OPTIMAL COMMUNICATION SYSTEM DESIGN

SYNDICATE

NC BILAL RASHEED(LEADER)
NC ZOHAIB NAVEED
PC ZAIN MOHATAREM
PC ALI ASIF

PROJECT SUPERVISOR:
MAJ®. FAZAL AHMED
External DS: MR.SALMAN ANWAR

Dissertation submitted in partial fulfillment of the 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 2005

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 required in accomplishing this extremely challenging and gigantic task.

After Almighty Allah we are extremely thankful to Mr.Salman Anwar Rao for his providential and able supervision. We are proud of our effort truly because of him. He has been a constant source of inspiration and available to us despite his innumerable commitments.

We would also like to extend our gratitude to our friend Ali Abbas Zaidi who helped us a great deal in this endeavor. We are also grateful to Maj® Fazal Ahmed who has helped us not only in our project but he has been a complete source of inspiration and motivation throughout our course of studies. Special thanks to Electrical Engineering Department its staff members for their readily available assistance and cooperation in the provision of material resources. We also extend our deep gratitude to all those who have encouraged us in the strive for achieving the goals.

ABSTRACT

We have endeavored to develop an optimal communication system by considering different trade-offs pertinent to the communication system**.** The design objective is achieved by an acceptable compromise between information flow, error rate, capital and operation cost.

Focusing onto the maximum bit rate and minimum bit error rate, we have analyzed different digital modulation and channel coding techniques . Maintaining the same optimization criteria an efficient receiver design has been undertaken Receiver is incorporated with sophisticated algorithms including adaptive equalization and synchronization algorithms. Optimality of the selected algorithms is proven analytically through rigorous simulations.

TABLE OF CONTENTS

CHAPTER 4

MODULATION , MATCH FILTERING AND PULSE SHAPING

CHAPTER 5

SYNCHRONIZATION

LIST OF FIGURES

CHAPTER 1

OPTIMAL COMMUNICATION SYSTEM CONCEPT

1.1 INTRODUCTION

The extension of man's nervous system around the surface of the earth is shrinking our world to a "global village". The revolutionary potential of new communications "electric" technology is the subject of many articles and books .

The communications satellite, on-line real time computers, large scale integration, the helical wave guide and the laser are just a few of the technological inventions of the revolution. Just as exciting to the specialist, although inaccessible to the layman, are the theoretical break- throughs which will make their impact in the efficient utilization of this new technology. The aim of this project is to briefly introduce the reader with a science or engineering background to some of the theoretical problems and Trade-offs associated with communication systems, and to illustrate by example the application of theoretical results to a practical communication system problem.

1.2 WHY OPTIMIZE.?

Optimization is to get the best out of your available resources. With the advent of new technologies the EM spectrum is getting expensive day by day. Competition between different companies is on mounting. One has to use its available resources in a best possible way in order to achieve the best results and gain customer satisfaction

1.3 OUR DESIGN

Our project is an analytical research based project where we have implemented different algorithms. We consider the usual communication system model (see fig. 1) consisting of transmitter (modulator and channel encoder), communication channel and receiver (demodulator).Clearly the design objective is to achieve an acceptable compromise between information flow, error rate,

capital equipment cost and operating cost.

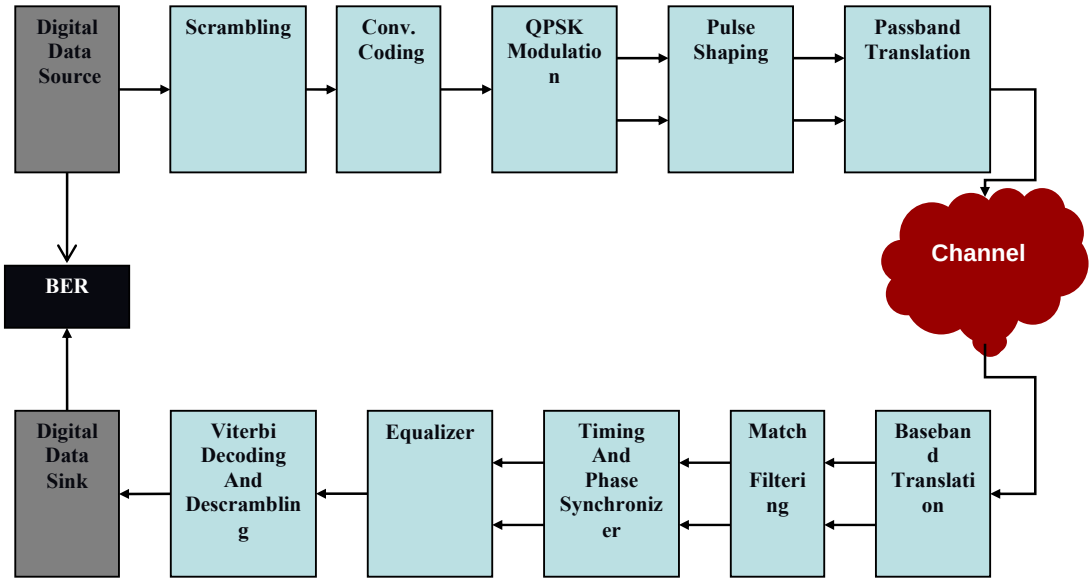

Fig 1..Our Design

In order to achieve this we have to compromise between different situations and deal with different kinds of trade-offs. for example moving to a higher modulation scheme improves bandwidth efficiency but degrades probability of bit error. Similarly introducing error correction coding gives improved performance but requires extra bandwidth. We have studied these different trades-offs and tried to simulate a good communication system. We have handled each module of our system individually and considered different algorithms for that module and compared their performance. In the end we have developed a QPSK transmitter/Receiver capable of functionalities like scrambling channel codeing..timing and phase recovery and equalization. Simulation is done in matlab and care has been taken to avoid complexity.

CHAPTER 2

SCRAMBLING

2.1 SCRAMBLING

In general, a scrambler tends to make the data more random by removing long strings of 1's or 0's. Scrambling can be helpful in timing, extraction by removing long strings of 0's in binary data. Scramblers, however, are primarily used for preventing unauthorized access to the data, and are optimized for that purpose.

Figure 2.1 shows a typical scrambler and descrambler. The scrambler consists of a feedback shift register, and the matching descrambler has a feed forward shift register. as illustrated in Fig. 2.1 Each stage in the shift register delays a bit by one unit. To analyze the scrambler and the matched descrambler, consider the output sequence T of the scrambler (Fig. 2.1(a)). If *S* is the input sequence to the scrambler, then

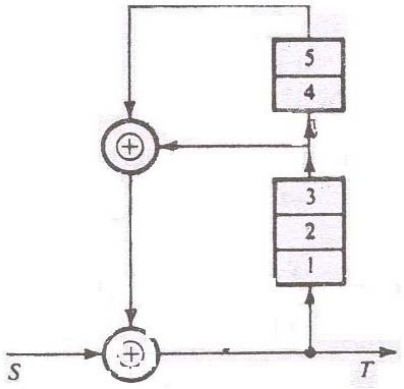

Figure 2.1(a) Scrambler

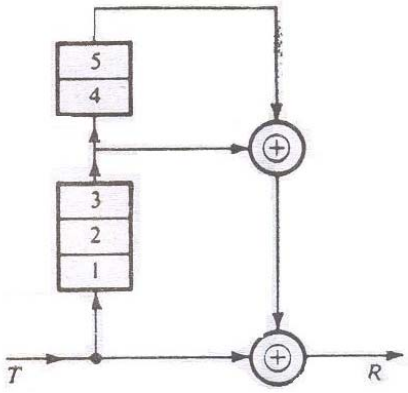

Figure (2.1)b Descrambler

$$S \oplus D^3T \oplus D^5T = T \qquad (2.1)$$

Where *D* represents the delay operator that is D^nT is the sequence *T* delayed by n units. The symbol $\oplus$ indicates modulo 2 sum.Now recall that the modulo 2 sum of any sequence with itself gives a sequence of all 0's. Adding $(D^3 \oplus D^5)T$ to both sides of Eq. (2.1), we get

$$\begin{aligned} S &= T \oplus (D^3 \oplus D^5)\, T \\ &= [1 \oplus (D^3 \oplus D^5)]T \\ &= (1 \oplus F)T \qquad \text{where } F = (D^3 \oplus D^5) \end{aligned} \qquad (2.2)$$

To design the descrambler at the receiver, we start with *T,* the sequence received

at the descrambler. From Eq. (2.2) , it follows that

$$= T \oplus FT = T \oplus (D^3 \oplus D^5)T$$

This equation, where we regenerate the input sequence S from the received sequence *T,* is readily implemented by the descrambler shown in Fig.2.1 (b).

Note that a single detection error in the received sequence *T* will affect three output bits in *R.* Hence, scrambling has the disadvantage of causing multiple errors for a single received bit error.[8]

CHAPTER 3

CHANNEL CODING

3.1 INFORMATION THEORY

Information theory describes the fundamental limits in communication .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 . 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, 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.

3.2 WHY USE ERROR CORRECTION CODING?

Error correction coding can be considered as a vehicle for effecting various system trade-offs. Figure 3.1 compares two curves depicting bit-error performance versus E_b/N_0.One curve represents a typical Modulation scheme without coding. The Second curve represents the same Modulation with coding. Now we discuss the benefits and trade-offs that can be achieved with the help of channel coding

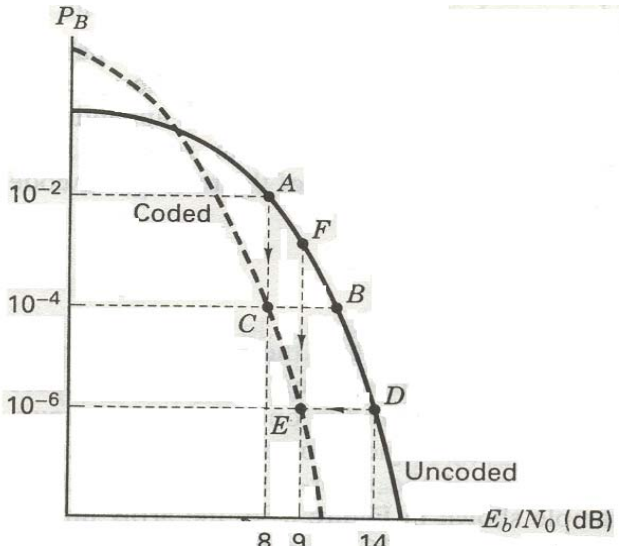

Fig.3.1 TRADE OFF : Performance versus bandwidth

Imagine that a simple, inexpensive voice communication system has just been developed and delivered to a customer. The system does not use error-correction coding. Consider that the operating point of the system can be depicted by point *A* in Figure 3.1 *(E_b/N_o* = 8 dB, and *PB* = 10^{-2}). After a few trials, there are complaints about the voice quality; the customer suggests that the bit-error probability should be lowered to $10^{-4.}$ The usual way of obtaining better error performance in such a system would be by effecting an operating point movement from point *A* to, say, point *B* in Figure 3.1. However, suppose that the E_blN_o of 8 dB is the most that is available in this system. The figure suggests that one possible trade-off is to move the operating point from point *A* to point C. That is, "walking"' down the vertical line to point C on the coded curve can provide the customer with improved error performance. What does it cost? Aside from the new components (encoder and de. coder) needed, the price is more transmission bandwidth. Error-correction coding needs redundancy. If we assume that the system is a real-time communication system (such that the message may not be delayed),The addition of redundant bits dictates a faster rate of transmission, which of course means more bandwidth.

3.2.1 TRADE-OFF:2 POWER VERSUS BANDWIDTH

Consider that a system without coding, operating at point *D* in Figure 3.1 *(E_b/N_o* = 14 dB, and *PB* = 10^{-6}), has been delivered to a customer. The customer has no complaints about the quality of the data, but the equipment is having some reliability problems as a result of providing an E_blN_o of 14 dB. In other words, the equipment keeps breaking down. If the requirement on *EblNo* or power could be reduced, the reliability difficulties might also be reduced. Figure 3.1 suggests a trade-off by moving the operating point from point *D* to point *E.* That is, if error correction coding is introduced, a reduction in the required *EblNo* can be achieved. Thus, the trade-off is one in which the same quality of data is achieved, but the coding allows for a reduction in power or E_blN_o. What is the cost? The same as before-more bandwidth.

Notice that for *non-real-time* communication systems, error-correction coding can be used with a somewhat different trade-off. It is possible to obtain improved bit-error probability or reduced power (similar to trade-off 1 or 2 above) by paying the price of *delay* instead of bandwidth.

3.3 CODING GAIN

The trade-off example described in the previous section has allowed a reduc tion in *EblNo* from 14 dB to 9 dB, while maintaining the same error performance. In the context of this example and Figure 6.9, we now define *coding gain.* For a *given bit-error probability,* coding gain is defined as the "relief" or reduction in $E_b lN_o$ that can be realized through the use of the code. Coding gain G is generally expressed in dB, such as

$$G(dB) = (E_b/N_0)_u\ (dB) - (E_b/N_0)_c\ (dB)$$

Where $(E_b/N_0)_u$ and $(E_b/N_0)_c$ represent the required E_b/N_0 uncoded and coded respectively.[7]

3.4 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 . 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.

3.5 LINEAR BLOCK CODES

Linear block codes are a class of parity check codes that can be characterized by the *(n, k)* notation. The encoder transforms a block of *k* message digits (a message vector) into a longer block of *n* codeword digits (a code vector) constructed from a given alphabet of elements. When the alphabet consists of two elements (0 and 1), the code is a binary code comprising binary digits (bits). Our discussion of linear block codes is restricted to binary codes unless otherwise noted.

The *k-bit* messages form 2^k distinct message sequences, referred to as *k- tuples* (sequences of *k* digits). The *n-bit* blocks can form as many as 2" distinct sequences, referred to as *n-tuples.* The encoding procedure assigns to each of the 2^k message k-tuples *one* of the 2" n-tuples. A block code represents a one-to-one assignment, whereby the 2^k message k-tuples are *uniquely* mapped into a new set of 2^k codeword n-tuples; the mapping can be accomplished via a look-up table. For *linear codes,* the mapping transformation is, of course *linear.*

3.5.1 VECTOR SPACES AND SUBSPACES

The set of all binary n-tuples, V_n is called a *vector space* over the binary field of two elements (0 and 1).

A subset S of the vector space V_n is called a *subspace* if the following two conditions are met:

1. The all-zeros vector is in S.
2. The sum of any two vectors in S is also in S (known as the *closure property).*

For example, the vector space V_4 is totally populated by the following 2^4 = sixteen 4-tuples

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

An example of a subset of V_4 that forms a subspace is

0000 0101 1010 1111

3.5.2 GENERATOR MATRIX

If *k* is large, a *table look-up* implementation of the encoder becomes prohibitive. For a (127, 92) code there are 292 or approximately 5 x 1027 code vectors. If the encoding procedure consists of a simple look-up table, imagine the size of the memory necessary to contain such a large number of codewords. Fortunately, it is possible to reduce complexity by generating the required codewords as needed, instead of storing them.

Since a set of codewords that forms a linear block code is a k-dimensional subspace of the n-dimensional binary vector space *(k < n),* it is always possible to find a set of n-tuples fewer than *2k,* that can generate all the *2k* codewords of the subspace. The generating set of vectors is said to *span* the subspace. The smallest *linearly independent* set that spans the subspace is called a *basis* of the subspace, and the number of vectors in this basis set is the dimension of the subspace. Any basis set of *k* linearly independent *n-tuples* V_1 V_2, . . . , V_k can be used to generate the required linear block code vectors, since each code vector is a linear combination of V_1 to V_2, . . . , V_k . That is, each of the set of *2k* codewords (U) can be described by

$$U= m_1V_1 + m_2V_2 +\ldots\ldots+ m_kV_k$$

In general we can define a generator matrix by the following k*n array

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_k \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & & & \\ v_{k1} & v_{k2} & \cdots & v_{kn} \end{bmatrix}$$

Code vectors, by convention, are usually designated as row vectors. Thus, the message m a sequence of *k* message bits, is shown below as a row vector (1 x *k* matrix having one row and *k* columns):

$$M= m_1, m_2, m_3\ldots\ldots\ldots m_k$$

The generation of the codeword U is written in matrix notation as the product of m and G, and we write

$$U=mG$$

3.5.3 SYSTEMATIC LINEAR BLOCK CODES

A systematic *(n, k)* linear block code is a mapping from a k-dimensional message vector to an n-dimensional codeword in such a way that part of the sequence generated coincides with the *k* message digits. The remaining *(n - k)* digits are parity digits. A systematic linear block code will have a generator matrix of the

form

$$\mathbf{G} = \left[\begin{array}{c:c} \mathbf{P} & \mathbf{I}_k \end{array}\right]$$

$$= \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,(n-k)} & 1 & 0 & \cdots & 0 \\ p_{21} & p_{22} & \cdots & p_{2,(n-k)} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & \vdots & \\ p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Where P is the parity array portion of the generator matrix *Pij* = (0 or 1), and I_k is the *k* x *k* identity matrix. Notice that with the systematic generator, the encoding complexity is further reduced since it is not necessary to store the identity matrix portion of the array.

