SOFTWARE RADIO

(SIMULATION OF QPSK TRANSCEIVER)

SYNDICATE
PC MUHAMMAD AZAM (LEADER)
NC FARHAN AZEEM
NC EZAZ AKRAM
NC WAQAS AKHTER

PROJECT SUPERVISOR:
Dr. NOMAN JAFRI
External DS: Dr. ISMAIL SHAH

Dissertation submitted for partial fulfillment of requirement of MCS/NUST for the award of the B.E. degree in Telecommunication Engineering.

Department of Electrical Engineering
Military College of Signals
Rawalpindi

APRIL 2003

DEDICATION

Dedicated to our parents who have been a source of constant encouragement for us and to our idols who have given us inspiration throughout our degree.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award of qualification either at this institution or elsewhere.

ACKNOWLEDGMENTS

All praises for Allah (The Almighty) who enlightened us with the requisite knowledge on portion of this subject enabling us to accomplish this extremely challenging and gigantic task.

After Almighty Allah we are extremely thankful to Dr. Jamil Ahmad because we could not have been successful in our endeavors with out his providential and able supervision. We are proud of our effort truly because of him. He is not our supervisor but he never ever minded helping us in spite of all his commitments.

We are also thankful to Dr. Ismail Shah for his kind guidance. We are also grateful to Dr. Noman Jafri for his extremely useful contribution. Special thanks to EE Dept. and the staff members of EE Dept. for their assistance and cooperation in the provision of material resources. We also extend our deep gratitude to all those who encouraged us in strive for achieving the goals.

PREFACE

Software radio is an emerging technology, thought to build flexible radio systems, multiuser, multistandard, multiband, reconfigurable and reprogrammable by software. In fact the term software radio stands for the radio functionalities defined by software, meaning the possibility to define by software the typical functionality of a radio, usually implemented in transmitter and receiver equipment by dedicated hardware. The presence of software defining the radio necessarily implies the use of DSPs to replace dedicated hardware, to execute, in real time, the necessary software.

Software radio will provide the means to realize signal processing tasks of communication transceiver by means of software .The replacement of ASIC technology with DSP technology opens the road to new horizons. Software implementation of baseband functions such as coding, modulation, equalization and pulse shaping.

An approach towards this project required adequate knowledge of Digital Signal Processing and Digital Communications to implement the communication functions in software . We have designed a complete all digital communication transceiver in Matlab which is capable of the modulation, pulse shaping, phase synchronization, timing synchronization, equalization, channel coding etc. The wireless channel effects like multipath delay spread and mobility are created in software and the performance of the transceiver is evaluated for such channel.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1

SOFTWARE RADIO CONCEPT

1.1 NEED FOR SOFTWARE RADIOS

With the emergence of new standards and protocols, wireless communications is developing at a furious pace. Rapid adoption of the wireline-base Internet has led to demand for wireless Internet connectivity but with added capabilities, such as integrated services that offer seamless global coverage and user-controlled quality of service (QoS). The challenge in creating sophisticated wireless Internet connectivity is compounded by the desire for future-proof radios, which keep radio hardware and software from becoming obsolete as new standards, techniques, and technology become available. The concept of integrated seamless global coverage requires that the radio support two distinct features: first, global roaming or seamless coverage across geographical regions; second, interfacing with different systems and standards to provide seamless services at a fixed location. Multimode phones that can switch between different cellular standards like IS-95 and Global System Mobile (GSM) fall in the first category, while the ability to interface with other services like Bluetooth or IEEE 802;11 networks falls in the second category. Further, the rate of technology innovation is accelerating, and predicting technological change and its ramifications to business is especially problematic. As a result, to keep their systems up to date, wireless systems manufacturers and service providers must respond to changes as they occur by upgrading systems to incorporate the latest innovations or to fix bugs as they are discovered. Many manufacturers tell horror stories of releasing hundreds of thousands of defective phones that had to be recalled and discarded. Since frequent redesign is expensive, time-consuming, and inconvenient to end users, interest is increasing in future-proof radios.

Existing technologies for voice, video, and data use different packet structures, data types, and signal processing techniques. Integrated services can be obtained with either a single device capable of delivering various services or with a radio that can communicate with devices providing complementary services. The supporting technologies and networks that might have to use can vary with the physical location of the user. To successfully communicate with different systems, the radio has to communicate and decode the signals of devices using different air interfaces. Furthermore, to manage changes in networking protocols,

Services and environments, mobile protocols, such as IP(Internet Protocol) and MExE (Mobile Execution Environment). Such radios can be implemented efficiently using software radio architectures in which the radio reconfigures itself based on the system it will be interfacing with and the functionalities it will be supporting.

Second generation (2G) wireless technology consists of a handful of incompatible standards, and the goal behind the development of third generation (3G) standards is compatibility among these standards within and between different generations' standards. Even if cellular standards globally converge, 3G systems require multimode operation and automatic mode selection. With fourth generation (4G) and possibly 3G Systems , the user' s application will likely have the ability to control the quality of service and obtain a higher QoS for a higher cost . Higher QoS can be achieved through priority scheduling of packets, changes in data packaging, improved protection coding, better channel equalization techniques, implementation of smart antennas, and so on. The mobile subscriber must have the ability to select the network provider as well as the services needed.

1.2 WHAT IS SOFTWARE RADIO?

The term "*software radio"* was coined by Joe Mitola in 1991 to refer to the class of reprogrammable or reconfigurable radios. In other words, the same piece of hardware can perform different functions at different times. The SDR forum defines the ultimate software radio (USR) as a radio that accepts fully programmable traffic and control information and supports a broad range of

frequencies, air interfaces, and applications software. The user can switch from one air interface format to another in milliseconds,

Use the Global Positioning System (GPS) for location, store money using smartcard technology, or watch a local broadcast station and or receive a satellite transmission.

The exact definition of a software radio is controversial and no consensus exists about the level of reconfigurability needed to qualify a radio as a software radio. A radio that includes a microprocessor or digital signal processor (DSP) does not necessarily qualify as a software radio [1]. However, a radio that defines in software its modulation, error correction, and encryption processes, exhibits some control over the RF hardware, and can be reprogrammed is clearly a software radio. A good working definition of a software radio:

"A radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software."

The degree of reconfigurability is largely determined by a complex interaction between a number of common issues in radio design , including systems engineering , antenna form factors , RF electronics , Baseband processing , speed and reconfigurability of the hardware and power supply management .

The term software radio generally refers to a radio that derives its flexibility through software while using a static hardware platform. On the other hand, a *soft radio* denotes a completely configurable radio that can be programmed in software to configure the physical hardware. In other words, the same piece of hardware can be modified to perform different functions at different times, allowing the hardware to be specifically t tailored to the application at hand. Nonetheless, the term software radio is sometimes used to encompass soft radios as well.

The functionality of conventional radio architectures is usually determined primarily by hardware with minimal configurability through software. The hardware consists of the amplifiers, filters, mixers (probably several stages), and oscillators. The software is confined to controlling the interface with the network, stripping the headers and error correction codes from the data packets, and determining where the data packets need to be routed based on

the header information. Because the hardware dominates the design, upgrading a conventional radio design essentially means completely abandoning the old design and starting over again. In upgrading a software radio design, the vast majority of the new content is software and the rest is improvements in hardware component design. In short, software radios represent a paradigm shift from fixed, hardware-intensive radios to multi-band, multimode, software-intensive radios.

1.3 CHARACTERISTICS AND BENEFITS OF A SOFTWARE RADIO

Implementation of the ideal software radio would require either the digitization at the antenna, allowing complete flexibility in the digital domain, or the design of a completely flexible radio frequency (RF) front-end for handling a wide range of carrier frequencies and modulation formats [2]. The ideal software radio, however. is not yet fully exploited in commercial systems due to technology limitations and cost considerations.

A model of a practical software radio is shown in figure. The receiver begins with a smart antenna that provides a gain versus direction characteristic to minimize interference, multipath, and noise. The smart antenna provides similar benefits for the transmitter. Most practical software radios digitize the signal as early as possible in the receiver chain while keeping the signal in the digital domain and converting to the analog domain as late as possible for the transmitter using a digital to analog converter (DAC). Often the received signal is digitized in the intermediate frequency (IF) band. Conventional radio architectures employ a super heterodyne receiver, in which the RF signal is picked up by the antenna along with other spurious/unwanted signals, filtered, amplified with a low noise amplifier (LNA), and mixed with a local oscillator (LO) to an IF. Depending on the application, the number of stages of this operation may vary. Finally, the IF is then mixed exactly to baseband. Digitizing the signal with an analog to digital converter (ADC) in the IF range eliminates the last stage in the conventional model in which problems like carrier offset and imaging are encountered. When sampled, digital IF signals give spectral replicas that can be placed accurately near the baseband frequency, allowing frequency translation

and digitization to be carried out simultaneously. Digital filtering

(channelization) and sample rate conversion are often needed to interface the output of the ADC to the processing hardware to implement the receiver. Likewise, digital filtering and sample rate conversion are often necessary to interface the digital hardware that creates the modulated waveforms to the digital to analog converter. Processing is performed in software using DSPs, field programmable gate arrays (FPGAs), or application specific integrated circuits (ASICs). The algorithm used to modulate and demodulate the signal may use a variety of software methodologies, such as middleware, e.g., common object request broker architecture (CORBA), or virtual radio machines, which are similar in function to JAVA virtual machines. This forms a typical model of a software radio.

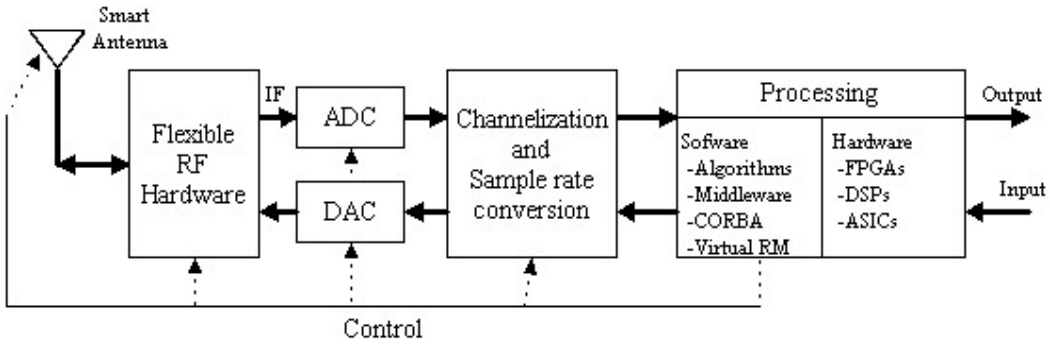

Figure 1.1: Model of a software radio.

The software radio provides a flexible radio architecture that allows changing the radio personality, possibly in real-time, and in the process somewhat guarantees a desired QoS. The flexibility in the architecture allows service providers to upgrade the infrastructure and market new services quickly. This flexibility in hardware architecture combined with flexibility in software architecture, through the implementation of techniques such as object - oriented programming and object brokers, provides software radio with the ability to seamlessly integrate itself into multiple networks with wildly different air and data interfaces. In addition, software radio architecture gives the system, new capabilities that are easily implemented with software. For example, typical upgrades may include interference rejection techniques, encryption, voice recognition and compression, software-enabled power minimization and control, different addressing protocols, and advanced error recovery schemes. Such capabilities are well-suited for 3G and 4G wireless

requirements and advanced wireless networking approaches [3]. In summary, five factors are expected to push wider acceptance of software radio.

- *Multifunctionality:* With the development of short-range networks like Bluetooth and IEEE 802.11, it is now possible to enhance the services of a radio by leveraging other devices that provide complementary services. For instance, a Bluetooth-enabled fax machine may be able to send a fax to a nearby laptop computer equipped with a software radio that supports the Bluetooth interface. Software radio's reconfiguration capability can support an almost infinite variety of service capabilities in a system.

- *Global mobility:* A number of communication standards exist today. In the 2G alone, there are IS-136, GSM, IS-95ICDMA1, and many other, less well known standards. The 3G technology tried to harmonize all the standards. However, there are many standards under the 3G umbrella. The need for transparency, i.e., the ability of radios to operate with some, preferably all, of these standards in different geographical regions of the world has fostered the growth of the software radio concept. Military services also face a similar issue with incompatible radio standards existing between as well as within branches of the military.

- *Compactness and power efficiency*: Multifunction, multimode radios designed using the "Velcro" approach of including separate silicon for each system can become bulky and inefficient as the number of systems increases. The software radio approach, however, results in a compact and, in some cases, a power-efficient design, especially as the number of systems increases, since the same piece of hardware is reused to implement multiple systems and interfaces.

- *Ease of manufacture:* RF components are notoriously hard to standardize and may have varying performance characteristics. Optimization of the components in terms of performance may take a few years and thereby delay product introduction. In general, digitization of the signal early in the receiver chain can result in a design that incorporates significantly fewer parts, meaning a reduced inventory for the manufacturer.

• *Ease of upgrades:* In the course of deployment, current services may need to be updated or new services may have to be introduced. Such enhancements have to be made without disrupting the operation of the current infrastructure. A flexible architecture allows for improvements and additional functionality without the expense of recalling all the units or replacing the user terminals. Vocoder technology, for example, is constantly improving to offer higher quality voice at lower bit rates. As new vocoders are developed, they can be quickly fielded in software radio systems. Furthermore, as new devices are integrated into existing infrastructures, software radio allows the new devices to interface seamlessly, from the air-interface all the way to the application, with the legacy network.

Users/customers expect service regardless of the geographical areas in which they travel and the wireless technologies that are in use in different regions in the world, but carrying several devices that cover the broad range of technology alternatives is impractical. Users expect one device to utilize services in all regions, which is possible only by reconfiguring the receiver to the air-interface standards used in the respective regions, By dynamically downloading the software to cover the needed air-interface standard, perhaps through transmission of the software configuration to the remote terminal, such over-the-air updates will allow for speedy implementation of software upgrades and new features.

1.4 OUR DESIGN

So far we have explained the software radio and its benefits. Since the major characteristic of software radio is that all the communication functionalities are defined in software. We have designed a complete digital radio which defined all the communication functionalities in software as shown in figure 1.2. We have used the latest techniques for all the functionalities especially for timing synchronization. Since one of biggest challenge for software radio is computational complexity because of unavailability of strong processors in small size a special care has been taken to reduce the computational complexity as low as possible.

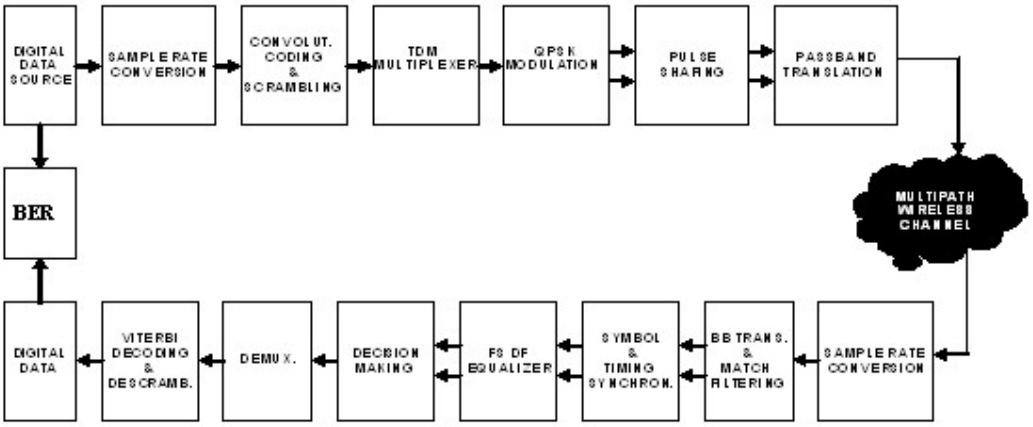

Figure 1.2: Our Design

In the preceding chapters of this report we will examine each block separately in detail.

CHAPTER 2

SAMPLE RATE CONVERSION

2.1 INTRODUCTION

Since different communications standards are based upon different master clock rates it is principally necessary to provide these different clock rates in a terminal that is to process signals according to those different standards. Due to the strong requirements on the quality of the clock, specifically required for the analog-to digital converter, it is reasonable to assume that only *one fixed* master clock will be provided in practical software radio applications. A solution to this conflict is to provide the different clock rates virtually by means of digital sample rate conversion (SRC). SRC is the task of converting the sample rate of a first digital signal to another sample rate resulting in a second digital signal, while a certain amount of information - usually in a limited frequency band - must not be corrupted [3]. In this chapter we will discuss the sample rate conversion in digital domain only.

2.2 DOWNSAMPLING BY INTERGER FACTOR (DECIMATION)

Decimation can be regarded as the discrete-time counterpart of sampling. Whereas in sampling we start with a continuous-time signal $x(t)$ and convert it into a sequence of samples $x(n)$, in decimation we start with a discrete-time signal $x(n)$ and convert it into another discrete-time signal $y(n)$, which consists of *sub-samples* of $x(n)$. Thus, the formal definition of *M*-fold decimation, or down-sampling, is defined by equation 2.1. In decimation, the sampling rate is reduced from F_s to F_s/M by discarding $M-1$ samples for every M samples in the original sequence.

$$y(n) = v(nM) = \sum_{k=-\infty}^{\infty} h(k)x(nM-k) \tag{2.1}$$

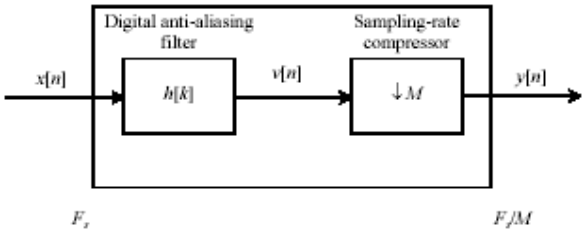

Figure 2.1: Block diagram notation of decimation, by a factor of *M*.

The block diagram notation of the decimation process is depicted in figure 2.1. An anti-aliasing digital filter precedes the down-sampler to prevent aliasing from occurring, due to the lower sampling rate. In figure 2.2, it illustrates the concept of 3-fold decimation i.e. $M = 3$. Here, the samples of $x[n]$ corresponding to $n = \ldots, -2, 1, 4,$ and $n = \ldots, -1, 2, 5, \ldots$ are lost in the decimation process. In general, the samples of $x[n]$ corresponding to $n \neq kM$, where *k* is an integer, are discarded in M-fold decimation. In figure 2.2 (b), it shows samples of the decimated signal $y[n]$ spaced three times wider than the samples of $x[n]$. This is not a coincidence. In real time, the decimated signal appears at a slower rate than that of the M. If the sampling frequency of $x[n]$ is then that of $y[n]$ is M.

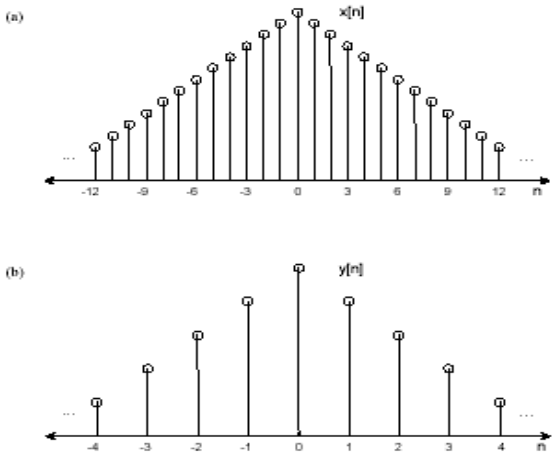

Figure 2.2: Decimation of a discrete-time signal by a factor of 3.

2.2.1 Frequency Domain Analysis of Decimation

The implications of aliasing caused by decimation are very similar to those in the case of sampling a continuous-time signal. In general, if the Fourier transform of a signal, $X(\theta)$, occupies the entire bandwidth from $[-\pi, \pi]$, then the

Fourier transform of the decimated signal, $X_{\downarrow M}(\theta)$, will be aliased because the spectrum now occupies the bandwidth [-$M\pi$, $M\pi$]. This is due to the superposition of the M shifted and frequency-scaled transforms. This is illustrated in figure 2.3 below, which shows the aliasing phenomenon for $M = 3$. From figure 2.3 it is clear that the cut off frequency for anti-aliasing filter must be less than equal to π / M.

Mathematically in the z-transform the relation between $x(n)$ and $y(n)$ can be derived.

$$\begin{aligned} Y(z) &= \sum_n y(n) z^{-n} \\ &= \sum_n x(nM) z^{-n} \\ &= X(z^{1/M}) \end{aligned} \tag{2.2}$$

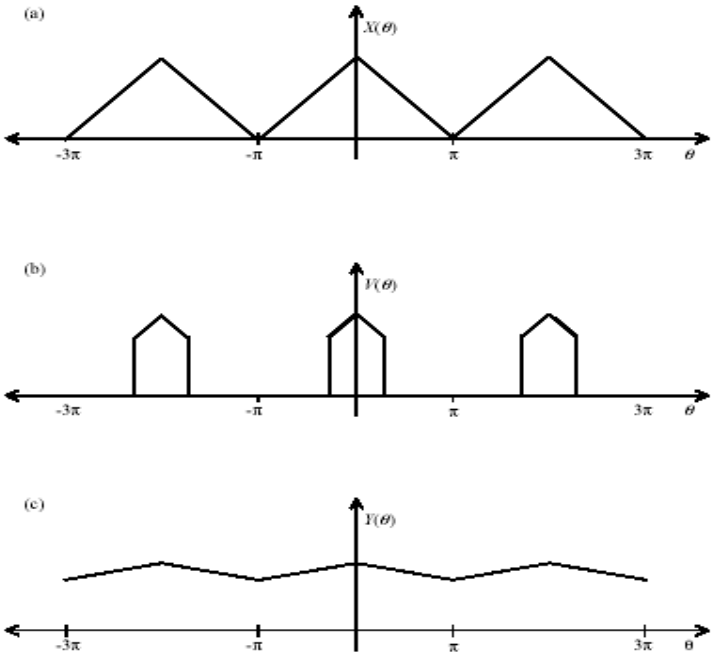

Figure 2.3: Aliasing caused by decimation. (a) Fourier transform of the original signal. (b) After anti-aliasing filter. (c) Fourier transform of the decimated signal.

Normally decimation process is implemented using a transversal structure as shown in figure 2.4 (a,b). First it is filtered and then at the output every Mth sample is taken out. But the question is why to compute those values which are not required because we are only interested in every Mth sample [6]. This

can be achieved by putting the decimator first and then perform the filtering as shown in figure 2.4 (c).

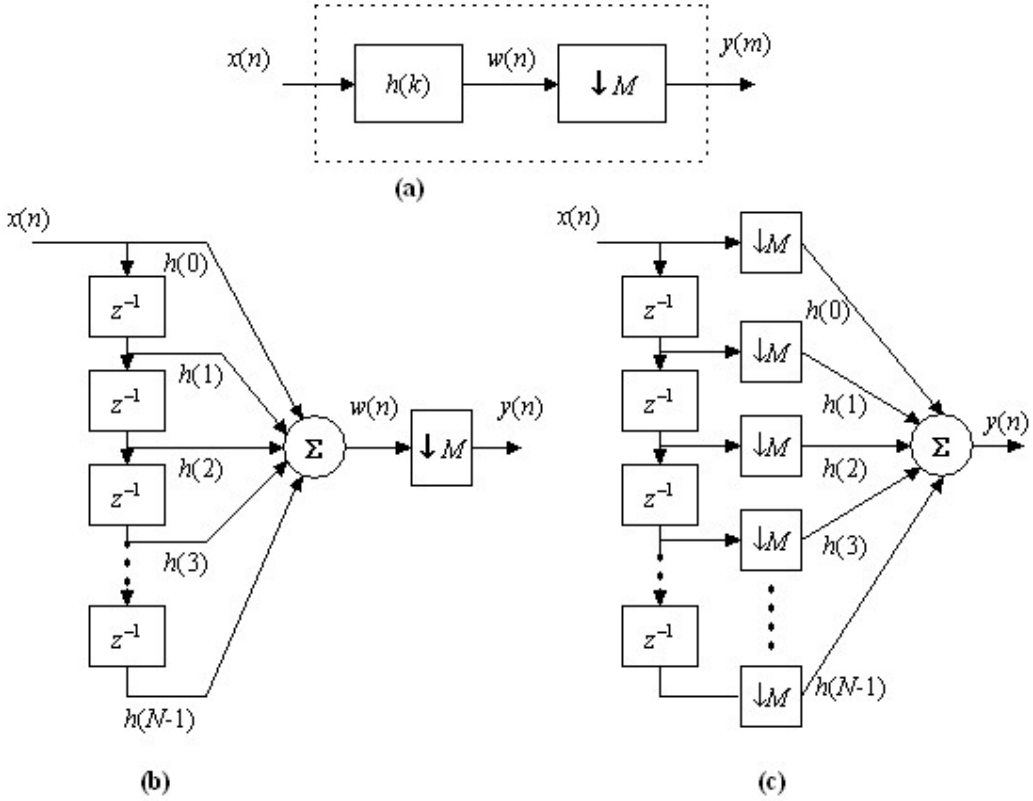

Figure 2.4: Efficient implementation of decimator

2.3 UPSAMPLING BY INTEGER FACTOR (INTERPOLATION)

Interpolation is the exact opposite of decimation. It is an information preserving operation, in that all samples of $x(n)$ are present in the expanded signal $y(n)$. The mathematical definition of L-fold interpolation is defined by equation 2.3 and the block diagram notation is depicted in figure 2.5. Interpolation works by inserting $(L-1)$ zero-valued samples for each input sample. The sampling rate therefore increases from F_s to LF_s. With reference to figure 2.5, the expansion process is followed by a unique digital low-pass filter called an *anti-imaging filter* [5]. Although the expansion process does not cause aliasing in the interpolated signal, it does however yield undesirable replicas in the signal's frequency spectrum. We shall see how this special filter, is necessary to remove these replicas from the frequency spectrum.

$$. \ y(n) = L \sum_{k=-\infty}^{\infty} h(k)w(n-k) \tag{2.3}$$

where

$$w(v) = \begin{cases} x(n/L) & \text{if } n/L \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}$$

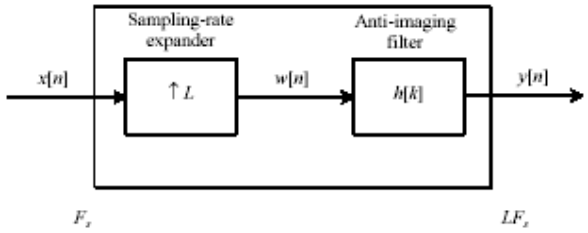

Figure 2.5: Block diagram notation of interpolation, by a factor of L.

In figure 2.6, it depicts 3-fold interpolation of the signal $x(n)$ i.e. $L = 3$.

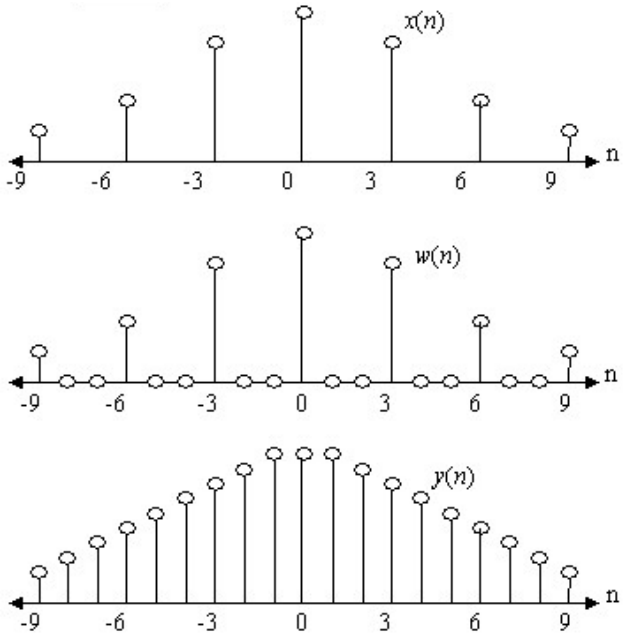

Figure 2.6: Interpolation of a discrete-time signal by a factor of 3.

The insertion of zeros effectively attenuates the signal by L, so the output of the anti-imaging filter must be multiplied by L, to maintain the same signal magnitude.

2.3.1 Frequency Domain Analysis of Interpolation

The effect of expansion on a signal in the frequency domain is illustrated in figure 2.7. Part (a) shows the Fourier transform of the original signal; part (b)

illustrates the Fourier transform of the signal with zeros added $W(\theta)$; and part (c) shows the Fourier transform of the signal after the interpolation filter. It is clearly visible that the shape of the Fourier transform is compressed by a factor *L* in the frequency axis and is also repeated *L* times in the range of [-π, π]. Despite the compression of the signal in the frequency axis, the shape of the Fourier transform is still preserved, confirming that expansion does not lead to aliasing [4]. These replicas are removed by a digital low-pass filter called an *anti-imaging filter* shown in figure 2.5. From figure it is very clear that cut off frequency for anti-imaging filter must be less than equal to π / L to remove the images.

Mathematically the z-transform relations are

$$\begin{aligned} Y(z) &= \sum_n y(n) z^{-n} \\ &= \sum_n x(n) z^{-n} \quad when\ n/L\ is\ \text{integer} \\ &= X(z^L) \end{aligned} \tag{2.4}$$

Interpolator can be implemented using the same transversal structure as the decimator. The flow diagram is shown in figure 2.8. However, this type of structure is very inefficient owing to the interpolation process, which introduces $(L-1)$ zeros between consecutive points in the signal. If *L* is large, then the majority of the signal components fed into the FIR filter are zero. As a result, most of the multiplications and additions are zero i.e. many pointless calculations. We will see very shortly that how design an efficient interpolator who can take care of such point less calculations

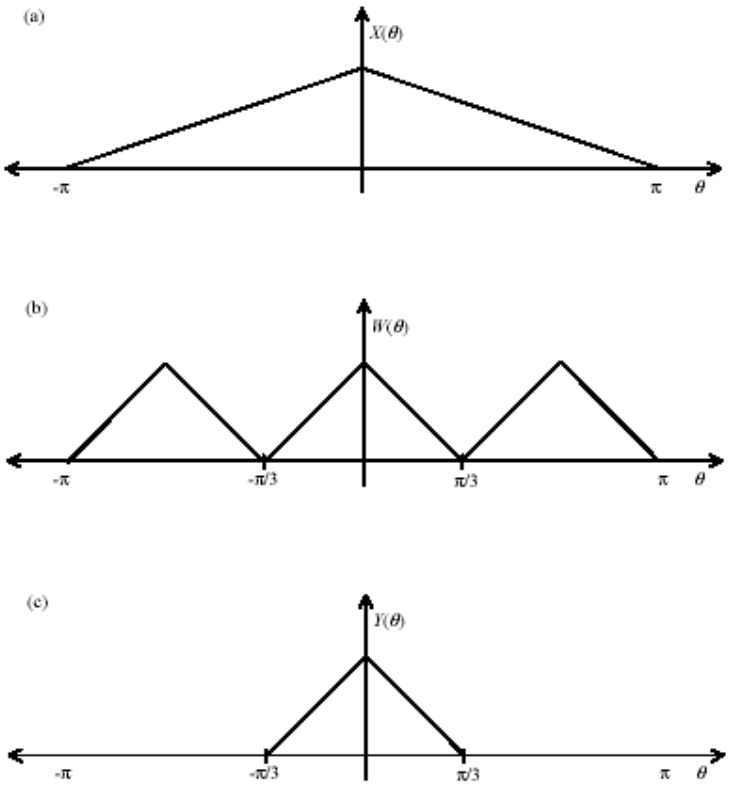

Figure 2.7: Expansion in the frequency domain of original signal (a) and the expanded signal (b).

2.4 CHANDING SAMPLING RATE BY NON INTEGER FACTOR

A common use of multirate signal processing is for sampling-rate conversion. Suppose a digital signal $x(n)$ is sampled at an interval T_1, and we wish to obtain a signal $y(n)$ sampled at an interval T_2. Then the techniques of decimation and interpolation enable this operation, providing the ratio T_1/T_2 is a rational number i.e. L/M. Sampling-rate conversion can be accomplished by L-fold expansion, followed by low-pass filtering and then M-fold decimation, as depicted in figure 2.9. It is important to emphasis that the interpolation should be performed first and decimation second, to preserve the desired spectral characteristics of $x(n)$. Furthermore by cascading the two in this manner, both of the filters can be combined into one single low-pass filter because both filters are linear. Actually while combining the filters we will select the filter which has the less cut off frequency from both of them.

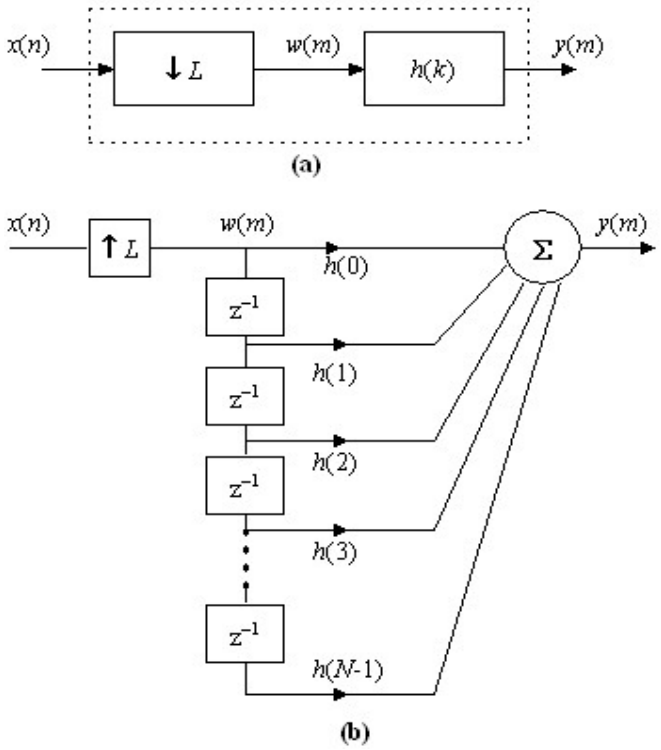

Figure 2.8: Interpolator Implementation flow diagram

An example of sampling-rate conversion would take place when data from a CD is transferred onto a DAT. Here the sampling-rate is increased from 44.1 kHz to 48 kHz. To enable this process the non-integer factor has to be approximated by a rational number:

$$\frac{L}{M} = \frac{44.1}{48} = \frac{160}{147} = 1.08844$$

Hence, the sampling-rate conversion is achieved by interpolating by L i.e. from 44.1 kHz to [44.1x160] = 7056 kHz. Then decimating by M i.e. from 7056 kHz to [7056/147] = 48 kHz.

2.5 MULTISAGE APPROACH

When the sampling-rate changes are large, it is often better to perform the operation in multiple stages, where *M*i (*L*i), an integer, is the factor for the stage *i*

$$M = M_1, M_2, \ldots, M_i \quad and \quad L = L_1, L_2, \ldots, L_I .$$

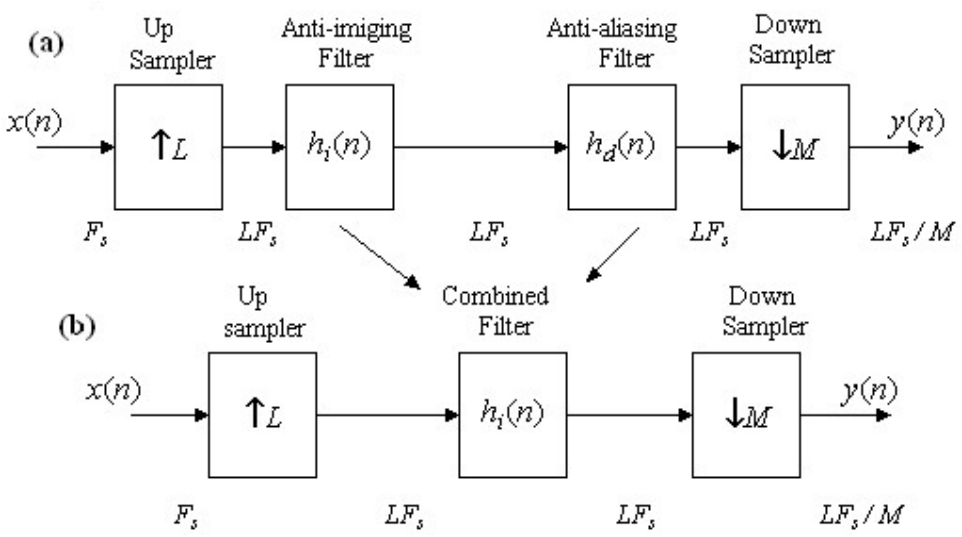

Figure 2.9: Sampling-rate conversion using (a) separate anti-imaging and anti-aliasing filters. (b) combined low pass filter

An example of the multistage approach for decimation is shown in figure 2.10. The multistage approach allows a significant relaxation of the anti-alias and anti-imaging filters, with a consequent reduction in the filter complexity [6]. The optimum number of stages is one that leads to the least computational effort in terms of either the multiplications per second (MPS), or the total storage requirement (TSR).

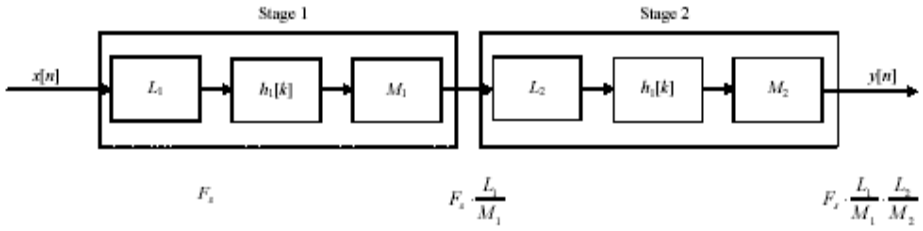

Figure 2.10: Multistage approach for the decimation process.

2.6 POLYPHASE STRUCTURES

Potential computational savings can be made within the process of decimation, interpolation, and sampling-rate conversion. *Polyphase filters* are the name given to certain realizations of multirate filtering operations, which facilitate computational savings in both hardware and software. Actually a filter is broken in to sub filters called polyphase filters. Their name polyphase

is because of the fact that they have same amplitude response and differ in phases.

We will derive the polyphase components for a factor 2. The extension to any number is straightforward. In z-domain any filter can be divided into polyphase components as

$$\begin{aligned} H(z) &= \sum_{n=0}^{N-1} h(n) z^{-n} \\ &= \sum_{n=0}^{N/2-1} h(2n) z^{-2n} + \sum_{n=0}^{N/2-1} h(2n) z^{-2n} \\ &= \underbrace{E_0(z^2) + z^{-1} E_1(z^2)}_{\text{Polyphase Components}} \end{aligned} \tag{2.5}$$

where $E_0(z^2)$ and $E_1(z^2)$ are the polyphase components.

2.6.1 Decimator

For decimation by a factor of *M* the anti-aliasing filter can be divided into *M* polyphase components. In figure 2.11 the decimation process using polyphase structure is shown.

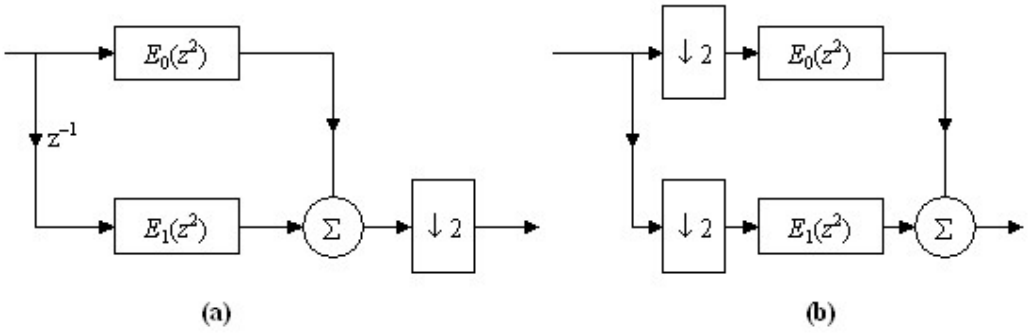

Figure 2.11: Decimation by a factor of 2 using polyphase structure

Now we will derive the time domain relation for decimation. If $y(m)$ is the decimated signal and $x(n)$ is the input signal then and $h(n)$ is the anti-aliasing filter then decimator filter output is

$$y(m) = \sum_{n=0}^{N-1} h(n) x(Mm - n) \tag{2.6}$$

By changing the variables

$$y(m) = \sum_{k=0}^{N-1} h(mM - k) x(k) \tag{2.7}$$

Signal parallelization is

$$y(k) = \sum_{q=0}^{M-1} \sum_{i=0}^{N/M-1} h_q(i) x_q(k-i) \tag{2.8}$$

where the polyphase components of the decimation filter are computed by the relation

$$h_q(i) = h(iM + q) \tag{2.9}$$

and inputs signal to polyphase components are given by the relation

$$x_q(i) = x(iM - q) \tag{2.10}$$

Actually you can see that y(k) is computed in parallel. Every polyphase filter is computing a single output for every value of n in parallel. Since we require every Mth sample so we can only use a single polyphase filter for a single value of n. This can be done using a commutator as shown in figure 2.12. They are also called type-1 polyphase structure. If $h(n)$ is the anti-aliasing filter , then the M polyphase components are:

$$\begin{aligned} h_0 &= h(0), h(M), h(2M), \ldots \\ h_1 &= h(1), h(M+1), h(2M+1), \ldots \\ &\vdots \\ h_{M-1} &= h(M-1), h(2M-1), h(3M-1), \ldots \end{aligned} \tag{2.11}$$

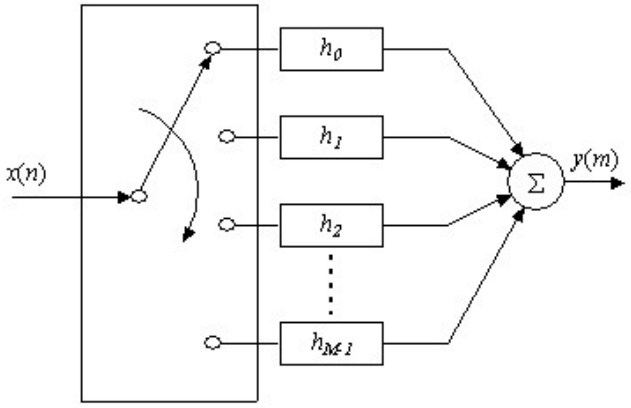

Figure 2.12: Decimation using type-1 polyphase structure.

