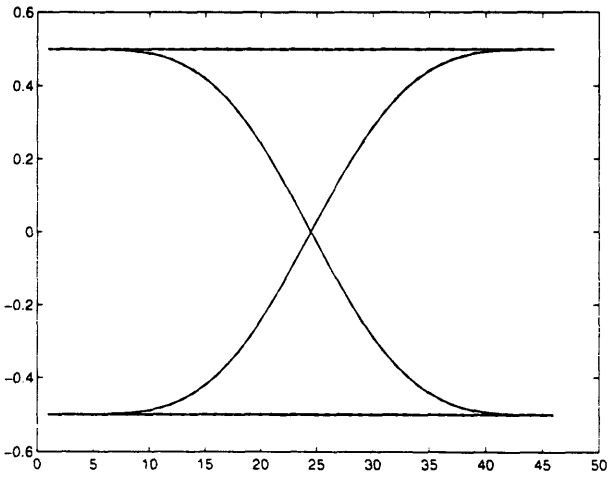

Figure 2.14 QPSK eye diagram: 23 samples per chip

2.1.2.3 QPSK Transmitter/Receiver

The block diagrams for the parallel QPSK transmitter and receiver are shown in Figure 2.15 and Figure 2.16. Compared to the GMSK transmitter, there is no time offset between I and

Q arms in the QPSK transmitter. Instead of using a Gossip filter as for GMSK, this transmitter uses a square root raised-cosine (SRRC) filter as the base-band shaping filter. The same filter is also used by the receiver after the carrier is removed. The desired magnitude response is given by [14]:

$$|H(f)| = \begin{cases} 1 & 0 \le f \le \frac{1-\alpha}{2T} \\ \sqrt{0.5\left\{1-\sin\left[\frac{\pi(2fT-1)}{2\alpha}\right]\right\}} & \frac{1-\alpha}{2T} \le f \le \frac{1+\alpha}{2T} \\ 0 & f > \frac{1+\alpha}{2T} \end{cases} \qquad \text{(Equation 2.8)}$$

where T is the symbol period, which in this case is 1/1.2288 msec, α is the roll-off factor, which determines the width of the transition band, and is set to be 0.35 [14]. Figure 2.14 QPSK eye diagram: 23 samples per chipshows a eye diagrams generated using this transmitter-receiver structure.

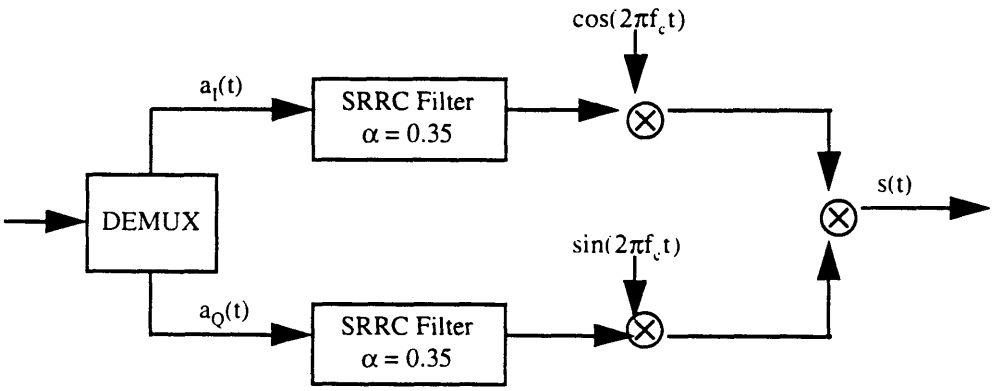

Figure 2.15 QPSK transmitter used in the CDMA simulation

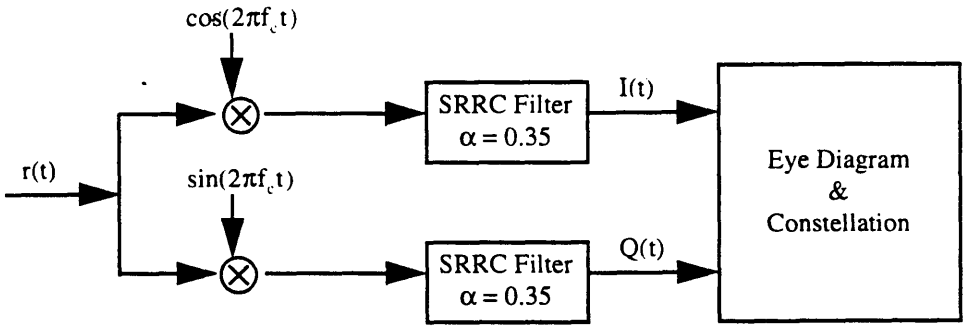

Figure 2.16 QPSK receiver used in the CDMA simulation

2.1.2.4 Results and Analysis

The QPSK transmitter/receiver is simulated using MATLAB with data bits generated by a random number generator. Similar to what we have done for the GSM case, three sets of simulated CDMA signal are generated using three different carrier frequencies so that we can study the relationship between the magnitude of the distortion and frequency. A single set of data bits is used throughout the whole simulation to ensure that we can analyze the results with a minimum number of variables. Since both the front end model and the data bits used are identical in all three carrier frequencies, any difference in distortion behavior noticeable at the receiver end can be solely ascribed to the different front end response at various frequency locations. The measure for the distortion is provided by PMSRVE values as described in Equation 2.1.

Like in the GSM case, the center frequency of the front end is approximated around 6 MHz, with the assumption that most signal we will be interested in will be concentrated around the that number, we choose 5Mhz, 6 MHz, and 7 MHz to be the three carrier frequencies to be used in the simulation. Three error values are calculated based on Equation 2.1, and they are listed in Table 2-2. Similar to the error values in Table 2-1, the error values are closely related to the frequency placement of the signal with respect to the ripples in the front end magnitude response. Since the CDMA signal has a much wider bandwidth, the effect of the front end is

averaged over the band occupied by the signal. That is why at 6MHz, the error value is actually smaller for CDMA than that of GSM.

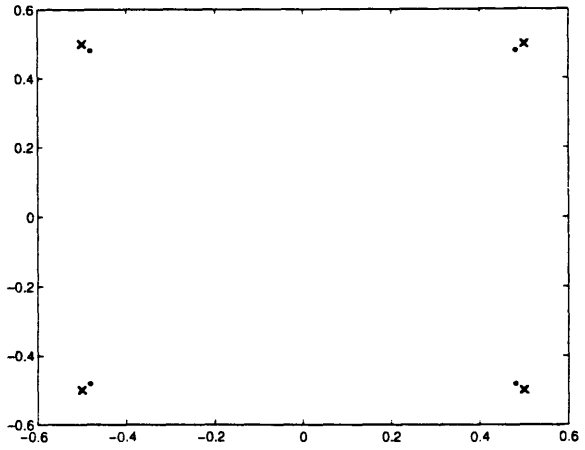

Figure 2.17 QPSK constellation diagram: ' x ' represents undistorted data, while " • " represents front end corrupted data

Figure 2.17 shows two superimposed constellation plots generated using data of 6MHz carrier frequency, where the cross represents the undistorted constellation, and the dot represents the front end corrupted version. Clearly both I and Q for the distorted version show smaller values than their distortion free counterparts. Since we are dealing with binary ±1 symbols, this brings the + side closer to the - side, thus distinguishing them becomes more difficult. An error of 3.79% for the 6MHz case in Table 2-2 is not acceptable. Some kind of correction scheme must be devised to remove this distortion. In the next section, we are going to introduce an error correction solution that uses a FIR adaptive filter. We are going to show that when the adaptive filter is properly set, it can be very effective in removing the distortion.

Carrier Frequency	5MHz	6MHz	7MHz
Percent Error (%)	0.8094	3.7926	1.5889

Table 2-2 CDMA signal error measured for various carrier frequencies

2.2 Adaptive Front End Equalization

The results thus far clearly show that the distortion introduced by the front end on both GSM and CDMA signals is well above what specifications allow, and corrections must be applied to remove the distortion. One might consider using a fixed correction filter such that when it is cascaded with the front end, the overall response would more or less resemble an ideal allpass system with both gain and group delay being constant. However, this is not a feasible approach. Modeling the front end as a fixed band pass filter (shown in Figure 2.3) was solely for the purpose of characterizing the extent of its nonidealness. Specifically, we would like to know from the model what is the largest amount of distortion it can afflict on the signal that passes through it. Based on that information, we can then make a decision about whether correction will be needed. In real time operation, however, the front end displays a slow[4] time-varying response, with its exact shape at any time instant unknown. The model is simply an approximation of its average response. So a fixed correction filter may work during one time period, but fail at the next. Clearly we can not keep on replacing new correction filter for the old one, unless we take a adaptive system approach. Specifically we propose the use of a linear adaptive filter that is able to track the slow changes of the front end response, so that the overall response of their cascade would be all-pass.

After we give a brief overview of the concept of adaptive filtering, we are going to present a linear adaptive equalization structure that can be applied to the front end, and describe the least mean-square (LMS) algorithm that can be used to derive the correction filter. And finally, we are going to present the results of the simulation. As they will show, by choosing appropriate design parameters, front end distortion can be successfully removed for both GSM and CDMA signals.

---

[4] It is slow compared to the convergence time required for the adaptive filter.

2.2.1 Adaptive Filters

In the statistical approach to the solution of the linear filtering problem, we assume the availability of certain statistical parameters, such as the mean and correlation functions, of the useful signal and unwanted additive noise. The requirement is then to design a linear filter with the noisy data as input so as to minimize the effects of noise at the filter output according to some statistical criterion. A useful approach to this filter-optimization problem is to minimize the mean-square value of the error signal, which is defined as the difference between some desired response and the actual filter output. For stationary inputs, the resulting solution is commonly known as the Wiener filter, which is said to be optimum in the mean-square sense.

The design of a Wiener filter requires a priori information about the statistics of the data to be processed. The filter is optimum only when the statistical characteristics of the input data match the a priori information on which the design of the filter is based. When this information is not known completely, however, it may not be possible to design the Wiener filter or else the design may no longer be optimum. A straightforward approach that we may use in such situations is the "estimate and plug" procedure. This is a two-stage process whereby the filter first "estimates" the statistical parameters of the relevant signals and then "plugs" the results into a nonrecursive formula for computing the filter parameters. For real-time operation, this procedure has the disadvantage of requiring excessively elaborate and costly hardware. A more efficient method is to use an adaptive filter. An adaptive filter is self-designing in that it relies for its operation on a recursive algorithm, which makes it possible for the filter to perform satisfactorily in an environment where complete knowledge of the relevant signal characteristics is not available. Such is the case with our front end, which has, to some extent, unknown frequency response. The algorithm starts from some predetermined set of initial conditions, representing complete ignorance about the environment. Yet, in a stationary environment, after successive iterations of the algorithm, the filter will converge to the optimum Wiener solution in some

statistical sense. In a nonstationary environment, the algorithm offers a tracking capability, whereby it can track time variations in the statistics of the input data, provided that the variations are sufficiently slow. Again, fortunately, our front end offers this feature.

As a direct consequence of the application of a recursive algorithm, whereby the parameters of an adaptive filter are updated from one iteration to the next, the parameters become data dependent. This, therefore, means that an adaptive filter is a nonlinear device in the sense that it does not obey the principle of superposition. In another context, an adaptive filter is often referred as linear in the sense that the estimate of a quantity of interest is obtained adaptively as a linear combination of the available set of observations applied to the filter input.

The operation of an adaptive filter involves two basic processes: (1) a filtering process designed to produce an output in response to a sequence of input data, and (2) an adaptive process, the purpose of which is to provide a mechanism for the adaptive control of an adjustable set of parameters used in the filtering process. While the filtering process depends its operation on its filter structure, the adaptive process is governed by a adaptive algorithm. Thus a general adaptive filter can be presented as in Figure 2.18, where u(n) is the input, d(n) is the desired response, $\hat{d}(n|U_n)$ is the estimate of the desired response given the input data that span the space U_n up to and including time n. The front end equalization method that we are going to simulate here uses a type of filter structure called the transversal filter, and a particular adaptive algorithm called the least mean-square (LMS) algorithm.

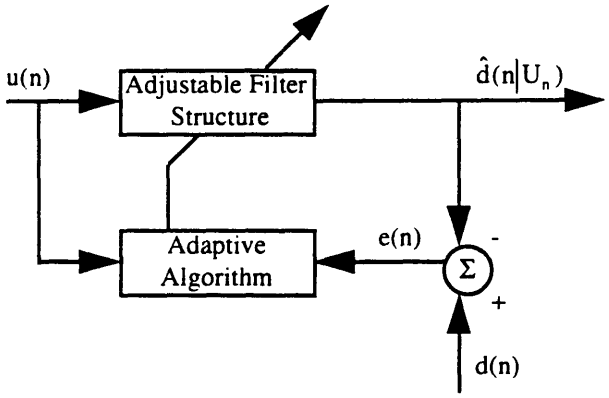

Figure 2.18 Block diagram of a general adaptive filter.

2.2.1.1 Transversal Filter

The choice of a structure for the filtering process has a profound effect on the operation of a adaptive filter as a whole. There are mainly three types of filter structures that distinguish themselves in the context of an adaptive filter with finite memory or, equivalently, finite impulse response (FIR). They are transversal filter, lattice predictor, and systolic array. We choose transversal filter as our filter structure because it is the most straight forward of the three for software simulation, and also it is usually the least expensive for hardware implementation. If the reader is interested in learning more about the other two types of filter structures, [15] provides a good overview.

The transversal filter, also referred to as a tapped-delay line filter, consists of three basic elements, as depicted in Figure 2.19: (1) unit-delay element (square box), (b) multiplier (large circle), and (c) adder (small circle). The number of delay elements used in the filter determines the duration of its impulse response. The number of delay elements, shown as M - 1 in Figure 2.19, is commonly referred to as the order of the filter. In this figure, the delay elements are each identified by the unit-delay operator z^{-1}. The role of each multiplier in the filter is to multiply the

tap inputs (to which it is connected) by a filter coefficient referred to as a tap weight. Thus a multiplier connected to the k-th tap input u(n - k) produces the inner product $w_k u(n - k)$, where w_k is the respective tap weight and k = 0, 1, . . . , M - 1. Here we assume that the tap inputs and therefore the tap weights are all real valued. For complex values, in place of w_k should be its complex conjugate. The combined role of adders in the filter is to sum the individual multiplier outputs (i.e., inner products) and produce an overall filter output. For the transversal filter described in Figure 2.19, the filter output is given by

$$y(n) = \sum_{k=0}^{M-1} w_k u(n-k) \qquad \text{(Equation 2.9)}$$

which is a finite convolution sum of the filter impulse response $\{w_n\}$ with the filter input $\{u(n)\}$. One thing worth noting is that an FIR filter is inherently stable, which is characterized by its feedforward only paths, unlike an infinite impulse response (IIR) filter which contains feedback paths. This is also one reason we choose a FIR filter as the structural basis for the design of adaptive filter. Now we have settled the filter structure, what kind of guarantee do we have on that $\{w_n\}$ are going to converge eventually? To answer that question, we have to know the adaptive algorithm that is used to control the adjustment of these coefficients

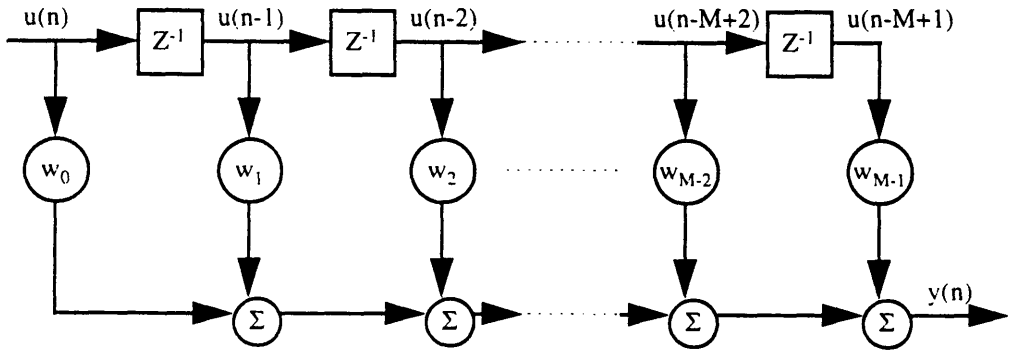

Figure 2.19 Block diagram of a transversal filter

2.2.1.2 Least Mean-Square (LMS) Algorithm

A wide variety of recursive algorithms have been developed in the literature for the operation of adaptive filters. In the final analysis, the choice of one algorithm over another is determined by various factors. Some of the important ones are (1) rate of convergence, (2) tracking ability, and (3) computational requirements. Rate of convergence is defined as the number of iterations required for the algorithm, in response to stationary inputs, to converge "close enough" to the optimum Wiener solution in the mean-square sense. A fast rate of convergence allows the algorithm to adapt rapidly to a stationary environment of unknown statistics. The tracking ability of an algorithm applies when an adaptive filter is operating in a nonstationary environment. It is more desirable if the algorithm has the ability to track statistical variations in the environment. Computational requirements involve (a) the number of operations (i.e., multiplications, divisions, and additions/subtractions) required to make one complete iteration of the algorithm, (b) the size of memory locations required to store the data and the program, and (c) the investment required to program the algorithm on a computer. Depending on individual circumstances, one can not say which factor is more important than the other. A good design is usually a sound compromise of all the above factors.