3.5.4 PARITY CHECK MATRIX

Let us define a matrix H. called the parity-check matrix that will enable us to decode the received vectors. For each *(k* x *n)* generator matrix G, there exists an *(n - k)* x *n* matrix H. such that the rows of G are orthogonal to the rows of H; that is, $GH^T = 0$, where H^T is the *transpose* of H, and 0 is a *k* x *(n - k)* all-zeros matrix. H^T is an *n* x *(n - k)* matrix whose rows are the columns of H and whose columns are the rows of H. To fulfill the orthogonality requirements for a systematic code, the components of the H matrix are written as

$$\mathbf{H} = \left[\mathbf{I}_{n-k} \,\vdots\, \mathbf{P}^T\right]$$

Hence H^T can be written as

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \text{-----} \\ \mathbf{P} \end{bmatrix}$$

3.5.5 SYNDROME TESTING

Let r = r_1, r_2,…….; r_n be a received vector resulting from the transmission *of U* = U_1 *U_2, . . . , U_n* (one *of* 2^k n-tuples). We can therefore describe r as

r=U + e

Where e = e_1, e_2 ... , e_n is an error vector or error pattern introduced by the channel.The syndrome of r is defined as

$$S = rH^T$$

The syndrome is the result of a parity check performed on r to determine whether r is a valid member of the codeword set. If in fact. r is a member, the syndrome S has a value 0, If r contains detectable errors, the syndrome has some nonzero value. If r contains correctable errors, the syndrome has some nonzero value that can earmark the particular error pattern. The decoder, depending upon whether it has been implemented to perform FEC or ARQ, will then take actions to locate the errors and correct them (FEC), or else it will request a retransmission (ARQ). Combining Equations (6.34) and (6.35), the syndrome of r is seen to be

$$S = (U+ e)H^T$$
$$= UH^T + eH^T$$

However, $UH^T = 0$ *for* all members of the codeword set. Therefore,

$$S = eU^T \quad (3.1)$$

Equation (3.1), is evidence that the syndrome test, whether performed on either a corrupted code vector or on the error pattern that caused it, yields the same syndrome. An important property of linear block codes, fundamental to the decoding process, is that the mapping between correctable error patterns and syndromes is one to one.

3.5.6 ERROR CORRECTION

The last discussion gives us a clue that we not only can detect the error, but since there is a one-to-one correspondence between correctable error patterns and syndromes, we can correct such error patterns. Let us arrange the 2^n n-tuples that represent possible received vectors in an array, called the ***standard array,*** such that the first row contains all the codewords, starting with the all-zeros codeword, and the first column contains all the correctable error patterns.Each row, called a ***coset,***

consists of an error pattern in the first column,followed by the code words perturbed by that error pattern.. The array contains all2^n n-tuples in the space V_n. n-tuple appears in only one location-none are missing, and none are replicated. Each coset consists of 2^k n-tuples. Therefore, there are $(2^n/2^k)$
$= 2^{n-k}$ cosets.

U_1	U_2	.. ….U_i	……U_2^k
e_2	U_2 +e2	……Ui + e_2	…… U_2^k + e2
e_3	U_2 + e3	……Ui + e_2	..… U_2^k + e3
.	.	.	.
.	.	.	.
.	.	.	.
e_2^{n-k}	$U_2 + e_2^{n-k}$	$U_i + e_2^{n-k}$	$U_2^k + e_2^{n-k}$

The decoding algorithm calls for replacing a corrupted vector with a valid codeword from the top of the column containing the corrupted vector. Suppose that a codeword V_i (i = 1, . . . , 2^k) is transmitted over a noisy channel, resulting in a received (corrupted) vector $V_i + e_j$ If the error pattern e_j caused by the channel is a coset leader, where the index j = 1, . . . , 2^{n-k}, the received vector will be decoded correctly into the transmitted codeword V_i, If the error pattern is not a coset leader, then an erroneous decoding will result.[7]

3.5.7 ERROR CORRECTION CODING

The procedure for error correction decoding proceeds as follows

1. Calculate the syndrome of r using $S = rH^T$
2. Locate the coset leader (error pattern)e_j whose syndrome equals rH^T
3. This error pattern is assumed to be the corruption caused by the channel.
4. The corrected received vector, or codeword, is identified as U = r + *ej*. We can say that we retrieve the valid codeword by subtracting out the identified error; in modulo-2 arithmetic, the operation of subtraction is identical to that of addition

3.5.8 ERROR CORRECTION AND DETECTION CAPABILITY

The task of the decoder, having received the vector **r,** is to estimate the transmitted codeword V_i, The optimal decoder strategy can be expressed in terms of

the *maximum likelihood* algorithm as follows: Decide in favor of U_i if

$P(r \mid U_i) = \max P(r \text{ I } U_i)$ over all U_i

Since for the binary symmetric channel (BSC), the likelihood of U_i with respect to r is inversely proportional to the distance between r and U_i, we can write: Decide in favor of U_i if

$d(r. U_i) = \min d(r,U_i)$
over all U_i

In other words the decoder determines the distance between r and each of the possible transmitted codewords U_j and selects as most likely a U_i for which
$d(r,U_i) \leq d(r,U_j)$ for i,,j =1,……,M and $i \neq j$
where $M = 2^k$

In general, the *error-correcting capability t* of a code is defined as the maximum number of guaranteed correctable errors per codeword, and is written

$$t = \left[\frac{d_{\min} - 1}{2} \right]$$

where t is the largest integer not to exceed [d-1]/2
In general a t -error correcting (n,k) linear code is capable of correcting a total of 2^{n-k} error pattern. The Decoded Bit Error probability P_B (for BSC channel with transition probability p) depends on the particular code and decoder. It can be expressed by the following approximation [7]

$$P_B = \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j}$$

An (n,k) code is capable of detecting 2^n-2^k error patterns of length n.For large n where $2^k << 2^n$ only a small fraction of error patterns are undetected.

3.6 CONVOLUTIONAL CODING

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. [14]

3.7 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 3.1: 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 3.2 to see a picture of a convolutional encoder for a rate 1/2, K = 3, m = 2 code.

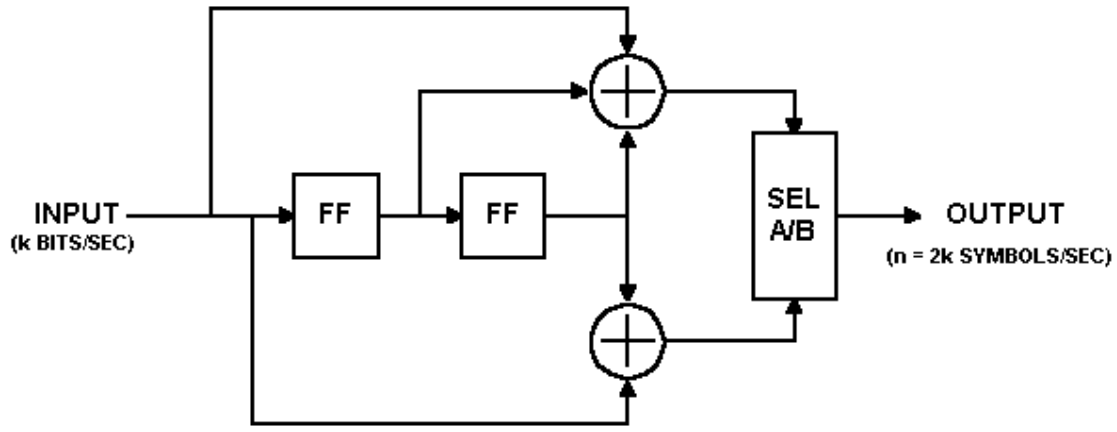

Figure 3.2: A typical convolutional encoder implementation for code rate 1/2

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.

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.

The table 3.2 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 3.2: State transition table

The table 3.2 is often called a state transition table. We'll refer to it as the `next state` table 3.3. 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 3.3: 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.

3.8 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

We will assume hard decision for Viterbi decoding. [7]

3.9 VITERBI DECODING

The single most important concept to aid in understanding the Viterbi algorithm is the trellis diagram. The figure 3.3 shows the trellis diagram for our example rate 1/2 K = 3 convolutional encoder, for a 15-bit message:

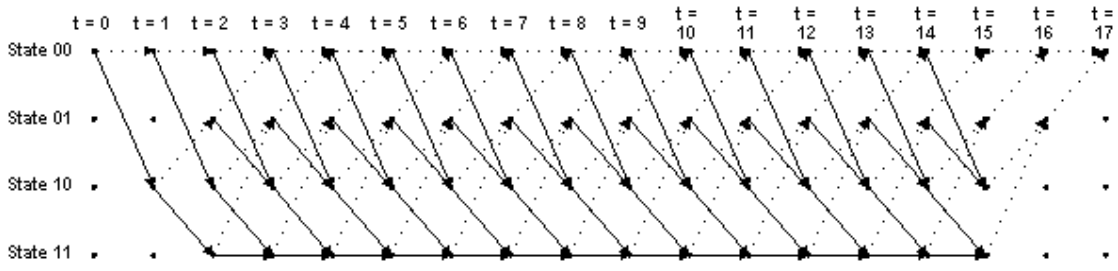

Figure 3.3: Trellis diagram

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 [7].

The figure 3.4 shows the states of the trellis that are actually reached during the encoding of our example 15-bit message:

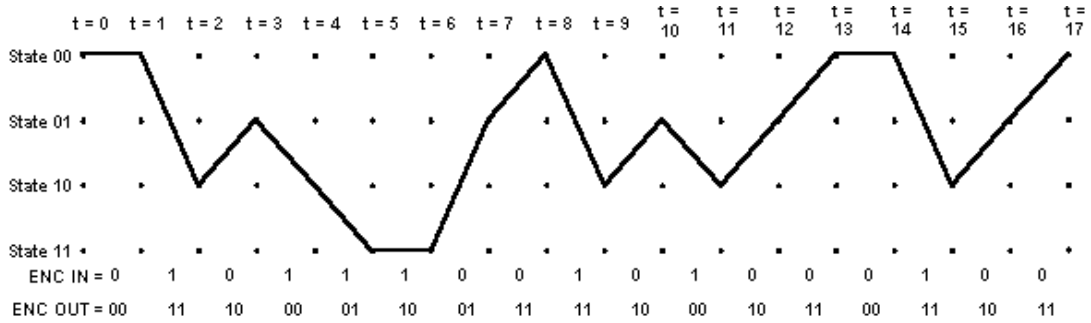

Figure 3.4: Actually reached path through trellis diagram

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 3.5

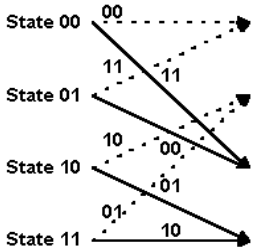

Figure 3.5: Diagram showing transition from one state to other

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 3.6.

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.

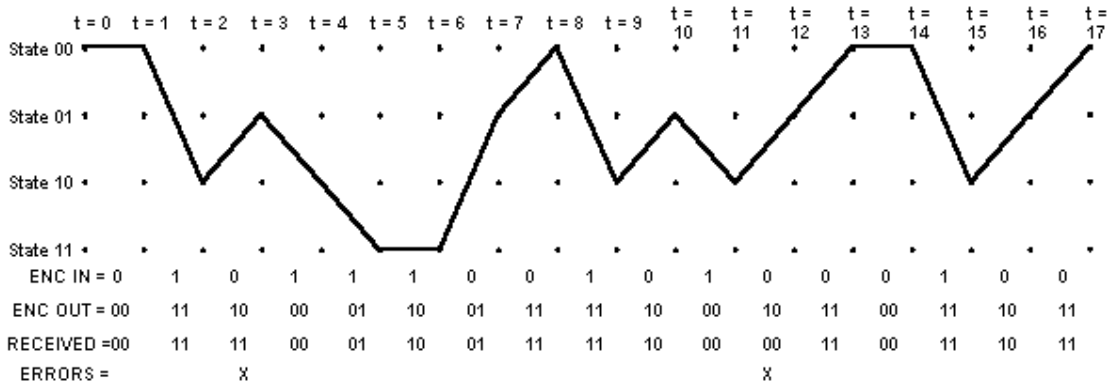

Figure 3.6: Path through the trellis diagram for received bits

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.[7]

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 3.7 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.

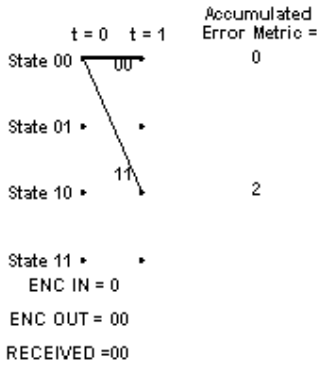

Figure 3.7: Trellis diagram at t=1

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 3.8 below shows the calculation of the accumulated error metric associated with each state, at t = 2.

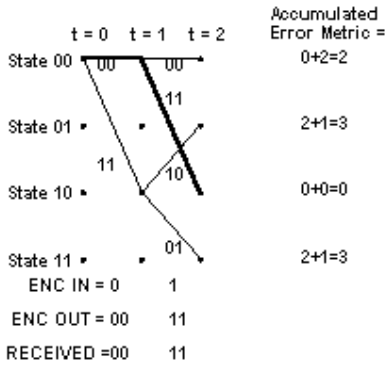

Figure 3.8: Trellis diagram at t=2

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 3.9 shows the results of processing t = 3:

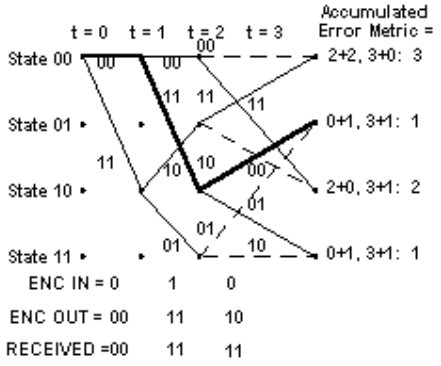

Figure 3.9: Trellis diagram at t=3

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 3.10:

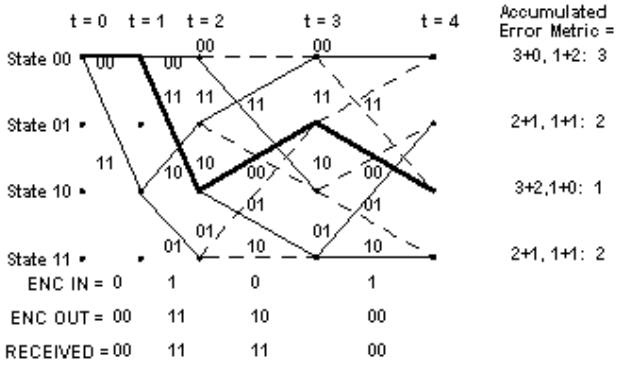

Figure 3.10: Trellis diagram at t=4

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 3.11:

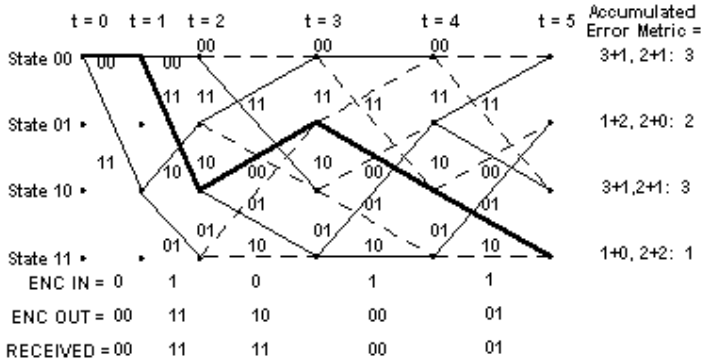

Figure 3.11: Trellis diagram at t=5

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:

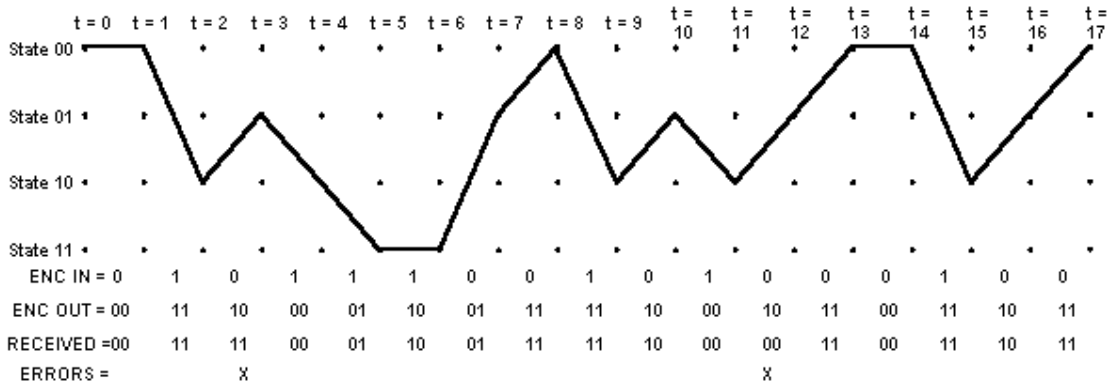

Figure 3.12: Trellis diagram for complete packet

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.[7]

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.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.