The beauty of polyphase structure is that they are computing those samples which are not required but actually throwing them away. You can see in the above figure that there is no decimator. The input is at the rate N and the output is at the rate N/M

2.6.2 Interpolator

In the previous section we have seen how to design a computationally efficient polyphase decimator. The interpolation is just the reverse process of decimation. Using the same notations as earlier the time domain derivation for interpolator is:

Interpolator filter output is

$$y(m) = \sum_{n=0}^{N-1} h(n)x(m-nL) \tag{2.12}$$

After parallelization the output signal is

$$y_p(k) = \sum_{i=0}^{N/M-1} h_p(i)x(k-i) \tag{2.13}$$

and the relation for polyphase components is

$$h_p(i) = h(iL+p), \qquad p = 0,1,\ldots,L-1 \tag{2.14}$$

you can the components are the same as for decimator. The flow diagram for interpolation process using polyphase structure is shown in figure 2.13.

You can see in the above figure there is no insertion of zeros. The input is at rate N and the output is at rate NL. Actually the pointless calculations which are there in the case of transversal structure are avoided.

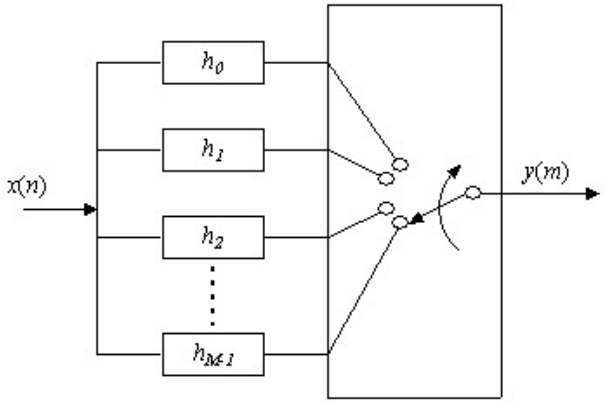

Figure 2.13: Interpolation using type-2 polyphase structure

2.6.3 Sample Rate Conversion by Non-integer Factor

So far we have discussed interpolation and decimation using polyphase structures. It is also possible to change the sampling rate by non-integer factors using polyphase structure.

As described earlier for a non-integer SRC the anti-imaging and anti-aliasing filters can be combined into a single filters. Actually the filter which has lower cut-off frequency is selected. Now if we have to change the

sampling rate by L / M then for polyphase implementation the combined filter should be divided into $L \times M$ polyphase components. The polyphase structure for L=2, and M=3 is shown in figure 2.14.

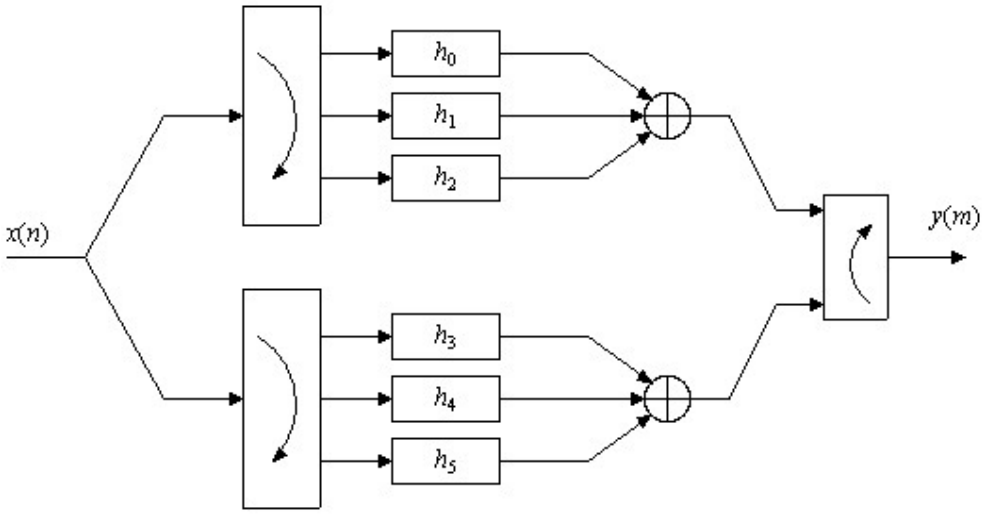

Figure 2.14: Sample rate conversion using polyphase structures for *L*=2 and *M*=3

In figure 2.14 L=2 and M=3 therefore we have 6 polyphase components $h_0, h_1, \ldots, h_5$. These polyphase components can be obtained by using the relation in equation 2.11.

2.7 RELATIONSHIP FOR ESTMATING FILTER LENGTH, *N*

Now one question arises how many taps for the combined filter or any anti-imaging or anti-aliasing filters. The answer is the filter taps can be determined by using the relation derived by Herrman in 1973. The relation is for the optimal method (equaripple method) for lowpass filters only.

$$N = \frac{D_\infty(\delta_P, \delta_S)}{\Delta F} - f(\delta_P, \delta_S)\Delta F + 1 \qquad (2.15)$$

where ΔF is the width of the transition band normalized to the sampling frequency, δ_P is the passband ripple or deviation and δ_S is the stopband ripple or deviation and

$$D(\delta_P, \delta_S) = \log_{10}\delta_S\left[a_1(\log_{10}\delta_P)^2 + a_2\log_{10}\delta_P + a_3\right] + \left[a_4(\log_{10}\delta_P)^2 + a_5\log_{10}\delta_P + a_6\right]$$

$$f(\delta_P, \delta_S) = 11.01217 + 0.51244[\log_{10}\delta_P - \log_{10}\delta_S]$$

$$a_1 = 5.309 \times 10^{-3} \quad a_2 = 7.114 \times 10^{-2} \quad a_3 = -4.761 \times 10^{-1}$$

$$a_4 = -2.66 \times 10^{-3} \quad a_5 = -5.941 \times 10^{-1} \quad a_6 = -4.278 \times 10^{-1}$$

2.8 SIMULATION RESULTS

2.8.1 Interpolation

A simulation for interpolation by a factor of 3 is run in matlab. The spectrum of the signal is given below before interpolation.

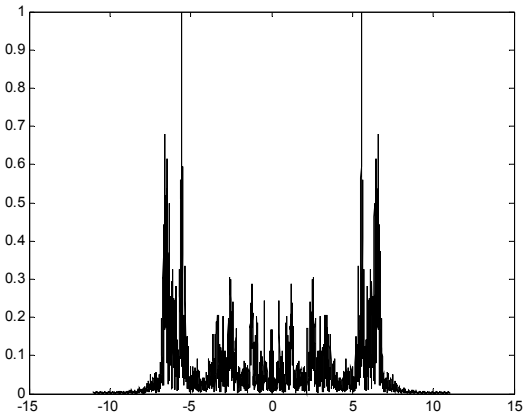

Figure 2.15: Spectrum before interpolation

After insertion of zeros the spectrum is given on next page.

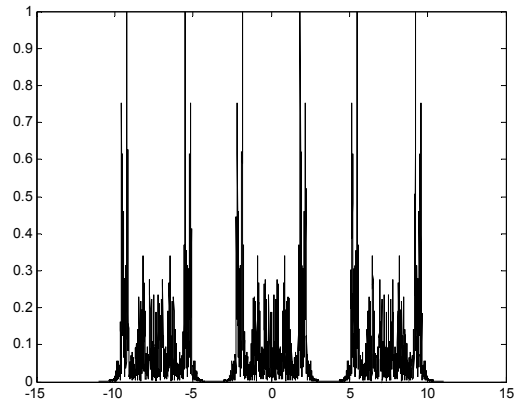

Figure 2.16: Spectrum after insertion of zeros interpolation

The spectrum of interpolator filter is

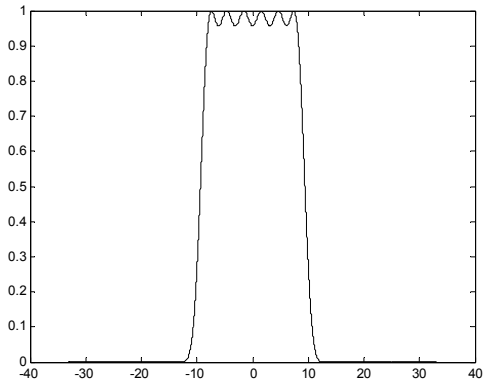

Figure 2.17: Spectrum of interpolator filter

At the output of interpolator filter the spectrum is

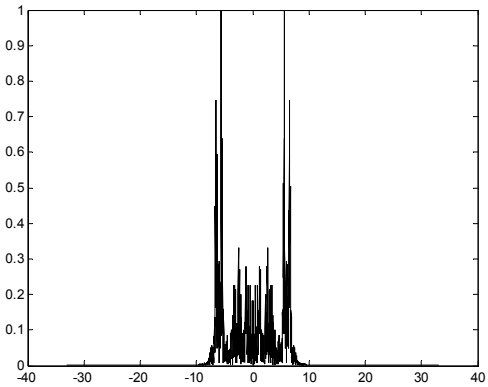

Figure 2.18: Spectrum of interpolator filter using transversal structure.

The spectrum using the polyphase approach is given below which is same as using the conventional approach in figure 18.

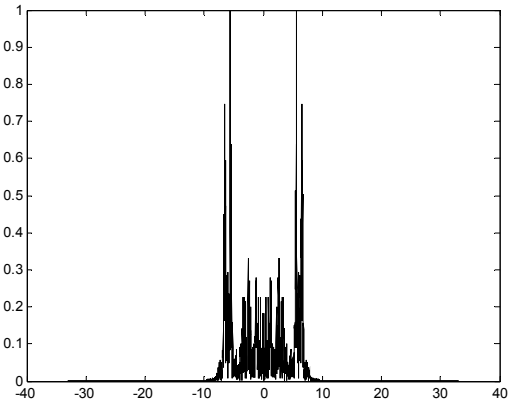

Figure 2.19: Spectrum of interpolator filter using polyphase structure.

2.8.2 Decimation

A simulation is run for decimation by a factor of 2. The spectrum of signal before decimation is

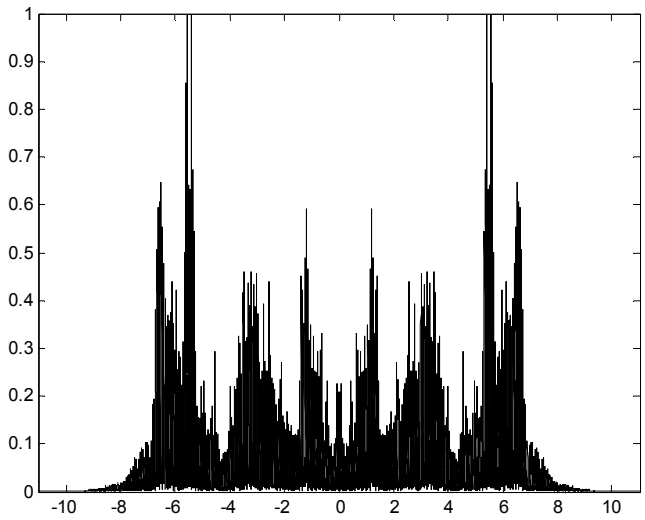

Figure 2.20: Spectrum of the signal before decimation

The spectrum of decimator filter is given below

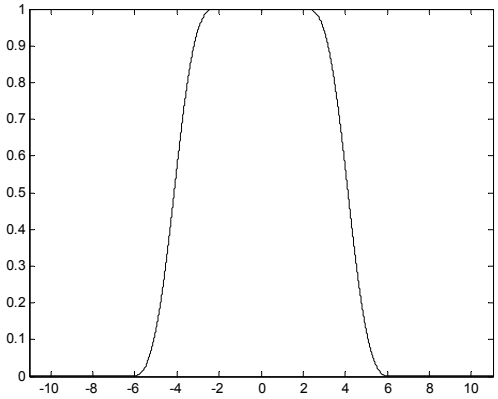

Figure 2.21: Spectrum of decimator filter

The spectrum at the out of decimator filter using transversal structure is

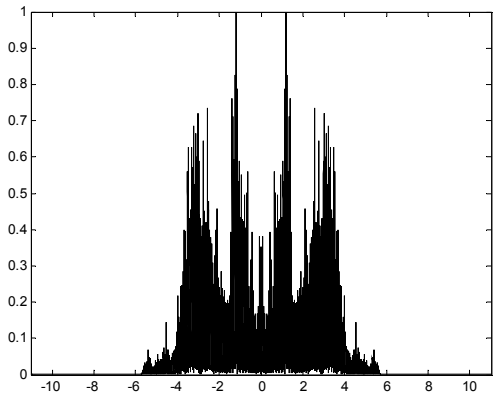

Figure 2.22: Spectrum at the output of decimator filter

Spectrum after decimation by 2 is given below

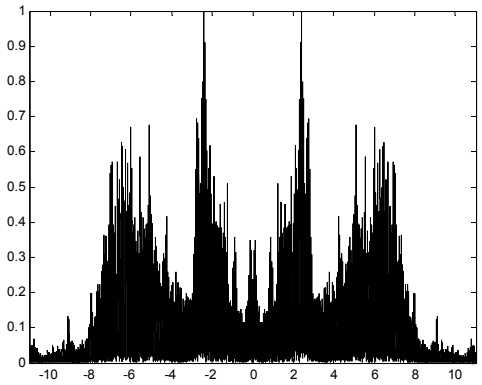

Figure 2.23: Spectrum after decimation by 2

The spectrum at the out of decimator filter using the polyphase structure for decimator filter is given below which is same as for transversal filter.

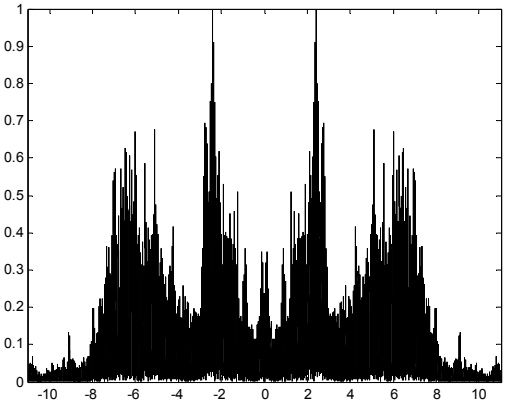

Figure 2.24: Spectrum after decimation using polyphase structure

CHAPTER 3

THE WIRELESS CHANNEL

3.1 PHYSICAL MODELING FOR WIRELWSS CHANNELS

Wireless channels operate through electromagnetic radiation from transmitter to receiver. In principle, one could solve the electromagnetic field equations, in conjunction with the transmitted signal, to find the electromagnetic field impinging on the receiver antenna. This would have to be done taking into account the obstructions caused by ground, buildings, vehicles, etc. in the vicinity of this electromagnetic wave. By obstructions, we mean not only objects in the line of sight between transmitter and receiver, but also objects in locations that cause non-negligible changes in the electromagnetic field at the receiver.

The wavelength λ of electromagnetic radiation at any given frequency f is given by $\lambda=c/f$, where $c = 3 \times 10^8$ mps is the velocity of light. The wavelength is thus a fraction of a meter, so to calculate the electromagnetic field at a receiver, the locations of the receiver and the obstructions would have to be known within sub meter accuracies. The electromagnetic field equations are therefore too complex to solve, especially on the fly as vehicles move. Thus we have to ask what we really need to know about these channels, and what approximations might be reasonable. The solution is to construct stochastic models of the channel, assuming that different channel behaviors appear with different probabilities, and change over time in particular stochastic ways.

3.1.1 Free space, fixed transmitting and receiving antennas

First consider a fixed antenna radiating into free space. In the far field, the electric field (the far field is far enough away from the transmitting antenna that the antenna looks like a point) and magnetic field at any given location are perpendicular both to each other and to the direction of propagation from

the antenna. They are also proportional to each other, so it is sufficient to know only one of them (just as in wired communication, where we view a signal as simply a voltage waveform or a current waveform). In response to a transmitted sinusoid $\cos(2\pi ft) = \text{Real}[\exp(2\pi ft)]$, we can express the electric far field at time t, in its particular direction, as

$$E(f,t,(r,\theta,\psi)) = \frac{\Re[a_s(\theta,\psi,f)\exp\{2\pi if(t-r/c)\}]}{r} \tag{3.1}$$

Here (r, θ, ψ) represents the point u in space at which the electric field is being measured, where r is the distance from the transmitting antenna to u and where (θ, ψ) represent the vertical and horizontal angles from the antenna to u. $c = 3 \times 10^8$ m/s is the velocity of light, and $a_s(\theta, \psi, f)$ is the radiation pattern of the sending antenna at frequency f in the direction (θ, ψ). a_s also contains a scaling factor to account for antenna losses and an initial phase. Note that the phase of the field varies with fr/c, corresponding to the delay caused by the radiation traveling at the speed of light.

We are not concerned here with actually finding the radiation pattern for any given antenna, but only with recognizing that antennas have radiation patterns, and that the free space far field behaves as above.

It is important to observe that as the distance r increases, the electric field goes down as r^{-1} and thus the power per square meter in the free space wave goes down as r^{-2}. This is expected, since if we look at concentric spheres of increasing radius r around the antenna, the total power radiated through the sphere remains constant, but the surface area increases with r^2. Thus the power per unit area must decrease as r^{-2}. We will see shortly that this r^{-2} reduction of power with distance is often not valid when there are obstructions to free space propagation.

Next, suppose there is a fixed receiving antenna at location $u = (r, \theta, \psi)$. The received waveform (in the absence of noise) in response to the above transmitted sinusoid is then

$$E_r(f,t,u) = \frac{\Re[a(\theta,\psi,f)\exp\{2\pi if(t-r/c)\}]}{r} \tag{3.2}$$

Where $a(\theta, \psi, f)$ is the product of the antenna patterns of transmitting and receiving antennas in the given direction. Again, $a(\theta, \psi, f)$ takes into account

losses and phase changes at both antennas. We have done something a little strange here in starting with the free space field at u in the absence of an antenna. Placing a receiving antenna there changes the electric field in the vicinity of u, but this is taken into account by the antenna pattern of the receiving antenna. Now suppose, for the given u, that we define

$$\hat{h}(f) = \frac{a(\theta, \psi, f)\exp(-2\pi i f r / c)}{r} \tag{3.3}$$

We then have $E_r(f, t, u) = \text{Real}[\hat{h}(f)\exp(2\pi i f t)]$, We have not yet mentioned it yet, but (3.1) and (3.2) are both linear in the input. That is, the received field (waveform) at u in response to a weighted sum of transmitted waveforms is simply the weighted sum of responses to those individual waveforms. Thus, $\hat{h}(f)$ is the system function for an LTI (linear time invariant) channel, and its inverse Fourier transform is the impulse response. For this simple example, the channel is described in exactly the same way as the wire line channels that we have been dealing with. The only need for understanding electromagnetism is to determine what this system function is. We will find in what follows that linearity is a good assumption for all the wireless channels we consider, but that the time invariance does not hold when either the antennas or obstructions are in relative motion.

3.1.2. Free space, moving antenna

Next consider the fixed antenna and free space model above with a receiving antenna that is moving with velocity v in the direction of increasing distance from the transmitting antenna. That is, we assume that the receiving antenna is at a moving location described as $u(t) = (r(t), \theta, \psi)$ with $r(t) = r_0 + vt$. Using (3.1) to describe the free space electric field at the moving point $u(t)$ (for the moment with no receive antenna), we have

$$E(f, t, (r_o + vt, \theta, \psi)) = \frac{\Re[a_s(\theta, \psi, f)\exp\{2\pi i f(t - r_o/c - vt/c)\}]}{r_o + vt} \tag{3.4}$$

Note that we can rewrite $f(t - r_0/c - vt/c)$ as $f(1-v/c)t - fr_0/c$. Thus the sinusoid at frequency f has been converted to a sinusoid of frequency $f(1-v/c)$; there has been a Doppler shift of $-fv/c$ due to the motion of the observation point. Physically, each successive crest in the transmitted sinusoid has to travel a little further before it gets observed at this moving observation point. If the

antenna is now placed at $u(t)$, and the change of field due to the antenna presence is again represented by the receiver antenna pattern, the received waveform, in analogy to (3.2), is

$$E_r(f,t,(r_o+vt,\theta,\psi)) = \frac{\Re[a(\theta,\psi,f)\exp\{2\pi i(f(1-v/c)t - fr_o/c)\}]}{r_o+vt} \qquad (3.5)$$

This channel cannot be represented as an LTI channel. If we ignore the time varying attenuation in the denominator of (3.5), however, we can represent the channel as an LTI system followed by a translation of each frequency f by a Doppler shift $-fv/c$. For narrow band communication around a carrier f_c, this is essentially a fixed frequency shift of $-f_cv/c$. We will come back to discussing the importance of this Doppler shift and of the time varying attenuation after considering the next example.

The above analysis does not depend on whether it is the transmitter or the receiver (or both) that are moving. So long as $r(t)$ is interpreted as the distance between the antennas (and the relative orientations of the antennas are constant), (3.4) and (3.5) are valid.

3.1.3. Moving antenna, reflecting wall

Consider figure 3.1 below in which there is a fixed sending antenna transmitting the sinusoid $\cos(2\pi ft)$, a mobile receiving antenna, and a single perfectly reflecting large fixed wall. We assume that in the absence of the receiving antenna, the electromagnetic field at the point where the receiving antenna will be placed is the sum of the free space field coming from the sending antenna plus a reflected wave coming from the wall.

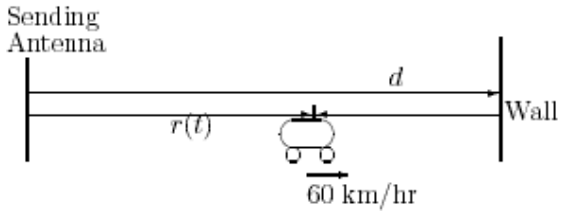

Figure 3.1: Illustration of a direct path and a reflected path

As before, in the presence of the receiving antenna, the perturbation of the field due to the antenna is represented by the antenna pattern. Additional assumptions are that the presence of the receiving antenna does not

appreciably affect the plane wave impinging on the wall and that the wall is large enough that the reflected wave can be modelled as a plane wave. In essence, what we are doing is approximating the solution of Maxwell's equations by an approximate method called ray tracing. The assumption here is that the received waveform can be approximated by the sum of the far field wave from the sending transmitter plus a reflected far field wave via each reflecting obstacle.

In the present situation, if we assume that the wall is very large, the reflected wave at a given point is the same (except for a sign change) as the free space wave that would exist on the opposite side of the wall if the wall were not present (see Figure 3.2). This means that the reflected wave from the wall has the intensity of a free space wave at a distance equal to the distance to the wall and then back to the cell phone, i.e., $2d - r(t)$. Taking $r(t) = r_0 + vt$ again, using (3.5) for both the direct and the reflected wave, and assuming the same antenna gain a for both waves,

$$E_r(f,t) = \frac{\Re[a\exp\{2\pi i(f(1-v/c)t - fr_0/c)\}]}{r_0+vt} - \frac{\Re[a\exp\{2\pi i(f(1+v/c)t + (fr_0-2fd)/c)\}]}{2d-r_0-vt} \tag{3.6}$$

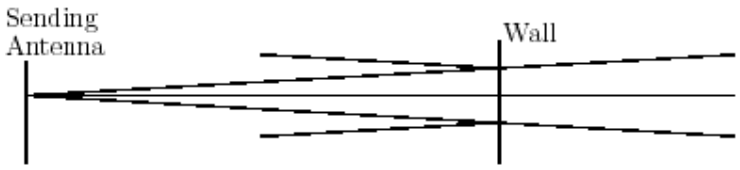

Figure 3.2: Relation of reflected wave to wave without wall.

The first term, the direct wave, is a sinusoid of slowly decreasing magnitude at frequency $f(1 - v/c)$. The second is a sinusoid of smaller but increasing magnitude at frequency $f(1+v/c)$. The combination of the two creates a beat frequency at fv/c. As an example, if the cell phone is moving at 60 km/hr and f = 900MH, this beat frequency is 50Hz. The waveform can be visualized most easily when the cell phone is much closer to the wall than to the sending antenna. In this case we can approximate the denominator of the second term by $r_0 + vt$. Then, combining the exponentials, we get

$$E_r(f,t) \approx \frac{2\sin[2\pi t\frac{fv}{c}+\frac{2\pi f(r_o-d)}{c}]\times \Re[-a\exp\{2\pi i(ft-fd/c)\}]}{r_o+vt} \tag{3.7}$$

This is the product of two sinusoids, one at the input frequency f, which is typically on the order of GHz, and the other at the Doppler shift fv/c, which might be on the order of 50Hz. Thus the response to a sinusoid at f is another sinusoid at f whose amplitude is varying with peaks going to zeros every 5 ms or so. This is called multipath fading. Note that in (3.6) we are viewing the response as the sum of two sinusoids, each of different frequency, while in (3.7), we are viewing the response as a single sinusoid of the original frequency with a time varying amplitude. These are just two different ways to view the same phenomenon.

We now see why we have partially ignored the denominator terms in (3.6) and (3.7). When the difference between two paths changes by a quarter wavelength, the phase difference between the responses on the two paths change by $\pi/2$, which causes a very significant change in the overall received amplitude. Since the carrier wavelength is very small relative to the path lengths, the time over which this phase effect causes a significant change is far smaller than the time over which the denominator terms cause a significant change. The effect of the phase changes is on the order of milliseconds, whereas the effect of changes in the denominator is relevant over periods of seconds or minutes. In terms of modulation and detection, the time scales of interest are in the range of milliseconds and less, and the denominators are effectively constant over these periods of interest.

The reader might notice that we are constantly making approximations in trying to understand wireless, much more so than for wired communication. This is partly because the standard LTI assumptions of wired communication are taught from the sophomore year on, and questioning those assumptions is rarely necessary. The wireless systems here are typically time varying, and appropriate models depend very much on the time scales of interest. For wireless systems, the most important issue is what approximations to make. Solving and manipulating equations is far less important. Thus it is important to understand these modeling issues thoroughly.

3.1.4. Reflection from a Ground Plane

Consider a transmitting and receiving antenna, both above a plane surface such as a road **(see figure 3)**. If the angle of incidence between antenna and road is sufficiently small, then even a dielectric will reflect most of the incident wave, with a sign change. When the horizontal distance r between the antennas becomes very large relative to their vertical displacements from the ground plane, a very surprising thing happens. In particular, the difference between the direct path length and the reflected path length goes to zero as r^{-1} with increasing r.

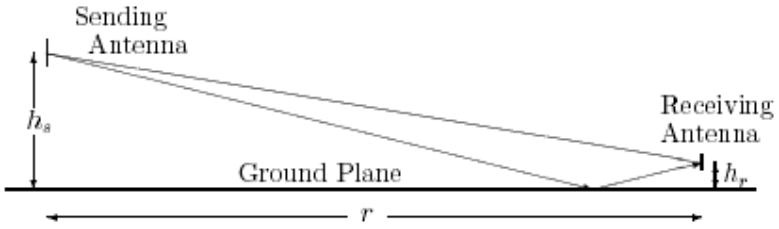

Figure 3.3: Illustration of a direct path and a reflected path of a ground plane.

When r is large enough, this difference between the path lengths becomes small relative to a wavelength c/f. Since the sign of the electric field is reversed on the reflected path, these two waves start to cancel each other out. The electric wave at the receiver is then attenuated as r^{-2}, and the received power goes down as r^{-4}. What this example shows is that the received power can decrease with distance considerably faster than r^{-2} in the presence of disturbances to free space. This situation is particularly important in rural areas where base stations tend to be placed on roads. Note, however, that the way the power decreases with distance is both helpful and harmful. It is helpful in reducing the interference between adjoining cells, but it is harmful in reducing the coverage of cells. As cellular systems become more popular, however, the major determinant of cell size is the number of cell phones in the cell. The size of cells has been steadily decreasing, and one talks of micro cells and pico cells as a response to this effect.

3.1.5. Shadowing

Shadowing is a phenomenon that occurs when partially absorbing materials lie between the sending and receiving antenna. This is called shadowing because it is similar to the effect of clouds partly blocking sunlight. Shadowing occurs when cell phones are inside buildings and the electromagnetic wave must pass through building walls. It also occurs when an outside cell phone is temporarily shielded from the base station by a building or some other structure.

The effect of shadow fading differs from multipath fading in two important ways.

- First, the duration of a shadow fade lasts for multiple seconds or minutes. For this reason, shadow fading is often called slow fading and multipath fading is called fast fading.
- Second, the attenuation due to shadowing is exponential in the width of the barrier that must be passed through. Thus the overall attenuation contains not only the r^{-2} effect of free space transmission, but also the exponential attenuation over the depth of the obstructing material.

3.1.6. Moving antenna, multiple reflectors

Dealing with multiple reflectors, under the assumption of ray tracing, is in principle simply a matter of modeling the received waveform as the sum of many responses from different paths rather than just two paths. We have seen enough examples, however, to understand that finding the magnitude and phase of these responses is no simple task. Even for the very simple large wall assumed in figure 3.1, the reflected field calculated in (3.6) is valid only at small distances from the wall relative to the dimensions of the wall. At very large distances, the total power reflected from the wall is proportional to both d^{-2} and to the cross section of the wall. The part of this reaching the receiver is proportional to $(d - r(t))^{-2}$. Thus the power attenuation from transmitter to receiver (for the large distance case) is proportional to $[d (d - r(t)]^{-2}$ rather than to $[2d - r(t)]^{-2}$. This shows that ray tracing must be used with some caution. Fortunately, however, linearity still holds in these more complex cases. Another type of reflection is known as scattering and can occur in the atmosphere or in reflections from very rough objects. Here there are a very

large number of individual paths, and the received waveform is better modeled as an integral over infinitesimally small paths rather than as a sum.

Knowing how to find the amplitude of the reflected field from each type of reflector above is an important topic if our objective is trying to determine where to place base stations, since this type of analysis is helpful in determining the coverage of a base station (although ultimately experimentation is necessary). Studying this in more depth, however, would take us too far into electromagnetic theory. In addition, we are primarily interested in questions of modulation, detection, multiple access, and network protocols rather than location of base stations. Thus, we turn our attention to understanding the nature of the aggregate received waveform, given a representation for each reflected wave. Thus we turn to modeling the input/output behavior of a channel rather than the detailed response on each path.

3.2 INPUT/OUTPUT MODEL FOR WIRELESS CHANNELS

Now we will derive the input/output model for wireless channel and show that the multipath effects can be modeled as a linear time varying system. Then we obtained the baseband representation of this model. The continuous time model is sampled to obtain a discrete time model. Finally we add noise in our model.

Suppose a transmitting antenna sends a sinusoid, $\cos(2\pi ft)$, which is received at a receiving antenna after reflection from some intermediate object. The response will be a function of each antenna pattern and of the intermediate object's reflection pattern. In addition, there will be an attenuation factor that is a function of the distance from transmitting antenna to reflector and from reflector to receiving antenna. To describe this path in terms of an input/output relationship between transmitter and receiver, we simply multiply all of these attenuation terms together as a single attenuation factor $\alpha_j(t)$ at time t from transmitter to receiver via a given path j. For the example of a perfectly reflecting wall in the previous section, then,

$$\alpha_1(t) = \frac{|\alpha|}{r_o + vt} \quad , \quad \alpha_2(t) = \frac{|\alpha|}{2d - r_o - vt} \tag{3.8}$$

Where the first expression is for the direct path and the second for the reflected path. Similarly, we define $\tau_j(t)$ as the propagation delay on path j from transmitter to receiver. Thus, for the reflecting wall example,

$$\tau_1(t) = \frac{r_o + vt}{c} - \frac{\angle\phi_1(f)}{2\pi f}, \quad \tau_2(t) = \frac{2d - r_o - vt}{c} - \frac{\angle\phi_2(f)}{2\pi f} \tag{3.9}$$

The term $\angle\phi_j(f)$ here is to account for possible phase changes at the transmitter, reflector, and receiver. For the example here, there is a phase reversal at the reflector so we can take $\phi_1 = 0$ and $\phi_2 = \pi$. With these definitions, the response to a sinusoid for the reflecting wall example can be expressed as

$$E_r(f,t) = \Re[\alpha_1(t)\exp\{i2\pi f(t-\tau_1(t))\}] + \Re[\alpha_2(t)\exp\{i2\pi f(t-\tau_2(t))\}] \tag{3.10}$$

For an arbitrary number *k* of paths, this expression becomes

$$E_r(f,t) = \sum_{j=1}^{k} \Re[\alpha_j(t)\exp\{i2\pi f(t-\tau_j(t))\}] \tag{3.11}$$

In the previous lecture, our focus was on the electromagnetic effects which give rise to time varying attenuation and path delay (along with the very notion of multiple propagation paths). Today, we abstract from these electromagnetic effects to study their effect on communication. The attenuations and path delays are now taken as given and we want to find an input/output characterization of the channel. We will use the physical mechanisms to get some order-of-magnitude sense of how the parameters vary with time, but otherwise, we simply explore the consequences of the assumed sinusoidal response in (3.11).

The effect of the Doppler shift is not immediately evident in **(3.11).** Recall that the Doppler shift D_j on path j is defined as $-fv_j/c$ where v_j is the velocity at which the path length is changing. Thus we can express $\tau_j(t)$ in (3.9) as

$$\tau_j(t) = \tau'_j - \frac{D_j t}{f} \tag{3.12}$$

Here τ'_j is assumed to be constant with respect to both t and f. D_j is linearly increasing in f, so that D_j / f is not a function of f, and thus $\tau_j(t)$ is also independent of f. The attenuations in (3.11) are usually slowly varying functions of frequency. These variations follow from the time-varying path lengths (as in (3.8)) and also from frequency dependent antenna gains. For

bands that are narrow relative to the carrier frequency, we can safely omit this frequency dependence. As we see later, however, there is an important frequency dependence in (3.11) that arises from multiple paths at different delays and Doppler shifts. The behavior of (3.11) does not depend critically on the number of paths, k, so this will be suppressed from now on.

3.2.1 Time-varying System Functions

We now derive a time-varying system function and then a time-varying impulse response for the above channel; the procedure is quite similar to that for linear time invariant (LTI) channels. View (3.11) as giving the response to sinusoids at arbitrary frequencies (within the band of interest). Define the *time-varying system function* $\hat{h}(f, t)$ as

$$\hat{h}(f,t) = \sum_j \alpha_j(t) \exp\{i2\pi f \tau_j(t)\} \tag{3.13}$$

Substituting this in (3.11), we see that the response to an input $cos(2\pi ft)$ is $\Re[\hat{h}(f,t)\exp\{i2\pi ft\}]$. More generally, the response to an input $cos(2\pi ft + \phi)$ is $\Re[\hat{h}(f,t)\exp\{i2\pi ft + \phi\}]$. As usual with system functions, it is convenient to define $\hat{h}(f, t)$ for negative frequencies as $\hat{h}(f, t) = \hat{h}^*(f, t)$. We can then view the response to an input $exp(i2\pi ft)$ as $\hat{h}(f, t)exp(i2\pi ft)$ for both $f>0$ and $f<0$. Using linearity, the response to a weighted sum of sinusoids, say

$$x(t) = \sum_k x_k \exp\{i2\pi f_k t\}$$

$$y(t) = \sum_k x_k \hat{h}(f_k, t) \exp\{i2\pi f_k t\} \tag{3.14}$$

We can represent any input $x(t)$ (in the frequency band of interest) by a Fourier transform

$$x(t) = \int_{-\infty}^{\infty} \hat{x}(f) \exp(i2\pi ft) df \quad ; \quad \hat{x}(f) = \int_{-\infty}^{\infty} x(t) \exp(-i2\pi ft) dt$$

Using linearity on a continuum of sinusoids in the same way as on the sum in (3.14), the response to $x(t) = \int_{-\infty}^{\infty} \hat{x}(f) \exp(i2\pi ft) df$ is

$$y(t) = \int_{-\infty}^{\infty} \hat{x}(f) \hat{h}(f,t) \exp(i2\pi ft) df \tag{3.15}$$

There is a temptation here to blindly imitate the theory of LTI linear systems and to confuse the Fourier transform of $y(t)$, namely $\hat{y}(f)$, with $\hat{x}(f)\hat{h}(f,t)$. This is wrong mathematically whenever $\hat{h}(f, t)$ is a non-constant function of t; this dependence on t prevents taking the Fourier transform of (3.8) in any straightforward way.

Confusing $\hat{y}(f)$, with $\hat{x}(f)\hat{h}(f,t)$ is also wrong physically. The response, for a given f, to $\hat{x}(f)\exp(i2\pi ft)$ is $\hat{x}(f)\hat{h}(f,t)\exp(i2\pi ft)$. This is a narrow band waveform rather than a sinusoid because of Doppler shifts. This means that $\hat{y}(f)$ at a given f also depends on $\hat{x}(f')$ over a range of f'.

Finally, confusing $\hat{y}(f)$, with $\hat{x}(f)\hat{h}(f,t)$ is non sensical, because $\hat{y}(f)$ is not a function of t and $\hat{x}(f)\hat{h}(f,t)$ is.

3.2.2. The Impulse Response and the Convolution Equation

Fortunately, (3.15) can still be used to derive a very satisfactory form of impulse response and convolution equation. Define the time-varying impulse response $h(\tau, t)$ as the inverse Fourier transform (in τ) of $\hat{h}(f,t)$, where t is viewed as a parameter. In particular,

$$h(\tau,t)=\int_{-\infty}^{\infty}\hat{h}(f,\tau)\exp(i2\pi f\tau)df \quad ; \quad \hat{h}(f,t)=\int_{-\infty}^{\infty}h(\tau,t)\exp(-i2\pi f\tau)d\tau \tag{3.16}$$

Intuitively, we regard $\hat{h}(f, t)$ as a system function that is slowly changing with t, and view $h(\tau, t)$ as a channel filter whose impulse response (as a function of τ) is slowly changing with t. If we substitute the second part of (3.16) into (3.15)

$$y(t)=\int_{f=-\infty}^{\infty}\int_{\tau=-\infty}^{\infty}\hat{x}(f)h(\tau,t)\exp(i2\pi f(t-\tau))d\tau df \tag{3.17}$$

Interchanging the order of integration and recognizing the integration over f as the inverse Fourier transform of $\hat{x}(f)$, we get the *convolution equation for time-varying filters*,

$$y(t)=\int_{-\infty}^{\infty}x(t-\tau)h(\tau,t)d\tau \tag{3.18}$$

This expression is really quite nice. It says that the effect of mobile users, arbitrarily moving reflectors and absorbers, and all of the complexities of solving Maxwell's equations, finally reduce to an input/output relation between

transmit and receive antennas which is simply represented as the impulse response of a linear time varying channel filter. That is, $h(\tau, t)$ is the response at time t to an impulse at time $t - \tau$. If $h(\tau, t)$ is a constant function of t, then this is the conventional LTI impulse response.

For the particular form of $\hat{h}(f, t)$ in (3.13), the inverse transform $h(\tau, t)$ is

$$h(\tau, t) = \sum_j \alpha_j(t)\delta[\tau - \tau_j(t)] \tag{3.19}$$

Where δ is the Dirac impulse function. These idealized, non-physical, impulses arise here because of our earlier assumption that $\alpha_j(t)$ and $\tau_j(t)$ are not functions of frequency, which we justified by our interest only in inputs over a narrow band of frequencies around some carrier f_c. Physically, these delta functions arose from viewing reflectors solely through the ray tracing approximation and by ignoring the frequency attenuation of the antennas. We can see in (3.13) that if $\hat{x}(f)$ is limited to a given band, then it makes no difference what $\hat{h}(f, t)$ is outside of that band. In the same way, if the impulses in (3.19) were filtered to eliminate the out-of-band components, the response to a band-limited input would remain the same. To see this more clearly, we can substitute (3.19) into (3.18), getting

$$y(t) = \sum_j \alpha_j(t)x[t - \tau_j(t)] \tag{3.20}$$

Note that if $\delta\{\tau - \tau_j(t)\}(t)$ in (3.19) were replaced with a sinc function centered on $\tau_j(t)$ with a bandwidth wider than of $x(t)$, then the response in (3.20) would not be changed. Perhaps more to the point, if we used a more elaborate electromagnetic model, the response from the jth path would be a linear time varying filter in its own right, so that the overall response would again be a linear time-varying filter.

3.3 PARAMETERS OF MOBILE MULTIPATH CHANNELS

Recall from previous section that significant changes in α_j occur over periods of seconds or more. Significant changes in the phase for each path occur at intervals of 1/(2D), where D is the Doppler spread for the channel. Multipath fading occurs because different paths have different Doppler shifts [7]. Typical intervals for such changes are on the order of 10msec.Thus, the fastest

changes in the filter taps occur because of the phase changes, and these are significant over delay changes of 1/(2D). So there has to be some formulization for these.

3.3.1 Coherence time

The time coherence, Tc, of a narrowband wireless channel was defined (in an order of magnitude sense) as the interval over which $h(t,\tau)$ changes significantly. What we have found, then, is the important relation.

$$T_c = 1/4D \tag{3.21}$$

This is a somewhat imprecise relation, since different paths have different Doppler shifts, and the largest Doppler shifts may belong to paths that are too weak to make a difference.

3.3.2 Multipath Delay Spread

Another important general parameter of a wireless system is the multipath delay spread, T_d, defined as the difference in propagation time between the longest and shortest path, where we assume, in all of the above sums over different paths, that only the significant paths are included [7]. Thus,

$$T_d = \max_j(\tau_j) - \min_j(\tau_j) \tag{3.22}$$

3.3.3 Coherence Bandwidth

There is one additional gross mechanism called frequency coherence. Wireless channels change both in time and frequency. The time coherence shows us how quickly the channel changes in time, and similarly, the frequency coherence shows how quickly it changes in frequency. We first understood about channels changing in time, and correspondingly about the duration of fades, by studying the simple example of a direct path and a single reflected path. That same example shows us how channels change with frequency. For a particular path, α_j has linear phase in f [8]. For multiple paths, there is a differential phase, $2\pi f(2\pi f(\tau_j(t) - \tau_k(t))$. This differential phase causes frequency selective fading (equalization is required for frequency selective fading) in frequency.

This says that not only does $E_r(f; t)$ change significantly when t changes by $1/(2D)$, but also when f changes by $1/2T_d$. This argument extends to an arbitrary number of paths, so the coherence, bandwidth F_c is given by

$$F_c = 1/2T_d \tag{3.23}$$

This relationship, like (3.19) is intended as an order of magnitude relation, essentially pointing out that frequency coherence is reciprocal to multipath spread. When the bandwidth of the input is considerably less than F_c, the channel is usually referred to as at frequency flat fading, and, in essence, a single channel filter tap is sufficient to represent the channel (no equalization required). Note that flat fading is not a property of the channel alone, but of the relationship between W and F_c.