As we mentioned in 2.4.2, we choose to use the transversal filter as the structural basis for implementing the adaptive filter. For the case of stationary inputs, the mean-squared error (i.e., the mean-square of the difference between the desired response and the transversal filter output) is precisely a second-order function of the tap weights in the transversal filter. The dependence of the mean-squared error on the unknown tap weights may be viewed as a multidimensional paraboloid with a uniquely defined bottom or minimum point. We refer to this paraboloid as the error performance surface. The tap weights corresponding to the minimum point of the surface define the optimum Wiener solution. In matrix form, the following relation holds [16]

$$\mathbf{R}\mathbf{w}_o = \mathbf{p} \qquad \text{(Equation 2.10)}$$

where **R** is given by

$$R = \mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{u}^T(\mathbf{n})] \qquad \text{(Equation 2.11)}$$

where **u**(n) the column vector made up of the tap inputs u(n), u(n -1), . . . , u(n - M + 1) as shown in Figure 2.19, **p** is the M-by-1 cross-correlation vector between **u**(n) and the desired response d(n), and $\mathbf{w}_o$ is the M-by-1 optimum tap-weight vector of the transversal filter. Equation 2.10 is also called the Wiener-Hopf equation. To solve this equation, we assume that the correlation matrix **R** is nonsingular. We may then premultiply both sides by $\mathbf{R}^{-1}$ obtaining

$$\mathbf{w}_o = \mathbf{R}^{-1} \quad \mathbf{p} \qquad \text{(Equation 2.12)}$$

The computation of the optimum tap-weight vector $\mathbf{w}_o$ thus requires knowledge of two quantities: (1) the inverse of the correlation matrix **R**, and (2) the cross correlation vector **p**. Using the method of steepest descent, a well-known technique in optimization theory, we can actually compute the updated value of the tap-weight **w**(n+1) without knowing $\mathbf{R}^{-1}$ [17]:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)] \quad \text{(Equation 2.13)}$$

where we call μ the step-size parameter. Here, though simplified compared to Equation 2.12, in order to find the optimum tap-weight, we still need to know the correlation matrix **R** and the cross correlation vector **p**. In reality, however, exact knowledge of both **R** and **p** is usually not attainable. Consequently, they must be estimated from the available data. In other words, the tap-weight vector must be updated in accordance with an algorithm that adapts to the incoming data. One such algorithm is the least mean-square (LMS) algorithm [17]. Rather than using **R** and **p** directly, LMS algorithm uses two estimators for **R** and **p**. The simplest choice of estimators for **R** and **p**, which we are going to represent them by **R**' and **p**', respectively, is to use instantaneous

estimates that are based on sample values of the tap-input vector and desired response, as defined by, respectively,

$$\mathbf{R}'(n) = \mathbf{u}(n)\mathbf{u}^T(n) \qquad \text{(Equation 2.14)}$$

and

$$\mathbf{p}'(n) = \mathbf{u}(n)\, d(n) \qquad \text{(Equation 2.15)}$$

Using these two equations, the new update formula becomes

$$\mathbf{w}'(n+1) = \mathbf{w}'(n) + \mu\mathbf{u}(n)e(n) \qquad \text{(Equation 2.16)}$$

where

$$e(n) = d(n) - y(n) \qquad \text{(Equation 2.17)}$$

is the estimation error, in which $y(n) = \mathbf{w}'^T(n)\mathbf{u}(n)$ is the filter output. The detailed derivation of both LMS and steepest-descent algorithm is given in the Appendix. Clearly a significant feature of the LMS algorithm is its simplicity; it does not require measurements of the pertinent correlation functions, nor does it require matrix inversion. Indeed, it is the simplicity of the LMS algorithm that has made it the standard against which other adaptive filtering algorithms are benchmarked. In particular, the LMS algorithm requires only 2M + 1 multiplications and 2M additions per iteration, where M is the number tap weights used in the adaptive transversal filter.

If we compare $\mathbf{R}'$ and $\mathbf{p}'$ to $\mathbf{R}$ and $\mathbf{p}$, respectively, the main difference is the expectation operator that appears in the latter cases. Accordingly, the recursive computation of each tap weight in the LMS algorithm suffers from a gradient noise. For a stationary environment, the method of steepest descent computes a tap-weight vector $\mathbf{w}(n)$ that moves down the error performance surface along a deterministic trajectory, which terminates on the Wiener solution $\mathbf{w}_o$. The LMS algorithm, on the other hand, behaves differently because of the presence of gradient noise. Rather than terminating on the Wiener solution, the tap-weight vector $\mathbf{w}'(n)$ computed by

the LMS algorithm executes a random motion around the minimum point of the error performance surface. This random motion gives rise to two forms of convergence behavior for the LMS algorithm:

1. Convergence in the mean, which means that

$$E[\mathbf{w}'(n)] \rightarrow \mathbf{w}_o \text{ as } n \rightarrow \infty$$

2. Convergence in the mean square, which means that

$$J(n) \rightarrow J(\infty) \text{ as } n \rightarrow \infty$$

where J(n) is the mean-square error defined by $E[e^2(n)]$, and $J(\infty)$ is always greater than the minimum mean-square error J_{min} that corresponds to the Wiener solution.

For these two forms of convergence to hold, the step size parameter μ has to satisfy different conditions related to the eigenvalues of the correlation matrix of the tap inputs.

The difference between the final value $J(\infty)$ and J_{min} is called the excess mean-squared error $J_{ex}(\infty)$. This difference represents the price paid for using the adaptive mechanism to control the tap weights in the LMS algorithm in place of a deterministic approach as in the method of steepest descent. It is important to realize, however, the extent of excess error is under the designer's control. In particular, the feedback loop acting around the tap weights behaves like a low-pass filter, whose "average" time constant is inversely proportional to the step-size parameter μ [18]. Hence, by assigning a small value to μ the adaptive process is made to progress slowly, and the effects of gradient noise on the tap weights are largely filtered out. This in turn has the effect of reducing the excess error. We may therefore justifiably say that the LMS algorithm is simple in implementation, yet capable of delivering high performance by adapting to its external environment. To do so, however, we have to pay particular attention to the choice of a suitable value for the step-size parameter μ. With these ideas in mind, we are now ready to apply LMS adaptive transversal filter for front end equalization.

2.2.2 Front End Equalization

The desirable features of an adaptive filter, namely, the ability to operate satisfactorily in an unknown environment and also track time variations of input statistics, make the adaptive filter a powerful device for signal-processing and control applications. In this section, we are going to present an application of a LMS adaptive transversal filter. Specifically, we are going to use this filter to remove excessive distortion introduced by the front end we have studied in 2.1.

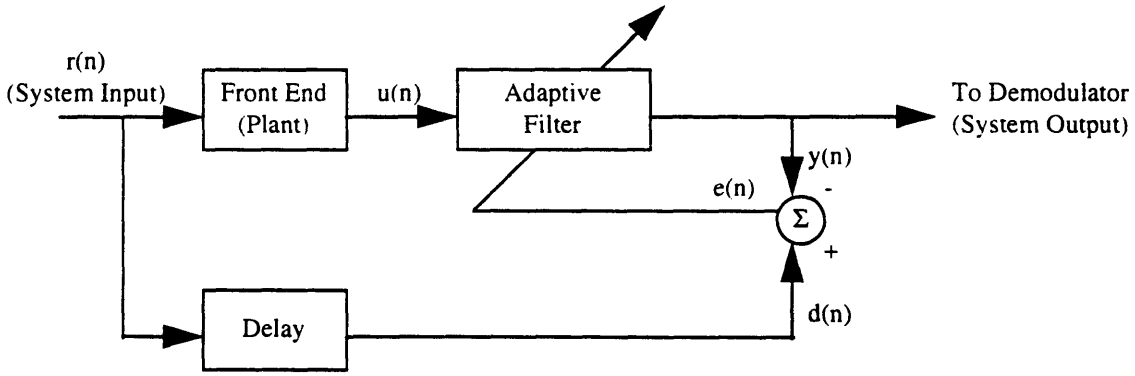

Figure 2.20 Block diagram for adaptive front end equalization. The terms in parenthesis are related to the general inverse modeling structure

This application falls under a general class of adaptive filtering applications called inverse modeling. The block diagram of this application is depicted in Figure 2.20. The function of the adaptive filter is to provide an inverse model that represents the best fit (in some sense) to an unknown noisy plant, which is the front end in our application. Ideally, the inverse model has a transfer function equal to the reciprocal (inverse) of the plant's transfer function. Recall from the front end model shown in Figure 2.3, both the magnitude response and the group delay response show undesirable features. For the former, it is the excessive ripples in the passband, while the non-constant group delay has more to be desired in the latter. The role of the adaptive filter is then to make the overall system response, that of the cascade of the front end and the adaptive filter, without these two undesirable features.

Also note from Figure 2.20, a delayed version of the system input, which is the received signal r(n) for our case, constitutes the desired response d(n) for the adaptive filter. The adaptive filter algorithm, specifically LMS for our case, requires knowledge of the desired response so as to form the error signal needed for the adaptive process to function. In theory, the received data (originating at the transmitter output) is the desired response for adaptive equalization. In practice, however, with the adaptive equalizer located in the receiver, the equalizer is physically separated from the origin of its ideal desired response. A training signal is therefore needed to generate the desired response at least until the tap weights of the adaptive filter have converged. The time period in which the training signal is in use is called, appropriately enough, the training period. And training, in general, must be done before allowing received data signal reach the demodulation stage for high performance.

A widely used training signal consists of a pseudo-noise (PN) sequence with a broad and even power spectrum. We are going to talk more about the properties of PN sequence when we discuss delay-locked loop in the next chapter. Here briefly, the PN sequence has noiselike properties, but it also has a deterministic waveform that repeats periodically. For the generation of a PN sequence, we may use a linear feedback shift register that consists of a number of consecutive two-state memory stages (flip-flops) regulated by a single timing clock [19]. A feedback term, consisting of the modulo-2 sum of the outputs of various memory stages, is applied to the first memory stage of the shift register and thereby prevents it from emptying. With a known training sequence, the adaptive filtering algorithm can adjust the equalizer coefficients corresponding mathematically to searching for the unique minimum of a quadratic error performance surface. The unimodal nature of this surface assures convergence of the algorithm.

2.2.2.1 Software Implementation Issues

The computer simulation of front end equalization is based on a slight modification of Figure 2.20, which is shown in Figure 2.21. As we have discussed before, in order to generate the inverse model of the front end, there has to be a training period, during which the adaptive filter tap weights are allowed to converge. Only after the training is done can the real data be sent through the combination to the rest of the receiver. We shall call the time other than the training period the operational period. The two switches used in Figure 2.21 are serving the purpose of differentiating these two periods. At the beginning of the training period, both switches will be in places as shown in the figure. The input, which is a training signal, to the whole structure will be supplied by a white noise generator available in MATLAB. The ideal front end is an ideal band-pass filter of similar bandwidth as our front model, but with exact unit gain in magnitude response, and constant group delay response in its passband. Our hope is that by the end of the training period, the combined response of the front end cascading with the adaptive filter would resemble that of the ideal front end. The delay after the ideal front end block is used to make sure desired response d(n) and the adaptive filter output y(n) are aligned properly to ensure convergence. After convergence has been reached, both switches will be closed in the direction of their corresponding arrows to initiate the operational period. The input to the system becomes regular data signals, which are the same as we used to generate the results in 2.1.1.4 and 2.1.2.4.

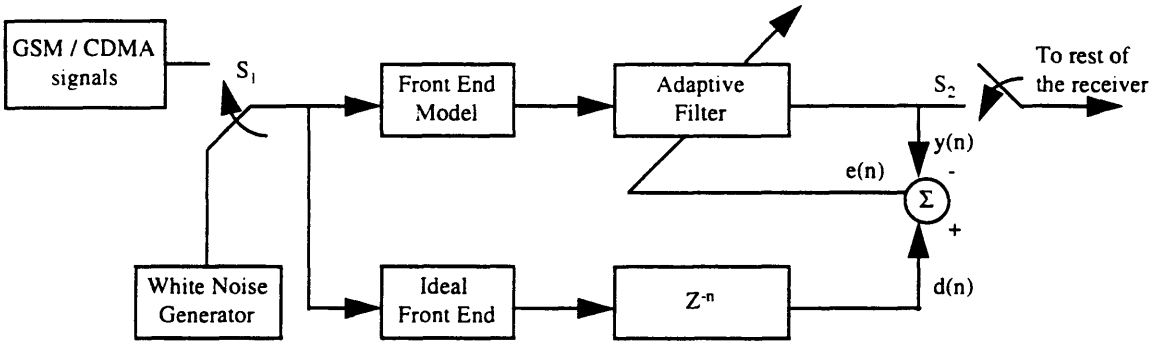

Figure 2.21 Software implementation of front end equalization

2.2.2.2 Equalization Results

To measure the benefit of the equalization process, we are going to compute the PMSRVE values associated with the equalized front end, and compare these with the corresponding values using our original front end model. A set of PMSRVE values are computed for data located at various frequency bands within the front end bandwidth. Just as what we have done for the GSM and CDMA simulation, various frequency locations are achieved by applying different carrier frequencies to the data. For easier interpretation of the results for both GSM and CDMA case, we use the same set of data that is used for its front end study (see 2.1.2 and 2.1.3, respectively).

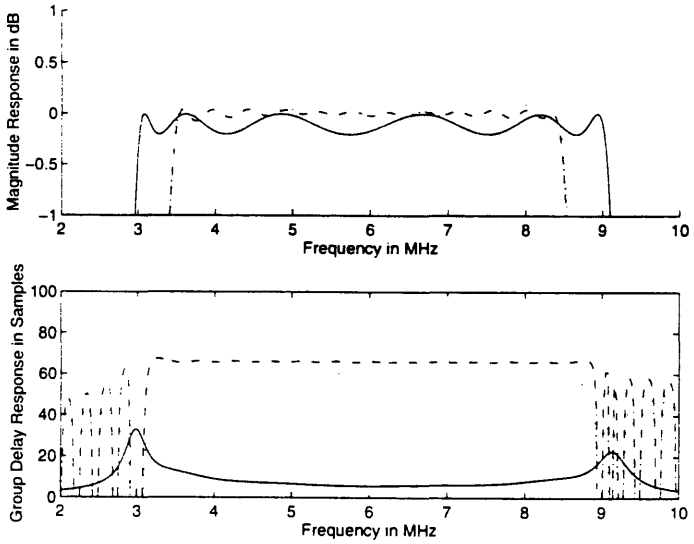

Figure 2.22 Frequency responses: (solid line): front end model, (dotted line): 121-tap adaptive filter at the end of the training period, and (dashed line): equalized front end

First let us look at the frequency domain behavior of the adaptive filter, which is shown in Figure 2.22. Here the solid line represents the frequency response (which is shown in terms of magnitude and group delay response) of our front end model, which is the same as what appeared in Figure 2.3. The dotted line represents the frequency response of a 121-tap adaptive filter at the end of the training period. Since magnitude response has units of dB, we can add the solid line dotted line to get a combined response (of course, we can do the same thing for the group delay

response), which is represented by the dashed line in Figure 2.22. Compare with the original response (solid line), we can easily see that the equalized front end (dashed line) shows improvements in terms of both magnitude and group delay responses. It not only has much smaller ripples in the magnitude response, it also shows flatter, i.e., better, group delay response. Both improvements can help to reduce the front end distortions that we saw in 2.1.1.4 and 2.1.2.4.

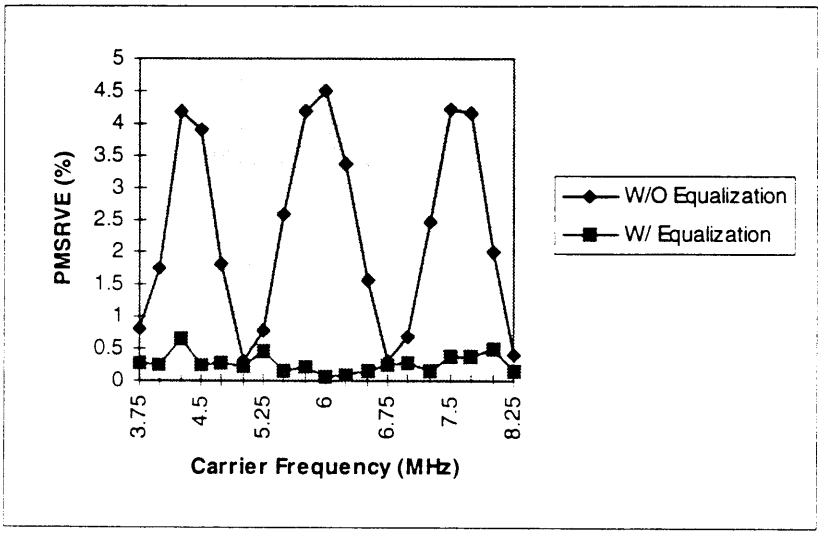

Figure 2.23 Comparison of GSM PMSRVE values with and without equalization. For this particular plot, equalization is achieved using step size of 0.02, and adaptive filter length of 121.

To show this is indeed the case, Figure 2.23 is a plot of two sets of PMSRVE values using equalized and unequalized GSM receiver outputs. Figure 2.24 is a similar plot for CDMA outputs. Adaptive fitlers used to generate thse plots all have tap length of 121, and step size of 0.02. It is evident that for both narrow band (GSM) and wide band (CDMA) signals, equalization process can be applied to reduce distortions stemming from the non-ideal front end response.

