Chapter **Page**

Chapter 1

INTRODUCTION

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.

Error corrective coding is used to enhance the efficiency and accuracy of information transmitted. In a communication transmission system, data is transferred from a transmitter to a receiver across a physical medium of transmission or channel. The channel is generally affected by noise or fading which introduces errors in the data being transferred. Error-correcting code is a signal processing technique used for correcting errors introduced in the channel. It is done by encoding the data to be transmitted and introducing redundancy in it such that the decoder can later reconstruct the data transmitted using the redundant information. Figure 2.1 is a block diagram of a simplified model of a coded system.

The information source is a digital source. If it is not, it will be converted to digital. The digital source output u is sent to an encoder to generate encoder output x which is modulated and then transmitted over the physical channel. The decoder will make a best guess $\tilde{u}$ of the original information u based on the received signal y which is distorted by the channel.

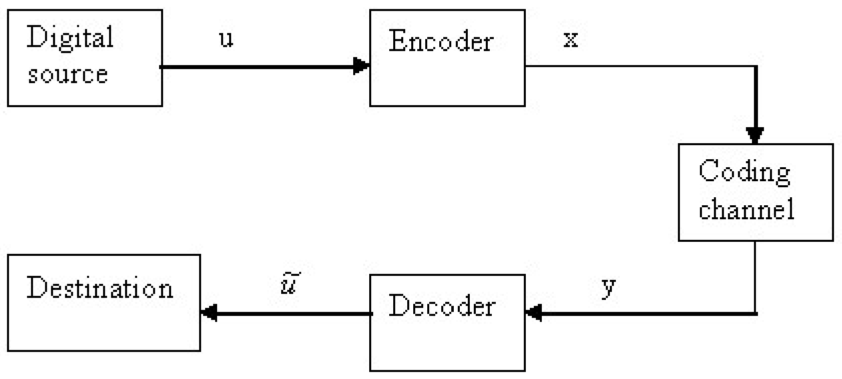


Figure 1.1 Simplified model of coded system

A major concern in coding technique is the control of errors so that reliable communications can be obtained, i.e., $\tilde{u}$ is as close to u as possible. There are many coding schemes available. Turbo code is the most exciting and potentially important development in the coding theory in recent years. This powerful code is capable of achieving near Shannon capacity performance [1].

Our Design

Soft input soft output detector and equalizer are used to improve the error correcting capability of communication systems

CHAPTER 2

Channel Coding (Part 1)

2.1 Introduction

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.

Error corrective coding is used to enhance the efficiency and accuracy of information transmitted. In a communication transmission system, data is transferred from a transmitter to a receiver across a physical medium of transmission or channel. The channel is generally affected by noise or fading which introduces errors in the data being transferred. Error-correcting code is a signal processing technique used for correcting errors introduced in the channel. It is done by encoding the data to be transmitted and introducing redundancy in it such that the decoder can later reconstruct the data transmitted using the redundant information. Figure 2.1 is a block diagram of a simplified model of a coded system.

The information source is a digital source. If it is not, it will be converted to digital. The digital source output u is sent to an encoder to generate encoder output x which is modulated and then transmitted over the physical channel. The decoder will make a best guess $\tilde{u}$ of the original information u based on the received signal y which is distorted by the channel.

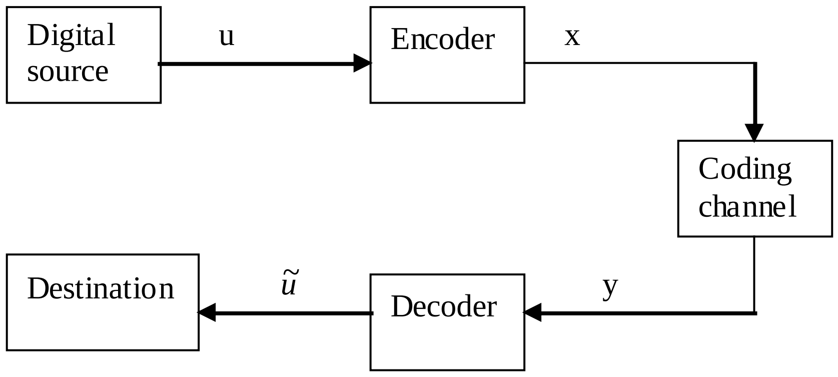


Figure 2.1 Simplified model of a coded system

A major concern in coding technique is the control of errors so that reliable communications can be obtained, i.e., $\tilde{u}$ is as close to u as possible. There are many coding schemes available. Turbo code is the most exciting and potentially important development in the coding theory in recent years. This powerful code is capable of achieving near Shannon capacity performance.

In the past several years, convolutional coding with Viterbi decoding has begun to be supplemented in the geostationary satellite communication arena with Reed-Solomon coding. The two coding techniques are usually implemented as serially concatenated block and convolutional coding. Typically, the information to be transmitted is first encoded with the Reed-Solomon code, then with the convolutional code. On the receiving end, Viterbi decoding is performed first, followed by Reed-Solomon decoding. This is the technique that is used in most if not all of the direct-broadcast satellite (DBS) systems, and in several of the newer VSAT products as well. At least, that's what the vendors are advertising.

Recently (1993) a new parallel-concatenated convolutional coding technique known as turbo coding has emerged. Initial hardware encoder and decoder implementations of turbo coding have already appeared on the market. This technique achieves substantial improvements in performance over concatenated Viterbi and Reed-Solomon coding. A variant in which the codes are product codes has also been developed, along with hardware implementations

2.2 Convolutional Coding

Convolutional coding and block coding are the two major forms of channel coding. Convolutional codes operate on serial data, one or a few bits at a time. Block codes operate on relatively large (typically, up to a couple of hundred bytes) message blocks. There are a variety of useful convolutional and block codes, and a variety of algorithms for decoding the received coded information sequences to recover the original data. Convolutional encoding with Viterbi decoding is a FEC technique that is particularly suited to a channel in which the transmitted signal is corrupted mainly by additive white gaussian noise (AWGN). AWGN channel is a noise whose voltage distribution over time has characteristics that can be described using a Gaussian, or normal, statistical distribution, i.e. a bell curve. This voltage distribution has zero mean and a standard deviation that is a function of the signal-to-noise ratio (SNR) of the received signal. Let's assume for the moment that the received signal level is fixed. Then if the SNR is high, the standard deviation of the noise is small, and vice-versa. In digital communications, SNR is usually measured in terms of E_b/N_0, which stands for energy per bit divided by the one-sided noise density.

Let's take a moment to look at a couple of examples. Suppose that we have a system where a '1' channel bit is transmitted as a voltage of -1V, and a '0' channel bit is transmitted as a voltage of +1V. This is called bipolar non-return-to-zero (bipolar NRZ) signaling. It is also called binary "antipodal" (which means the signaling states are exact opposites of each other) signaling. The receiver comprises a comparator that decides the received channel bit is a '1' if its voltage is less than 0V, and a '0' if its voltage is greater than or equal to 0V. One would want to sample the output of the comparator in the middle of each data bit interval. We will see the behavior of our example system when E_b/N_0 is high, and then when the E_b/N_0 is lower.

The following figure shows the results of a channel simulation where one million (1×10^6) channel bits are transmitted through an AWGN channel with an E_b/N_0 level of 20 dB (i.e. the signal voltage is ten times the rms noise voltage). In this simulation, a '1' channel bit is transmitted at a level of -1V, and a '0' channel bit is transmitted at a level of +1V. The x axis of this figure corresponds to the received signal voltages, and the y axis represents the number of times each voltage level was received:

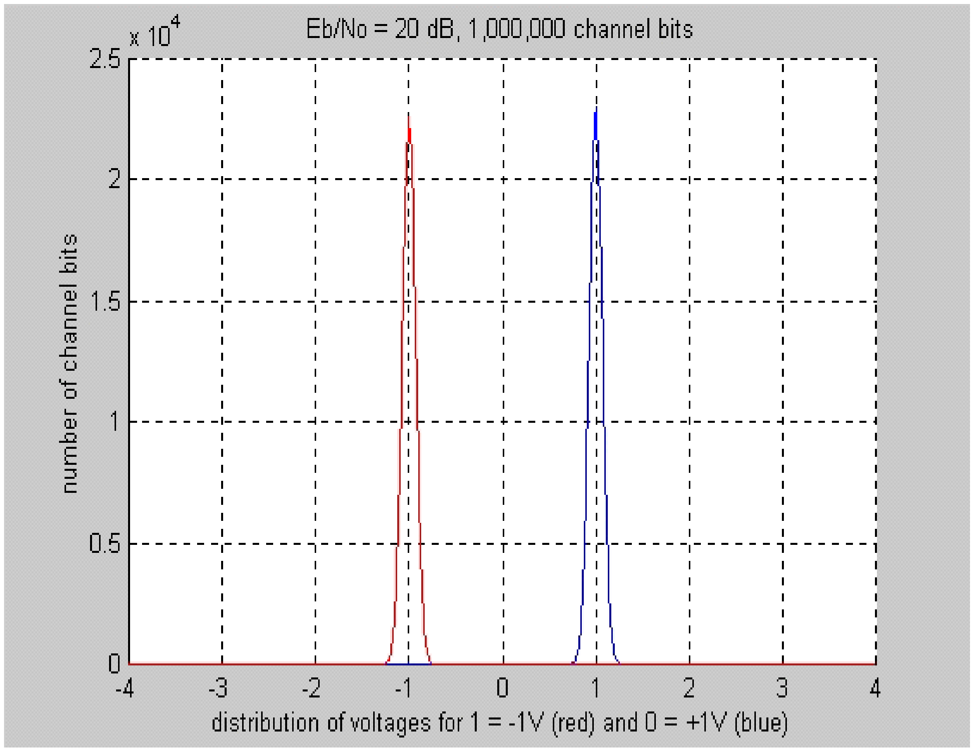


Figure 2.2 AWGN Channel Simulations without channel coding

Our simple receiver detects a received channel bit as a '1' if its voltage is less than 0V, and as a '0' if its voltage is greater than or equal to 0V. Such a receiver would have little difficulty correctly receiving a signal as depicted in the figure above. Very few (if any) channel bit reception errors would occur. In this example simulation with the E_b/N_0 set at 20 dB, a transmitted '0' was never received as a '1', and a transmitted '1' was never received as a '0'.

The next figure shows the results of a similar channel simulation when 1 x 10^6 channel bits are transmitted through an AWGN channel where the E_b/N_0 level has decreased to 6 dB (i.e. the signal voltage is two times the rms noise voltage):

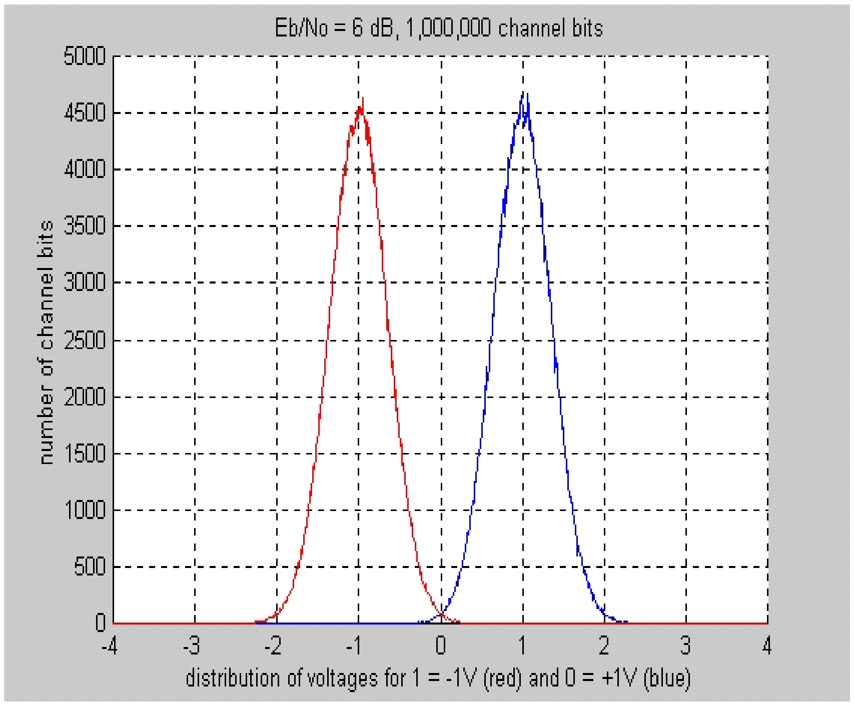


Figure 2.3 AWGN Channel Simulations on 6dbs

Now right-hand side of the curve in the figure above crosses 0V, and left-hand side of the curve also crosses 0V. The points on the red curve that are above 0V represent events where a channel bit that was transmitted as a one (-1V) was received as a zero. The points on the right curve that are below 0V represent events where a channel bit that was transmitted as a zero (+1V) was received as a one. These events correspond to channel bit reception errors in our simple receiver. In this example simulation with the E_b/N_0 set at 6 dB, a transmitted '0' was received as a '1' 1,147 times, and a transmitted '1' was received as a '0' 1,207 times, corresponding to a bit error rate (BER) of about 0.235%. Which is not good, especially if you're trying to transmit highly compressed data, such as digital television? Using convolutional coding with Viterbi decoding, you can achieve a BER of better than 1×10^{-7} at the same E_b/N_0, 6 dB.

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. 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.

For years, convolutional coding with Viterbi decoding has been the predominant FEC technique used in space communications, particularly in geostationary satellite communication networks, such as VSAT (very small aperture terminal) networks. Most common variant used in VSAT networks is rate 1/2 convolutional coding using a code with a constraint length K = 7. With this code, you can transmit binary or quaternary phase-shift-keyed (BPSK or QPSK) signals with at least 5 dB less power than you'd need without it. That is a reduction in Watts of more than a factor of three and is very useful in reducing transmitter and/or antenna cost or permitting increased data rates given the same transmitter power and antenna sizes.

There is a tradeoff-the same data rate with rate 1/2 convolutional coding takes twice the bandwidth of the same signal without it, given that the modulation technique is the same. That's because with rate 1/2 convolutional encoding, you transmit two channel symbols per data bit. However, if you think of the tradeoff as a 5 dB power savings for a 3 dB bandwidth expansion, you can see that you come out ahead. Remember: if the modulation technique stays the same, the bandwidth expansion factor of a convolutional code is simply n/k.

Many radio channels are AWGN channels, but many, particularly terrestrial radio channels also have other impairments, such as multipath, selective fading, interference, and atmospheric (lightning) noise. Transmitters and receivers can add spurious signals and phase noise to the desired signal as well. Although convolutional coding with Viterbi decoding might be useful in dealing with those other problems, it may not be the best technique.

2.3 Description of the Algorithms

2.3.1 Convolutionally Encoding the Data

Convolutionally encoding the data is accomplished using a shift register and associated combinatorial logic that performs modulo-two addition. 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. As a reminder, exclusive-or gates are two-input, one-output gates often represented by the logic symbol shown below,

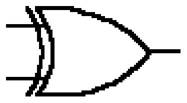


Figure 2.4 x-or Gate

The truth table implemented by this gate is given below...

Input A	**Input B**	**Output (A xor B)**
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1 x-or Truth Table

Shown below is the convolutional encoder of rate 1/2, K = 3, m = 2

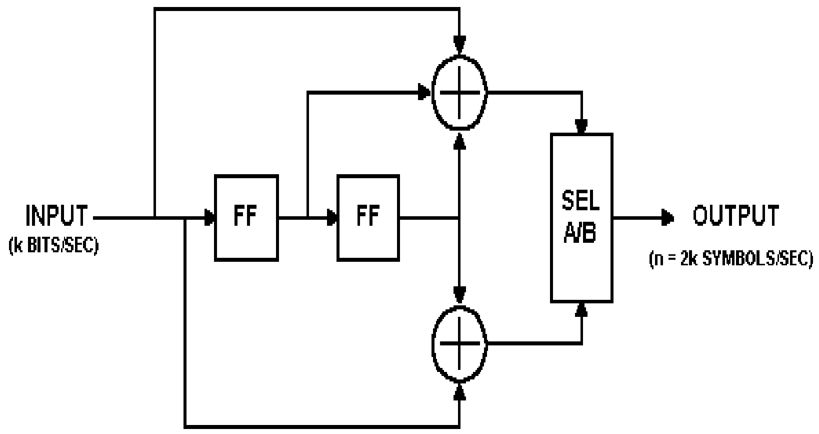


Figure 2.5 Convolutional encoder

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 010111001010001_2.

Assume that the outputs of both of the flip-flops in the shift register are initially cleared, i.e. their outputs are zeroes. The first clock cycle makes the first input bit, a zero, available to the encoder. The flip-flop outputs are both zeroes. The inputs to the modulo-two adders are all zeroes, so the output of the encoder is 00_2.

The second clock cycle makes the second input bit available to the encoder. The left-hand flip-flop clocks in the previous bit, which was a zero, and the right-hand flip-flop clocks in the zero output by the left-hand flip-flop. The inputs to the top modulo-two adder are 100_2, so the output is a one. The inputs to the bottom modulo-two adder are 10_2, so the output is also a one. So the encoder outputs 11_2 for the channel symbols.

The third clock cycle makes the third input bit, a zero, available to the encoder. The left-hand flip-flop clocks in the previous bit, which was a one, and the right-hand flip-flop clocks in the zero from two bit-times ago. The inputs to the top modulo-two adder are 010_2, so the output is a one. The inputs to the bottom modulo-two adder are 00_2, so the output is zero. So the encoder outputs 10_2 for the channel symbols.

And so on. The timing diagram shown below illustrates the process:

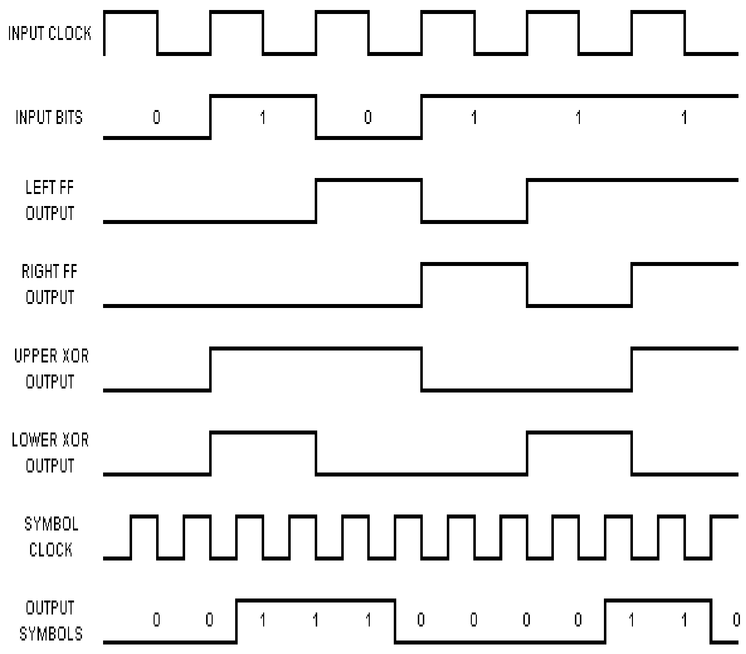


Figure 2.6 Clock cycle examples

After all of the inputs have been presented to the encoder, the output sequence will be:

$00\ 11\ 10\ 00\ 01\ 10\ 01\ 11\ 11\ 10\ 00\ 10\ 11\ 00\ 11_2$.

Notice that encoder outputs are paired the first bit in each pair is the output of the upper modulo-two adder; the second bit in each pair is the output of the lower modulo-two adder.

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 (= m) more times, while holding the input at zero. This is called "flushing" the encoder, and results in two more pairs of output symbols. The final binary output of the encoder is thus 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11_2. If we don't perform the flushing operation, the last m bits of the message have less error-correction capability than the first through (m - 1)th bits had. This is a pretty important thing to remember if you're going to use this FEC technique in a burst-mode environment. So's the step of clearing the shift register at the beginning of each burst. The encoder must start in a known state and end in a known state for the decoder to be able to reconstruct the input data sequence properly.

Now, let's look at the encoder from another perspective. You can think of the encoder as a simple state machine. The example encoder has two bits of memory, so there are four possible states. Let's give the left-hand flip-flop a binary weight of 2^1, and the right-hand flip-flop a binary weight of 2^0. Initially, the encoder is in the all-zeroes state. If the first input bit is a zero, the encoder stays in the all zeroes state at the next clock edge. But if the input bit is a one, the encoder transitions to the 10_2 state at the next clock edge. Then, if the next input bit is zero, the encoder transitions to the 01_2 state, otherwise, it transitions to the 11_2 state. The following table 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 2.2 Next state table

The above table is often called a state transition table. We'll refer to it as the next state table. 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 2.3 out put 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.

2.3.2 Mapping the Channel Symbols to Signal Levels

Mapping the one/zero output of the convolutional encoder onto an antipodal baseband signaling scheme is simply a matter of translating zeroes to +1s and ones to -1s. This can be accomplished by performing the operation y = 1 - 2x on each convolutional encoder output symbol.

2.3.3 Adding Noise to the Transmitted Symbols

Adding noise to the transmitted channel symbols produced by the convolutional encoder involves generating Gaussian random numbers, scaling the numbers according to the desired energy per symbol to noise density ratio, E_s/N_0, and adding the scaled Gaussian random numbers to the channel symbol values.