CHAPTER 4

MODULATION, MATCH FILTERING AND PULSE SHAPING

Many information bearing signals are transmitted by some type of carrier modulation. The channel over which the signal is transmitted is limited in bandwidth to an interval of frequencies centered about the carrier, as in double sideband modulation Signals and channel which satisfy the condition that their bandwidth is much smaller than the carrier frequency are termed narrow bandpass signals and channels. The modulation performed at the transmitting end of the communication system to generate the bandpass signal and the modulation performed at the receiving end to recover the digital information involves the frequency translations. With no loss in generality and mathematical convenience, it is desirable to reduce all the bandpass signals and channels to equivalent lowpass signals and channels [8]. This leads to the complex envelop representation of real bandpass signals.

4.1 COMPLEX ENVELOP REPRESENTATION

The real-valued signal $x(t)$ is a passband signal when its nonzero Fourier transform is near ω_c, as in Figure 4.1. Passband signals never have DC content, so $X(0) = 0$. A carrier modulated signal is any passband signal that can be written in the following form

$$x(t) = a(t)\cos(\omega_c t + \theta(t)) \tag{4.1}$$

where $a(t)$ is the time-varying amplitude or envelope of the modulated signal and $\theta(t)$ is the time-varying phase. ω_c is called the carrier frequency (in radians/sec). The carrier frequency ω_c is chosen sufficiently large compared with the amplitude and phase variations of $a(t)$ so that the power spectral

density does not have significant energy at $\omega = 0$. See Figure 4.1, wherein the spectrum of $X(\omega)$ is concentrated in the passband $\omega_{low} < |\omega| < \omega_{high}$. In digital communication, $x(t)$ is equivalently written in quadrature form using the trigonometric identity $\cos(u + v) = \cos(u)\cos(v) - \sin(u)\sin(v)$, leading to a quadrature decomposition. The quadrature decomposition of a carrier modulated signal is

$$x(t) = x_I(t)\cos(\omega_c t) - x_Q \sin(\omega_c t) \tag{4.2}$$

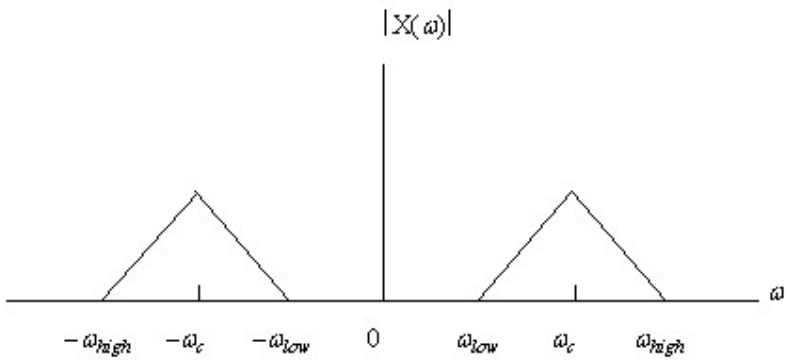

Figure 4.1: Fourier spectrum of Bandpass signal

where $x_I(t) = a(t)\cos(\theta(t))$ is the time-varying inphase component of the modulated signal, and $x_Q(t) = a(t)\sin(\theta(t))$ is the time-varying quadrature component. Relationships determining $a(t)$, $\theta(t)$ from $x_I(t) - x_Q(t)$ are

$$a(t) = \sqrt{x_I{}^2(t) + x_Q{}^2(t)} \tag{4.3}$$

$$\theta(t) = \tan^{-1}\left[\frac{x_Q(t)}{x_I(t)}\right] \tag{4.4}$$

In equation 4.4, the inverse tangent is taken with the polarities of the numerator and denominator independently known, so there is no quadrant ambiguity in computing $\theta(t)$. In passband processing and analysis, the objective is to eliminate explicit consideration of the carrier frequency ω_c and directly analyze systems using only the inphase and quadrature components. These inphase and quadrature components can be combined into a two-dimensional vector, or into an equivalent complex signal. By convention, a graph of a quadrature-modulated signal plots the inphase component along the real axis and the quadrature component along the imaginary axis as

shown in figure 4.2. The resultant complex vector $x_{bb}(t)$ is known as the complex baseband-equivalent signal. The complex baseband-equivalent signal for $x(t)$ in equation 4.1 is

$$x_{bb}(t) = x_I(t) + j\, x_Q(t) \tag{4.5}$$

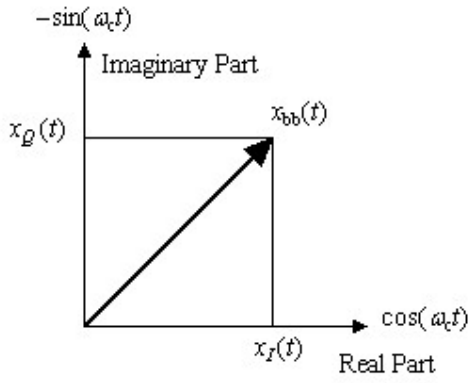

Figure 4.2: Decomposition of baseband-equivalent signal.

The baseband-equivalent signal expression no longer explicitly contains the carrier frequency ω_c Another complex representation that does explicitly contain ω_c is the analytic equivalent signal for $x(t)$. The analytic-equivalent signal for $x(t)$ in equation 4.1 is

$$x_A(t) = x_{bb}(t)\, e^{j\omega_c t} \tag{4.6}$$

The original real-valued passband signal $x(t)$ is the real part of the analytic equivalent signal

$$x(t) = \Re\left[x_A(t)\right] \tag{4.7}$$

The Hilbert transform of $x(t)$, denoted by $\breve{x}(t)$, is the imaginary part of the analytic signal as

$$\breve{x}(t) = \Im\left[x_A(t)\right] \tag{4.8}$$

Finally, the inphase component $x_I(t)$ and the quadrature component $x_Q(t)$ can be expressed using the signal $x(t)$ and its Hilbert transform $\breve{x}(t)$ as (using $x_{bb}(t) = x_I(t) + j\, x_Q(t) = x_A(t)\, e^{j\omega_c t}$:

$$x_I(t) = x(t)\cos(\omega_c t) + \breve{x}(t)\sin(\omega_c t) \tag{4.9}$$

$$x_Q(t) = x(t)\cos(\omega_c t) - \breve{x}(t)\sin(\omega_c t) \tag{4.10}$$

4.1.2 Spectrum of Analytic and Baseband-Equivalent Signals

Using equations 4.7 and 4.8 the analytic signal is represented as shown in figure 4.3

$$x_A(t) = x_I(t) + j\, x_Q(t) \tag{4.11}$$

Taking the Fourier Transform of both sides of equation 4.11 yields

$$\begin{aligned} X_A(\omega) &= \left[1 + \operatorname{sgn}(\omega)\right] X(\omega) \\ &= \begin{cases} 2X(\omega) & \omega > 0 \\ X(0) = 0 & \omega = 0 \\ 0 & \omega < 0 \end{cases} \end{aligned} \tag{4.12}$$

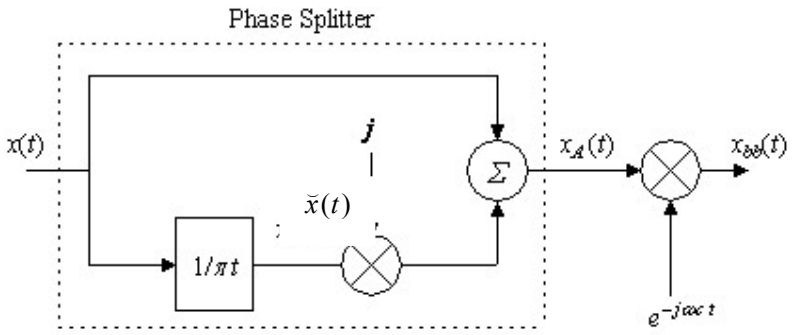

Figure 4.3: Complex baseband signal recovery from real passband signal

The analytic equivalent signal, $x_A(t)$, contains only the positive frequencies of $x(t)$ and is identically zero for negative frequencies. The Fourier transform $X(\omega)$ of the real signal $x(t)$ has two symmetry properties. The real part $\Re[X(\omega)]$ is even in ω, while the imaginary part $\Im[X(\omega)]$ is odd in ω. Knowledge of only the non-negative frequencies of $X(\omega)$, such as are supplied by the analytic signal, is sufficient for reconstruction of $X(\omega)$. Thus, one confirms that the analytic signal $x_A(t)$ is truly equivalent to the original signal $x(t)$. Using equation 4.6, the Fourier transform of the baseband equivalent signal is simply the Fourier transform of the analytic signal translated in frequency ω. Thus

$$X_A(w) = X_{bb}(\omega - \omega_c) \tag{4.13}$$

$$X_{bb}(w) = X_A(\omega + \omega_c) \tag{4.14}$$

Use of equations 4.6 and 4.7 allows reconstruction of the signal $x(t)$ from the baseband equivalent signal $x_{bb}(t)$ and the carrier frequency ω_c. The baseband equivalent signal, in general, may be complex-valued, and thus as

shown in figure 4.4 the spectrum of $x_{bb}(t)$ may be asymmetric about the origin $\omega = 0$

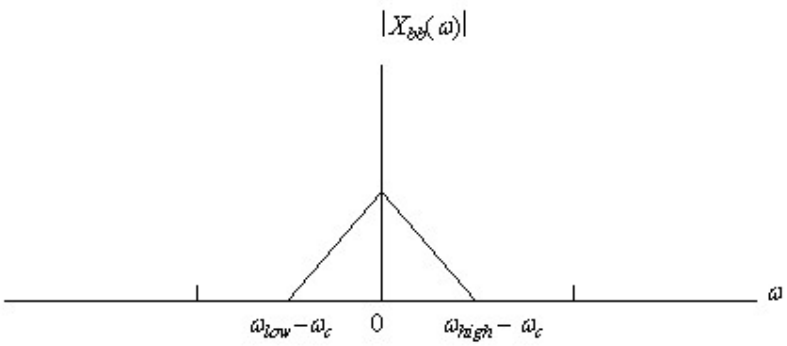

Figure 4.4: Baseband signal spectrum.

4.1.2 Generation of the baseband equivalent

To generate the baseband equivalent of a signal, the structure in figure 4.3 is used, where the second complex multiply simply is 4 real multiplies using Euler's formula $e^{j\omega_c t} = \cos(\omega_c t) + j\sin(\omega_c t)$. The first multiply by j alone is, of course, symbolic and simply means that the receiver processing views the signal on that path as the imaginary part in complex arithmetic.

4.2 DIGITAL BANDPASS MODULATION

Digital modulation is the process by which digital symbols are transformed into waveforms that are compatible with the characteristics of the channel and demodulation is the reverse process of again recovering the original message. In the case of baseband modulation, these waveforms usually take the form of shaped pulses. But in the case of *bandpass modulation* the shaped pulses modulate a sinusoid called a *carrier wave,* or simply a carrier; for radio transmission the carrier is converted to an electromagnetic (EM) field for propagation to the desired destination [14].

The modulating process transforms the low frequency baseband signal to a bandpass signal around a carrier frequency as sketched in figure 4.5. The bandpass signal is the one actually transmitted to the receiver where the demodulator reconstructs the low-frequency baseband message.

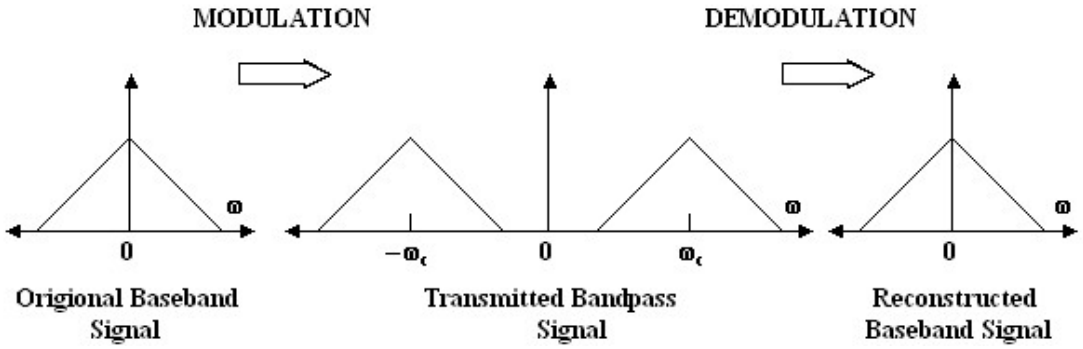

Figure 4.5: Power spectra for signals in the modulation and demodulation processes.

Bandpass modulation (either analog or digital) is the process by which an information signal is converted to a sinusoidal waveform; for digital modulation, such a sinusoid of duration T is referred to as a digital symbol. The sinusoid has just three features that can be used to distinguish it from other sinusoids: amplitude, frequency, and phase. Thus bandpass modulation can be defined as the process whereby the amplitude, frequency, or phase of an RF carrier, or a combination of them, is varied in accordance with the information to be transmitted. The general form of the carrier wave is

$$s(t) = A(t)\cos\ \theta(t) \tag{4.15}$$

where $A(t)$ is the time-varying amplitude and $\theta(t)$ is the time-varying angle. It is convenient to write

$$\theta(t) = \omega_O t + \phi(t) \tag{4.16}$$

so that

$$s(t(= A(t)\cos\left[\omega_o t + \phi(t)\right] \tag{4.17}$$

Where W_O is the *radian frequency* of the carrier and $\phi(t)$ is the *phase.* The terms f and ω will each be used to denote frequency. When f is used, frequency in hertz is intended; when ω is used, frequency in radians per second is intended. The two frequency parameters are related by $\omega = 2\pi f$.

4.2.1 Phase Shift Keying

Under phase-shift keying (PSK), the information bits determine the phase of a carrier, which takes values from a discrete set in accordance with the information bits. The general form of PSK signals is given by

$$s_i(t) = \sqrt{\frac{2E}{T}} \cos(\omega_c t + \theta_i t) \qquad 0 \le t \le T, \qquad i = 1,2,\ldots,M \tag{4.18}$$

where the phase term, θ_i , will have M discrete values, typically given by

$$\theta_i = \frac{2\pi i}{M} \qquad i = 1,2,\ldots,M \tag{4.19}$$

$$E = \int_0^T s_0{}^2(t)\,dt \tag{4.20}$$

is the signal energy (the same for all signals). We will assume that the signal is bipolar rectangular pulse of duration T until the discussion about pulse shaping. Equation 4.18 can be re-written in a slightly different form as

$$\begin{aligned} s_i(t) &= \sqrt{E}\left[\cos(\theta_i)\sqrt{\frac{2}{T}} \cos(2\pi\omega_c t) - \sin(\theta_i)\sqrt{\frac{2}{T}} \sin(2\pi\omega_c t) \right] \\ &= \sqrt{E}\left[\cos(\theta_i)\,\phi_1(t) - \sin(\theta_i)\,\phi_2(t)\right] \end{aligned} \tag{4.21}$$

where $\phi_1(t)$ and $\phi_2(t)$ are easily seen to be orthonormal. Thus, PSK signals are points in a two-dimensional space spanned by $\phi_1(t)$ and $\phi_2(t)$ [14].

4.2.2 QPSK Transmitter

For the binary PSK $M = 2$ this means that modulating data signal shifts the phase of the waveform $s_i(t)$ to one of the two states either zero or π. Similarly for quadriphase or quadrature shift keying $M = 4$ and the waveform $s_i(t)$ shifts the phase to one of 4 pahses separated by π/4. The constellation diagram for QPSK signal using relation in equation 4.21 is shown in figure 4.6.

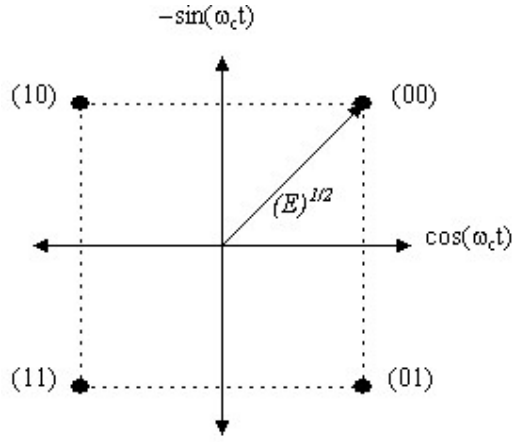

Figure 4.6: QPSK Constellation diagram

The illustrated constellation mapping in figure 6, known as Gray coding, has the property that adjacent signals are assigned binary sequences that differ in only one bit. This is desirable in practice, because, when a detection error is made, it is more likely to be to a signal adjacent to the transmitted signal. Then Gray coding results in a single bit error for the most likely signal errors. QPSK is also a real bandpass signal and using the complex baseband envelop representation of real bandpass signals the baseband representation of QPSK is

$$s_{bb}(t) = s_I(t) + j\, s_Q(t) = s_A(t)e^{j\omega_c t} \tag{4.22}$$

where

$$s_I(t) = s(t)\cos(\omega_c t) + \breve{s}(t)\sin(\omega_c t)$$

$$s_Q(t) = s(t)\cos(\omega_c t) + \breve{s}(t)\sin(\omega_c t)$$

are the inphase and quadrature phase components of $s_i(t)$. The relation between $s_i(t)$ and $s_{bb}(t)$ is given as

$$s_i(t) = \Re\left[s_{bb}(t)e^{j\omega_c t}\right] = \Re\left[s_A(t)\right] \tag{4.23}$$

Using the above relation the QPSK constellation using the complex baseband representation is redrawn in figure 4.7.

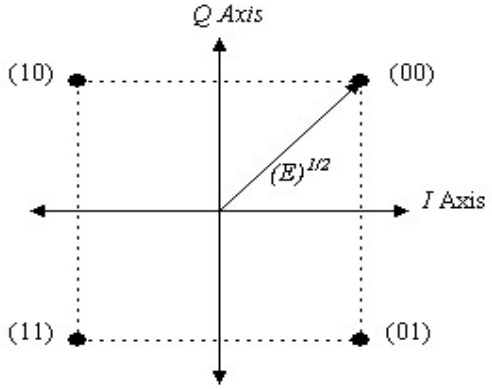

Figure 4.7: QPSK Constellation diagram for complex baseband representation.

Now using the complex baseband representation the QPSK transmitter is drawn in figure 4.8. QPSK lookup table in figure 4.8 is simply the assignment of one phase to each of four symbols. Transmit filter is a filter which shapes the bitstream to a waveform. After constellation mapping we are having only phase as shown in table below. It is the transmit filter which convert them in a waveform We will discuss it later.

Table 4.1: QPSK lookup table

Symbol	Phase
00	$e^{j\pi/4}$
01	$e^{-j\pi/4}$
10	$e^{j3\pi/4}$
11	$e^{j-3\pi/4}$

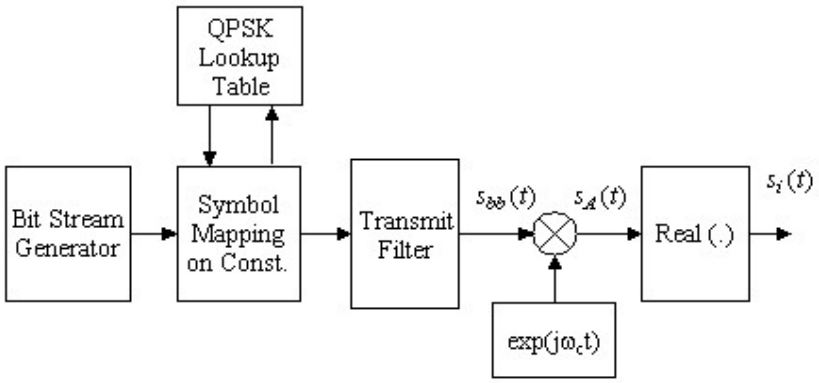

Figure 4.8: QPSK Transmitter

To recover the analytic signal from the received real QPSK signal same approach is used which is described in figure 4.3.

4.2.3 QPSK Detection

There are two approaches for detection of any modulated signal. One is called coherent detection and other is called non-coherent detection. When the receiver exploits knowledge of the carrier's phase to detect the signals, the process is called *coherent detection;* when the receiver does not utilize such phase reference information, the process is called *no coherent detection.* In ideal coherent detection, there is available at the receiver a prototype of each possible arriving signal. These prototype waveforms attempt to duplicate the transmitted signal set in every respect, even RF phase. The receiver is then said to be phase locked with the incoming signal. While for non-coherent detection the there is no need for the receiver to be phase locked with the transmitter because the phase information is provided to the receiver by differentially encoding the symbols at the receiver. Thus the receiver design is simplified.

We are not using the differential encoding, so we will only discuss the coherent detection. In coherent detection the receiver has the decision regions as shown in figure 4.9.

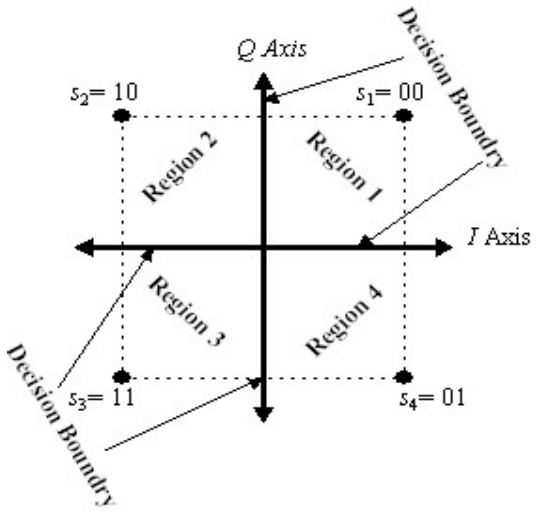

Figure 4.9: Decision regions for QPSK signal

If the symbol is in region 1 then the receiver makes the decision that it is 00 and so on. The receiver structure is shown in figure 4.10. After recovering the complex baseband signal from the real received signal we have to determine only the phase and according to that phase we have to select that particular region in which phase is lying. After selecting the region the only job left is to recover the two bits which conveyed by the phase using the same lookup table as shown before. We will discuss the receive filter later.

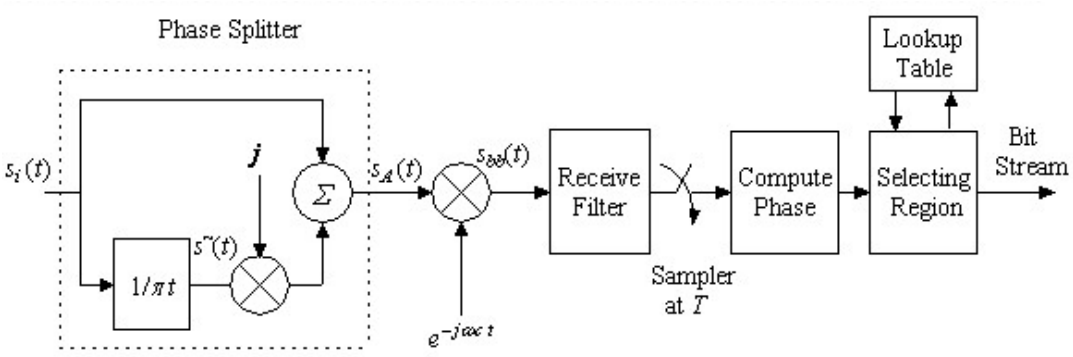

Figure 4.10: QPSK Receiver structure

So far we have explained complex envelop representation of bandpass signals and QPSK modulation/demodulation. From now on we will not use the

receiver structure explained above but only the complex baseband part will be used. The frequency translation part will be removed because it the same for all. The baseband receiver structure is given in figure 4.11.

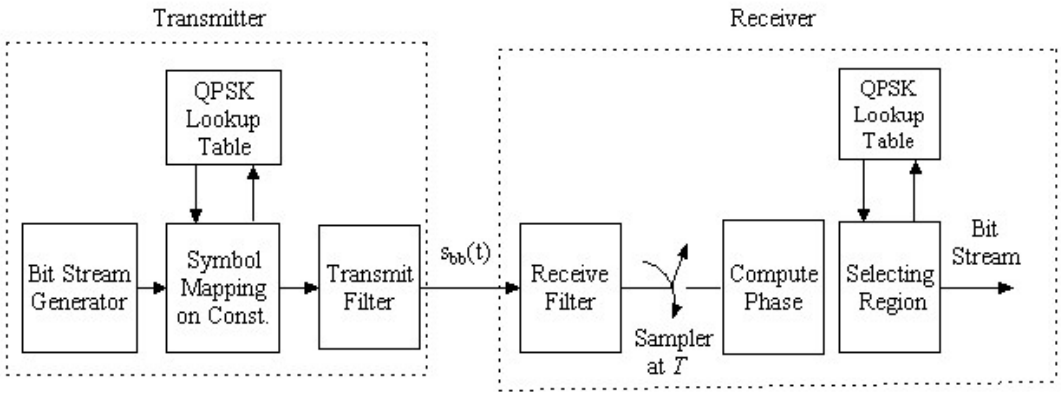

Figure 4.11: Baseband model for QPSK communication system

4.3 DETECTION IN ADDITIVE WHITE GAUSSIAN NOISE (AWGN)

4.3.1 Additive White Gaussian Noise

So far we have assuming the ideal channel because we are receiving the same signal which we are transmitting. But this will never happen in practice. There are many sources of noise which corrupt the transmitted signal like galaxy and atmospheric noise, switching transients, interfering signals from other sources and many many more sources. With proper precautions much of the noise and interference entering the receiver can be reduced or even eliminated. However there is one noise source that cannot be eliminated and that is the noise caused by the thermal motion of electrons in any conducting media. This motion produces thermal noise in amplifiers and circuits and corrupts the signal.

The noise can be thought as a random process. Any random process can be modelled statistically using normal or Gaussian. An important case of a random signal is the case where the autocorrelation function is a *dirac delta* function which has zero value everywhere except when $\tau = 0$. In other words, the case where

$$R_x(\tau) = \begin{cases} N_0 & for\ \tau = 0 \\ 0 & elsewhere \end{cases} \tag{4.24}$$

where $R_x(\tau)$ is the auto-correlation function of a random variable $x(t)$ and N_0 is any constant. The auto-correlation at $\tau = 0$ is also called the power of the signal. The Fourier transform of auto-correlation function is called as the power spectral density and power spectral density for noise is

$$S_x(\omega) = \int_{-\infty}^{\infty} R_x(\tau) e^{-j\omega\tau} d\tau = \frac{N_0}{2} \tag{4.25}$$

In this special case where the autocorrelation is a "spike" the Fourier transform results in a *constant* frequency spectrum as shown in figure 4.12. This is in fact a description of *white noise*, which be thought of both as having power at all frequencies in the spectrum, and being completely uncorrelated with itself at any time except the present $(\tau = 0)$. This latter interpretation is what leads white noise signals to be called *independent*. Any sample of the signal at one time is completely independent (uncorrelated) from a sample at any other time. While impossible to achieve or see in practice (no system can exhibit infinite energy throughout an infinite spectrum), white noise is an important building block for design and analysis. Often random signals can be modeled as filtered or *shaped* white noise. Literally this means that one could filter the output of a (hypothetical) white noise source to achieve a non-white or *colored* noise source that is both band-limited in the frequency domain, and more correlated in the time domain.

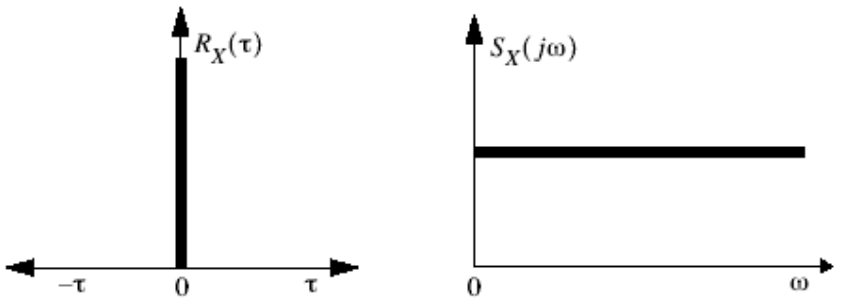

Figure 4.12: White noise shown in both the time (left) and frequency domain (right).

From above it is clear why thermal noise is called Additive white Gaussian noise (AWGN). Additive because it adds in the signal not multiplies. White because it has the same power for all the frequencies. Gaussian because it can be modelled using Gaussian or normal distribution and power for any normally distributed random variable is ${\sigma_0}^2$, where ${\sigma_0}^2$ is the variance of the

random variable.

4.3.2 SNR Maximization with a Matched Filter

SNR is a good measure for a system's performance, describing the ratio of signal power (message) to unwanted noise power. The SNR at the output of a filter is defined as the ratio of the modulated signal's energy to the mean-square value of the noise. The SNR can be defined for both continuous- and discrete-time processes; the discrete SNR is SNR of the samples of the received and filtered waveform. A matched filter is a linear filter designed to provide the maximum signal-to-noise power ratio at its output for a given transmitted symbol waveform. It is called match filter because it impulse response exactly matches with the impulse response of the transmitted signal1 [14]. It will be proved now.

Consider that a known signal *s(t)* plus AWGN *n(t)* is the input to a linear, time-invariant (receiving) filter followed by a sampler, as shown in Figure 4.13. Actually the receive filter is replaced with match filter**.** At time $t = T$, the sampler output *z(T)* consists of a signal component a_i and a noise component n_0.

$$z(T) = a_i + n_0 \tag{4.26}$$

The variance of the output noise (average noise power) is denoted by $\sigma_{0.}^2$, so that the ratio of the instantaneous signal power to average noise power. $(S/N)_T$. at time t = *T,* out of the sampler in step 1, is

$$\left(\frac{S}{N}\right)_T = a_i^2 \Big/ \sigma_O^2 \tag{4.27}$$

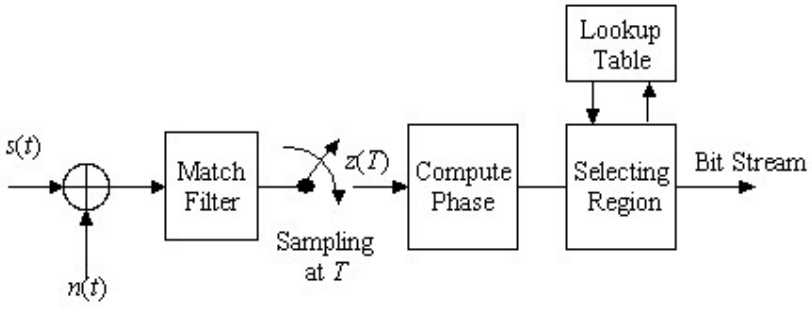

Figure 4.13: QPSK receiver with sampler at symbol rate *T* and match filter

We wish to find the filter transfer function $H_o(f)$ that *maximizes e*quation 4.27. We can express the signal $a_i(t)$ at the filter output in terms of the filter transfer function $H(f)$ (before optimization) and the Fourier transform of the input signal,

$$a_i(t) = \int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df \tag{4.28}$$

where $S(f)$ is the Fourier transform of the input signal, S(t). If the two-sided power spectra} density of the input noise is $No/2$ watts/hertz, then, we can express the output noise power as

$$\sigma_o^2 = \frac{N_o}{2}\int_{-\infty}^{\infty} |H(f)|^2\,df \tag{4.29}$$

We then combine equations 4.27 to 4.29 to express $(S/N)_T$. as follows

$$\left(\frac{S}{N}\right)_T = \frac{\left|\int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df\right|^2}{\frac{N_O}{2}\int_{-\infty}^{\infty}|H(f)|^2\,df} \tag{4.30}$$

We next find that value of $H(f) = H_o(f)$ for which the maximum $(S/N)_T$ is achieved, by using *Schwarz's inequality.* One form of the inequality can be stated as

$$\left|\int_{-\infty}^{\infty} f_i(x)f_2(x)dx\right|^2 \le \int_{-\infty}^{\infty}|f_1(x)|^2\,dx\int_{-\infty}^{\infty}|f_2(x)|^2\,dx \tag{4.31}$$

The equality holds if $f_1(x) = kf^*_2(x)$ where k is an arbitrary constant and * indicates complex conjugate. If we identify $H(f)$ with $f_1(x)$ and $S(f)\ e^{Tf\square j2}$ with $f_2(x)$, then

$$\left|\int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df\right|^2 \le \int_{-\infty}^{\infty}|H(f)|^2\,df\int_{-\infty}^{\infty}|S(f)|^2\,df \tag{4.32}$$

Substituting into Equation 4.30 yields

$$\left(\frac{S}{N}\right)_T \le \frac{2}{N_O}\int_{-\infty}^{\infty}|S(f)|_2\,df \tag{4.33}$$

Where the energy E of the input signal $S(t)$ is

$$E = \int_{-\infty}^{\infty}|S(f)|^2\,df \tag{4.34}$$

Thus, the maximum output $(S/N)_T$ depends on the input *signal energy* and the power spectral density of the noise, *not on the particular shape* of the

waveform that is used. The equality in Equation 4.33 holds only if the optimum filter transfer function $H_O(f)$ is employed, such that

$$H(f) = H_O(f) = kS^*(f)e^{-j2\pi Tf} \tag{4.35}$$

$$h(t) = \Im^{-1}\left\{kS^*(f)e^{-j2\pi Tf}\right\} \tag{4.36}$$

Since S(t) is a real-valued signal, we can write,

$$h(t) = \begin{cases} ks(T-t) & 0 \le t \le T \\ 0 & elsewhere \end{cases} \tag{4.37}$$

Thus, the impulse response of a filter that produces the maximum output signal-to-noise ratio is the mirror image of the message signal s(t), delayed by the symbol time duration T. Note that the delay of T seconds makes Equation 4.37 causal; that is, the delay of T seconds makes h(t) a function of positive time in the interval $0 < t < T$ as shown in figure 4.14. Without the delay of T seconds, the response $s(-t)$ is unrealizable because it describes a response as a function of negative time.

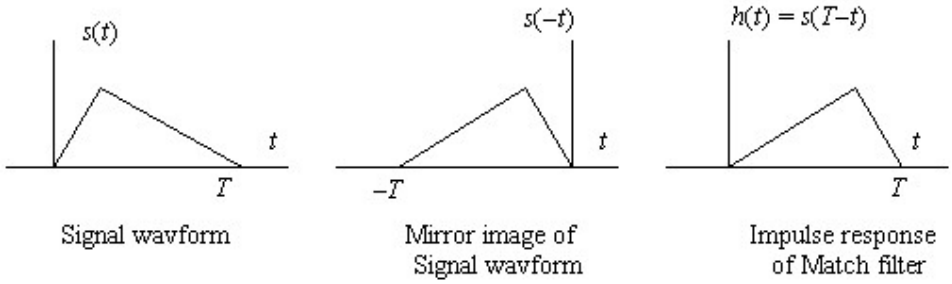

Figure 4.14: Impulse responses of received signal and match filter

The above mathematical discussion proves that if the impulse responses of the received signal and the match filter are mirror images of each other then at $t=T$ the SNR is maximized. Actually the convolution with itself is a process of integration. By match filtering we are actually integrating the received signal. AWGN is a zero mean random variable. By averaging we are trying to force it to zero. It can be verified that as T approaches ∞ the noise averaged to zero.

The QPSK receiver with match filter can be redrawn in figure 4.15.

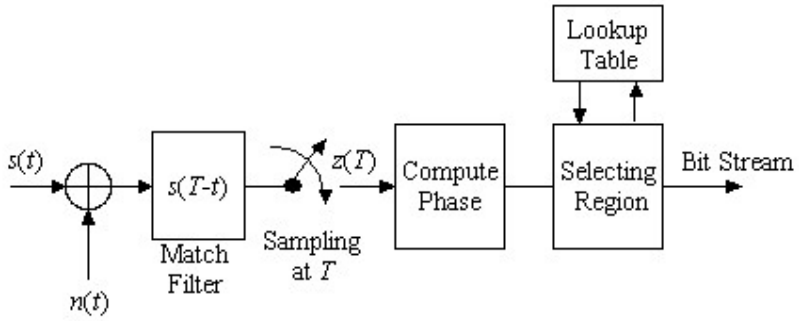

Figure 4.15: QPSK receiver with match filter

4.4 INTERSYMBOL INTERFERENCE AND PULSE SHAPING

The spreading and smearing of symbols such that the energy from one symbol effects the next ones in such a way that the received signal has a higher probability of being interpreted incorrectly is called inter symbol interference (ISI).

Let's assume that the transmit filter has a impulse response of a rectangular pulse as shown. We know that the frequency response of rectangular pulse is a sinc function which is from $[-\infty, \infty]$. This means that it has infinite bandwidth which is not the requirement and also the rectangular pulse is not possible to design practically.

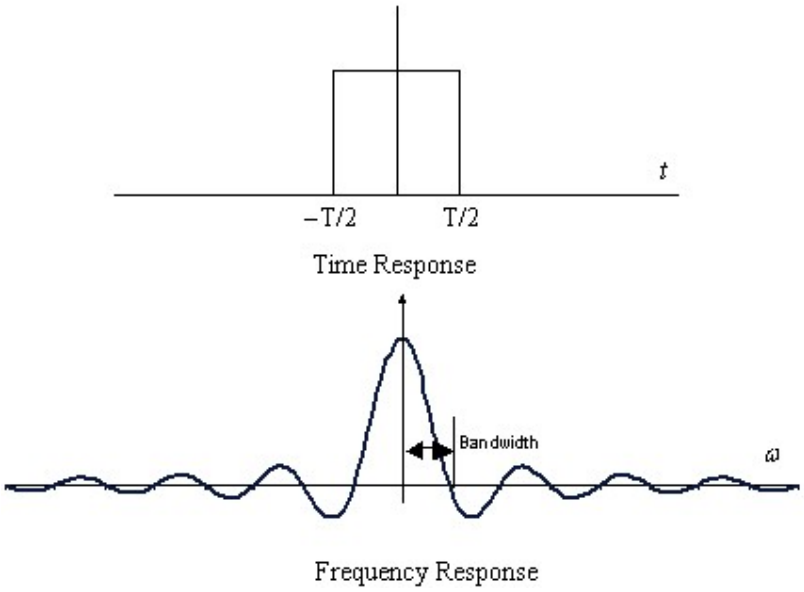

Figure 4.16: Time and frequency response of rectangular pulse

Another solution is that we used the sinc in time domain because it has a gate function in frequency domain which has very pleasant from bandwidth requirement. But the problem is that the impulse response of one pulse has infinite length. But the sinc pulse is passing through zero after every multiple of T as shown above. Now if we transmit the successive pulses such that a pulse has its max peak value when the others are passing through zero. In this case we may have ISI at the other time but this will ensure that there is no ISI at the multiples of symbol interval as shown in figure 4.16.

Sinc pulse has problem that it is also impractical to design and also it has infinite impulse response. A single pulse is affecting all the pulses before or after it. Slight misadjustment in time will result in effecting all the pulses.

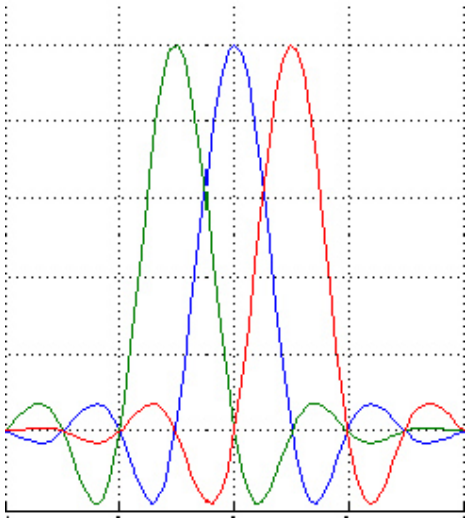

Figure 4.17: pulse shaping using sinc.

Nyquist offered ways to build (realizable) shapes that had the same good qualities as the sinc pulse and less of the disadvantages. One class of pulses he proposed are called the raised cosine pulses. They are really a modification of the sinc pulse. Where the sinc pulse has a bandwidth of W, which is given as

$$W = 1/2T \tag{4.38}$$

The raised cosine pulses have an adjustable bandwidth which can be varied from W to $2W$. We want to get as close to W, which is called the Nyquist bandwidth, as possible with a reasonable amount of power. The factor α related the achieved bandwidth to the ideal bandwidth W as

$$\alpha = 1 - \frac{W}{W_0} \tag{4.39}$$

where W is Nyquist bandwidth, and W_0 is the utilized bandwidth.

The factor α is called the roll-of factor. It indicates how much bandwidth is being used over the ideal bandwidth. Smaller this factor, the more efficient will be the scheme. The percentage over the minimum required W is called the excess bandwidth. It is 100% for roll-off of 1.0 and 50% for roll-off of 50%. The alternate way to express the utilized bandwidth is.

$$W_0 = (1+\alpha)R_s \tag{4.40}$$

Typical roll-off values used for wireless communications range from 2 to 4. Obviously we want to use as small a roll-off as possible, since this gives the smallest bandwidth. Here is how the class of raised cosine pulse is defined in time domain.

$$h(t) = \frac{\sin(2\pi t / T_s)}{2\pi t} \frac{\cos(2\pi\alpha t)}{1-(2\alpha t/\pi)^2} \tag{4.40}$$

The first part is the sinc pulse. The second part is a cosine correction applied to the sinc pulse to make it behave better. The sinc pulse insures that the function transitions at integer multiples of symbol rate which makes it easy to extract timing information of the signal. The cosine part works to reduce the excursion in between the sampling instants. The bandwidth is now adjustable. It can be any where from $1/2\ R_s$ to R_s. It is greater than the Nyquist bandwidth by a factor $(1+\alpha)$. For $\alpha = 0$, the above equation reduces to the sinc pulse, and for $\alpha = 1$, the equation becomes that of a pure square pulse.