, 4.1 WHY MODULATE?

Digital modulation is the process by which digital symbols are transformed into waveforms that are compatible with the characteristics of the channel. 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. One might ask why it is necessary to use a carrier for the radio transmission of baseband signals. The answer is as follows. The transmission of EM fields through space is accomplished with the use of antennas. The size of the antenna depends on the wavelength λ and the application. For cellular telephones, antennas are typically $\lambda/4$ in size, where wavelength is equal to c/f and *c,* the speed of light, is 3 x 10^8 *m/s.* Consider sending a baseband signal (say, f = 3000 Hz) by coupling it to an antenna directly without a carrier wave. How large would the antenna have to be? Let us size it by using the telephone industry benchmark of $\lambda/4$ as the antenna dimension. For the 3,000 Hz baseband signal, $\lambda/4$ = 2.5 x 10^4 m= 15 miles. To transmit a 3,000 Hz signal through space. without carrier-wave modulation, an antenna that spans 15 miles would be required. However. if the baseband information is first modulated on a higher

frequency carrier, for example a 900 MHz.[7]. carrier, the equivalent antenna diameter would be about 8 cm. For this reason, carrier-wave or bandpass modulation is an essential step for all systems involving radio transmission.

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 [7].

The modulating process transforms the low frequency baseband signal to a bandpass signal around a carrier frequency as sketched in figure 4.1. The bandpass signal is the one actually transmitted to the receiver where the demodulator reconstructs the low-frequency baseband message.

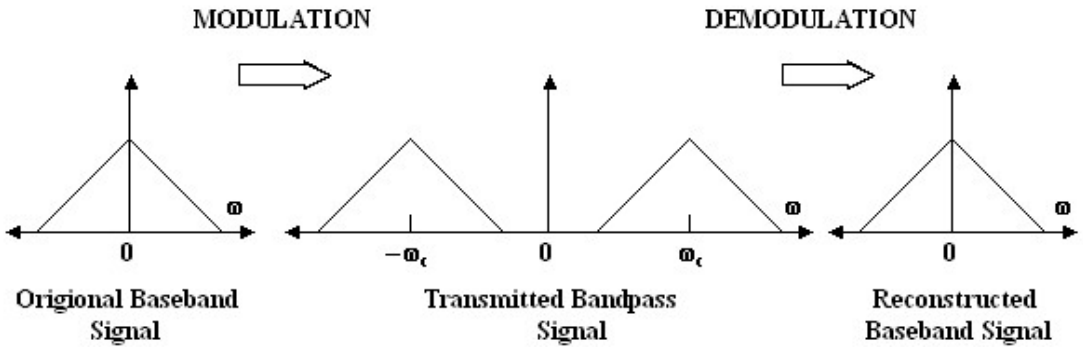

Figure 4.1: Power spectra for signals in the modulation and demodulation processes.

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

$$s(t) = A(t)\cos\ \theta(t) \tag{4.1}$$

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

$$\theta(t) = \omega_O t + \phi(t) \tag{4.2}$$

so that

$$s(t(= A(t)\cos\left[\omega_o t + \phi(t)\right] \tag{4.3}$$

Where W_O is the *radian frequency* of the carrier and $\theta(t)$ is the *phase.* The terms *f* and θ will each be used to denote frequency. When *f* is used, frequency in hertz is intended; when θ is used, frequency in radians per second is intended.

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

$$s_i(t) = \sqrt{\frac{2E}{T}}\cos(\omega_c t + \theta_i t) \qquad 0 \le t \le T, \qquad i = 1,2,\ldots,M \tag{4.4}$$

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

$$\theta_i = \frac{2\pi i}{M} \qquad i = 1,2,\ldots,M \tag{4.5}$$

$$E = \int_0^T {s_0}^2(t)\,dt \tag{4.6}$$

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 4.6 can be re-written in a slightly different form as

$$\begin{aligned} s_i(t) &= \sqrt{E}\left[\cos(\theta_i)\sqrt{\frac{2}{T}}\cos(2\pi\omega_c t) - \sin(\theta_i)\sqrt{\frac{2}{T}}\sin(2\pi\omega_c t)\right] \\ &= \sqrt{E}\left[\cos(\theta_i)\,\phi_1(t) - \sin(\theta_i)\,\phi_2(t)\right] \end{aligned} \tag{4.7}$$

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 $\phi_2(t)$ [7].

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 phases separated by π/4. The constellation diagram for QPSK signal using relation in equation 4.7 is shown in figure 4.2.

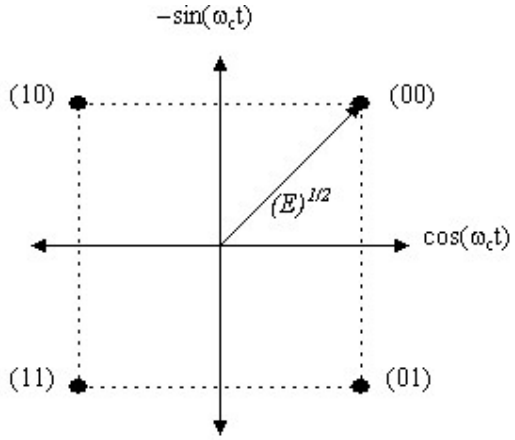

Figure 4.2: QPSK Constellation diagram

The illustrated constellation mapping in figure 4.2, known as Gray coding, has the property that adjacent signals are assigned binary sequences that differ in only one bit.

Now using the complex baseband representation the QPSK transmitter is drawn in figure 4.3. QPSK lookup table in figure 4.3 is simply the assignment of one phase to each of four symbols. Transmit filter is a filter which shapes the bit stream 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	$e^{j-3\pi/4}$

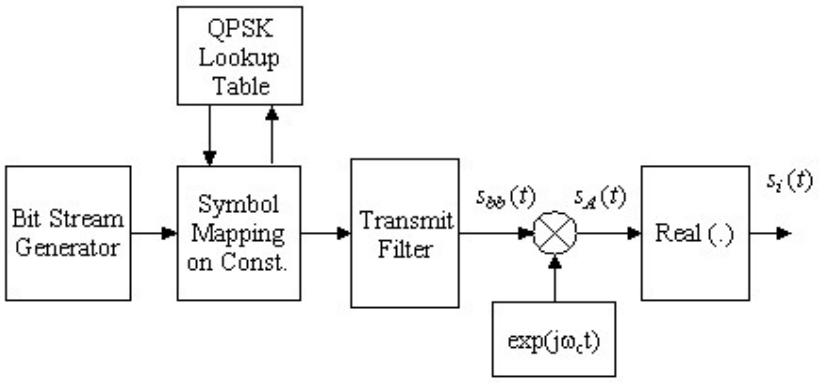

Figure 4.3: QPSK Transmitter

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 4.4.

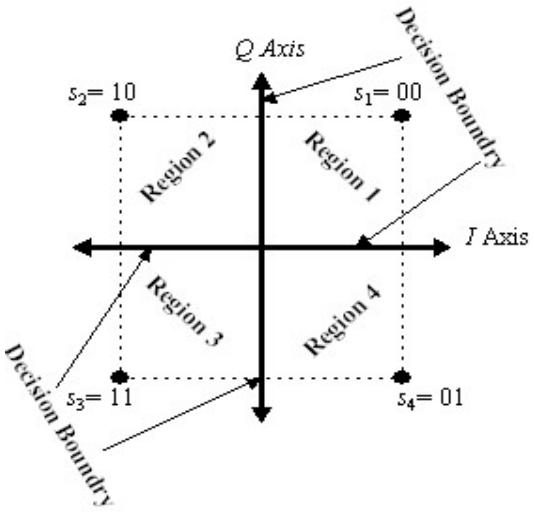

Figure 4.4: Decision regions for QPSK signal

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 4.5. 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.

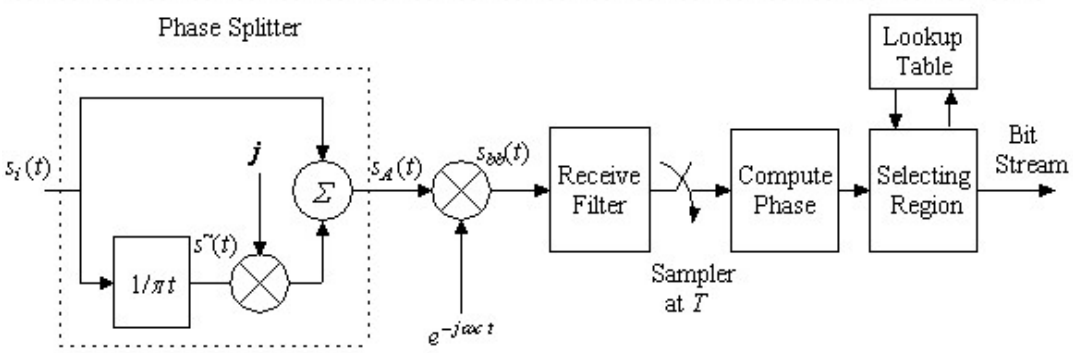

Figure 4.5: QPSK Receiver structure

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 4.6.

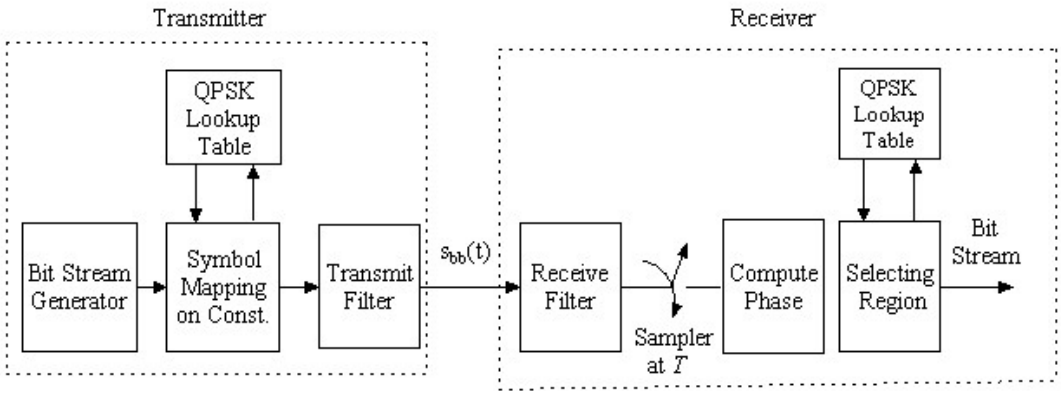

Figure 4.6: Baseband model for QPSK communication system

4.3 DETECTION IN ADDITIVE WHITE GAUSSIAN NOISE

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

$$R_x(\tau) = \begin{cases} N_0 & for\ \tau = 0 \\ 0 & elsewhere \end{cases} \tag{4.8}$$

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

$$S_x(\omega) = \int_{-\infty}^{\infty} R_x(\tau) e^{-j\omega\tau} d\tau = \frac{N_0}{2} \tag{4.9}$$

In this special case where the autocorrelation is a "spike" the Fourier transform results in a *constant* frequency spectrum as shown in figure 4.7. 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 (τ = 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.

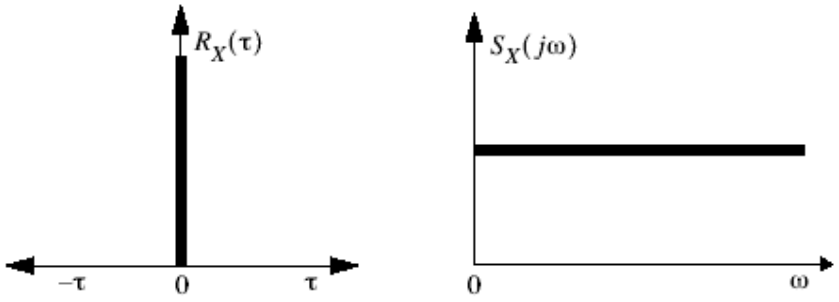

Figure 4.7: White noise shown in both the time (left) and frequency domain (right).

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 σ_0^2, where σ_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 [7]. 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 4.8 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.

$$z(T) = a_i + n_0 \tag{4.10}$$

The variance of the output noise (average noise power) is denoted by σ_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

$$\left(\frac{S}{N}\right)_T = a_i^2 \Big/ \sigma_O^2 \tag{4.11}$$

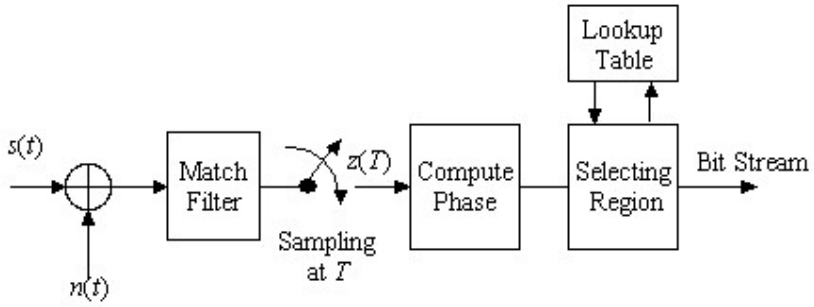

Figure 4.8 QPSK receiver with sampler at symbol rate *T* and match filter

We wish to find the filter transfer function $H_o(f)$ that *maximizes e*quation 4.11 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,

$$a_i(t) = \int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df \tag{4.12}$$

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

$$\sigma_o^2 = \frac{N_o}{2}\int_{-\infty}^{\infty} |H(f)|^2\,df \tag{4.13}$$

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

$$\left(\frac{S}{N}\right)_T = \frac{\left|\int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df\right|^2}{\frac{N_O}{2}\int_{-\infty}^{\infty}|H(f)|^2\,df} \tag{4.14}$$

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

$$\left|\int_{-\infty}^{\infty} f_i(x)f_2(x)dx\right|^2 \le \int_{-\infty}^{\infty}|f_1(x)|^2\,dx\int_{-\infty}^{\infty}|f_2(x)|^2\,dx \tag{4.15}$$

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

$$\left|\int_{-\infty}^{\infty} H(f)S(f)e^{j2\pi Tf}\,df\right|^2 \le \int_{-\infty}^{\infty}|H(f)|^2\,df\int_{-\infty}^{\infty}|S(f)|^2\,df \tag{4.16}$$

Substituting into Equation 4.15 yields

$$\left(\frac{S}{N}\right)_T \le \frac{2}{N_O}\int_{-\infty}^{\infty}|S(f)|_2\,df \tag{4.17}$$

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

$$E = \int_{-\infty}^{\infty}|S(f)|^2\,df \tag{4.18}$$

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 4.17 holds only if the optimum filter transfer function $H_O(f)$ is employed, such that

$$H(f) = H_O(f) = kS^*(f)e^{-j2\pi Tf} \tag{4.19}$$

$$h(t) = \Im^{-1}\left\{kS^*(f)e^{-j2\pi Tf}\right\} \tag{4.20}$$

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

$$h(t) = \begin{Bmatrix} ks(T-t) & 0 \le t \le T \\ 0 & elsewhere \end{Bmatrix} \tag{4.21}$$

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 4.21 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 4.9. Without the delay of T seconds, the response $s(–t)$ is unrealizable because it describes a response as a function of negative time.

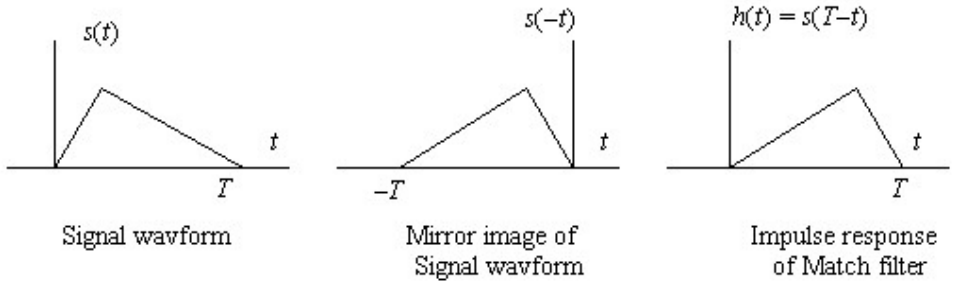

Figure 4.9: Impulse responses of received signal and match filter

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 4.10.