For the uncoded channel, $E_s/N_0 = E_b/N_0$, since there is one channel symbol per bit. However, for the coded channel, $E_s/N_0 = E_b/N_0 + 10\log_{10}(k/n)$. For example, for rate 1/2 coding, $E_s/N_0 = E_b/N_0 + 10\log_{10}(1/2) = E_b/N_0 - 3.01$ dB. Similarly, for rate 2/3 coding, $E_s/N_0 = E_b/N_0 + 10\log_{10}(2/3) = E_b/N_0 - 1.76$ dB.

The Gaussian random number generator is the only interesting part of this task. C only provides a uniform random number generator, `rand()`. In order to obtain Gaussian random numbers, we take advantage of relationships between uniform, Rayleigh, and Gaussian distributions:

Given a uniform random variable U, a Rayleigh random variable R can be obtained by:

$$R = \sqrt{2 \cdot \sigma^2 \cdot \ln\left(1/(1-U)\right)} = \sigma \cdot \sqrt{2 \cdot \ln\left(1/(1-U)\right)} \qquad (2.1)$$

where σ^2 is the variance of the Rayleigh random variable, and given R and a second uniform random variable V, two Gaussian random variables G and H can be obtained by

$$G = R \cos V \text{ and } H = R \sin V. \qquad (2.2)$$

In the AWGN channel, the signal is corrupted by additive noise, n(t), which has the power spectrum *No*/2 watts/Hz. The variance σ^2 of this noise is equal to $No/2$. If we set the energy per symbol E_s equal to 1, then $E_s/N_0 = 1/2\sigma^2$. So $\sigma = \sqrt{1/(2 \cdot (E_s/N_0))}$.

2.3.4 Performing Viterbi Decoding

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

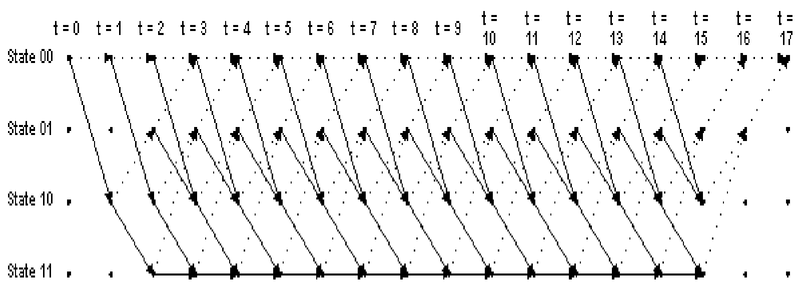


Figure 2.7 trellis diagram

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.

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

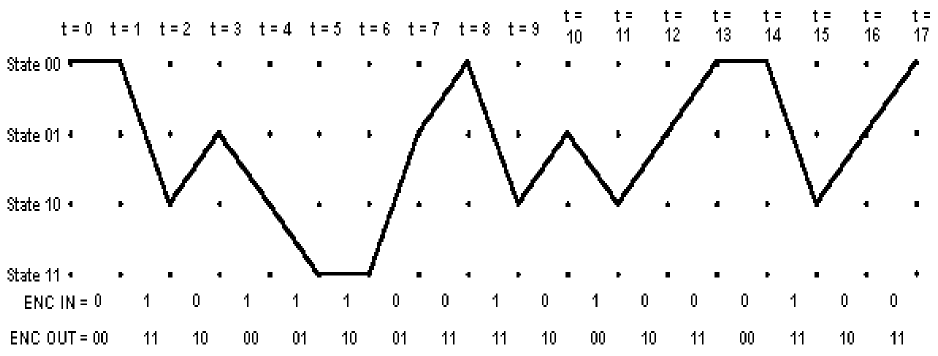


Figure 2.8 states of trellis

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 next shown below:

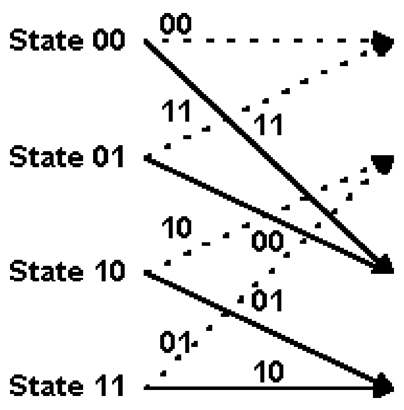
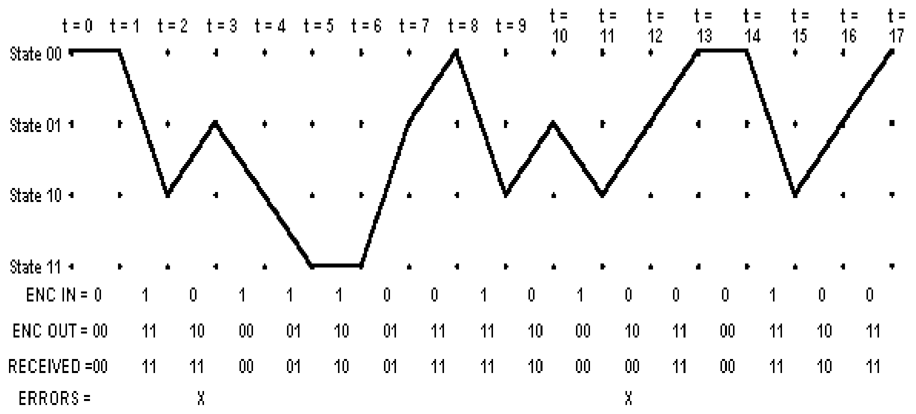


Figure 2.9 state transition and outputs of encoder

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.)

We have used hard-decision symbol inputs to keep things simple. Suppose we receive the above encoded message with a couple of bit errors:



	t = 0	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6	t = 7	t = 8	t = 9	t = 10	t = 11	t = 12	t = 13	t = 14	t = 15	t = 16	t = 17
ENC IN =	0	1	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	
ENC OUT =	00	11	10	00	01	10	01	11	11	10	00	10	11	00	11	10	11	
RECEIVED =	00	11	11	00	01	10	01	11	11	10	00	00	11	00	11	10	11	
ERRORS =			X									X						

Figure (2.10) trellis diagram of received message

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.

The metric we're going to use for now is the Hamming distance between the received channel symbol pair and the possible channel symbol pairs. The Hamming distance is computed by simply counting how many bits are different between the received channel symbol pair and the possible channel symbol pairs. The results can only be zero, one, or two. The Hamming distance (or other metric) values we compute at each time instant for the paths between the states at the previous time instant and the states at the current time instant are called branch metrics. For the first time instant, we're going to save these results as "accumulated error metric" values, associated with states. For the second time instant on, the accumulated error metrics will be computed by adding the previous accumulated error metrics to the current branch metrics.

At t = 1, we received $00_{2.}$ The only possible channel symbol pairs we could have received are 00_2 and 11_2. The Hamming distance between 00_2 and 00_2 is zero. The Hamming distance between 00_2 and 11_2 is two. Therefore, the branch metric value for the branch from State 00_2 to State 00_2 is zero, and for the branch from State 00_2 to State 10_2 it's two. Since the previous accumulated error metric values are equal to zero, the accumulated metric values for State 00_2 and for State 10_2 are equal to the branch metric values. The accumulated error metric values for the other two states are undefined. The figure below illustrates the results at t = 1:

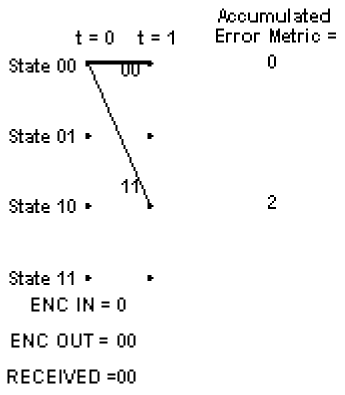


Figure 2.11 metric calculation at t=1

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.

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

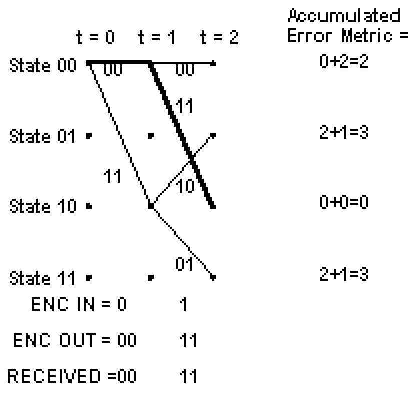


Figure2.12 metric calculation at t=2

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 below shows the results of processing t = 3:

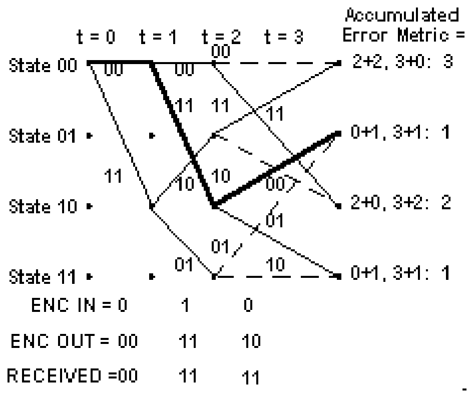


Figure2.13 metric calculation at t=3

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:

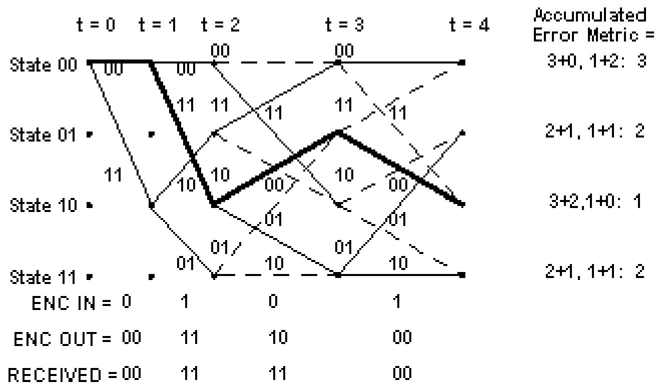


Figure (2.14) metric calculation at t=4

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:

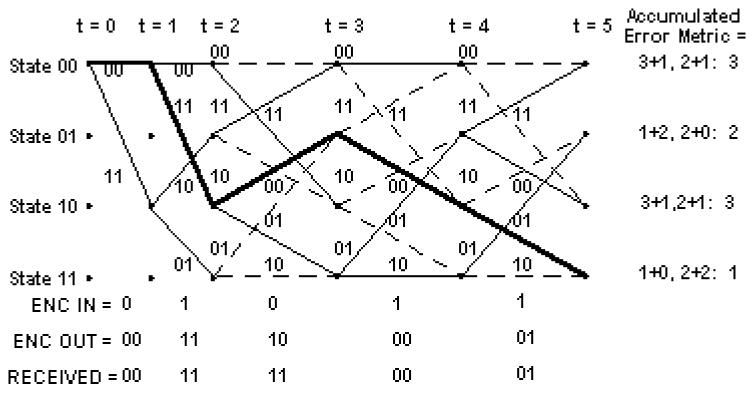


Figure (2.15) metric calculation at t=5

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.

Perhaps you're getting tired of stepping through the trellis. I know I am. Let's skip to the end.

At t = 17, the trellis looks like this, with the clutter of the intermediate state history removed:

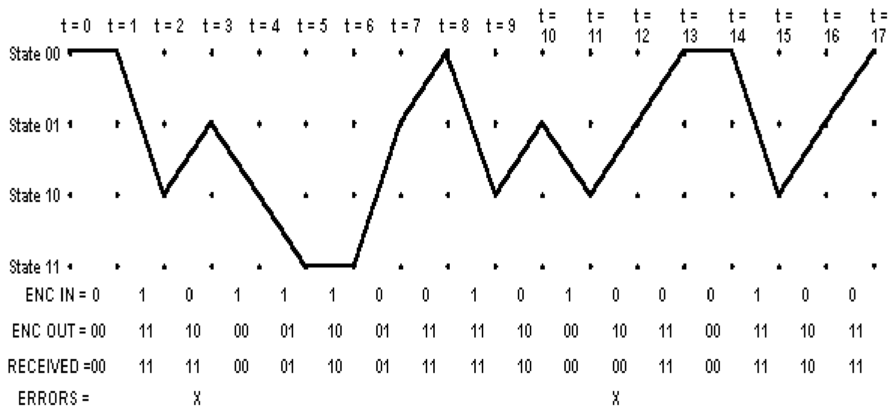

t = 0 t = 1 t = 2 t = 3 t = 4 t = 5 t = 6 t = 7 t = 8 t = 9 t = 10 t = 11 t = 12 t = 13 t = 14 t = 15 t = 16 t = 17
State 00
State 01
State 10
State 11
ENC IN = 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0
ENC OUT = 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11
RECEIVED =00 11 11 00 01 10 01 11 11 10 00 00 11 00 11 10 11
ERRORS = X X

Figure 2.16 Final trellis

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 the following steps:

- First, select the state having the smallest accumulated error metric and save the state number of that state.
- Iteratively perform the following step until the beginning of the trellis is reached: Working backward through the state history table, for the selected state, select a new state which is listed in the state history table as being the predecessor to that state. Save the state number of each selected state. This step is called traceback.
- Now work forward through the list of selected states saved in the previous steps. Look up what input bit corresponds to a transition from each predecessor state to its successor state. That is the bit that must have been encoded by the convolutional encoder.

The following table shows the accumulated metric for the full 15-bit (plus two flushing bits) example message at each time t:

t =	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
State 00_2		0	2	3	3	3	3	4	1	3	4	3	3	2	2	4	5	2
State 01_2			3	1	2	2	3	1	4	4	1	4	2	3	4	4	2	
State 10_2		2	0	2	1	3	3	4	3	1	4	1	4	3	3	2		
State 11_2			3	1	2	1	1	3	4	4	3	4	2	3	4	4		

Table 2.4 Accumulated Metric table

It is interesting to note that for this hard-decision-input Viterbi decoder example, the smallest accumulated error metric in the final state indicates how many channel symbol errors occurred.

The following state history table shows the surviving predecessor states for each state at each time t:

t =	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
State 00_2	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	1
State 01_2	0	0	2	2	3	3	2	3	3	2	2	3	2	3	2	2	2	0

State 10_2	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0
State 11_2	0	0	2	2	3	2	3	2	3	2	2	3	2	3	2	2	0	0

Table 2.5 State History Table

The following table shows the states selected when tracing the path back through the survivor state table shown above:

t =	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0	2	1	0

Table 2.6 Trace back table

Using a table that maps state transitions to the inputs that caused them, we can now recreate the original message. Here is what this table looks like for our example rate 1/2 K = 3 convolutional code:

	Input was, Given Next State =			
Current State	**00_2 = 0**	**01_2 = 1**	**10_2 = 2**	**11_2 = 3**
00_2 = 0	0	x	1	x
01_2 = 1	0	x	1	x
10_2 = 2	x	0	x	1
11_2 = 3	x	0	x	1

Table 2.7 Next State Table

Note: In the above table, x denotes an impossible transition from one state to another state.

So now we have all the tools required to recreate the original message from the message we received:

t =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	1	1	1	0	0	1	0	1	0	0	0	1

Table 2.8 Decoded bits table

The two flushing bits are discarded.

Here's an insight into how the traceback algorithm eventually finds its way onto the right path even if it started out choosing the wrong initial state. This could happen if more than one state had the smallest accumulated error metric, for example. I'll use the figure for the trellis at t = 3 again to illustrate this point:

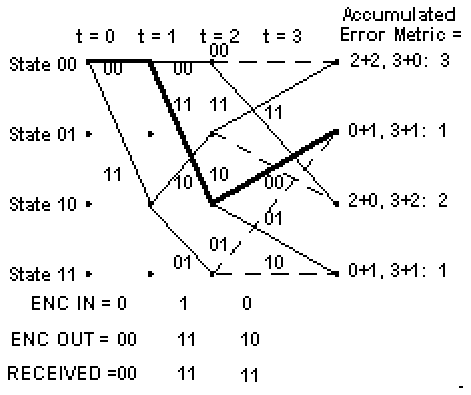


Figure 2.17 Metric calculation at t=3

See how at t = 3, both States 01_2 and 11_2 had an accumulated error metric of 1. The correct path goes to State 01_2 -notice that the bold line showing the actual message path goes into this state. But suppose we choose State 11_2 to start our traceback. The predecessor state for State 11_2 , which is State 10_2 , is the same as the predecessor state for State 01_2! This is because at t = 2, State 10_2 had the smallest accumulated error metric. So after a false start, we are almost immediately back on the correct path.

For the example 15-bit message, we built the trellis up for the entire message before starting traceback. For longer messages, or continuous data, this is neither practical or desirable, due to memory constraints and decoder delay. Research has shown that a traceback depth of K x 5 is sufficient for Viterbi decoding with the type of codes we have been discussing. Any deeper traceback increases decoding delay and decoder memory requirements, while not significantly improving the performance of the decoder. The exception is punctured codes, which I'll describe later. They require deeper traceback to reach their final performance limits.

To implement a Viterbi decoder in software, the first step is to build some data structures around which the decoder algorithm will be implemented. These data structures are best implemented as arrays. The primary six arrays that we need for the Viterbi decoder are as follows:

- A copy of the convolutional encoder next state table, the state transition table of the encoder. The dimensions of this table (rows x columns) are $2^{(K - 1)}$ x 2^k. This array needs to be initialized before starting the decoding process.

- A copy of the convolutional encoder output table. The dimensions of this table are $2^{(K - 1)}$ x 2^{k}. This array needs to be initialized before starting the decoding process.
- An array (table) showing for each convolutional encoder current state and next state, what input value (0 or 1) would produce the next state, given the current state. We'll call this array the input table. Its dimensions are $2^{(K - 1)}$ x $2^{(K - 1)}$. This array needs to be initialized before starting the decoding process.
- An array to store state predecessor history for each encoder state for up to K x 5 + 1 received channel symbol pairs. We'll call this table the state history table. The dimensions of this array are $2^{(K - 1)}$ x (K x 5 + 1). This array does not need to be initialized before starting the decoding process.

An array to store the accumulated error metrics for each state computed using the add-compare-select operation. This array will be called the accumulated error metric array. The dimensions of this array are $2^{(K - 1)}$ x 2. This array does not need to be initialized before starting the decoding process.

An array to store a list of states determined during traceback (term to be explained below). It is called the state sequence array. The dimensions of this array are (K x 5) + 1. This array does not need to be initialized before starting the decoding process.

Puncturing is a common way of achieving higher code rates, i.e. larger ratios of k to n. Punctured codes are created by first encoding data using a rate 1/n encoder and then deleting some of the channel symbols at the output of the encoder. The process of deleting some of the channel output symbols is called puncturing. For example, to create a rate 3/4 code from the rate 1/2 code described, one would simply delete channel symbols in accordance to the following pattern,

1	0	1
1	1	0

Table (2.9) puncturing pattern

Where a one indicates that a channel symbol is to be transmitted, and a zero indicates that a channel symbol is to be deleted. To see how this make the rate be 3/4, think of each column of the above table as corresponding to a bit input to the encoder, and each one in the table as corresponding to an output channel symbol. There are three columns in the table, and four ones. You can even create a rate 2/3 code using a rate 1/2 encoder with the following puncturing pattern:

1	1
1	0

Table (2.10) puncturing pattern for K=2/3

Which has two columns and three ones.

To decode a punctured code, one must substitute null symbols for the deleted symbols at the input to the Viterbi decoder. Null symbols can be symbols quantized to levels corresponding to weak ones or weak zeroes, or better, can be special flag symbols that when processed by the ACS circuits in the decoder, result in no change to the accumulated error metric from the previous state.

Chapter 3

Channel Coding (Part 2)

3.1 Concatenated Coding

Concatenated coding is illustrated in Figure 3.1. Here we see the information frame illustrated as a square - assuming block interleaving - and we see the parity from the vertical encoding and the parity from the horizontal encoding. For serial concatenation the parity bits from one of the constituent codes are encoded with the second code and we have parity of parity. If the codes are working in parallel, we do not have this additional parity. The idea of concatenated coding fits well with Shannon's channel coding theorem, stating that as long as we stay on the right side of the channel capacity we can correct everything - if the code is long enough. This also means that if the code is very long, it does not have to be optimal. The length in itself gives good error correcting capabilities, and concatenated coding is just a way of constructing - and especially decoding - very long codes.

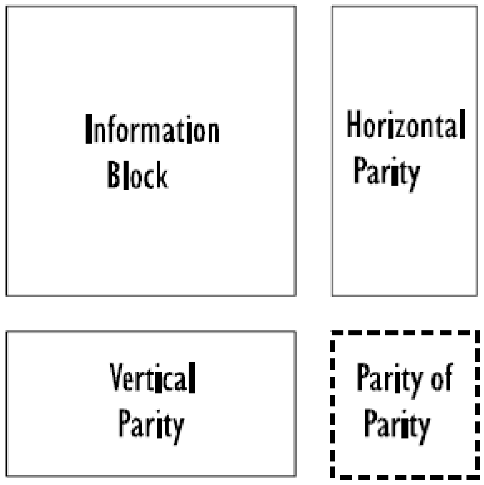


Figure 3.1 Concatenated coding

There are many papers discussing the Turbo codes . The first one is related to the Maximum A Posteriori (MAP) decoding algorithm for convolutional codes which was proposed in 1974 by Bahl et al., but initially received very little attention because of its increased complexity over alternative convolutional decoder for a minimal advantage in bit error rate (BER) performance. Recently, however, an iterative decoder developed by Berrou et al. in 1993 has enjoyed renewed and greatly increased attention. They considered the iterative decoding of two Recursive Systematic Convolutional (RSC) codes concatenated in parallel through a non-uniform interleaver. For decoding the component codes they used a Soft Input/Soft Output (SISO) decoder based on the MAP algorithm. Since then, researchers around the world have investigated the performance and design of Turbo codes. Both serial and parallel concatenated convolutional codes have been studied. The interleaver patterns with high performance also have been evaluated. In iterative decoding, several decoding algorithms have been used, including the optimal MAP symbol estimation and its simplification called the max-log-MAP algorithm (Additive MAP Algorithm) and . A further simplification of log MAP is offered by the modified soft-output Viterbi algorithm (SOVA) which works in a sliding-window SISO decoding algorithm.