The next question comes to mind is whether we can further improve the results by appropriately adjusting adaptive filter design parameters. The answer is yes, but it comes at a price. In our particular case, there are basically two design parameters that can be adjusted. One

is the step size μ used by the LMS algorithm, and the other one is the tap length M associated with the adaptive transversal filter.

Let us first examine the effects of using different μ's. When we were presenting LMS algorithm, we introduced the idea of excess mean-square error $J_{ex}(\infty)$, which represents the price paid for using the adaptive mechanism to control the tap weights in the LMS algorithm in place of a deterministic approach as in the method of steepest descent (of course the immediate benefit of using LMS algorithm is its simplicity). However, we also pointed out that this value is under designer's control. In particular, by assigning a small value to μ the effects of gradient noise on the tap weights can be largely filtered out, which in turn has the effect of reducing the misadjustment. Figure 2.25 plots the mean-square error used to drive the adaptive fitler for various μ's. One can observe a clear trend that as μ becomes smaller, so does the mean-square error. However there is a price paid for this decrease in error in the form of longer convergence time (see Figure 2.26). Also this improvement in mean-square error during adaptation does not translate directly into better performance in terms of PMSRVE values as shown in Figure 2.27. This plot is generated using GSM data, but similar plot is also obtained using CDMA data. Besides the step size of 0.03, all the other choices of μ lead to similar satisfactory PMSRVE results. From implementation point of view, we would like to have a equalizer that has the fastest adaptation time, yet at the same time delivers satisfactory performance in terms of PMSRVE values. Smaller μ means longer adaptation time, yet, at least shown by Figure 2.27, does not necessarily guarantee better performance. The conclusion is that we should stick with the largest μ that delivers satisfactory results.

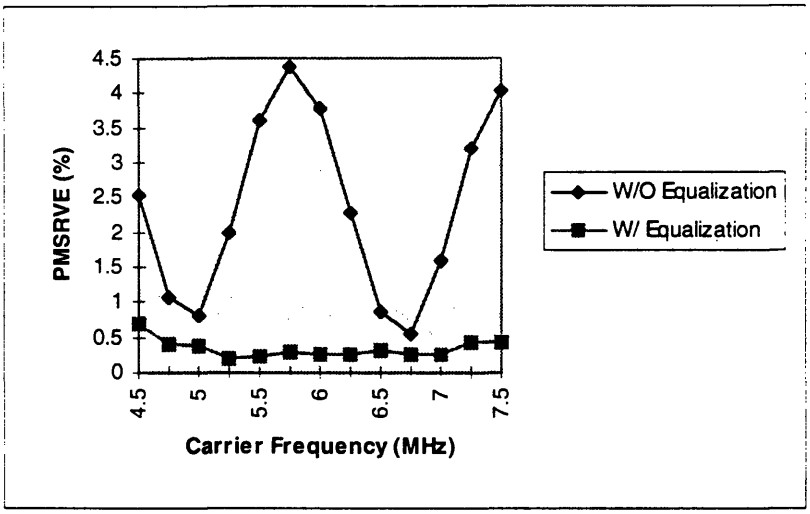

Figure 2.24 Comparison of CDMA PMSRVE values with and without equalization. For this particular plot, equalization is achieved using step size of 0.02, and adaptive filter length of 121.

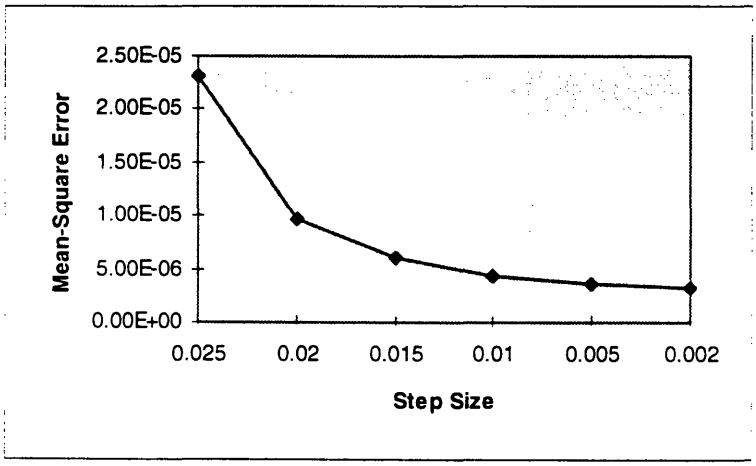

Figure 2.25 LMS Mean-square error vs. step size

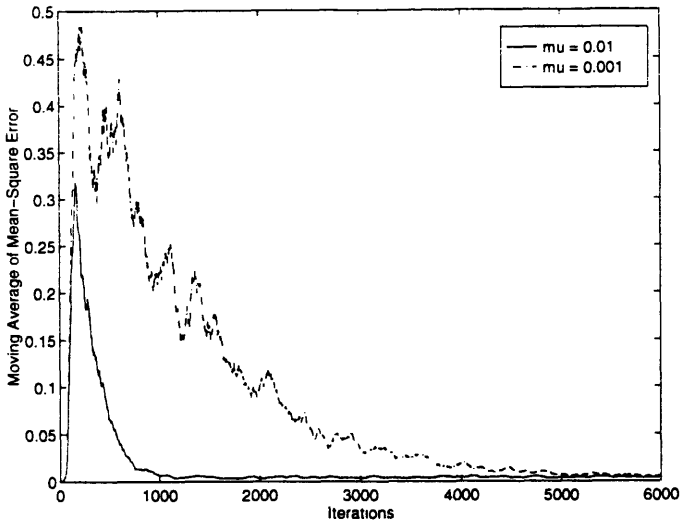

Figure 2.26 LMS learning curve for different step sizes (mu)

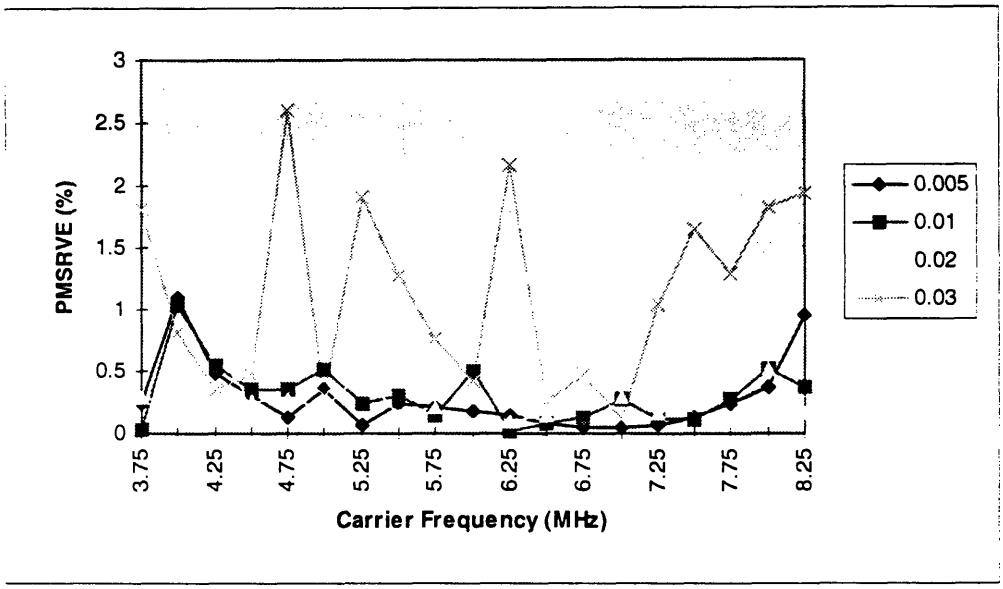

Figure 2.27 PMSRVE vs. carier frequencies for different choices of μ.

Compared to our previous discussion on choices of step size μ, the effect of different tap length M of the transversal filter is more straight forward. Figure 2.28 is a 3-D plot of PMSRVE vs. carrier frequency for different length adaptive filters. Very clearly, using the same step size, longer filter length means better PMSRVE values. Again, there is a clear price paid for better performance. Longer filter also means higher costs of implementation in terms of both hardware

and computational requirements. As we have shown before, LMS, though simpler compare to other algorithms, has number of operations per iteration grow in $O(M)$. Figure 2.29 compares the PMSRVE values for M = 121 with these for M = 171. While there is a clear improvement, but whether it justifies the extra cost involved is still debatable.

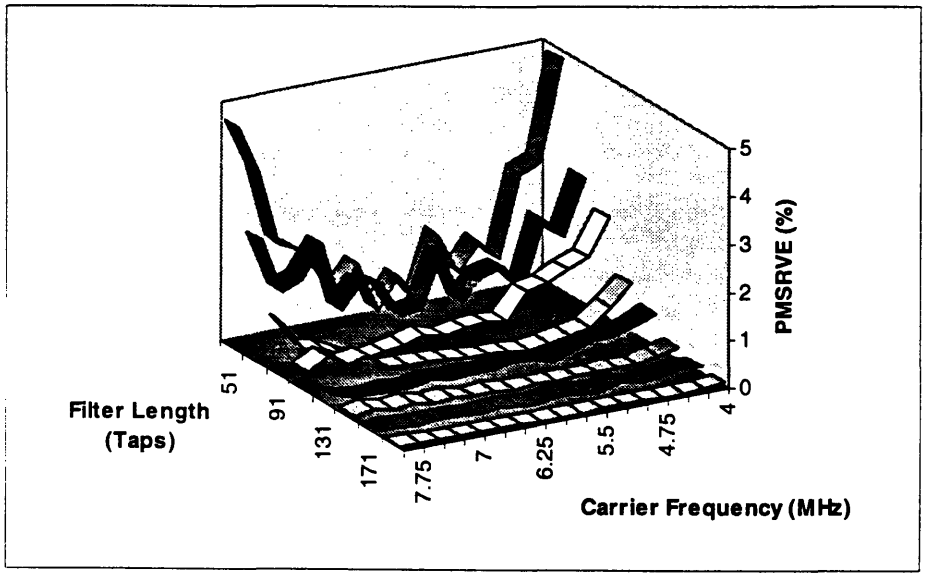

Figure 2.28 Comparison of PMSRVE values for adaptive fitlers of various length (the step size for each case is set to be 0.002)

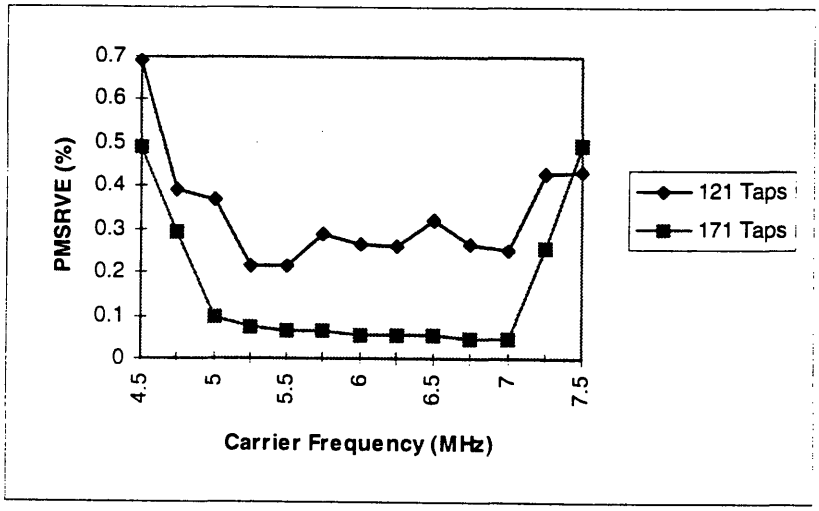

Figure 2.29 Comparison of CDMA PMSRVE values resulting from using two adaptive filters with different tap lengths (step size for both cases are 0.02)

2.3 Conclusions

In this chapter, we first presented a study on the extent of the distortion introduced by a front end model. We then introduced a distortion correction method using linear adaptive filtering, specifically, through the use of a LMS algorithm-driven adaptive transversal filter. Most of the results presented in this chapter are in the form of percentage mean-square-root vector error, or PMSRVE, values. PMSRVE are introduced as a distortion measure because it can be easily understood using a constellation diagram.

Through computer simulations, we showed that by using an adaptive filter as the front end equalizer, most of the distortion can be removed with certain implementation costs. We made an effort to address the question whether it is possible to locate an equalizer structure that incorporates the optimal step size μ and the optimal filter tap length M. An equalizer is optimal when the corresponding PMSRVE is minimal. However, we think while it is possible to reduce PMSRVE by using better design parameters, it is simply not worth the trouble. This trouble comes in the form of implementation costs. One of the easiest way to improve PMSRVE is to increase the tap length M of the adaptive fitler, but clealry this would not only increase the hardware cost, but also the computational burden. Another way of improving the results may be using a smaller step size. But a smaller step size requires longer adaptation time, thus may become impratical for tracking in a nonstationary environment,which is one feature of our front end. Longer adaptation time also translates into larger memory requirement for implementation.

Chapter 3 : Code Synchronization Using Delay-Locked Loops

In this chapter we are going to present the analysis and simulation results of delay-locked loops (DLLs). One motivation for studying DLL in particular is because of its essential role in achieving code synchronization in a direct-sequence spread-spectrum (DS/SS) communication system. Recently DS/SS has received considerable attention because its unique features have made it a viable technology for implementing secure wireless multiple access communications, and high precision navigation systems. Before getting into the details of DLLs, we are going to give an overview on spread-spectrum communication systems, from which we are going to learn the importance of code synchronization. Then the analysis and simulation of both non-coherent and coherent DLLs will be presented.

3.1 Spread-Spectrum Communication System

The major objective of signal design for conventional digital communication systems has been the efficient utilization of transmitter power and channel bandwidth [20] [21]. These are legitimate concerns, and in most communication systems are the concerns of paramount importance. There are situations, however, in which it is necessary for the system to resist external interference, to operate with a low-energy spectral density, to provide multiple-access capability without external control, or to make it difficult for unauthorized receivers to observe the message. In such a situation, it may be appropriate to sacrifice the efficiency aspects of the system in order to enhance these other features. Spread-spectrum techniques offer one way to accomplish this objective.

The definition of spread spectrum may be stated in two parts [22]:

- Spread spectrum is a means of transmission in which the data of interest occupies a bandwidth in excess of the minimum bandwidth necessary to send the data.
- The spectrum spreading is accomplished before transmission through the use of a code that is independent of the data sequence. The same code is used in the receiver (operating in synchronism with the transmitter) to despread the received signal so that the original data be recovered.

Although standard modulation techniques such as frequency modulation (FM) and pulse-code modulation (PCM) do satisfy the first part of the definition, they are not spread-spectrum techniques because they do not satisfy the second part of the definition.

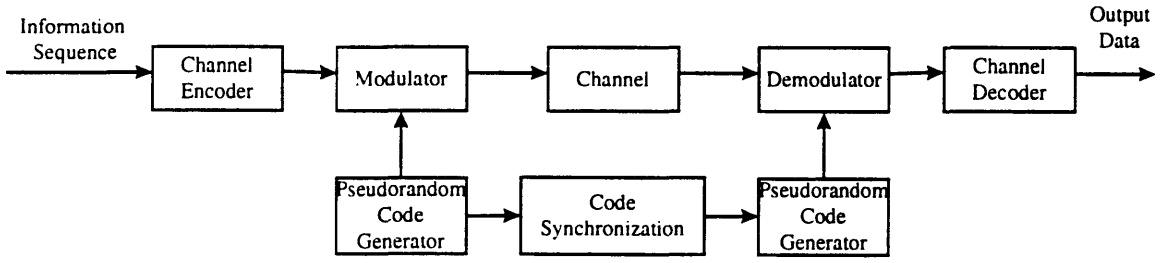

Figure 3.1 Block diagram of a general spread-spectrum communication system

The basic elements of a spread spectrum digital communication system are illustrated in Figure 2.1. We observe that the channel encoder and decoder, the modulator and demodulator are the basic elements of a conventional digital communication system. In addition to these elements, a spread-spectrum system employs two identical code generators, one which interfaces with the modulator at the transmitting end, and the second which interfaces with the demodulator at the receiving end. These two generators produce a pseudo-random or pseudo-noise (PN) binary-valued sequence that is used to spread the transmitted signal in frequency at the modulator and to despread the received signal at the modulator. The overall efficiency of any spread-spectrum communication system is highly dependent on the capacity of the receiver to continuously maintain satisfactory synchronization between the spreading and despreading codes.

There are two general spread spectrum techniques, direct-sequence (DS) and frequency hopping (FH). In a DS system, two stages of modulations are used. First , the incoming data sequence is used to modulate a wide-band code. This code transforms the narrow-band data sequence into a noise-like wide-band signal. The resulting wide-band signal undergoes a second modulation using a phase-shift-keying (PSK) technique. In a FH system, the spectrum of a data-modulated carrier is widened by changing the carrier frequency in a pseudo-random manner. Because of its simplicity and the limitation of today's technology, DS spread-spectrum systems have received most of the attention for developing new communication systems. We will only consider DS systems in this thesis. Both DS and FH systems rely their operation on a noise-like spreading code called a pseudo-random or pseudo-noise (PN) sequence.