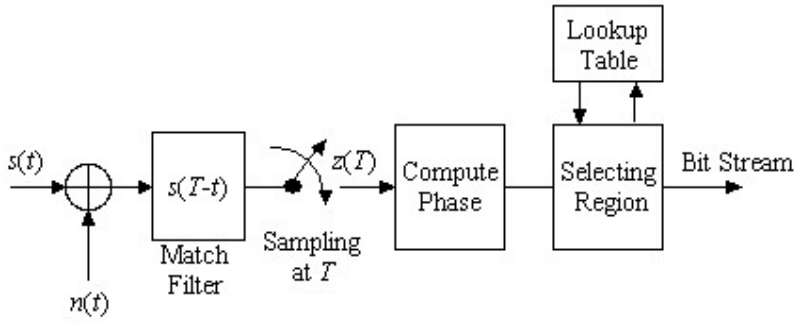

Figure 4.10: QPSK receiver with match filter

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.

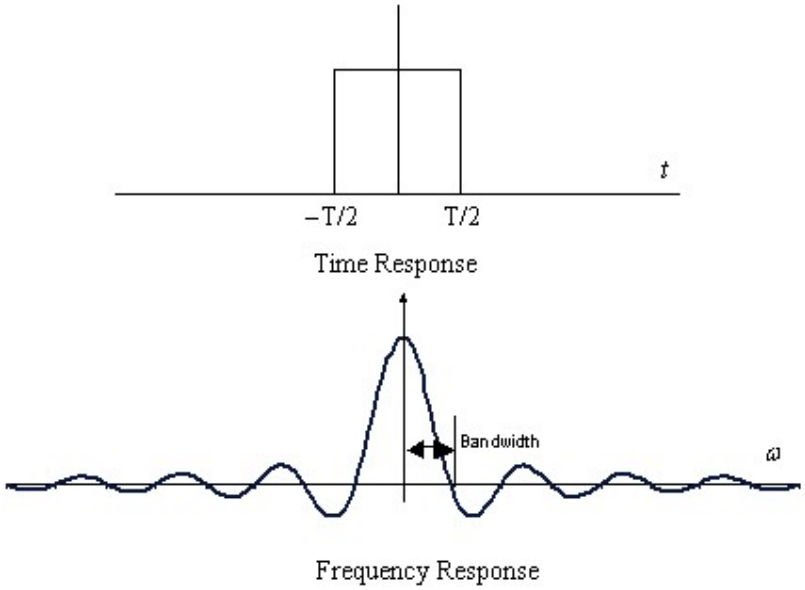

Figure 4.11: Time and frequency response of rectangular pulse

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 4.11.

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.

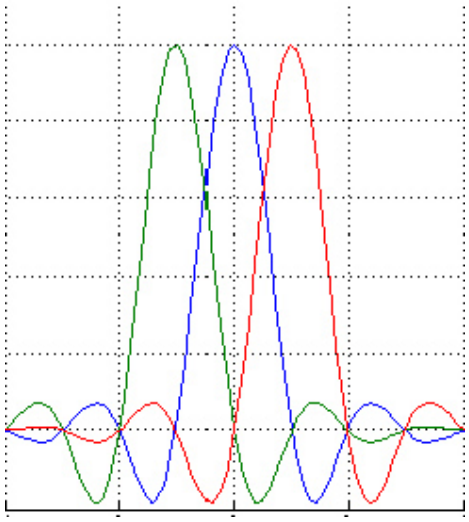

Figure 4.12: pulse shaping using sinc.

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

$$W = 1/2T \tag{4.22}$$

The raised cosine pulses have an adjustable bandwidth which can be varied from *W* to 2*W*. 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

$$\alpha = 1 - \frac{W}{W_0} \tag{4.23}$$

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

The factor □ 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.

$$W_0 = (1+\alpha)R_s \tag{4.24}$$

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.

$$h(t) = \frac{\sin(2\pi t / T_s)}{2\pi t} \frac{\cos(2\pi\alpha t)}{1-(2\alpha t/\pi)^2} \tag{4.25}$$

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+ α). For α = 0, the above equation reduces to the sinc pulse, and for α = 1, the equation becomes that of a pure square pulse.

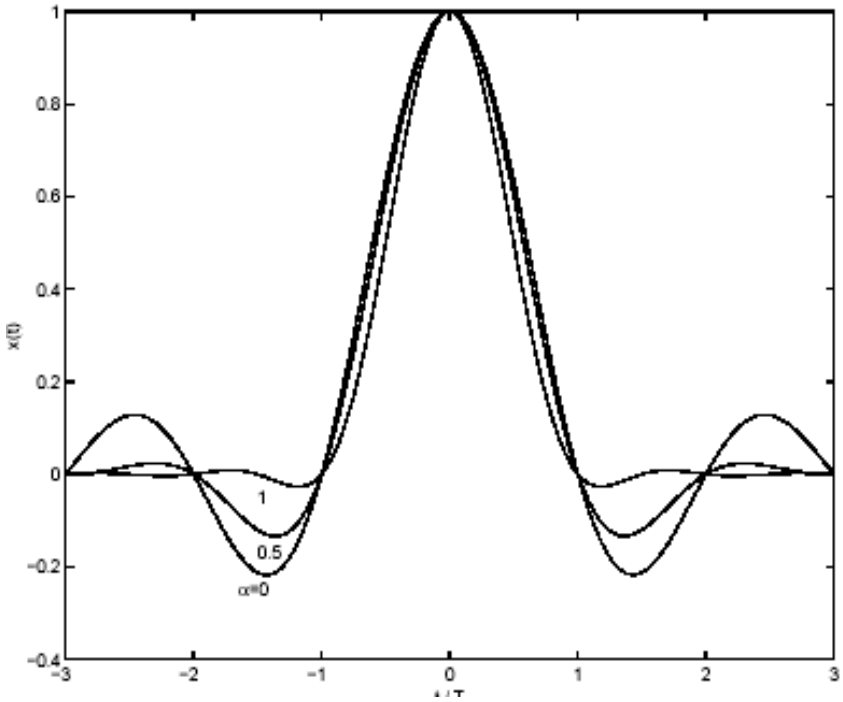

Figure 4.13: Impulse responses of raised cosine filter with $\alpha = 0,0.5,1$

In frequency domain, the relationship is given by

$$H(f)=\begin{cases}1 & for\ |f|\le\dfrac{(1-\alpha)}{2T_s}\\ \left\{\cos^2\dfrac{\pi T_s}{2\alpha}\left(|f|-\dfrac{(1-\alpha)}{2T_s}\right)\right\} & for\ \dfrac{(1-\alpha)}{2T_s}\le|f|\le\dfrac{(1+\alpha)}{2T_s}\\ 0 & for\ |f|>\dfrac{(1+\alpha)}{2T_s}\end{cases} \tag{4.26}$$

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 □. 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 + α). For α = 1 the above equation reduces to the sinc pulse, and for α = 1 the equation becomes that of a pure square pulse.

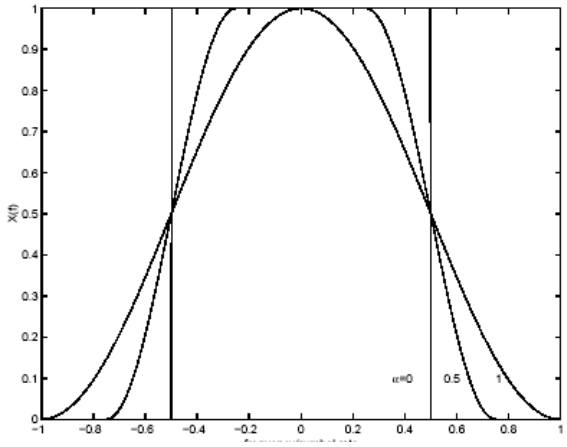

Figure 4.14: Frequency responses of raised cosine filter with $\alpha = 0,0.5,1$

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

$$H\ (f) = \begin{cases} 1 & for\ |f| \le \dfrac{(1-\alpha)}{2T_s} \\ \left\{\cos\dfrac{\pi T_s}{2\alpha}\left(|f| - \dfrac{(1-\alpha)}{2T_s}\right)\right\} & for\ \dfrac{(1-\alpha)}{2T_s} \le |f| \le \dfrac{(1+\alpha)}{2T_s} \\ 0 & for\ |f| > \dfrac{(1+\alpha)}{2T_s} \end{cases} \tag{4.27}$$

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

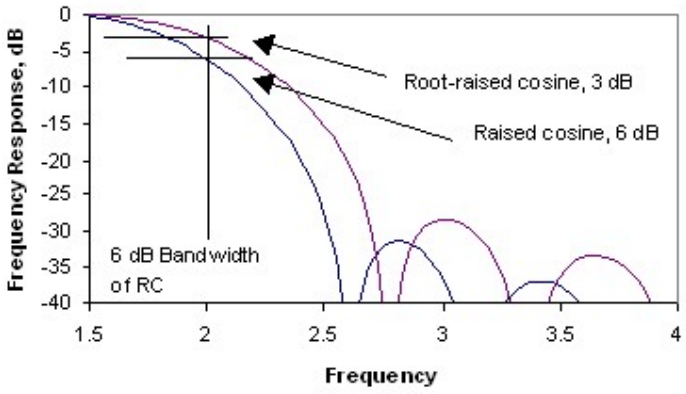

Figure 4.15: Frequency responses of raised cosine and rootraised cosine filters

By splitting the raised cosine into two root raised 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.

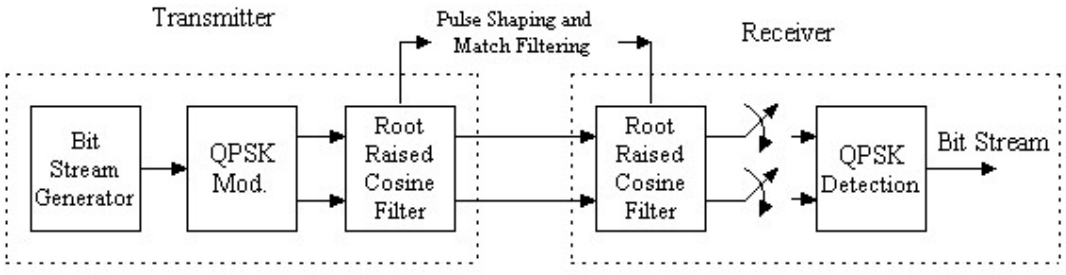

Figure 4.16: Baseband model of QPSK communication system

4.5 DISCRETE TIME MODEL OF COMMUNICATION SYSTEM

So far we have explained discrete partial analog communication system. In figure 4.15 the only analog portion is raised cosine filter. If we realize it in discrete time the complete system will become digital. 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

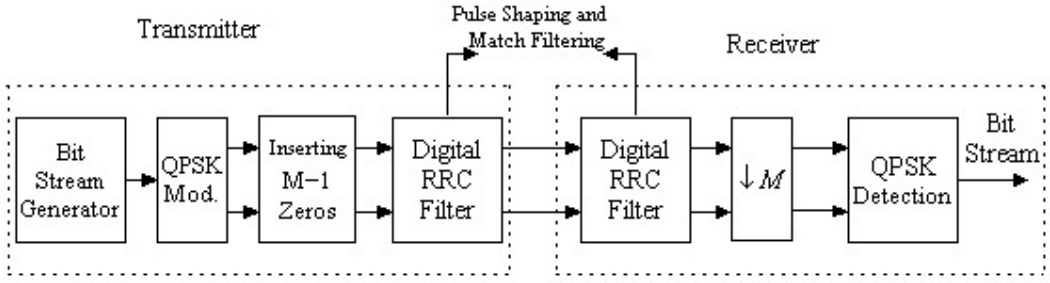

Figure 4.17: Discrete time baseband model of QPSK communication system

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

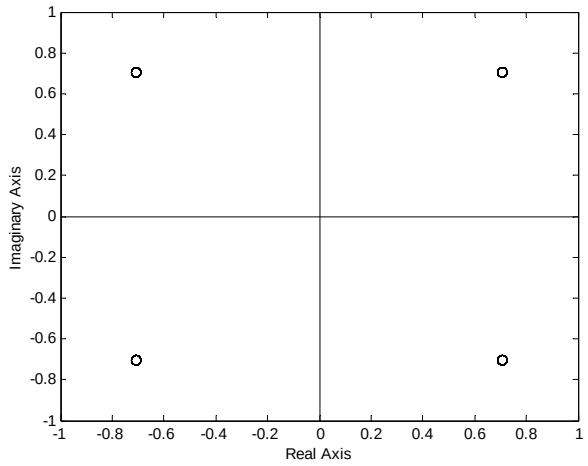

Figure 4.18: QPSK Constellation diagram at transmitter

QPSK constellation diagram after the coherent detection at the receiver is

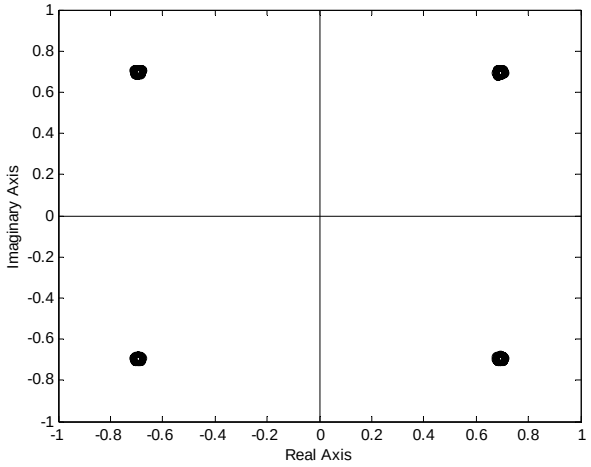

Figure 4.19: QPSK Constellation diagram at receiver

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.

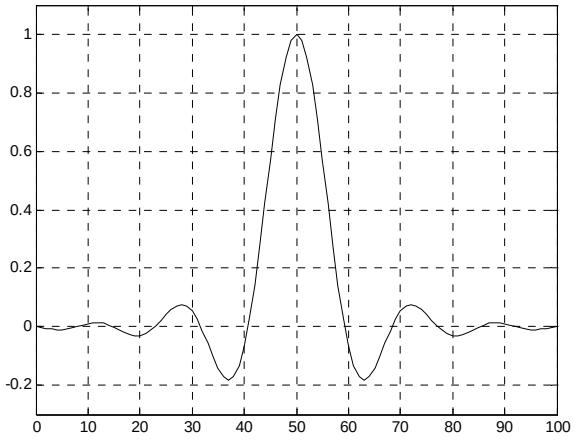

Figure 4.20: Impulse response of root raised cosine pulse

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

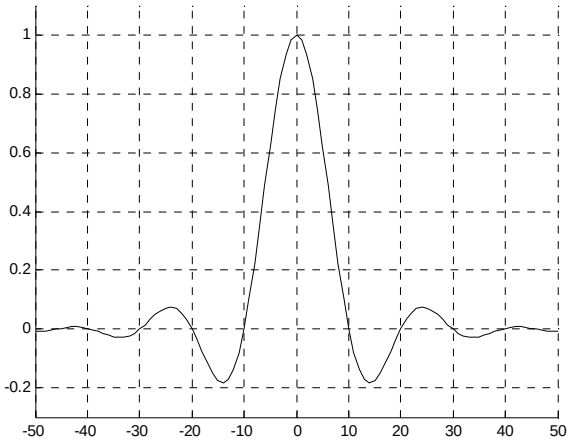

Figure 4.21: Impulse response of raised cosine pulse

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.

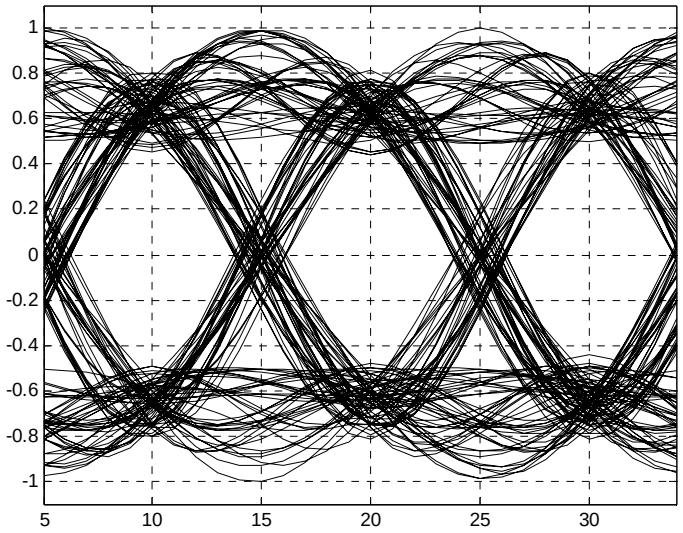

Figure 4.22: Eye diagram at transmitter (there is ISI)

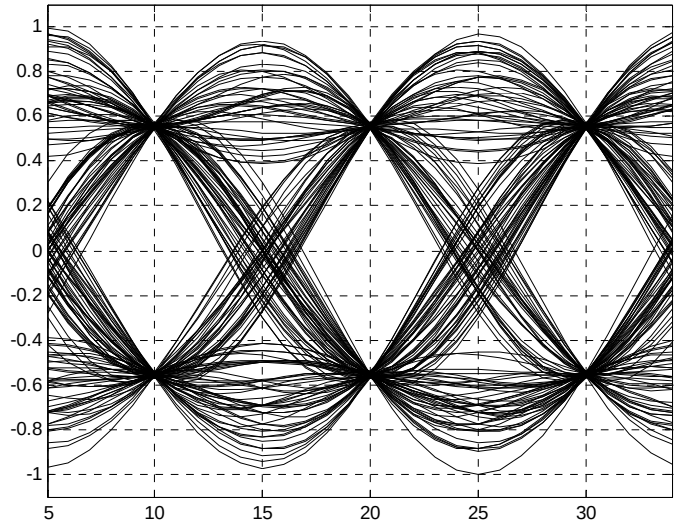

Figure 4.23: Eye diagram at receive (no ISI)

CHAPTER 5

SYNCHRONIZATION

5.1 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. 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 [4]. Before going into the discussion of carrier and timing recovery techniques it is necessary to provide a detailed overview of PLL.