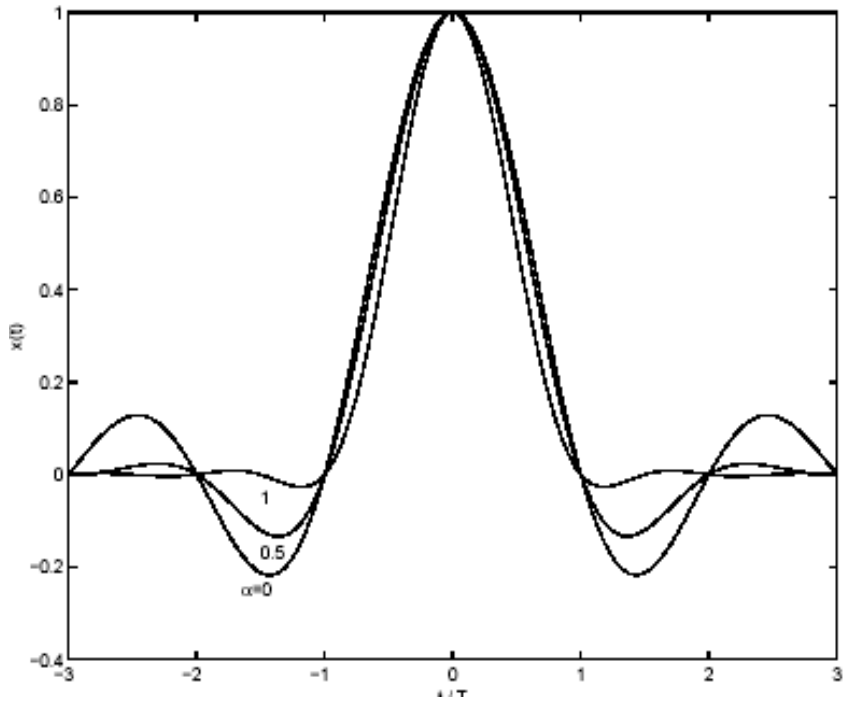

Figure 4.18: Impulse responses of raised cosine filter with $\alpha = 0,0.5,1$

In frequency domain, the relationship is given by

$$H(f) = \begin{cases} 1 & for\ |f| \le \dfrac{(1-\alpha)}{2T_s} \\ \left\{\cos^2 \dfrac{\pi T_s}{2\alpha}\left(|f| - \dfrac{(1-\alpha)}{2T_s}\right)\right\} & for\ \dfrac{(1-\alpha)}{2T_s} \le |f| \le \dfrac{(1+\alpha)}{2T_s} \\ 0 & for\ |f| > \dfrac{(1+\alpha)}{2T_s} \end{cases} \tag{2.41}$$

Why do they call it raised cosine? Because the above response has a cosine function in the frequency domain, although other many other trigonometric representations of this equation that do not have the cosine-squared term, so it is not always clear why these are called raised cosine.

The frequency response looks somewhat like a square pulse as we would expect. A range of bandwidths are possible depending on the chosen α. The bandwidth can be anywhere from $1/2\ R_s$ (this term same as W, the Nyquist bandwidth) for the sinc pulse to R_s for the square pulse. The bandwidth utilized is greater than the Nyquist bandwidth by a factor $(1 + \alpha)$. For $\alpha = 1$ the above

equation reduces to the sinc pulse, and for $\alpha = 1$ the equation becomes that of a pure square pulse.

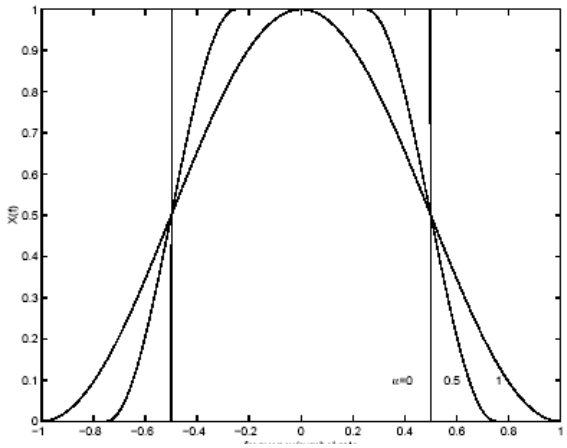

Figure 4.19: Frequency responses of raised cosine filter with $\alpha = 0,0.5,1$

To implement the raised cosine response, we split the filtering in two parts to create a matched set. When we split the raised cosine filtering in two parts, each part is called the root-raised cosine. In frequency domain, we take the square root of the frequency response hence the name root-raised cosine.

Yes, the whole raised cosine can be applied at once at the transmitter but in practice it has been found that concatenating two filters each with a root raised cosine response (called split-filtering) works better.

The root raised cosine shaping of pulses is also called baseband filtering. The frequency response of the root raised cosine is given by

$$H\ (f) = \begin{cases} 1 & for\ |f| \le \dfrac{(1-\alpha)}{2T_s} \\ \left\{\cos\dfrac{\pi T_s}{2\alpha}\left(|f| - \dfrac{(1-\alpha)}{2T_s}\right)\right\} & for\ \dfrac{(1-\alpha)}{2T_s} \le |f| \le \dfrac{(1+\alpha)}{2T_s} \\ 0 & for\ |f| > \dfrac{(1+\alpha)}{2T_s} \end{cases} \tag{4.42}$$

Compare the impulse response of the root raised filter to that of the raised cosine. We do not see much of a difference except that there is a little bit more excursion in the root-raised cosine response. The time domain function is of course NOT the square root. The root part applies to frequency domain

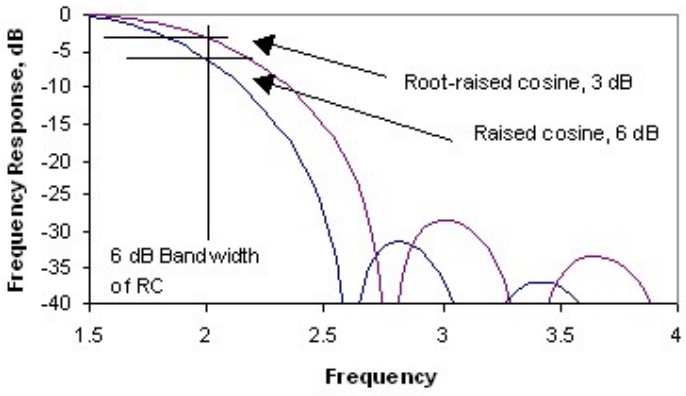

Figure 4.20: Frequency responses of raised cosine and rootraised cosine filters

By splitting the raised cosine into two rootraised cosine filter we are achieving two things. First is that by doing so we are forcing the ISI to zero at the receiver. There is ISI when we are transmitting but there is no ISI at the receiver. Second is match filtering because the impulse response of transmit and receiver filters are exactly matched. This will give us the highest SNR point at the symbol interval. Now the final baseband communication system is given below. Two wire connections are for complex data.

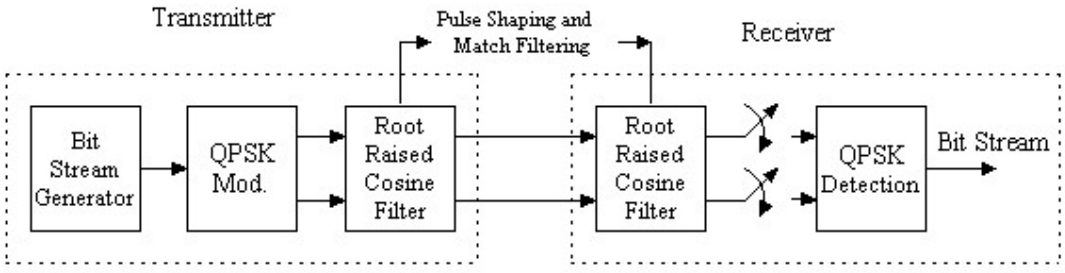

Figure 4.21: Baseband model of QPSK communication system

4.5 DISCRETE TIME MODEL OF COMMUNICATION SYSTEM

So far we have explained discrete partial analog communication system. In figure 4.20 the only analog portion is raised cosine filter. If we realize it in discrete time the complete system will become digital. We have already explained the upsampling using interpolation in a previous chapter. We know that the raised cosine filter requires the excess bandwidth. So in order to filter the QPSK symbol we have to upsample by at least factor of 2 because the maximum excess bandwidth is equal to the Nyquist bandwidth. The

upsampling is done by the insertion of $M-1$ zeros in between the QPSK symbol stream and then interpolation is done by the digital raised cosine filter. If we are sampling by a factor of M then we will say that we have M samples per symbol. At the receiver we have to downsample by a factor of M in order to recover the QPSK symbol stream. The complete discrete time model is given below

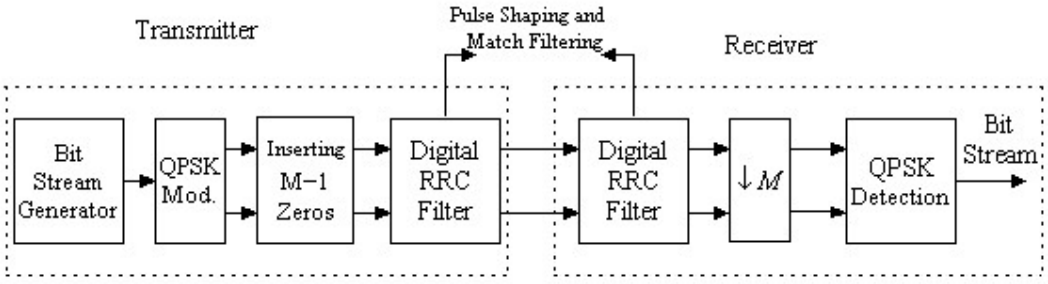

Figure 4.22: Discrete time baseband model of QPSK communication system

4.6 SIMULATION RESULTS

A simulation is run for QPSK modulation and pulse shaping using raised cosine pulse shaping. The constellation diagram for QPSK symbols at transmitter is

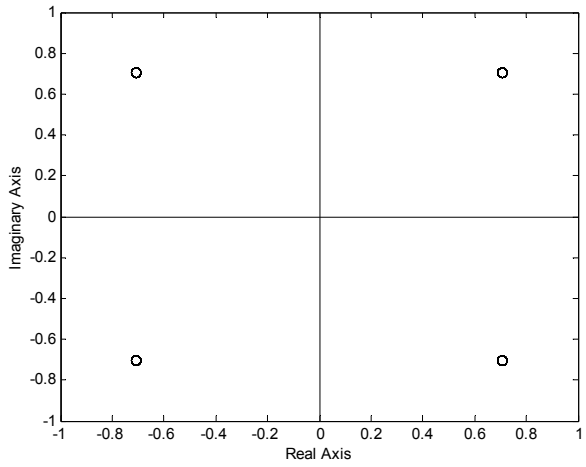

Figure 4.23: QPSK Constellation diagram at transmitter

QPSK constellation diagram after the coherent detection at the receiver is

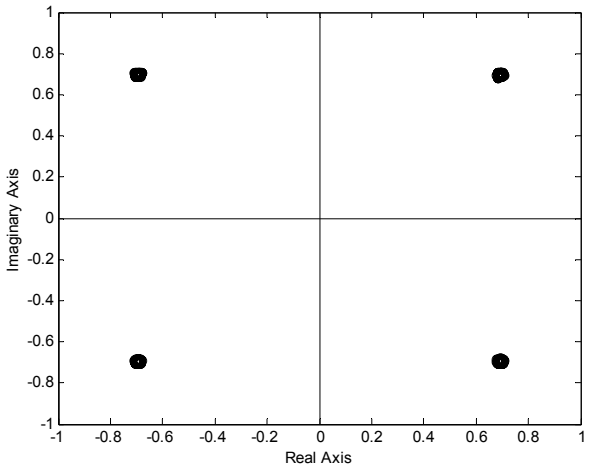

Figure 4.24: QPSK Constellation diagram at receiver

Raised cosine filter is splitted into root raised cosine filters at transmitter and receiver for match filtering and pulse shaping. 10 samples per symbol are chosen for convenience in plotting and the length of filters is 101. The impulse response of the raised cosine and root raised cosine filters are shown in figures.

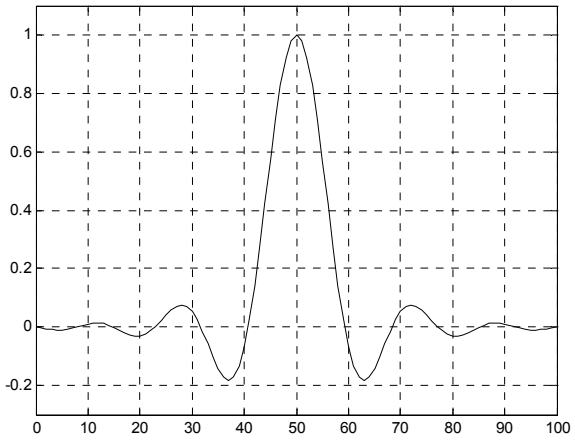

Figure 4.25: Impulse response of root raised cosine pulse

From the above figure you can see that impulse response is not passing through zero crossings at symbol intervals (multiples of 10)

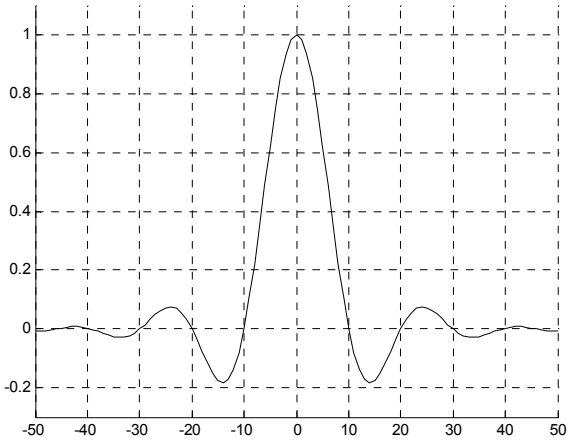

Figure 4.26: Impulse response of root raised cosine pulse

In the above the impulse response is passing through zero crossings at symbol intervals and this guarantees the zero ISI at symbol intervals.

In the following two figures the eye-diagram at transmitter and receiver is plotted and from these figures it is clear that there is ISI at the transmitter after filtering using root raised cosine filter but the ISI is removed at the receiver after root raised cosine filtering because both of them collectively make a raised cosine filter.

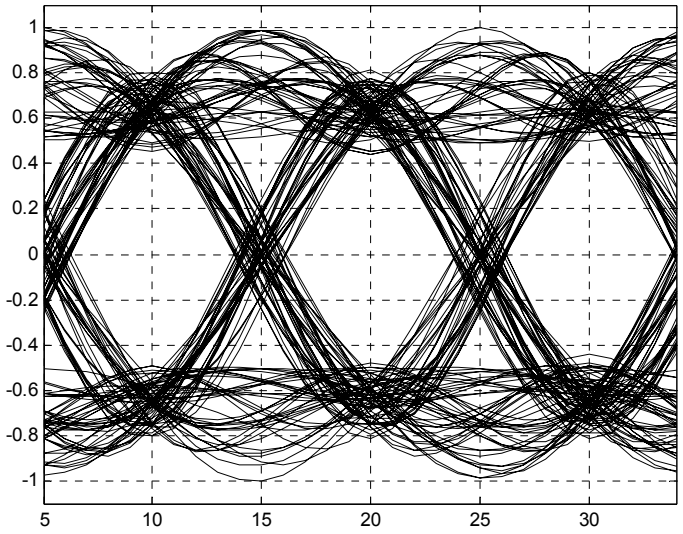

Figure 4.27: Eye diagram at transmitter (there is ISI)

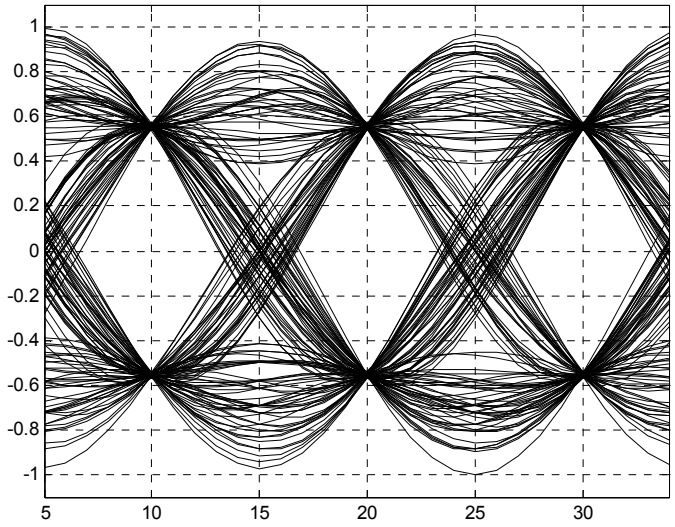

Figure 4.28: Eye diagram at receive (no ISI)

CHAPTER 5

ADAPTIVE FILTERS

5.1 INTRODUCTION

Filters are devices that are used in a variety of applications, often with very different aims. For example, a filter may be used, to reduce the effect of additive noise or interference contained in a given signal so that the useful signal component can be discerned more effectively in the filter output. Much of the available theory deals with linear filters, where the filter output is a (possibly time-varying) linear function of the filter input. There are basically two distinct theoretical approaches to the design of such filters.

- The 'classical' approach is aimed at designing frequency selective filters such as lowpass/bandpass/notch filters etc. For a noise reduction application, for example, it is based on knowledge of the gross spectral contents of both the useful signal and the noise components. It is applicable mainly when the signal and noise occupy clearly different frequency bands.
- The 'Optimal filter design', on the other hand, is based on optimization theory, where the filter is designed to be "best" (in some sense). The fundamental difference is that in adaptive filters the desired signal and the unwanted noisy signal occupies the same frequency band and therefore linear filters which are frequency selective filters are no more useful. If the signal and noise are viewed as stochastic processes, based on their statistical parameters, an optimal filter is designed that, for example, minimizes the effects of the noise at the filter output according to some statistical criterion. In the context of 'adaptive filtering' where we do not assume knowledge of the stochastic parameters but which is based on a very similar idea.

5.2 LINEAR OPTIMUM FILTERS

The theory of optimal filter design dates back to the work of Wiener in 1942 and Kolmogorov in 1939. The resulting solution is often referred to as the Wiener filter.

We may classify filters as linear or nonlinear. A filter is said to be *linear* if the filtered, smoothed, or predicted quantity at the out put of the filter is a *linear function of the observations applied to the filter input.* Otherwise, filter is *nonlinear.*

In the statistical approach to the solution of the linear filtering problem, we assume the availability of certain statistical parameters (i.e., mean & correlation functions) of the useful signal & unwanted additive noise, & the requirement is to design a linear filter with the noisy data as input so as to minimize the effect of noise at the filter out put according to statistical criterion [10]. A useful approach to this filter optimization problem is to minimize the mean-square value of the error *signal* defined as the difference between some desired response & the actual filter output. For stationary inputs, the resulting solution commonly known as the *wiener filter,* which is said to be optimum in the mean square error sense. A plot of the mean-square value of the error signal versus the adjustable parameters of a linear filter is referred to as the error signal *performance surface.* The minimum point of this surface represents the wiener solution.

The wiener filter is inadequate for dealing with situation in which *nonstationarity* of the signal and/or noise is intrinsic to the problem. In such situations, the optimum filter has to assume a *time-varying form*. A highly successful solution to this more difficult problem is found in the *Kalman filter*, which is a powerful system with a wide variety of engineering applications.

Linear filter theory, encompassing both Wiener and Kalman filters, is well developed in the literature for continuous time as well as discrete time signals. However, for technical reasons influenced by the wide availability of digital computers and the ever-increasing use of digital signal processing devices, the discrete time representation is often the preferred method.

5.3 ADAPTIVE FILTERS

The design of a Weiner 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 priori information on which the design of the filter is based. When this information is not known completely, however, it may not possible to design the wiener filter or else the design may no longer be optimum. A straightforward approach that we may use in such situation is the “estimate and plug” procedure. This is a two-stage process whereby the filters first “estimate” the statistical parameters of the relevant signals and then “plug” the results so obtained into a *nonrecursive* formula for computing the filter parameters. For real-time operation, this procedure has the disadvantage of requiring excessively elaborated & costly hardware. To mitigate this limitation, we may use *adaptive filter*. By such a system we mean one that is *self-designing* in that the adaptive filter relays for it operation on a *recursive algorithm*, which make it possible for the filter to perform satisfactorily in an environment where complete knowledge of the relevant signal characteristics is not available. The algorithm starts from some predetermined set of *initial conditions*, representing whatever we know about the environment. Yet, in stationary environment, we find that after successive iterations of the algorithm it *converges* to the optimum Weiner solution in some statistical sense. In a nonstationary environment, the algorithm the algorithm offers a *tracking capability* in that it can track time variations in the statistics of the input data provided that the variation are sufficiently slow [10].

As a direct consequence of the application of the 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 in reality a *nonlinear system, in a sense that it does not obey the principle of superposition.* Notwithstanding this property, adaptive filters are classified as linear or nonlinear. An adaptive filter is said to be *linear* if its input output obeys the principle of superposition whenever its parameter are held fixed. Otherwise, the adaptive filter is said to be *nonlinear*.

5.4 WIENER FILTER THEORY

We will explain the Weiner filter theory using its application in adaptive channel equalization. The equalizer considered here is known as the MMSE linear equalizer.

Let us assume that Weiner filter is a FIR filter with $2K+1$ coefficients

$$c_{-K}, c_{-K+1}, c_{-K+2}, \ldots, c_K, \tag{5.1}$$

The input to the filter is the received signal $r(n)$ and the output signal is

$$\hat{s}(n) = \sum_{k=-K}^{K} c_k r(n-k) \tag{5.2}$$

The filter coefficients are chosen to minimize the mean square value of the

$$\varepsilon(n) = s(n) - \hat{s}(n) \tag{5.3}$$

where $s(n)$ is the transmitted symbol and $\hat{s}(n)$ is the its estimate(see figure 5.1). Note that after equalization, there will still be residual ISI. On top of that, there is an additive Gaussian noise term. The MMSE equalizer minimizes the combined residual ISI plus noise power. The non-casuality in the mathematical description of the MMSE equalizer translates into a decision delay in the actual implementation (see figure 5.1). The delay is due to the filtering with channel filter and equalizer filter.

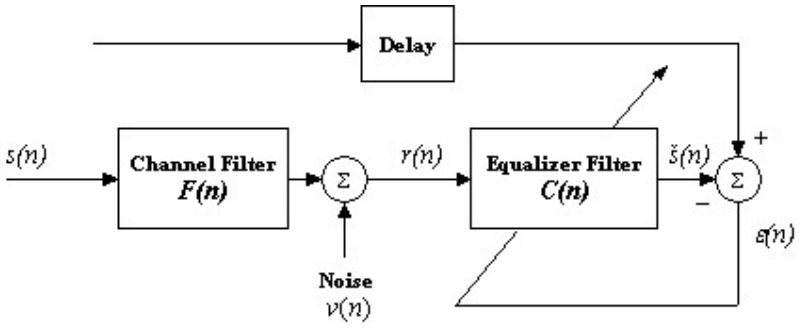

Figure 5.1: The Weiner filter($C(n)$) configuration for equalization

To obtain the filter coefficients of the equalizer filter, we first express all signals involved in matrix form. Specifically, let

$$\mathbf{R}(n) = \begin{bmatrix} r(n+K) \\ r(n+K-1) \\ \vdots \\ r(n-K) \end{bmatrix} \tag{5.4}$$

be the received vector at time n and

$$\mathbf{C} = [c_{-K}, c_{-K+1}, c_{-K+2}, \ldots, c_K] \tag{5.5}$$

be a general transversal (i.e. FIR) equalizer. By substituting for the matrix notation into equation 5.2, it is possible to represent the estimated error signal by equation 5.6 below. The equalizer output at time *n* is thus

$$\hat{s}(n) = \mathbf{CR}(n) \tag{5.6}$$

and instantaneous *squared error* of the signal can be found by squaring equation 5.6 such that it can be represented as the following equation:

$$\begin{aligned} \varepsilon(n)^2 &= \left(s(n) - \hat{s}(n)\right)^2 = \left(s(n) - \mathbf{CR}(n)\right)^2 \\ &= \left(s(n) - \mathbf{CR}(n)\right)\left(s(n) - \mathbf{C}^T\mathbf{R}(n)^T\right) \\ &= s(n)^2 - s(n)\mathbf{CR}(n) - s(n)\mathbf{C}^T\mathbf{R}(n)^T + \mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T \\ &= \sigma^2 - s(n)\mathbf{CR}(n) - s(n)\mathbf{C}^T\mathbf{R}(n)^T + \mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T \end{aligned} \tag{5.7}$$

where σ^2 is the variance of $s(n)$ because $s(n)$ has a zero mean, i.e., $\sigma^2 = E(x^2) - \left(E(x)\right)^2$. Also assuming $s(n)$ a bipolar signal (BPSK or QPSK) with values ±1 then we can say $\sigma^2 = 1$. *Mean square error* (MSE), γ, is defined by the "expectation" of the squared error, from equation 5.7. Hence the MSE can be represented by equation 5.8.

$$\begin{aligned} \gamma = E\left[\varepsilon(n)^2\right] &= 1 - E\left[s(n)\mathbf{CR}(n)\right] - E\left[s(n)\mathbf{C}^T\mathbf{R}(n)^T\right] + E\left[\mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T\right] \\ &= 1 - \mathbf{C}E\left[s(n)\mathbf{R}(n)\right] - E\left[s(n)\mathbf{R}(n)^T\right]\mathbf{C}^T + \mathbf{C}E\left[\mathbf{R}(n)\mathbf{R}(n)^T\right]\mathbf{C}^T \\ &= 1 - \mathbf{C}\mathbf{u}_{Rs} - \mathbf{u}_{sR}\mathbf{C}^T + \mathbf{C}\mathbf{U}_{RR}\mathbf{C}^T \\ &= 1 - 2\mathbf{u}_{ss}\mathbf{C}^T + \mathbf{C}\mathbf{U}_{RR}\mathbf{C}^T \end{aligned} \tag{5.8}$$

where

$$\mathbf{u}_{sR} = E\left[s(n)\mathbf{R}(n)^T\right] = \mathbf{u}_{Rs}{}^T \tag{5.9}$$

is the *2K+1* length cross correlation vector between $s(n)$ and the received vector $\mathbf{R}(n)$, and

$$\mathbf{U}_{RR} = E\left[\mathbf{R}(n)\mathbf{R}(n)^T\right] = E\left[\begin{bmatrix} r(n+K) \\ r(n+K-1) \\ \vdots \\ r(n-K) \end{bmatrix}\begin{bmatrix} r(n+K) & r(n+K-1) & \dots & r(n-K) \end{bmatrix}\right]$$

$$= E\begin{bmatrix} r(n+K)r(n+K) & r(n+K)r(n+K-1) & \dots & r(n+K)r(n-K) \\ r(n+K-1)r(n+K) & r(n+K-1)r(n+K-1) & \dots & r(n+K-1)r(n+K-1) \\ \vdots & \vdots & \ddots & \vdots \\ r(n-K)r(n+K) & r(n-K)r(n+K-1) & \cdots & r(n-K)r(n-K) \end{bmatrix}$$

(5.10)

is the $(2K+1)\times(2K+1)$ order auto correlation matrix of the received vector $\mathbf{R}(n)$. Note that $\mathbf{U}_{RR}$ is a symmetric matrix and $r(i,j) = r(j,i)$.

It is clear from this expression that the mean square error γ is a *quadratic function* of the weight vector $\mathbf{C}$ (filter coefficients). That is, when Equation 5.8 is expanded, the elements of $\mathbf{C}$ will appear in the first and second order only. This is valid when the input components and desired response inputs are wide-sense stationary stochastic (random) variables [18].

5.5 PERFORMANCE SURFACE

A portion of a typical two-dimensional *MSE* function is illustrated in figure 5.2. The vertical axis represents the mean square error and the two horizontal axes represent the values of two filter coefficients. The quadratic error function, or *performance surface*, can be used to determine the optimum weight vector $\mathbf{C}_{opt}$ (or Wiener filter coefficients). With a quadratic performance function there is only one global optimum; no local minima exist. The shape of the function would be hyper-parabolic if there were more than two weights.

In this example, the filter coefficient $c(0)$ varies between $[1,\ldots,3]$ while $c(1)$ varies between the range $[-1,\ldots,1]$. The optimum weight vector is given by $\mathbf{C}_{opt} = [2, -0.1]$, corresponding to the values for which the mean square error takes the minimum value, $\gamma_{\min}$.

Many adaptive processes that cause the weight vector to search for the minimum of the performance surface do so by the *gradient method* [11]. The gradient of the mean square error of the performance surface, designated ∇, can be obtained by differentiating equation 5.8 with respect to each component of the weight vector.

$$\nabla = \frac{\partial \gamma}{\partial \mathbf{C}} = \left[\frac{\partial \gamma}{\partial c_{-K}}, \frac{\partial \gamma}{\partial c_{-K+1}}, \ldots, \frac{\partial \gamma}{\partial c_{K}} \right] \tag{5.12}$$

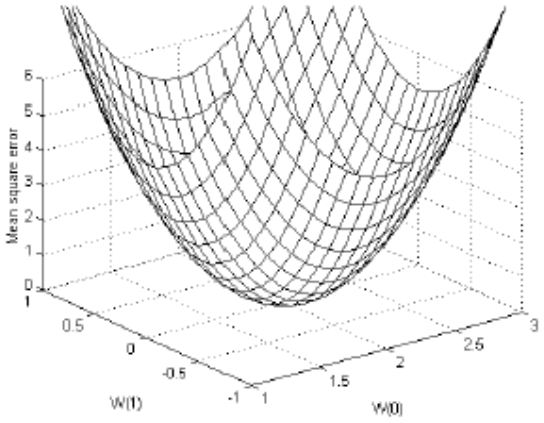

Figure 5.2: A two dimensional error performance surface

Remember that the MSE was derived from the expectation of the squared error function, from equation 5.8. So as an alternative method, the gradient can also be found by differentiating the expected squared error function with respect to the weight vector.

$$\begin{aligned}
\nabla &= E\left[\frac{\partial \varepsilon(n)^2}{\partial \mathbf{C}} \right] = E\left[2\varepsilon(n) \frac{\partial \varepsilon(n)}{\partial \mathbf{C}} \right] \\
&= 2E\left[\left(s(n) - \mathbf{CR}(n) \right) \frac{\partial}{\partial \mathbf{C}} \left(s(n) - \mathbf{CR}(n) \right) \right] \\
&= -2E\left[\left(s(n) - \mathbf{CR}(n) \right) \mathbf{R}(n)^T \right] \\
&= -2E\left[s(n)\mathbf{R}(n)^T \right] + 2E\left[\mathbf{R}(n)\mathbf{R}(n)^T \right] \mathbf{C} \\
&= -2\mathbf{u}_{sR} + 2\mathbf{U}_{RR}\mathbf{C}
\end{aligned} \tag{5.13}$$

The gradient vector is the tangent of the *N*-dimensional surface γ at the point $\mathbf{C}_{opt}$ When the gradient vector is zero, the surface γ reaches its lowest value and hence the mean square error will be minimized. The point in the *N*-dimensional space where this occurs is. Therefore, setting equation 5.13 to zero we get

$$0 = -2\mathbf{u}_{sR} + 2\mathbf{U}_{RR}\mathbf{C}_{opt}$$

$$\mathbf{C}_{opt} = \mathbf{u}_{sR}\mathbf{U}_{RR}^{-1} \tag{5.14}$$

and corresponding MSE is

$$\begin{aligned}\gamma_{\min} &= 1 - 2\mathbf{u}_{sR}\mathbf{C}_{opt}{}^{T} + \mathbf{C}_{opt}\mathbf{U}_{RR}\mathbf{C}_{opt}{}^{T} \\ &= 1 - \mathbf{u}_{sR}\mathbf{U}_{RR}{}^{-1}\mathbf{u}_{sR}{}^{T}\end{aligned} \tag{5.15}$$

This equation is known as the *Wiener-Hopf* equation in matrix form, and the filter given by $\mathbf{C}_{opt}$ in equation 5.14 is the Wiener filter. However, in practice it is not usual to evaluate. In addition, $\mathbf{C}_{opt}$ has to be calculated repeatedly for non-stationary signals and this can be computationally intensive because it requires matrix inversions. But another question is how to determine the values of $\mathbf{u}_{sR}$ and $\mathbf{U}_{RR}$ in case of channel equalization. Recall that $\mathbf{u}_{sR} = E\left[s(n)\mathbf{R}(n)^{T}\right]$. The m-th element of this row vector, where m= 1,2 ….,2K+1 is given as

$$\begin{aligned}u_{sR}(m) &= E\left[s(n)r(n+K+1-m)^{T}\right] \\ &= E\left[s(n)\left\{\sum_{k=0}^{L} f(k)s(n+K+1-m-k) + v(n+K+1-m)\right\}\right] \\ &= \left[\sum_{k=0}^{L} f(k)E\left(s(n)s(n+K+1-m-k)\right)\right] + E\left[s(n)v(n+K+1-m)\right]\end{aligned} \tag{5.16}$$

where $f(n)$ is the impulse response of channel filter and L is the length of $f(n)$. $v(n)$ is the additive white Gaussian noise (see figure 5.1). Assuming that different data bits are statistically independent, then

$$E\left[s(i)s(j)\right] = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{5.17}$$

Furthermore, the data bits are independent of the channel noise. Consequently,

$$E\left[s(i)v(j)\right] = E\left[s(i)v(j)\right] = 0 \tag{5.18}$$

Substituting equations 5.17 and 5.18 into equation 5.16 yields

$$\begin{aligned}u_{sR}(m) &= \left[\sum_{k=0}^{L} f(k)E\left(s(n)s(n+K+1-m-K)\right)\right] + E\left[s(n)v(n+K+1-m)\right] \\ &= \begin{cases} f(K+1-m) & if\ \ 0 \le K+1-m \le L \\ 0 & otherwise \end{cases}\end{aligned} \tag{5.19}$$

Now if $L = 4$ and $K = 3$ (7 tap equalizer), we have

$$\mathbf{u}_{sR} = \left[f(3), f(2), f(1), f(0), 0, 0, 0\right]$$

this is not desired because $f(4)$ is not included. However if K=5, then

$$\mathbf{u}_{sR} = [0, f(4)f(3), f(2), f(1), f(0), 0, 0, 0, 0, 0]$$

which is a desired result that's why selection of K (equalizer length $2K+1$) is very critical for a particular L (channel impulse response length).

The element on the i-th row and j-th column i , j =1,2,...,$2K$ = 1, of the covariance matrix $\mathbf{U}_{RR}$ is

$$\begin{aligned} U_{RR}(i,j) &= E\left[r(n+K+1-i)r(n+K+1-j)\right] \\ &= E\left[\begin{array}{l}\sum_{k=0}^{L} f(k)s(n+K+1-i-k)+v(n+K+1-i)\times \\ \sum_{m=0}^{L} f(m)s(n+K+1-j-m)+v(n+K+1-j)\end{array}\right] \\ &= \sum_{k=0}^{L}\sum_{m=0}^{L} f(k)f(m)E\left(s(n+K+1-j-m)s(n+K+1-i-k)\right) \\ &= \quad +\sum_{m=0}^{L} f(k)E\left(s(n+K+1-i-k)v(n+K+1-j)\right) \\ &= \quad +\sum_{m=0}^{L} f(k)E\left(s(n+K+1-j-m)v(n+K+1-i)\right) \\ &= \quad +v(n+K+1-i)v(n+K+1-j) \end{aligned} \tag{5.20}$$

Substituting equations 5.17 and 5.18 into equation 5.20 yields

$$\begin{aligned} U_{RR}(i,j) &= \sum_{k=o}^{L} f(k)f(k+i-j)+\sigma_v{}^2\delta(i-j) \\ &= U_{RR}(j,i) \end{aligned} \tag{5.21}$$

where it is understood that

$$f(k+i-j) = \begin{cases} f(k+i-j) & 0 \le k+i-j \le L \\ 0 & otherwise \end{cases} \tag{5.22}$$

Note that $\delta(n)$ is the discrete-time impulse function and that the first term $U_{RR}(i,j)$ is actually the auto correlation function of the channel's impulse response. We will denote it by

$$\sum_{k=o}^{L} f(k)f(k+(i-j)) = \phi_{ff}(i-j) = \phi_{ff}(j-i) \tag{5.23}$$

and in the matrix form

$$\mathbf{U}_{RR} = \begin{bmatrix} \phi_{ff}(0)+\sigma_v{}^2 & \phi_{ff}(-1) & \cdots & \phi_{ff}(-2K) \\ \phi_{ff}(1) & \phi_{ff}(0)+\sigma_v{}^2 & \cdots & \phi_{ff}(-2K+1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{ff}(2K) & \phi_{ff}(2K-1) & \cdots & \phi_{ff}(0)+\sigma_v{}^2 \end{bmatrix} \tag{5.24}$$

In order to implement the MMSE equalizer the receiver needs to know what the correlation vector $\mathbf{u}_{sR}$ and the covariance matrix $\mathbf{U}_{RR}$ are. As shown in

equations 5.19 and 5.24, these parameters depends on the channel conditions, namely the impulse response *f*(*n*) of the channel and the noise variance σ_v^2. The channel conditions are usually unknown to the receiver prior to the communication session. Moreover, they may vary slowly with time after a connection has been established. What this means is that the receiver must estimate $\mathbf{u}_{sR}$ and $\mathbf{U}_{RR}$ online, either explicitly or implicitly. There is an iterative procedure called the *least mean square (LMS)* algorithm for determining the coefficients of a MMSE equalizer in an unknown channel.

The LMS algorithm is based on the *method of steepest descent*. The key difference between the method of steepest descent and LMS is that the former uses the true gradient vector (which depends on the channel condition) of the error surface γ in the iterations while the latter uses an estimate of the gradient vector. The use of noisy gradients in the LMS algorithm leads to a MSE γ that is slightly larger than that of the true MMSE equalizer.

5.6 STEEPEST DESCENT ALGORITHM

In practice it is not usual to calculate the optimum filter $\mathbf{C}_{opt}$ using **equation 14** directly. The problem is that the evaluation of $\mathbf{U}_{RR}^{-1}$ involves the inversion of a matrix of dimension $2K+1$ by $2K+1$ which is computationally very complex. Furthermore, if the channel statistics are non-stationary, which is quite often the case, then the calculation has to be undertaken periodically in order to track the changing conditions. An alternative method of calculation is therefore the *steepest descent* algorithm. In this method the weights are adjusted iteratively in the direction of the gradient. Let $\mathbf{C}(n)$ be the estimate of $\mathbf{C}_{opt}$ at discrete-time (or iteration index) n. Then based on $\mathbf{C}(n)$, we can obtain

$$\nabla(n) = -2\left[\mathbf{u}_{sr} - \mathbf{C}(n)\mathbf{U}_{RR}\right] \tag{5.25}$$

The gradient vector of the error surface γ at $\mathbf{C}(n)$; see the equation 5.13 Based $\mathbf{C}(n)$ and $\nabla(n)$, we obtain the next estimate of $\mathbf{C}_{opt}$ according to

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n) - \frac{\mu}{2}\nabla(n) \\ &= \mathbf{C}(n) + \mu\left[\mathbf{u}_{sr} - \mathbf{C}(n)\mathbf{U}_{RR}\right] \\ &= \mathbf{C}(n)\left[\mathbf{I} - \mu\mathbf{U}_{RR}\right] + \mu\mathbf{u}_{sr}\end{aligned} \tag{5.26}$$

where μ is the step size of this iterative procedure, and $\mathbf{I}$ is an identity matrix of $2K+1$ by $2K+1$. Basically, the method of steepest descent is based on the idea that we can reach the global minimum by searching in the opposite direction as indicated by the gradient vector (slope) at the current estimate; see the scalar case below as an example.

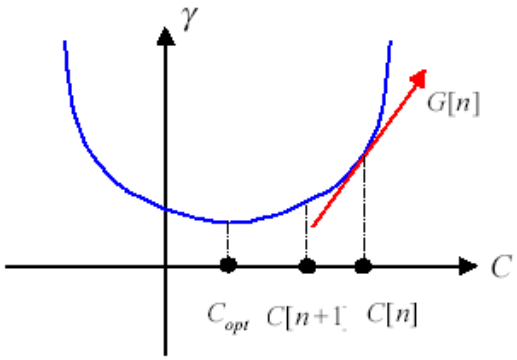

Figure 5.3: Operation of steepest descent algorithm.

To prove that the global minimum can indeed be reached through this iterative procedure, let us first rewrite $\mathbf{C}(n+1)$ as

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n)\left[1-\mu\mathbf{U}_{RR}\right]+\mu\mathbf{u}_{sR}\\ &= \left[\mathbf{C}(n-1)\left\{\mathbf{I}-\mu\mathbf{U}_{RR}\right\}+\mu\mathbf{u}_{sR}\right]\left(1-\mu\mathbf{U}_{RR}\right)+\mu\mathbf{u}_{sR}\\ &= \quad\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots\\ &= \mathbf{C}(0)\left[\mathbf{I}-\mu\mathbf{U}_{RR}\right]^{n+1}+\mu\mathbf{u}_{sR}\sum_{k=0}^{n}\left[\mathbf{I}-\mu\mathbf{U}_{RR}\right]^{k}\end{aligned} \tag{5.27}$$

where $\mathbf{C}(0)$ denotes the initial estimate. Since the matrix $\mathbf{U}_{RR}$ is a covariance matrix, it is positive definite and can be written as

$$\mathbf{U}_{RR} = \mathbf{V}\mathbf{D}\mathbf{V}^{T} \tag{5.28}$$

where $\mathbf{V}$ is a *unitary matrix* with the property

$$\mathbf{V}\mathbf{V}^{T} = \mathbf{V}^{T}\mathbf{V} = \mathbf{I} \tag{5.29}$$

and

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_{2K+1} \end{bmatrix} \tag{5.30}$$

is a diagonal matrix containing all the eigenvalues of $\mathbf{U}_{RR}$. Note that

$$\lambda_i > 0; \qquad i = 0,1,\ldots,2K+1 \tag{5.31}$$

because $\mathbf{U}_{RR}$ is positive definite, and that

$$\mathbf{U}_{RR}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T \tag{5.32}$$

Based on these properties of $\mathbf{U}_{RR}$, we can express the terms $\mathbf{I} - \mu\mathbf{U}_{RR}$ and $(\mathbf{I} - \mu\mathbf{U}_{RR})^k$ in equation 5.27 as

$$\begin{aligned} \mathbf{I} - \mu\mathbf{U}_{RR} &= \mathbf{V}\mathbf{V}^T - \mu\mathbf{V}\mathbf{D}\mathbf{V}^T \\ &= \mathbf{V}(\mathbf{I} - \mu\mathbf{D})\mathbf{V}^T \\ &= \mathbf{V}\mathbf{Q}\mathbf{V}^T \end{aligned} \tag{5.33}$$

and

$$(\mathbf{I} - \mu\mathbf{U}_{RR})^k = \mathbf{V}\mathbf{Q}^k\mathbf{V}^T \tag{5.34}$$

where

$$\mathbf{Q} = \mathbf{I} - \mu\mathbf{D} = \begin{bmatrix} 1-\mu\lambda_1 & & & \\ & 1-\mu\lambda_2 & & \\ & & \ddots & \\ & & & 1-\mu\lambda_{2K+1} \end{bmatrix} \tag{5.35}$$

if

$$\left|1 - \mu\lambda_i\right| < 1; \qquad i = 0,1,2,\ldots,2K+1 \tag{5.36}$$

then

$$\lim_{n\to\infty}(\mathbf{I} - \mu\mathbf{U}_{RR})^{n+1} = 0 \tag{5.37}$$

and

$$\begin{aligned} \lim_{n\to\infty}\sum_{k=o}^{n}(\mathbf{I} - \mu\mathbf{U}_{RR})^k &= \sum_{k=0}^{\infty}\mathbf{V}\mathbf{Q}^k\mathbf{V}^T \\ &= \mathbf{V}\left[\sum_{k=0}^{\infty}\mathbf{Q}^k\right]\mathbf{V}^T \\ &= \mathbf{V}(\mathbf{I} - \mathbf{Q}^{-1})\mathbf{V}^T \\ &= \mathbf{V}(\mu\mathbf{D}^{-1})\mathbf{V}^T \\ &= \frac{1}{\mu}\mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T \\ &= \frac{1}{\mu}\mathbf{U}_{RR} \end{aligned} \tag{5.38}$$

In this case, equation 5.27 becomes

$$\lim_{n\to\infty}\mathbf{C}(n+1) = \mathbf{u}_{sR}\mathbf{U}_{RR} = \mathbf{C}_{opt} \tag{5.39}$$

In other word, the equalizer's coefficients eventually converge to the optimal values. Equation 5.36 provides the requirements for the convergence of the

steepest descent algorithm. These requirements can be written alternatively as

$$0 < \mu < \frac{2}{\lambda_i}; \qquad i = 1,2,\ldots,2K+1 \tag{5.40}$$

or simply

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{5.41}$$

where $\lambda_{\max}$ is the largest eigenvalue of the covariance matrix $\mathbf{U}_{RR}$.