3.1.1 Pseudo-Noise (PN) Sequence

A pseudo-noise (PN) sequence is defined as a coded sequence of 1s and 0s with certain autocorrelation properties. The class of sequences used in spread-spectrum communications is usually periodic in that a sequence of 1s and 0s repeats itself exactly with a known period.[5] The maximum-length sequence, a type of cyclic code represents a commonly used periodic PN sequence. Such sequences have long periods and require simple instrumentation in the form of a linear feedback shift register. Indeed, they possess the longest possible period for this method of generation.

---

[5] PN sequences may also be aperiodic. Such sequences are known as Barker sequences, which are too short to be practical use for spectrum spreading.

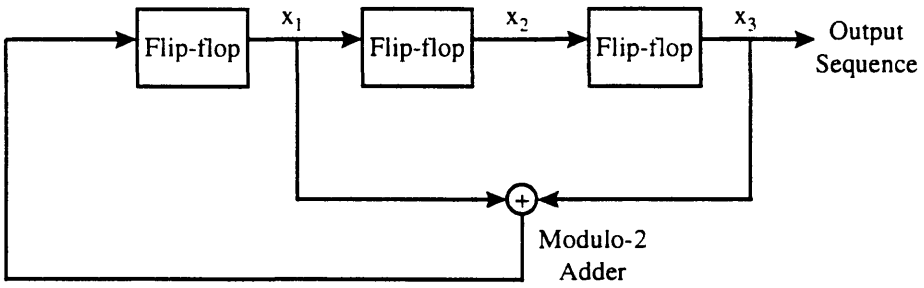

Figure 3.2 Maximum-length sequence generator for the case of m = 3.

A shift register of length m consists of m flip-flops (two-state memory stages) regulated by a single timing clock. At each pulse of the clock, the state of each flip-flop is shifted to the next one down the line. In order to prevent the shift register from emptying by the end of m clock pulses, a logical function of the states of the m flip-flops is used to compute a feedback term, which is then supplied to the input of the first flip-flop. In a feedback shift register of the linear type, the feedback function is obtained using modulo-2 addition of the outputs of the various flip-flops. This operation is illustrated in Figure 3.2 for the case of m = 3. Representing the states of the three flip-flops are x_1, x_2, and x_3. For the case shown in the figure, the feedback function is equal to the modulo-2 sum of x_1 and x_3. A maximum-length sequence so generated is always periodic with a period of

$$N = 2^m - 1 \qquad \text{(Equation 3.1)}$$

where m is the length of the shift register.

Maximum-length sequence has many of the properties possessed by a truly random binary sequence. A random binary sequence is a sequence in which the presence of a binary symbol 1 or 0 is equally probable. Some important properties of maximum-length sequences are [23]:

- Balance Property: in each period of a maximum-length sequence, the number of 1s is always one more than the number of 0s.

- Run Properties: Among the runs of 1s and 0s in each period of a maximum-length sequence, one-half the runs of each kind are of length one, one-fourth are of length two, one-eighth are of length three, and so on as long as these fractions represent meaningful numbers of runs.
- Correlation property: The autocorrelation function of a maximum-length sequence is periodic and binary valued.

By definition, the autocorrelation sequence of a binary sequence $\{c_n\}$ equals

$$R_c(k) = \frac{1}{N}\sum_{n=1}^{N} c_n c_{n-k} \qquad \text{(Equation 3.2)}$$

where N is the length or period of the sequence, and k is the lag of the autocorrelation sequence. For a maximum-length sequence of length N, the autocorrelation sequence is periodic with period and two-valued, as shown by

$$R_c(k) = \begin{cases} 1 & k = lN \\ -\frac{1}{N} & \text{otherwise} \end{cases} \qquad \text{(Equation 3.3)}$$

where l is any integer. When the length N is infinitely large, the autocorrelation sequence $R_c(k)$ approaches that of a completely random binary sequence.

3.1.2 Direct-Sequence Spread-Spectrum System

In direct-sequence systems, spreading is achieved directly by modulating the data signal d(t) by a wide-band PN signal c(t). For this operation to work, both sequences are represented in their polar forms, that is, in terms of two levels equal in amplitude and opposite in polarity, e.g., -1 and +1. We know from Fourier transform theory that multiplication of two unrelated signals produces a signal whose spectrum equals the convolution of the spectra of the two component signals. Thus, if the data signal d(t) is narrow-band and the PN code signal c(t) is wide-band, the product signal m(t) will have a spectrum that is nearly the same as the PN sequence (see Figure 3.3).

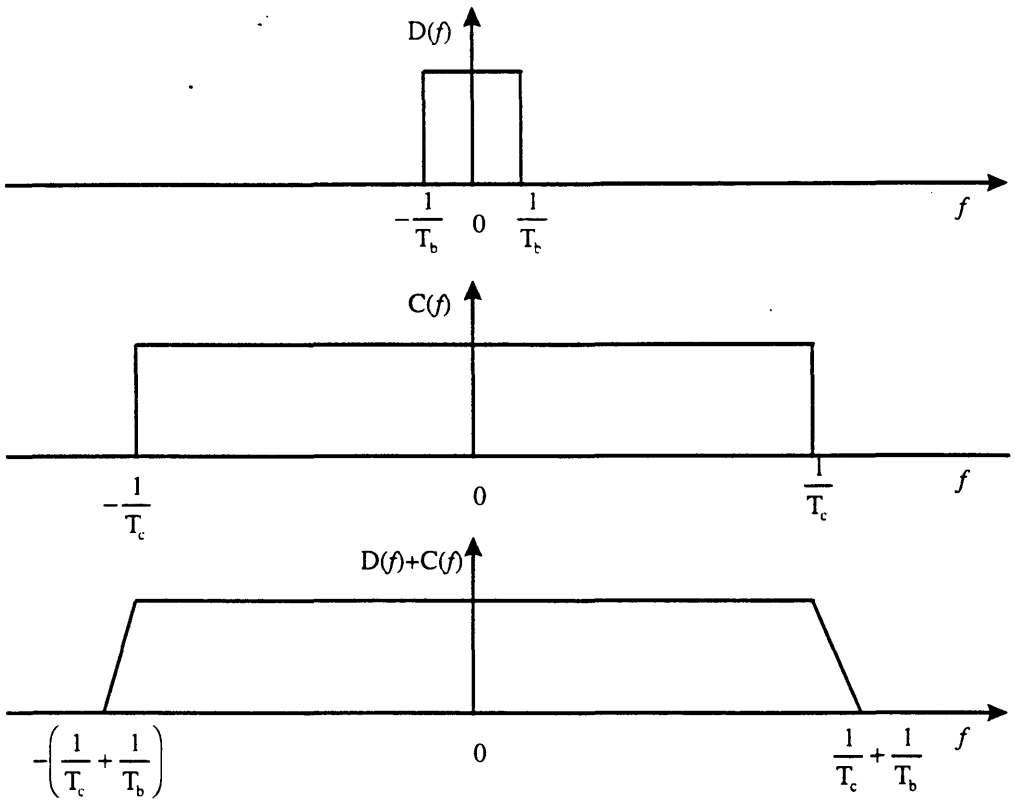

Figure 3.3 Convolution of spectra of the data signal d(t) with that of the PN code signal c(t)

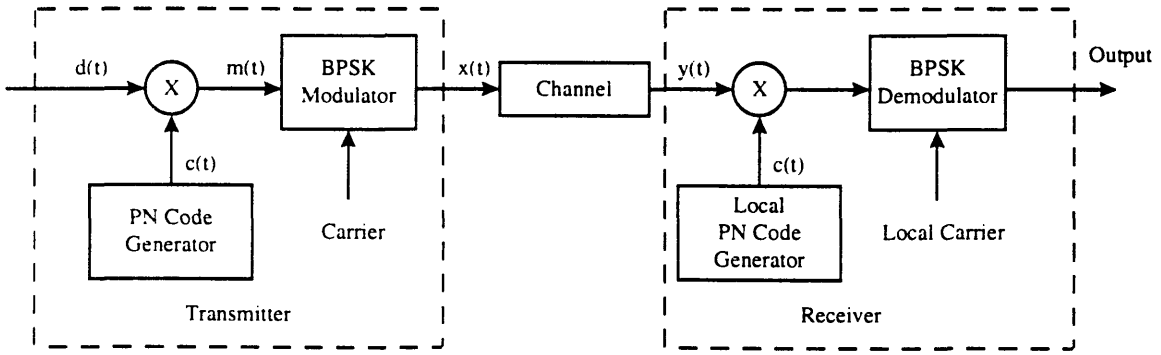

Figure 3.4 Non-coherent direct-sequence spread-spectrum system using binary phase-shift-keying (BPSK)

Figure 3.4 illustrates a block diagram of a direct-sequence system that uses binary phase-shift-keying (BPSK), based on which our analysis and simulation of DLLs are built. Let us take a closer look on its operation. We are going to assume that the binary data sequence d_n has a rate of

R bits per second (or equivalently, a bit interval of $T_b = 1/R$ seconds), and PN sequence c_n has a rate of W chips per second (or equivalently, a chip interval of $T_c = 1/W$ seconds), where $W >> R$. In practice, T_b is usually an integer multiple of T_c, such that $T_b = NT_c$, where N is an integer and is often called the processing gain of a spread-spectrum system [22]. We can express the data sequence d(t) as

$$d(t) = \sum_{n=-\infty}^{\infty} d_n g(t - nT_b) \qquad \text{(Equation 3.4)}$$

where g (t) is a rectangular pulse of duration T_b. This signal is multiplied by the PN code signal c(t), which can be expressed as

$$c(t) = \sum_{n=-\infty}^{\infty} c_n p(t - nT_c) \qquad \text{(Equation 3.5)}$$

where p(t) is a rectangular pulse of duration T_c,.

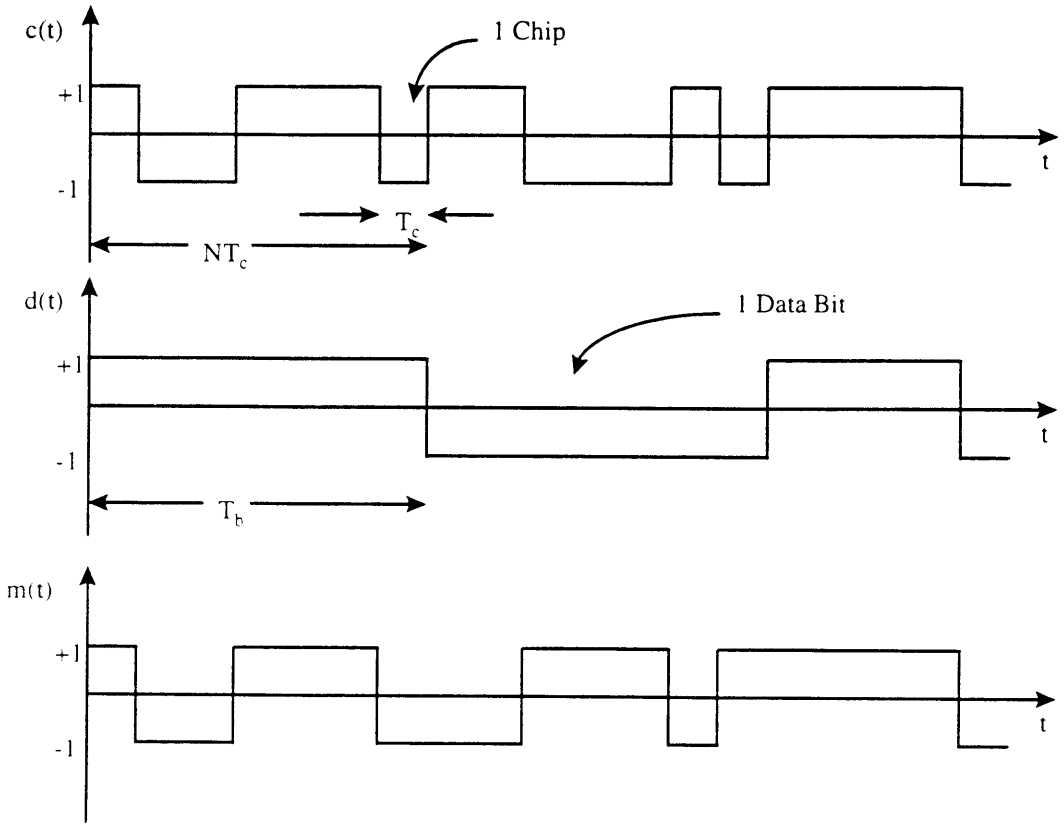

Figure 3.5 PN code modulation

The product signal m(t) (an example of m(t) is shown in Figure 3.5) is then used to amplitude modulate a carrier $A_c\cos 2\pi f_c t$, as for the case of BPSK, to generate the resulting signal

$$x(t) = A_c d(t)c(t)\cos 2\pi f_c t \quad \text{(Equation 3.6)}$$

where f_c is the carrier frequency.

The demodulation of the signal consists of two steps, code despreading and carrier demodulation. In Figure 3.4 we have used a non-coherent spread-spectrum receiver in that code despreading is done prior to carrier demodulation. The code spreading process involves the multiplication of the received signal y(t) by a locally generated replica of the original PN code signal c(t). In order for despreading to be successful, the locally generated PN signal must be synchronized to the PN signal contained in the received signal. Thus we have

$$A_c d(t)c^2(t)\cos 2\pi f_c t = A_c d(t)\cos 2\pi f_c t \quad \text{(Equation 3.7)}$$

since $c^2(t) = 1$ for all t. The resulting signal $A_c d(t)\cos 2\pi f_c t$ occupies a bandwidth (approximately) of R Hz, which is the bandwidth of the original data signal. Therefore, the carrier demodulation for the despread signal is simply any conventional crosscorrelator or matched filter. Non-coherent receivers have received most of the attention since the spread signal-to-noise ratio (SNR) is typically too low to allow for carrier synchronization prior to code synchronization [24] [25].

3.1.3 Code Synchronization

The efficiency of any spread-spectrum communication system is highly dependent on the capacity of the receiver to continuously maintain satisfactory synchronization between the received (spreading) and the locally generated (despreading) codes. Code synchronization is usually achieved in two steps: initially, a coarse alignment of the two PN signals is produced to within a small (typically less than a fraction of a chip) relative timing offset. This process of bringing the two codes into coarse alignment is referred to as PN code acquisition. Once the incoming PN code has been acquired, a fine synchronization system takes over and continuously maintains the

best possible waveform alignment by means of a closed loop operation. This process of maintaining the two codes in fine synchronism is referred to as PN code tracking.

A typical PN code synchronizer for a direct sequence spread spectrum system is shown in Figure 3.6. During code acquisition, the acquisition unit continually adjusts the phase of the local code until the incoming and local codes are aligned, so that the code phase error $\varepsilon(t)$ is within the permissible range $(\varepsilon_{min}, \varepsilon_{max})$. The code phase error $\varepsilon(t) = [\tau(t) - \tau'(t)]/T_c$ is defined as the normalized phase difference between the incoming and local codes, where $\tau(t)$ and $\tau'(t)$ are the absolute phases of the incoming and local codes, respectively, and T_c is the chip duration. For a code tracking loop, the permissible range $(\varepsilon_{min}, \varepsilon_{max})$ is usually the range of the discriminator characteristic, which is also called the *S* curve of the code tracking loop, i.e., the range for which the *S* curve is not zero. We will derive the *S* curve for the DLL in the next section. Whenever the code phase error is within this permissible range, there is a probability that the lock detector will declare that the synchronizer is in-lock and switch the synchronization to the code tracking unit to obtain fine alignment of the chip boundaries. During the code tracking process the code phase error might exceed the permissible range $(\varepsilon_{min}, \varepsilon_{max})$ because of the presence of channel dynamics and system noise, and cause the code synchronizer to be out-of-lock. In this case, the lock detector will trigger a reacquisition process and switch the PN code synchronizer back to the code acquisition unit.

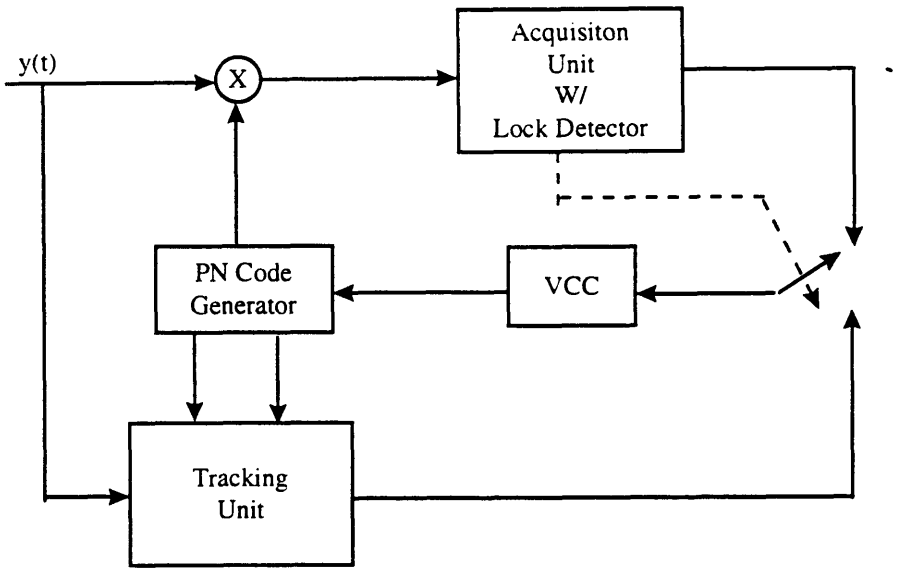

Figure 3.6 A typical PN code synchronizer for direct-sequence spread spectrum systems.