This thesis evaluates the BER performance of Turbo codes Additive White Gaussian Noise (AWGN) and slow fading channels. In addition, this thesis presents more accurate approximation in the modified Turbo code decoding algorithm under slow Rayleigh fading channel and corrects the error in the decoding algorithm under AWGN channel presented in.

3.2 TURBO ENCODER

In this chapter, a system model of Turbo code is introduced and is briefly discussed, then focus on a general Turbo encoder which consists of the RSC encoder, interleaver, and puncturing.

3.2.1 Turbo code system model

Figure 3.2 shows the basic elements of a communication system with Turbo code. The source generates an information sequence of N symbols with a constant a priori probability distribution $P(u_k = u)$. The u_k denotes the trasmitted symbol at time k with value (0, 1), i.e., $u_k \in \{0,1\}$. The u_k is encoded by two RSC encoders whose trellis states start at state $s_0(i)$, e.g. $s_0(1)$, and terminal at the final state $s_k(i) = s_N(1)$, which the final state returns to the starting state for encoding the next information block. k and i denote time index and state index, respectively. The encoder generates a sequence of N output coded symbols c_N. The trellis state structure is supposed to be known at the receiver side. $c_k = \{c_k^1, c_k^2, \ldots, c_k^q, c'^2_k, c'^3_k, \ldots, c'^q_k\}$ represents one coded symbol at time k with a length of 2q-1, where $c_k^1, c_k^2, \ldots, c_k^q$ are generated by the first RSC encoder, and $c'^2_k, c'^3_k, \ldots, c'^q_k$ are generated by the second RSC encoder. Each element c_k^l is binary signal, i.e., $c_k^l \in \{-1,1\}, l = 1, \cdots, q$. After modulation, the coded symbols are mapped one by one into transmitted signals x_k. $x_k = \{x_k^{1,s}, \mathrm{x}_k^{2,p}, \ldots, x_k^{q,p}, \mathrm{x}'^{2,p}_k, \mathrm{x}'^{3,p}_k, \ldots, x'^{q,p}_k\}$ represents the transmitted codeword at time k. $x_k^{1,s}$ and $\mathrm{x}_k^{2,p}, \mathrm{x}_k^{3,p}, \ldots, x_k^{q,p}, \mathrm{x}'^{2,p}_k, \mathrm{x}'^{3,p}_k, \ldots, x'^{q,p}_k$ are the systematic bit and the parity check bits for the kth symbl, respectively. The signal x_k is transmitted over the stationary memoryless channel. At the destination, the decoder will evaluate the demodulator output y based on the statistic characteristic of the channel, i.e., the conditional probability density function (pdf) of y_k, $p(y/c) \underline{\underline{\Delta}} \; p(Y_k = y / C_k = c]$,

$y_k = \{y_k^{1,s},\ y_k^{2,p}, \cdots, y_k^{q,p}, y'^{2,p}_k, \cdots, y'^{q,p}_k\}$ represents the received symbol at time k, and then make a decision.

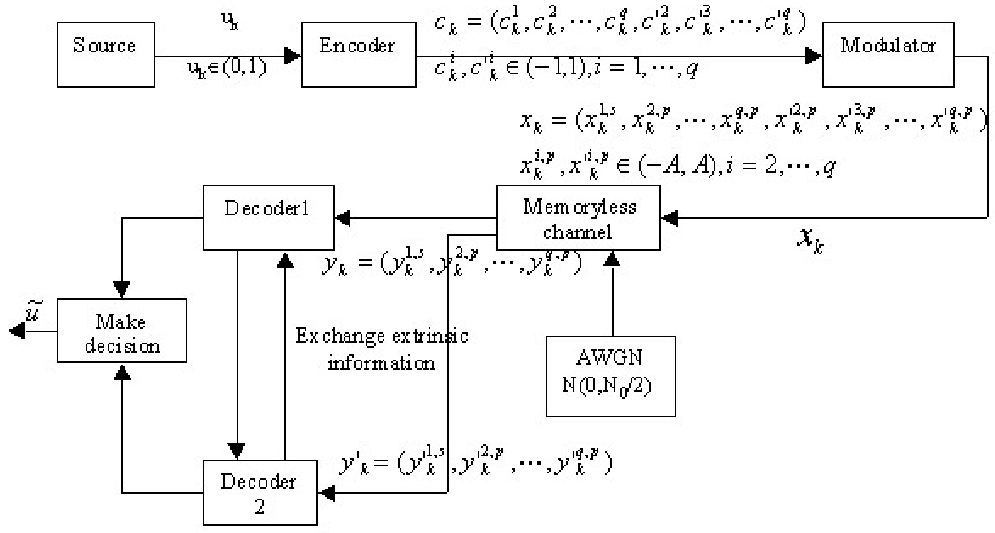

Source
u_k
$u_k \in (0,1)$
Encoder
$c_k = (c_k^1, c_k^2, \cdots, c_k^q, c'^2_k, c'^3_k, \cdots, c'^q_k)$
$c_k^i, c'^i_k \in (-1,1), i = 1, \cdots, q$
Modulator
$x_k = (x_k^{1,s}, x_k^{2,p}, \cdots, x_k^{q,p}, x'^{2,p}_k, x'^{3,p}_k, \cdots, x'^{q,p}_k)$
$x_k^{i,p}, x'^{i,p}_k \in (-A, A), i = 2, \cdots, q$
Memoryless channel
x_k
Decoder1
$y_k = (y_k^{1,s}, y_k^{2,p}, \cdots, y_k^{q,p})$
Exchange extrinsic information
AWGN N(0,N$_0$/2)
$\tilde{u}$
Make decision
Decoder 2
$y'_k = (y'^{1,s}_k, y'^{2,p}_k, \cdots, y'^{q,p}_k)$

Figure (3.2) Block Diagram of Turbo Code System

3.2.2 General Turbo code encoder

A general Turbo encoder is shown in Figure 3.3. The Turbo code encoder employs two identical systematic recursive convolutional encoders connected in parallel with an interleaver (the “Turbo interleaver”) preceding the second recursive convolutional encoder. The two recursive convolutional encoders are called the constituent encoders of the Turbo encoder. The information bits are encoded by both RSC encoders. The first encoder operates on the input bits in their original order, while the second encoder operates on the input bits as permuted by the Turbo interleaver. If the input symbol is of length 1 and output symbol size is R, then the encoder is of code rate r_c=1/R. The information bits are always transmitted across the channel. Depending on the code rate desired, the parity bits from the two constituent encoders are punctured before transmission. The tail bits will be added at the end of the transmitted frame.

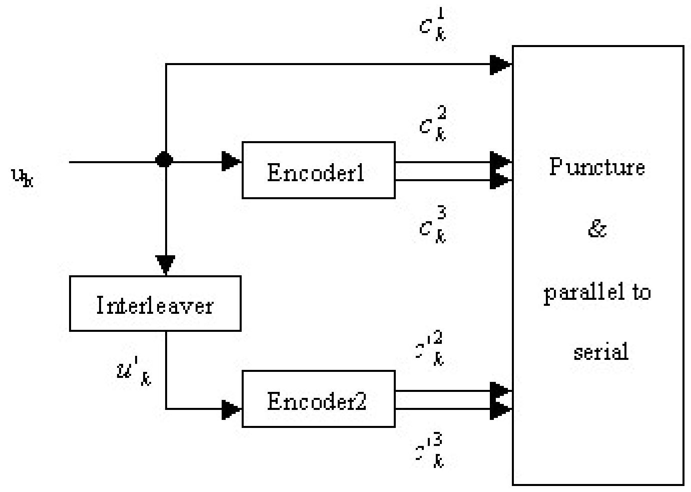


Figure (3.3) General Turbo encoder

3.2.2.1 RSC encoder

Figure 3.4 shows a complete RSC encoder used in Turbo encoder. u_k is an input information bit, c_k is the coded output symbol which is represented by $c_k = \{c_k^1, c_k^2, c_k^3\}$. The system constraint length K is 4. The generator connection matrix is $G(D) = \left[1, \frac{1+D+D^3}{1+D^2+D^3}, \frac{1+D+D^2+D^3}{1+D^2+D^3}\right]$ or [1, 15/13, 17/13] in octal. Its trellis states are shown in figure 2.3.2.

A systematic convolution code is generated by passing the information sequences to be transmitted through a linear finite-state shift register. The input data to the encoder is shifted into the register. Hence codewords are generated. The encoder is defined by a set of generator polynomials $G(D) = \left[1, \frac{g_1(D)}{g_b(D)}, \frac{g_2(D)}{g_b(D)}, \cdots, \frac{g_m(D)}{g_b(D)}\right]$ and memory size M of the shift register (or the convolutional constraint length K which is equal to M+1). $g_m(D)$ is the mth forward generator and $g_b(D)$ is the feed back generator. The information bits are passed into the encoder frame by frame. The shift register starts at initialized state and end at the initial state. To ensure this happens, extra bits are needed to flush the memory, which are called encoder tail bits. For example, switch SW connects to up when data is inputted, down when tail bit is inputted. If the encoder has code rate 1/3, it maps N data bits to 3N coded bits plus 3M tail bits. The received frame for N bit data will have 3N+3M coded bits.

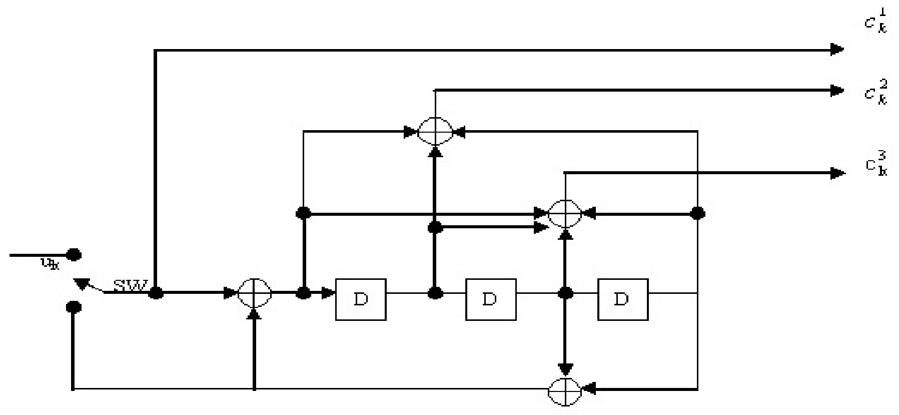


Figure (3.4) A RSC Turbo Encoder with generator

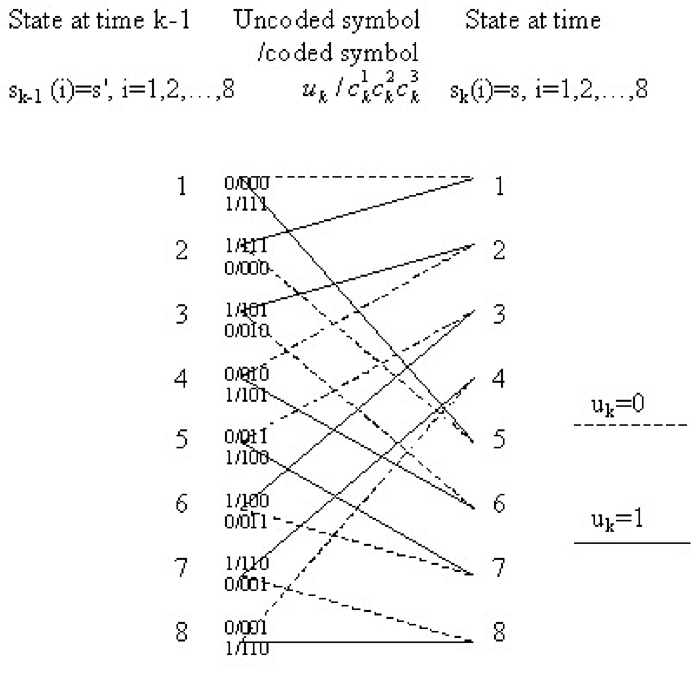


Figure 3.5 An 8 trellis states under a RSC encoder

3.2.2.2 Interleaver

A burst of errors is defined as a sequence of bit errors. The method of interleaver has proved to increase the reliability in the burst error channel. In Turbo code, the structure of interleaver has been carefully chosen. It allows that input sequences for which one encoder produces low weight codewords will usually cause the other encoder to produce high weight codewords. Although the constituent codes are individually weak, the combination is surprisingly powerful. The resulting code has features similar to a “random” block code with Z information bits.

3.2.2.3 Trellis Termination

After encoding block of input data encoder is required to be drawn in state zero. Knowing the final state helps decoder to perform decoding process successfully. Extra bits are inputted to decoder to draw encoder to zero state. These bits are called trellis terminating bits.

3.2.2.4 Output puncturing

The role of Turbo code puncturing is to periodically delete selected bits to reduce coding overhead. In Turbo code, the systematic bits c^1 in constituent encoder 1 are always transmitted for the non-tail bits, the systematic bits c'^1 in constituent encoder 2 are always punctured for the non-tail bits. Other puncturing patterns depend on the code rate selected .

3.3 Interleavers for Turbo Encoder

The interleaver design is a key factor which determines the good performance of a turbo code. Some interleaver types used in turbo codes are presented in the following sections.

3.3.1 “Row–Column” interleaver

The simplest interleaver is a memory in which data is written row–wise and read column–wise. This is called a “row–column” interleaver and belongs to the class of “block” interleavers. For example, data is written as shown in Table below 3.1

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9
x_{10}	x_{11}	x_{12}
x_{13}	x_{14}	x_{15}
x_{16}	x_{17}	x_{18}
x_{19}	x_{20}	x_{21}

Table 3.1 row column interleaver

The interleaving process consists in reading data as shown in Table 3.2

x_1	x_4	x_7	x_{10}	x_{13}	x_{16}	x_{19}	x_2	x_5	x_8	x_{11}	x_{14}	...	...	...

Table 3.2 output of row column interleaver

A few interesting constructions for other block interleavers are given below.

3.3.2“Helical” interleaver

A “helical” interleaver writes data row–wise as in Table 3.1 but reads data diagonal–wise as shown in Table 3.3

x_{19}	x_{17}	x_{15}	x_{10}	x_8	x_6	x_1	x_{20}	x_{18}	x_{13}	x_{11}	x_9	...	...	...

Table 3.3 helical interleaver

3.3.3 “Odd–even” interleaver

We found that for a rate half encoder as, a particular type of interleaver called “odd–even”, gives significant improvements when used in a turbo encoder design. Let us assume that we have a random sequence of binary data input to a rate one half systematic encoder and we store only the odd–positioned coded bits, as shown in Table 3.4

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
y_1	–	y_3	–	y_5	–	y_7	–	y_9	–	y_{11}	–	y_{13}	–	y_{15}

Table 3.4 odd-even interleaver

If we were now to interleave the same sequence of binary data in a pseudo–random order, encode it and store the even–positioned coded bits, the result would be as in Table

x_a	x_b	x_c	x_d	x_e	x_f	x_g	x_h	x_i	x_j	x_k	x_l	x_m	x_n	x_o
–	z_b	–	z_d	–	z_f	–	z_h	–	z_j	–	z_l	–	z_n	–

Table 3.5 odd-even interleaver

The data which is actually sent through the channel is shown in Table 3.6; the original sequence of information bits xi, i = 1,... 15 as in Table 3.4 and a multiplexed sequence of the odd– and even–positioned coded bits from Tables 3.4 and 3.5.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
y_1	z_b	y_3	z_d	y_5	z_f	y_7	z_h	y_9	z_j	y_{11}	z_l	y_{13}	z_n	y_{15}

Table 3.6 output of odd even interleaver

In Table 3.4 all the odd–positioned information bits have their own coded bit. Due to the pseudo–random way of interleaving, some of the coded bits stored in Table 3.4 can be for even–positioned information bits and some for odd–positioned information bits. This means that some of the information bits will have two coded bits associated with them and others will have no coded bit associated with them. Thus, the coding power is not uniformly distributed across all the bits. So for errors which affect information bits not associated with any coded bit the decoder will perform worse in both dimensions.

An example of an "odd–even" type of interleaver is a block interleaver with an odd number of rows and an odd number of columns as in Table 3.7 in which we store row–wise the sequence of random data.

x_1	x_2	x_3	x_4	x_5
x_6	x_7	x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}

Table 3.7 odd even interleaver

We produce the coded bits and store only the odd–positioned coded bits as in Table 3.4. Now we read column–wise, encode and store the even–positioned coded bits as in Table 3.8

x_1	x_6	x_{11}	x_2	x_7	x_{12}	x_3	x_8	x_{13}	x_4	x_9	x_{14}	x_5	x_{10}	x_{15}
–	z_6	–	z_2	–	z_{12}	–	z_8	–	z_4	–	z_{14}	–	z_{10}	–

Table 3.8 odd-even interleaver

In Table 3.8 all the even–positioned information bits have their own coded bit present and in table 3.4 all the odd information bits have their own coded bit present as well. When we multiplex the coded bits from both Table 3.4 and Table 3.8 we produce the coded sequence as in Table 3.9. This means that each of the information bits will have its own associate coded bit associated with it. Thus the coding power is now uniformly distributed.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
y_1	z_6	y_3	z_2	y_5	z_{12}	y_7	z_8	y_9	z_4	y_{11}	z_{14}	y_{13}	z_{10}	y_{15}

Table 3.9 odd-even interleaver

3.3.4“Simile” interleaver

In Section 3.3 we introduced an “odd–even” type of interleaver where each information bit is associated with one and only one coded bit. In this way the correction capability of the code is uniformly distributed over all information bits. We now impose another restriction on the interleaver design: after encoding both sequences of information bits, (the original and the interleaved one), the state of both encoders of the turbo code are to be the same. This allows only one “tail” to be appended to the information bits, which drives both encoders to the same zero state. This is why we called it a ‘‘simile” type of interleaver The idea behind the simile interleaver is that the whole block of N information bits can be divided in _ + 1 sequences, where _ is the memory length of the code. For _ = 2, we get:

Sequence 0 = {dk | k mod __ + 1) = 0}

Sequence 1 = {dk | k mod __ + 1) = 1}

Sequence 2 = {dk | k mod __ + 1) = 2}

For example, consider the particular four state encoder shown in Figure 3.7

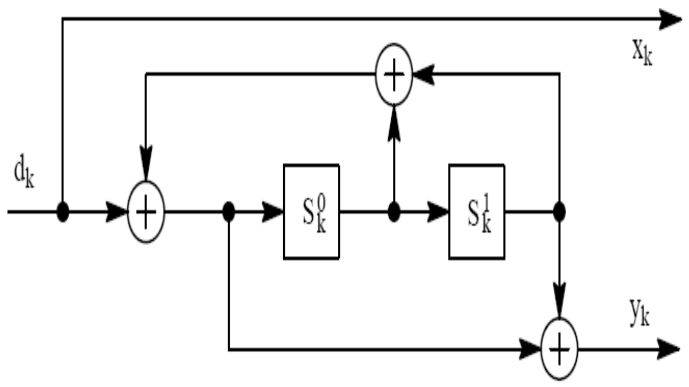


Figure 3.7 Four state RSC encoder

For a given N, the final state of the encoder represented by the state of the two D flip–flops will be a combination of the above sequences as shown in Table

N mod (ν + 1)	S^0_N	S^1_N
0	Sequence1 + Sequence2	Sequence0 + Sequence1
1	Sequence2 + Sequence0	Sequence1 + Sequence2
2	Sequence0 + Sequence1	Sequence2 + Sequence0

Table 3.10 state table of RSC encoder

The important conclusion is that from the point of view of the final encoder state, the order of the individual bits in each sequence does not matter, as long as they belong to the same sequence. The simile interleaver has to perform the interleaving of the bits within each particular sequence in order to drive the encoder to the same state as that which occurs without interleaving.

Since both encoders end in the same state, we need only one tail to drive both encoders to state zero at the same time. The above interleaver types can be combined in a single interleaver. An example of a “simile odd–even” block helical interleaver that can be used with the four state RSC encoder given in Figure 3.7 is shown in Table3.12.

x_1	x_2	x_3	x_4	x_5	x_6
x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}

Table 3.11 odd even simile interleaver

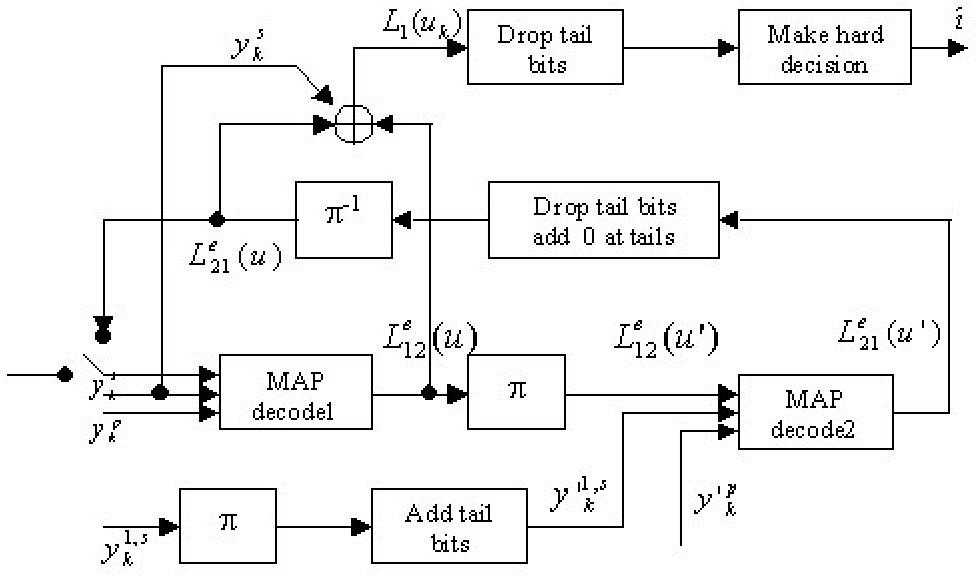
Part of the interleaved sequence is shown in Table

	x_{25}	x_{20}	x_{15}	x_{10}	x_5	x_{30}	x_{19}	x_{14}	x_9	x_4	x_{29}	x_{24}	x_{13}	x_8	x_3	x_{28}	x_{23}	x_{18}	x_7	...
j mod 3	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	...