For the steepest descent algorithm at hand, it is desirable to use a step size as close to the upper limit, $2/\lambda_{\max}$, as possible. This guarantees the fastest convergence rate possible. However, for the LMS algorithm discussed in next section, using too large a step size will increase the MSE of the equalizer. Assume we select a step size of

$$\mu = \frac{2}{\lambda^{+}_{\max}} \tag{5.42}$$

where $\lambda^{+}_{\max}$ is a number slightly greater than $\lambda_{\max}$. Then the absolute values of the elements of the diagonal matrix $\mathbf{Q}^{n+1} = (\mathbf{I} - \mu\mathbf{D})^{n+1}$ are

$$\left|1 - \mu\lambda_i\right|^{n+1} = \left|1 - \frac{2\lambda_i}{\lambda^{+}_{\max}}\right|^{n+1}; \qquad i = 1,2,\ldots,2K+1 \tag{5.43}$$

The rate of decay of the matrix, and consequently the rate of decay of the matrix $(\mathbf{I} - \mu\mathbf{U}_{RR})^{n+1} = \mathbf{V}\mathbf{Q}^{n+1}\mathbf{V}^{T}$ in equation 5.27, is determined by the smallest eigenvalue $\lambda_{\min}$ according to

$$\left|1 - \frac{2\lambda_{\min}}{\lambda^{+}_{\max}}\right|^{n+1} \tag{5.44}$$

If $\lambda_{\min} << \lambda_{\max}$, i.e. a large eigenvalue spread, then $\left|1 - 2\lambda_{\min}/\lambda_{\max}\right| \to 1$ and the algorithm converges slowly. A large eigenvalue spread occurs when the frequency response of the channel has deep spectral nulls.

5.6.1 A simple choice for μ

From the equation 5.24, we see that all the diagonal elements of the covariance matrix $\mathbf{U}_{RR}$ equal $\phi_{ff}(0) + \sigma_v^{\,2}$, where $\phi_{ff}(0)$ is the autocorrelation

function of the channel $f(n)$ at a delay of zero, and σ_v^2 is the power (or variance) of the channel's noise. The sum of all the diagonal elements of $\mathbf{U}_{RR}$ is the trace of the matrix. It is the same as the sum of all the eigenvalues of the matrix. Since

$$\lambda_{\max} < \sum_{i=1}^{2K+1} \lambda_i = trace\{\mathbf{U}_{RR}\} = (2K+1)\left(\phi_{ff}(0) + \sigma_v^2\right) \tag{5.45}$$

if we use as step size

$$\eta = \frac{2}{(2K+1)\left(\phi_{ff}(0) + \sigma_v^2\right)} \tag{5.46}$$

then it is guaranteed that $\mu < 2/\lambda_{\max}$.

For the LMS algorithm discussed in the next section, it is import to make μ substantially smaller than $2/\left[(2K+1)\left(\phi_{ff}(0) + \sigma_v^2\right)\right]$ that the excess MSE is relatively small compared to the MSE of the MMSE equalizer.

5.7 THE LMS ALGORITHM

While the steepest descent method is able to determine the optimal equalizer coefficients without performing any matrix inversion, its operation is still based on the assumption that the channel parameters the correlation vector $\mathbf{u}_{sR}$ and the covariance matrix $\mathbf{U}_{RR}$ known to the receiver. Recall that the receiver uses these parameters to compute the gradient vector $\nabla(n)$ required for updating the the equalizer coefficients. In the LMS algorithm, the gradient vector is replaced by its estimate. Let us consider the correlation of the received vector $\mathbf{R}(n)$ (see equation 5.4) with the equalization error

$$\varepsilon(n) = s(n) - \hat{s}(n) = s(n) - \mathbf{C}\mathbf{R}(n) \tag{5.47}$$

for the equalizer $\mathbf{C}$. The result is

$$\begin{aligned} \mathbf{w}_{\varepsilon R} &= E\left[\varepsilon(n)\mathbf{R}(n)^T\right] \\ &= E\left[\varepsilon(n)\mathbf{R}(n)^T - \mathbf{C}\mathbf{R}(n)\mathbf{R}(n)^T\right] \\ &= E\left[\varepsilon(n)\mathbf{R}(n)^T\right] - E\left[\mathbf{C}\mathbf{R}(n)\mathbf{R}(n)^T\right] \\ &= \mathbf{u}_{sR} - \mathbf{C}\mathbf{U}_{RR} \end{aligned} \tag{5.48}$$

Now, if we evaluate this correlation vector at $\mathbf{C} = \mathbf{C}(n)$, then we obtain

$$\mathbf{w}_{\varepsilon R}\Big|_{\mathbf{C}=\mathbf{C}(n)} = \mathbf{u}_{sR} - \mathbf{C}(n)\mathbf{U}_{RR} = -\frac{1}{2}\nabla(n) \tag{5.49}$$

where $\nabla(n)$ is the gradient vector at the point $\mathbf{C} = \mathbf{C}(n)$ of the error surface γ. To simply put, the updating equation in the steepest descent method can be rewritten as

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n) - \frac{\mu}{2}\nabla(n) \\ &= \mathbf{C}(n) + E\left[\varepsilon(n)\mathbf{R}(n)^T\right]\end{aligned} \tag{5.50}$$

The LMS algorithm is obtained by removing the average operator in the above equation, i.e.

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n) + E\left[\varepsilon(n)\mathbf{R}(n)^T\right] \\ &= \mathbf{C}(n) + \left[s(n) - \hat{s}(n)\right]\mathbf{R}(n)^T\end{aligned} \tag{5.51}$$

In other word, the LMS algorithm uses a noisy estimate of the true gradient in updating the filter coefficients. Intuitively, this substitution is justified when all the random processes in the system are ergodic. In this case, the statistical average (as in the steepest gradient algorithm) equals the time average (as in the case of LMS). Notice that the LMS-based equalizer does not need any information about the channel to update its coefficients. The algorithm however does require knowledge of the transmitted symbols, which are supposed to be unknown to the receiver. In practice, we can get around this problem by using the detected symbol $\tilde{s}(n)$ instead of $s(n)$ when we update the equalizer coefficients. Assuming $s(n) = \pm 1$ (BPSK) signal

$$\tilde{s}(n) = \begin{cases} +1 & if\ \hat{s}(n) > 0 \\ -1 & if\ \hat{s}(n) < 0 \end{cases} \tag{5.52}$$

Furthermore, we can send an initial training sequence to help the equalizer converges quickly. The symbols in the training sequence will be known to the receiver. However, frame/bit synchronization are required in order for the receiver to locate the training pattern. The use of noisy gradients in the adaptation process results in excess MSE. Specifically, the MSE of a LMS-based adaptive transversal equalizer is

$$\gamma \approx \gamma_{\min} + \gamma_{\mu} \tag{5.53}$$

where $\gamma_{\min}$ is the MMSE defined in equation 5.15, and

$$\gamma_{\mu} \approx \frac{1}{2}\mu\left[(2K+1)\left(\phi_{ff}(0) + \sigma_v{}^2\right)\right]\gamma_{\min} \tag{5.54}$$

is the excess MSE. In order to make γ_μ substantially smaller than $\gamma_{\min}$ while maintaing a reasonable convergence rate, the step size μ should only be a fraction of $2/\left[(2K+1)\left(\phi_{ff}(0)+\sigma_v{}^2\right)\right]$. For example, when

$$\mu = \frac{1}{10}\left[\frac{2}{(2K+1)\left(\phi_{ff}(0)+\sigma_v{}^2\right)}\right] = \frac{0.2}{(2K+1)\left(\phi_{ff}(0)+\sigma_v{}^2\right)}$$

then $\gamma_\mu - 0.1\gamma_{\min}$ and the increase in total MSE is

$$10\log_{10}\left(\frac{\gamma_{\min}+\gamma_\mu}{\gamma_{\min}}\right) = 0.414\,\mathbf{dB}$$

As shown previously, using a step size of $2/\left[(2K+1)\left(\phi_{ff}(0)+\sigma_v{}^2\right)\right]$ or smaller will guarantee convergence.

While the LMS algorithm enables us to find the optimal equalizer coefficients without any prior knowledge of the channel, it suffers from one drawback – slow convergence. A faster algorithm, but more complex, is the recursive least square (RLS) algorithm. The figure below compares the convergence rate of LMS (labeled as Gradient algorithm in the figure) and RLS (labeled as Kalman). We will not discuss RLS algorithm because we are interested in only MMSE equalizer.

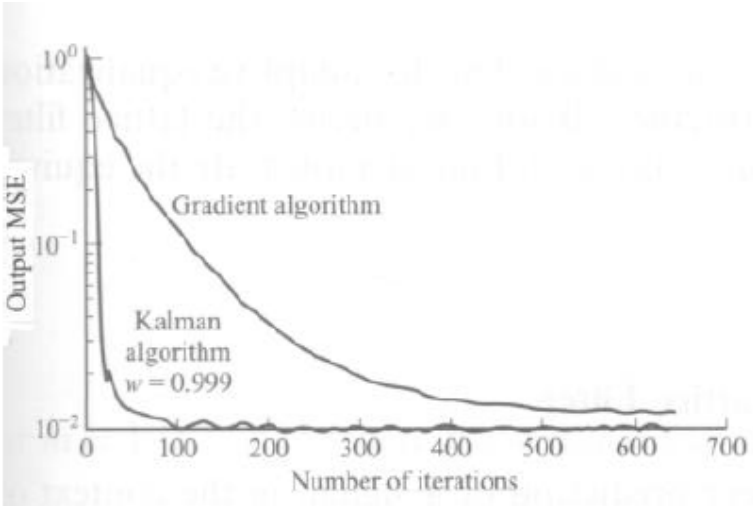

Figure 5.4: Learning curves for LMS and RLS algorithms

5.8 SIMULATION RESULTS

We have run a simulation to investigate the effect of step size μ and eigenvalue spread on the convergence of LMS algorithm. The channel is raised cosine filter with 3 coefficients

$$h_n = \begin{cases} \frac{1}{2}\left[1+\cos\left\{\frac{2\pi}{W}(n-2)\right\}\right] & n = 1,2,3 \\ 0 & Otherwise \end{cases} \quad (5.55)$$

where W controls the amount of amplitude distortion. The impulse response of the channel for W =2.9 is given below

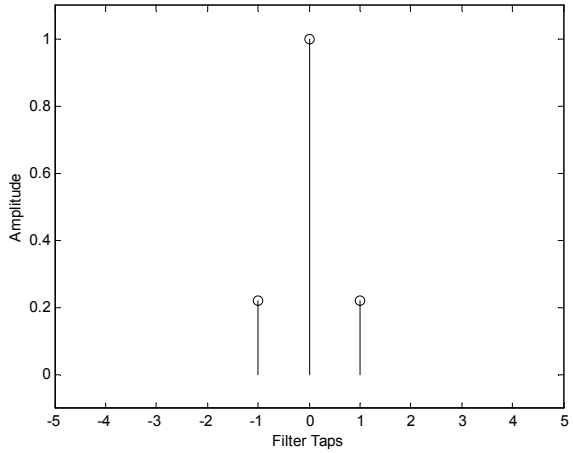

Figure 5.5: Impulse response of channel for W =2.9

The equalizer filter taps are chosen to be equal to 11. Equalizer impulse response after running the simulation for 100 independent trials having 1500 hundred samples each is given below. The data signal is BPSK signal with values ±1 and the variance of noise is .001.

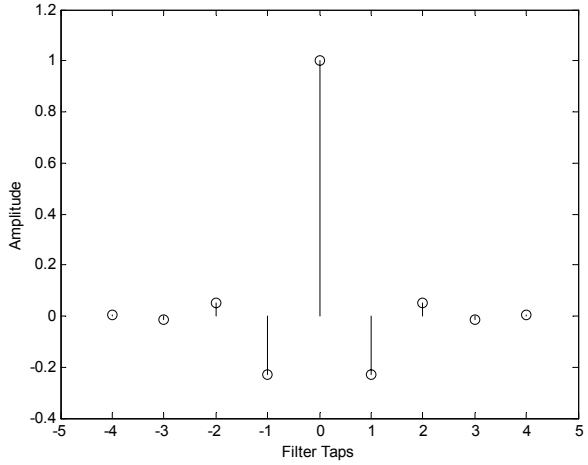

Figure 5.6: Equalizer impulse response

From above two figures it is clear that impulse responses of equalizer and channel are inverse of each other. Now varying the step size while keeping the W constant (same eigenvalue spread) the learning curves for LMS algorithm are given below

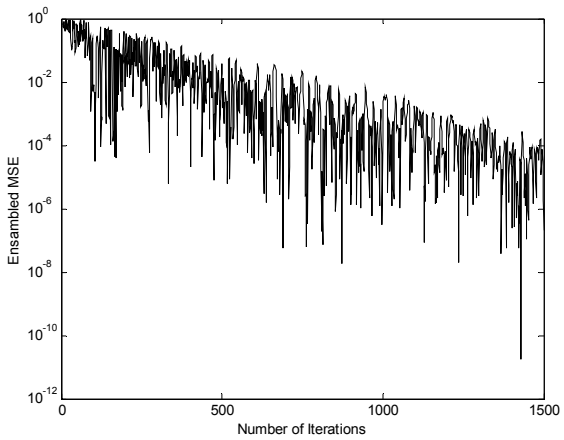

Figure 5.7: Learning curves for LMS algorithm for $\mu = 0.0075$

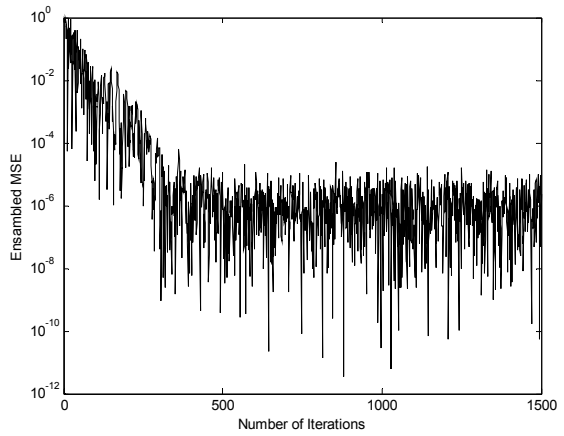

Figure 5.8: Learning curve for LMS algorithm $\mu = 0.0375$

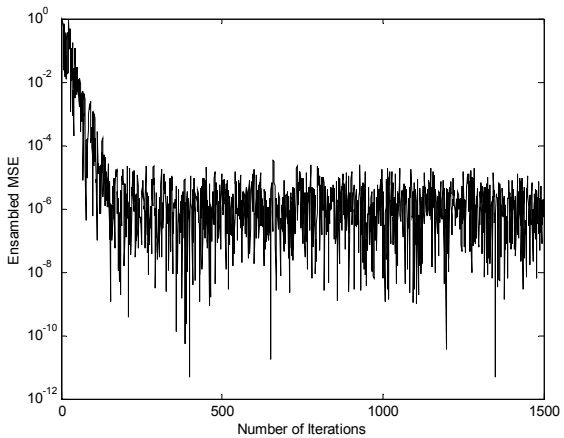

Figure 5.9: Learning curve for LMS algorithm $\mu = 0.075$

From the simulation results it is clear that by increasing the step size the faster convergence of LMS algorithm can be achieved. But care must be taken while increasing the step size because LMS algorithm may become unstable if the upper bound on the step size is reached.

CHAPTER 6

CHANNEL EQUALIZATION

6.1 INTRODUCTION

Communication channels are susceptible to Intersymbol Interference (ISI). Without channel equalization, the utilization of the channel bandwidth becomes inefficient. Channel equalization is a process of compensating for the effects caused by a band-limited channel, hence enabling higher data rates. Equalization describes a set of operations intended to eliminate ISI and the effects of multipath propagation in communication channels. One can define an equalizer as

> *"An equalizer is a device that compensates for unwanted channel effects and provides the receiver with a sequence of samples with acceptable levels of ISI".*

These disruptive effects are due to the dispersive transmission medium (e.g. telephone cables) and the multipath effects in the radio channel. A typical communication system is depicted in figure 6.1 where the equalizer is incorporated within the receiver while the channel introduces intersymbol interference. The transfer function of the equalizer is an estimate of the direct inverse of the channel transfer function. To transmit high speed data over a bandlimited channel, the frequency response of the channel is usually not known with sufficient precision to design an optimum match filter. The equalizer is, therefore, designed to be adaptive to the channel variation. The configuration of an adaptive linear equalizer is depicted in figure 6.2. Based on the observed channel output, an adaptive algorithm recursively updates the equalizer to reconstruct the output signal.

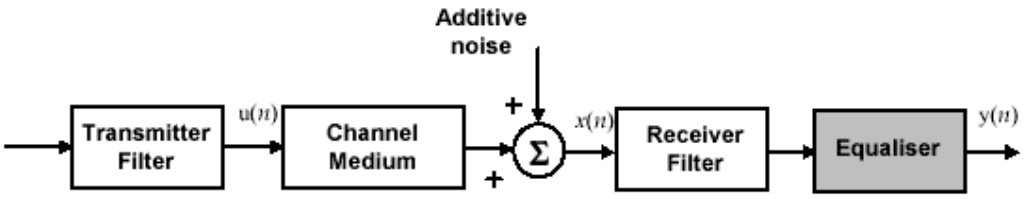

Figure 6.1: A typical communication system.

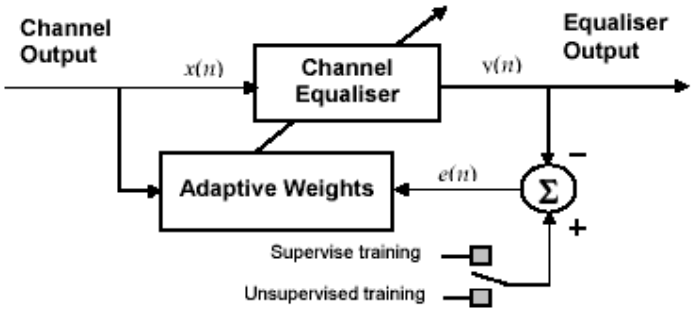

Figure 6.2: A simple linear channel equalizer configuration.

Equalization does not mean that all the channel distortions are completely removed but its job is to provide the receiver enough information which is necessary to make a decision.

6.2 ISI DUE TO MULTIPATH EFFECTS

In a wireless radio channel, the ISI is caused by the multipath effects when the multipath spread as explained previously is greater than the symbol interval. Multipath effects describe the situation in which there are several propagation paths from transmitter to receiver. Most commonly, this results when there are reflected signals detected at the receiver following the direct path. The multipath phenomenon can be modeled by an FIR system (see chapter 3). The center tap represents the direct path, while the succeeding tap weights represent the amplitudes, delays, and phases of the reflected paths. For simple examples, see the two cases described in Figure 6.3 and Figure 6.4.

Figure 6.3(a) shows the time response of an ideal transmission path, which is a δ function. Such a channel exerts no spectral distortion or delayed signals. Figure 6.3(b) shows the spectral response of such a system. Note that the frequency magnitude response is perfectly flat, as indicated by the solid horizontal line.

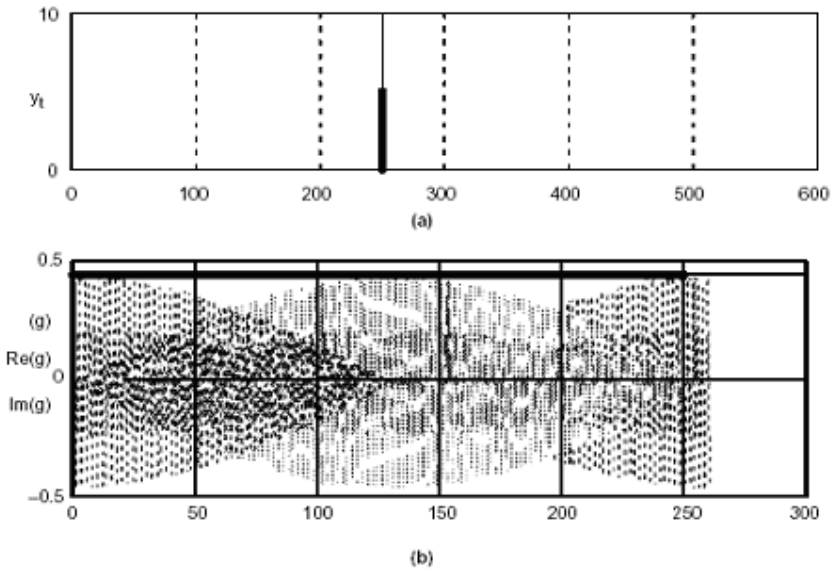

Figure 6.3: (a) Impulse response and (b) frequency response of ideal single path channel

Figure 6.4(a) shows the time response of a system that contains a single multipath channel [8]. The first nonzero sample of the response represents the direct path, while the second represents a delayed path to the receiver. In this instance, the pulses are identical in amplitude and phase and are separated by ten sample intervals.

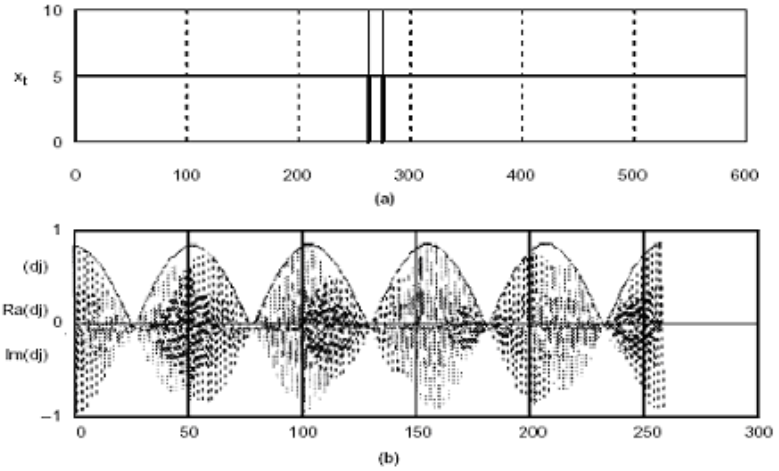

Figure 6.4: (a) Time response and (b) frequency response of two path wireless channel

Notice in Figure 6.4(b) that the magnitude response exhibits $t_0/2$ nulls, where t_0 represents the sample delay. Even though you are effectively adding two identical flat spectra (as shown in Figure 6.3(b)) the time delay results in a phase delay in the spectral domain. This phase delay results in nulls where

the two signals are of equal amplitude but opposite phase. Obviously, multipath effects can have major effects on the system spectral response, thereby providing another justification for channel equalization

Now it is time to represent it mathematically what we all said so far. Suppose that a channel model (see figure 6.5) is used to describe the distortion effect and it is given as a sum of weighted time delayed discrete-time channel impulse responses, *H(z)*:

$$H(z) = \sum_i h_i z^{-i} = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots \tag{6.1}$$

The coefficients h_i represent the strength of the dispersion and the multipath delay. For FIR modelled channel, the output from the channel can be written as:

$$y(k) = \sum_{i=0}^{L-1} h_i u(k-i) + n(k) \tag{6.2}$$

where $y(k)$ is the input to the equalizer (received signal after passing through channel) which is simply the convolution of h_i and $u(k)$. $u(k)$ is the transmitted sequence, h_i is the channel impulse response, $n(k)$ represents additive white Gaussian noise (AWGN) added to the channel and L represents the length of the channel impulse response. Equation (6.2) shows that the transmitted symbol $u(k)$ is affected by the weighted delay symbols of $u(k\text{-}i)$, thus causing intersymbol interference.

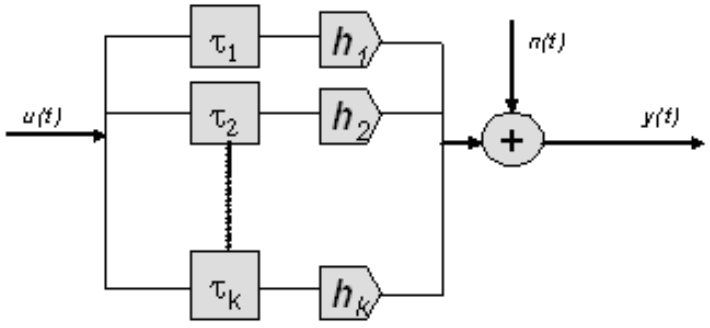

Figure 6.5: A multipath wireless channel model.

6.3 COMMUNICATION SYSTEM MODEL WITH EQUALIZER

6.3.1 Continuous-time Model

For our communication system which is employing a linear modulation, QPSK, through a dispersive channel, the whole system can be described the conceptual model in Figure 6.6, in which the sequence of information symbols is denoted by $\{I_k\}$ and $H_T(f), H_C(f)$ and $H_R(f)$ are the transfer functions of the transmission (root raised cosine pulse-shaping) filter, the dispersive channel and the receiving filter, respectively. The Nyquist condition for no ISI developed in previous chapter can be easily generalized to the above communication system. Letting $X(f) = H_T(f)H_C(f)H_R(f)$ the condition for no ISI is that the folded spectrum $X(f)$, is constant for all frequencies, i.e.

$$\sum_{n=-\infty}^{\infty} X(f - \frac{n}{T}) = T \tag{6.3}$$

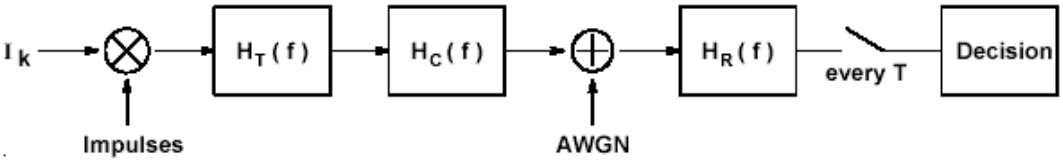

Figure 6.6: Continuous-time communication model over a multipath dispersive channel

One method to achieve the Nyquist condition is to fix the receiving filter to be the matched filter, i.e. set $H_R(f) = H_T{}^*(f)H_C{}^*(f)$, and choose the transmission filter so that (6.5) is satisfied. This is the Nyquist pulse design method described in previous chapter. The major disadvantage of this pulse shaping method is that it is in general difficult to construct the appropriate analog filters for $H_T(f)$ and $H_R(f)$ in practice. Moreover, we have to know the channel response $H_C(f)$ in advance to construct the transmission and receiving filters.

An alternative method is to fix the transmission filter4 and choose the receiving filter $H_R(f)$ to satisfy the condition in **(6.3)**. As for the previous method, it is also difficult to build the appropriate analog filter $H_R(f)$ to eliminate ISI. However, notice that what we want eventually are the samples at intervals T at the receiver. Therefore, we may choose to build a simpler (practical) filter $H_R(f)$, take samples at intervals T, and put a digital filter, called *equalizer*, at the output to eliminate ISI as shown below in Figure 6.7.

This approach to remove ISI is usually known as *equalization*. The main advantage of this approach is that a digital filter is easy to build and is easy to alter for different equalization schemes, as well as to fit different channel conditions.

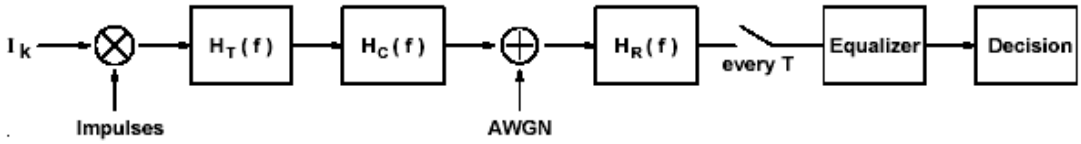

Figure 6.7: Communication system with equalizer

6.3.2 Equivalent discrete-time model

Our goal is to design the equalizer which can remove (or suppress) ISI. To do so, we translate the continuous-time communication system model in Figure 6.7 to an equivalent discrete-time model that is easier to work with. The following steps describe the translation process:

- Instead of considering AWGN being added before the receiving filter $H_R(f)$, we can consider an equivalent colored Gaussian noise being added after $H_R(f)$ when we analyze the system. The equivalent colored noise is the output of $H_R(f)$ due to AWGN. The resulting model is shown in Figure 6.8.
- We input a bit or a symbol to the communication system every T seconds, and get back a sample at the output of the sampler every T seconds. Therefore, we can represent the communication system in Figure 6.8 from the information source to the sampler as a digital filter.

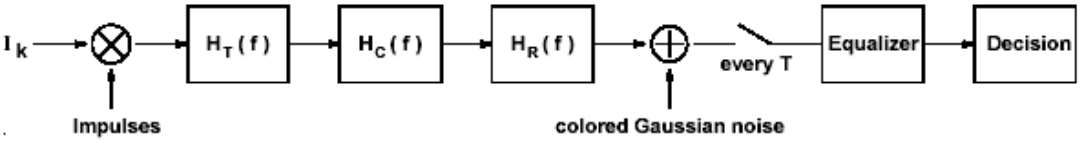

Figure 6.8: Equivalent communication system with colored Gaussian noise

Since $H_T(f), H_C(f)$ and $H_R(f)$ are LTI filters, they can be combined and represented by an equivalent digital LTI filter. Denote its transfer function by *H(z)* and its impulse response by $\{h_k\}_{k=-\infty}^{\infty}$. The result is the *discrete time-*

linear filter model shown in Figure 6.9, in which the output sequence $\{I'_k\}$ is given by

$$\begin{aligned} I'_k &= \sum_j I_k h_{k-j} + n_k \\ &= I_k h_0 + \sum_{j \neq k} I_k h_{k-j} + n_k \end{aligned} \tag{6.4}$$

In general, $h_j \neq 0$ for some $j \neq 0$. Therefore, ISI is present. Notice that the noise sequence $\{n_k\}$ consists of samples of the colored Gaussian noise (AWGN filtered by $H_R(f)$), and is not white in general.

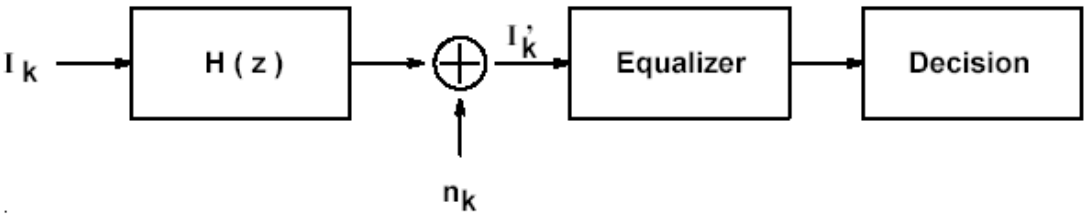

Figure 6.9: Equivalent discrete-time communication system model with colored noise

- Usually, the equalizer consists of two parts, namely, a *noise-whitening* digital filter $H_W(z)$ and an equalizing circuit that equalizes the noise-whitened output as shown in Figure 6.9. The effect of $H_W(z)$ is to "whiten" the noise sequence so that the noise samples are uncorrelated. Notice that $H_W(z)$ depends only on $H_R(f)$, and can be determined a prior according to our choice of $H_R(f)$. At the output of $H_W(z)$, the noise sequence is white. Therefore, equivalently, we can consider the equivalent discrete-time model shown in Figure 6.11, in which $\{\overline{n}_k\}$ is an AWGN sequence.

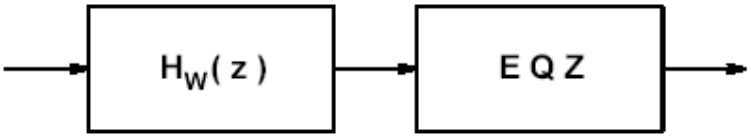

Figure 6.10: Typical equalizer

- Let $G(z) = H(z)H_W(z)$. The communication system from the information source to the output of the noise whitening filter can now be represented by the *discrete-time white-noise linear filter model* in Figure 6.12. The output sequence $\{\tilde{I}_k\}$ is given by:

$$\begin{aligned}\tilde{I}_k &= \sum_j I_k g_{k-j} + \bar{n}_k \\ &= I_k g_0 + \sum_{j \neq k} I_k g_{k-j} + \bar{n}_k\end{aligned} \tag{6.5}$$

Where $\{g_k\}$ is the impulse response corresponding to the transfer function *G(z)*, and $\{\bar{n}_k\}$ is an AWGN sequence. We will work with this discrete-time model in all the following sections.

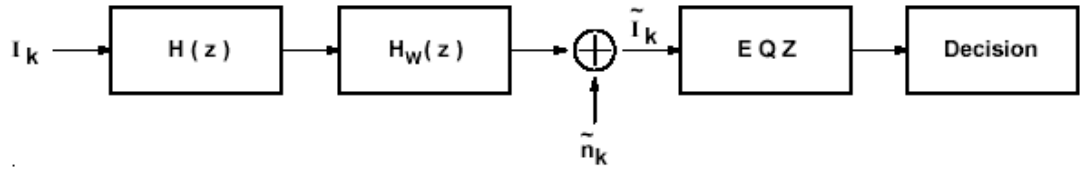

Figure 6.11: Equivalent discrete-time communication system model with white noise

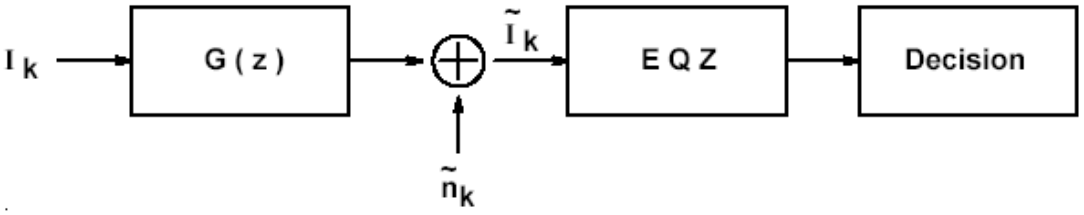

Figure 6.12: Equivalent discrete-time white-noise linear filter model

Finally, the equalizing circuit (we simply call it the equalizer from now on) attempts to remove ISI from the output of *G(z)*. The focus of our coming discussion is the design of this equalizer. Suppose that the equalizer is also an LTI filter with transfer function $H_E(z)$ and corresponding impulse response $\{h_{Ej}\}$. Then the output of the equalizer is given by

$$\hat{I}_k = \sum_j \tilde{I}_{k-j} h_{Ej} \tag{6.6}$$

Ideally $\hat{I}_k$ contains only contributions from the current symbol and the AWGN sequence with small variance.

6.4 CLASSIFICATION OF EQUALIZERS

Equalizers are classified into two main classes. Linear Equalizers Non-linear Equalizers

Linear equalizers (LE) only have feedback from linear devices and can be implemented as a simple FIR filter (transversal filter)also called linear transversal equalizer (LTE). They are easy to implement cheap, suboptimal performance, high BER they have problem like enhances noise and bad for channels with spectral nulls as we will see shortly. They can also be implemented as lattice filter to achieve numerical stability and fast convergence but they are more complicated to implement.

Non-linear equalizers (NLE) have feedback from non-linear devices (i.e. quantizer) and they have better performance than linear equalizers like fine with spectral nulls and also fine with large distortion.

Decision Feedback Equalizers (DFE) are non linear equalizers. They are more complex than a linear transversal equalizer. They have both feed forward and feedback filters. They are cheap, better performance than LTE and they can equalize severely distorted channels & handle spectral nulls.

Maximum Likelihood Symbol Detection (MLSD) is also included in the class of non linear equalizers. They have optimal performance at the cost of high cost & exponential computational complexity. They use trellis approach with probability methods and Viterbi algorithm.

Maximum Likelihood Sequence Estimation (MLSE) is another category of non linear equalizers. Like MLSD they also have optimal performance. They are different from MLSD because ML applied to sequences rather than symbols. They are often too computationally complex to implement in a mobile receiver.

In figure 6.13 types of equalizers, their structures and the algorithm for learning channel environment is given.

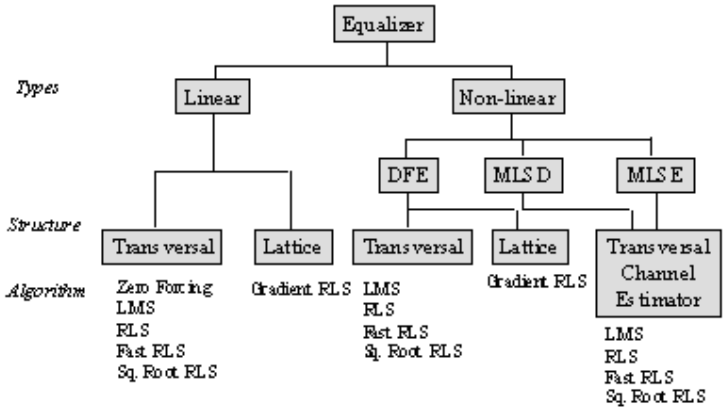

Figure 6.13: Classification of equalizers

Non-linear equalization is important in providing optimum performance for ill-conditioned channels that non-linear techniques require more computation and controls. However, in order the study the gradient descent-based adaptive algorithms' performance, the linear equalizer is more appropriate. Table 6.1 also gives some performance, computation complexity and implementation cost measures of the equalizers.

Table 6.1: Cost, performance and complexity analysis of equalizers

Equalization System	Complexity	Cost	Performance
Linear Transversal Equalizer	Low	Cheap	Suboptimal
Decision Feedback Equalizer	Medium	Average	Suboptimal
Maximum Likelihood Sequenc Estimation	High	High	Optimal
Maximum Likelihood Symbol Detection	High	High	Optimal

6.5 MMSE EQUALIZER

The zero-forcing equalizer, although removes ISI, may not give the best error performance for the communication system because it does not take into account noises in the system. A different equalizer that takes noises into account is the minimum mean square error (MMSE) equalizer. It is based on the mean square error (MSE) criterion. Before going into the mathematics of MMSE we want to clear that the detail derivation for minimizing the MSE and

reaching the Weiner solution can be found from the previous chapter on adaptive filters. Here only the necessary steps are repeated.

Without knowing the values of the information symbols I_k beforehand, we model each symbol I_k as a random variable. Assume that the information sequence $\{I_k\}$ is WSS. We choose a linear equalizer $H_E(z)$ to minimize the MSE between the original information symbols I_k and the output of the equalizer $\hat{I}_k$:

$$MSE = E[e_k{}^2] = E[(I_k - \hat{I}_k)^2] \tag{6.7}$$

Let us employ the FIR filter of order *2L+1* shown in Figure 6.14 as the equalizer. We note that a delay of *L* symbols is incurred at the output of the FIR filter. Then

$$\begin{aligned} MSE &= E[(I_k - \sum_{j=-L}^{L} \tilde{I}_{k-j} h_{E,j})^2] \\ &= E[(I_k - \hat{\mathbf{I}}_k{}^T \mathbf{h}_E)^2] \end{aligned} \tag{6.8}$$

where

$$\hat{\mathbf{I}}_k = [\hat{I}_{k+L}, \ldots, \hat{I}_{k-L}]^T \tag{6.9}$$

$$\mathbf{h}_E = [h_{E,-L}, \ldots, h_{E,L}]^T \tag{6.10}$$

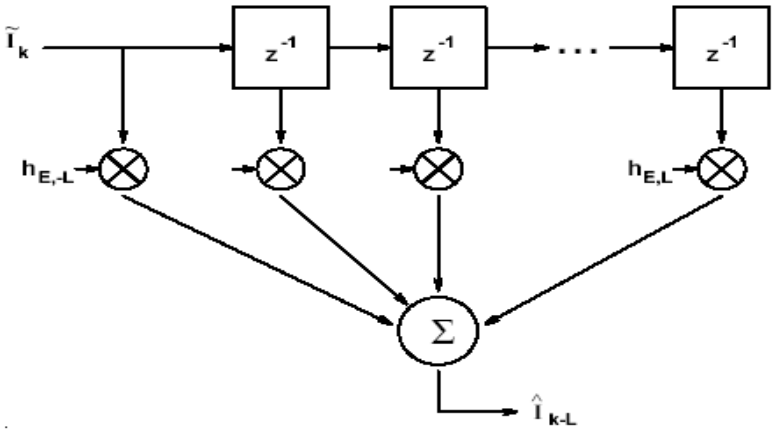

Figure 6.14: FIR Filter as a MMSE Equalizer

We want to minimize MSE by suitable choices of $h_{E,-L}, \ldots, h_{E,L}$. Differentiating with respect to each $h_{E,j}$ and setting the result to zero, we get

$$E[\tilde{\mathbf{I}}_k (I_k - \tilde{\mathbf{I}}_k{}^T \mathbf{h}_E)] = 0 \tag{6.11}$$

Rearranging we get

$$\mathbf{R}\mathbf{h}_E = \mathbf{d} \tag{6.12}$$

where

$$\mathbf{R} = E[\tilde{\mathbf{I}}_k \tilde{\mathbf{I}}_k{}^T] \tag{6.13}$$

$$\mathbf{d} = E[\tilde{I}_k \tilde{\mathbf{I}}_k] \tag{6.14}$$

If $\mathbf{R}$ and $\mathbf{d}$ are available, then the MMSE equalizer can be found by solving the linear matrix equation 6.12. It can be shown that the signal-to-noise ratio at the output of the MMSE equalizer is better than that of the zero-forcing equalizer.