From the above discussion, the PN code synchronizer follows a combined tracking/reacquisition process after initial code acquisition. Although PN acquisition is an extremely important problem, e.g., the code must customarily be acquired in as short a time as possible, the development of closed loop techniques for accurate PN tracking plays an equally important role in supporting the acquisition process once the code has been acquired. As such, the optimum design and true assessment of the performance of the PN tracking loop is an essential component of the overall receiver design. In this thesis, we would only consider code tracking process. The code tracking process can be treated separately from the code reacquisition process, if one is only interested in the stationary behavior of the code tracking loops [24]. In effect, the individual code tracking processes will be treated as being statistically identical. In the rest of the chapter, we will present the analysis and simulation of delay-locked loops (DLLs), which are commonly used for code tracking in direct-sequence spread spectrum systems.

3.2 Analysis and Simulations of Non-coherent DLLs

By researching past literature, we find that over the years there have been predominantly two PN tracking loop configurations that have been proposed and analyzed: the delay-locked loops (DLL) [24]-[28], and tau-dither loop (TDL) or time-shared loop [29] [30]. We are going to focus our attention on the DLL since the results obtained can be readily applied to the analysis of TDL [25]. Either of these configurations can be operated in a coherent or non-coherent mode depending on the system application. We are only going to consider non-coherent DLLs because non-coherent loops are more often employed in spread spectrum applications, because the spread energy-to-noise ratio is typically too low to obtain carrier recovery before code synchronization. In the remainder of this chapter, we are first going to present a mathematical model for a non-coherent DLL. Specifically, we are going to derive its discriminator characteristic, or the S-curve, and its equivalent base-band transfer function. Then simulation results of a DLL using BPSK inputs will be shown. And finally, an optimization in terms of tracking jitters will be discussed.

3.2.1 Non-Coherent Delay-Locked Loop

A non-coherent delay-locked loop with an early-late gate time-separation (offset) of $2\delta T_c$ seconds is shown in Figure 3.7. The input signal y(t) is cross-correlated with advanced and delayed versions of the local PN code generator sequence. The results of these cross-correlation operations are then band-pass filtered, square-law envelope detected, and compared to produce an error (discriminator) characteristic. The loop is closed by applying this difference output to a loop filter and a voltage controlled oscillator (VCO) that drives the PN code generator from which the PN reference sequence used for despreading is obtained.

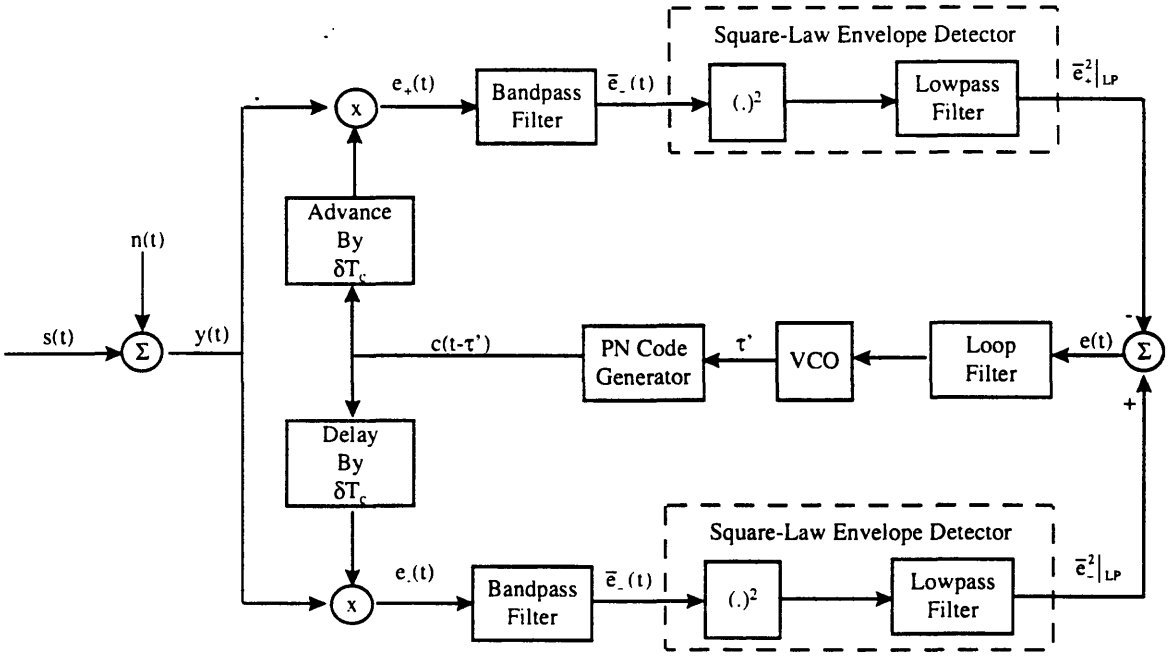

Figure 3.7 A non-coherent delay-locked loop

The advance (and delay) interval δT_c or, equivalently, the corrector spacing is restricted to a range of $\delta \leq 1$. More conveniently, we shall define $\delta = 1/N$ where N is any integer larger than unity. Thus, the advanced and delay PN signals are $2T_c/N$ apart. When the advance (and delay interval is equal to one-half of a PN code chip, i.e., N = 2, the loop is commonly referred to as a "one-delta" loop.

3.2.1.1 DLL Discriminator Characteristic

In mathematical terms, the above statements are expressed as follows. The received signal y(t) is the sum of signal s(t) plus additive noise n(t) where

$$s(t) = \sqrt{2S}c(t-\tau)d(t-\tau)\cos(\omega_0 t + \theta) \qquad \text{(Equation 3.7)}$$

and n(t) has the band-pass representation

$$n(t) = \sqrt{2}\left[n_I(t)\cos(\omega_0 t + \theta) - n_Q \sin(\omega_0 t + \theta)\right] \qquad \text{(Equation 3.8)}$$

In Equation 3.7, S denotes the average signal power, $c(t-\tau)$ is the received PN code signal with transmission delay τ, $d(t-\tau)$ is the digital data modulation in the presence of the same delay, and ω_0 and θ are the carrier radian frequency and phase, respectively. The noise processes $i_n(t)$ and $N_Q(t)$ are approximately statistically independent, stationary, low-pass white Gossip noise processes with single-sided noise spectral density N_0 W/Hz (see [31]) and one-sided bandwidth B_N much smaller than the carrier frequency.

The advanced and delayed versions of the local PN code signal are $c(t-\tau'+\delta T_c)$ and $c(t-\tau'-\delta T_c)$, respectively, where τ' denotes the DLL's estimate of the transmission delay τ. Upon cross correlating with the incoming waveform and filtering through the arm band-pass filter , the outputs $e_+(t)$ and $e_-(t)$ in Figure 3.7 become

$$e_{\pm}(t) = K_m c(t-\tau'\pm\delta T_c)[\sqrt{2S}d(t-\tau)c(t-\tau)\cos(\omega_0 t+\theta)+n(t)] \qquad \text{(Equation 3.9)}$$

with K_m denoting the multiplier gain. The product $c(t-\tau)c(t-\tau'\pm\delta T_c)$ can be written as the sum of an ensemble average or autocorrelation function R_c, which depends only on the time difference $(t-\tau'\pm\delta T_c)-(t-\tau)=\tau-\tau'\pm\delta T_c$, and a residual process which is conventionally called the "code self-noise." [24] [27] For the case of PN sequence, (using Equation 3.2 and Equation 3.3) R_c takes the following form

$$R_c(\tau-\tau'\pm\delta T_c) = E[c(t-\tau)c(t-\tau'\pm\delta T_c)] = \begin{cases} 1-|\tau-\tau'\pm\delta T_c| & |\tau-\tau'\pm\delta T_c| \le T_c \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Equation 3.10)}$$

Let ε denote the normalized transmission delay error $(\tau-\tau')/T_c$, then Equation 3.10 can be rewritten in normalized form

$$R_c(\varepsilon\pm\delta) = \begin{cases} 1-|\varepsilon\pm\delta| & |\varepsilon\pm\delta| \le 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Equation 3.11)}$$

Figure 3.8 illustrates the autocorrelation functions for both the advanced and delayed PN codes.

The exact statistical nature of the code self-noise process is a rather complicated issue. It has been shown [27] that, to a first approximation, self-noise can be neglected for $\delta = 1$ and

coherent loops. The same approximation will be employed here in order to keep the analysis tractable.

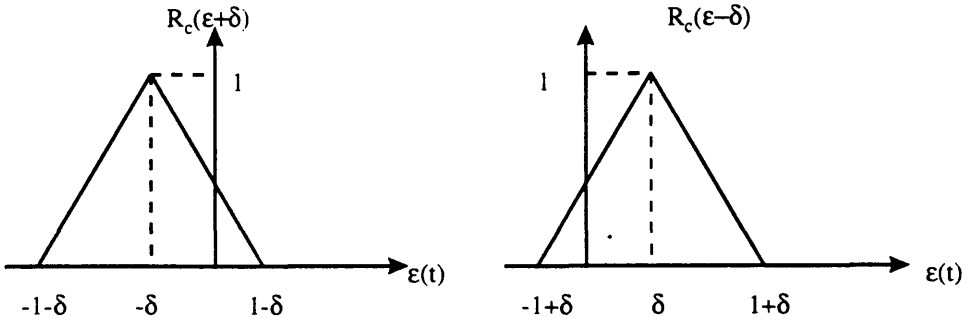

Figure 3.8 Autocorrelation functions of the δ–advanced and δ–delayed PN codes

With this simplification inserted into Equation 3.9, the bandpassed outputs $\bar{e}_{\pm}(t)$ can be expressed as

$$\bar{e}_{\pm}(t) = \sqrt{2S}K_m\bar{d}(t-\tau)R_c(\varepsilon \pm \delta)\cos(\omega_0 t + \theta) + K_m\overline{c(t-\tau'\pm\delta T_c)n(t)} \quad \text{(Equation 3.12)}$$

where the overhead bar indicates filtering. Here the filtering effect on R_c has been ignored since the autocorrelation function is not drastically dependent on time t (its dependence is only implicit through the assumed slowly varying error process $\tau(t)$-$\tau'(t)$) [24]. Now ignoring the second harmonic terms produced by the square-law envelope detector, we find that the input to the loop filter is given by

$$e(t) = \bar{e}_-^2\big|_{LP} - \bar{e}_+^2\big|_{LP} = SK_m^2\bar{d}^2(t-\tau)D(\varepsilon,\delta) + K_m^2(\text{noise}-\text{term}) \quad \text{(Equation 3.13)}$$

where[6]

---

[6] This is reprinted from (32) in [24]. The first three rows are for $\delta \le 1/2$, the next three are for $\delta > 1/2$.

$$D(\varepsilon,\delta) \equiv R_c^2(\varepsilon-\delta) - R_c^2(\varepsilon+\delta) = \begin{cases} 1+(\varepsilon-\delta)(\varepsilon-\delta-2) & 1-\delta \le \varepsilon \le 1+\delta \\ 4\delta(1-\varepsilon) & \delta \le \varepsilon \le 1-\delta \\ 4\varepsilon(1-\delta) & 0 \le \varepsilon \le \delta \\ 1+(\varepsilon-\delta)(\varepsilon-\delta-2) & \delta \le \varepsilon \le 1+\delta \\ 1+(\varepsilon-\delta)(\varepsilon-\delta+2) & 1-\delta \le \varepsilon \le \delta \\ 4\varepsilon(1-\delta) & 0 \le \varepsilon \le 1-\delta \\ D(-\varepsilon,\delta) = -D(\varepsilon,\delta) & 0 \le \varepsilon \le 1+\delta \\ 0 & \text{otherwise} \end{cases} \quad \text{(Equation 3.14)}$$

is called the loop S-curve or discriminator characteristic. For a PN sequence $D(\varepsilon, \delta)$ can be considered aperiodic since PN code period is much larger than the tracking range, which is $[-T_c, T_c]$. This is one major difference between a DLL and a phase-locked loop (PLL). Figure 3.9 shows loop S-curves for various values of δ. Clearly the gain at the origin $g_0 = dD(\varepsilon, \delta)/d\varepsilon|_{\varepsilon=0}$ dependent strictly on the choice of δ. Together with multiplier gain K_m and K_{VCO}, they constitute some form of "effective" loop gain, which we are going to discuss in detail when we cover the linear model of the DLL.

Another item in Equation 3.13 that is worth noting is $\overline{d}^2(t-\tau)$. Ideally, in the absence of filtering, $d^2(t-\tau)$ would equal 1 since we assumed that the data symbols use binary not-return-to-zero (NRZ) polar format.[7] Because of filtering, it actually fluctuates around its mean value

$$D_2 \equiv E[\overline{d}^2(t-\tau)] = \int_{-\infty}^{\infty} S_d(j\omega)\left|H_l(j\omega)\right|^2 d\omega \quad \text{(Equation 3.15)}$$

where $S_d(j\omega)$ is the power spectral density (PSD) of the data sequence and $H_l(f)$ is the low-pass equivalent transfer function of each of the identical arm band-pass filters. It has been previously shown [25] that the residual fluctuation can be ignored with negligible error if the loop bandwidth B_L is much less than the data symbol rate $1/T_b$. Thus we can write e(t) as

---

[7] Binary NRZ polar format means that 1 is represented by 1V, and 0 is represented by -1V.

$$e(t) = \bar{e}_-^2\big|_{LP} - \bar{e}_+^2\big|_{LP} = SK_m^2 D_2 D(\varepsilon,\delta) + K_m^2 n_\varepsilon(t,\varepsilon) \qquad \text{(Equation 3.16)}$$

where $n_e(t, \varepsilon)$ is the equivalent noise term

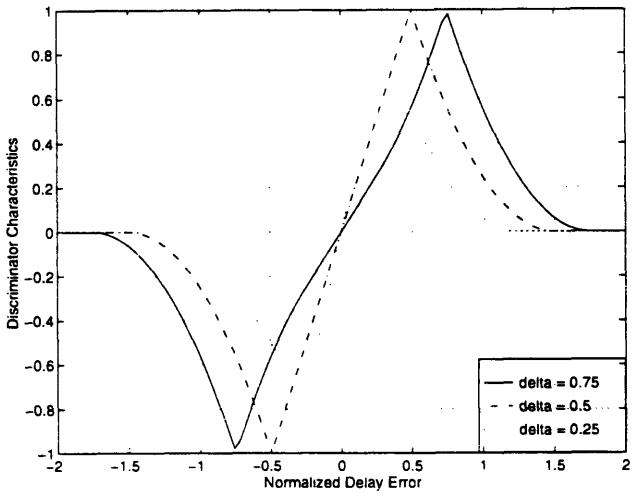

Figure 3.9 Loop S-curve as a function of ε for different values of the offset δ.

3.2.1.2 Linear DLL Model

As shown in Figure 3.7, the waveform e(t) is the input to the loop filter F(s), whose output drives the voltage-controlled oscillator (VCO) to produce the instantaneous delay estimate τ'. Let $K = K_m^2 K_{VCO}$, then the relationship between $\tau'(t)$ and e(t) can be expressed as[8]

$$\tau'(t) = KT_c \frac{F(p)}{p} e(t) = KT_c \frac{F(p)}{p}\left[SD_2 D(\varepsilon,\delta) + n_e(t,\varepsilon)\right] \qquad \text{(Equation 3.17)}$$

where we have used the result from Equation 3.16. Since normalized delay error $\varepsilon(t)$ is defined as

---

[8] (See pp. 73 of [31]) Here we have used the Heaviside operator $p \equiv d/dt$. In general, if the input to a linear filter with transfer function F(s) is i(t), then the time domain representation of the differential equation which relates the output o(t) to the input is written compactly as o(t) = F(p)i(t).

$$\varepsilon(t) = (\tau - \tau')/T_c$$

Equation 3.17 is equivalent to

$$\varepsilon(t) = \frac{\tau(t)}{T_c} - \frac{KF(p)}{p}\left[SD_2D(\varepsilon,\delta) + n_\varepsilon(t,\varepsilon)\right] \quad \text{(Equation 3.18)}$$

We can illustrate Equation 3.18 using a block diagram shown in Figure 3.10. For now, we are going to analyze the loop ignoring the noise term. Noise effect will be discussed in the *Optimization* section towards the end of this chapter.

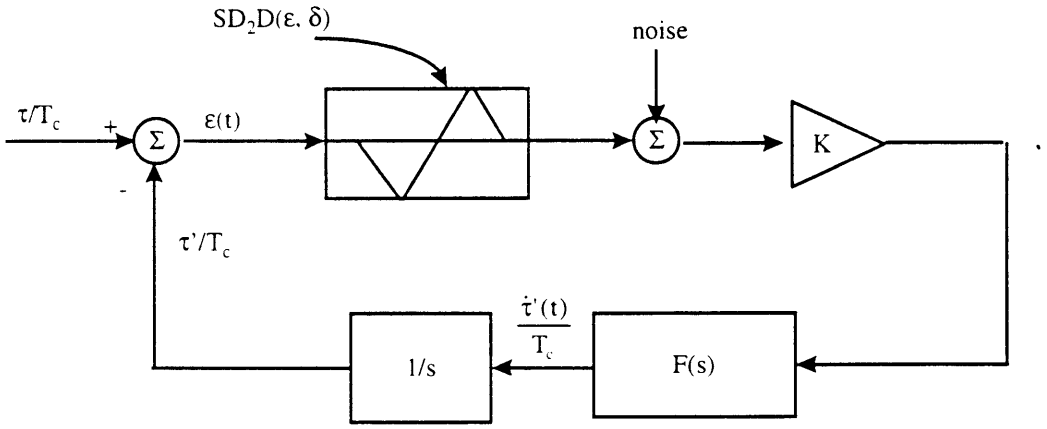

Figure 3.10 Equivalent model of DLL

As evidenced by Equation 3.14, a region exists around $\varepsilon = 0$ (whose extent depends on δ), where $D(\varepsilon, \delta)$ is linear in ε with slope $g_0 = dD(\varepsilon, \delta)/dt\,|_{\varepsilon=0} = 4(1 - \delta)$. The linear analysis stems from the fact that, for high signal-to-noise ratio (SNR), the error $\varepsilon(t)$ will be small most of the time, fluctuating within the aforementioned linear region. Hence, the discriminator characteristic shown in Figure 3.10 can be replaced by a simple gain factor g_0. With noise-term ignored, we arrive at a linear model shown in Figure 3.11.