Table 3.12 interleaved sequence

3.4 Turbo code decoding algorithm under AWGN channel

AWGN channel model is a simple and common channel model in a communication system. It is easier to be studied. In this section, a Turbo code decoding algorithm under AWGN channel will be discussed. Figure 3.8 shows a block diagram of a Turbo decoder.



π is the notation for interleaver,
π^{-1} is the notation for de-interleaver.

Figure3.8 Block Diagram of Turbo decoder

A log ratio of the posteriori probability of u_k conditioned on the received signal y is defined as

$$L(u_k) \triangleq \log\left[\frac{P(u_k = 1 / y_1^N)}{P(u_k = 0 / y_1^N)}\right] \qquad \text{(3-1)}$$

The decoding decision of $\tilde{u}_k$ is made based on the sign of $L(u_k)$, i.e.,

$$\tilde{u}_k = sign[L(u_k)]. \qquad \text{(3-2)}$$

$L(u_k)$ is computed by three terms which are L_apriori, L_channel , and $L^e(u_k)$. L_apriori is a priori information based on the input bit u_k at time k. It is provided by the previous decoder. L_channel is the received systematic bit at time k, referring to Appendix A for details.

$$L(u_k) = \left[L^e(u_k) + Lc \cdot y_k^{1,s}\right] + \log \frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s', s)}{\sum_{u^-} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s', s)}$$

$$= L\_apriori + L\_channel + L^e(u_k), \qquad \text{(3-3)}$$

Where L_ aprior and L_channel denote $L^e(u_k)$ and $Lc \cdot y_k^{1,s}$ respectively. $\sum_{u^+}()$ is the summation over all the possible transition branch pair (s_{k-1}, s_k) at time k given input u_k=1 and $\sum_{u^-}()$ is the summation over all the possible transition branch pair (s_{k-1}, s_k) at time k given input u_k=0. Lc is the channel reliable factor, its computation is given as the following,

$$Lc = \frac{4 \cdot A \cdot SNR\_b}{p}, \qquad \text{(3-4)}$$

where A=1 for AWGN channel, SNR_b is the uncoded bit-energy-to-noise-ratio ($\frac{E_b}{N_0}$), p denotes $1/r_c$, r_c is code rate of the Turbo encoder.

$L^e(u_k)$ is an extrinsic information based on all parity and systematic information except the systematic value at time k. It can be passed on to a subsequent decoder. It is computed using the following equations:

$$L^e(u_k) \triangleq \log \frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s',s) \cdot \tilde{\beta}_k(s)}{\sum_{u^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s',s) \cdot \tilde{\beta}_k(s)}, \qquad (3\text{-}5)$$

where

$$\gamma^{\mathrm{e}}(s',s) = \exp\left[\sum_{i=2}^{q}\left(Lc \cdot \frac{1}{2} \cdot y_k^{i,p} \cdot c_k^i\right)\right]. \qquad (3\text{-}6)$$

$\tilde{\alpha}_k(s),\ \tilde{\beta}_{k-1}(s')$ can be computed recursively with initial conditions described below:

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s',s)},$$

(3-7)

$$\tilde{\alpha}_0(s) = \begin{cases} 1 & \text{if s} = 1 \\ 0 & otherwise \end{cases}.$$

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s} \tilde{\beta}_k(s) \cdot \gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-2}(s') \cdot \gamma_{k-1}(s',s)},$$

(3-8)

$$\tilde{\beta}_N = \begin{cases} 1 & \text{if s} = 1 \\ 0 & \text{otherwise} \end{cases}.$$

$$\gamma_k(s',s) \propto \exp\left[\frac{1}{2} \cdot L^e(u_k) \cdot u_k + Lc \cdot \frac{1}{2} \cdot y_k^{1,s} \cdot c_k^1\right] \exp\left[\sum_{i=2}^{q}\left(Lc \cdot \frac{1}{2} \cdot y_k^{i,p} \cdot c_k^i\right)\right]. \qquad (3\text{-}9)$$

For example, at any given iteration, decoder 1 $L_1(u_k)$ is computed as

$$L_1(u_k) = Lc \cdot y_k^{1,s} + L_{21}^e(u_k) + L_{12}^e(u_k), \qquad (3\text{-}10)$$

$$\tilde{u}_k = \mathrm{sign}[\mathrm{L}_1(u_k)],$$

Where $L_1(u_k)$ is given in equation (3-3). $L_{21}^e(u_k)$ is extrinsic information for decoder 1 derived from decoder 2, and $L_{12}^e(u_k)$ is the third term in equation (3-3) which is used as the extrinsic information for decoder 2 derived from decoder 1. The decoders are sharing the information with each other. The value of $L_1(u_k)$ decides the degree of the reliability of $\tilde{u}_k$.

CHAPTER 4

MODULATION, MATCH FILTERING AND PULSE SHAPING

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

4.1 COMPLEX ENVELOP REPRESENTATION

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

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

where $a(t)$ is the time-varying amplitude or envelope of the modulated signal and $\theta(t)$ is the time-varying phase. ω_c is called the carrier frequency (in radians/sec).

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

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

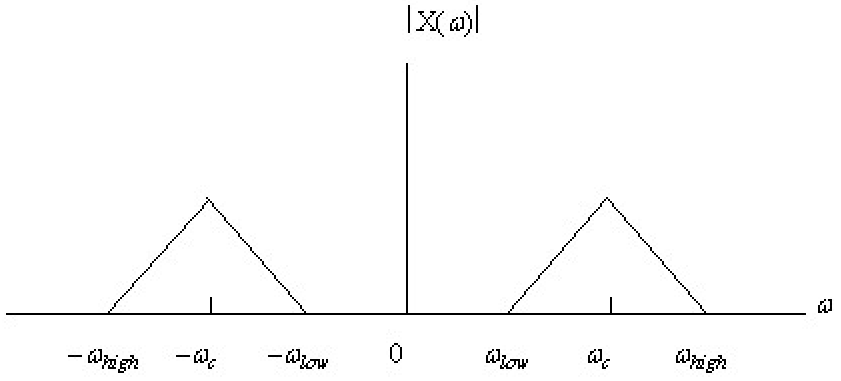


Figure 4.1: Fourier spectrum of Bandpass signal

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

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

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

In equation 4.4, the inverse tangent is taken with the polarities of the numerator and denominator independently known, so there is no quadrant ambiguity in computing θ(t). In passband processing and analysis, the objective is to eliminate explicit consideration of the carrier frequency ω_c and directly analyze systems using only the inphase and quadrature components. These inphase and quadrature components can be combined into a two-dimensional vector, or into an equivalent complex signal. By convention, a graph of a quadrature-modulated signal plots the inphase component along the real axis and the quadrature component along the imaginary axis as shown in figure 4.2. The resultant complex vector $x_{bb}(t)$ is known as the complex baseband-equivalent signal. The complex baseband-equivalent signal for x(t) in equation 4.1 is

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

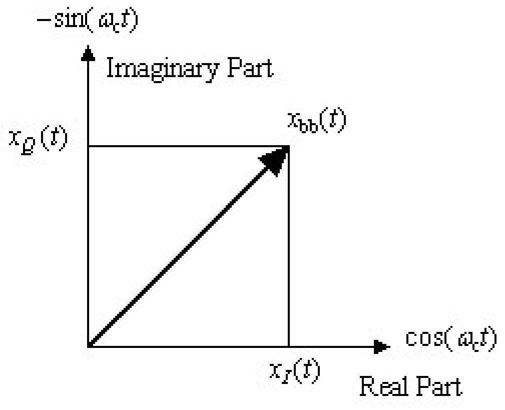


Figure 4.2: Decomposition of baseband-equivalent signal.

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

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

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

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

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

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

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

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

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

4.1.2 Spectrum of Analytic and Baseband-Equivalent Signals

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

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

Taking the Fourier Transform of both sides of equation 4.11 yields

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

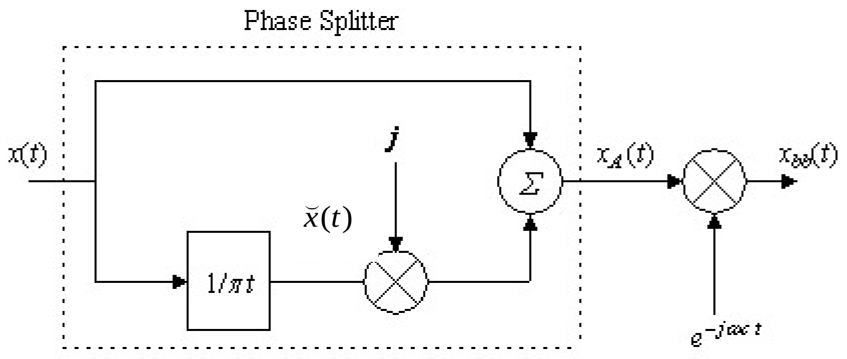


Figure 4.3: Complex baseband signal recovery from real passband signal

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

$x_A(t)$ is truly equivalent to the original signal x(t). Using equation 4.6, the Fourier transform of the baseband equivalent signal is simply the Fourier transform of the analytic signal translated in frequency ω. Thus

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

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

Use of equations 4.6 and 4.7 allows reconstruction of the signal x(t) from the baseband equivalent signal $x_{bb}(t)$ and the carrier frequency ω_c. The baseband equivalent signal, in general, may be complex-valued, and thus as shown in figure 4.4 the spectrum of $x_{bb}(t)$ may be asymmetric about the origin $\omega = 0$

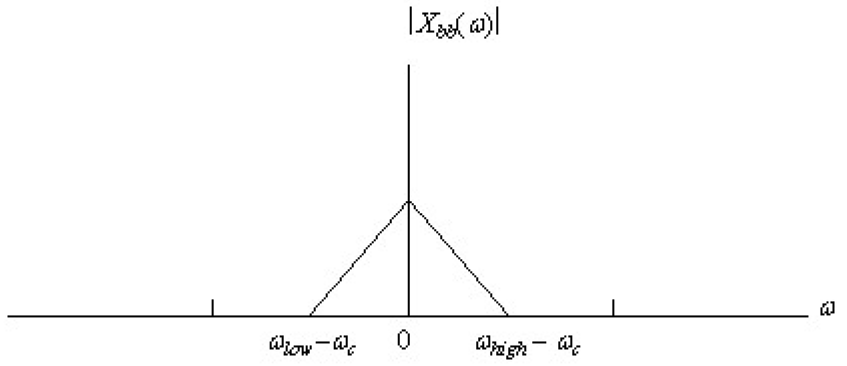


Figure 4.4: Baseband signal spectrum.

4.1.2 Generation of the baseband equivalent

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

4.2 DIGITAL BANDPASS MODULATION

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

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

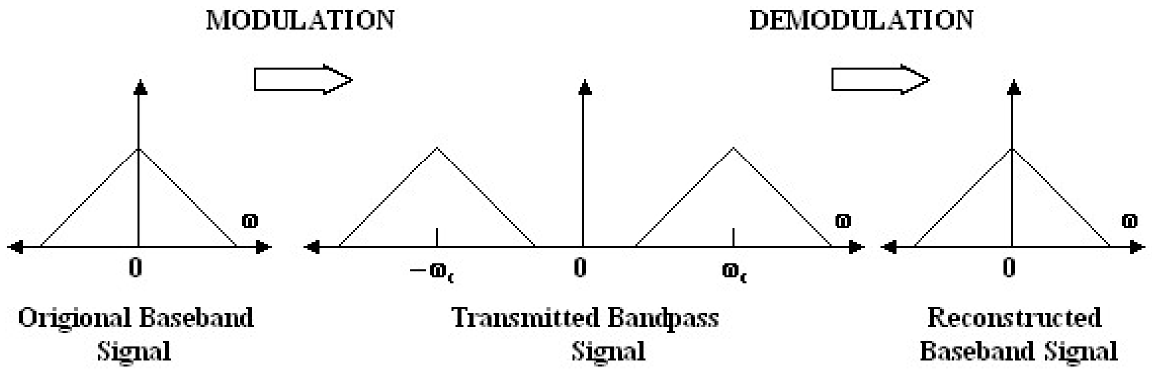

MODULATION
DEMODULATION
ω
0
−ω_c
ω_c
Origional Baseband Signal
Transmitted Bandpass Signal
Reconstructed Baseband Signal

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

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

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

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

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

so that

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

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

4.2.1 Phase Shift Keying

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

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

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

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

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

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

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

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

4.2.2 QPSK Transmitter

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

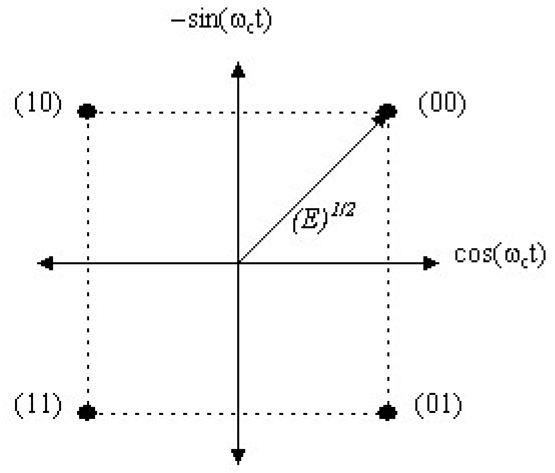


Figure 4.6: QPSK Constellation diagram

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

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

where

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

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

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

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

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

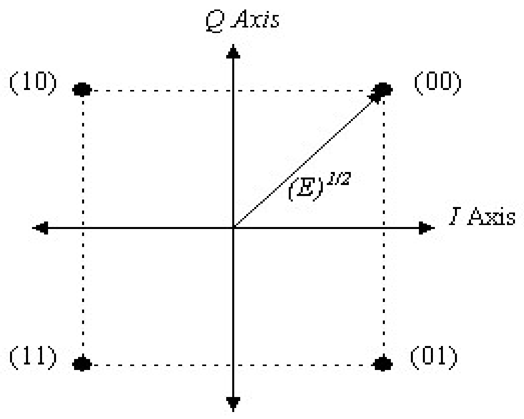


Figure 4.7: QPSK Constellation diagram for complex baseband representation.

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

Table 4.1: QPSK lookup table

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

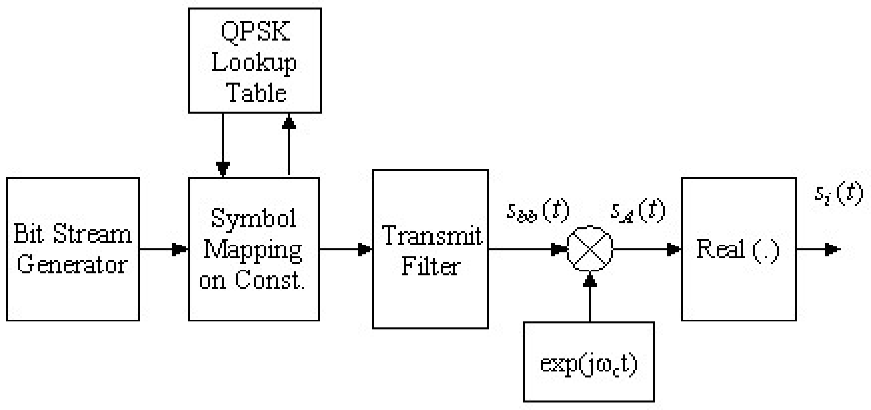


Figure 4.8: QPSK Transmitter

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

4.2.3 QPSK Detection

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

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

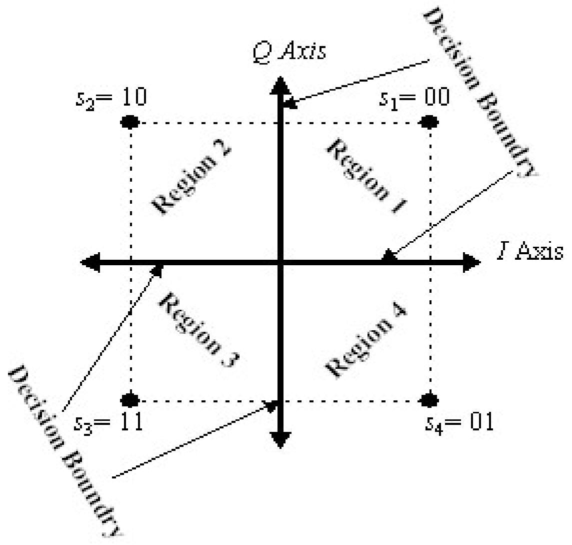


Figure 4.9: Decision regions for QPSK signal

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

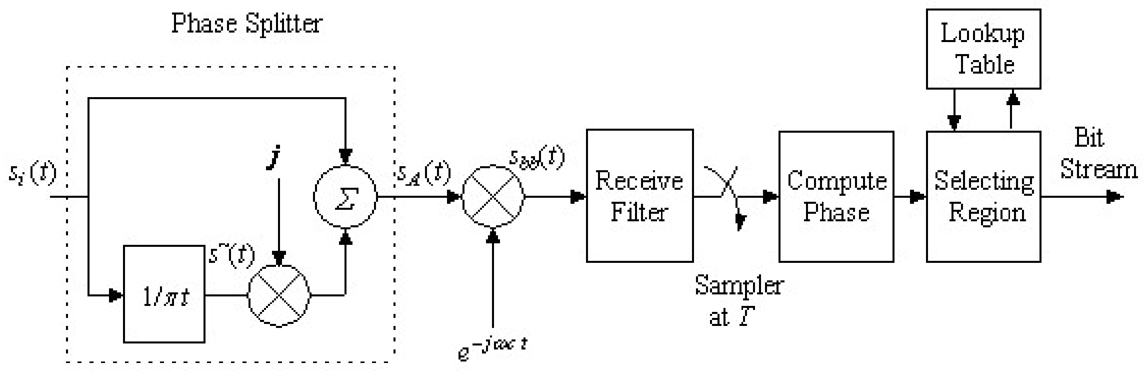


Figure 4.10: QPSK Receiver structure

So far we have explained complex envelop representation of bandpass signals and QPSK modulation/demodulation. From now on we will not use the receiver structure explained above but only the complex baseband part will be used. The frequency translation part will be removed because it the same for all. The baseband receiver structure is given in figure 4.11.

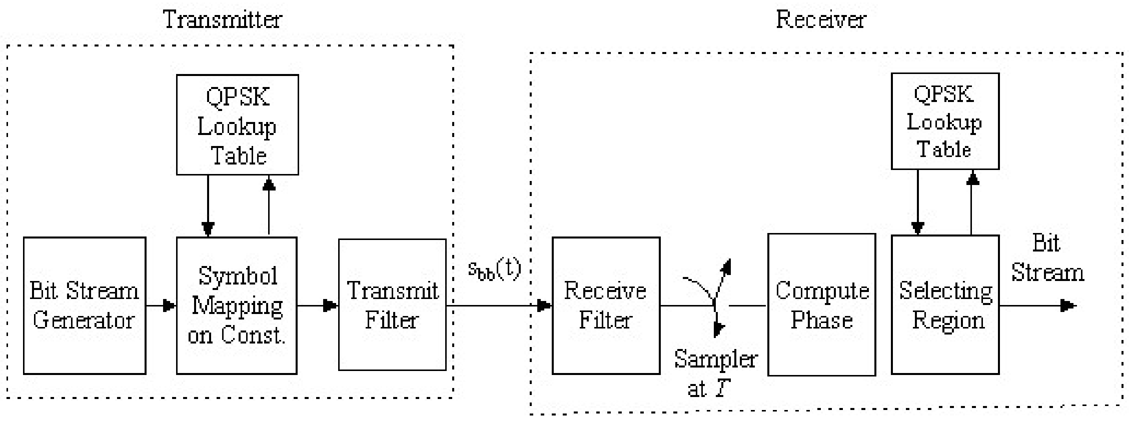


Figure 4.11: Baseband model for QPSK communication system

4.3 DETECTION IN ADDITIVE WHITE GAUSSIAN NOISE (AWGN)

4.3.1 Additive White Gaussian Noise

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

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

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

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

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

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

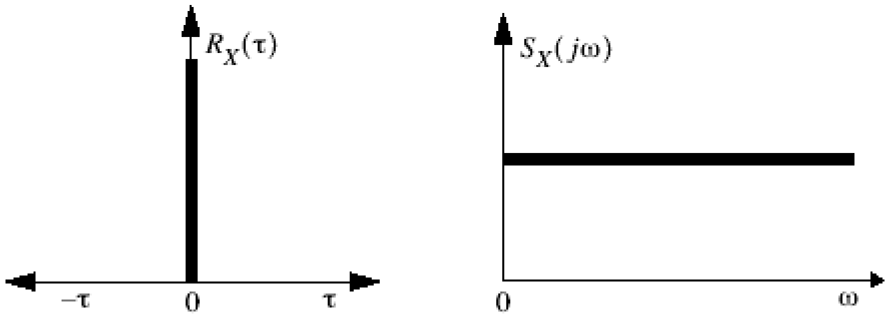


Figure 4.12: White noise shown in both the time (left) and frequency domain (right).