The linear MMSE equalizer can also be found iteratively. First, notice that the MSE is a quadratic function of equalizer filter taps $\mathbf{h}_E$. The gradient of the MSE with respect to $\mathbf{h}_E$ gives the direction to change $\mathbf{h}_E$ for the largest increase of the MSE. In our notation, the gradient is $-2(\mathbf{d} - \mathbf{R}\mathbf{h}_E)$. To decrease the MSE, we can update $\mathbf{h}_E$ in the direction opposite to the gradient. This is the *steepest descent algorithm*: At the k*th* step, the vector $\mathbf{h}_E(k)$ is updated as

$$\mathbf{h}_E(k) = \mathbf{h}_E(k-1) + \mu(\mathbf{d} - \mathbf{R}\mathbf{h}_E(k-1)) \tag{6.15}$$

where μ is a small positive constant that controls the rate of convergence to the optimal solution. Once again we are repeating that all this mathematics is done in the previous chapter.

In many applications, we do not know $\mathbf{R}$ and $\mathbf{d}$ in advance. However, the transmitter can transmit a *training sequence* that is known *a priori* by the receiver. With a training sequence, the receiver can estimate $\mathbf{R}$ and $\mathbf{d}$. Alternatively, with a training sequence, we can replace $\mathbf{R}$ and $\mathbf{d}$ at each step in the steepest descent algorithm by the rough estimates $\tilde{\mathbf{I}}_k \tilde{\mathbf{I}}_k{}^T$ and $\tilde{I}_k \tilde{\mathbf{I}}_k$, respectively. The algorithm becomes:

$$\mathbf{h}_E(k) = \mathbf{h}_E(k-1) + \mu[I_k - \tilde{\mathbf{I}}_k{}^T \mathbf{h}_E(k-1)]\tilde{\mathbf{I}}_k \tag{6.16}$$

This is a stochastic steepest descent algorithm called the *least mean square (LMS) algorithm*.

The beauty of the approach is that the only parameter to be adjusted is the adaptation step size μ. Through an iterative process, explained above, all filter tap weights are adjusted during each sample period in the training sequence.

Eventually, the filter will reach a configuration that minimizes the mean square error between the equalized signal and the stored reference. As might be expected, the choice of μ involves a tradeoff between rapid convergence and residual steady-state error. A too-large setting for μ can result in a system that converges rapidly on start-up, but then chops around the optimal coefficient settings at steady state.

The LMS equalizer can also be shown to have better noise performance than the ZFE. Heuristically, the ZFE calculates coefficients based upon the received samples of one training signal. Since the captured data will always contain some noise, the calculated coefficients will be noisy (noise in / noise out). On the other hand, the LMS algorithm gradually adapts a filter based on many cycles of the training signal. If the noise is zero mean and is averaged over time, its effect will be minimized (noise integrates to 0).

6.6 FRACTIONALLY SPACED EQUALIZERS

In the previous section we have designed the discrete time model of the communication system. In that model we have said that we sample the output of the receive filter $H_R(f)$ at the symbol rate T (for convenience the figure 6.8 is redrawn here as 6.15). But the sampling at the symbol rate my lead to aliasing. Why we will shortly. The equalizer which is operating at the symbol rate or one sample per symbol, as in the figure below, is called *symbol spaced equalizer* or *baud spaced equalizer*.

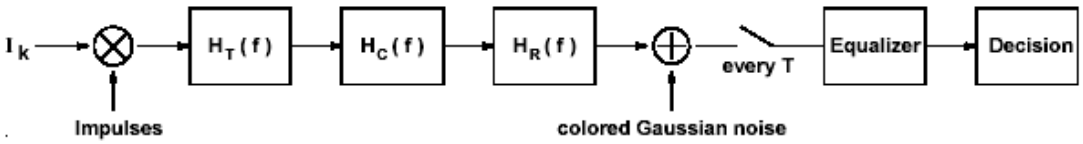

Figure 6.15: Equivalent communication system with colored Gaussian noise

If we are using raised cosine filter for pulse shaping then we should remember that because of the excess bandwidth sampling higher than the symbol rate is required to satisfy the sampling theorem. Actually we can sample at the symbol rate because the raised cosine spectrum has the odd symmetry at the symbol rate. But in order to do that the spectrum of receive filter has to be matched with the spectrum of the received signal. This happens only for the ideal channel but never happens in the practical situation

because the channel distorts the transmitted signal. One solution is that we adjust the receive filter such that it becomes a match filter but the channel is always unknown. Now if we are sampling the output of the receive filter at the symbol rate this means that we are not satisfying the Nyquist sampling theorem (under sampling). This symbol rate sampling leads to the aliasing. Symbol spaced equalizer always operating on the aliased signal and never gives optimum result (see figure 6.16). The answer to this is to sample the output of the receive filter higher than the symbol rate which leads us to another class of equalizers called fractionally spaced equalizers.

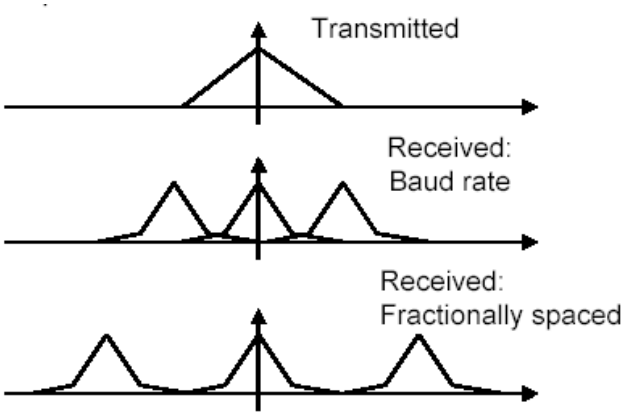

Figure 6.16: Spectrums for transmitted and received signals for SSE and FSE

The aliased spectrum of the signal at the input to the equalizer is given as

$$Y_T(f) = \frac{1}{T}\sum_n X(f - \frac{n}{T}) e^{j2\pi(f-\frac{n}{T})\tau_0} \tag{6.17}$$

where $Y(f)$ is the aliased spectrum, the folding frequency is $F_h = F_S / 2 = 1/2T$ and τ_0 is the delay induced by the channel. The symbol spaced equalizer (SSE) can be represented as the FIR filter

$$H_{SSE}(f) = \sum_{k=-L}^{L} h_{SSE,k} e^{-j2\pi f kT} \tag{6.18}$$

where $2L+1$ is the number of tap weights in the equalizer.

Thus the SSE cannot undo the effects introduced by the channel to the actual spectrum. $X(f)$.So that the FSE is used in which tap spacing is less than the symbol duration T and the sampling of the signal is done at least at the Nyquist rate. Let $\frac{M}{N}T, N > M$, where N and M are integers, ($N=M$ for symbol

spaced equalizers) denote the tap spacing of the FSE, we can see that the transfer function of the equalizer can be expressed as

$$H_{FSE}(f) = \sum_{k=-L}^{L} h_{FSE,k} e^{-j2\pi f kT'} \tag{6.19}$$

where $T' = \frac{M}{N} T$ and the spectrum at the output of the equalizer is given by

$$H_{FSE}(f) Y'(f) = H_{FSE}(f) \sum_{n} X(f - \frac{n}{T'}) e^{j2\pi (f - \frac{n}{T'})\tau_0} \tag{6.20}$$

Since the transmitted pulse is the raised cosine spectrum with roll off factor β the spectrum is bandlimited to $\frac{1+\beta}{2T}$ and hence the sampling must be done atleast at $F_S = \frac{1+\beta}{T}$ the tap spacing is then

$$T' = \frac{M}{N} T = \frac{1}{F_S} = \frac{T}{1+\beta} \tag{6.21}$$

Thus from equations 6.20 and 6.21 it can be concluded that there will be no aliasing in the spectrum and hence

$$H_{FSE}(f) Y'(f) = H_{FSE}(f) X(f) e^{j2\pi (f)\tau_0} \quad , \ |f| \le \frac{F_S}{2} = \frac{1}{2T'} \tag{6.22}$$

Thus from the above equation it is clear that the optimum FSE is equivalent to the case where the receive filter matched to the channel distorted transmitted pulse and the symbol spaced equalizer is used.

In our model the match filter $H_R(f)$ precedes the sampler, as was shown in Figure 6.15. While this may be simple from an analytical viewpoint, there are several practical problems with the use of the matched filter. $H_R(f)$ is a continuous time filter and may be much more difficult to design accurately than an equivalent digital filter and in not applicable to software radio which is an all digital radio. For these reasons, sophisticated data transmission systems often replace the matched filter/sampler/equalizer system of Figure 6.15 with the structure of Figure 6.17. Basically, the sampler and the matched filter have been interchanged with respect to Figure 6.15. Now with this structure we have more flexibility. Because of FSE we can have a digital match or receive filter which is much easy to design [9]. There is another advantage with this. Since the receive filter and FSE filter both are linear they

can be realized as a single combine filter which is actually the case in the practical situation.

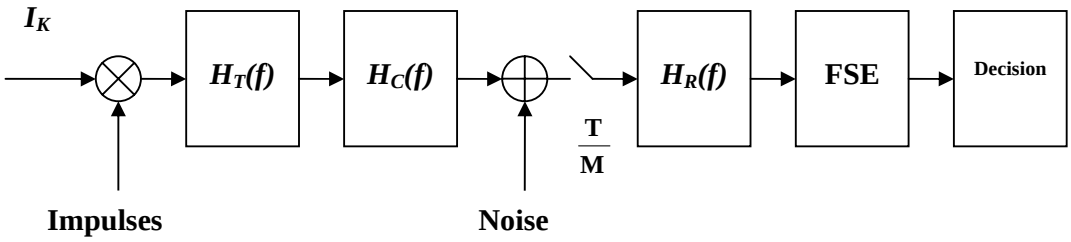

Figure 6.17: Communication system with digital receive filter and FSE

The FSE can also exhibit a significant improvement in sensitivity to sampling-phase errors. When the received signal is sampled greater than twice the highest frequency, then information about the entire signal waveform is retained. Equivalently, the FSE can synthesize, via its transfer characteristic, a phase adjustment (effectively interpolating to the correct phase) so as to correct the timing offset in the sampling device. The symbol-spaced equalizer cannot interpolate to the correct phase, as no interpolation is correctly performed at the symbol rate. Equivalently, information has been lost about the signal by sampling at a speed that is too low in the symbol-spaced equalizer without matched filter. This possible notch is an example of information loss at some frequency; this loss cannot be recovered in symbol-spaced equalizer without a matched filter. In effect, the FSE equalizes before it aliases (aliasing does occur at the output of the equalizer where it decimates by *M* for symbol-by-symbol detection), whereas the symbol-spaced equalizer aliases before it equalizes; the former alternative is often the one of choice in practical system implementation, if the extra memory and computation can be accommodated. Effectively, with an FSE, the sampling device need only be locked to the symbol rate, but can otherwise provide any sampling phase. The phase is tacitly corrected to the optimum phase inside the linear filter implementing the FSE.

It should be noted, however, that the sensitivity to sampling phase is channel dependent. In particular, there is usually significant channel energy near the Nyquist frequency in applications that exhibit a significant improvement of the FSE with respect to the symbol-spaced equalizer. In channels with little

energy near the Nyquist frequency, the FSE is avoided, as it provides little performance gain, and is significantly more complex to implement (more parameters and higher sampling rate).

In practical systems, the four most commonly found values for *M* are 4/3, 2, 3, and 4. The major drawback of the FSE is that to span the same interval in time (when implemented as an FIR filter, as is typical in practice), it requires *M* more coefficients, leading to an increase in memory by a factor of *M*. The FSE outputs may also appear to be computed *M* times more often, also, in real time. However, only $\left(\frac{1}{M}\right)^{th}$ of the output samples need be computed (that is, those at the symbol rate, as that is when we need them to make a decision), so computation is approximately *M* times that of symbol-spaced equalization corresponding to *M* times as many coefficients to span the same time interval, or equivalently, to *M* times as many input samples per symbol. But this disadvantage cab be removed using the multi-channel model of the FSE

6.6.1 Multi-Channel Model for FSE

As explained earlier for a fractionally spaced equalizer (FSE), the tap spacing of the equalizer is a fraction of the baud spacing (in time) or the transmitted symbol period. As the output of the equalizer has the same rate as the input symbol rate, the output of the FSE needs to be calculated once in every symbol period. This means that we have to decimate the output of the FSE by *M* (we have to chose every *M*th sample from the output), where *M* is the upsampling factor. This is shown in figure 6.18.

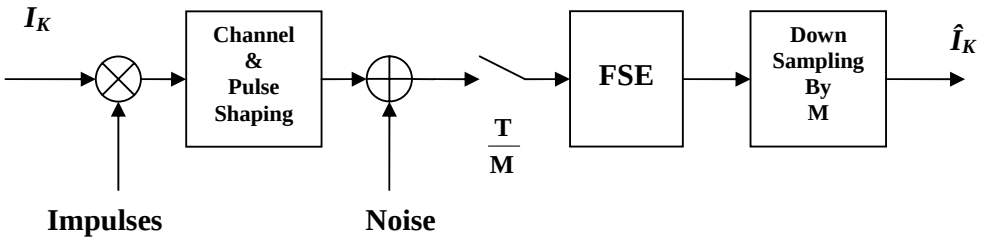

Figure 6.18: Communication system with FSE

In this situation, the FSE can be modeled as a parallel combination of a number of baud spaced equalizers. This parallel combination of baud spaced equalizers is known as the *Multi-Channel Model* of FSE. Actually this approach is the same as the polyphase decomposition of the filter, which we have already discussed in a previous chapter, but in the literature it is named as the *Multi-Channel Model.*

A tap-spacing of T_S/M is assumed for FSE, where M is an integer ($M=2$ for our design). The structure of the FSE in the discrete domain is shown in figure 6.19.

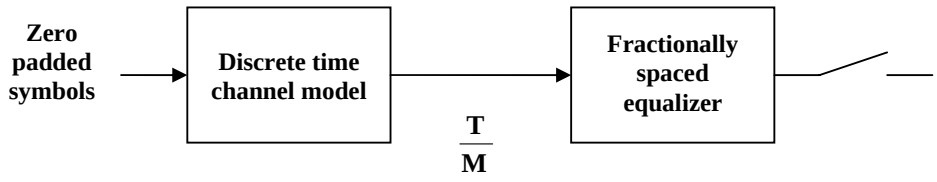

Figure 6.19: A Discrete time channel model with a FSE

Assuming the channel has an impulse-response of length $(L - 1)T_S$, the discrete-time representation of the channel will have a length $(L - 1)M$ because the sampling rate is M/T_S.

Let g be the $(L - 1)M$-vector representing the discrete time channel filter. For simplicity we are using the word channel filter actually it includes all the filters before equalizer **(**see the baseband model in figure 6.12**)**. The input pulse sequence I_k at a rate Ts is zero-padded with M zeros in-between samples to set the sampling rate to M/Ts (already explained in previous chapter) and the output signal from the FSE is sampled at a rate T_S (decimated). Let h be the vector representing the fractionally-spaced equalizer tap coefficients. Because of the presence of these M zeros in-between samples, the convolution $I_k * g * h$ can be easily represented as a mutichannel, single-rate (1/Ts) structure, as shown in figure 6.20.

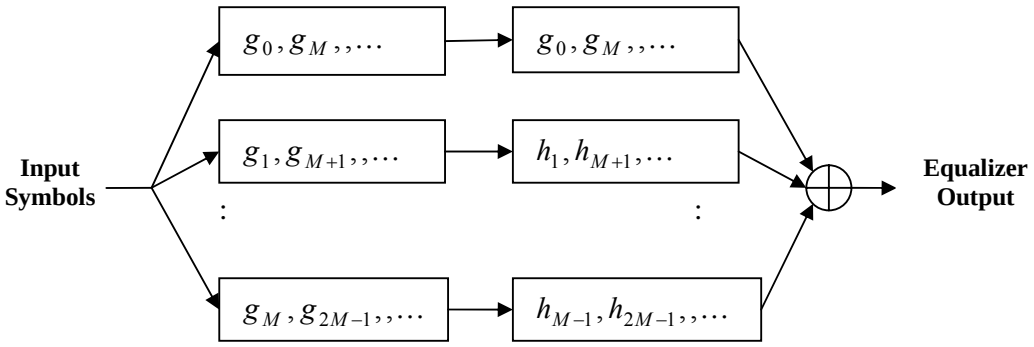

Figure 6.20: A communication model with multi-channel representation of FSE

The channel filter can be subdivided into M sub-channels as:

$$\begin{aligned} &g_0, g_M, g_{2M}, \cdots \\ &g_1, g_{M+1}, g_{2M+1}, \cdots \\ &\vdots \\ &g_{M-1}, g_{2M-1}, g_{3M-1}, \cdots \end{aligned} \tag{6.23}$$

Similarly, the FSE filter can be subdivided into M sub-filters as

$$\begin{aligned} &h_0, h_M, h_{2M}, \cdots \\ &h_1, h_{M+1}, h_{2M+1}, \cdots \\ &\vdots \\ &h_{M-1}, h_{2M-1}, h_{3M-1}, \cdots \end{aligned} \tag{6.24}$$

The total response of the equalizer and the channel is given by

$$y_t(n) = \sum_{m=1}^{M} \{g_n\}_m * \{h_n\}_m \tag{6.25}$$

where $\{g_n\}_m$ and $\{h_n\}_m$ represent the channel and the equalizer impulse response respectively in the *m*th sub-channel. Because of this multi-channel approach the computation complexity of the FSE becomes equal to the symbol spaced equalizer and hence we are only optimizing the performance of equalizer while keeping the computational complexity same.

In our design M=2 so the actual channel and the equalizer response is split into two sub-channels (one even and one odd).

6.7 DECISION FEEDBACK EQUALIZER

We are using the raised cosine filter for pulse shaping and we know that any *n*th received symbol is primarily influenced by the *n*th symbol transmitted; however, there are ISI components contributed by prior and subsequent

transmit symbols. The terms due to prior symbols are termed *postcursor* ISI because the *n*th transmitted symbol affects on symbols following the nth received symbol. The nature of this ISI can be determined by examining the right-hand portion of the raised cosine impulse response. Alternately, the ISI terms due to subsequent transmit symbols exert *precursor* ISI because the *n*th transmit symbol influences received symbols prior to the *n*th. These ISI terms are determined by the shape of the left-hand portion of the raised cosine impulse response (see figure 6.21).

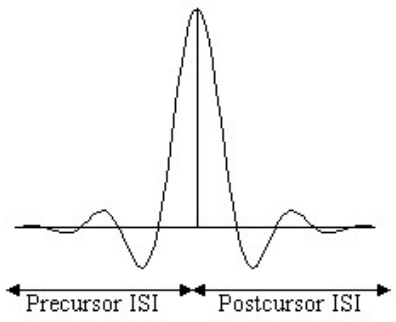

Figure 6.21: Raised cosine filter impulse response

In case of the linear MMSE equalizers (SSE or FSE) what we are doing. We are only removing the ISI from any symbol I_k due to the subsequent symbol following him. That's why the linear equalizers are also called precursor equalizers because they only take care of the ISI contributed by the symbols following them. So there performance is not optimal. To remove the postcursor ISI another equalizer is used which is called decision feedback equalizer (DFE). DFE is based on the principle that once you have determined the value of the current transmitted symbol, you can exactly remove the ISI contribution of that symbol to future received symbols. Recall from the equivalent discrete-time model in Figure 6.12 that

$$\begin{aligned} \tilde{I}_k &= \sum_j I_k g_{k-j} + \bar{n}_k \\ &= I_k g_0 + \sum_{j \neq k} I_k g_{k-j} + \bar{n}_k \end{aligned} \tag{6.26}$$

The current symbol we want to determine is I_k. If we had known the other symbols exactly, an obvious approach to eliminate ISI would be to subtract their effects off, i.e., the equalizer would give

$$\hat{I}_k = \tilde{I}_k - \sum_{j \neq k} I_j g_{k-j} \tag{6.27}$$

In general, we do not know all the symbols that are affecting the reception of the current symbol. However, it is possible to use previously decided symbols (output from the decision device) provided that we have made correct decisions on them. This approach is called *decision feedback equalization*. With decision feedback, we can think of the equalizer to contain two parts: a *feedforward* part and a *feedback* part. Feedforward forward equalizer (FFE) part is a normal FSE which we have already discussed to remove precursor ISI and feedback equalizer (FBE) is an SSE to remove postcursor ISI (see figure 6.22). The FFE *shapes* the channel response (shorten) such that the postcursors can be c*ancelled* by the FBE. DFE is classified as a non linear equalizer because of this decision device.

Suppose that feedforward filter is of order *L* and the feedback filter is of order *W*

$$\hat{I}_k = \sum_{j=-L}^{0} \tilde{I}_{k-j} h_{E,j} + \sum_{j=1}^{W} I'_{k-j} h_{E,j} \tag{6.28}$$

where I'_j are the decided symbols. Again, the filter coefficients $h_{E,j}$ can be found by minimizing the MSE. In general, significant improvement over linear equalizers can be obtained with the decision feedback equalizer.

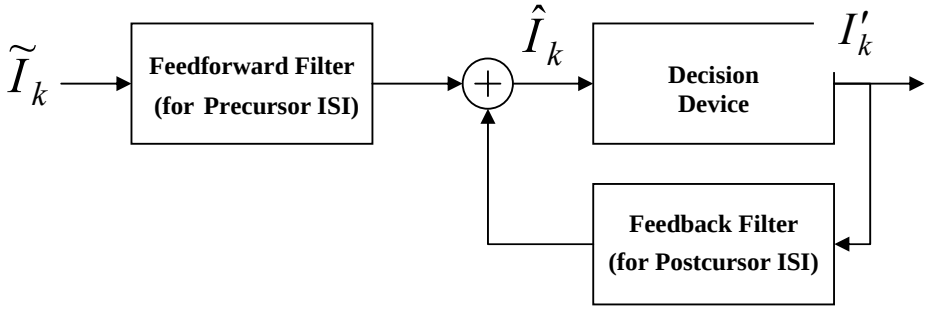

Figure 6.22: A Decision Feedback Equalizer Structure

Consider a DFE with a feedforward filter of order $L+1$ and a feedback filter of order W. Assume perfect decision feedback, i.e., $I'_j = I_{jk}$. Then

$$\hat{I}_k = \mathbf{I_F}^T \mathbf{H_{E,F}} + \mathbf{I_B}^T \mathbf{H_{E,B}} \tag{6.29}$$

where

$$\mathbf{I_F} = \left[\tilde{I}_{k+L}, \tilde{I}_{k+L-1}, \tilde{I}_{k+L-2}, \ldots, \tilde{I}_k\right]^T$$
$$\mathbf{I_B} = \left[I_{k-1}, I_{k-2}, I_{k-3}, \ldots, I_{k-W}\right]^T]$$
$$\mathbf{H_{E,F}} = \left[h_{E,-L}, h_{E,-L+1}, h_{E,-L+2}, \ldots, h_{E,0}\right]^T$$
$$\mathbf{H_{E,B}} = \left[h_{E,1}, h_{E,2}, h_{E,3}, \ldots, h_{E,W}\right]^T$$

Further assume that the data symbols I_k are zero-mean unit-variance identically distributed random variables. We seek the filters $\mathbf{H_{E,F}}$ and $\mathbf{H_{E,B}}$ that minimize the MSE given by

$$E\left[(I_k - \hat{I}_{k)}\right] = E\left[\left(I_k - \mathbf{I_F}^T \mathbf{H_{E,F}} - \mathbf{I_B}^T \mathbf{H_{E,B}}\right)^2\right] \tag{6.30}$$

Differentiating with respect to $\mathbf{H_{E,F}}$ and $\mathbf{H_{E,B}}$, we get

$$E\left[\mathbf{I_F}\left(I_k - \mathbf{I_F}^T \mathbf{H_{E,F}} - \mathbf{I_B}^T \mathbf{H_{E,B}}\right)\right] = 0 \tag{6.31}$$

$$E\left[\mathbf{I_B}\left(I_k - \mathbf{I_F}^T \mathbf{H_{E,F}} - \mathbf{I_B}^T \mathbf{H_{E,B}}\right)\right] = 0 \tag{6.32}$$

Notice that $E[I_k \mathbf{I_B}] = 0$ and $E\left[\mathbf{I_B}\mathbf{I_B}^T\right] = \mathbf{I}_{W\times W}$ i.e., the identity matrix. The equations for optimal $\mathbf{H_{E,F}}$ and $\mathbf{H_{E,B}}$ reduce to

$$E\left[\mathbf{I_F}\mathbf{I_F}^T\right]\mathbf{H_{E,F}} + E\left[\mathbf{I_F}\mathbf{I_B}^T\right]\mathbf{H_{E,B}} = E[I_k \mathbf{I_F}] \tag{6.33}$$

$$E\left[\mathbf{I_B}\mathbf{I_F}^T\right]\mathbf{H_{E,F}} + \mathbf{H_{E,B}} = 0 \tag{6.34}$$

Solving these equations, we have

$$\mathbf{H_{E,F}} = \left(E\left[\mathbf{I_F}\mathbf{I_F}^T\right] - E\left[\mathbf{I_F}\mathbf{I_B}^T\right]E\left[\mathbf{I_B}\mathbf{I_F}^T\right]\right)^{-1} E[I_k \mathbf{I_F}] \tag{6.35}$$

$$\mathbf{H_{E,B}} = -E\left[\mathbf{I_B}\mathbf{I_F}^T\right]\mathbf{H_{E,F}} \tag{6.36}$$

Similar to the case of the MMSE equalizer, we can also solve for the feedforward and feedback filters using the steepest descent approach. If we do not know the expectations of the matrices above *a priori*, we can send a training sequence to facilitate the estimation of them.

For the above discussion we have assumed that all the decision are correct but it is not the case in practice. DFE minimizes the MSE if the wrong decisions are less than 10%. Otherwise a potential problem for the DFE is that any decision errors will cause a corrupted estimate of the postcursor ISI generated by the postcursor equalizer. This is called error propagation (in tracking mode). This problem can be alleviated by using transmitter precoding

also called Tomlinson-Harashima coding. The idea is to move the cancellation of the postcursor ISI to the *transmitter* where no error will occur. The basis of this approach is that the channel and the feedback filter are linear. Thus, we can change the cascade order without affecting the final result.

When the channel response has spectral nulls, the linear equalizer cannot compensate for the distortion. The decision feedback equalizer (DFE) can remedy this problem because it can handle the spectral nulls.

6.8 SIMULATION RESULTS

The two ray channel model is used for simulation with equal gain for both rays. The normalized delay spread is greater than T where T is symbol interval to create frequency selective fading effects. Two samples per symbol are used.

6.8.1 Linear FSE

Various equalizer taps are tried and 11 taps are found to be optimum. Similarly various step sizes are tried and 0.075 is found to be optimum. The learning curve for LMS algorithm is given below.

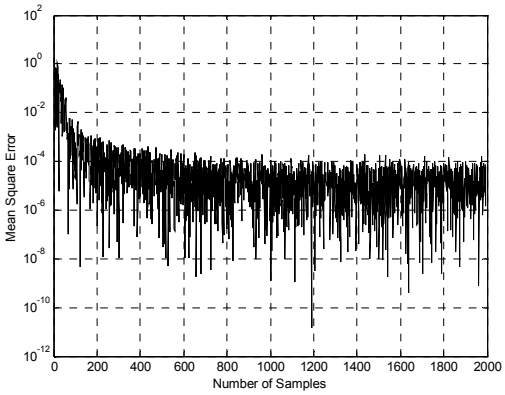

Figure 6.23: Learning curve for LMS algorithm

From the above figure you can see that the algorithm converges after approximately 600 to 800 samples. Eye diagram for faded received signal is

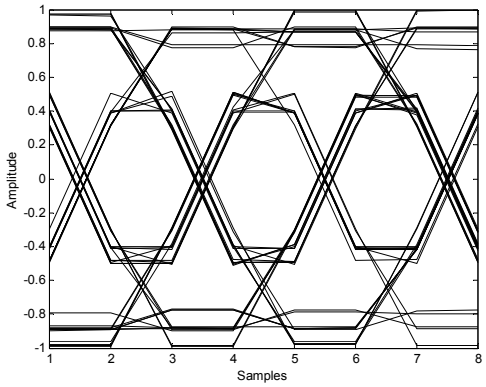

Figure 6.24: Eye diagram for received faded signal

Eye diagram for the equalized signal is given below

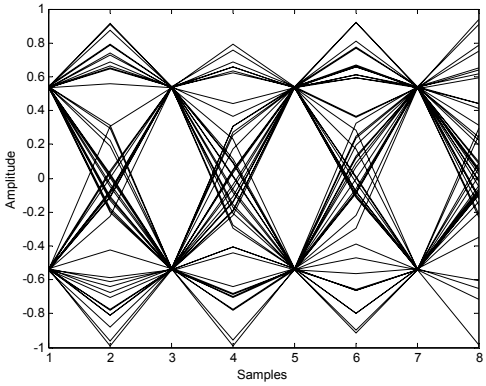

Figure 6.25: Eye diagram for equalized signal

6.8.2 Decision Feedback Equalizer

Optimum length of DFE is found to be equal to 8. 5 taps for feedforward filter and 3 taps for feedback filter. Learning curve for LMS algorithm given below

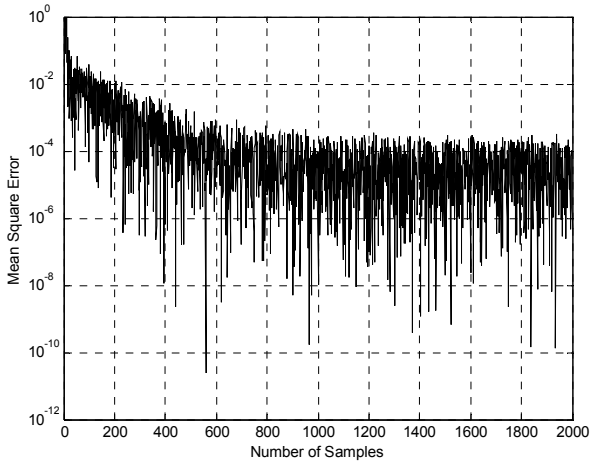

Figure 6.26: Learning curve for LMS algorithm

From the above figure you can see that the algorithm converges after approximately 800 to 1000 samples. Eye diagram for faded received signal is

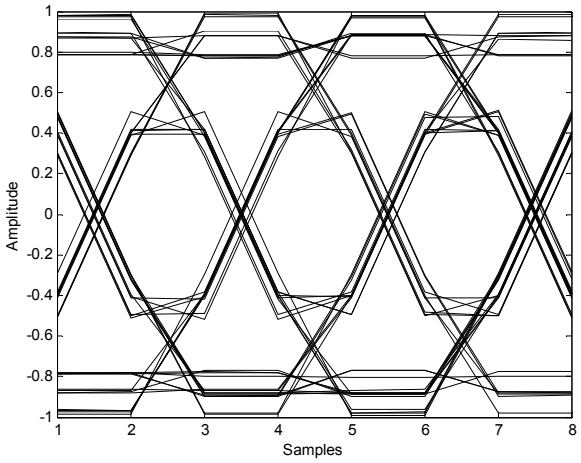

Figure 6.27: Eye diagram for received faded signal

The eye diagram before decision making device is plotted on the next page. Which shows that the ISI is forced to zeros at the input of the decision device.

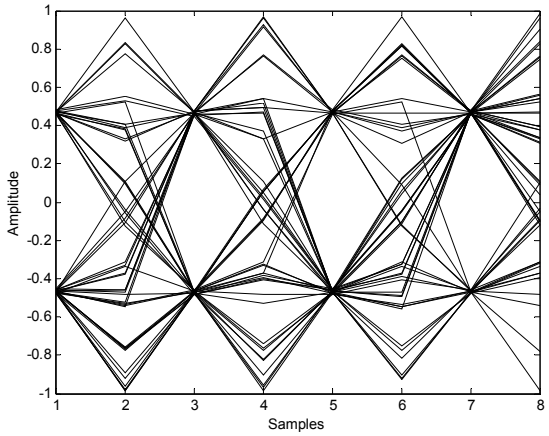

Figure 6.28: Eye diagram for equalized signal

CHAPTER 7

SYNCHRONIZATION

7.1 INTRODUCTION

Synchronization ("syn" meaning "together" and "chronous" meaning "time") is the process of reconstructing this time base. There are two synchronization problems encountered in passband communication systems: Symbol timing recovery and Carrier recovery. Symbol timing recovery enables the receiver to select the proper samples for decision. Carrier recovery is required for the operation of a phase-coherent demodulator.

The carrier frequency of the received signal may be different from that of the nominal value of the transmitter carrier frequency. This discrepancy can be the results of the deviation of the transmitter oscillator from the nominal frequency and, more importantly, the *Doppler Effect* when the transmitter is in motion relative to the receiver. In reality, it takes a finite amount of time for the information-bearing electromagnetic wave to travel from the transmitter to the receiver.

In reality, it takes a finite amount of time for the information-bearing electromagnetic wave to travel from the transmitter to the receiver. This *transmission delay* introduces a mismatch between the symbol timing at the transmitter and that at

the receiver. Recall that we need to sample the output of the matched filter at an exact time to optimize the error performance. We need to know the symbol timing at the receiver (or equivalently, the transmission delay) in order to eliminate the performance degradation. Due to the timing mismatch Implementation of the receiver by digital techniques implies sampling of the signal. In some circumstances, the sampling can be synchronized to the symbol rate of the incoming signal

Phase lock loops (PLLs) are key components of modern communication systems which are used in synchronization [9]. Before going into the discussion of carrier and timing recovery techniques it is necessary to provide a detailed overview of PLL.

7.2 PHASE LOCKED LOOPS

The basic PLL structure is shown in Figure 7.1. The voltage-controlled oscillator (VCO) attempts to produce a signal $v(t)$ that tracks the phase of the input $y(t)$. A phase detector measures the phase error between the input $y(t)$ and the VCO output $v(t)$. The resulting error signal can be filtered to become a control signal that drives the VCO. The basic idea is obvious—if the VCO phase gets ahead of the phase of the input, the control signal should be reduced. If the VCO phase gets behind, the control signal should be increased. As with any feedback system, the parameters must be chosen to ensure stability. The goal in design of the PLL varies with the application.

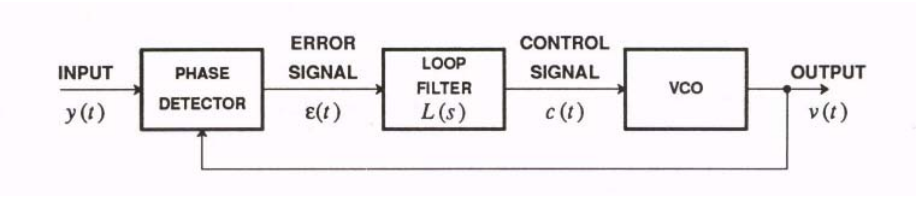

Figure 7.1: Basic structure of a continuous-time PLL.

7.3 IDEAL CONTINUOUS TIME PLL

PLLs are conceptually simple, but they are inherently non-linear systems and their analysis can be difficult. However, with some carefully crafted simplifying assumptions we can develop powerful analytical tools that simplify the analysis.

First assume a particular form for the input

$$y(t) = A_y \cos\left(w_v t + \theta(t)\right) \tag{7.1}$$

where A_y and w_v are constants. The output of the VCO is assumed to have a similar form

$$v(t) = A_v \cos\left(w_v t + \phi(t)\right) \tag{7.2}$$

When $\phi(t)$ is a constant the frequency of the VCO output is w_v, called the *natural or free-running frequency* of the VCO.

7.3.1 Ideal Phase Detector

Assuming forms (7.1) and (7.2) the output of an ideal phase detector is

$$\varepsilon(t) = W\big(\theta(t) - \phi(t)\big) \tag{7-3}$$

where the function $W(.)$, shown in Figure 7.2, reflects the 2π ambiguity in the phase difference. Because of the shape of $W(.)$, this phase detector is called a sawtooth phase detector. We have assumed unity slope for the function $W(.)$, although in practice the phase detector may exhibit some other gain, often written K_P. That gain is easily modeled as part of the loop filter gain, so its explicit inclusion is not necessary. Because of the 2π ambiguities in an ideal phase detector, sudden changes of 2π in $\theta(t)$ or $\phi(t)$ have no effect on the system (they are not detected by the phase detector). Such changes are called clicks, and are usually detrimental.

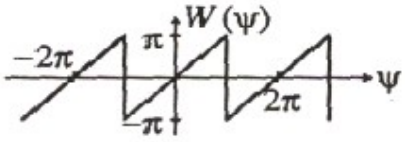

Figure 7.2: An ideal phase detector which can detect phase errors ψ modulo 2π.

7.3.2 Ideal VCO

The ideal VCO, with properties summarized in Figure 7.3, produces the output (7.2), which has instantaneous frequency

$$\frac{d}{dt}\big[w_v t + \phi(t)\big] = w_v + \frac{d\phi(t)}{dt} \tag{7.4}$$

Again, a practical VCO may have gain, often written K_v, that can be modeled as part of the gain of the loop filter. Intuitively, we would like to directly control the instantaneous frequency with the control input $c(t)$. The VCO should therefore be designed so that

$$\frac{d(\phi(t))}{dt} = c(t) \tag{7.5}$$

Taking the Laplace transform of equation (7.5),

$$s\Phi(s) = C(s) = L(s)E(s) \tag{7.6}$$

where $C(s)$ is the Laplace transform of the control signal and $E(s)$ is the Laplace transform of the error signal $\varepsilon(t)$.

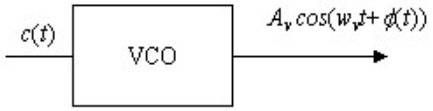

Figure 7.3: An ideal VCO

7.3.3 Phase and Average-Frequency Lock

The ideal PLL is phase locked if

$$\phi(t) = \theta(t) + \phi \tag{7.7}$$

for some constant ϕ.If $\phi = 0$, the PLL is perfectly phase locked. In other words, the VCO output is exactly tracking the phase of the input. It is locked to an average frequency $w_v + K$ if

$$\phi(t) = Kt \tag{7.8}$$

for some constant K. The VCO output frequency is presumably exactly the same as the input average frequency. Intuitively, there must be some limitations on the input phase Q(t) for the PLL to be phase or average-frequency locked because the phase detector output is bounded by $\pm\pi$. To find the limitations, assume a simple form for the phase of the input,

$$\theta(t) = w_0 t + \theta \tag{7.9}$$

In other words, the input $y(t)$ is a sinusoid with frequency $w_v + w_0$ and phase θ, a constant. Assume the PLL is phase locked. In order for it to remain phase locked, the frequency offset w_0 must not exceed a limited range called the lock range or hold-in range of the PLL.

7.3.4 Analysis of the Linearized Dynamics

Phase and average-frequency lock are static concepts. They assume the PLL is in steady state. If we assume that the phase error is small enough for all t

$$,\left|\theta(t) - \phi(t)\right| < \pi \tag{7.10}$$

then the phase detector is operating in its linear range (Figure 7.2)

$$\varepsilon(t) = \theta(t) - \phi(t) \tag{7.11}$$

and the analysis of the dynamics of the PLL is simple. The transfer function from the phase $\theta(t)$ of the input to the phase $\phi(t)$ of the VCO follows by taking the Laplace transform of (7. 11),

$$E(s) = \Theta(s) - \Phi(s) \tag{7.12}$$

and from (7.6),

$$E(s) = \frac{s\Phi(s)}{L(s)} \tag{7.13}$$

Combining these and solving for $\Phi(s)/\Theta(s)$ we get the phase transfer function

$$\frac{\Phi(s)}{\Theta(s)} = \frac{L(s)}{L(s)+s} \tag{7.14}$$

From above transfer function it is clear that the order of PLL (transfer function) is always 1 greater than the order of the loop filter. Evaluating transfer function (7. 14) at s =0, for a zero order loop filter, the PLL has unity gain for dc phase errors. In other words, when the input phase is constant, $\theta(t) = K$, then the output phase is the same constant $\phi(t) = K$, In this case we get perfect phase lock with any loop filter.

The bandwidth of a PLL is loosely defined to be the bandwidth of the transfer function $\Phi(s)/\Theta(s)$. Lowering the bandwidth means increasing the attenuation of high frequency components in the input phase or noise, but for the first order PLL, it also reduces the lock range. It is possible to reduce the bandwidth without reducing the lock range by using a second-order PLL (first order loop filter) [9].

Transfer function for the typical first order PLL is

$$L(s) = K_L \frac{s + K_1}{s} \tag{7.15}$$

This is sometimes called a proportional plus integral loop filter. The closed-loop phase response is

$$\frac{\Phi(s)}{\Theta(s)} = \frac{K_L K_1 + K_L s}{K_L K_1 + K_L s + s^2} \tag{7.16}$$

The PLL with above loop filter has unity gain at dc. It has an integrator in the loop filter. In fact, by convention, the "type" of a PLL is the number of integrators in the loop filter plus one. Its main advantage is that the integrator leads to perfect phase lock even in the face of frequency offset. A disadvantage is that it always exhibits peaking.

7.3.5 Steady-State Response

It is often useful to know precisely the steady-state operating point of a PLL given certain inputs. The steady-state phase error is defined to be

$$\varepsilon_{ss} = \lim_{t\to\infty} \varepsilon(t) \tag{7.17}$$

If the PLL does not achieve perfect phase lock then $\varepsilon_{ss} \neq 0$. If $\varepsilon(t) = 0$ for t < 0 then we can (usually) find ε_{ss} using the final value theorem for Laplace transforms,

$$\varepsilon_{ss} = \lim_{s\to 0} sE(s) \tag{7.18}$$

Laplace transform of $\varepsilon(t)$ in terms of the input phase is,

$$E(s) = \frac{s\Theta(s)}{L(s)+s} \quad , \quad \varepsilon_{ss} = \lim_{s\to 0} \frac{s^2\Theta(s)}{L(s)+s} \tag{7.19}$$

7.4 DISCRETE TIME PLL

In digital communications systems, especially for software radio applications, analog continuous-time PLLs like those discussed in section 7.3 are rare. So we have to have the discrete time equivalent for the continuous time PLLs [9].