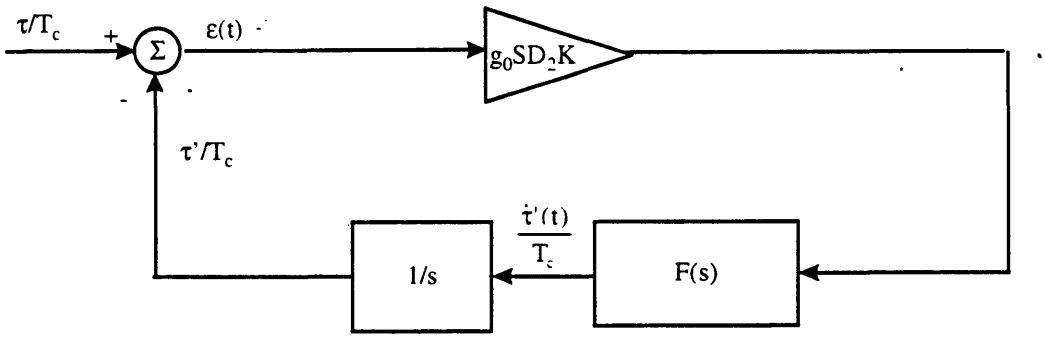

Figure 3.11 Noise-free linearized model of DLL

For the loop shown above, if we define $\hat{\tau} = \tau/T_c$ and $\hat{\tau}' = \tau/T_c$, we can use the Black's formula to write the closed-loop transfer function as

$$H(s) = \frac{\hat{T}'(s)}{\hat{T}(s)} = \frac{GF(s)}{s + GF(s)} \Leftrightarrow H(z) = H(s)\Big|_{s=1-z^{-1}} \qquad \text{(Equation 3.19)}$$

where $G = g_0SD_2K$, and we have made clear that impulse invariance (or bilinear transformation) method can be used to easily convert the system function to the discrete time domain. Notice that this transfer function is identical to the one that describes the linear model of a noise-free phase-locked loop (PLL), understanding that tracking time delay, in some sense, is equivalent to phase tracking.

3.2.1.3 First-Order DLL

First -order DLL means that the loop filter F(s) is 1. The loop transfer function can be written as

$$H(s) = \frac{G}{s+G} \qquad \text{(Equation 3.20)}$$

We can also write the transfer function between the delay error $\varepsilon(t)$ and the input $\hat{\tau}(t)$ as

$$\frac{E(s)}{\hat{T}(s)} = 1 - H(s) = \frac{1}{1+G/s} \qquad \text{(Equation 3.21)}$$

This transfer function, along with the final value theorem of Laplace transforms, can be used to determine the steady-state error response. The final value theorem states that, when the limit for $t \to \infty$ exists,

$$\lim_{t\to\infty}\varepsilon(t) = \lim_{s\to 0} sE(s) \qquad \text{(Equation 3.22)}$$

If the input is a step function representing a constant delay offset Δ, then $\hat{T}(s)$ is simply Δ/s. From Equation 3.22, we then get

$$\lim_{t\to\infty}\varepsilon(t) = \lim_{s\to 0}\frac{\Delta}{1+G/s} = 0 \qquad \text{(Equation 3.23)}$$

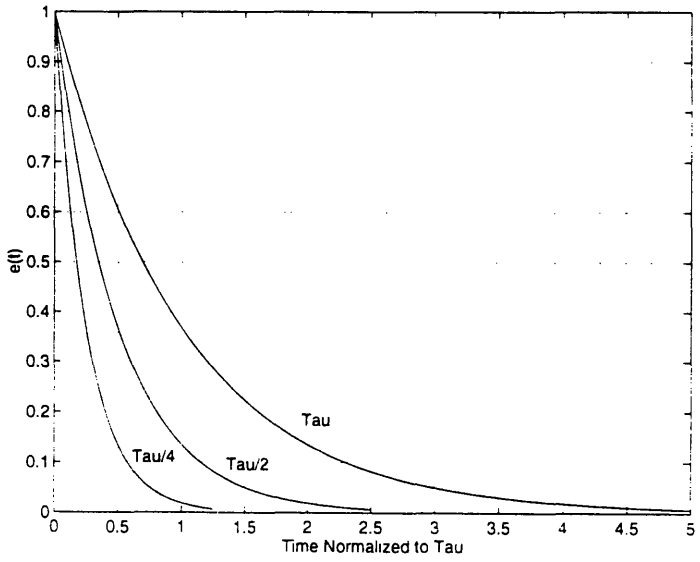

Figure 3.12 Transient response of a first-order DLL to a delay step function

Thus, the first-order DLL will reduce any delay error to zero if the input is a simple delay offset. While it is important to know the steady-state error, it is sometimes useful to look at its time evolution, or its transient response. For the first order loop, with the help of Equation 3.21, it is rather straight forward. Multiplying both sides of the equation by Δ/s, we get

$$E(s) = \frac{\Delta}{s+G} \qquad \text{(Equation 3.24)}$$

After performing inverse Laplace transform, we find that $\varepsilon(t)$ is a decaying exponential with a time constant of 1/G. Figure 3.12 shows transient response for three different values of G. The time axis on the plot is normalized to the largest time constant. It might be tempting to conclude that G should be set as large as possible, since larger G corresponds to faster acquisition. It is a wrong conclusion because we have ignored the effect of noise so far in the analysis. The *Optimization* section will demonstrate the fact that the error variance is directly proportional to G.

3.2.1.4 Second-Order DLL

In a spread-spectrum communication system, there often exists a PN code rate offset between the transmitter and the receiver. This might be caused by a discrepancy in the respective VCO clock rate, or it could be the result of the Doppler effect. Code rate offset is similar to the frequency step input used to study the transient response of PLLs. We can express the input as

$$\hat{T}(s) = \frac{\dot{\Delta}}{s^2} \qquad \text{(Equation 3.25)}$$

where $\dot{\Delta}$ is the magnitude of the code rate offset. Substituting Equation 3.25 into Equation 3.21, and using the final value theorem, we arrive with the following

$$\lim_{t\to\infty}\varepsilon(t) = \lim_{s\to 0}\frac{\dot{\Delta}}{s + GF(s)} = \frac{\dot{\Delta}}{GF(0)} \qquad \text{(Equation 3.26)}$$

It is clear that in order to drive the steady-state error to zero, F(0) must be large, which is not the case for the first-order loop. Second- or higher order loops must therefore be used to keep the delay error small whenever there is a code rate mismatch. Specifically, we need F(s) to have at least one pole at the origin, which corresponds to s = 0.

In order to satisfy this requirement, F(s) can simply be an integrator such as

$$F(s) = \frac{1 + T_2 s}{T_1 s} \qquad \text{(Equation 3.27)}$$

Now the second-order loop transfer function can be written in the form of [31]

$$H(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad \text{(Equation 3.28)}$$

where $\omega_n = \sqrt{\left(\frac{G}{T_1}\right)}$ is called the natural frequency and $\zeta = \frac{T_2\omega_n}{2}$ is the loop damping ratio. One nice thing about this particular form of F(s) is that for positive T_1 and T_2, the loop is unconditionally stable [33]. Figure 3.13 is a plot of the magnitude response of the second-order loop described above for various damping ratios. Figure 3.14 is a plot of its transient response to the input shown in Equation 3.25.

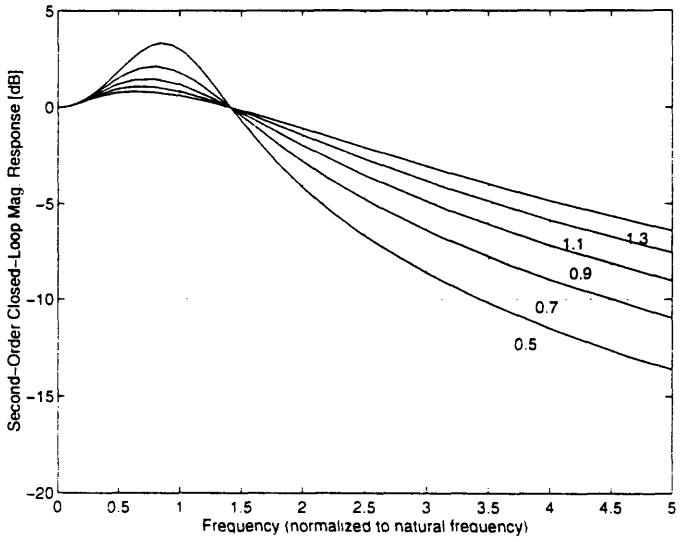

Figure 3.13 Magnitude response of a perfect second-order loop for different damping ratios

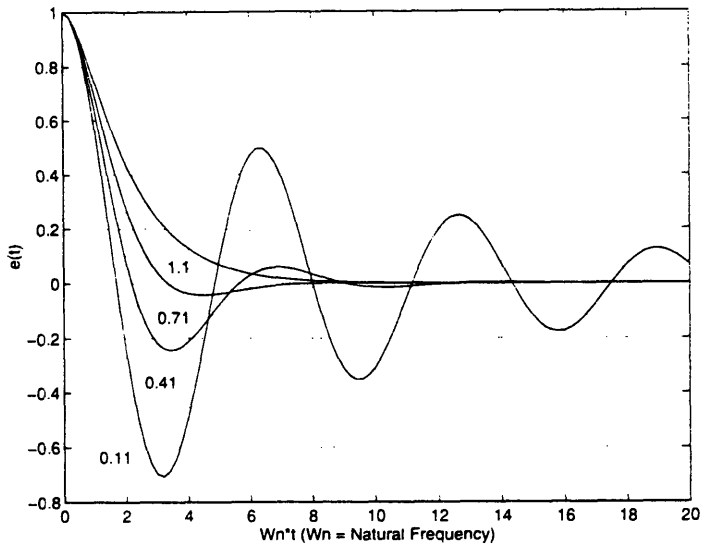

Figure 3.14 Transient response to a constant code rate mismatch.

3.2.2 DLL Simulation

In this section, we are going to present the computer simulation of a first-order non-coherent delay-locked loop. First, we are going to give an overview of the simulation, during which major programming blocks will be discussed. Then, the simulation results of the DLL will be presented. Specifically we are going to present (1) the discriminator characteristic of the computer simulated DLL, (2) its transient response to an initial delay offset, and (3) its steady-state error variance in relationship to its loop parameters.

3.2.2.1 Simulation Overview

A first-order non-coherent delay-locked loop is simulated under noise-free and single-user environment. The overall software structure is illustrated in Figure 3.15. It basically consists of four programming blocks: a Maximum-Length Sequence Generator similar to what is shown in Figure 3.2, a Random Not-Return-to-Zero (NRZ) Symbol Generator, a Input Signal Generator that is essentially a modulator, and a First-Order Non-coherent DLL Simulator block that performs the operations shown in Figure 3.7.

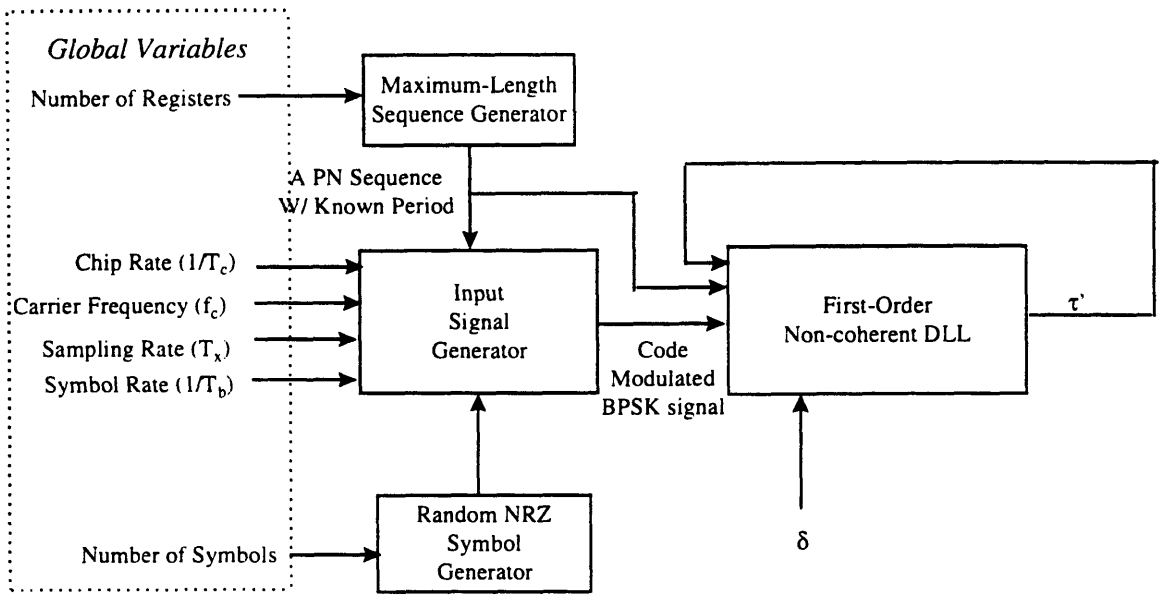

Figure 3.15 Overall software structure

Let us look the latter two in more detail. The Input Signal Generator takes a total of six inputs, two of which are vectors. One of the vectors is a PN sequence with a known period[9] coming out of the Maximum-Length Sequence Generator to be used for code spreading. And the other one is an array of random binary symbols. The rest of the inputs are four global variables that specify chip rate, carrier frequency, sampling rate, and symbol rate. With these inputs, the Input Signal Generator then produces an array of samples that simulates a code modulated BPSK signal, whose parameters are specified by the four input global variables.

The DLL block is essentially identical to the block diagram shown in Figure 3.7. The band-pass filter is a 2-pole Butterworth filter with one-sided bandwidth equals to the symbol rate $1/T_b$. Since we only consider first-order loops, loop filter is set to be a constant gain of 1. The operation of the VCO is identical to that of an integrator. The PN sequence generator is simulated using a subroutine *findpn.m*, which produce the correct PN chip at the next clock cycle. The

---

[9] The PN code generated for this simulation has a period of $1023 = 2^{10}-1$.

selection of the PN chip resembles a finite-state-machine (FSM). Depending on the VCO output, it can produce the same chip as in the previous cycle, the next chip in the sequence, or the previous chip in sequence. (*Appendix* contains a code listing for all the MATLAB routines used for the DLL simulation)

3.2.2.2 Simulation Results

Simulation were run to construct the discriminator characteristic (S-curve) of the DLL. The result is shown in Figure 3.16. The stair-like feature is the direct result of two things: (1) the way we implemented this DLL as sampling system, and (2) the algorithm we used to implement *findpn.m*. For this particular

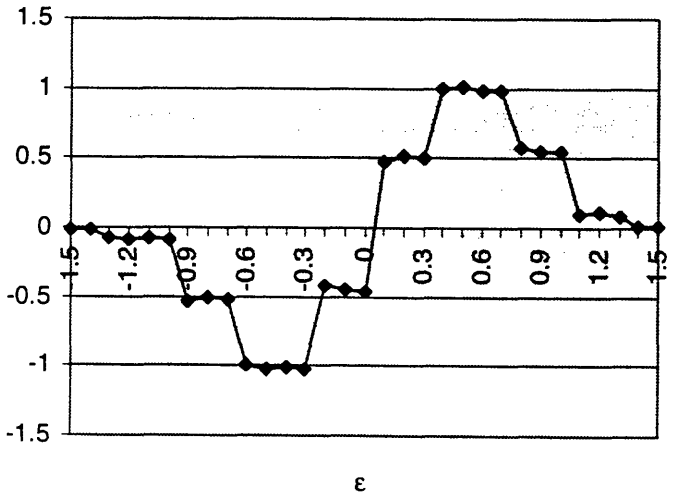

Figure 3.16 Discriminator characteristic for the simulated DLL ($\delta = 0.5$).