5.2 PHASE LOCKED LOOPS

The basic PLL structure is shown in Figure 5.1. 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.

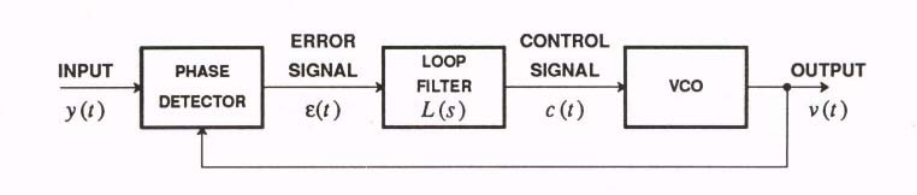

Figure 5.1: Basic structure of a continuous-time PLL.

5.3 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

$$y(t) = A_y \cos\left(w_v t + \theta(t)\right) \tag{5.1}$$

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

$$v(t) = A_v \cos\left(w_v t + \phi(t)\right) \tag{5.2}$$

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.

5.3.1 IDEAL PHASE DETECTOR

Assuming forms (5.1) and (5.2) the output of an ideal phase detector is

$$\varepsilon(t) = W\left(\theta(t) - \phi(t)\right) \tag{5-3}$$

where the function $W(.)$, shown in Figure 5.2, 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.

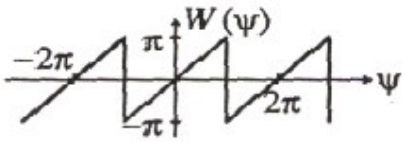

Figure 5.2: An ideal phase detector which can detect phase errors ψ modulo 2π.

5.3.2 IDEAL VCO

The ideal VCO, with properties summarized in Figure 5.3, produces the output (5.2), which has instantaneous frequency

$$\frac{d}{dt}\left[w_v t + \phi(t)\right] = w_v + \frac{d\phi(t)}{dt} \tag{5.4}$$

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

$$\frac{d(\phi(t))}{dt} = c(t) \tag{5.5}$$

Taking the Laplace transform of equation (7.5),

$$s\Phi(s) = C(s) = L(s)E(s) \tag{5.6}$$

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

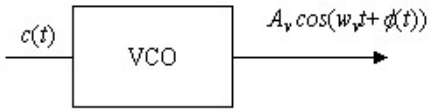

Figure 5.3: An ideal VCO

5.3.3 PHASE AND AVERAGE-FREQUENCY LOCK

The ideal PLL is phase locked if

$$\phi(t) = \theta(t) + \phi \tag{5.7}$$

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

$$\phi(t) = Kt \tag{5.8}$$

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,

$$\theta(t) = w_0 t + \theta \tag{5.9}$$

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.[4]

5.3.4 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

$$,\left|\theta(t) - \phi(t)\right| < \pi \tag{5.10}$$

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

$$\varepsilon(t) = \theta(t) - \phi(t) \tag{5.11}$$

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 (5. 11),

$$E(s) = \Theta(s) - \Phi(s) \tag{5.12}$$

and from (5.6),

$$E(s) = \frac{s\Phi(s)}{L(s)} \tag{5.13}$$

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

$$\frac{\Phi(s)}{\Theta(s)} = \frac{L(s)}{L(s) + s} \tag{5.14}$$

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 (5. 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) [4].

Transfer function for the typical first order PLL is

$$L(s) = K_L \frac{s + K_1}{s} \tag{5.15}$$

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

$$\frac{\Phi(s)}{\Theta(s)} = \frac{K_L K_1 + K_L s}{K_L K_1 + K_L s + s^2} \tag{5.16}$$

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.

5.3.5 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

$$\varepsilon_{ss} = \lim_{t \to \infty} \varepsilon(t) \tag{5.17}$$

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,

$$\varepsilon_{ss} = \lim_{s \to 0} sE(s) \tag{5.18}$$

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

$$E(s) = \frac{s\Theta(s)}{L(s) + s} \quad , \quad \varepsilon_{ss} = \lim_{s \to 0} \frac{s^2 \Theta(s)}{L(s) + s} \tag{5.19}$$

5.4 DISCRETE TIME PLL

In digital communications systems, especially for software radio applications, analog continuous-time PLLs like those discussed in section 5.3 are rare. So we have to have the discrete time equivalent for the continuous time PLLs [4].

A typical all digital PLL is shown in figure 5.4**.** 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.

5.4.1 PHASE ERROR DETECTOR

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

$$\varepsilon_k = W(\theta_k - \phi_k) \tag{5.20}$$

Where $W(.)$ is shown in figure 5.2.

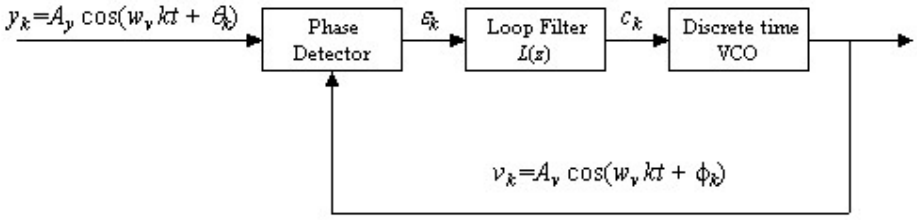

Figure 5.4: A typical discrete time PLL

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.

5.4.2 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 5.5. For small phase errors, the phase detector is approximately linear,

$$\varepsilon(t) \approx A_v A_y \left[\theta(t) - \phi(t)\right] \tag{5.21}$$

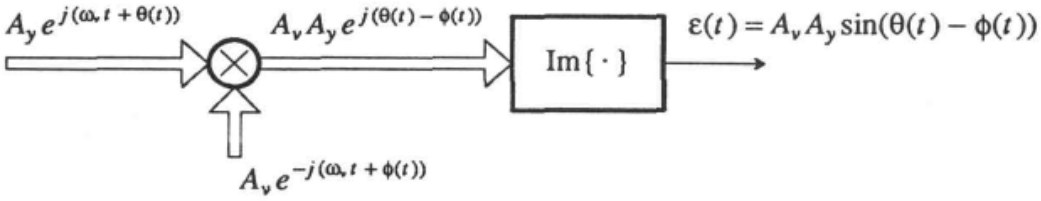

Figure 5.5: A simple phase error detector for complex signals

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. [4]

5.5 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 5.6.

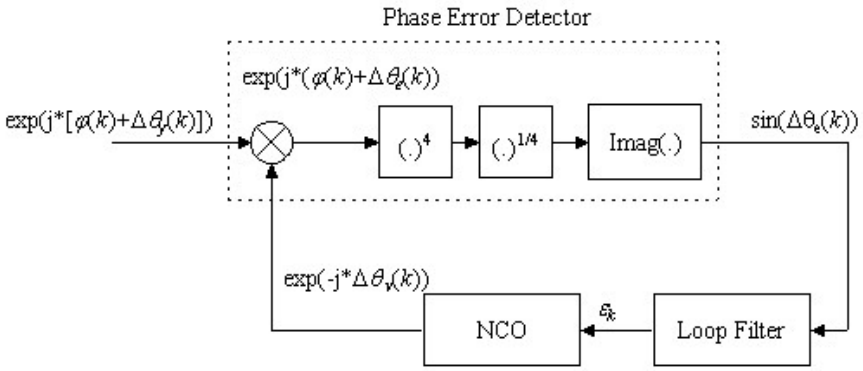

Figure 5.6: NDA Phase recovery loop for one sample per symbol

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

$$\phi_i = \frac{\pi}{4} i \qquad where\ i = \pm 1, \pm 3 \tag{5.22}$$

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.

5.5.1 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

$$y(k) = \exp(j \times [\phi(k) + \Delta\theta_y(k)]) \tag{5.23}$$

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

$$\theta_y(k) = \phi(k) + \Delta\theta_y(k) \tag{5.24}$$

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

$$\theta_e(k) = \theta_y(k) - \theta_v(k) \tag{5.25}$$

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

$$\begin{aligned} z(k) &= \left[\exp(j \times \theta_e(k))\right]^4 \\ &= \left[\exp j \times \left(\Delta\theta_y(k) - \Delta\theta_v(k)\right)\right]^4 \end{aligned} \tag{5.26}$$

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.

$$r(k) = \left[z(k)\right]^{1/4} \tag{5.27}$$

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

$$e(k) = \Im(r(k)) \tag{5.28}$$

The open loop characteristic curve for PED which is also called S-curve is shown in figure below . This is the same as shown in figure 5.2.

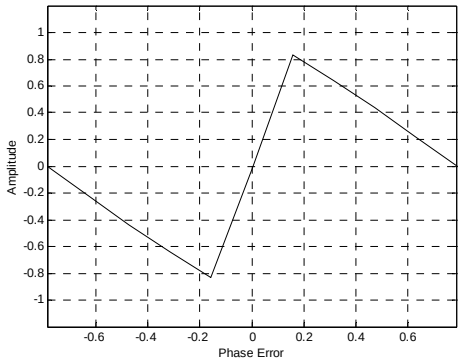

Figure 5.7: Characteristic S-Curve for PED from pi/4 to –pi/4

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.

5.5.2 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

$$\theta_v(k+1) = \Delta\theta_v(k) + K(\varepsilon(k)) \tag{5.29}$$

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

5.5.3 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.

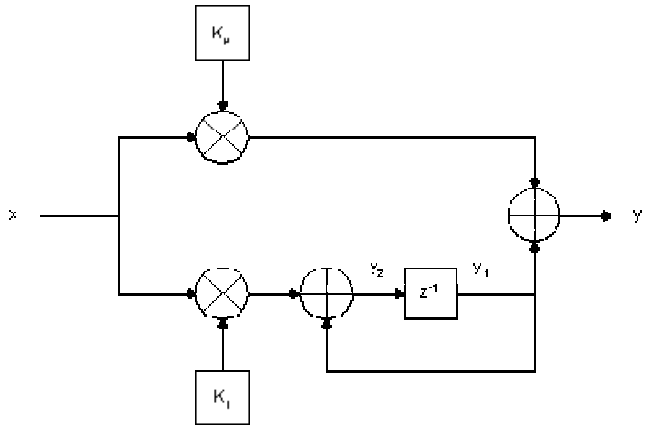

Figure 5.8: First order digital loop filter

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:

$$y(n) = K_p * x(n) + K_i * x(n-1) + y(n-1) - K_p * x(n-1) \tag{5.30}$$

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

$$K_p = \frac{4B_L T_s}{1+1/4\zeta^2} \tag{5.31}$$

$$K_i = \left(\frac{2B_L T_s}{\zeta+1/4\zeta}\right)^2 \tag{5.32}$$

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.

5.6 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. [4]

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

$$y(t;\varepsilon) \equiv \sum_m a_m g(t - mT - \varepsilon T) + n(t) \tag{5.33}$$

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

$$r(t;\varepsilon') = \sqrt{2}K_r \sin\left(\frac{2\pi}{T}(t - \varepsilon' T)\right) \tag{5.34}$$

and is multiplied with the noisy QPSK signal $y(t;\varepsilon)$ as shown in Figure 5.9. Taking into account equation (i), the timing error detector output signal equals

$$x(t;\varepsilon,\varepsilon') = \left[\sum_m a_m g(t - mT - \varepsilon T) + n(t)\right]\sqrt{2}K_r \sin\left(\frac{2\pi}{T}(t - \varepsilon' T)\right) \qquad (5.35)$$

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 .

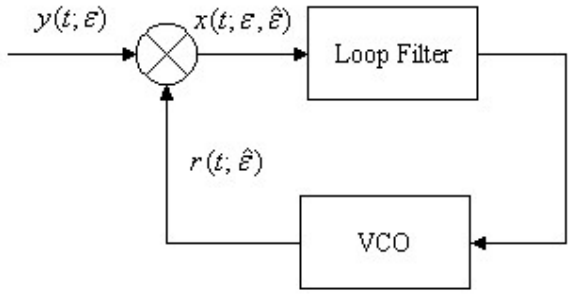

Figure 5.9: Ordinary PLL operating on baseband QPSK signal

5.6.1 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.

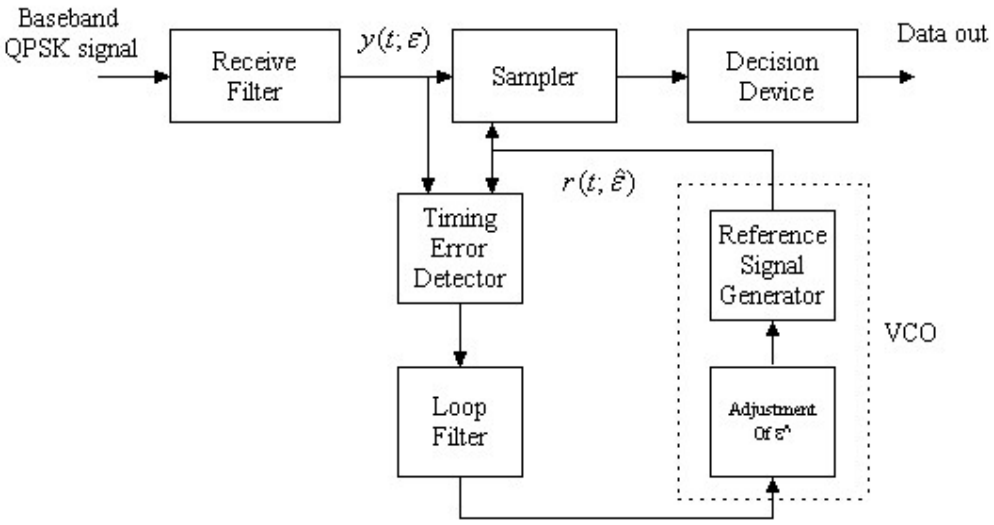

Figure 5.10: General Error Tracking Synchronizer

Figure 5.11 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).

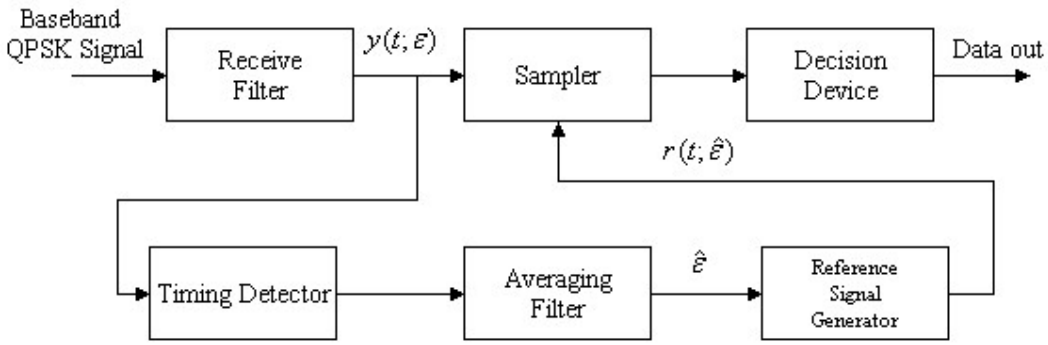

Figure 5.11: General Feedforward Synchronizer

5.7 TIMING RECOVERY USING EARLY LATE GATE

Among the most popular of the closed-loop symbol synchronizers is the early late-gate synchronizer. An example of such a synchronizer is shown schematically in Figure 7.13. The synchronizer operates by performing two separate integrations of the incoming signal energy over two different (T - d) second portions of a symbol interval. The first integration (the early gate) begins integration at

the loop's best estimate of the beginning of a symbol period (the nominal time zero) and integrates for the next (T -- d) seconds. The second integral (the late gate) delays the start of its integration for d seconds, and then integrates to the end of the symbol period (the nominal time T). The difference in the absolute values of the outputs of these two integrations, y_1 and y_2, is a measure of the receiver's symbol timing error, and it can be fed back to the loop's timing reference to correct loop timing.