A typical all digital PLL is shown in figure 7.4**.** Assumptions about the form of the input signal and the output of the digital VCO which is also called numerically controlled oscillator (NCO) are analogous to that of continuous time PLL.

7.4.1 Phase Error Detector

The PED is a discrete time version of the continuous PED discussed previously.

$$\varepsilon_k = W(\theta_k - \phi_k) \tag{7.20}$$

Where $W(.)$ is shown in figure 7.2.

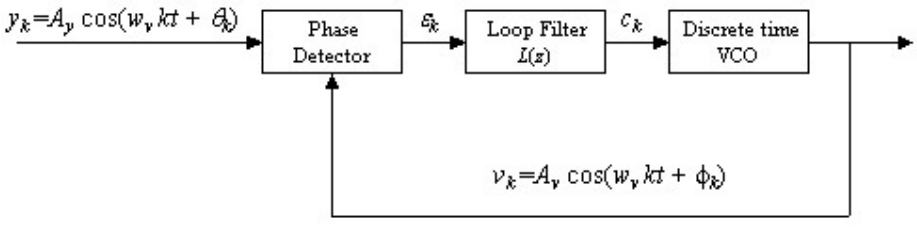

Figure 7.4: A typical discrete time PLL

7.4.2 Discrete Time VCO

Discrete time VCO also called numerically controlled oscillator (NCO). The NCO, although analogous to VCO, is not quite as obvious. The phase ϕ_k satisfies the difference equation

$$\phi_{k+1} - \phi_k = c_k \tag{7.21}$$

Using this output of VCO can be written as

$$\begin{aligned} v_{k+1} &= A_v \cos(w_v(k+1)T + \phi_{k+1}) \\ &= A_v \cos(w_v kT + \phi_k + w_v T + c_k) \end{aligned} \tag{7.22}$$

This leads to the structure in figure 7.5. Taking z-transform we get

$$\Phi(z) = \frac{1}{z-1} C(z) = \frac{L(z)}{z-1} E(z) \tag{7.23}$$

where $L(z)$ is the loop filter transfer function and $E(z)$ is the z-transform of the error ε_k.

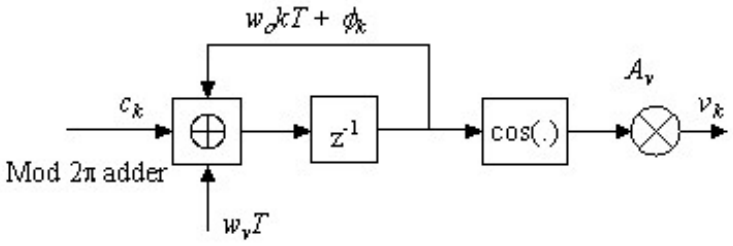

Figure 7.5: A typical NCO implementation

In the above figure you can see that NCO consists of an accumulator and a cosine computation, along with some of constant and multiplications. The modulo two adders reflect the fact the numbers being added are angles in radians. The cosine function can be implemented using a look up table or Taylor series expansion. Look up table is the most common method.

7.4.3 Analysis of the Dynamics

As before, to analyze the dynamics we assume that the phase error is small enough that the phase detector is linear. The phase detector output is

$$\varepsilon_k = \theta_k - \phi_k \tag{7.24}$$

or taking Z transforms

$$E(z) = \Theta(z) - \Phi(z) \tag{7.25}$$

Combining (15.43) with (15.39) we get the phase transfer function of the PLL,

$$\frac{\Phi(z)}{\Theta(z)} = \frac{L(z)}{L(z) + z - 1} \tag{7.26}$$

By evaluating this at z = 1 we see that just as with the continuous-time PLL, discrete-time PLLs have unity gain to dc phase inputs.

7.4.4 Steady-State Error

Just as with continuous-time PLLs, the steady-state error is

$$\varepsilon_{ss} = \lim_{k\to\infty} \varepsilon_k \tag{7.27}$$

If $\varepsilon_k = 0$ for k < 0 we can use the final value theorem for z-transform to write

$$\varepsilon_{ss} = \lim_{z\to 1} (z-1)E(z) \tag{7.28}$$

Combining (7.25) and (7.26) we get an expression for $E(z)$,

$$E(z) = \frac{\Theta(z)(z-1)}{L(z)+z-1} \tag{7.29}$$

The "type" of a discrete-time PLL is defined to be one plus the number of poles at z=1. First order PLL (with zero order loop filter) can track a phase step with a zero steady state error. But it is unable to track a frequency step or phase ramp with a zero steady state error. In fact it does track the phase ramp but with a constant steady state error. Second order PLL (with a first order loop filter) can track the both phase step and phase ramp with a zero steady state error. But it unable to track the frequency ramp, which may result if the transmitter and receiver are moving with constant acceleration. But the frequency ramp very rarely occurs in practical conditions. There is another advantage of second order PLLs that they are unconditionally stable and they are most of the time used for synchronization. Third order PLL (with a second order loop filter) can track the phase step, phase ramp and frequency ramp with a zero steady state error. But they are not unconditionally stable and they are never used in synchronization. They have some applications in GPS.

7.4.5 Complex Phase Error Detectors

We are using the QPSK modulation technique. The input to the phase error detector is a complex QPSK signal. So we must have a complex phase error detector. A simple phase detector for complex signals is shown in Figure 7.6. For small phase errors, the phase detector is approximately linear,

$$\varepsilon(t) \approx A_v A_y \left[\theta(t) - \phi(t)\right] \tag{7.30}$$

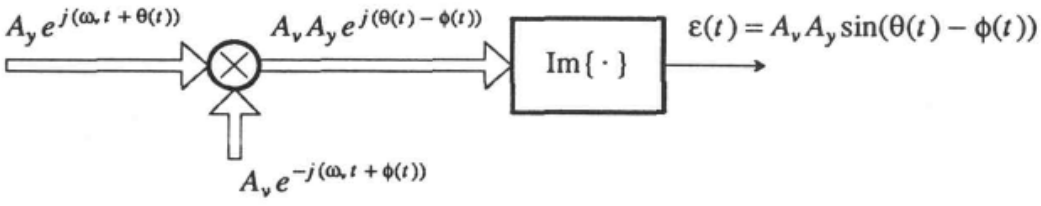

Figure 7.6: A simple phase error detector for complex signals

But we are having the discrete time data. We have to modify the above PED by just replacing the *t* by *k*.

It concludes our discussion about the PLL. Now we will discuss phase recovery.

7.5 PHASE RECOVERY

As described in chapter 4 we are doing coherent detection of QPSK symbols and for coherent detection the receiver and transmitter must be locked in phase. One solution is to use PLL before the timing recovery (discussed in the next section). But it is slightly difficult because we have more than one sample per symbol. So we are putting it after the timing recovery because our timing recovery loop is independent of phase errors. If there is any mismatch in phases of transmitter and receiver carriers then it will rotate the constellation according to the amount of mismatch. Our PLL is operating at baseband complex QPSK signal. We are employing all digital, non data aided and feedback phase recovery. The phase recovery loop recovery loop is given in figure 7.7.

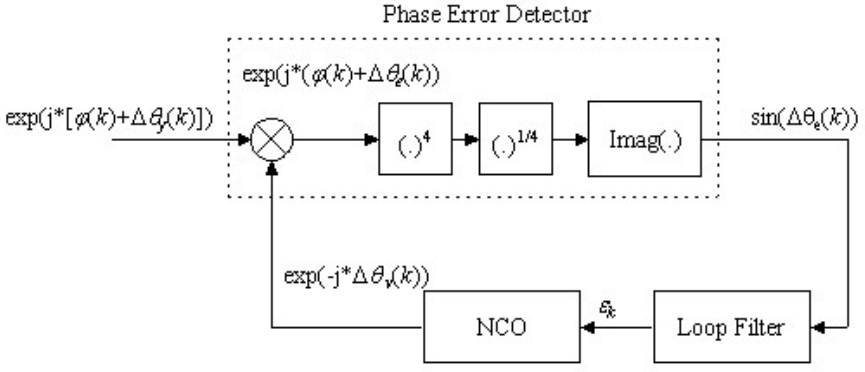

Figure 7.7: NDA Phase recovery loop for one sample per symbol

The above phase recovery method is called raise to power *M* phase recovery. The BPSK signal at the baseband is just ±1(either cos(0) or cos(π)). By taking power of 2

for BPSK signal the modulation is removed. Similarly for QPSK which has 4 phases taking power of 4 the modulation is removed.

The incoming signal has phase $\phi(k)+\Delta\theta_y(k)$, where $\phi(k)$ is the QPSK modulation phase at time instant *k* and given by the relation

$$\phi_i = \frac{\pi}{4} i \qquad where\ i = \pm 1, \pm 3 \tag{7.31}$$

and $\Delta\theta_y(k)$ is the phase distortion. Before discussing each component individually one important thing we want to mention. The values of VCO and PED constants are considered in calculating loop filter's constants (explained shortly). So we will assume them 1 from now on.

7.5.1 Phase Error Detector

The PED operation is to multiply the incoming signal with the NCO output and then take raise to power 4 to remove modulation. The signal at the output of PED is the imaginary part of raised to power 4 error signal. The imaginary part is chosen because it is proportional and odd function of error signal.

Let's assume that the input signal to the PED at any time k is

$$y(k) = \exp(j \times [\phi(k) + \Delta\theta_y(k)]) \tag{7.32}$$

the phase $\theta_y(k)$ of the signal *y*(*k*) is

$$\theta_y(k) = \phi(k) + \Delta\theta_y(k) \tag{7.33}$$

the phase $\theta_y(k)$ is added in the NCO phase $\theta_v(k) = -\Delta\theta_v$ and the output

$$\theta_e(k) = \theta_y(k) - \theta_v(k) \tag{7.34}$$

is the phase difference between the two. By taking raise to power 4

$$\begin{aligned} z(k) &= \left[\exp(j \times \theta_e(k))\right]^4 \\ &= \left[\exp j \times \left(\Delta\theta_y(k) - \Delta\theta_v(k)\right)\right]^4 \end{aligned} \tag{7.35}$$

You can see that the modulation is removed but the error is also becomes 4 times. Taking power ¼ will give us the error $\Delta\theta_y(k) - \Delta\theta_v(k)$ only.

$$r(k) = \left[z(k)\right]^{1/4} \tag{7.36}$$

Now next step is to take imaginary part of the output because it proportional part and also sin is odd function of input. The input signal to the loop filter is

$$e(k) = \Im(r(k)) \tag{7.37}$$

The open loop characteristic curve for PED which is also called S-curve is shown in figure below [13]. This is the same as shown in figure 7.2.

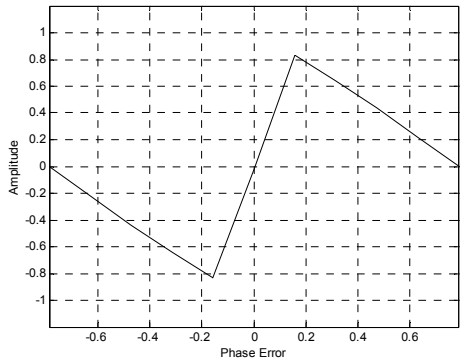

Figure 7.8: Characteristic S-Curve for PED from pi/4 to –pi/4

This is from [–pi/4,pi/4] and it shows that the PED will track the phase variations. It has zero value at the center when phase error is zero. Actually PLL will start locking at any point and then recursively take it to the origin where error is zero. This S-curve will guarantee that the in the close loop the PLL will lock the phase.

7.5.2 NCO

Output signal from the loop filter ε_k is the input to the NCO. The NCO has the same operation as explained above. The output phase of NCO is

$$\theta_v(k+1) = \Delta\theta_v(k) + K(\varepsilon(k)) \tag{7.38}$$

The loop is said to be phase locked if the difference between $\Delta\theta_v(k)$ and $\Delta\theta_y$ is zero.

7.5.3 Loop Filter

As explained before the order of the PLL is always one greater than the order of the loop filter because the NCO is also an integrator. The bandwidth of the loop filter is actually the bandwidth of the PLL. So in deciding the loop bandwidth one must take care of the bandwidth of the error. For example the max frequency offset between the transmitter and receiver in case of carrier recovery. Here is the block diagram of most commonly used first order loop filter.

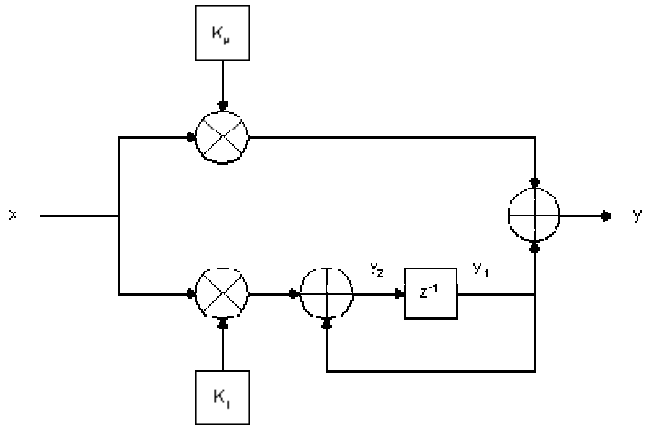

Figure 7.9: First order digital loop filter

Let us examine this block diagram. The phase detector output, x, is multiplied by the proportional gain constant K_p in the upper arm. In the lower arm, the phase detector output is first multiplied by K_i, the integral gain constant. The result of this multiplication is fed into an integrator comprising an adder and a register (unit delay). The final output y is the sum of the product of the proportional gain constant K_p and the phase error computed in the upper arm, and the output of the integrator in the lower arm.

This loop filter could be implemented in software using the following equation:

$$y(n) = K_p * x(n) + K_i * x(n-1) + y(n-1) - K_p * x(n-1) \quad (7.39)$$

The phase detector output is computed and the filter output updated every T_s seconds, where T_s is the sampling interval. K_p and K_i can be calculated using the relation

$$K_p = \frac{4B_L T_s}{1+1/4\zeta^2} \quad (7.40)$$

$$K_i = \left(\frac{2B_L T_s}{\zeta+1/4\zeta}\right)^2 \quad (7.41)$$

where B_L is the single sided loop bandwidth and T_s is the sampling time. Typical values for B_L are $10<B_L<100$ Hz. $B_L T_s$ is called normalized loop bandwidth.

We have explained the phase recovery in detail. Now we will discuss the timing recovery.

7.6 TIMING RECOVERY

Symbol Timing Recovery (STR) or Clock Synchronization is the process of recovering the optimum sampling time that corresponds to the maximum opening of

the eye diagram. This process is often overlooked but it is in fact the most critical in the design of digital communication systems: its failure has devastating effects in the receiver data. The inherent problem of clock synchronization is that sampling clock of the receiver is not synchronized to the strobes of the transmitter.

The digital information embedded in the transmitted signal is recovered at the receiver by means of a decision device. This decision device operates on samples of the noisy signal $y(t;\varepsilon)$ taken at symbol rate $1/T$ at the receive filter output, which is given by

$$y(t;\varepsilon) \equiv \sum_m a_m g(t - mT - \varepsilon T) + n(t) \tag{7.42}$$

In equation above $\{a_m\}$ is a sequence of zero-mean data symbols (complex QPSK symbols). g*(t)* is the baseband pulse at the receive filter output, εT is an unknown fractional time delay $\left(-1/2 \le \varepsilon \le 1/2\right)$ and *n(t)* represents zero-mean additive noise. For maximum noise immunity, the samples upon which the receiver's decision is based should be taken at the instants of maximum eye" opening. As the decision instants are a priori unknown (because of the unknown delay εT *)* the receiver must contain a device which makes an estimate of the normalized delay. Such a device is called a *clock synchronizer* or *symbol synchronizer.* The timing estimate is used to bring the sampling clock, which activates the sampler at the receive filter output, in close synchronism with the received PAM signal. This is achieved by adjusting the phase of this sampling clock according to the value of the estimate.

The received noisy signal contains no periodic components, because the channel symbols $\{a_m\}$ have *zero* mean. Therefore, an ordinary PLL operating on the filtered received signal $y(t;\varepsilon)$ cannot be used to generate a clock signal which is in synchronism with the received QPSK signal. Let us illustrate this fact by considering a PLL with multiplying timing error detector: the local reference signal $r(t;\varepsilon')$ given by

$$r(t;\varepsilon') = \sqrt{2}K_r \sin\left(\frac{2\pi}{T}(t - \varepsilon' T)\right) \tag{7.43}$$

and is multiplied with the noisy QPSK signal $y(t;\varepsilon)$ as shown in Figure 7.10. Taking into account equation (i), the timing error detector output signal equals

$$x(t;\varepsilon,\varepsilon') = \left[\sum_m a_m g(t - mT - \varepsilon T) + n(t)\right]\sqrt{2}K_r \sin\left(\frac{2\pi}{T}(t - \varepsilon' T)\right) \tag{7.44}$$

For any values of ε and ε' .the statistical average of the timing error detector output is identically zero, because the channel symbols $\{a_m\}$and the additive noise *n(t)* have zero mean. As the average timing error detector output is zero irrespective of ε and ε'there is no deterministic force that makes the PLL lock onto the received signal [13].

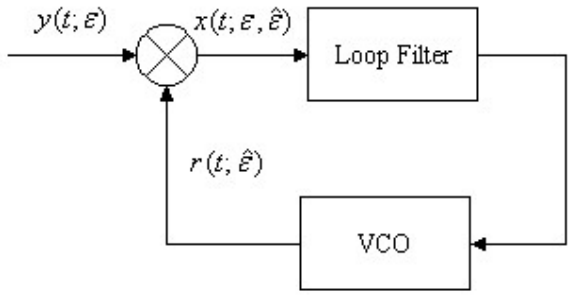

Figure 7.10: Ordinary PLL operating on baseband QPSK signal

7.6.1 Categorization of Timing Synchronizers

From the operating principle point of view, two categories of synchronizers are distinguished i.e., *error-tracking* (or feedback, or closed Loop} synchronizers and *feedforward* (or open loop) synchronizers.

A general error-tracking synchronizer is shown in figure 7.11. The noisy baseband QPSK signal $y(t;\varepsilon)$ and a locally generated reference signal $r(t;\varepsilon')$ are "compared" by means of a *timing error detector,* whose output gives an indication of the magnitude and the sign of the *timing error* $e = \varepsilon - \varepsilon'$. The filtered timing error detector output signal adjusts the timing estimate ε' in order to reduce the timing error *e.* The timing estimate ε' is the normalized delay of the reference signal $r(t;\varepsilon')$ which activates the sampler operating on $y(t;\varepsilon)$. Hence, error-tracking synchronizers use the principle of the PLL to extract a sampling clock which is in close synchronism with the received baseband QPSK signal.

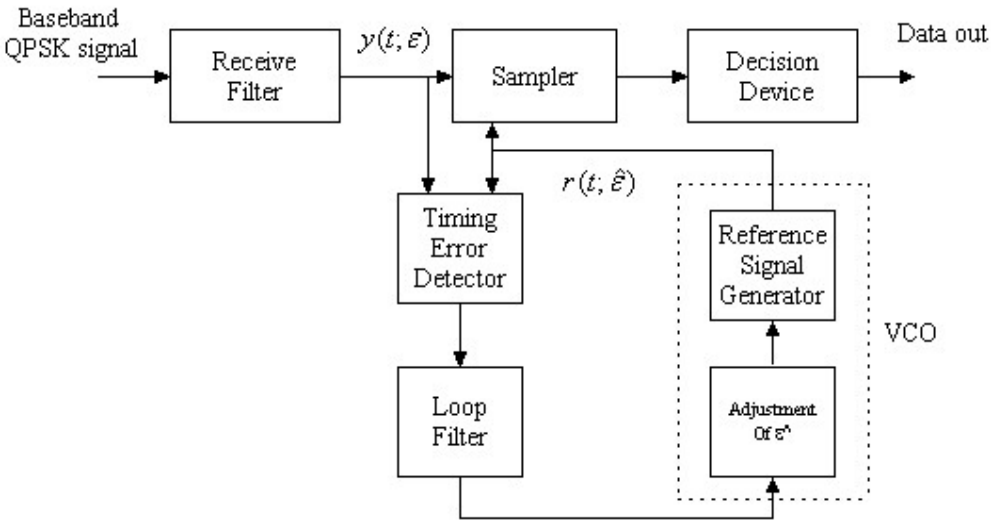

Figure 7.11: General Error Tracking Synchronizer

Figure 7.12 below shows a general feed forward synchronizer. The noisy baseband QPSK receive signal $y(t;\varepsilon)$ enters a *timing detector,* which "measures" the instantaneous value of ε (or a function thereof). The noisy measurements at the timing detector output are averaged to yield the timing estimate ε' (or a function thereof).

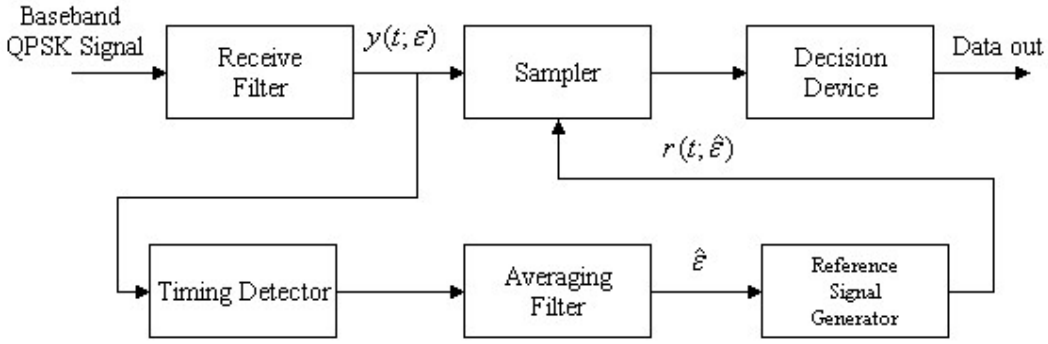

Figure 7.12: General Feedforward Synchronizer

Besides the above categorization into error-tracking and feed forward synchronizers, other categorizations can be made:

- When a synchronizer makes use of the receiver's decisions about the transmitted data symbols for producing a timing estimate, the synchronizer is said to be *decision-directed;* otherwise, it is *non-data-aided.*
- The synchronizer can operate in continuous time or in discrete time. Discrete time synchronizers use samples of the QPSK baseband signal $y(t;\varepsilon)$, and are therefore well-suited for digital implementation [13].

7.7 TIMING RECOVERY USING INTERPOLATION

As described above all digital synchronizer can be obtained by replacing the sample by the interpolator. Now the incoming signal $y(t;\varepsilon)$ can be a discrete time signal. We have designed all digital, non data aided and feedback synchronizer which is described in figure 7.13.

The loop shown in figure 7.13 is an asynchronous, non-data aided, all-digital symbol timing recovery proposed by Gardner .It consists of an Interpolator, Timing Error Detector (TED), Digital Loop Filter, and an accumulator (NCO). The all-digital symbol timing recovery is a feedback timing error synchronizer that can be characterized as a Phase Locked Loop (PLL). The loop parameters are designed based on the linearized model of the PLL. We will evaluate each block one by one.

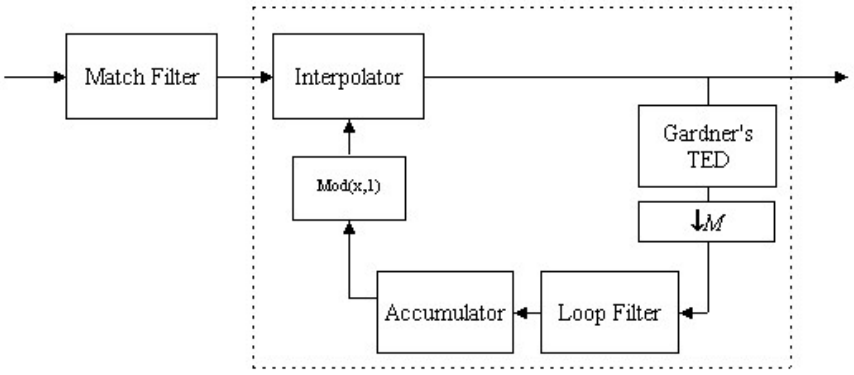

Figure 7.13: All digital symbol timing recovery

7.7.1 Interpolator

The incoming signal is discrete time signal. The interpolator computes the intermediate values between the adjacent signal samples. There are many interpolators which are found in literature. But we have selected cubic Lagrange interpolator for our loop [19]. In numerical mathematics the task of interpolation for our purpose can be stated as follows.

Given the function $x(t)$ defined for $t_{-(N-1)},.....t_0,.....,t_N$, find a polynomial $P(t)$ of degree (2N-1) which assumes the given values $x(t_n)$

$$P(t_n) = x(t_n) \qquad (n = -(N-1),....,-1,0,1,....N) \qquad (7.44)$$

In general, the points t_n do not need to be equidistant, nor does the number of points have to be an even number 2*N*. There exists a Lagrange polynomial of a degree 2*N*-1:

$$P(t) = \sum_{n=-(N-1)}^{N} \lambda_n \left[(t - t_{-(N-1)}) \ldots\ldots (t - t_{n-1})(t - t_{n+1}) \ldots (t - t_N) \right] x(t_n) \tag{7.45}$$

with

$$\lambda_n = \frac{1}{(t_n - t_{-(N-1)}) \ldots\ldots (t_n - t_{n-1})(t_n - t_{n+1}) \ldots (t_n - t_N)} \tag{7.46}$$

Using the definition of (7.45) it is verified that for every t_n, we have $P(t_n) = x(t_n)$ as required. The polynomial $P(t)$ is linear combination of values $x(t_n)$

$$P(t) = \sum_{n=-(N-1)}^{N} q_n(t) x(t_n) \tag{7.47}$$

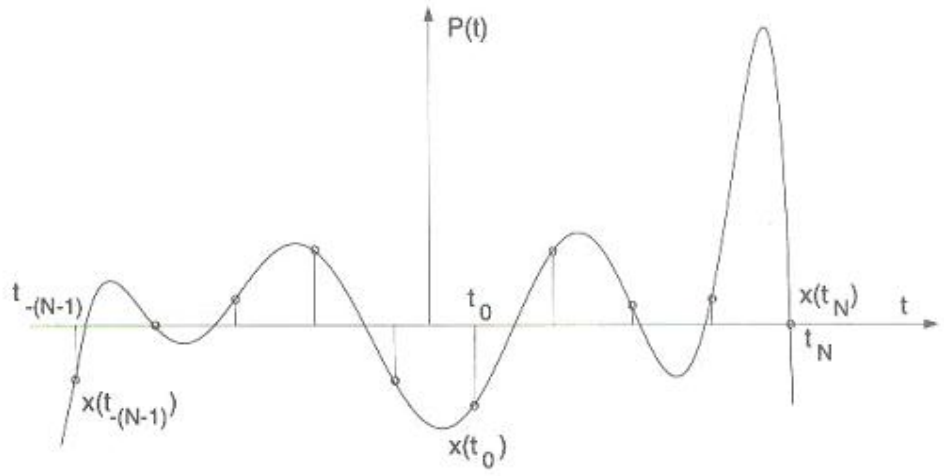

Figure 7.14: Lagrange Interpolation

with so called Lagrange coefficients.

$$q_n(t) = \lambda_n \left[(t - t_{-(N-1)}) \ldots\ldots (t - t_{n-1})(t - t_{n+1}) \ldots (t - t_N) \right] \tag{7.48}$$

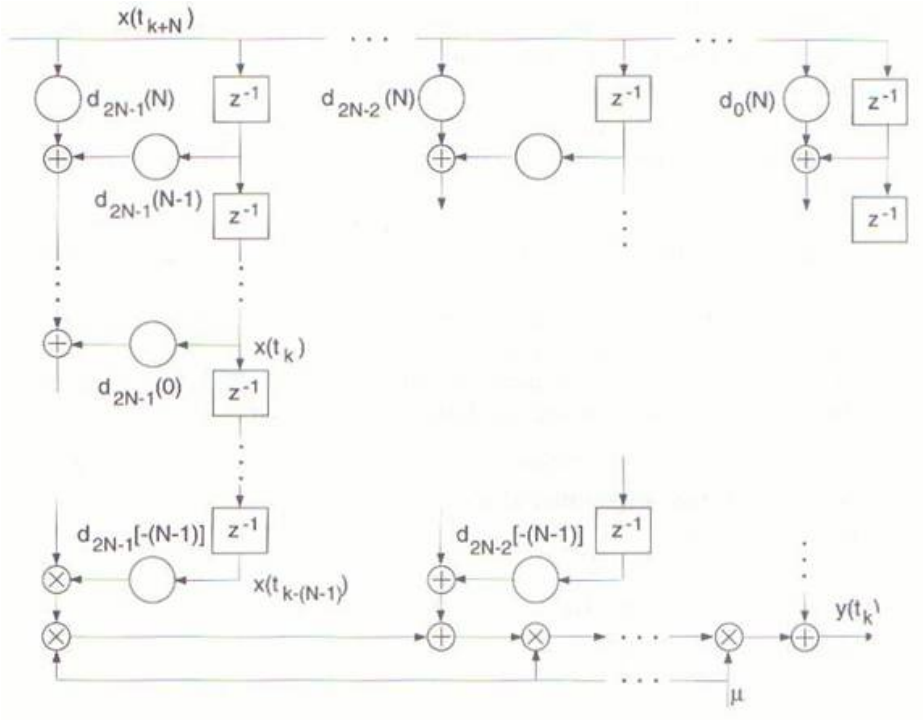

Figure 7.15: Farrow structure of Lagrange interpolation

Since FIR filter also computes a linear combination of sample values $x(kTs)$ it is necessary to point out the differences between the two approaches. As we are only interested in the interpolated values in the central interval $0 \le t \le 1$, we set $t = \mu$ in the definition of the Lagrange coefficients $q_n(t)$. Every $q_n(\mu)$ is a polynomial in μ which can be written as

$$q_n(\mu) = \sum_{m=0}^{2N-1} d_m(n)\mu^m \tag{7.49}$$

Inserting into equation (7.47), we obtain

$$P(\mu) = \sum_{n=-(N-1)}^{N} \left[\sum_{m=0}^{2N-1} d_m(n)\mu^m \right] x(t_n) \tag{7.50}$$

Thus from equation (9.50) we learn that $P(\mu)$ can be computed as the output of a Farrow structure. The main advantage of Lagrange interpolation is that the computations are done in real time which is much better then the classical approach in which the match filter is implemented using polyphase structures for interpolation. We have used cubic Lagrange interpolator. Using equation (7.49), we get

$$q_{-1}(\mu) = -\frac{1}{6}\mu^3 + \frac{1}{2}\mu^2 - \frac{1}{3}\mu$$

$$q_0(\mu) = \frac{1}{2}\mu^3 - \mu^2 - \frac{\mu}{2} + 1$$

(7.51)

$$q_1(\mu) = -\frac{1}{2}\mu^3 + \frac{1}{2}\mu^2 + \mu$$

$$q_2(\mu) = \frac{1}{6}\mu^3 - \frac{1}{6}\mu$$

The implementation of cubic Lagrange interpolator is shown in figure 7.16

7.7.2 Gardner Timing Error Detector

The Gardner algorithm has seen widespread use in many practical timing recovery loop implementations. The algorithm uses two samples per symbol and has the advantage of being insensitive to carrier offsets. This is the reason for selecting this algorithm . The timing recovery loop can lock first, therefore simplifying the task of carrier recovery.

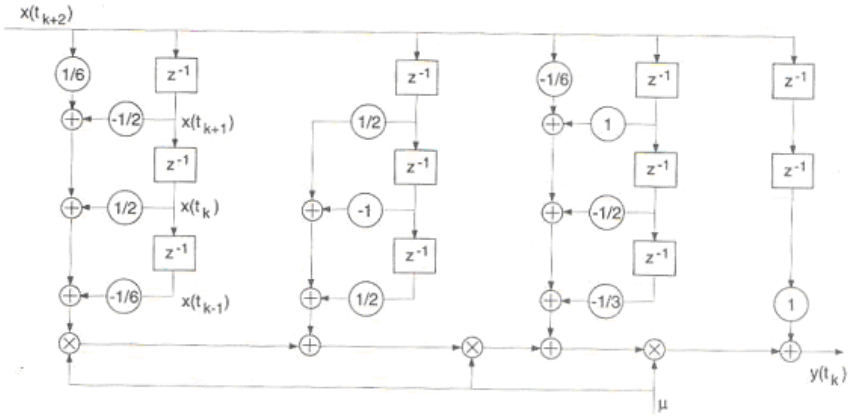

Figure 7.16: Farrow structure of a Cubic Lagrange Interpolator

The error for the Gardner algorithm is computed using the following equation:

$$e(n) = I_{n-1}(I_n - I_{n-2}) + Q_{n-1}(Q_n - Q_{n-2}) \qquad (7.52)$$

where the spacing between n and n-2 is T seconds, and the spacing between n and n-1 is $T/2$ seconds and T is the symbol time. Note that the Gardner error is most useful on symbol transitions (when the symbol goes from positive to negative or vice-versa). The Gardner error is relatively small when the current and previous symbols have the same polarity. That's why it is also called zero crossing detector. It is also shown that Gardner TED is very sensitive to self noise which is proportional to roll off factor for

the raised cosine filter. Greater the roll off factor lesser will be the self noise and better will be the performance.

A simulation was run for determination of open loop S-curve and the curve is found to be sinusoidal. This will show that the timing errors will be tracked.

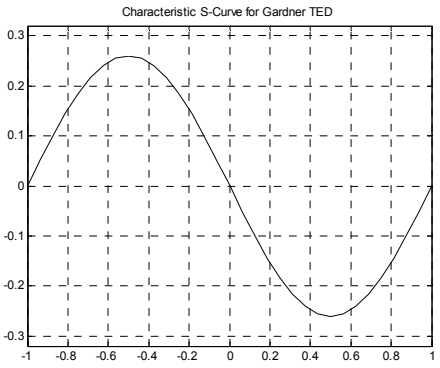

Figure 7.17: Characteristic S-curve for Gardner's TED

7.7.3 Decimator and Loop Filter

Gardner algorithm requires two samples per symbol. The output is required only at the symbol intervals. So after the timing error detector we have to have a decimator so that the output to the loop filter is at symbol interval. The loop filter is the same as described in the phase recovery.

7.7.4 Accumulator and Modulo 1 Adder

The accumulator or NCO is again has the same roll of integrator. As described earlier we have normalized the delay by sampling time. Now we are having the delay in the range of [-1,1]. The output of the integrator $x(k+1)$at any time $k+1$ is

$$x(k+1) = x(k) + y(k) \tag{7.53}$$

Where $y(k)$ is the output of the loop filter.

The modulo 1 adder is used to prevent the accumulator output from increasing 1 which is max delay. If the delay is greater than 1 then the mod operation is used is bring it down at also the basepoint of interpolator is changed by 1 [13].

This concludes our discussion about the timing recovery.

7.8 SIMULATION RESULTS

7.8.1 Phase Recovery

A simulation was run for phase recovery. The phase plus frequency errors are introduced. The constellation for received QPSK symbols is given below

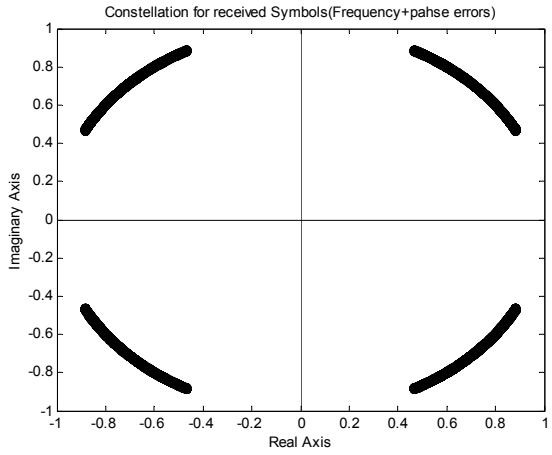

Figure 7.18: Constellation for received QPSK symbols

At the output of phase recovery PLL the constellation is given below. You can see that the phase errors are removed.

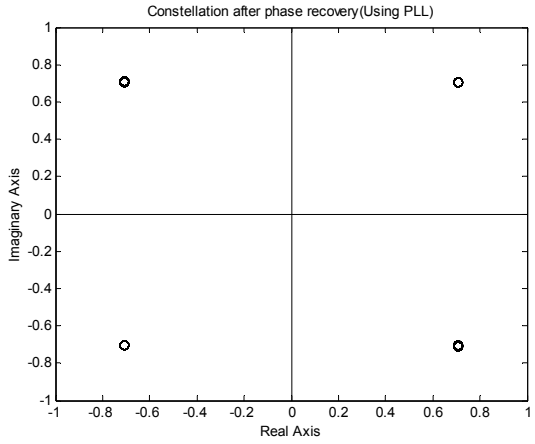

Figure 7.19: Constellation after phase recovery

The output error signal from the loop filter is shown below. The steady state value of the error is zero which sows that the phase errors are removed.

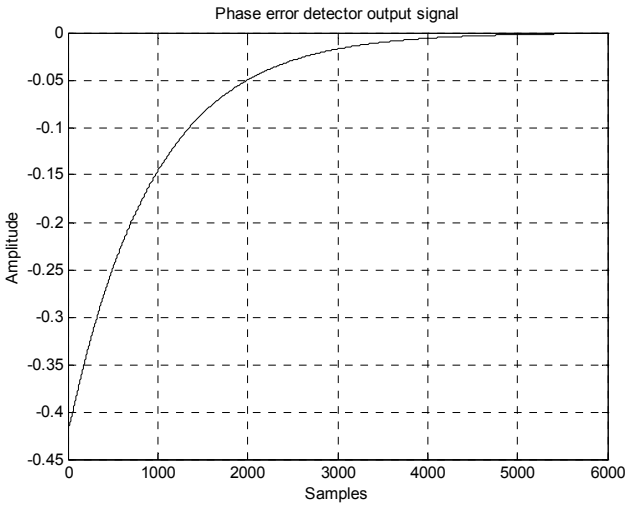

Figure 7.20: Output signal from the loop filter

From the above figure it can be seen that PLL the takes 3000 samples to converge. This can be improved by increasing the gain of the PLL. The signal at the output of the accumulator is shown in figure 7.21.

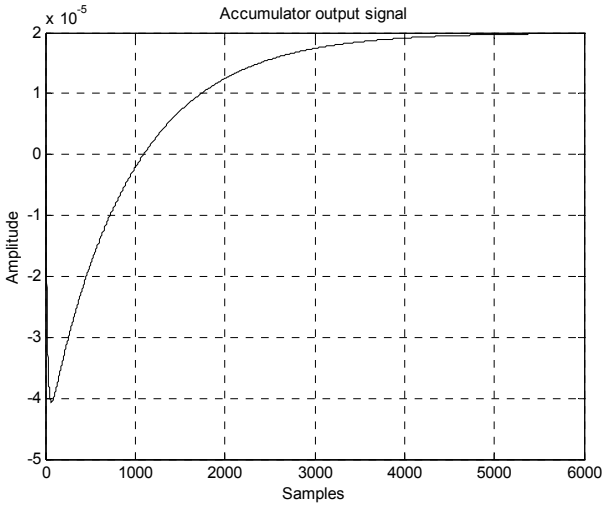

Figure 7.21: Accumulator output signal

7.8.2 Timing Recovery

Another simulation was run for timing recovery for two sample per symbols. Various fractional timing delays are introduced and the performance of the timing recovery loop is checked. For the fractional delay of *T*/2 the eye diagram for the signal at the input of the loop is

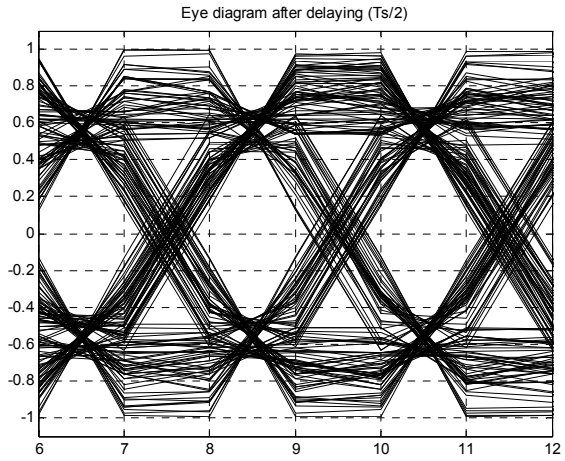

Figure 7.22: Eye diagram after delaying T/2.

You can see the ISI very clearly. At the output of the timing recovery loop the eye diagram is given in the figure 7.23. In this diagram you can see that the ISI is almost removed.

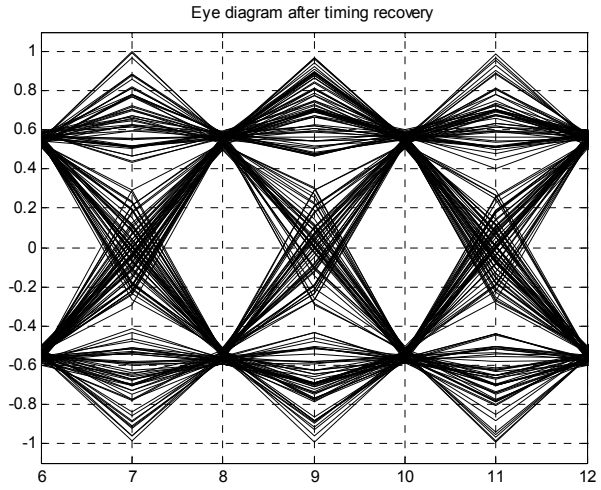

Figure 7.23: Eye diagram after timing recovery.

We have already explained that the Gardner TED is very sensitive to self noise.

To reduce the effect of self noise the stream of alternate ones and zeros is sent. After that the loop bandwidth is reduced and the random data is sent. The loop filter output signal is given below. The jitter is the curve is because of self noise.