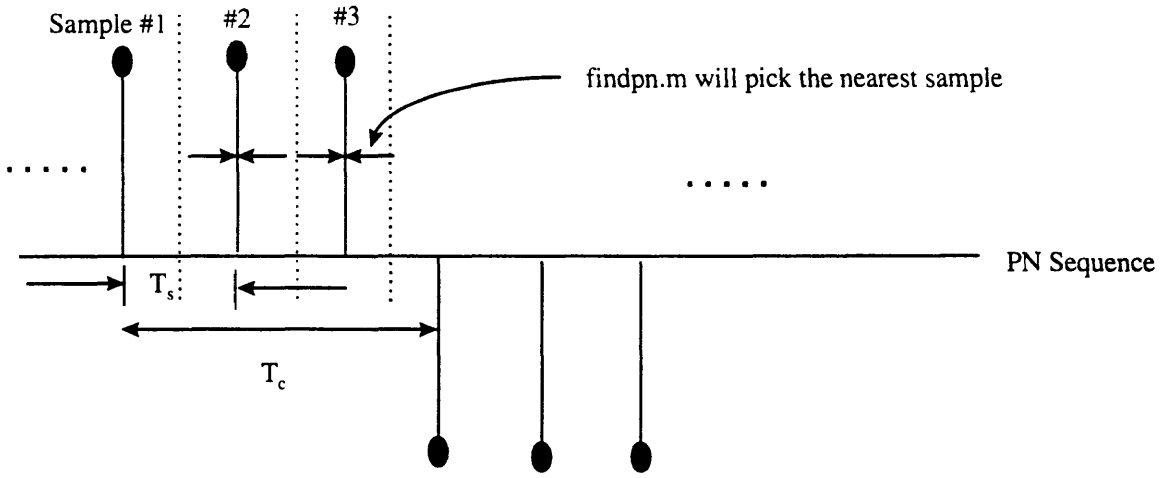

Figure 3.17 *Findpn.m* always picks the closes chip sample as its output. For this example, there are 3 samples for each chip

plot, we used 3 samples per chip. Remember that the S-curve is the difference of the square of the PN code autocorrelation function. The plot of Figure 3.9 assumes a continuous PN waveform, or at least very over-sampled PN sequence. The algorithm used by *findpn.m* also contributes to the appearance of the plateaus in the S-curve. It can be best described using Figure 3.17. In a way, *findpn.m* places each sample of a PN chip in the middle of a region of width equals to the sample interval T_s. If the output of the VCO (which is also the input to *findpn.m*) were mapped into, say the region around sample #2, then *findpn.m* will produce sample #2 during the next clock cycle. This means that within each region, the S-curve has the same value.

Transient response of the first-order loop to a step delay input for various VCO gain values is shown in Figure 3.18. K_{VCO} is included in K, which is part of the overall loop gain. Remember from the analysis of the first-order loop in (3.2.1.3), the loop gain is the inverse of the time constant. The larger the loop gain, the smaller the time constant, which means faster exponential decay. This is evident in Figure 3.18. The convergence time for a VCO gain of 0.5 is approximately twice as long as that for a gain of 1, and the convergence time for a gain of 0.375 is about 1.3 times as long as that of 0.5.

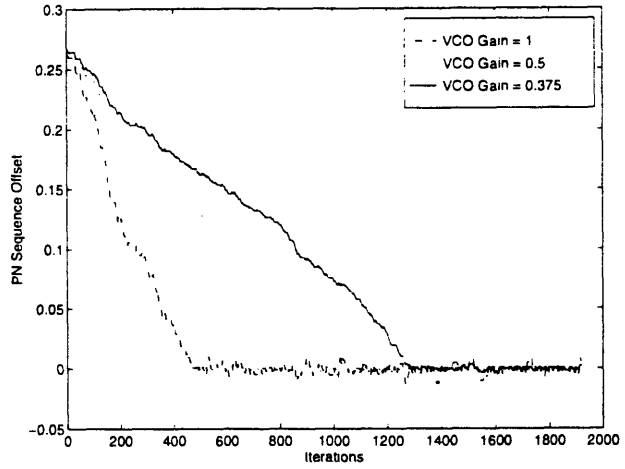

Figure 3.18 Transient response to a step delay input. The VCO gains are only to show relative values

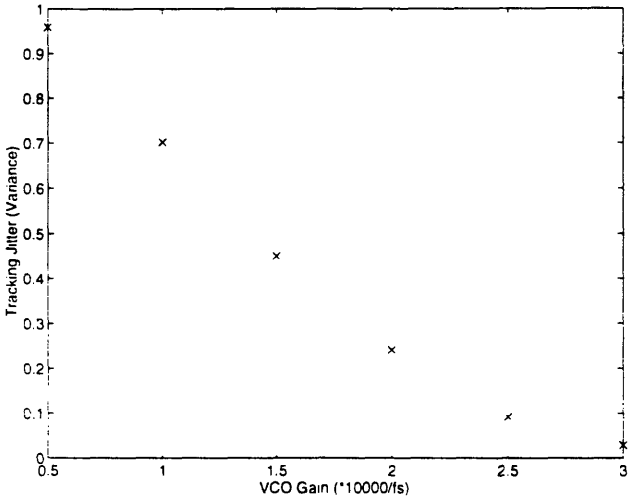

Figure 3.19 Relationship between first-order DLL tracking jitters and loop gain

Although larger loop gain results in shorter convergence time, it comes with a price in terms of increasing tracking jitters as shown in Figure 3.19. A good design thus involves an appropriate trade-off between tracking jitter and convergence time. This behavior is very similar to the learning curve of a LMS algorithm driven adaptive filter described in Chapter 2 (Figure 2.26). The existence of the finite variance should not lead us to conclude that the first order model is not valid. Rather the finite variance is the result of incorporating several approximations

into our loop model. First we assumed the code self-noise to be negligible when we were deriving the discriminator characteristic (see 3.2.1.1). Then we assumed the band-pass process used in both arms of the DLL to be ideal in order to build a simpler linear model (see 3.2.1.2).

3.2.3 Linear First-Order DLL For Optimum Tracking Performance

The tracking performance of a DLL is often measured by its mean-square tracking error, which actually contain four components as[10] (Chapter 4, [31])

$$\sigma_\varepsilon^2 = E[\varepsilon(t)^2] = \sigma_d^2 + \sigma_{\Delta\psi}^2 + \sigma_M^2 + \sigma_n^2 \qquad \text{(Equation 3.29)}$$

where σ_d^2 is the error due to the Doppler phase shift, $\sigma_{\Delta\psi}^2$ is due to VCO instabilities at the transmitter and at the receiver, σ_M^2 is due to the code modulation process, and σ_n^2 is due to the presence of additive noise process. To derive the minimum mean-square tracking error solution, we are going to assume that the first three terms of the error variance to be negligible[11]. Thus we want to minimize

$$\sigma_\varepsilon^2 = \sigma_n^2 \qquad \text{(Equation 3.30)}$$

So far in our discussion, we have ignored the effect of the noise-term that appeared in Equation 3.17 and 3.18. Now we have to include the noise in our analysis of tracking error.

---

[10] Chapter 4 of [31] actually describe the tracking jitter for a PLL. But the results there should also be applicable to a DLL.

[11] From steady-state analysis, we know that a first-order DLL will not be able to keep mean tracking error at zero if input contains frequency (clock rate) changes.

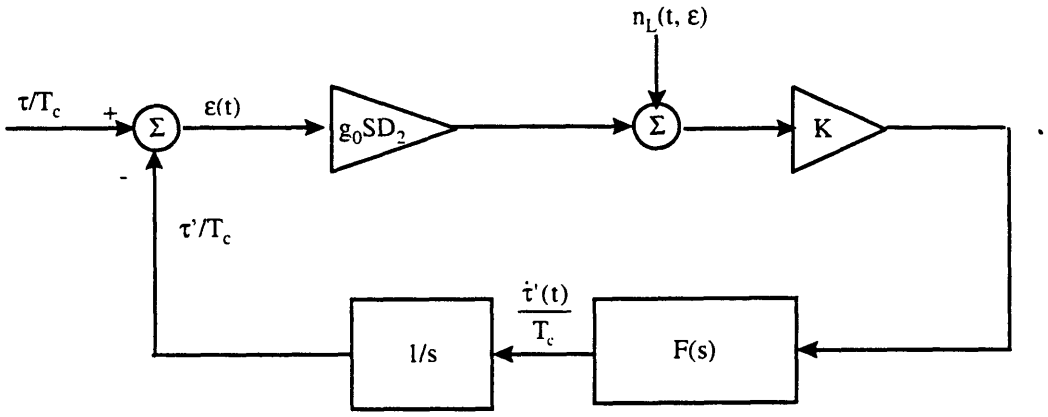

Figure 3.20 Linear DLL model in the presence of additive noise

If a DLL were to be of use in the reconstruction of the delay error process $\{\tau(t)\}$, one would anticipate that the total loop delay error $\{e(t)\}$ must be small. This forms the basis of the linear DLL model that if the loop is capable of reducing the delay error to a small value, we can use a simple gain in the place of the discriminator characteristic[12] (g_0 in Figure 3.11). To be consistent with the linear theory, we are going to assume that the noise-term in Equation 3.17 and 3.18 is, for small errors ε, effectively independent of ε. Therefore, we shall assume that Equation 3.18 is driven by a very wide-band (i.e., white) "linear noise" $n_L(t) = n_e(t,0)$ (Figure 3.20) [24].

Using (4-12) in [31] we can express tracking error variance as

$$\sigma_\varepsilon^2 = \frac{N_L B_L(\delta)}{(g_0 SD_2)^2} \quad \text{(Equation 3.31)}$$

where N_L is the one-sided power spectral density (PSD) of $n_L(t)$ that is assumed constant over the region of the linear model, and $B_L(\delta)$ is the single-sided bandwidth of the close-loop transfer function H(s) in Equation 3.19:

---

[12] This is similar to replace $\sin\phi$, an often used discriminator for PLL , by the first term of its Tayler series ϕ, where ϕ is the phase error. Thus the discriminator becomes a unit gain factor.

$$B_L(\delta) = \int_0^{\infty} |H(j\omega)|^2 d\omega \qquad \text{(Equation 3.32)}$$

The dependence of $B_L(\delta)$ on δ stems from the fact that $|H(j\omega)|^2$ depends on $g_0 = 4\,(1 - \delta)$. Hence, for fixed signal power, arm filters, and closed-loop gain (G in Equation 3.19), the closed-loop bandwidth decreases linearly with δ. This does not, however, imply a decrease in σ_e^2 since error variance also depends on both g_0 in the denominator of Equation 3.31, and N_L which is given in (21) of [25].

Assuming the linear case that signal-to-noise ratio in the close-loop bandwidth is large, we can approximate the tracking error variance as (see (2.29), Part 4 of [11])

$$\sigma_e^2 = \frac{1}{2\gamma_L S_L} \qquad \text{(Equation 3.33)}$$

where $\gamma_L = \dfrac{S}{N_0 B_L}$ is the signal-to-noise ratio in the loop bandwidth, and S_L is called the "squaring loss" of the DLL. To reduce tracking error, one could make γ_L larger, which can be accomplished by reducing the loop-bandwidth B_L. Using the notation as in Equation 3.20, we can express B_L as (4-17 in [31])

$$B_L = \frac{G}{4} \qquad \text{(Equation 3.34)}$$

where G is the open-loop gain. Clearly lowering the loop gain by using smaller K_{VCO} or K_M will reduce tracking error, but it will result in longer convergence time (see Figure 3.19).

Another way to reduce tracking error is to increase[13] the "square loss" term S_L. In terms of γ_d the data symbol signal-to-noise ratio (i.e. SNR in the data bandwidth)

$$\gamma_d = \frac{ST_b}{N_0} \qquad \text{(Equation 3.35)}$$

---

[13] Well, this actually means that we want to reduce the loss.

and r the ratio of band-pass filter bandwidth $B_H = \int_{-\infty}^{\infty} |H(j\omega)|^2 d\omega$ to data rate $R_b = 1/T_b$, we can express S_L as [11]

$$S_L = \frac{D_2^2}{D_4 + K_L\left(\frac{\delta}{1-\delta}\right)^2 \frac{r}{2\gamma_d}} \qquad \text{(Equation 3.36)}$$

where

$$K_L = \frac{\int_{-\infty}^{\infty} |H_1(j\omega)|^4 d\omega}{B_H}$$
$$D_4 = \int_{-\infty}^{\infty} S_c(j\omega)|H_1(j\omega)|^4 d\omega \qquad \text{(Equation 3.37)}$$

In NRZ polar format, $S_d(\omega) = T_b \text{sinc}^2(T_b/2)$ (see 6.12 in [21]). For one- and two-pole Butterworth filters, values of D_2 and D_4, both expressed in terms of r, are listed in Table 2.1 and Table 2.2 of [11], respectively. Also for a n-pole Butterworth filter, K_L has the simple expression of (n-1)/n [34]. Using these values, we can plot the square loss with respect to $r = B_H T_b$.

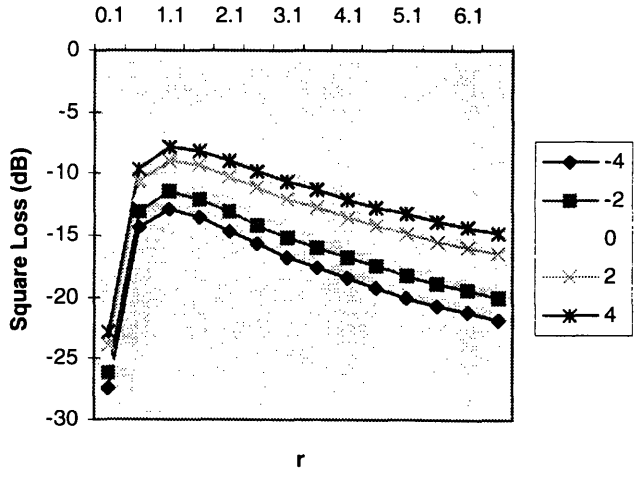

Figure 3.21 Square Loss (dB) vs. r for various values of γ_d (dB); two-pole Butterworth filter, δ = 0.5, NRZ polar coding

Figure 3.21 shows a plot of S_L vs. r for various values of γ_d for the case of $\delta = 0.5$[14]. We can observe that for each value of γ_d, there is a corresponding value of r that minimizes the loop's squaring loss, which in turn minimizes the loop's tracking error (see Equation 3.33). Thus we can optimize the tracking error by choosing the appropriate bandwidth for the arm band-pass filter.

3.3 Conclusion

In this chapter, we described in detail the operation and analysis of delay-locked loops, with emphasis on first-order loops. First we derived the DLL discriminator characteristic (Figure 3.9) using the autocorrelation properties of PN sequences (Equation 3.2 and 3.3). Then through linear approximation, we arrived at a simple closed-loop model (Figure 3.11). Using this model, we were able to look at both steady-state and transient responses of both first-order and second-order loops. In the *Results* section, we presented the simulation results of a non-coherent first-order DLL under noise-free environment. Its steady state response to a delay step (analogous to a phase step in PLL) show finite variance. We accounted for that by ignoring PN code self-noise and assuming ideal arm band-pass filter. And finally we brought noise into our analysis in order to find a set of design parameters that optimize mean-square tracking error, which is an important performance parameter for any tracking loop. Based on the work in [11] and [24], we were able to present a optimal solution in terms of bandwidth used by the DLL's arm band-pass filter.

---

[14] Plots for other values of δ can be easily extrapolated by manipulating Equation 3.36.

Chapter 4 : Conclusion

4.1 General Adaptive System

In this thesis, we presented two applications of adaptive system design in modern digital communications. First we describe the application of a linear adaptive transversal filter as an receiver front end equalizer. Then we presented the application of delay-locked loop (DLL) for code tracking in direct-sequence spread-spectrum communication systems. Both adaptive filters and closed-loop tracking devices such as DLLs,

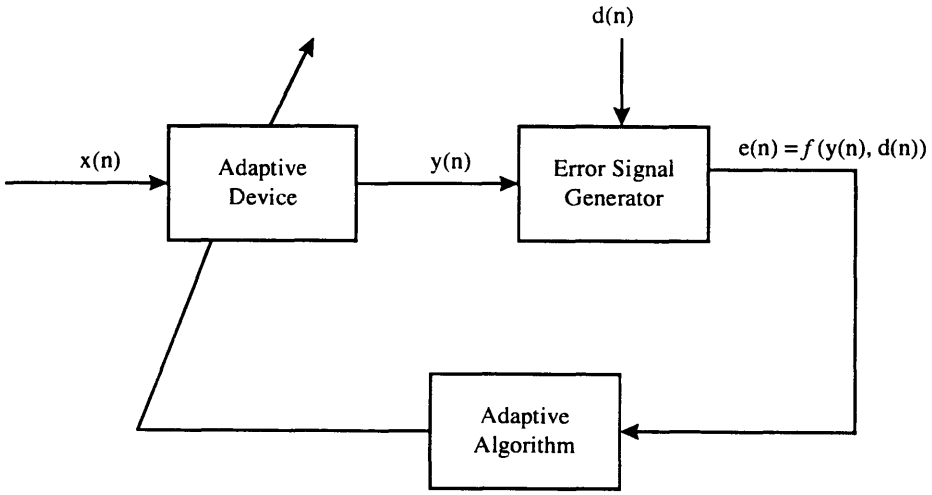

Figure 4.1 Block diagram of a generalized adaptive system

or its closely related cousin phase-locked loops (PLLs) as adaptive systems. All adaptive systems share one common feature: an input vector and a desired response are used to compute an estimation error, which is in turn used by a predetermined algorithm to control the values of some adjustable parameters of an adaptive device. The goal of the algorithm is to minimize the estimation error according to some statistical criterion. A commonly used one is the mean-square

error criterion. A general adaptive system can be represented by the block diagram shown in Figure 4.1. Depending on the type of adaptive device, error signal generator, and adaptive algorithm being used, different adaptive systems can be constructed. The front end equalizer discussed in Chapter 2 has its adaptive device a transversal filter that has adjustable tap weights, its error signal generator a simple difference operator, and an adaptive algorithm that based on the least-mean-square (LMS) criterion. In the DLL case, though less obvious, we can still show that it fits into the model in Figure 4.1. The adaptive device in this case would be the PN code generator with adjustable output frequency. The error generator is more complicated than that for the front end equalizer (compare Equation 3.13 with Equation 2.17). It actually consists of a difference operator preceded by two correlators, each of which is, for the case of non-coherent DLL, made up of a multiplier, a band-pass filter and a square-law envelop detector (see Figure 3.7 for details). The adaptive algorithm is embodied in a voltage-controlled-oscillator (VCO), which is simply an accumulator.