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

4.3.2 SNR Maximization with a Matched Filter

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

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

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

The variance of the output noise (average noise power) is denoted by $\sigma_0{}^2$□, so that the ratio of the instantaneous signal power to average noise power. $(S/N)_T$. at time t = T, out of the sampler in step 1, is

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

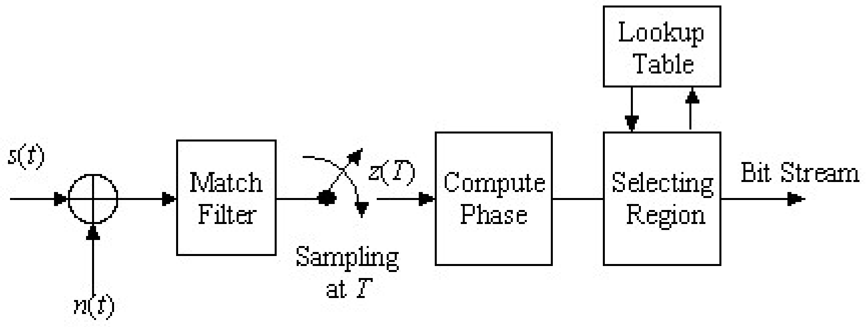


Figure 4.13: QPSK receiver with sampler at symbol rate *T* and match filter

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

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

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

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

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

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

We next find that value of $H(f) = H_o(f)$ for which the maximum $(S/N)_{\mathrm{T}}$ is achieved, by using *Schwarz's inequality.* One form of the inequality can be stated as

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

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

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

Substituting into Equation 4.30 yields

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

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

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

Thus, the maximum output $(S/N)_{\mathrm{T}}$ depends on the input *signal energy* and the power spectral density of the noise, *not on the particular shape* of the waveform that is used. The equality in Equation 4.33 holds only if the optimum filter transfer function $H_{\mathrm{O}}(f)$ is employed, such that

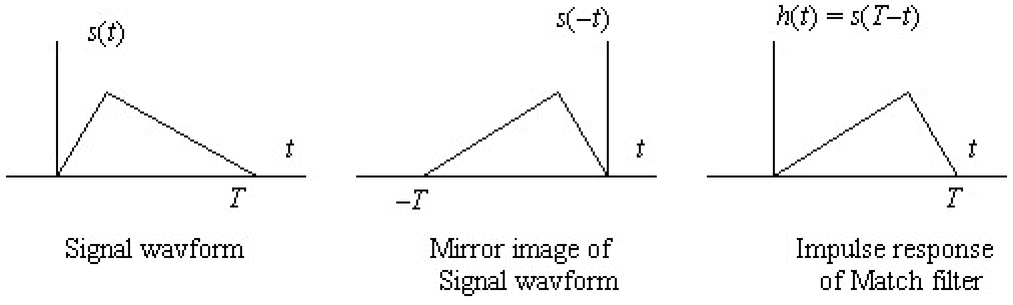
$$H(f) = H_O(f) = kS^*(f)e^{-j2\pi Tf} \tag{4.35}$$

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

Since $\mathrm{S}(\mathrm{t})$ is a real-valued signal, we can write,

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

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



s(t)
t
T
Signal wavform
s(−t)
t
−T
Mirror image of Signal wavform
h(t) = s(T−t)
t
T
Impulse response of Match filter

Figure 4.14: Impulse responses of received signal and match filter

The above mathematical discussion proves that if the impulse responses of the received signal and the match filter are mirror images of each other then at $t=T$ the SNR is maximized. Actually the convolution with itself is a process of integration. By match filtering we are actually integrating the received signal. AWGN is a zero mean random variable. By averaging we are trying to force it to zero. It can be verified that as T approaches ∞ the noise averaged to zero.
The QPSK receiver with match filter can be redrawn in figure 4.15.

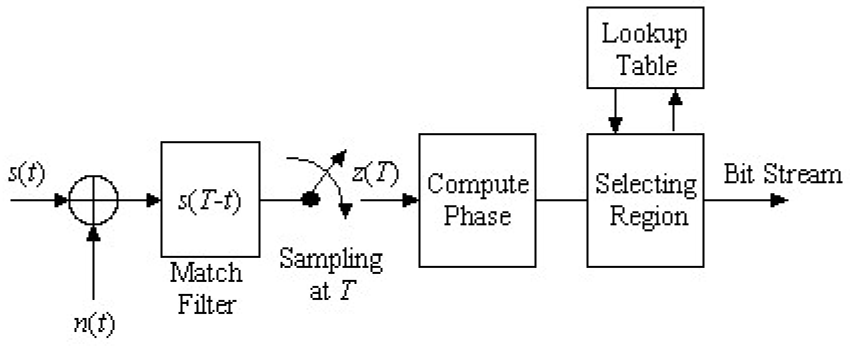


Figure 4.15: QPSK receiver with match filter

4.4 INTERSYMBOL INTERFERENCE AND PULSE SHAPING

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

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

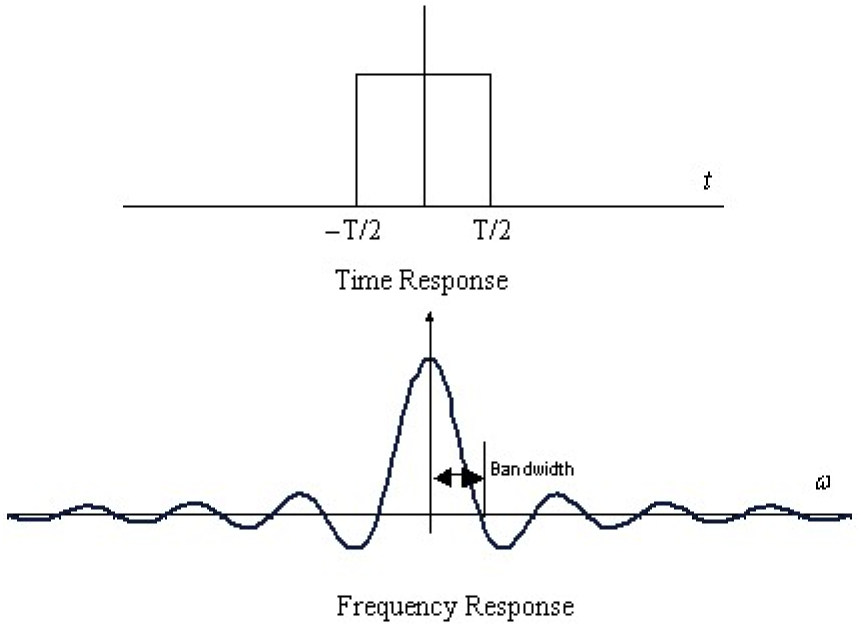


Figure 4.16: Time and frequency response of rectangular pulse

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

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

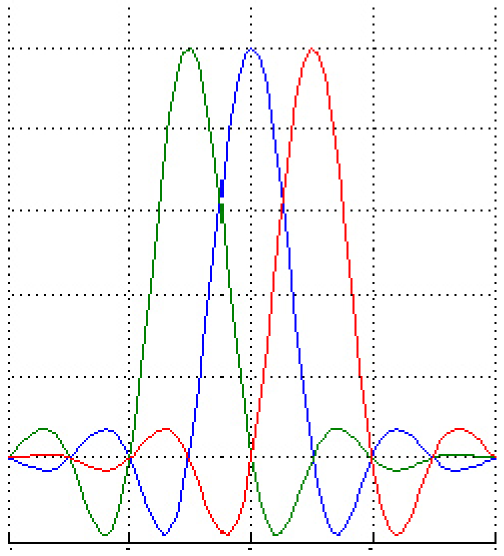


Figure 4.17: pulse shaping using sinc.

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

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

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

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

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

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

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

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

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

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

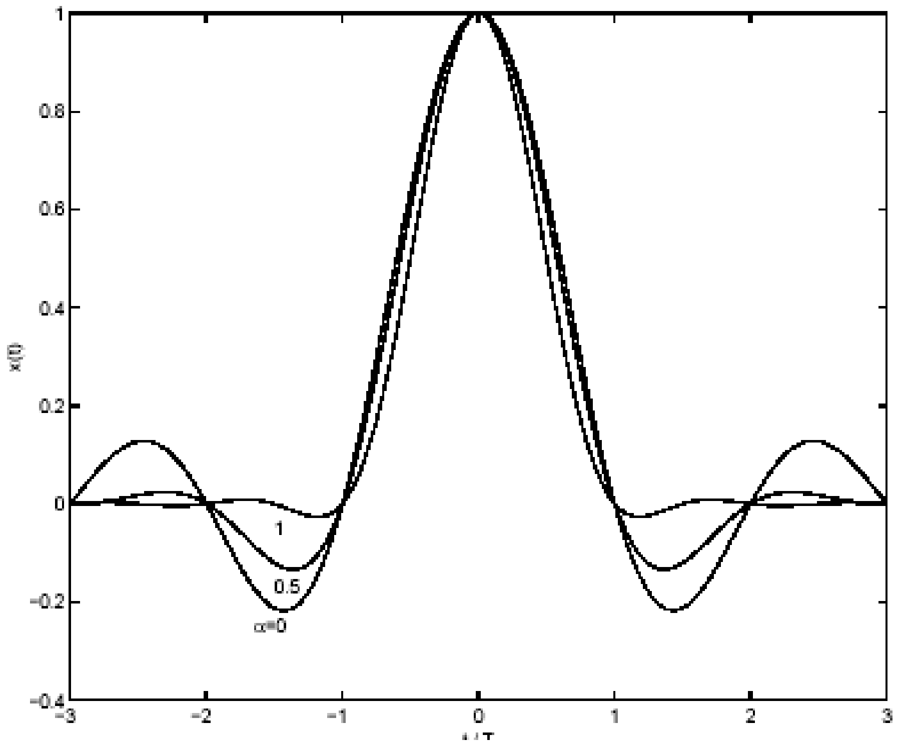


Figure 4.18: Impulse responses of raised cosine filter with $\alpha = 0, 0.5, 1$

In frequency domain, the relationship is given by

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

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

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

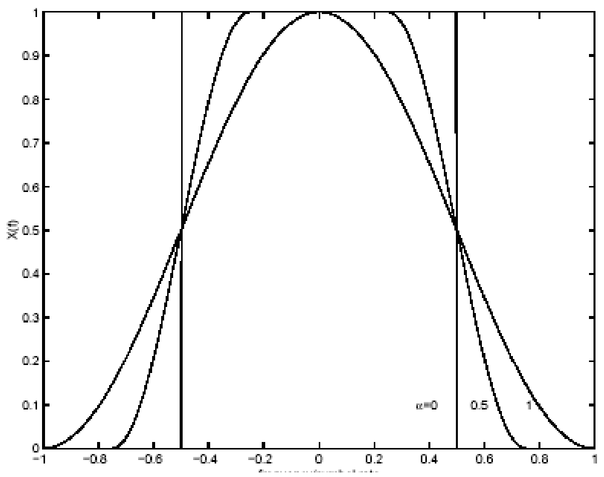


Figure 4.19: Frequency responses of raised cosine filter with $\alpha = 0,0.5,1$

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

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

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

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

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

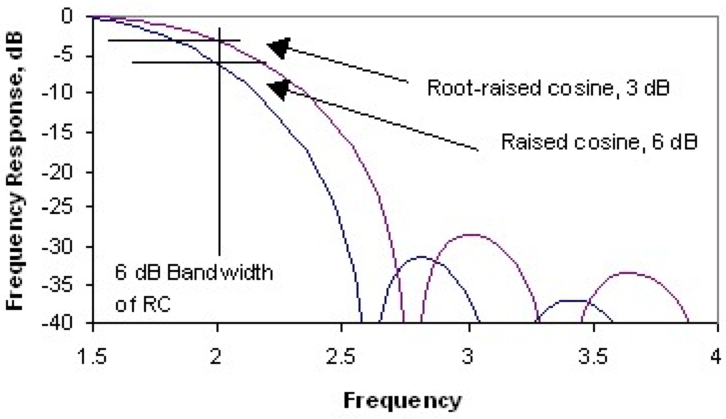


Figure 4.20: Frequency responses of raised cosine and rootraised cosine filters

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

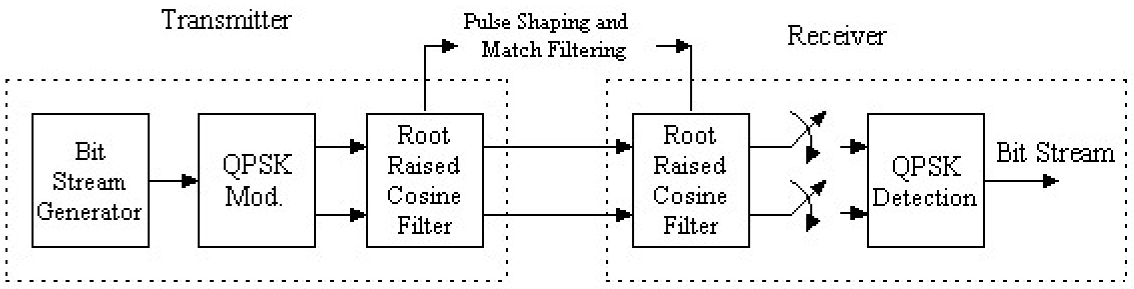

Transmitter
Pulse Shaping and Match Filtering
Receiver
Bit Stream Generator
QPSK Mod.
Root Raised Cosine Filter
Root Raised Cosine Filter
QPSK Detection
Bit Stream

Figure 4.21: Baseband model of QPSK communication system

4.5 DISCRETE TIME MODEL OF COMMUNICATION SYSTEM

So far we have explained discrete partial analog communication system. In figure 4.20 the only analog portion is raised cosine filter. If we realize it in discrete time the complete system will become digital. We have already explained the upsampling using interpolation in a previous chapter. We know that the raised cosine filter requires the excess bandwidth. So in order to filter the QPSK symbol we have to upsample by at least factor of 2 because the maximum excess bandwidth is equal to the Nyquist bandwidth. The upsampling is done by the insertion of $M-1$ zeros in between the QPSK symbol stream and then interpolation is done by the digital raised cosine filter. If we are sampling by a factor of M then we will say that we have M samples per symbol. At the receiver we have to downsample by a factor of M in order to recover the QPSK symbol stream. The complete discrete time model is given below

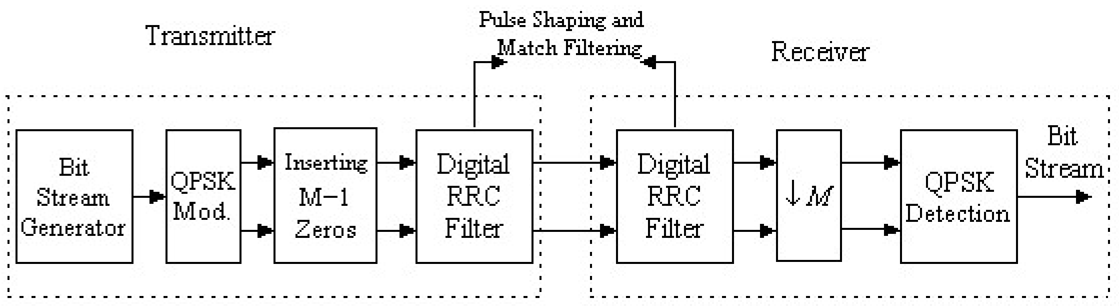


Figure 4.22: Discrete time baseband model of QPSK communication system

4.6 SIMULATION RESULTS

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

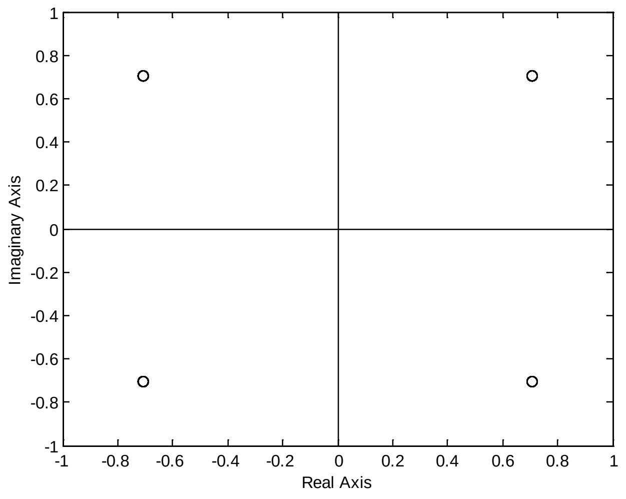


Figure 4.23: QPSK Constellation diagram at transmitter

QPSK constellation diagram after the coherent detection at the receiver is

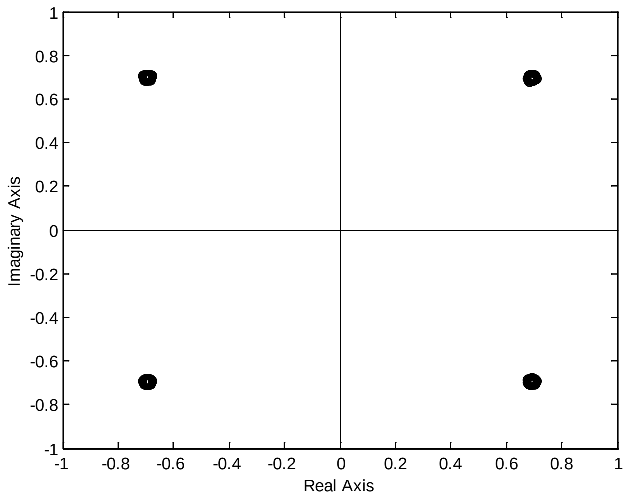


Figure 4.24: QPSK Constellation diagram at receiver

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

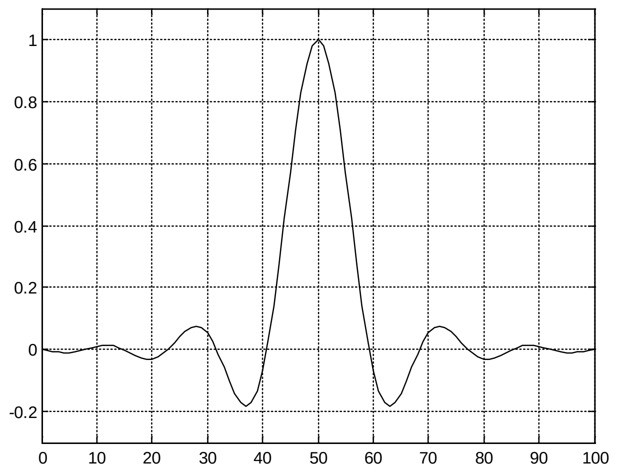


Figure 4.25: Impulse response of root raised cosine pulse

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

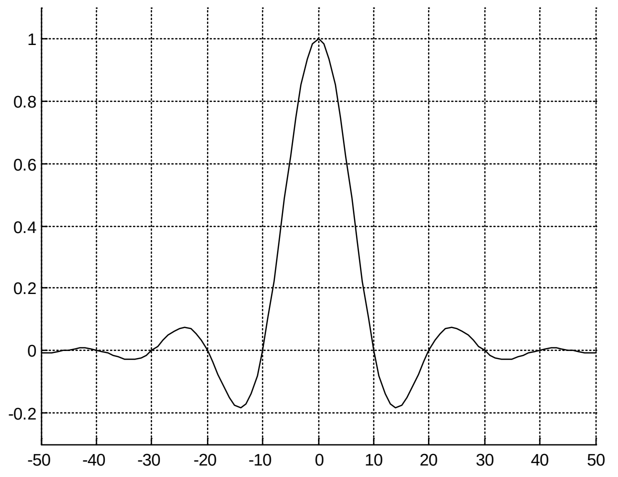


Figure 4.26: Impulse response of root raised cosine pulse

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

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

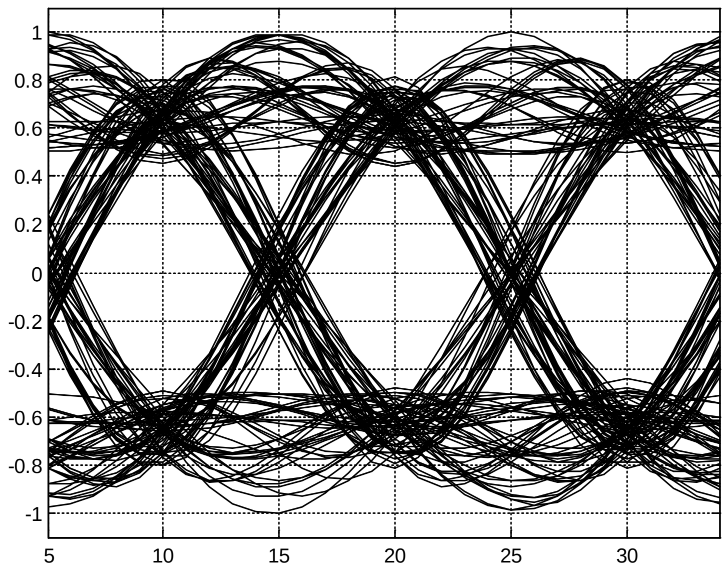


Figure 4.27: Eye diagram at transmitter (there is ISI)

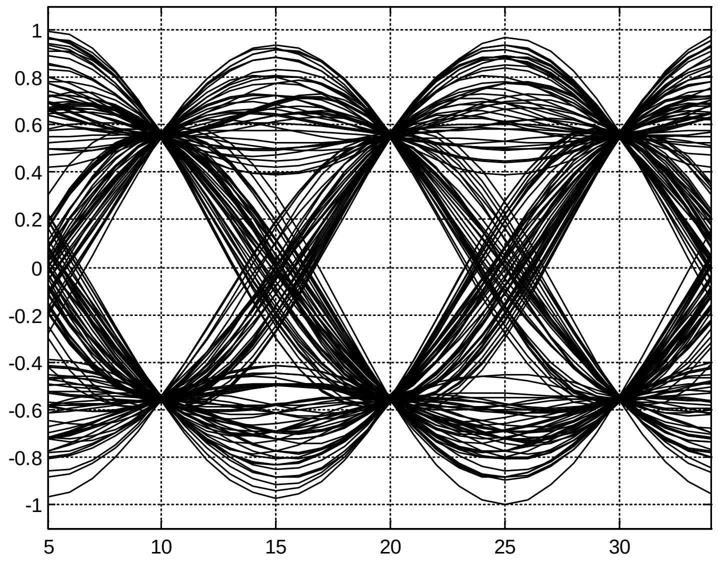


Figure 4.28: Eye diagram at receive (no ISI)

CHAPTER 5

WIRELESS CHANNEL MODELING

5.1 INTRODUCTION

Small scale fading is used to describe the rapid fluctuations of the amplitudes , phases or multipath delays of the radio signal over a short period of time or travel distance , so that large –scale path loss effects may be ignored . Fading is caused by the interference between two or more versions of transmitted signal arrive at the receiver antenna at slightly different times. These waves, called multipath waves, combine at the receiver antenna to give a resultant signals which can vary widely in amplitude and phase , depending on the distributions of the intensity and relative propagation time of the waves and the bandwidth of the transmitted signal.