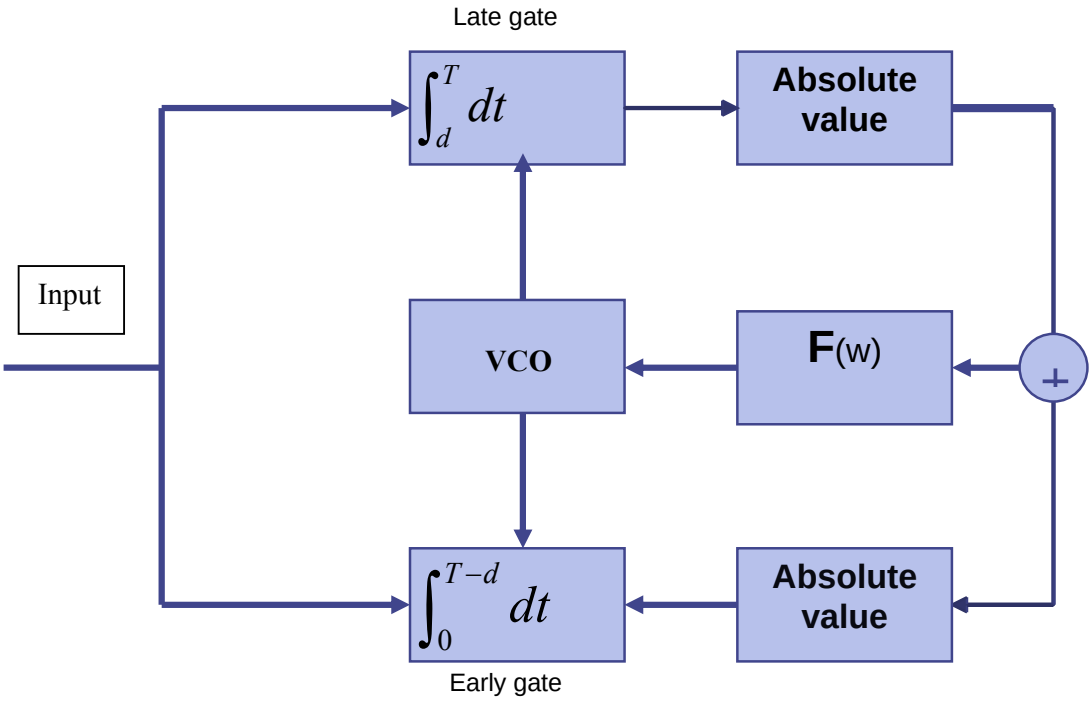

Figure 5.12 Early Late Gate

The action of the early/late-gate synchronizer can be understood by referring to Figure 5.13. In the case of perfect synchronization, Figure 5.13a shows that both gates are entirely within a signal symbol interval. In this case, both integrators will accumulate the same amount of signal, and their difference (the error signal *e,* in Figure 5.13) is zero. Thus, when the device is synchronized, it is stable-there is no tendency to drive itself away from synchronization. The case shown in Figure 5.13b is for a receiver whose data clock is early relative to the incoming data. In this case the first portion of the early gate falls in the previous bit interval, while the late gate is still entirely inside the current symbol. The late-gate integrator will accumulate signal

over its entire *(T - d)* integration interval, as in the case in Figure 5.13a; but the early-gate integrator will end up with energy accumulated only over *[(T - d)* - 2 □)], where □ is the portion of the early-gate interval falling in the previous bit interval. Thus, for this case, the error signal will be *e* = -2 □ , which will lower the input voltage to the VCO in Figure 5.12. This will reduce the VCO output frequency and retard the receiver's timing to bring it back toward the incoming signal's bit timing. Using Figure 5.13 as a guide, it can be seen that if the receiver's timing had been late, the amounts of energy integrated in the early gate and late gate would be reversed, as would the sign of the error signal. Thus, late receiver timing produces an increase in the VCO input voltage, increasing the output frequency and advancing the receiver's timing toward that of the incoming signal.

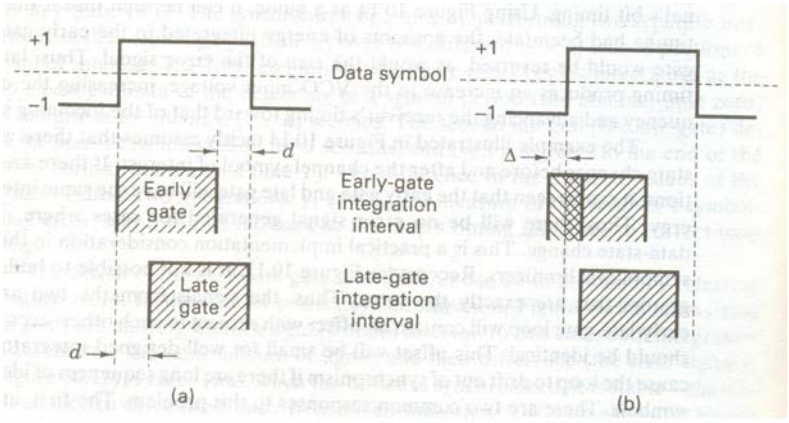

Fig 5.13 a)correct timing recovery b)Early receiver timing

The example illustrated in Figure 5.13 tacitly assumes that there will be data state changes before and after the channel symbol of interest. If there are no transitions, it can be seen that the early gate and late gate will have the same integrated energy. Thus, there will be no error signal generated for cases where there is no data-state change. This is a practical implementation consideration in the use of all symbol synchronizers. Reconsider Figure 5.12. It is not possible to build two integrators that are exactly the same. Thus, the signals. from the two arms of the early/late-gate loop will contain an offset with respect to each other, even when they should be identical. This offset will be small for well-designed integrators but will cause the loop to drift out of synchronism if there are long sequences of identical data symbols. There are two common responses to this problem. The first, and perhaps most obvious, is to

format the data in a manner which ensures that there will be no transition less intervals that are long enough to allow the loop to break lock. The second response is to modify the loop design so that it contains a single integrator.

5.8 SIMULATION RESULTS

5.8.1 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

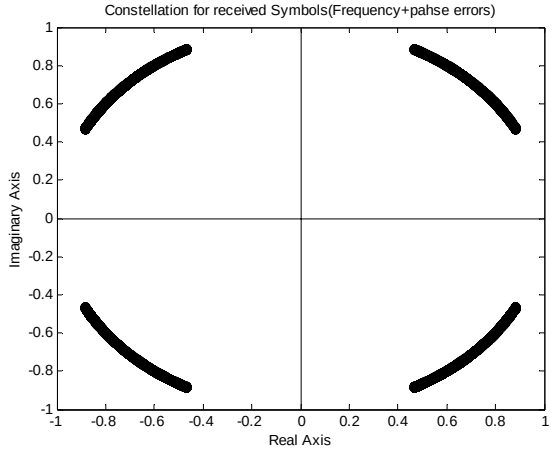

Figure 5.14: Constellation for received QPSK symbols

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

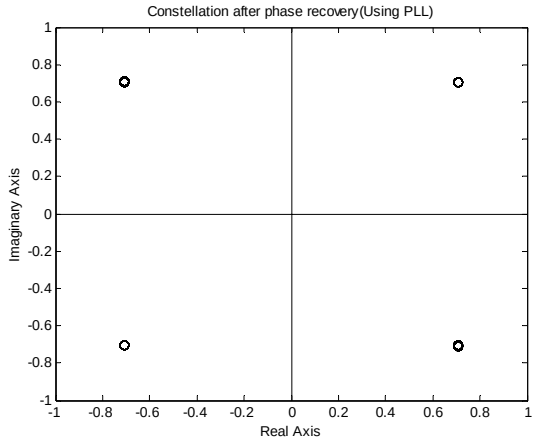

Figure 5.15: Constellation after phase recovery

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.

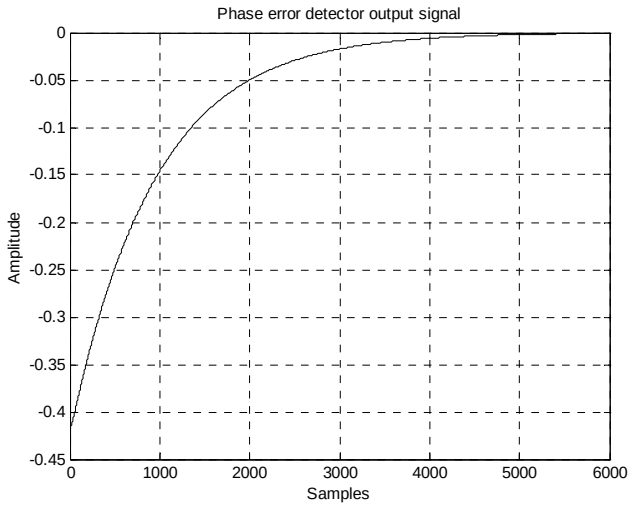

Figure 5.16: Output signal from the loop filter

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.

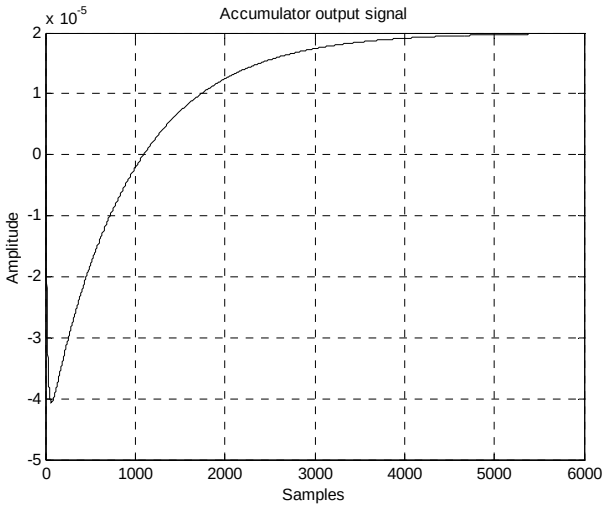

Figure 5.17: Accumulator output signal

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

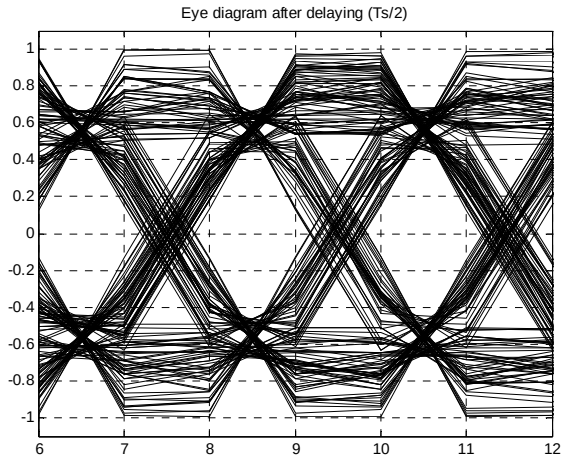

Figure 5.18: Eye diagram after delaying T/2.

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.

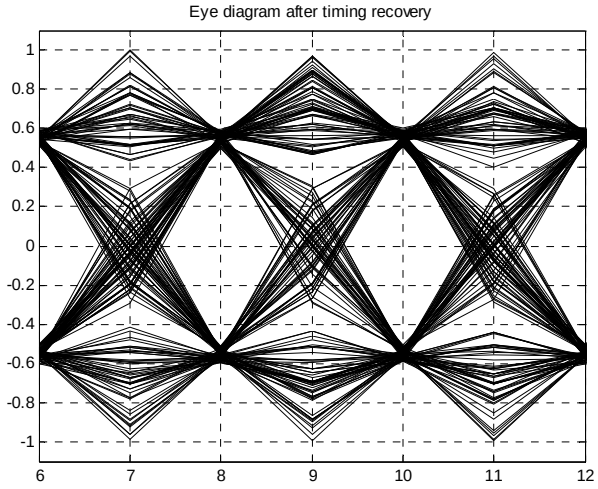

Figure 5.19: Eye diagram after timing recovery.

CHAPTER 6

CHANNEL EQUALIZATION

6.1 INTRODUCTION

Intersymbol interference and Communication channels are inseparable. Communication channels being susceptible to Intersymbol Interference (ISI), without channel equalization, the utilization of the channel bandwidth becomes inefficient. Objective of channel equalization is to compensate 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".*

Sources of corruption in the received symbol are due to the dispersive transmission medium (e.g. telephone cables) and the multipath effects in the radio channel. Figure 6.1 where the equalizer is incorporated within the receiver while the channel introduces intersymbol interference. If the additive noise could be ignored, then the task of equalizer would be rather straightforward. For a channel *H(z),* an equalizer with transfer function $W(z) = II\ H(z)$ could do the job perfectly, as this results in an overall channel-equalizer transfer function $H(z)\ W(z) = 1$, which implies that the transmitted data sequence, *s(n),* will appear at the detector input without any distortion. Unfortunately, this is an ideal situation which cannot be used in most of the practical applications.

We note that the inverse of the channel transfer function, i.e. 1/ *H(z),* may be non-causal if *H(z)* happens to have a zero outside the unit circle, thus making it unrealizable in practice. This problem is solved by selecting the equalizer so that $H(z)\ W(z) \approx z^{-\square}$ where $\square$ is an appropriate integer delay. This is equivalent to saying that a delayed replica of the transmitted symbols appears at the equalizer output.

We also note that the choice of $W(z) = 1/H(z)$ may lead to a significant enhancement of the additive noise, $v(n)$, in those frequency bands where the magnitude of $H(z)$ is small (i.e. $1/H(z)$ is large). Hence, in choosing an equalizer, $W(z)$, we should keep a balance between residual ISI and noise enhancement at the equalizer output. A Wiener filter is a solution with such a balance

Figure 6.3 presents the details of a baseband transmission system, equipped with an adaptive equalizer. The equalizer is usually implemented in the form of a transversal filter. Initial training of the equalizer requires knowledge of the transmitted data symbols (or, to be more accurate, a delayed replica of them) since they should be used as the desired signal samples for adaptation of the equalizer tap weights. This follows from the fact that the equalizer output should ideally be the same as the transmitted data symbols. We thus require an initialization period during which the transmitter sends a sequence of training symbols that are known to the receiver. This is called the *training mode.*

At the end of the training mode the tap weights of the equalizer would have converged close to their optimal values. The detected symbols would then be similar to the transmitted symbols with probability close to one. Hence, from then onwards, the detected symbols can be treated as the desired signal for further adaptation of the equalizer so that possible variations in the channel can be tracked. This mode of operation of the equalizer is called the *decision directed mode.* The decision directed mode successfully works as long as the channel variation is slow enough so that the adaptation algorithm is able to follow the channel variations satisfactorily. This is necessary for the purpose of ensuring low symbol error rates in detection so that these symbols can still be used as the desired signal.[5]

. 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.

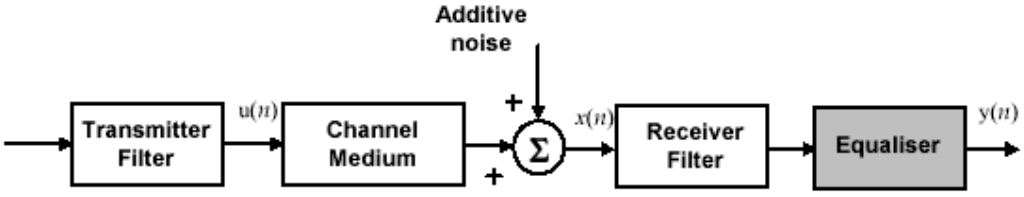

Figure 6.1: A typical communication system.

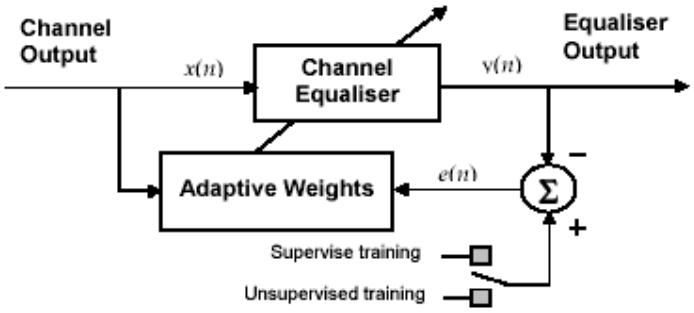

Figure 6.2: A simple linear channel equalizer configuration.