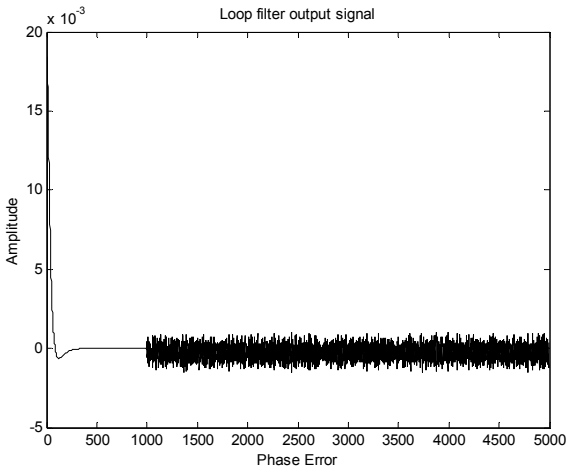

Figure 7.24: Loop filter output signal

CHAPTER 8

CHANNEL CODING

8.1 INFORMATION THEORY

The framework for studying fundamental limits in communication is *information theory.* The fundamental measure of performance is the *capacity* of a channel. Information theory was invented by Claude Shannon in 1948 to characterize the fundamental limits of reliable communication [18]. Before Shannon, it was widely believed that to the only way achieve reliable communication over a noisy channel, i.e. to make the error probability as small as desired, is to reduce the data rate, by say repeating the information symbol multiple times. Shannon however showed the surprising result that this is not necessary: by appropriate coding of the information, one can communicate at a positive rate but at the same time with as small an error probability as desired. However, there is a maximal rate, called the *capacity* of the channel, for which this can be done: if one attempts to communicate at rates above the channel capacity, then it is impossible to drive the error probability to zero. The capacity of a channel depends on the statistical characteristics of the channel, and for a wide class of channels, Shannon showed how the channel capacity can be computed. Channel capacity is therefore the fundamental measure of performance limit on reliable communication. The most common and probably the most important channel for a communication engineer is the gaussian *channel*, given by

$$y[n] = x[n] + w[n] \tag{8.1}$$

where $x[n]$ and $y[n]$ are real inputs at time n and $\{w[n]\}$ is an i.i.d. sequence of $N(0,\sigma^2)$ noise. Given a power constraint of $\overline{P}$ on the transmitted signal, the capacity is given by

$$C = \frac{1}{2}\log\left(1 + \frac{\overline{P}}{\sigma^2}\right) \tag{8.2}$$

Capacity is measured in bits per symbol. This is probably the most well-known result of information theory, but it is in fact only a special case of Shannon's general theory applied to a specific channel.

8.2 BASIC MODEL AND FORMULATION

To develop ideas, let us first start with the simplest class of channels: the *discrete memoryless channels* (DMCs). Both the input $x[n]$ and the output $y[n]$ of a DMC are discrete-valued and lie in finite sets X and Y respectively. (These sets are called the input and output alphabets of the channel respectively.) The statistics of the channel is described by conditional probabilities $\{Py/x)\}x \in X, y \in Y$ .These are also called *transition probabilities*. Given an input sequence x = $(x[1],\ldots\ldots, x[N])$, the probability of observing an output sequence y = $(y[1],\ldots\ldots.., y[N])$ is given by

$$p(y/x) = \prod_{n=1}^{N} p(y[n[x[n]]) \tag{8.3}$$

The interpretation is that the channel noise corrupts the input symbols independently (hence the term *memoryless*). The communication system is abstracted as shown in figure 8.1. The sender has one out of M equally likely messages it wants to transmit to the receiver. To convey the information, it uses a codebook *C* of block length N and size M, where $C = \{x_1.....,x_M\}$ and x*m*'s are the codewords. To transmit the *m*th message, the codeword x*m* is sent across the noisy channel. Based on the received vector y, the decoder generates an estimate ˆ*m* of the correct message. The error probability is $p_e = \mathrm{P}\{\overline{\hat{m}} \neq m\}$. We will assume that the maximum-likelihood (ML) decoder is used, since it minimizes the error probability for a given code. Since we are transmitting one of M messages, the number of bits conveyed is $\log M$. Since the block length of the code is N, the rate of the code is $R = \log M/N$ bits per unit time. The data rate *R* and the ML error probability p_e are the two key performance measures of a code.

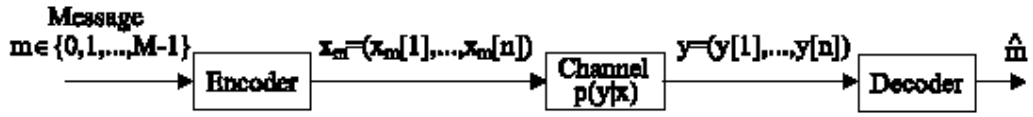

Figure 8.1: Basic communication model with channel encoder and decoder

Information is said to be *communicated reliably* at rate R if for every $\delta > 0$, one can find a code of rate *R* and block length N such that the error probability $p_e < \delta$. The capacity C of the channel is the maximum rate for which reliable communication is possible. Note the key feature of this definition is that one is allowed to code over arbitrarily large block lengths N. Since there is noise in the channel, it is clear that the error probability cannot be made arbitrarily small if the block length is fixed *a priori*. Only when one codes over long block lengths is there hope that one can rely on some kind of law of large numbers to average out the random effect of the noise. Still, it is not at all clear *a priori* whether a non-zero reliable information rate can be achieved.

Shannon showed not only that $C > 0$ for all but degenerate channels but in fact gives a simple way by which C can be computed as a function of $\{p(y/x)\}$.

That's concludes our discussion about information theory. We will not go into the depth of the topics. There are many error correcting codes found in the literature. Most of time forward error correcting codes are used which are further classified in block and convolutional codes. We will consider only the convolutional codes.

8.3 CHANNEL CODING

The purpose of forward error correction (FEC) is to improve the capacity of a channel by adding some carefully designed redundant information to the data being transmitted through the channel [14]. The process of adding this redundant information is known as channel coding. Convolutional coding and block coding are the two major forms of channel coding. Convolutional codes operate on serial data, one or a few bits at a time. Block codes operate on relatively large (typically, up to a couple of hundred bytes) message blocks.

There are a variety of useful convolutional and block codes, and a variety of algorithms for decoding the received coded information sequences to recover the original data.

Convolutional encoding with Viterbi decoding is a FEC technique that is particularly suited to a channel in which the transmitted signal is corrupted mainly by additive white Gaussian noise (AWGN). By using the convolutional channel coding significant improvement in SNR can be achieved.

Convolutional codes are usually described using two parameters: the code rate and the constraint length. The code rate, k/n, is expressed as a ratio of the number of bits into the convolutional encoder (k) to the number of channel symbols output by the convolutional encoder (n) in a given encoder cycle. The constraint length parameter, K, denotes the "length" of the convolutional encoder, i.e. how many k-bit stages are available to feed the combinatorial logic that produces the output symbols. Closely related to K is the parameter m, which indicates how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the convolutional encoder. The m parameter can be thought of as the memory length of the encoder. Since we have used code rate of 1/2,so we focus on rate 1/2 convolutional codes.

Viterbi decoding is one of two types of decoding algorithms used with convolutional encoding-the other type is sequential decoding. Sequential decoding has the advantage that it can perform very well with long-constraint-length convolutional codes, but it has a variable decoding time.

Viterbi decoding has the advantage that it has a fixed decoding time. It is well suited to hardware decoder implementation. But its computational requirements grow exponentially as a function of the constraint length, so it is usually limited in practice to constraint lengths of K = 9 or less.

But there's a tradeoff-the same data rate with rate 1/2 convolutional coding takes twice the bandwidth of the same signal without it, given that the modulation technique is the same. That's because with rate 1/2 convolutional

encoding, you transmit two channel symbols per data bit. However, if you think of the tradeoff as a 5 dB power savings for a 3 dB bandwidth expansion, you can see that you come out ahead. Remember: if the modulation technique stays the same, the bandwidth expansion factor of a convolutional code is simply n/k.

Many radio channels are AWGN channels, but many, particularly terrestrial radio channels also have other impairments, such as multipath, selective fading, interference, and atmospheric (lightning) noise. Transmitters and receivers can add spurious signals and phase noise to the desired signal as well. Although convolutional coding with Viterbi decoding might be useful in dealing with those other problems, it may not be the most optimal technique.

8.4 CONVOLUTIONAL ENCODING

Convolutionally encoding the data is accomplished using a shift register and associated combinatorial logic that performs modulo-two addition. (A shift register is merely a chain of flip-flops wherein the output of the nth flip-flop is tied to the input of the (n+1)th flip-flop. Every time the active edge of the clock occurs, the input to the flip-flop is clocked through to the output, and thus the data are shifted over one stage.) The combinatorial logic is often in the form of cascaded exclusive-or gates that implements the table 8.1

Input A	Input B	Output (A xor B)
0	0	0
0	1	1
1	0	1
1	1	0

Table 8.1: Convolutional Encoder output table

The exclusive-or gate performs modulo-two addition of its inputs. When you cascade q two-input exclusive-or gates, with the output of the first one feeding

one of the inputs of the second one, the output of the second one feeding one of the inputs of the third one, etc., the output of the last one in the chain is the modulo-two sum of the q + 1 inputs.

Now that we have the two basic components of the convolutional encoder (flip-flops comprising the shift register and exclusive-or gates comprising the associated modulo-two adders) defined, let's look at Figure 8.2 to see a picture of a convolutional encoder for a rate 1/2, K = 3, m = 2 code.

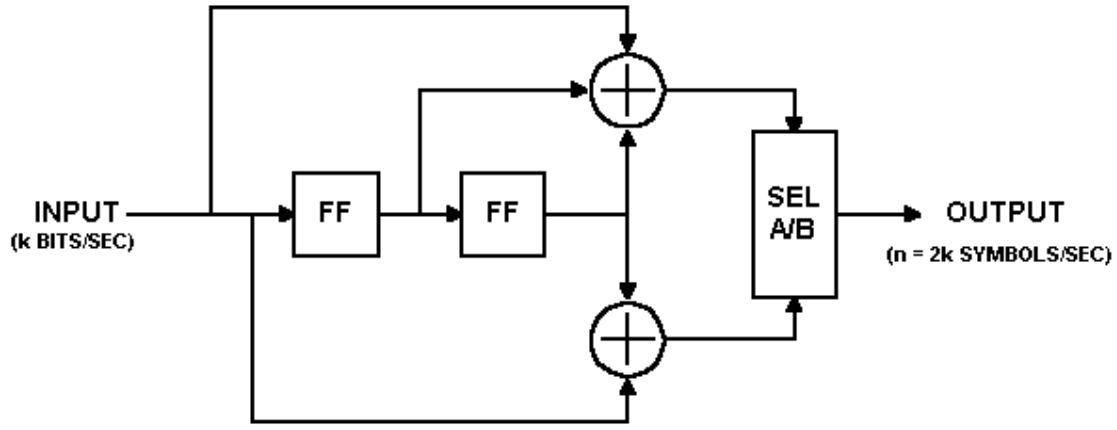

Figure 8.2: A typical convolutional encoder implementation for code rate 1/2

In this encoder, data bits are provided at a rate of k bits per second. Channel symbols are output at a rate of n = 2k symbols per second. The input bit is stable during the encoder cycle. The encoder cycle starts when an input clock edge occurs. When the input clock edge occurs, the output of the left-hand flip-flop is clocked into the right-hand flip-flop, the previous input bit is clocked into the left-hand flip-flop, and a new input bit becomes available. Then the outputs of the upper and lower modulo-two adders become stable. The output selector (SEL A/B block) cycles through two states-in the first state, it selects and outputs the output of the upper modulo-two adder; in the second state, it selects and outputs the output of the lower modulo-two adder.

The encoder shown above encodes the K = 3, (7, 5) convolutional code. The octal numbers 7 and 5 represent the code generator polynomials, which when read in binary (111_2 and 101_2) correspond to the shift register connections to the upper and lower modulo-two adders, respectively. This

code has been determined to be the "best" code for rate 1/2, K = 3. It is the code I will use for the remaining discussion and examples, for reasons that will become readily apparent when we get into the Viterbi decoder algorithm.

Let's look at an example input data stream, and the corresponding output data stream: Let the input sequence be 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1_2.

Assume that the outputs of both of the flip-flops in the shift register are initially cleared, i.e. their outputs are zeroes. The first clock cycle makes the first input bit, a zero, available to the encoder. The flip-flop outputs are both zeroes. The inputs to the modulo-two adders are all zeroes, so the output of the encoder is 00_2. The second clock cycle makes the second input bit available to the encoder. The left-hand flip-flop clocks in the previous bit, which was a zero, and the right-hand flip-flop clocks in the zero output by the left-hand flip-flop. The inputs to the top modulo-two adder are 100_2, so the output is a one. The inputs to the bottom modulo-two adder are 10_2, so the output is also a one. So the encoder outputs 11_2 for the channel symbols. The third clock cycle makes the third input bit, a zero, available to the encoder. The left-hand flip-flop clocks in the previous bit, which was a one, and the right-hand flip-flop clocks in the zero from two bit-times ago. The inputs to the top modulo-two adder are 010_2, so the output is a one. The inputs to the bottom modulo-two adder are 00_2, so the output is zero. So the encoder outputs 10_2 for the channel symbols. And so on. The timing diagram shown in figure 8.3 illustrates the process.

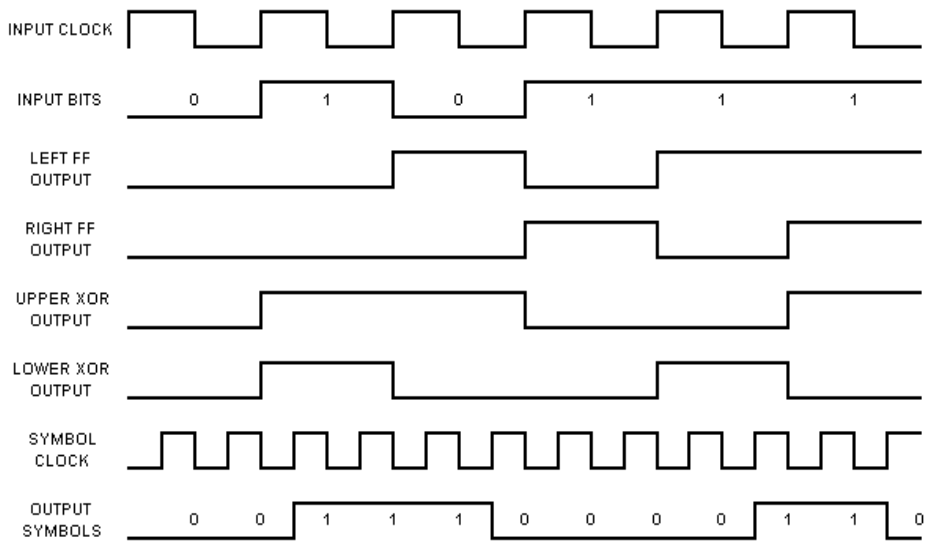

Figure 8.3: Timing diagram for convolutional encoder

After all of the inputs have been presented to the encoder, the output sequence will be: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11_2.

You can see from the structure of the rate 1/2 K = 3 convolutional encoder and from the example given above that each input bit has an effect on three successive pairs of output symbols. That is an extremely important point and that is what gives the convolutional code its error-correcting power. The reason why will become evident when we get into the Viterbi decoder algorithm. Now if we are only going to send the 15 data bits given above, in order for the last bit to affect three pairs of output symbols, we need to output two more pairs of symbols. This is accomplished in our example encoder by clocking the convolutional encoder flip-flops two more times, while holding the input at zero. This is called "flushing" the encoder, and results in two more pairs of output symbols. The final binary output of the encoder is thus 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11_2. If we don't perform the flushing operation, the last m bits of the message have less error-correction capability than the first through (m - 1)*th* bits had. This is a pretty important thing to remember if you're going to use this FEC technique in a burst-mode environment. So's the step of clearing the shift register at the beginning of

each burst. The encoder must start in a known state and end in a known state for the decoder to be able to reconstruct the input data sequence properly.

Now, let's look at the encoder from another perspective. You can think of the encoder as a simple state machine. The example encoder has two bits of memory, so there are four possible states. Let's give the left-hand flip-flop a binary weight of 2^1, and the right-hand flip-flop a binary weight of 2^0. Initially, the encoder is in the all-zeroes state. If the first input bit is a zero, the encoder stays in the all zeroes state at the next clock edge. But if the input bit is a one, the encoder transitions to the 10_2 state at the next clock edge. Then, if the next input bit is zero, the encoder transitions to the 01_2 state, otherwise, it transitions to the 11_2 state. The table 8.2 gives the next state given the current state and the input, with the states given in binary.

	Next State, if	
Current State	**Input = 0:**	**Input = 1:**
00	00	10
01	00	10
10	01	11
11	01	11

Table 8.2: State transition table

The table 8.2 is often called a state transition table. We'll refer to it as the next state table 8.3. Now let us look at a table that lists the channel output symbols, given the current state and the input data, which we'll refer to as the output table :

	Output Symbols, if	
Current State	**Input = 0:**	**Input = 1:**
00	00	11
01	11	00
10	10	01
11	01	10

Table 8.3: Output table

You should now see that with these two tables, you can completely describe the behavior of the example rate 1/2, K = 3 convolutional encoder. Note that both of these tables have $2^{(K-1)}$ rows, and 2^k columns, where K is the constraint length and k is the number of bits input to the encoder for each cycle. These two tables will come in handy when we start discussing the Viterbi decoder algorithm.

8.5 SOFT Vs HARD DECISION

An ideal Viterbi decoder would work with infinite precision, or at least with floating-point numbers. In practical systems, we quantize the received channel symbols with one or a few bits of precision in order to reduce the complexity of the Viterbi decoder, not to mention the circuits that precede it. If the received channel symbols are quantized to one-bit precision (< 0V = 1, > 0V = 0), the result is called hard-decision data. If the received channel symbols are quantized with more than one bit of precision, the result is called soft-decision data. A Viterbi decoder with soft decision data inputs quantized to three or four bits of precision can perform about 2 dB better than one working with hard-decision inputs. The usual quantization precision is three bits. More bits provide little additional improvement [14].

The selection of the quantizing levels is an important design decision because it can have a significant effect on the performance of the link. The following is a very brief explanation of one way to set those levels. Let's assume our received signal levels in the absence of noise are -1V = 1, +1V = 0. With noise, our received signal has mean ±1 and standard deviation $\sigma = \sqrt{1/(2(E_S/N_O))}$. Let's use a uniform, three-bit quantizer having the input/output relationship shown in the figure 8.4, where D is a decision level that we will calculate shortly:

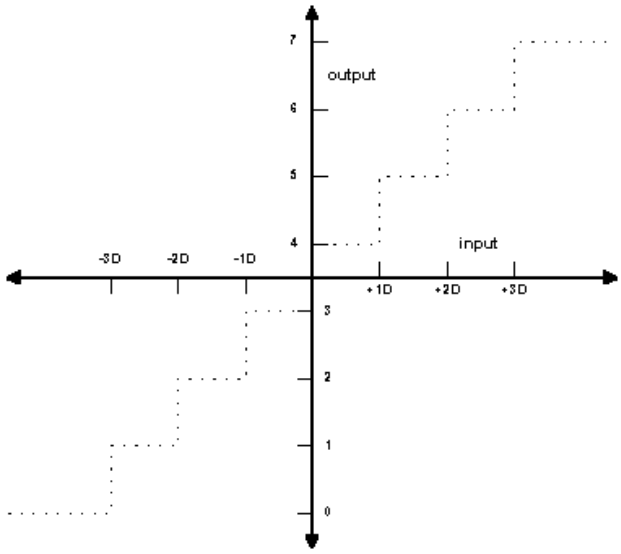

Figure 8.4: Quantizing the received signal

The decision level, D, can be calculated according to the formula $D = .5 \cdot \sigma = .5 \cdot \sqrt{1/\left(2\left(E_S / N_O\right)\right)}$, where E_s/N_0 is the energy per symbol to noise density ratio. We will assume hard decision for Viterbi decoding.

8.6 VITERBI DECODING

The single most important concept to aid in understanding the Viterbi algorithm is the trellis diagram. The figure 8.5 shows the trellis diagram for our example rate 1/2 K = 3 convolutional encoder, for a 15-bit message:

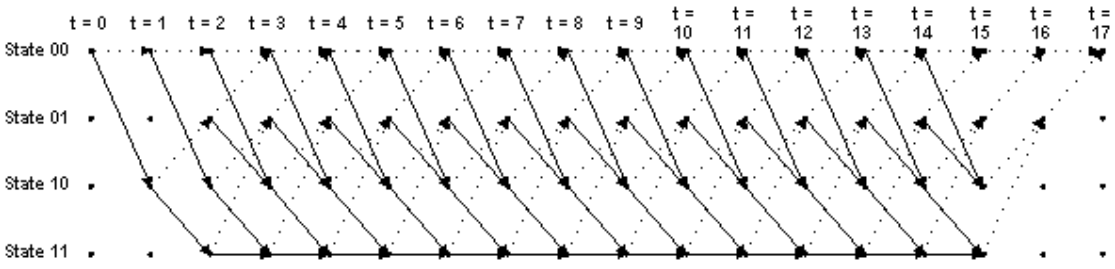

Figure 8.5: Trellis diagram

The four possible states of the encoder are depicted as four rows of horizontal dots. There is one column of four dots for the initial state of the encoder and one for each time instant during the message. For a 15-bit message with two encoder memory flushing bits, there are 17 time instants in addition to t = 0, which represents the initial condition of the encoder. The solid lines connecting dots in the diagram represent state transitions when the input bit is a one. The dotted lines represent state transitions when the input bit is a zero. Notice the correspondence between the arrows in the trellis diagram and the state transition table discussed above. Also notice that since the initial condition of the encoder is State 00_2, and the two memory flushing bits are zeroes, the arrows start out at State 00_2 and end up at the same state [14].

The figure 8.6 shows the states of the trellis that are actually reached during the encoding of our example 15-bit message:

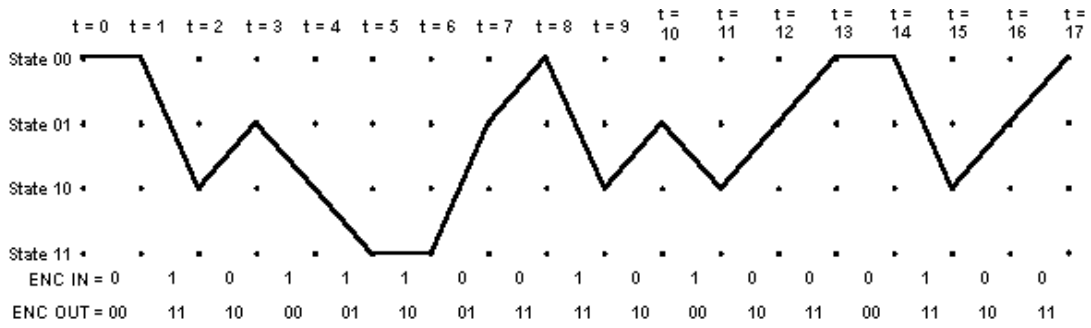

Figure 8.6: Actually reached path through trellis diagram

The encoder input bits and output symbols are shown at the bottom of the diagram. Notice the correspondence between the encoder output symbols and the output table discussed above. Let's look at that in more detail, using the expanded version of the transition between one time instant to the figure 8.7.

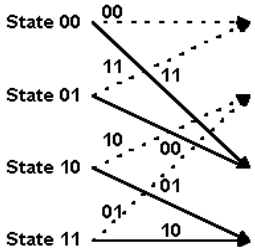

Figure 8.7: Diagram showing transition from one state to other

The two-bit numbers labeling the lines are the corresponding convolutional encoder channel symbol outputs. Remember that dotted lines represent cases where the encoder input is a zero, and solid lines represent cases where the encoder input is a one. (In the figure above, the two-bit binary numbers labeling dotted lines are on the left, and the two-bit binary numbers labeling solid lines are on the right. Now let's start looking at how the Viterbi decoding algorithm actually works. For our example, we're going to use hard-decision symbol inputs to keep things simple. Suppose we receive the above-encoded message with a couple of bit errors as shown in figure 8.8.

Each time we receive a pair of channel symbols, we're going to compute a metric to measure the "distance" between what we received and all of the possible channel symbol pairs we could have received. Going from t = 0 to t = 1, there are only two possible channel symbol pairs we could have received: 00_2, and 11_2. That's because we know the convolutional encoder was initialized to the all-zeroes state, and given one input bit = one or zero, there are only two states we could transition to and two possible outputs of the encoder. These possible outputs of the encoder are 00_2 and 11_2.

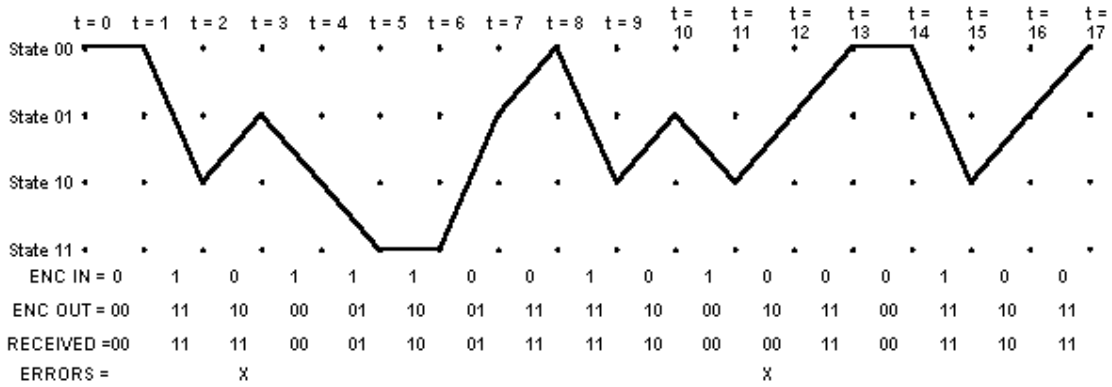

Figure 8.8: Path through the trellis diagram for received bits

The metric we're going to use for now is the Hamming distance between the received channel symbol pair and the possible channel symbol pairs. The Hamming distance is computed by simply counting how many bits are different between the received channel symbol pair and the possible channel symbol pairs. The results can only be zero, one, or two. The Hamming distance (or other metric) values we compute at each time instant for the paths between the states at the previous time instant and the states at the current time instant are called branch metrics. For the first time instant, we're going to save these results as "accumulated error metric" values, associated with states. For the second time instant on, the accumulated error metrics will be computed by adding the previous accumulated error metrics to the current branch metrics.

At t = 1, we received 00_2. The only possible channel symbol pairs we could have received are 00_2 and 11_2. The Hamming distance between 00_2 and 00_2 is zero. The Hamming distance between 00_2 and 11_2 is two. Therefore, the branch metric value for the branch from State 00_2 to State 00_2 is zero, and for the branch from State 00_2 to State 10_2 it's two. Since the previous accumulated error metric values are equal to zero, the accumulated metric values for State 00_2 and for State 10_2 are equal to the branch metric values. The accumulated error metric values for the other two states are undefined. The figure 8.9 illustrates the results at t = 1.Note that the solid lines between

states at t = 1 and the state at t = 0 illustrate the predecessor-successor relationship between the states at t = 1 and the state at t = 0 respectively. This information is shown graphically in the figure, but is stored numerically in the actual implementation. To be more specific, or maybe clear is a better word, at each time instant t, we will store the number of the predecessor state that led to each of the current states at t.

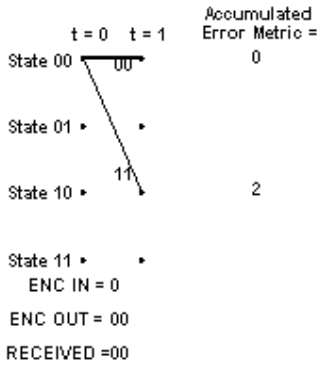

Figure 8.9: Trellis diagram at t=1

Now let's look what happens at t = 2. We received a 11_2 channel symbol pair. The possible channel symbol pairs we could have received in going from t = 1 to t = 2 are 00_2 going from State 00_2 to State 00_2, 11_2 going from State 00_2 to State 10_2, 10_2 going from State 10_2 to State 01_2, and 01_2 going from State 10_2 to State 11_2. The Hamming distance between 00_2 and 11_2 is two, between 11_2 and 11_2 is zero, and between 10_2 or 01_2 and 11_2 is one. We add these branch metric values to the previous accumulated error metric values associated with each state that we came from to get to the current states. At t = 1, we could only be at State 00_2 or State 10_2. The accumulated error metric values associated with those states were 0 and 2 respectively. The figure 8.10 below shows the calculation of the accumulated error metric associated with each state, at t = 2.

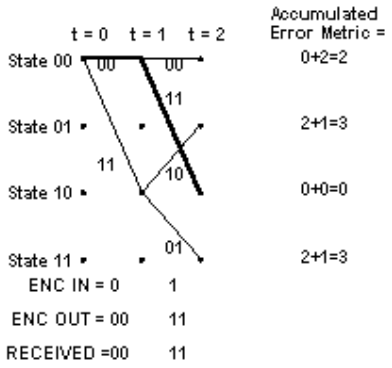

Figure 8.10: Trellis diagram at t=2

That's all the computation for t = 2. What we carry forward to t = 3 will be the accumulated error metrics for each state, and the predecessor states for each of the four states at t = 2, corresponding to the state relationships shown by the solid lines in the illustration of the trellis.

Now look at the figure for t = 3. Things get a bit more complicated here, since there are now two different ways that we could get from each of the four states that were valid at t = 2 to the four states that are valid at t = 3. So how do we handle that? The answer is, we compare the accumulated error metrics associated with each branch, and discard the larger one of each pair of branches leading into a given state. If the members of a pair of accumulated error metrics going into a particular state are equal, we just save that value. The other thing that's affected is the predecessor-successor history we're keeping. For each state, the predecessor that survives is the one with the lower branch metric. If the two accumulated error metrics are equal, some people use a fair coin toss to choose the surviving predecessor state. Others simply pick one of them consistently, i.e. the upper branch or the lower branch. It probably doesn't matter which method you use. The operation of adding the previous accumulated error metrics to the new branch metrics, comparing the results, and selecting the smaller (smallest) accumulated error metric to be retained for the next time instant is called the add-compare-select operation. The figure 8.11 shows the results of processing t = 3:

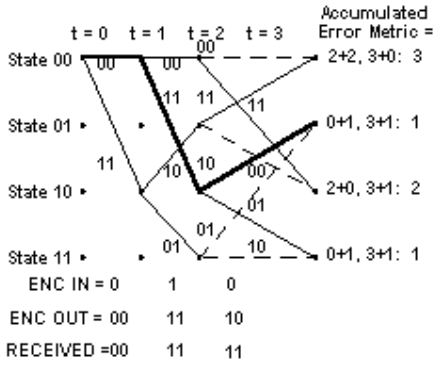

Figure 8.11: Trellis diagram at t=3

Note that the third channel symbol pair we received had a one-symbol error. The smallest accumulated error metric is a one, and there are two of these.

Let's see what happens now at t = 4. The processing is the same as it was for t = 3. The results are shown in the figure 8.12:

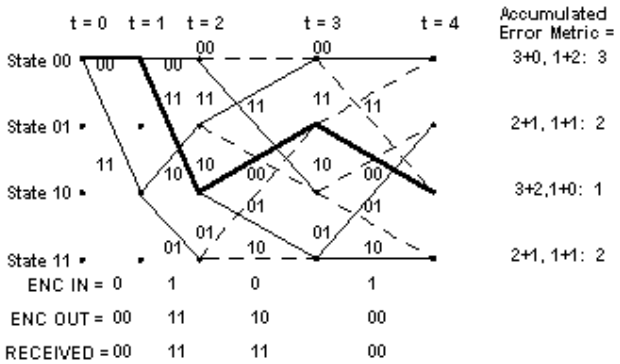

Figure 8.12: Trellis diagram at t=4

Notice that at t = 4, the path through the trellis of the actual transmitted message, shown in bold, is again associated with the smallest accumulated error metric. Let's look at t = 5 as shown in figure 8.13:

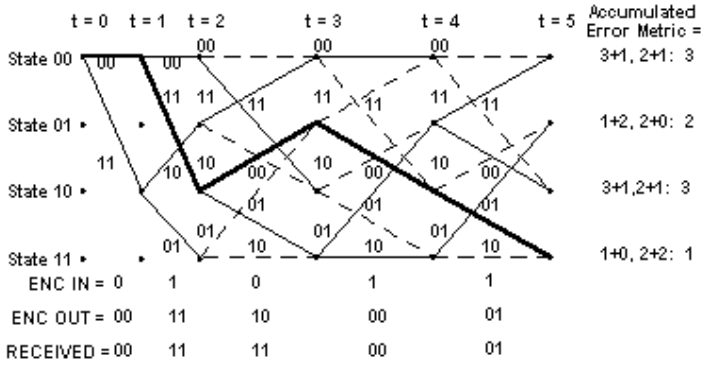

Figure 8.13: Trellis diagram at t=5

At t = 5, the path through the trellis corresponding to the actual message, shown in bold, is still associated with the smallest accumulated error metric. This is the thing that the Viterbi decoder exploits to recover the original message. At t = 17, the trellis looks like this, with the clutter of the intermediate state history removed:

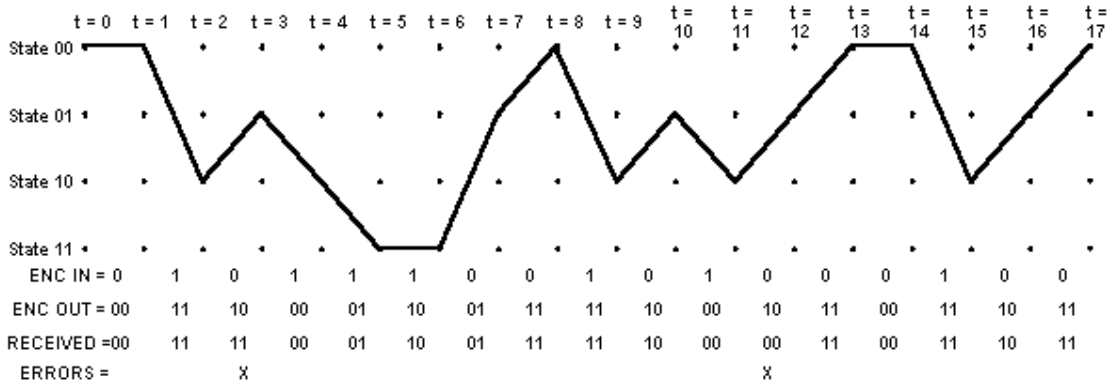

Figure 8.14: Trellis diagram for complete packet

The decoding process begins with building the accumulated error metric for some number of received channel symbol pairs, and the history of what states preceded the states at each time instant t with the smallest accumulated error metric. Once this information is built up, the Viterbi decoder is ready to

recreate the sequence of bits that were input to the convolutional encoder when the message was encoded for transmission. This is accomplished by following steps

- First, select the state having the smallest accumulated error metric and save the state number of that state.
- Iteratively perform the following step until the beginning of the trellis is reached: Working backward through the state history table, for the selected state, select a new state which is listed in the state history table as being the predecessor to that state. Save the state number of each selected state. This step is called traceback.
- Now work forward through the list of selected states saved in the previous steps. Look up what input bit corresponds to a transition from each predecessor state to its successor state. That is the bit that must have been encoded by the convolutional encoder. The table 8.4 shows the accumulated metric for the full 15-bit (plus two flushing bits) example message at each time t.

T =	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
State 00_2		0	2	3	3	3	3	4	1	3	4	3	3	2	2	4	5	2
State 01_2			3	1	2	2	3	1	4	4	1	4	2	3	4	4	2	
State 10_2		2	0	2	1	3	3	4	3	1	4	1	4	3	3	2		
State 11_2			3	1	2	1	1	3	4	4	3	4	2	3	4	4		

Table 8.4: Accumulated metric table

It is interesting to note that for this hard-decision-input Viterbi decoder example, the smallest accumulated error metric in the final state indicates how many channel symbol errors occurred. The table 8.5 shows the surviving predecessor states for each state at each time t.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State 00_2	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	1
State 01_2	0	0	2	2	3	3	2	3	3	2	2	3	2	3	2	2	2	0
State 10_2	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0
State 11_2	0	0	2	2	3	2	3	2	3	2	2	3	2	3	2	2	0	0

Table 8.5: Surviving predecessor states table

The table 8.6 shows the states selected when tracing the path back through the survivor state table shown above.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0	2	1	0

Table 8.6: Trace back table

Using a table that maps state transitions to the inputs that caused them, we can now recreate the original message. Here is what this table looks like for our example rate 1/2 K = 3 convolutional code.

	Input was, Given Next State =			
Current State	$00_2 = 0$	$01_2 = 1$	$10_2 = 2$	$11_2 = 3$
$00_2 = 0$	0	x	1	x
$01_2 = 1$	0	x	1	x
$10_2 = 2$	X	0	x	1
$11_2 = 3$	X	0	x	1

Table 8.7: Next state table

Note: In table 8.7, x denotes an impossible transition from one state to another state. So now we have all the tools required to recreate the original message from the message we received.

t =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	1	1	1	0	0	1	0	1	0	0	0	1

Table 8.7: Original message

The two flushing bits are discarded. Here's an insight into how the traceback algorithm eventually finds its way onto the right path even if it started out choosing the wrong initial state. This could happen if more than one state had the smallest accumulated error metric, for example. Figure 8.15 for the trellis at t = 3 again to illustrate this point.

See how at t = 3, both States 01_2 and 11_2 had an accumulated error metric of 1. The correct path goes to State 01_2 -notice that the bold line showing the actual message path goes into this state. But suppose we choose State 11_2 to start our traceback. The predecessor state for State 11_2 , which is State 10_2 , is the same as the predecessor state for State 01_2! This is because at t = 2, State 10_2 had the smallest accumulated error metric. So after a false start, we are almost immediately back on the correct path.

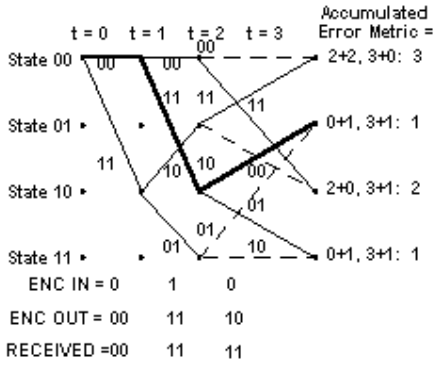

Figure 8.15: Trellis diagram at t=3

For the example 15-bit message, we built the trellis up for the entire message before starting traceback. For longer messages, or continuous data, this is neither practical nor desirable, due to memory constraints and decoder delay. Research has shown that a traceback depth of K x 5 is sufficient for Viterbi decoding with the type of codes we have been discussing. Any deeper traceback increases decoding delay and decoder memory requirements, while not significantly improving the performance of the decoder. The exception is punctured codes, which I'll describe later. They require deeper traceback to reach their final performance limits.

To implement a Viterbi decoder in software, the first step is to build some data structures around which the decoder algorithm will be implemented. These data structures are best implemented as arrays. The primary six arrays that we need for the Viterbi decoder are as follows:

- A copy of the convolutional encoder next state table, the state transition table of the encoder. The dimensions of this table (rows x columns) are $2^{(K - 1)}$ x 2^{k}. This array needs to be initialized before starting the decoding process.
- A copy of the convolutional encoder output table. The dimensions of this table are $2^{(K - 1)}$ x 2^{k}. This array needs to be initialized before starting the decoding process.
- An array (table) showing for each convolutional encoder current state and next state, what input value (0 or 1) would produce the next state, given the current state. We'll call this array input table. Its dimensions are $2^{(K - 1)}$ x $2^{(K - 1)}$. This array needs to be initialized before starting the decoding process.
- An array to store state predecessor history for each encoder state for up to K x 5 + 1 received channel symbol pairs. We'll call this table the state history table. The dimensions of this array are $2^{(K - 1)}$ x (K x 5 + 1). This array does not need to be initialized before starting the decoding process.
- An array to store the accumulated error metrics for each state computed using the add-compare-select operation. This array will be called the accumulated error metric array. The dimensions of this array are $2^{(K - 1)}$ x 2. This array does not need to be initialized before starting the decoding process.

- An array to store a list of states determined during traceback (term to be explained below). It is called the state sequence array. The dimensions of this array are (K x 5) + 1. This array does not need to be initialized before starting the decoding process.

REFRENCES

[1] Jeffrey H. Reed, “Software Radio, A Modern Approach to Radio Engineering”, Prentice Hall, 2002.

[2] Enrico Buracchini, CSELT, “The Software Radio Concept”, IEEE Communication Magazine, September 2000.

[3] Tim Hentschel and Gerhard Fettweis, “Sample Rate Conversion for Software Radio”, IEEE Communication Magazine, August 2000.

[4] Alan V. Oppenheim, Alan Willsky and S. Hamid Nawab, ‘’Signals and Systems”, Prentice Hall, 2nd Edition, 2000.

[5] Alan V. Oppenheim and Ronald W. Schafer, “Discrete-Time signal Processing”, Prentice Hall, 2001

[6] Emmanual C. Ifeachor and Barrie W. Jervis, “Digital Signal Processing, A Practical Approach”, Addison Wesely, 1999

[7] Theodore S. Rappaport, “Wireless Communications, Principals and Practice”, Prentice Hall, 1996.

[8] John G. Proakis, ”Digital Communications ”, McGraw Hill, Second Edition, 1989.

[9] Edward A. Lee and David G. Messerschmitt, “Digital Communications”, Kluwer Academic Publishers, Second Edition, 1994.

[10] Simon Haykin, “Adaptive Filter Theory”, Prentice Hall, Fourth Edition,2002.

[11] Dimitris G. Manolakis, Vinay K. Ingle and Stephen M.Kogon, “Statistical and Adaptive Signal Processing”, McGRAW-HILL Publishers, 2002.

[12] Erwin Kreyszig, ”Advanced Engineering Mathematics”, John Wiley and Sons, 8th Edition, 1999.

[13] Heinrich Meyr, Marc Moeneclaey and Stafen A. Fechtel, “Digital Communication Receivers”, John Wiley and Sons, 1998.

[14] Bernard Sklar, “Digital Communications, Fundamentals and Applications”, Prentice Hall, 2nd Edition, 2001.

[15] B.P Lathi, ” Modern Digital and Analog Communication Systems”, Holt Saunders, 1983.

[16] Andrew S. Tanenbaum, "Computer Networks", Prentice Hall, 3rd Edition, 1996.

[17] John A. C. Bingham, "The Theory and Practice of Modem Design", John Wiley and Sons, 1986.

[18] C. E Shanon, "A Mathematical Theory of Communication", Bell System Technical Journal, Vol.27, July, October, 1948.

[19] M. K Jain, S. R. K. Iyengar and R. K Jain, "Numerical Methods for Scientific and Engineering Computation", Wiley Eastern Limited, 3rd Edition, 1993.

[20] Athanasios Papoulis, "Probablity, Random Variables and Stochastic Processes", McGraw Hill, 3rd Edition, 1991.