5.2 Small Scale Multipath Propagation

Multipath in the radio channel creates small scale fading effects. The most important effects are

- Rapid changes in signal strength over a small travel distance or time interval.
- Random frequency modulation due to varying Doppler Shifts on different multipath signals.
- Time dispersion (Echoes) caused by the multipath propagation delays

In urban areas , fading occurs because of the height of the mobile antennas are well below the height of surroundings structures , so there is no single line of sight path to the base station. Even when line of sight exists, multipath still occurs due to reflections from the ground and surrounding structure. The incoming radio waves arrive from different from different directions with different propagation delays. The signal received by the mobile at any point in the space may consist of a large number of plane waves having randomly distributed amplitudes, phases and angles of arrival. These multipath components combine vectorially at the receiver antenna and can cause the signal received by the mobile to distort or fade. Even when the mobile is stationary, and received signal may fade due to movement of surroundings objects in the radio channel.

5.1.1 Factors effecting small scale fading

Many physical factors in the radio propagation channel influence small scale fading.
These include the following.

5.1.1.1 MultiPath Propagation

The presence of the reflecting objects and scatterers in the channel creates a constantly changing environment that dissipates the signal energy in the amplitude, phase and time. These effects results in amplitude versions of the transmitted signal that arrive at receiving antenna , displaced with respect to one another in time and spatial orientation. The random phase and amplitudes of the different multipath components cause fluctuation in the signal strength, thereby including small scale fading, signal distortion, or both.

5.1.1.2 Speed of Mobile

The relative motion between baseband signal and mobile results in random frequency modulation due to different Doppler shift on each multipath component. Doppler shift would be positive and negative depending on whether the mobile receiver is moving toward or away from the base station.

5.1.1.3 Speed of surrounding objects

If objects in the radio channel are in random motion, they induce a time varying Doppler shift on multipath components. If the surroundings objects move at a greater rate than mobile , then this effect dominates the small scale fading otherwise , motion of the surroundings objects may be ignored , only speed of mobile need be considered. The coherence time defines the STATICNESS of the channel and is directly impacted by the Doppler shift.

5.1.1.4 The Transmission bandwidth of the signal

If the transmitted radio signal bandwidth is greater than bandwidth of the multipath channel, the received signal will be distorted, but the received signal strength will not fade much over a local area.

5.2 Parameters of mobile multipath channels

Many multiptah channels are derived from the power delay profile. Power delay profile are generally represented as plots of relative received power as function of excess delay with respect to a fixed time delay reference. Power delay profiles are found by averaging instantaneous power delay measurements over a local area in order to determine an average small scale power delay profile.

5.2.1 Time and Frequency dispersion parameters

In order compare different multipath channels and to develop some general design guidelines for wireless systems, parameters which grossly quantify the multipath channel are used.

5.2.1.1 Delay spread

The Mean excess delay, rms delay spread and excess delay spread are multipath channel parameters that can be determined from a power delay profile. These delays are measured relative to the first detectable signal arriving at the receiver at *To=0.* The *maximum excess delay (X db)* of the power delay profile is defined to be a time delay during which multipath energy falls to *X* db below the maximum. In other words, excess delay is defined is *Tx- To.* Where *To* is 1st arriving signals and *Tx* is the maximum delay at which mulitpath energy falls to *X db* of the strongest arriving multipath signal. The *maximum excess excess delay* (X db) defines the temporal extent of the multipath that is above a particular threshold.

5.2.1.2 Coherence Bandwidth

While the delay spread is natural phenomenon caused by reflected and scattered propagation paths in the radio channel, the *Coherence bandwidth Bc*, is a defined relation derived from the *rms delay spread*. *Coherence bandwidth is statistical measure of the range of frequencies over which the channel can be considered* "*FLAT*" i.e. a channel which passes all spectral components with approximately equal gain and linear phase. In other words coherence bandwidth is range of frequencies over which two frequency components have a strong potential for amplitude correlation. Two sinusoids with frequency separation greater than *Bc* are treated differently by the channel.

5.2.1.3 Doppler spread

Delay spread and coherence bandwidth is parameters which describe the time dispersive nature of the channel in a local area. However, they don't offer information about varying nature of channel caused by either relative motion between mobile and base station or by movement of objects in the channel in the small scale region.

Doppler spread is measure of spectral broadening caused by the time rate of change of the mobile radio channel and defined as the range of frequencies over which the received Doppler spectrum is essentially non zero. When a pure sinusoidal tone of frequency *Fc* is transmitted in the range *Fc-Fd* to *Fc+Fd*, where *Fd* is Doppler shift. The amount of spectral broadening depends upon *Fd* which is function of relative velocity of the mobile and the angle of motion of the mobile and direction of arrival of the scattered waves.

5.2.1.4 Coherence time

Coherence time Tc , time domain dual of Doppler spread and used to characterize the time varying nature of the frequency dispersiveness of the channel in the time domain .The Doppler spread and coherence time are inversely proportional to one another.

Coherence time is actually a statistical measure of the time duration over which the channel impulse response is invariant and quantifies the similarity of the channel response at different times. The definition of coherence time implies that two signals arriving with a time separation greater than Tc are effected differently by the channel.

5.3 Types of small scale fading

Types of fading experienced by signal propagation through a mobile radio channel depends on the nature of the transmitted signal with respect to the characteristic of the channel. Depending on the relation between the signal parameters and the channel parameters and the channel parameters, different transmitted signals will go under different fading. The time dispersion and frequency dispersion mechanism in a mobile radio channel lead to four possible distinct effects.

Mulipath delay spread leads to *time dispersion and frequency selective fading*.

Doppler spread leads to *frequency dispersion and time selective fading*. The tow propagation mechanisms are independent of each other.

5.3.1 Fading Effects due to Multipath Time Delay spread

Time dispersion due to multipath causes the transmitted signal to undergo either flat or frequency selective fading.

5.3.1.1 FLAT FADING

If the mobile radio channel has a constant gain and linear response over a bandwidth which is greater than bandwidth of transmitted signal, then the receive signal will go under flat fading. In flat fading, the multipath structure of the channel is such that spectral characteristics of the transmitted signal are preserved at the receiver . However the strength of received signal changes with time, due to fluctuations in the gain of channel caused by multipath.

Flat fading channels are also known as amplitude varying channels and are sometimes referred to as narrowband channels, since bandwidth applied to signal s narrow as compared to channel flat fading bandwidth. Typical flat channels cause deep fades and they require 20 or 30 db more transmitted power to achieve low bit error rates during times of deep fades as compared to system operating over non-fading channels. The distribution of most common amplitude distributions is *Rayleigh distribution*. The *Rayleigh flat fading channel model* assumes that channel induces amplitude which varies in time according to *Rayleigh distribution.*

Mathematically,

$$\mathbf{Bs \ll Bc \quad or \quad Ts \gg Delay\ spread} \qquad (5.1)$$

Ts is reciprocal of bandwidth or symbol period and *Bs* is bandwidth. *Bc* is coherence bandwidth.

5.3.1.2 Frequency selective Fading

If the channel passes a constant-gain and linear phase response over a bandwidth that is smaller than the bandwidth of transmitted signal, then channel creates frequency selective fading on the received signal. Under such conditions, the channel impulse response has a multipath delay spread which is greater than the reciprocal bandwidth of the transmitted message waveform. When this occurs, the received signal includes multiple version of transmitted waveform which are attenuated and delayed in time, and hence the received signal is distorted.

Frequency selective fading is due to time dispersion of the transmitted symbols within the channel. Thus the channel induces

Intersymbol Interference (ISI). Viewed in the frequency domain, certain frequency components in the received signal spectrum have greater gains than others.

Frequency selective fading channels are much more difficult to model than flat fading channels since each multipath signal must be modeled and the channel must be considered to be a linear filter. Fir such model or fading the spectrum S(f) of the transmitted signal has a bandwidth which is greater than coherence bandwidth Bc of the channel. Viewed from frequency domain, the channel becomes frequency selective, where the gain is different for different frequency components. Frequency selective fading is caused by multipath delays which approach or exceed the symbol period of the transmitted symbol. Frequency selective fading channels are also known as wideband channels since the bandwidth of signal S(t) is greater than bandwidth of channel impulse response. As time varies, the channel varies in gain and phase across the spectrum of S(t), resulting in time varying distortion in the received signal r(t). To summarize, a signal undergoes frequency selective fading if

$$\mathbf{B_s \gg B_c \text{ or } T_s \ll \text{Delay spread}} \qquad (5.2)$$

Ts is symbol period. And *Bc* coherence bandwidth.

A common rule of thumb is that a channel is Flat fading if Ts >= 10*(Delay spread) and Frequency selective if Ts<10*(Delay spread)

5.3.2 Fading Effects due to Multipath Time Doppler spread

Depending upon how rapidly the transmitted baseband signal changes as compared to the rate of change of the channel, a channel mat be classified either as a Fast Fading or Slow Fading.

5.3.2.1 Fast Fading

In this fading, the channel impulse response changes rapidly within the symbol duration. That is, the coherence time of the channel is smaller than the symbol period of the transmitted signal. This causes frequency dispersion due to Doppler spreading, which leads to signal distortion. Viewed in the frequency domain, signal distortion due to fast fading increases with increasing Doppler spread relative to bandwidth of the transmitted signal. Therefore, signal undergoes fast fading if,

$$\mathbf{Ts > Tc} \qquad \mathbf{or} \qquad \mathbf{Bs < Bd} \qquad (5.3)$$

Fast fading only deals with rate of change of channel due to motion. In the case of the flat fading channel, we can approximate the impulse response to be simply a delta function. Hence, a *FLAT FADING, FAST FADING* channel is channel in which amplitude of the delta function varies faster than the rate of change of the transmitted baseband signal. In the case of a *FREQUENCY SELECTIVE, FAST FADING* channel, the amplitudes, phases and time delays of any one of the multipath components vary faster than the rate of change of transmitted signal. In practice, fast fading only occurs for very low data rates.

5.3.2.2 Slow Fading

In slow fading channel, the channel impulse response changes at a rate much slower than the transmitted baseband signal S (t). In this case, the channel may be assumed to be static over several reciprocal bandwidth intervals. In the frequency domain, this implies that the Doppler spread of the channel is much less than the bandwidth of the baseband signals. Therefore the signal goes undergoes slow fading if

$$\mathbf{Ts << Tc} \qquad \mathbf{and} \quad \mathbf{Bs >> Bd} \qquad (5.4)$$

It should be clear that velocity of the mobile and the baseband signaling determines whether a signal undergoes fast fading or slow fading.

5.4 Rayleigh Fading Envelope Generation

The generation of Rayleigh Fading envelopes follows from the basic fact that the envelope of a complex Gaussian process (with independent real and imaginary parts) has a Raleigh distribution. The general method to generate a Rayleigh Fading envelope is illustrated in fig (3).

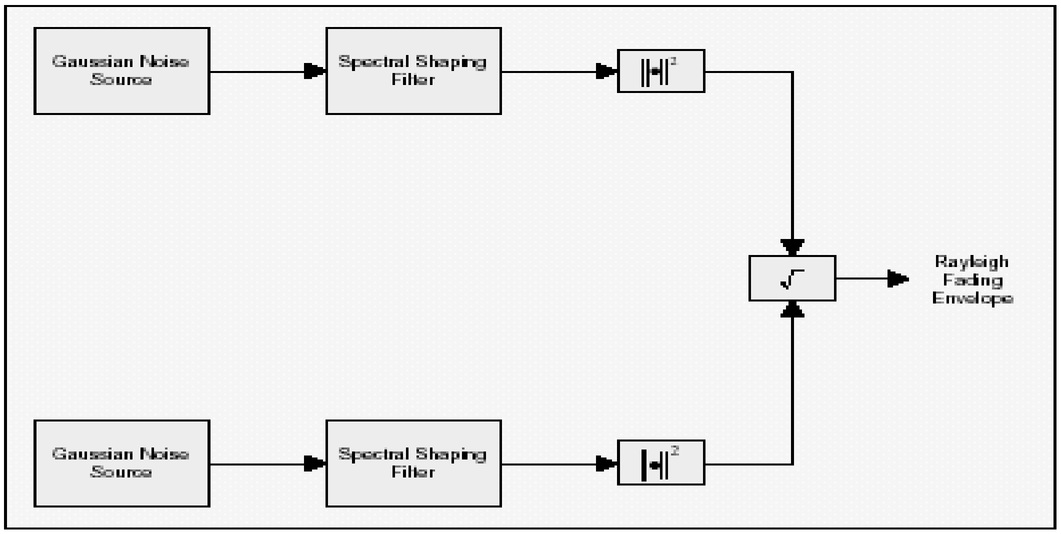


Figure 5.1 Rayleigh Fading Generation at Baseband

The Spectral Shaping filter is needed to introduce a desired amount of correlation into the Gaussian samples that produce the Rayleigh distribution. In case of Mobile Communication Systems where Rayleigh fading has to be generated for a particular speed of the mobile,the spectral shaping filter takes the form of a Doppler Filter with the maximum Doppler spread specified by the Mobile Speed (Clarke/Gans Model).

If Ig(n) and Qg(n) represent the in-phase and quadrature phase components of the complex Gaussian process (after spectral shaping), the Rayleigh fading envelope can be generated as,

$$\alpha(n) = \sqrt{I_g(n)^2 + Q_g(n)^2} \tag{5.5}$$

5.4.1 Spectral Shaping Filter

The Spectral Shaping filter is usually specified in terms of its Autocorrelation function or Power Spectral density. When a Power Spectral Density is specified, the colored Gaussian samples can be generated by passing the white Gaussian Noise samples through a filter whose transfer function H(f) can be obtained by solving,

$$H(f)H^*(f) = S_{xx}(f) \tag{5.6}$$

Where, Sxx(f) is the power spectral density of the filter. The digital implementation of H (f) can be done either using FFT Techniques or FIR/IIR filtering depending on the situation and form of H (f) obtained.

In the given problem, the PSD can be obtained taking the Fourier transform of the specified autocorrelation as,

$$R_{xx}(\tau) = e^{-1000\tau} \Rightarrow S_{xx}(f) = \frac{2000}{1000^2 + (2\pi f)^2} \tag{5.7}$$

Further, the H(f) can be obtained from the power spectral density as,

$$H(f) = \frac{\sqrt{2000}}{1000 + j2\pi f} \tag{5.8}$$

The digital implementation of the above transfer function can be done in a plethora of ways. But the best (and the most relevant) method is the IIR implementation of the above Filter using AR models. Even then, a choice has to be made between IIR filter synthesizing techniques such bilinear, impulse-invariance, backward/forward di_erence methods etc.

Since the filter has a simple one-pole type transfer functions, it is much better to use impulse invariance rather then other techniques. Proceeding further, the impulse response of the digital filter is obtained as,

$$H(s) = \frac{\sqrt{2000}}{s+1000} \Rightarrow H(z) = \frac{\sqrt{2000}}{1 - e^{-\frac{1000}{T}} z^{-1}} \tag{5.9}$$

Where T represents the sampling duration. In our case, because of the slow fading assumption, We have to generate one Rayleigh fading envelope sample per symbol (i.e., 8000 Samples per second). This works to to a sampling duration of T = 0.125ms. Thus we get,

$$H(z) = \frac{44.72}{1 - 0.8825 z^{-1}} \tag{5.10}$$

This converts to the simple differential equation,

$$y[n] = 0.8825 y[n-1] + 44.72 x[n] \tag{5.11}$$

5.4.2 Simulated Envelope.

The simulated Rayleigh fading envelope at baseband is shown in fig (5.2)

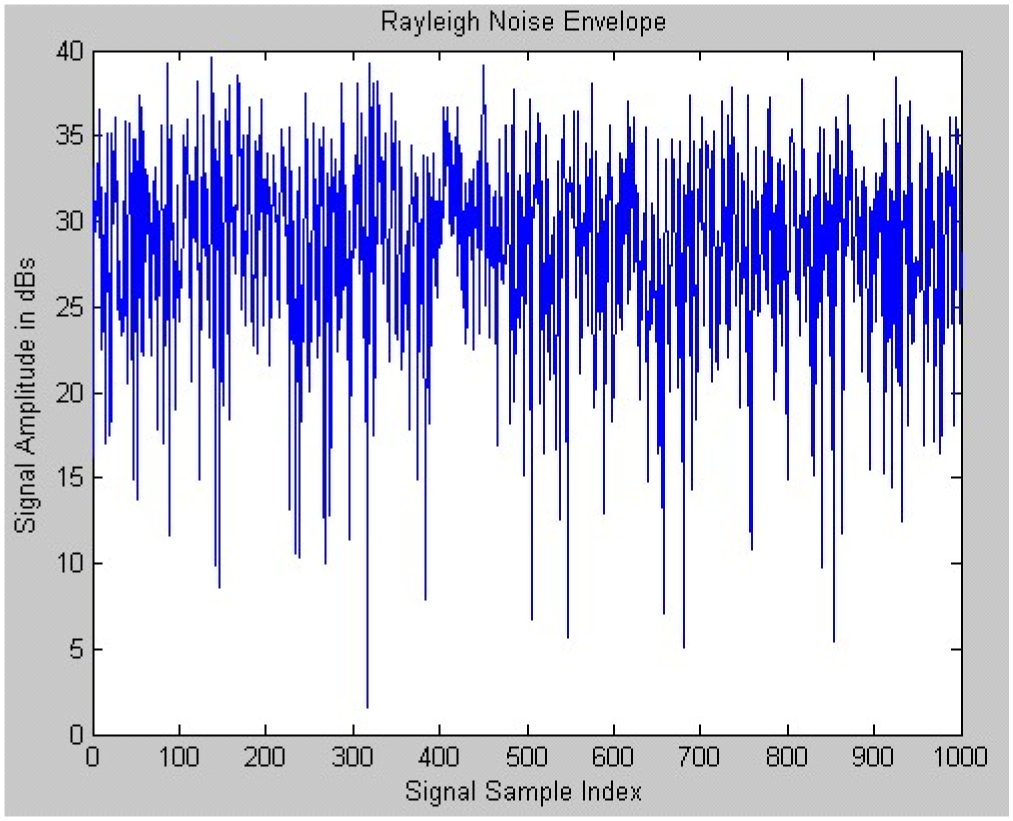


Figure 5.2 Rayleigh Noise Envelopes

5.4.3 Fade Power Adjustment.

Suppose we require a specified average fade power P, Given the generated fading samples $\{\alpha_k\}$, we can generate the fading samples with the given average fade power P using the transformation,

$$\tilde{\alpha}_k = \frac{\sqrt{P}}{\sqrt{E[\alpha^2]}} \times \alpha_k \tag{5.11}$$