4.2 Future Research

Chapter 3 addressed the application of DLL as a code tracking device for direct-sequence spread-spectrum communication system. In our simulation, only first-order loop was considered. As we know from the analysis, first-order loop is only adequate in tracking delays between two PN code sequences that have the same code rate. But often in practice, the two PN code generator used by the transmitter and receiver would have different clock rates. Also, since spread-spectrum system is considered for mobile communications, Doppler shift would be another concern. Under these circumstances, a second-order or higher loop will be necessary. Future research therefore may include simulations of these higher order loops for delay tracking when there exists clock mismatch and Doppler shift. Multipath effect may also be included in the simulation. From

multiple-access application point of view, it would be also be beneficial to study DLL under multi-user environment.

Appendix

A. Selected MATLAB Code for Front End Correction Problem

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following code segment calculates the vector error resulting from the front end with and %
%without the equalizer present for the GSM simulation%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cd /mit/garrison/thesis/lgumm
load gsmdef;
framelen = 2*sampsym;
delta = 100;

t1 = (-T:1/fs:(numsym-1)*T-1/fs); t2 = (0:1/fs:(numsym)*T-1/fs);
m1g = conv(g,o_bits); m1g = m1g(delay+1:length(m1g)-delay);
m2g = conv(g,e_bits); m2g = m2g(delay+1:length(m2g)-delay);
%m1g = real(filtfilt(g,1,o_bits));
%m2g = real(filtfilt(g,1,e_bits));
mu = 0.03; filt_len = 121; N = 10000;
[heq,error] = adaptgsm(fs,mu,N,filt_len,numd,dend);

results = [];
index = 1;
for fc = 3.5e6:0.25e6:8.5e6,
        phi1=[];        phi1=cos((pi/2/T)*t1).*cos(2*pi*fc*t1);
        phi2=[];        phi2=sin((pi/2/T)*t2).*sin(2*pi*fc*t2);
        mod_i=[];       mod_i=m1g.*phi1;
        mod_q=[];       mod_q=m2g.*phi2;
        gmsk=[];
        gmsk=[mod_i,zeros(size(1:sampsym))]+[zeros(size(1:sampsym)),mod_q];
        for j = 1:3
                if j == 1
                        gmskh = [];     gmskh = gmsk;
                elseif j == 2
                        gmskh = [];     gmskh = real(filtfilt(numd,dend,gmsk));
                else    gmskh = real(filtfilt(heq,1,gmskh));
                end
                out_i = []; out_q=[];
                [out_i, out_q] = gmskdemod(phi1,phi2,gmskh,sampsym,g,delay);
                if j == 1
                        I = []; Q = [];
                        [I,Q] =plotcons(0,framelen,out_i,out_q,delta);
                        results(index,j) = fc/(1e6)
                else    isample = []; qsample = [];
                        [isample,qsample] =plotcons(0,framelen,out_i,out_q,delta);
```

```
                        results(index,j) = vecerr(I,Q,isample,qsample)
                end
        end
        index = index +1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [vector_error] = vecerr(idef, qdef,itest, qtest)
%vecerr computs the vector error by comparing vector (idef, qdef)
%with vector (itest,qtest)

temp1 = (idef-itest).^2 + (qdef-qtest).^2;
temp2 = temp1 ./(idef.^2 + qdef.^2);
temp3 = temp2 .^0.5;
sum1 = sum(temp3)/length(idef);
vector_error = sum1*100;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y=ploteye(offset,len,stream,delta,linetype)
%plots the eyediagram. It takes input data stream, and trucates the first %offset+delta number of
samples, then group the rest of the stream in to groups %of len, and plots them on the same graph

data = stream(offset+delta+1:length(stream));
tmp = length(data);

i = 1;
while tmp > len,
        plot(data((i-1)*len+1:i*len),linetype);
        hold on;
        tmp = tmp - len;
        i = i+1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [iaxis, qaxis]=plotcons(offset, len, istream, qstream,delta)
%plotcons plots the constellation diagram using I and Q data
%present in istream and qstream, respectively

idata = istream(offset+1:length(istream));
qdata = qstream(offset+1:length(qstream));
tmp = length(idata);

i = 1;
while tmp > len,
        iaxis(i) = idata((i-1)*len+1);
        qaxis(i) = qdata((i-1)*len+1);
        tmp = tmp-len;
```

```
        i = i+1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [coef,error] = adaptcdma(sampfreq,stepsize,duration,len_fir,B,A)
%adaptcdma computes the converged adaptive filter coefficients, and convergence error
%for  specified sampling rate, step-size, training duration, tap-length, and desired filter
%response

F1 = [2e6 2.75e6 9.25e6 10e6]; M1=[0 1 0]; DEV1=[0.001 0.0000001 0.001];

[n1,fo1,mo1,w1] = remezord(F1,M1,DEV1,sampfreq);
if rem(n1,2) == 0;
        h_des = remez(n1,fo1,mo1,w1);
    [coef,error] = adapt(h_des,n1+1,stepsize,len_fir,B,A,length(B),sampfreq,duration);
else    h_des = remez(n1+1,fo1,mo1,w1);
    [coef,error] = adapt(h_des,n1+2,stepsize,len_fir,B,A,length(B),sampfreq,duration);
end
```

B. Selected MATLAB Code for Non-coherent DLL simulation

```
%%%%%%%%%%%%%%%%%%
% noncoherent delay lock loop.%
%modified from noncodll%%%%
%%%%%%%%%%%%%%%%%%
load mseq;              %maximum-length sequence
order = 10;             %total number of registers used to compute the PN sequence
q=3;                    %number of samples per chip
Tc = 1/1.25e6;          %chip interval
cpb = 32;               %chips per data bit
Tb = cpb*Tc;            %data symbol interval
Ts = Tc/q;              %sample interval
maxlen = 2^order-1;     %period of the PN sequence used in the simulation
numbit = 20;            %total number of simulated data symbol bits
numchip = numbit*cpb; %total number of chips numbit of symbol bits
fc = 1e6;wc = fc*2*pi;  %carrier frequency
x = rand(1,numbit);i = 1;         %x represents the random data bits
for i = 1:numbit,
    if x(i)>0.5
        x(i) = 1;
    else x(i) = -1;
    end
end
inpn=[];                          %PN code at each sample instant
what = [];              %data modulated PN code
what(1) = mseq(1)*x(1);inpn(1) = mseq(1);inind=[];inind(1) =1;
```

```
time = (0:Ts:fix(numchip*Tc./Ts)*Ts);N = length(time);  %time axis
for i = 2:N,
      inind(i) = modulus((i-1)/q,maxlen);
      inpn(i) = mseq(inind(i));
         what(i) = inpn(i)*x(ceil(time(i)/Tc/cpb))*cos(wc*time(i));
end

dmpn = what;              %BPSK signal
 %fid = fopen('corec4.dat','w');
 %fprintf(fid, '%2i\n',dmpn);

delta = 0.5;n=1;
%[b,a] = cheby1(n,0.05,[(fc-1/Tb)*2*Ts (fc+1/Tb)*2*Ts]);
[b,a] = butter(n,[(fc-1/Tb)*2*Ts (fc+1/Tb)*2*Ts]);        %arm bandpass filter used by DLL
      %fid = fopen('lpf','w');
      %for i= 1:n+1,
      %       fprintf(fid, '%e\t%e\n',b(i),a(i));
      %end
[c,d] = butter(2*n,1/Tb*2*Ts);


y1 = 0;
error =[];                %error signal that feeds to the loop filter, which is 1 for 1st-order loop
tau=[];                   %delay offset log
tau(1)=0.1;               %initial offset
k_vco =Ts*10000;          %VCO gain
chips4demod=[];                  %synchronized PN code that is being sent to the demodulator
temp1 = zeros(1,2*n+1);          temp2 = zeros(1,2*n); temp3 = temp1; temp4 = temp2;
temp5 = temp1; temp6=temp2; temp7=temp5; temp8=temp6;
fid = fopen('dll3.dat','w');

for i = 1:N,
          vco_in_now = (i-1)/q-tau(i)+delta;
          noise = randn*0;
          chip1 = findpn(mseq,vco_in_now);
          x1(i) = (dmpn(i)+noise)*chip1;
          chip2 = findpn(mseq,vco_in_now-2*delta);
          x2(i) = (dmpn(i)+noise)*chip2;
          chips4demod(i) = findpn(mseq,vco_in_now-delta);
          %fprintf(fid,'%i\t%i\t%i\t%i\t%f\t%f\t%f\t%f\n',i,inpn(i),chips4demod(i),...
          %       inpn(i)*chips4demod(i), (i-1)/q,(i-1)/q-tau(i),tau(i),error(i) );

          temp1 = [x1(i),temp1(1:2*n)];    %upper band-pass filtering
          y1(i) = (sum(temp1.*b)-sum(temp2.*a(2:2*n+1)));
          temp2 = [y1(i),temp2(1:2*n-1)];

          temp3 = [x2(i),temp3(1:2*n)];    %lower band-pass filtering
           y2(i) = (sum(temp3.*b)-sum(temp4.*a(2:2*n+1)));
          temp4 = [y2(i),temp4(1:2*n-1)];
          z1(i)=y1(i)^2; z2(i) = y2(i)^2;    %square-Law
```

```
        temp5 = [z1(i),temp5(1:2*n)];    %upper envelope detection
        a1(i) = (sum(temp5.*c)-sum(temp6.*d(2:2*n+1)));
        temp6 = [a1(i),temp6(1:2*n-1)];

        temp7 = [z2(i),temp7(1:2*n)];    %lower envelope detection
        a2(i) = (sum(temp7.*c)-sum(temp8.*d(2:2*n+1)));
        temp8 = [a2(i),temp8(1:2*n-1)];

        error(i) = a1(i)-a2(i);                    %generate error signal that feeds the loop filter
        tau(i+1) =tau(i)-error(i)*k_vco; %accumulating effect of the VCO
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [out] = findpn(pns,x_float)

%pns is a the PN sequence that is used for the transmission
%x-float is the VCO output that can be mapped onto a specific chip
%of the PN sequence
l = length(pns);
if rem(floor(x_float),l) >=0
     ind = rem(floor(x_float),l)+1;
     out = pns(ind);
else
     ind = rem(floor(x_float),l)+l+1;
     out = pns(ind);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [out] = modulus(x_float,y_pos_integer)
%similar to mod operation on integers

if x_float >=0
     out = rem(floor(x_float),y_pos_integer)+1;
else
     out = rem(floor(x_float)+y_pos_integer,y_pos_integer)+1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [mseq] = pnseq(reg);
%In order to generate maximal-length sequences, different order number requires different
%argument for the xor operator.  For example, for order = 10, one could use [10,3] if the registers
%are numbered 1-10.

order = length(reg);
mseq = [];
for i = 1:2^order-1,
```

```
    mseq(i) = reg(order);
    if mseq(i) == 0
        mseq(i) = -1;
    end
    reg = [xor(reg(order),reg(order-3)), reg(1:order-1)];
end
```

References

[1] S. Ariyavistakul and T-P. Liu, "Characterizing the effects of nonlinear amplifiers on linear modulation for digital portable radio communications," *IEEE Transactions on Vehicular Technology*, vol. 39, no. 4, pp. 383-389, Nov. 1990.

[2] J. Anderson, T. Aulin, and C-E. Sundberg, *Digital Phase Modulation*. New York: Plenum Publishing Company, 1986.

[3] Y. Akaiwa and Y Nagata, "Highly efficient digital mobile communications with a linear modulation method," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, no. 5, pp. 890-895, June 1987.

[5] T. Aulin and C-E. Sundberg, "Continuous phase modulation-partI: full response signaling," *IEEE Transactions on Comunications*, vol. COM-29, no. 3, pp. 196-209, March 1981.

[6] T. Aulin and C-E. Sundberg, "Continuous phase modulation-part II: partial response signaling," *IEEE Transactions on Communications*, vol. COM-29, no. 3, pp. 210-225, March 1981.

[7] European Digital Cellular Telecommunications System (Phase 2): Modulation (GSM 05.04), pp. 7-9, European Telecommunications Standards Institute (ETIS),1993.

[8] R. de Buda, "Coherent demodulation of frequency-shift keying with low deviation ratio," *IEEE Transactions on Communications*, vol. COM-20, no. 3, pp. 429-436, June 1972.

[9] S. Pasupathy, "Minimum shift keying: a spectrally efficient modulation," *IEEE Communications Magazine*, vol. 17, no. 4, pp. 14-22, July 1979.

[10] K. Murota and K. Hirade, "GMSK modulation for digital mobile telephony," *IEEE Trans. on Communications*, vol. COM-29, no. 7, pp. 1044-1050, July 1981.

[11] M. K. Simon, J. K. Omura, R. A Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*, New York: McGraw-Hill, Inc., 1994.

[12] A. J. Viterbi, *CDMA-Principles of Spread Spectrum Communication*, New York: Addison-Wesley, 1995.

[13] W. C. Y. Lee, "Overview of Cellular CDMA," *IEEE Trans. on Vehicular Technology,* vol. 40, no. 2, pp. 291-302, May 1991.

[14] Qualcomm Corporation, "An overview of the application of CDMA to digital cellular systems and personal cellular networks", May 21, 1992.

[15] S. Haykin, *Adaptive Filter Theory*, New York: Prentice Hall, 1991.

[16] B. Mulgrew, and C. F. N. Cowan, *Adaptive Filters and Equalisers*, Kluwer Academic Publishers, 1988.

[17] B. Widrow, and M. E. Hoff, Jr., "Adaptive switching circuits," *IRE WESCON Conv.* Rec. pt. 4, pp. 96-104, 1960.

[18] S. T. Alexander, "Transient weight misadjustment properties for the finite precision LMS algorithm," *IEEE Trans Acoust. Speech Signal Process.*, vol. ASSP-35, pp. 1250-1258.

[19] S. W. Golomb, *Digital Communications with Space Applications*, Prentice Hall, 1964.

[20] J. G. Proakis, and M. Salehi, *Communication Systems Engineering*, Prentice Hall, 1994.

[21] J. G. Proakis, *Digital Communications*, 2nd Ed., McGarw Hill, 1989.

[22] R. L. Pichholtz, D. L. Schilling, and L. B. Milstein, "Theory of Spread-Spectrum Communications-A Tutorial", *IEEE Transactions on Communications*, Vol. Com-30, No. 5, pp. 855-884, May 1982.

[23] d. V. Sarvate, and M. B. Pursley, "Crosscorrelation properties of pseudo-random and related sequences" *Proc. IEEE*, vol. 68, pp. 593-619.

[24] A. Polydoros and C. L. Weber, "Analysis and optimization of correlative code-tracking loops in spread-spectrum systems," *IEEE Transactions on Communications*, Vol. COM-30, pp. 30-43, 1985.

[25] M. K. Simon, "Noncoherent pseudo-noise code tracking performance of spread spectrum receivers," *IEEE Transactions on Communications*, vol. COM-25, no. 3, pp. 327-345, 1977.

[26] W. J. Gill, "A comparison of binary delay-lock loop implementations," *IEEE Transactions on Aerospace Electronic System*, Vol. AES-2, pp. 415-424, 1966.

[27] J. J. Spilker, Jr., "Delay-lock tracking of binary signals," *IEEE Transactions on Space Electronicsand Telecommunications*, Vol. SET-9, pp. 1-8, 1963.

[28] P. T. Nielsen, "On the acquisition behavior of binary delay-lock loops," *IEEE Trans. Aerosp. And Electr. Syst.*, vol. AES-11, pp. 415-418, May 1975.

[29] T. C. Huang and J. K. Holmes, "Performance of noncoherent time-shared PN code tracking loops," *1976 NTC Record*, PP. 45.4.1-45.4.5, November 29 -December 1, 1976, Dallas, TX

[30] H. P. Hartmann, "Analysis of a ditering loop for PN code tracking," *IEEE Trans. Aerosp. And Electr. Syst.*, vol. AES-10, no. 1, pp. 2-9, January 1974.

[31] W. C. Lindsey, *Synchronization Systems in Communications and Control*, Englewood Cliffs, NJ: Prentice-Hall, 1972.

[32] W. C. Lindsey, and M. K. Simon, *Telecommunication Systems Engineering*, Englewood Clifs, NJ: Prentice-Hall, 1973.

[33] F. M. Gardner, *Phase-lock Techniques*, John Wiley and Sons, Inc., 1966.

[34] K. Bakhru, "Communication receiver design using digital processing", Digital Signal Processing, vol. 2, no. 1, pp. 2-13, January 1992.

[35] M. K. Simon and W. C. Lindsey, "Optimum performance of suppressed carrier receivers with Costas loop tracking," IEEE Transactions in Communications, COM-25, no. 2, pp. 215-227, February 1977.