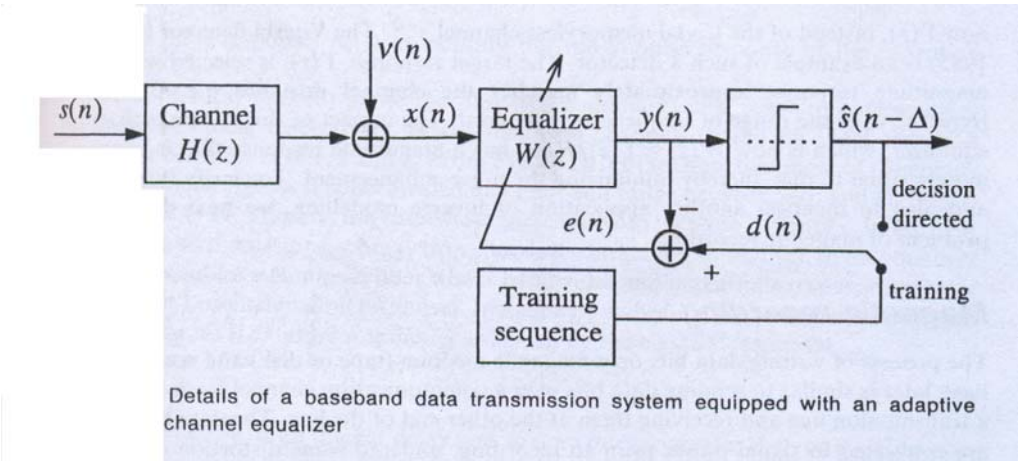

Figure 6.3:baseband system with adaptive equalizer.

Equalization does not have the effect of completely removing the channel distortions rather it provides the receiver enough information which is necessary to make a decision.

6.2 COMMUNICATION SYSTEM MODEL WITH EQUALIZER

6.2.1 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 6.3, 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.

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

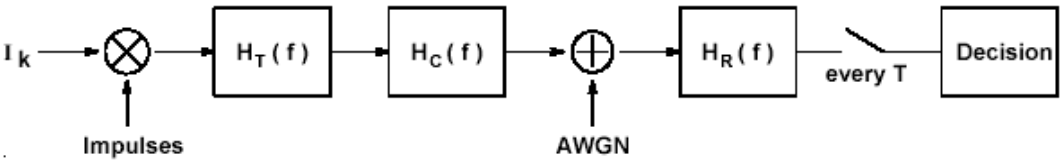

Figure 6.4: Continuous-time communication model over a multipath dispersive channel

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 (6.3) 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 6.5. 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.

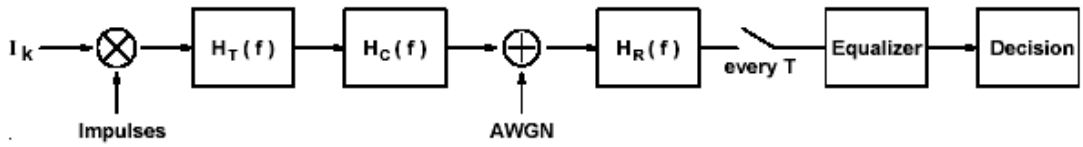

Figure 6.5: Communication system with equalizer

6.2.2 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 6.6
- 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 6.6 from the information source to the sampler as a digital filter.

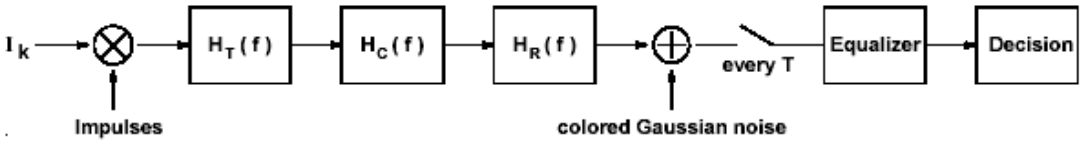

Figure 6.6: Equivalent communication system with colored Gaussian noise

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 6.7, 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.

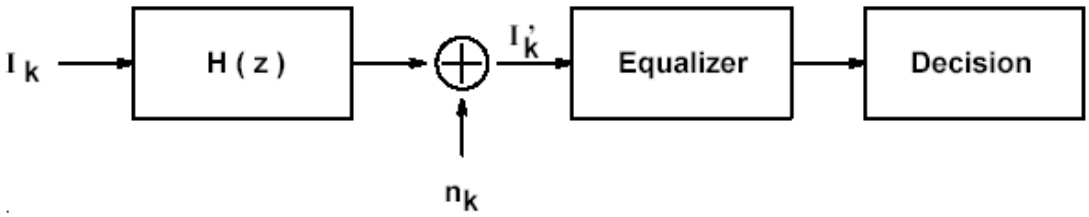

Figure 6.7: Equivalent discrete-time communication system model with colored noise

- 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 6.8. 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 6.9, in which $\{\overline{n}_k\}$ is an AWGN sequence.

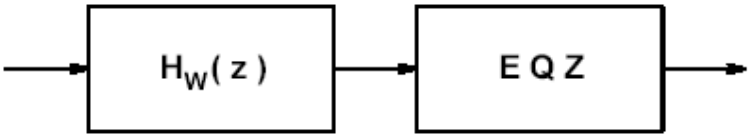

Figure 6.8: Typical equalizer

- 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 6.10. The output sequence $\{\widetilde{I}_k\}$ is given by:

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

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

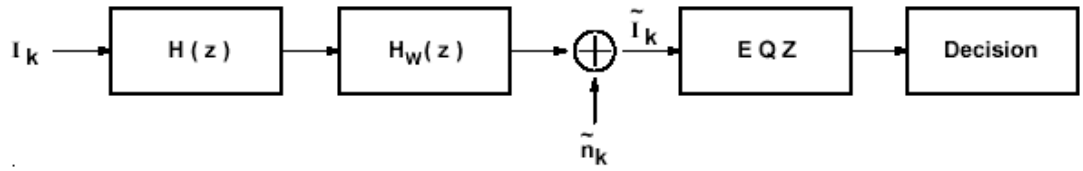

Figure 6.9: Equivalent discrete-time communication system model with white noise

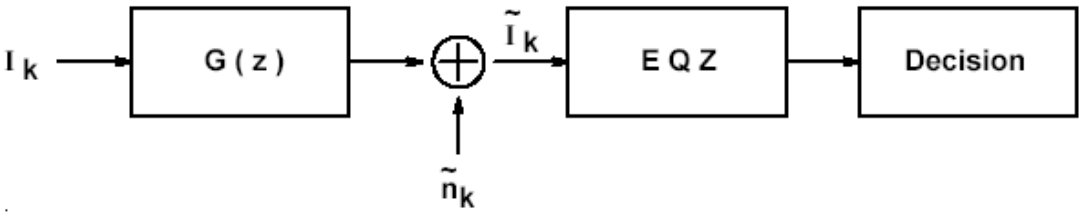

Figure 6.10: Equivalent discrete-time white-noise linear filter model

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 Ik and the AWGN sequence with small variance.[4]

6.3 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 6.11 types of equalizers, their structures and the algorithm for learning channel environment is given.

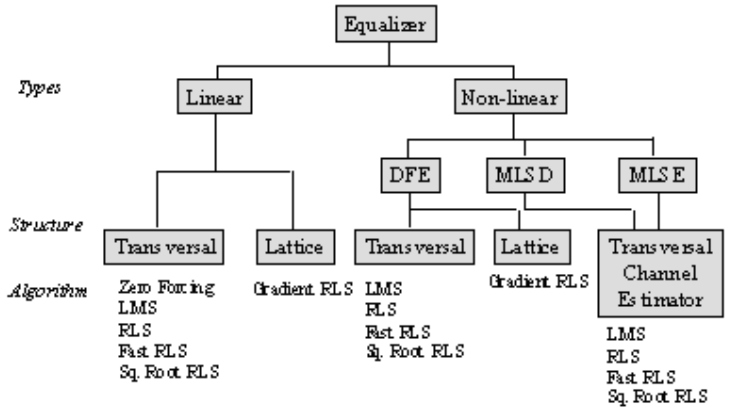

Figure 6.11: Classification of equalizers

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	Performanc
Linear Transversal Equaliz	Low	Cheap	Suboptimal
Decision Feedback Equaliz	Medium	Averag	Suboptimal
Maximum Likelihood Sequence Estimation	High	High	Optimal
Maximum Likelihood Symbol Detection	High	High	Optimal

6.4 ZERO FORCING EQUALIZER

It is really not necessary to eliminate or minimize ISI with neighboring pulses for all t. All That is needed is to eliminate or minimize interference with neighboring pulse at their respective sampling instants only, because the decision is based only on sample values. This can be accomplished by the transversal filter equalizer encountered called ZERO FORCING EQUALIZER, which forces the equalizer output pulse to have zero values at the sampling instants.Figure 6.12 shows a diagram of a ZFE.The time delay between successive taps is chosen to be T_b the interval between pulses.

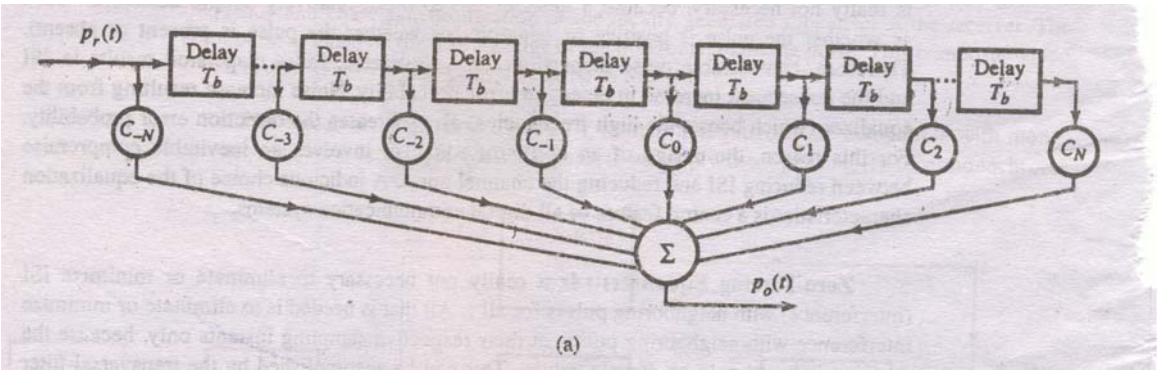

Fig.6.12(a) Zero Forcing Equalizer

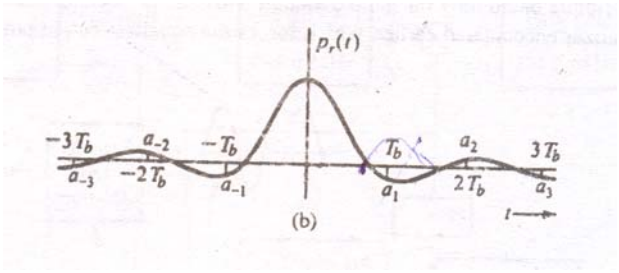

Fig 6.12(b) Before Equalization

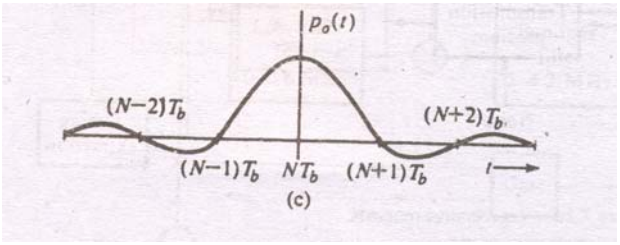

Fig 6.12(c) After Equalization

To begin with set the tap gains $c_0 = 1$ and $c_k = 0$ for all other values of k in the transversal filter in fig.(6.12)(a).Thus the output of the filter will be the same as the input delayed by NT_b.This delay is not relevant to our discussion so we will ignore it. We require that the output pulse $p_o(t)$ satisfy the nyquist criterion or the controlled ISI criterion, as the case may be.For the Nyquist criterion the output pulse $p_o(t)$ must have zero values at all the multiples of T_b.From Fig.(6.12)(b) we see that the pulse amplitudes a_1,a_{-1}, and a_2 at T_b, T_{-b} and $2T_b$,respectively are not negligible .by adjusting the tap gains (c_k 's),we generate additional shifted pulses of proper amplitudes that will force the resulting output pulse to have desired values at t=0,T_b,$2T_b$…..The output $p_0(t)$(fig 6.12(c)) is the sum of pulses of the form $c_k p_r(t-kT_b)$.Thus.

$$p_o(t) = \sum_{n=-N}^{N} c_n p_r(t - nT_b) \qquad (6.7)$$

The samples of $p_o(t)$ at $t=kT_b$ are

$$p_o(kT_b) = \sum_{n=-N}^{N} c_n p_r[(k-n)T_b] \qquad k=0, \ \pm 1, \pm 2, ... \qquad (6.8)$$

Using a more convenient notation pr[k] to denote pr(kTb) Eq. (6.8) can be expressed as.

$$p_o[k] = \sum_{n=-N}^{N} c_n p_r[k-n] \qquad k=0, \pm 1, \pm 2..... \qquad (6.9)$$

The Nyquist criterion requires the samples $p_o[k] = 0$ for $k = 0$ and $p_o[k] =$ for k=0.Substituting these values into Eq (6.9) we obtain a set of simultaneous equations in terms of 2N+1 variables. Clearly it is not possible to solve this set of equations. However if we specify the values of $p_o[k]$ only at 2N+1 points as

$$p[k]=\begin{cases}1 & k=0\\ 0 & k=\pm 1,\ \pm 2,.....\pm N\end{cases} \tag{6.10}$$

Then a unique solution exists. This assures that a pulse will have zero interference at the sampling instants of N preceding and N succeeding pulses. Because the pulse amplitudes decays rapidly interference beyond the Nth pulse is not significant for N>2 in general. Substitution of condition (6.10) in (6.9) yields a set of 2N+1 simultaneous equations in 2N+1 variables:[8]

$$\begin{bmatrix}0\\0\\....\\0\\1\\0\\....\\0\\0\end{bmatrix}=\begin{bmatrix}p_r[0] & p_r[-1] & ... & p_r[-2N]\\ p_r[1] & p_r[0] & ... & p_r[-2N+1]\\ ..... & & & \\ p_r[N-1] & p_r[N-2] & ... & p_r[-N-1]\\ p_r[N] & p_r[N-1] & ... & p_r[-N]\\ p_r[N+1] & p_r[N] & ... & p_r[-N+1]\\ ..... & & & \\ p_r[2N-1] & p_r[2N-2] & ... & p_r[1]\\ p_r[2N] & p_r[2N-1] & ... & p_r[0]\end{bmatrix}\begin{bmatrix}c_{-N}\\ c_{-N+1}\\ .....\\ c_{-1}\\ c_0\\ c_1\\ .....\\ c_{N-1}\\ c_N\end{bmatrix}$$

The tap gains c_k can be obtained by solving this set of equations.[8]

6.5 WEINER FILTER THEORY

In the theory of Wiener filters the underlying signals are assumed to be random processes and the filter is designed using the statistics obtained by ensemble averaging

6.5.1 MEAN-SQUARE ERROR CRITERION

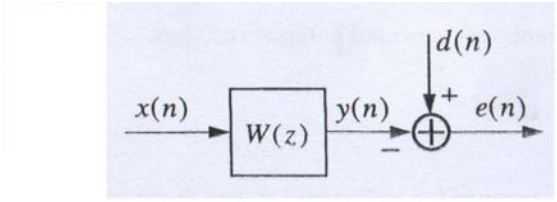

Figure 6.13

Figure 6.13 shows the block schematic of a linear discrete-time filter *W(z)* in the context of estimating a desired signal *den)* based on an excitation *x(n)*. Here, we

assume that both *x(n)* and *d(n)* are samples of infinite length, random processes. The filter output is *y(n)* and *e(n)* is the estimation error. Clearly, the smaller the estimation error, the better the filter performance. As the error approaches zero, the output of the filter approaches the desired signal, *d(n)*. Hence, the question that arises is the following: What is the most appropriate choice for the parameters of the filter which would result in the smallest possible estimation error? To a certain extent, the statement of this question itself gives us some hints on the choice of the filter parameters. Since we want the estimation error to be as small as possible, a straightforward approach to the design of the filter parameters appears to be 'to choose an *appropriate* function of this estimation error as a *cost function* and select that set of filter parameters which optimizes this cost function in some sense'. This is indeed the philosophy that underlies almost all filter design approaches. The various details of this design principle will become clear as we go along. Commonly used synonyms for the cost function are the *performance function* and the *performance surface.*

In choosing a performance function the following points have to be considered:

1. The performance function must be mathematically tractable.
2. The performance function should preferably have a single minimum (or maximum) point, so that the optimum set of filter parameters could be selected unambiguously.

The tractability of the performance function is essential, as it permits analysis of the filter and also greatly simplifies the development of adaptive algorithms for adjustment of the filter parameters. The number of minima (or maxima) points for a performance function is closely related to the filter structure. The recursive (infinite-duration impulse response - IIR) filters, in general, result in performance functions that may have many minima (or maxima) points, whereas the non recursive (finite-duration impulse response - FIR) filters are guaranteed to have a single global minimum (or maximum) point if a proper performance function is used. Because of this, application of the IIR filters in adaptive filtering has been very limited.
In Wiener filters the performance function is chosen to be

$$\xi = E[|e(n)|^2] \tag{6.11}$$

where E[.] denotes statistical expectation. In fact, the performance function ξ,

which is also called the *mean-square error criterion,* turns out to be the simplest possible function that satisfies the two requirements noted above. It can easily be handled mathematically, and in many cases of interest it has a single global minimum. In particular, in the case of FIR filters the performance function ξ is a hyperparaboloid (bowl shaped) with a single minimum point which can easily be calculated by using the second-order statistics of the underlying random processes.[5]

6.5.2 WEINER FILTER

Consider a transversal filter as shown in Figure 6.14

.