5.4.4 Passing a Signal through Rayleigh Fading

Let us say we have a Simple Sine wave, The Figure is given below

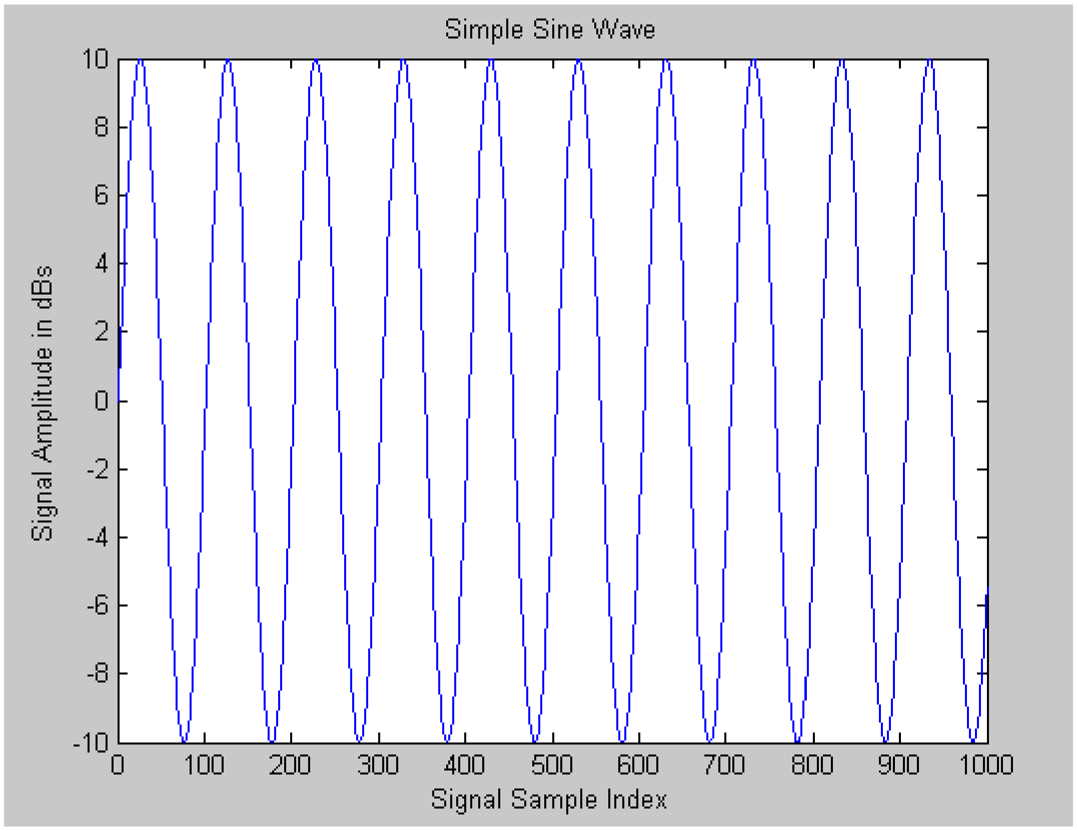


Figure 5.3 Simple Sine wave

After passing this Sine wave, when it goes under some Rayleigh Fading, the result is shown in the figure

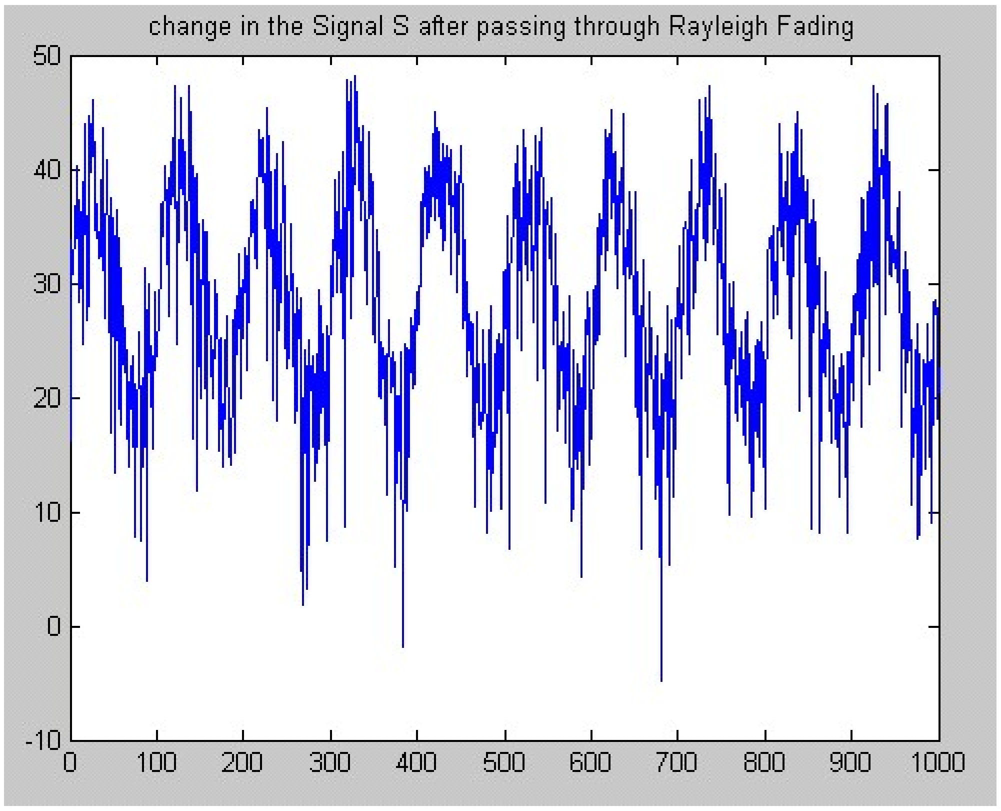


Figure 5.4 Sine wave after passing through Rayleigh Fading

CHAPTER 6

ADAPTIVE FILTERS

6.1 ADAPTIVE FILTERS

The design of a Weiner filter requires a priori information about the statistics of the data to be processed. The filter is optimum only when the statistical characteristics of the input data match the priori information on which the design of the filter is based. When this information is not known completely, however, it may not possible to design the wiener filter or else the design may no longer be optimum. A straightforward approach that we may use in such situation is the "estimate and plug" procedure. This is a two-stage process whereby the filters first "estimate" the statistical parameters of the relevant signals and then "plug" the results so obtained into a *nonrecursive* formula for computing the filter parameters. For real-time operation, this procedure has the disadvantage of requiring excessively elaborated & costly hardware. To mitigate this limitation, we may use *adaptive filter*.

6.4 WIENER FILTER THEORY

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

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

$$c_{-K}, c_{-K+1}, c_{-K+2}, \ldots, c_K, \tag{6.1}$$

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

$$\hat{s}(n) = \sum_{k=-K}^{K} c_k r(n-k) \tag{6.2}$$

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

$$\varepsilon(n) = s(n) - \hat{s}(n) \tag{6.3}$$

where $s(n)$ is the transmitted symbol and $\hat{s}(n)$ is the its estimate(see figure 6.1). Note that after equalization, there will still be residual ISI. On top of that, there is an additive Gaussian noise term. The MMSE equalizer minimizes the combined residual ISI plus noise power. The non-casuality in the mathematical description of the MMSE equalizer translates into a decision delay in the actual implementation (see figure 6.1). The delay is due to the filtering with channel filter and equalizer filter.

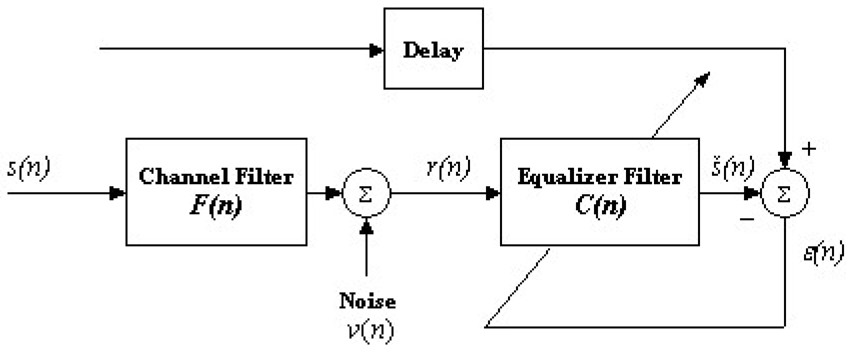


Figure 6.1: The Weiner filter(*C(n)*) configuration for equalization

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

$$\mathbf{R}(n) = \begin{bmatrix} r(n+K) \\ r(n+K-1) \\ \vdots \\ r(n-K) \end{bmatrix} \tag{6.4}$$

be the received vector at time *n* and

$$\mathbf{C} = [c_{-K}, c_{-K+1}, c_{-K+2}, \ldots, c_K] \tag{6.5}$$

be a general transversal (i.e. FIR) equalizer. By substituting for the matrix notation into equation 6.2, it is possible to represent the estimated error signal by equation 6.6 below. The equalizer output at time *n* is thus

$$\hat{s}(n) = \mathbf{CR}(n) \tag{6.6}$$

and instantaneous *squared error* of the signal can be found by squaring equation 6.6 such that it can be represented as the following equation:

$$\begin{aligned}\varepsilon(n)^2 &= \left(s(n)-\hat{s}(n)\right)^2 = \left(s(n)-\mathbf{CR}(n)\right)^2\\ &= \left(s(n)-\mathbf{CR}(n)\right)\left(s(n)-\mathbf{C}^T\mathbf{R}(n)^T\right)\\ &= s(n)^2 - s(n)\mathbf{CR}(n) - s(n)\mathbf{C}^T\mathbf{R}(n)^T + \mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T\\ &= \sigma^2 - s(n)\mathbf{CR}(n) - s(n)\mathbf{C}^T\mathbf{R}(n)^T + \mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T\end{aligned} \tag{6.7}$$

where σ^2 is the variance of $s(n)$ because $s(n)$ has a zero mean, i.e., $\sigma^2 = E(x^2) - \left(E(x)\right)^2$. Also assuming $s(n)$ a bipolar signal (BPSK or QPSK) with values ±1 then we can say σ^2=1. *Mean square error* (MSE), γ, is defined by the "expectation" of the squared error, from equation 6.7. Hence the MSE can be represented by equation 6.8.

$$\begin{aligned}\gamma = E\left[\varepsilon(n)^2\right] &= 1 - E\left[s(n)\mathbf{CR}(n)\right] - E\left[s(n)\mathbf{C}^T\mathbf{R}(n)^T\right] + E\left[\mathbf{CR}(n)\mathbf{R}(n)^T\mathbf{C}^T\right]\\ &= 1 - \mathbf{C}E\left[s(n)\mathbf{R}(n)\right] - E\left[s(n)\mathbf{R}(n)^T\right]\mathbf{C}^T + \mathbf{C}E\left[\mathbf{R}(n)\mathbf{R}(n)^T\right]\mathbf{C}^T\\ &= 1 - \mathbf{Cu}_{Rs} - \mathbf{u}_{sR}\mathbf{C}^T + \mathbf{CU}_{RR}\mathbf{C}^T\\ &= 1 - 2\mathbf{u}_{ss}\mathbf{C}^T + \mathbf{CU}_{RR}\mathbf{C}^T\end{aligned} \tag{6.8}$$

where

$$\mathbf{u}_{sR} = E\left[s(n)\mathbf{R}(n)^T\right] = \mathbf{u}_{Rs}{}^T \tag{6.9}$$

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

It is clear from this expression that the mean square error γ is a *quadratic function* of the weight vector $\mathbf{C}$ (filter coefficients). That is, when Equation 6.8 is expanded, the elements of $\mathbf{C}$ will appear in the first and second order only. This is valid when the input components and desired response inputs are wide-sense stationary stochastic (random) variables .

6.5 PERFORMANCE SURFACE

A portion of a typical two-dimensional *MSE* function is illustrated in figure 6.2. The vertical axis represents the mean square error and the two horizontal axes represent the values of two filter coefficients. The quadratic error function, or *performance surface*, can be used to determine the optimum weight vector $\mathbf{C}_{opt}$ (or Wiener filter coefficients). With a quadratic performance function there is only one global optimum; no local minima exist. The shape of the function would be hyper-parabolic if there were more than two weights.

Many adaptive processes that cause the weight vector to search for the minimum of the performance surface do so by the *gradient method* [11]. The gradient of the mean square error of the performance surface, designated ∇, can be obtained by differentiating equation 6.8 with respect to each component of the weight vector.

$$\nabla = \frac{\partial \gamma}{\partial \mathbf{C}} = \left[\frac{\partial \gamma}{\partial c_{-K}}, \frac{\partial \gamma}{\partial c_{-K+1}}, \ldots, \frac{\partial \gamma}{\partial c_{K}} \right] \tag{6.12}$$

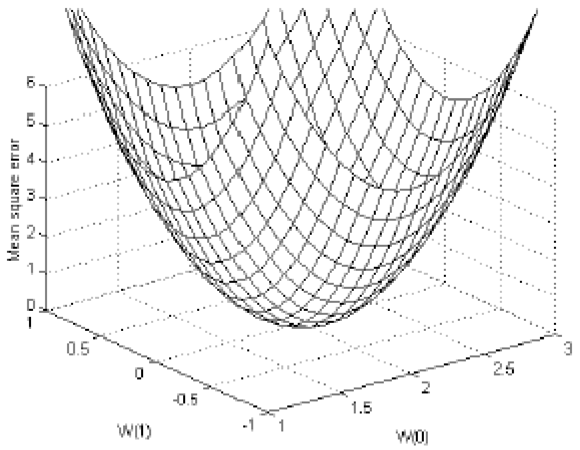


Figure 6.2: A two dimensional error performance surface

$$\begin{aligned} \nabla &= E\left[\frac{\partial \varepsilon(n)^2}{\partial \mathbf{C}} \right] = E\left[2\varepsilon(n) \frac{\partial \varepsilon(n)}{\partial \mathbf{C}} \right] \\ &= 2E\left[\left(s(n) - \mathbf{C}\mathbf{R}(n) \right) \frac{\partial}{\partial \mathbf{C}} \left(s(n) - \mathbf{C}\mathbf{R}(n) \right) \right] \\ &= -2E\left[\left(s(n) - \mathbf{C}\mathbf{R}(n) \right) \mathbf{R}(n)^T \right] \\ &= -2E\left[s(n)\mathbf{R}(n)^T \right] + 2E\left[\mathbf{R}(n)\mathbf{R}(n)^T \right]\mathbf{C} \\ &= -2\mathbf{u}_{sR} + 2\mathbf{U}_{RR}\mathbf{C} \end{aligned} \tag{6.13}$$

The gradient vector is the tangent of the *N*-dimensional surface γ at the point $\mathbf{C}_{opt}$ When the gradient vector is zero, the surface γ reaches its lowest value and hence the mean square error will be minimized. The point in the *N*-dimensional space where this occurs is. Therefore, setting equation 6.13 to zero we get

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

$$\mathbf{C}_{opt} = \mathbf{u}_{sR}\mathbf{U}_{RR}^{-1} \tag{6.14}$$

and corresponding MSE is

$$\begin{aligned} \gamma_{\min} &= 1 - 2\mathbf{u}_{sR}\mathbf{C}_{opt}^{T} + \mathbf{C}_{opt}\mathbf{U}_{RR}\mathbf{C}_{opt}^{T} \\ &= 1 - \mathbf{u}_{sR}\mathbf{U}_{RR}^{-1}\mathbf{u}_{sR}^{T} \end{aligned} \tag{6.15}$$

This equation is known as the *Wiener-Hopf* equation in matrix form, and the filter given by $\mathbf{C}_{opt}$ in equation 6.14 is the Wiener filter. However, in practice it is not usual to evaluate. In addition, $\mathbf{C}_{opt}$ has to be calculated repeatedly for non-stationary signals and this can be computationally intensive because it requires matrix inversions. But another question is how to determine the values of $\mathbf{u}_{sR}$ and $\mathbf{U}_{RR}$ in case of channel equalization.

6.6 STEEPEST DESCENT ALGORITHM

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

$$\nabla(n) = -2\left[\mathbf{u}_{sr} - \mathbf{C}(n)\mathbf{U}_{RR}\right] \tag{6.25}$$

The gradient vector of the error surface γ at $\mathbf{C}(n)$; see the equation 6.13 Based $\mathbf{C}(n)$ and $\nabla(n)$, we obtain the next estimate of $\mathbf{C}_{opt}$ according to

$$\begin{aligned} \mathbf{C}(n+1) &= \mathbf{C}(n) - \frac{\mu}{2}\nabla(n) \\ &= \mathbf{C}(n) + \mu[\mathbf{u}_{sr} - \mathbf{C}(n)\mathbf{U}_{RR}] \\ &= \mathbf{C}(n)[\mathbf{I} - \mu\mathbf{U}_{RR}] + \mu\mathbf{u}_{sr} \end{aligned} \qquad (6.26)$$

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

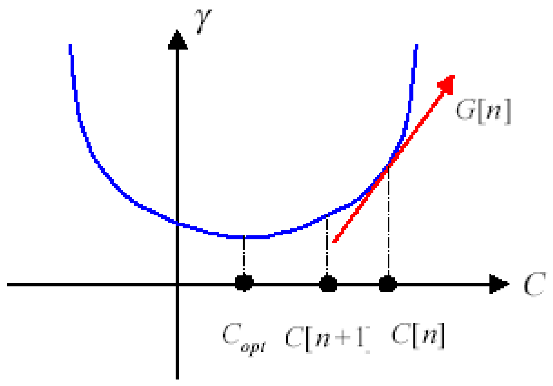


Figure 6.3: Operation of steepest descent algorithm.

6.7 THE LMS ALGORITHM

While the steepest descent method is able to determine the optimal equalizer coefficients without performing any matrix inversion, its operation is still based on the assumption that the channel parameters the correlation vector $\mathbf{u}_{sR}$ and the covariance matrix $\mathbf{U}_{RR}$ known to the receiver. Recall that the receiver uses these parameters to compute the gradient vector $\nabla(n)$ required for updating the the equalizer coefficients. In the LMS algorithm, the gradient vector is replaced by its estimate. Let us consider the correlation of the received vector $\mathbf{R}(n)$ (see equation 6.4) with the equalization error

$$\varepsilon(n) = s(n) - \hat{s}(n) = s(n) - \mathbf{CR}(n) \qquad (6.47)$$

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

$$\begin{aligned}\mathbf{w}_{\varepsilon R} &= E\left[\varepsilon(n)\mathbf{R}(n)^T\right] \\ &= E\left[\varepsilon(n)\mathbf{R}(n)^T - \mathbf{C}\mathbf{R}(n)\mathbf{R}(n)^T\right] \\ &= E\left[\varepsilon(n)\mathbf{R}(n)^T\right] - E\left[\mathbf{C}\mathbf{R}(n)\mathbf{R}(n)^T\right] \\ &= \mathbf{u}_{sR} - \mathbf{C}\mathbf{U}_{RR}\end{aligned} \tag{6.48}$$

Now, if we evaluate this correlation vector at **C** = **C**(n) , then we obtain

$$\mathbf{w}_{\varepsilon R}\Big|_{\mathbf{C}=\mathbf{C}(n)} = \mathbf{u}_{sR} - \mathbf{C}(n)\mathbf{U}_{RR} = -\frac{1}{2}\nabla(n) \tag{6.49}$$

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

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n) - \frac{\mu}{2}\nabla(n) \\ &= \mathbf{C}(n) + E\left[\varepsilon(n)\mathbf{R}(n)^T\right]\end{aligned} \tag{6.50}$$

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

$$\begin{aligned}\mathbf{C}(n+1) &= \mathbf{C}(n) + E\left[\varepsilon(n)\mathbf{R}(n)^T\right] \\ &= \mathbf{C}(n) + \left[s(n) - \hat{s}(n)\right]\mathbf{R}(n)^T\end{aligned} \tag{6.51}$$

In other word, the LMS algorithm uses a noisy estimate of the true gradient in updating the filter coefficients.

CHAPTER 7

CHANNEL EQUALIZATION

7.1 INTRODUCTION

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

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

These disruptive effects are due to the dispersive transmission medium (e.g. telephone cables) and the multipath effects in the radio channel. A typical communication system is depicted in figure 7.1 where the equalizer is incorporated within the receiver while the channel introduces intersymbol interference. The transfer function of the equalizer is an estimate of the direct inverse of the channel transfer function. To transmit high speed data over a bandlimited channel, the frequency response of the channel is usually not known with sufficient precision to design an optimum match filter. The equalizer is, therefore, designed to be adaptive to the channel variation. The configuration of an adaptive linear equalizer is depicted in figure 7.2. Based on the observed channel output, an adaptive algorithm recursively updates the equalizer to reconstruct the output signal.

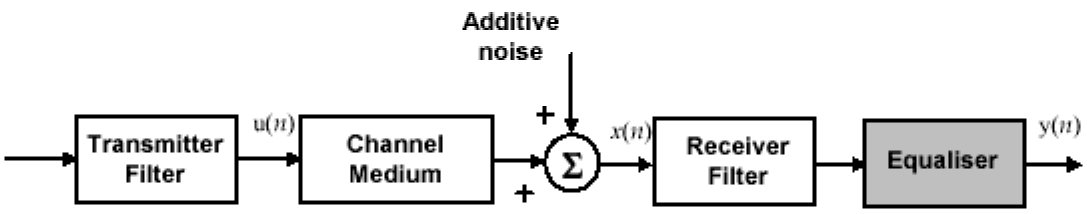


Figure 7.1: A typical communication system.

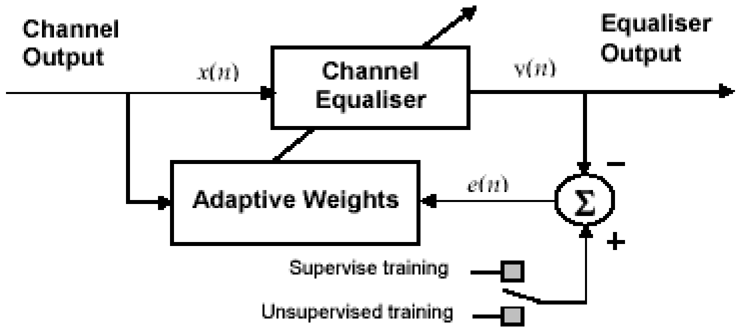


Figure 7.2: A simple linear channel equalizer configuration.

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

7.2 ISI DUE TO MULTIPATH EFFECTS

In a wireless radio channel, the ISI is caused by the multipath effects when the multipath spread as explained previously is greater than the symbol interval. Multipath effects describe the situation in which there are several propagation paths from transmitter to receiver. Most commonly, this results when there are reflected signals detected at the receiver following the direct path. The multipath phenomenon can be modeled by an FIR system (see chapter 3). The center tap represents the direct path, while the succeeding tap weights represent the amplitudes, delays, and phases of the reflected paths. For simple examples, see the two cases described in Figure 7.3 and Figure 7.4.

Figure 7.3(a) shows the time response of an ideal transmission path, which is a δ function. Such a channel exerts no spectral distortion or delayed signals. Figure 7.3(b) shows the spectral response of such a system. Note that the frequency magnitude response is perfectly flat, as indicated by the solid horizontal line.

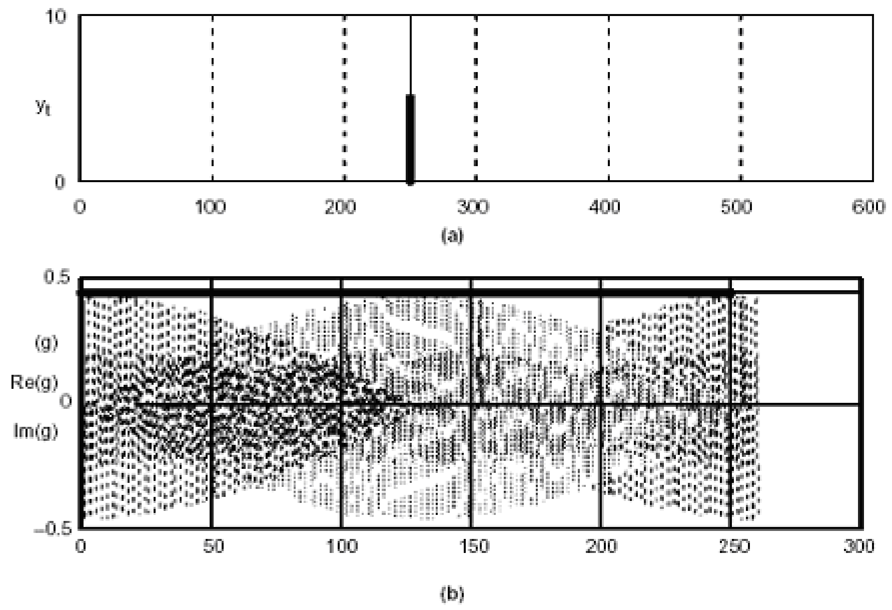


Figure 7.3: (a) Impulse response and (b) frequency response of ideal single path channel

Figure 7.4(a) shows the time response of a system that contains a single multipath channel [8]. The first nonzero sample of the response represents the direct path, while the second represents a delayed path to the receiver. In this instance, the pulses are identical in amplitude and phase and are separated by ten sample intervals.

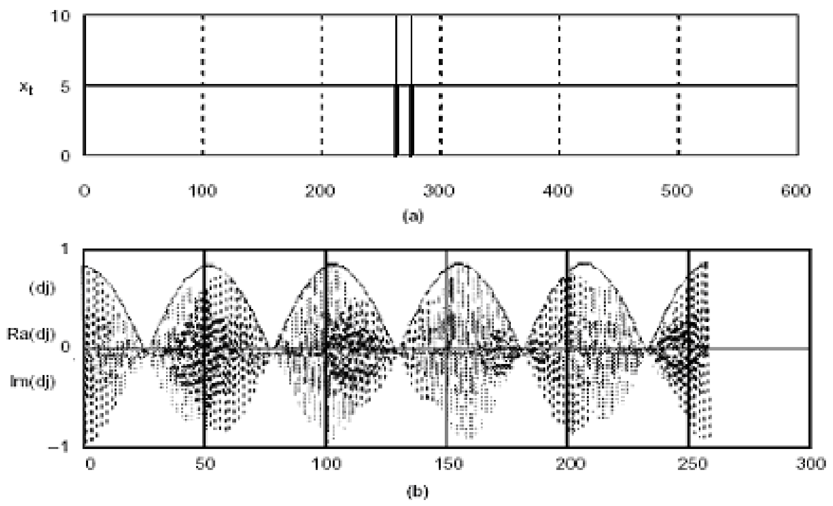


Figure 7.4: (a) Time response and (b) frequency response of two path wireless channel

Notice in Figure 7.4(b) that the magnitude response exhibits $t_0/2$ nulls, where t_0 represents the sample delay. Even though you are effectively adding two identical flat spectra (as shown in Figure 7.3(b)) the time delay results in a phase delay in the spectral domain. This phase delay results in nulls where the two signals are of equal amplitude but opposite phase. Obviously, multipath effects can have major effects on the system spectral response, thereby providing another justification for channel equalization

Now it is time to represent it mathematically what we all said so far. Suppose that a channel model (see figure 7.5) is used to describe the distortion effect and it is given as a sum of weighted time delayed discrete-time channel impulse responses, $H(z)$:

$$H(z) = \sum_i h_i z^{-i} = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots \tag{7.1}$$

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

$$y(k) = \sum_{i=0}^{L-1} h_i u(k-i) + n(k) \tag{7.2}$$

Where $y\ (k)$ is the input to the equalizer (received signal after passing through channel) which is simply the convolution of h_i and $u\ (k)$. $u(k)$ is the transmitted sequence, h_i is the channel impulse response, $n(k)$ represents additive white Gaussian noise (AWGN) added to the channel and L represents the length of the channel impulse response. Equation (7.2) shows that the transmitted symbol $u\ (k)$ is affected by the weighted delay symbols of $u(k\text{-}i)$, thus causing intersymbol interference.

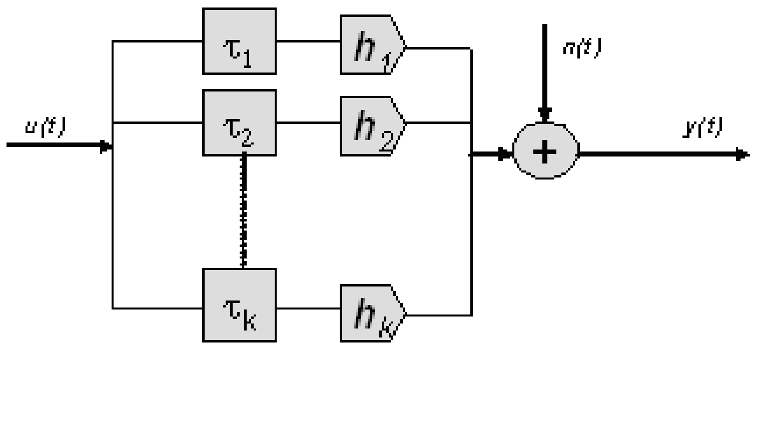


Figure 7.5: A multipath wireless channel model.

7.3 COMMUNICATION SYSTEM MODEL WITH EQUALIZER

7.3.1 Continuous-time Model

For our communication system which is employing a linear modulation, QPSK, through a dispersive channel, the whole system can be described the conceptual model in Figure 7.7, 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.

$$\sum_{n=-\infty}^{\infty} X(f - \frac{n}{T}) = T \tag{7.3}$$

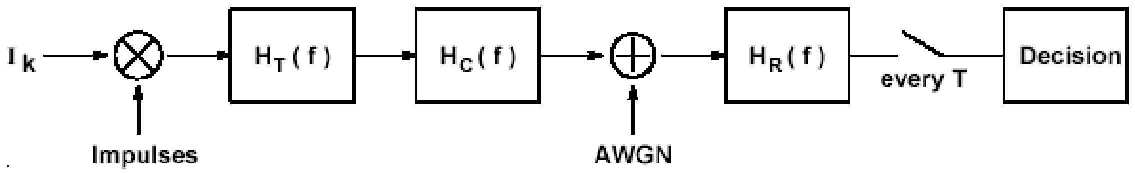


Figure 7.7: Continuous-time communication model over a multipath dispersive channel

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 (7.5) is satisfied. This is the Nyquist pulse design method described in previous chapter. The major disadvantage of this pulse shaping method is that it is in general difficult to construct the appropriate analog filters for $H_T(f)$ and $H_R(f)$ in practice. Moreover, we have to know the channel response $H_C(f)$ in advance to construct the transmission and receiving filters.

An alternative method is to fix the transmission filter4 and choose the receiving filter $H_R(f)$ to satisfy the condition in **(7.3)**. 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 7.7. This approach to remove ISI is usually known as *equalization*. The main advantage of this approach is that a digital filter is easy to build and is easy to alter for different equalization schemes, as well as to fit different channel conditions.

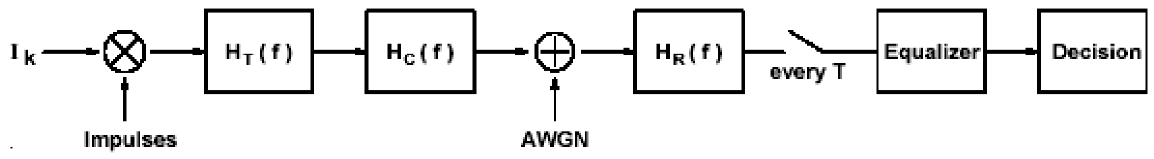


Figure 7.7: Communication system with equalizer

7.3.2 Equivalent discrete-time model

Our goal is to design the equalizer which can remove (or suppress) ISI. To do so, we translate the continuous-time communication system model in Figure 7.7 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 7.8.
- We input a bit or a symbol to the communication system every T seconds, and get back a sample at the output of the sampler every T seconds. Therefore, we can represent the communication system in Figure 7.8 from the information source to the sampler as a digital filter.

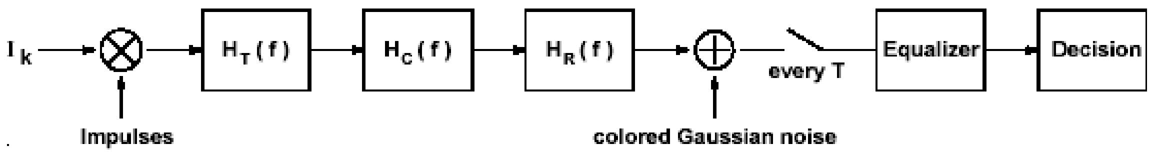


Figure 7.8: Equivalent communication system with colored Gaussian noise

Since $H_T(f), H_C(f)$ and $H_R(f)$ are LTI filters, they can be combined and represented by an equivalent digital LTI filter. Denote its transfer function by *H(z)* and its impulse response by $\{h_k\}_{k=-\infty}^{\infty}$. The result is the *discrete time-linear filter model* shown in Figure 7.9, in which the output sequence $\{I'_k\}$ is given by

$$\begin{aligned} I'_k &= \sum_j I_k h_{k-j} + n_k \\ &= I_k h_0 + \sum_{j \neq k} I_k h_{k-j} + n_k \end{aligned} \tag{7.4}$$

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.

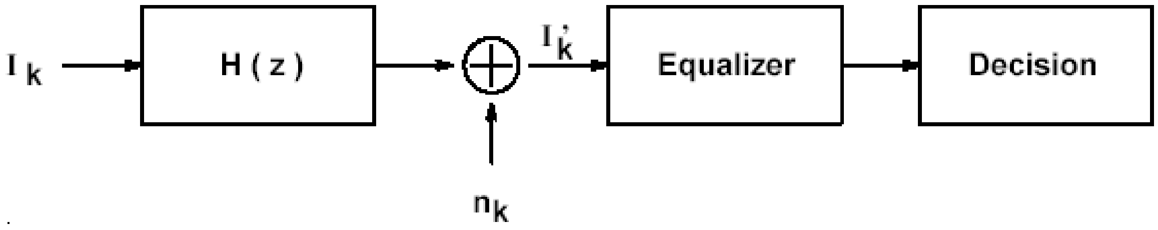


Figure 7.9: Equivalent discrete-time communication system model with colored noise

- Usually, the equalizer consists of two parts, namely, a *noise-whitening* digital filter $H_W(z)$ and an equalizing circuit that equalizes the noise-whitened output as shown in Figure 7.9. The effect of $H_W(z)$ is to "whiten" the noise sequence so that the noise samples are uncorrelated. Notice that $H_W(z)$ depends only on $H_R(f)$, and can be determined a prior according to our choice of $H_R(f)$. At the output of $H_W(z)$, the noise sequence is white. Therefore, equivalently, we can consider the equivalent discrete-time model shown in Figure 7.11, in which $\{\bar{n}_k\}$ is an AWGN sequence.

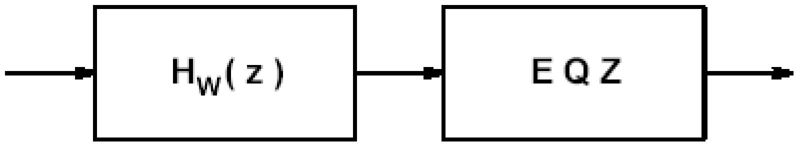


Figure 7.10: Typical equalizer

- Let $G(z) = H(z)H_W(z)$. The communication system from the information source to the output of the noise whitening filter can now be represented by the *discrete-time white-noise linear filter model* in Figure 7.12. The output sequence $\{\tilde{I}_k\}$ is given by:

$$\begin{aligned}\tilde{I}_k &= \sum_j I_k g_{k-j} + \bar{n}_k \\ &= I_k g_0 + \sum_{j \neq k} I_k g_{k-j} + \bar{n}_k\end{aligned} \tag{7.5}$$

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

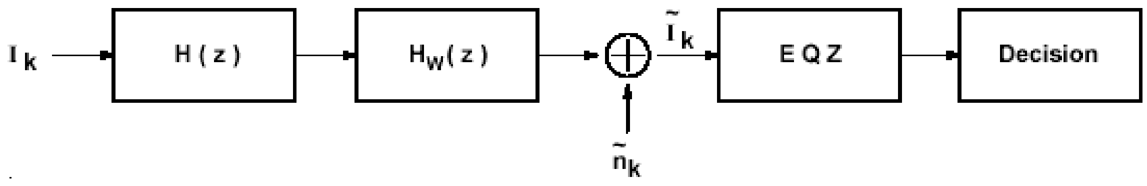


Figure 7.11: Equivalent discrete-time communication system model with white noise

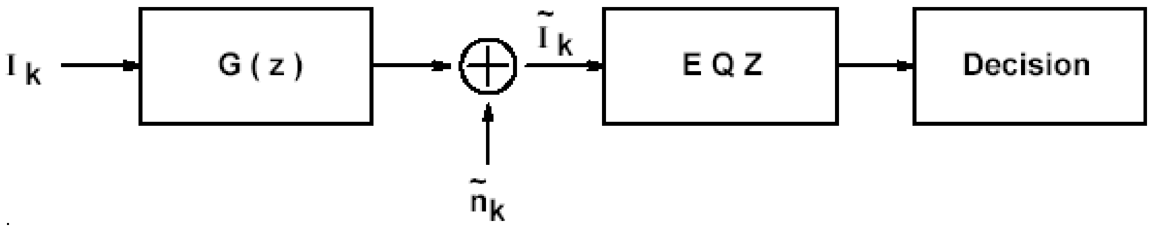


Figure 7.12: Equivalent discrete-time white-noise linear filter model

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

$$\hat{I}_k = \sum_j \tilde{I}_{k-j} h_{Ej} \tag{7.6}$$

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

7.4 CLASSIFICATION OF EQUALIZERS

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

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

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

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

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

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

In figure 7.13 types of equalizers, their structures and the algorithm for learning channel environment is given.

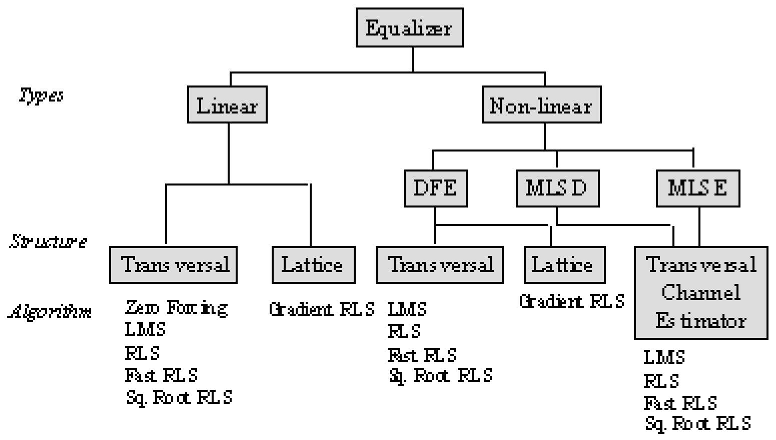


Figure 7.13: Classification of equalizers

Non-linear equalization is important in providing optimum performance for ill-conditioned channels that non-linear techniques require more computation and controls. However, in order the study the gradient descent-based adaptive algorithms' performance, the linear equalizer is more appropriate. Table 7.1 also gives some performance, computation complexity and implementation cost measures of the equalizers.

Table 7.1: Cost, performance and complexity analysis of equalizers

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

7.5 MMSE EQUALIZER

The zero-forcing equalizer, although removes ISI, may not give the best error performance for the communication system because it does not take into account noises in the system. A different equalizer that takes noises into account is the minimum mean square error (MMSE) equalizer. It is based on the mean square error (MSE) criterion. Before going into the mathematics of MMSE we want to clear that the detail derivation for minimizing the MSE and reaching the Weiner solution can be found from the previous chapter on adaptive filters. Here only the necessary steps are repeated.

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

$$MSE = E[e_k{}^2] = E[(I_k - \hat{I}_k)^2] \tag{7.7}$$

Let us employ the FIR filter of order *2L+1* shown in Figure 7.14 as the equalizer. We note that a delay of *L* symbols is incurred at the output of the FIR filter. Then

$$\begin{aligned} MSE &= E[(I_k - \sum_{j=-L}^{L} \tilde{I}_{k-j} h_{E,j})^2] \\ &= E[(I_k - \hat{\mathbf{I}}_k{}^T \mathbf{h}_E)^2] \end{aligned} \tag{7.8}$$

where

$$\hat{\mathbf{I}}_k = [\hat{I}_{k+L}, \ldots, \hat{I}_{k-L}]^T \tag{7.9}$$

$$\mathbf{h}_E = [h_{E,-L}, \ldots, h_{E,L}]^T \tag{7.10}$$

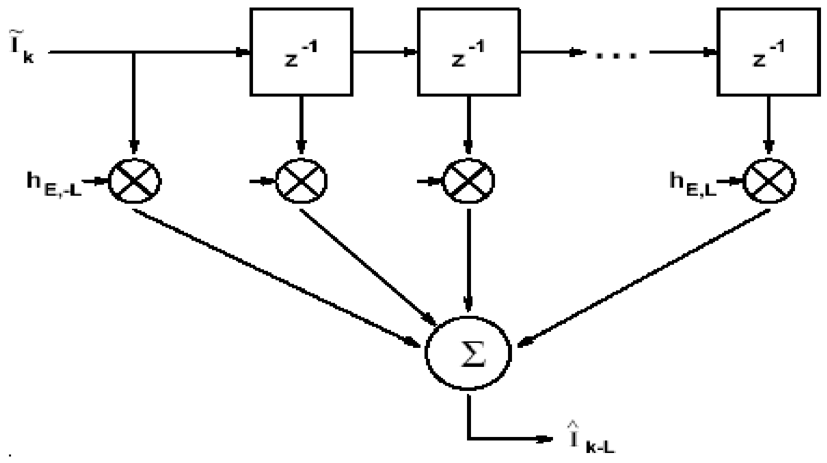


Figure 7.14: FIR Filter as a MMSE Equalizer

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

$$E[\tilde{\mathbf{I}}_k(I_k-\tilde{\mathbf{I}}_k{}^T\mathbf{h}_E)]=0 \tag{7.11}$$

Rearranging we get

$$\mathbf{R}\mathbf{h}_E=\mathbf{d} \tag{7.12}$$

Where

$$\mathbf{R}=E[\tilde{\mathbf{I}}_k\,\tilde{\mathbf{I}}_k{}^T] \tag{7.13}$$

$$\mathbf{d}=E[\tilde{I}_k\,\tilde{\mathbf{I}}_k] \tag{7.14}$$

If $\mathbf{R}$ and $\mathbf{d}$ are available, then the MMSE equalizer can be found by solving the linear matrix equation 7.12. It can be shown that the signal-to-noise ratio at the output of the MMSE equalizer is better than that of the zero-forcing equalizer.

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

$$\mathbf{h}_E(k)=\mathbf{h}_E(k-1)+\mu(\mathbf{d}-\mathbf{R}\mathbf{h}_E(k-1)) \tag{7.15}$$

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

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

$$\mathbf{h}_E(k)=\mathbf{h}_E(k-1)+\mu[I_k-\tilde{\mathbf{I}}_k{}^T\mathbf{h}_E(k-1)]\tilde{\mathbf{I}}_k \tag{7.16}$$

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

The beauty of the approach is that the only parameter to be adjusted is the adaptation step size μ. Through an iterative process, explained above, all filter tap weights are adjusted during each sample period in the training sequence. Eventually, the filter will reach a configuration that minimizes the mean square error between the equalized signal and the stored reference. As might be expected, the choice of μ involves a tradeoff between rapid convergence and residual steady-state error. A too-large setting for μ can result in a system that converges rapidly on start-up, but then chops around the optimal coefficient settings at steady state.

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