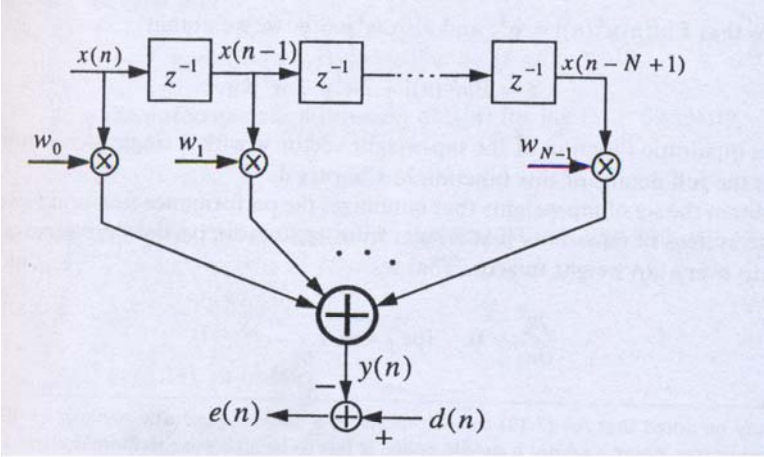

Figure 6.14 A Transversal filter

The filter input, x(n), and its desired output, d(n), are assumed to be real-valued stationary processes. The filter tap weights, $w_0, w_1, w_2, \ldots\ldots w_{N-1}$ are also assumed to be real-valued. The filter input and tap-weight vectors are defined, respectively, as the column vectors

$$w = [w_0 \;\; w_1 \;\; w_2 \;\ldots\ldots\; w_{N-1}]$$

And

$$x(n) = [\, x(n) \;\; x(n-1) \;\; \ldots\ldots \; x(n-N+1)]^T$$

where T stands for transpose.

The filter output is

$$y(n) = \sum_{i=0}^{N-1} w_i x(n-1) = w^T x(n) \tag{6.12}$$

Which can also be written as

$$y(n) = x^T(n) w \tag{6.13}$$

Since $w^T x(n)$ is a scalar and thus it is equal to its transpose i.e $w^T x(n) = (w^T x(n))^T$

Thus we may write

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - w^T x(n) \\ &= d(n) - x^T x(n) w \end{aligned} \tag{6.14}$$

Using (6.10) in (6.7) we get

$$\xi = E[e^2(n)] = E[(d(n) - w^T x(n))(d(n) - x^T(n))] \tag{6.15}$$

Expanding the right hand side of (6.11)

$$\xi = E[d^2(n)] - w^T E[x(n)d(n)] - E[d(n)x^T(n)]w + w^T E[x(n)x^T(n)]w$$

(6.16)

Next we define the N*1 cross-correlation vector

$p = E[x(n)d(n)]$

and the $N \times N$ autocorrelation matrix

$R = E[x(n)x^T(n)]$

and also note that $E[d(n)x^T(n)] = p^T$ and also $w^T p = p^T w$ we obtain

$$\xi = E[d^2(n)] - 2w^T p + w^T R w$$

(6.17)

This is a quadratic function of the tap weight vector **w** with a single global minimum

To obtain the set of tap weights that maximizes the performance function ξ we need to solve the system of equations that results from setting the partial derivatives of ξ with respect to every tap weight to zero. That is,

$\frac{\delta\xi}{\delta w_i}$ for i=0,1,.....N-1

These equations may collectively be written as

$\nabla\xi = 0$

solving these equations we reach to the conclusion

$\nabla\xi = 2Rw - 2p$

letting $\nabla\xi = 0$ gives the following equation from which the optimum tap-weights can be obtained

$$Rw_o = p \tag{6.18}$$

Equation (6.14) which is known as Weiner-Hopf equation has the following solution:

$$W_0 = R^{-1}p$$

Where w_o is the optimum filter tap weights vector.

6.6 SEARCH METHODS

We saw that the optimum filter coefficients can be obtained by solving Weiner-Hopf equation an alternative way of finding the optimum filter tap weights is to use an iterative algorithm that starts at some arbitrary initial point and progressively moves towards the optimum tap weights vectors in steps.Now we will discuss two recursive algorithms the steepest descent algorithm and the least mean square algorithm

6.6.1 METHOD OF STEEPEST DESCENT

Consider the Filter shown in Figure 6.14.In finding the solution of Weiner-Hopf equation we assumed that R and p are available, but here we resort to a different approach to find w_o. Instead of trying to solve equation (6.14) directly, we choose an *iterative search method* in which starting with an initial guess for wo, say w(0), a recursive search method that may require many iterations (steps) to converge to Wo is used. An understanding of this method is basic to the development of the iterative *algorithms* which are commonly used in the implementation of adaptive filters in practice.

The method of steepest descent is a general scheme that uses the following steps to search for the minimum point of any convex function of a set of parameters:

1. Start with an initial guess of the parameters whose optimum values arc to be found

for

minimizing the function.

2. Find the gradient of the function with respect to these parameters at the present point.

3. Update the parameters by taking a step in the opposite direction of the gradient vector obtained in Step 2. This corresponds to a step in the direction of steepest descent in the cost function at the present point. Furthermore, the size of the step taken is chosen proportional to the size of the gradient vector.

4. Repeat Steps 2 and 3 until no further significant change is observed in the parameters.

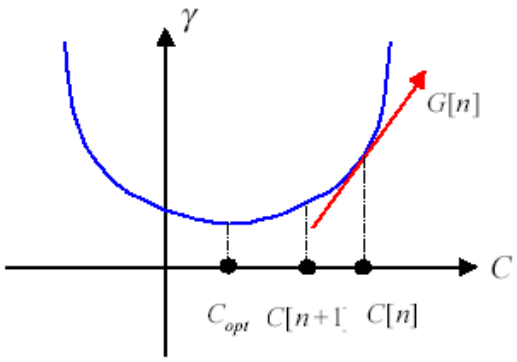

Figure 6.15: Operation of steepest descent algorithm

To implement this procedure in the case of the transversal filter shown in Figure 6.13, we recall equation (6.14)

$$\nabla\xi = 2Rw - 2p \tag{6.15}$$

Where ∇ is the gradient operator defined as the column vector

$$\nabla = \left[\frac{\partial}{\partial w_o} \quad \frac{\partial}{\partial w_1} \quad \ldots\ldots\ldots\ldots \quad \frac{\partial}{\partial w_{N-1}} \right]^T$$

According to the above procedure, if *w(k)* is the tap-weight vector at the *kth* iteration, then the following recursive equation may be used to update *w(k):*

$$w(k+1) = w(k) - \mu\nabla_k\xi \tag{6.19}$$

where μ is a positive scalar called the step size and $\nabla_k \xi$ denotes the gradient vector evaluated at the point w=w(k).Substituting (6.154) in (6.16) we get

$$w(k+1) = w(k) - 2\mu(Rw(k) - p) \tag{6.20}$$

6.6.2 A SIMPLE CHOICE FOR μ

The value of $\boldsymbol{\mu}$ should be carefully chosen so that the recursive algorithm converges quickly, choosing large value for $\boldsymbol{\mu}$ can cause the algorithm to diverge whereas a very small value can make the algorithm converge after a long time.If $\lambda_0, \lambda_1, \lambda_2, \ldots\ldots\ldots \lambda_{N-1}$ are the eigenvalues of R then the step size $\boldsymbol{\mu}$ should be chosen such that

$|1 - 2\mu\lambda_i| < 1$ for i=0 ,1,.....,N-1

Or

$0 < \mu < \frac{1}{\lambda_i}$, for i=0,1,……,N-1

Noting that the step size parameter is common for all values of i, convergence of the steepest descent algorithm is guaranteed only when

$$0 < \mu < \frac{1}{\lambda_{\max}} \tag{6.21}$$

Where $\lambda_{\max}$ is the maximum of the eigenvalues $\lambda_0, \lambda_1, \lambda_2, \ldots\ldots\ldots \lambda_{N-1}$.The left limit in (6.18) refers to the fact that the tap\weight correction must be in opposite direction of the gradient vector.[5]

6.6.3 THE LMS ALGORITHM

The LMS algorithm, which was first proposed by Widrow and Hoff in 1960, is the most widely used adaptive filtering algorithm, in practice. This wide spectrum of applications of the LMS algorithm can be attributed to its simplicity and robustness to signal statistics.

DERIVATION OF THE LMS ALGORITHM

Figure 6.13 depicts an N-tap transversal adaptive filter. The filter input, *x(n),* desired output, *d(n),* and the filter output

$$y(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i) \qquad (6.22)$$

are assumed to be real-valued sequences. The tap weights $w_o(n), w_1(n), \ldots, w_{N-1}$ are selected so that the difference (error)

$$e(n) = d(n) - y(n), \qquad (6.23)$$

is minimized in some sense. It may be noted that the filter tap weights are explicitly indicated to be functions of the time index n. This signifies that in an adaptive filter, in general, tap weights are time varying, since they are continuously being adapted so that any variations in the signal's statistics could be tracked. The LMS algorithm changes (adapts) the filter tap weights so that e(n) is minimized in the mean-square sense, thus the name least mean square. When the processes x(n) and d(n) are jointly stationary, this algorithm converges to a se. In other words, the LMS algorithm is a practical scheme for realizing Wiener filters, without explicitly solving the Wiener-.Hopf equation. It is a sequential algorithm which can be used to adapt the tap weights of a filter by continuous observation of its input, x(n), and desired output, d(n).

The Conventional LMS Algorithm is a stochastic implementation of the steepest descent algorithm.It replaces the cost function $\xi = E[e^2(n)]$ by its instantaneous coarse estimate $\hat{\xi} = e^2(n)$.Thus replacing the iteration index k by the time index n in equation (6.16) we obtain

$$w(n+1) = w(n) - \mu \nabla e^2(n) \qquad (6.24)$$

substituting (6.20) in (6.21) we get the LMS recursion equation

$$w(n+1) = w(n) + 2\mu e(n)x(n) \qquad (6.25)$$

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.[5]

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).

6.7 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.

6.7.1 LINEAR LMS

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.

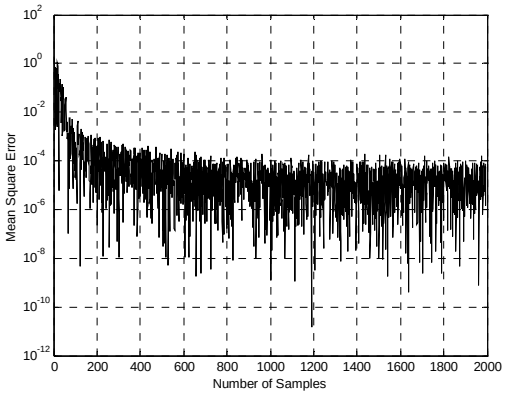

Figure 6.16: Learning curve for LMS algorithm

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

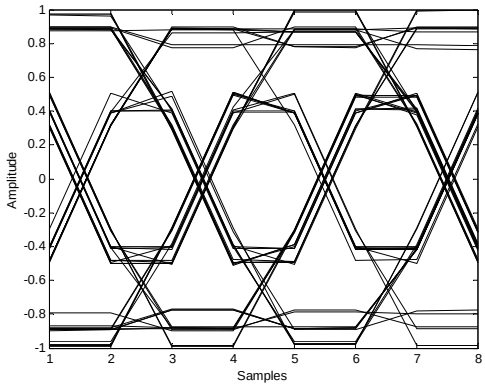

Figure 6.17: Eye diagram for received faded signal

Eye diagram for the equalized signal is given below

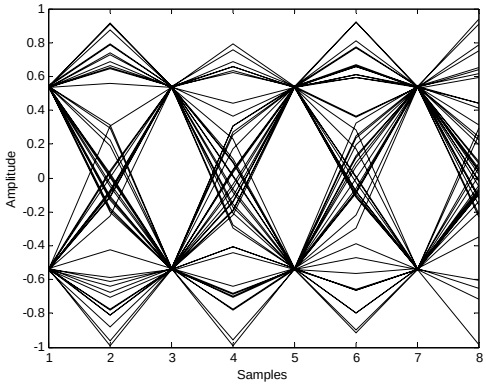

Figure 6.18: Eye diagram for equalized signal

6.7.2 ZERO FORCING EQUALIZER

Optimum length of ZFE is found to be equal to 5 taps . Eye diagram for faded received signal is

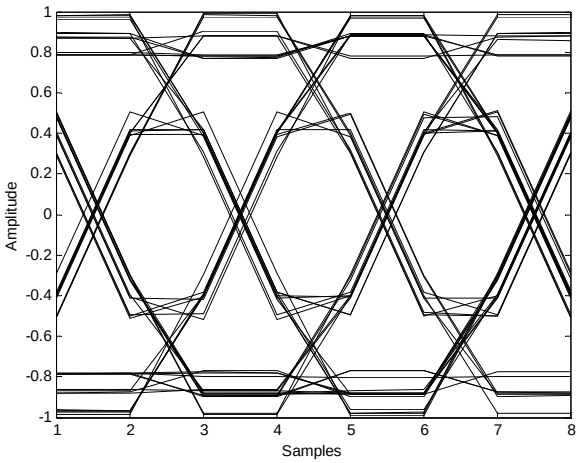

Figure 6.19: Eye diagram for received faded signal

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.

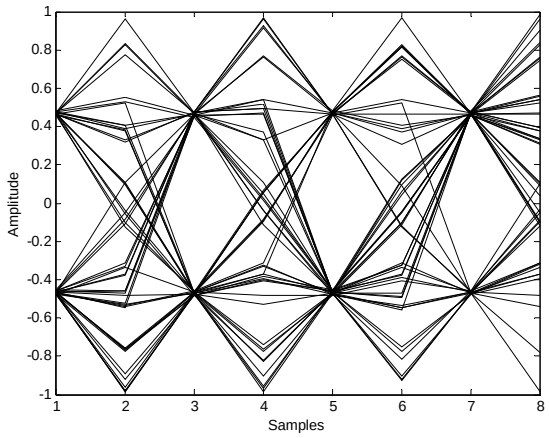

Figure 6.20: Eye diagram for equalized signal

CHAPTER 7

CONCLUSIONS

We have covered all individual modules of a communication system and studied various algorithms of each module. When developing an optimal communication system one must know each part of a communication system in detail in order come u with the best possible solution. The main task in the end is how to integrate those individual modules .If the user has ample bandwidth available then channel coding should be used just as we purposed. However if bandwidth is limited then block codes can be left out from channel coding. Similarly the use of higher modulation scheme should be done if user has enough power available because sending more bits per symbol increases probability of bit error and u need more power in order to achieve the same BER as before. Timing is the first issue one has to deal at receiver because one must know where is the start the symbol before so we purpose that after match filtering our timing loop should be used because it requires more than one sample per symbol and is independent of phase errors. After that the problem is to whether use the equalizer or phase recovery. Since our phase recovery loop requires one sample per symbol we purpose that it should be put after equalization. So in the end our receiver should look like this

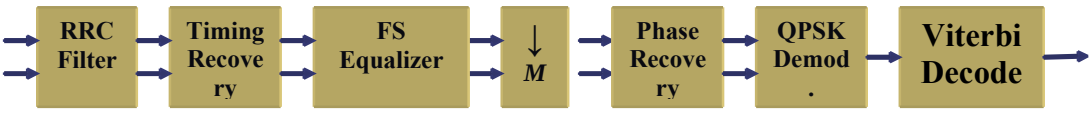

Fig 7.1 Complete Receiver Design

REFRENCES

[1] Alan V. Oppenheim, Alan Willsky and S. Hamid Nawab, ‘’Signals and Systems”, Prentice Hall, 2nd Edition, 2000.

[2] Alan V. Oppenheim and Ronald W. Schafer, “Discrete-Time signal Processing”, Prentice Hall, 2001

[3] John G. Proakis, ”Digital Communications ”, McGraw Hill, Second Edition, 1989.

[4] Edward A. Lee and David G. Messerschmitt, “Digital Communications”, Kluwer Academic Publishers, Second Edition, 1994.

[5] B.Farhang-Boroujeny “Adaptive Filters Theory and application”, John Wiley and Sons, 1998.

[6] Erwin Kreyszig, ”Advanced Engineering Mathematics”, John Wiley and Sons, 8th Edition, 1999.

[7] Bernard Sklar, “Digital Communications, Fundamentals and Applications”, Prentice Hall, 2nd Edition, 2001.

[8] B.P Lathi, ” Modern Digital and Analog Communication Systems”, Holt Saunders, 1983.

[9] Andrew S. Tanenbaum, “Computer Networks”, Prentice Hall, 3rd Edition,