AN INVESTIGATION INTO THE EFFECTS OF USING LIMITED PRECISION INTEGER ARITHMETIC IN DIGITAL MODEMS

By

R.G. Hale, BSc

A Doctorial Thesis Submitted in Partial Fulfilment of the Requirements for the Award of Doctor of Philosophy of the Loughborough University of Technology.

June 1990

Supervisor: Dr. S.C. Bateman
Department of Electronic and Electrical Engineering

CONTENTS

ABSTRACT

The main aim of this thesis is to study the effects of using a reduced level of arithmetical precision (as found in a 16-bit microprocessor) whilst running various algorithms in the detection stages of a digital modem. The reason for using a lower precision is to see if these algorithms will run on a limited precision device, such as a Texas Instruments TMS320C25 digital signal processor, in real time.

In the study, data is transmitted over voice-band channels, such as telephone circuits and H.F. radio links, where the main imparements are intersymbol interference and additive noise. The characteristics of these channels is briefly studied.

The thesis includes a study of quadrature amplitude modulated (QAM) signals transmitted over voice-band channels when the transmission path has, both, time invariant characteristics and when it introduces Rayleigh fading into the transmitted signal. Based on this study, an equivalent baseband model of the QAM system is derived, which is used in extensive computer simulation tests.

Two near maximum likelihood detectors are studied, one of which is a modified version of the pseudobinary reduced state Viterbi algorithm and is easily implemented in real time. The linear feedforward channel estimator is also studied, along with an efficient root finding algorithm that adaptively adjusts a pre-filter placed ahead of the detector such that the combined impulse response of the channel and filter is minimum phase. To show the effects of using integer arithmetic, all of these algorithms are simulated using integer variables.

Results of computer simulation tests are presented, showing the performance of the above algorithms whilst they are running independently from each other and when they are combined to form a complete modem receiver. High precision numerical values, calculated using the NAG mathematical library, are used to demonstrate the accuracy of the root finding algorithm when it is running with integer arithmetic. Results of running the root finding algorithm on a TMS320C25 development board are also given. These results are also compared with those found using the NAG library.

ACKNOWLEDGEMENTS

The author would like to thank his supervisor, Dr. S.C. Bateman, for his help and guidance.

The financial support of the Science and Engineering Research Council is gratefully acknowledged. The author would also like to thank all of his friends, especially, Binni, Tasneem, Anjli, Aamir, Farid, Andy, Shuja and Jagruti for their support.

GLOSSARY OF SYMBOLS

a(t)	Impulse response of a filter
A(f)	Frequency response of a filter
|A(f)|	Absolute value of A(f)
a(t)*b(t)	Convolution between a(t) and b(t)
E[.]	Expectation operator
g+1	Number of samples in the sampled impulse response of the linear baseband channel
Im[.]	Imaginary part of a complex number
j	When used as a subscript is an integer, otherwise it has a value equal to $\sqrt{-1}$
n(t)	White Gaussian noise with zero mean and two sided power spectral density $n_0/2$
$n_0/2$	Power spectral density of n(t)
Re[.]	Real pert of a complex number
r(t)	Received signal
$\{r_i\}$	Sequence of received samples
s_i	Data symbol
Superscript *	Complex conjugate
T	Sampling period
w(t)	Complex valued, baseband Gaussian noise with zero mean
x(t)	Hilbert transform of x(t)

$y(t)$	Impulse response of linear baseband channel
Y_i	Sampled impulse-response of linear baseband channel at time $t=iT$
Y_i'	Estimate of Y_i at time $t=iT$
$Y_{i+1,i}'$	Prediction of Y_{i+1} at time $t=iT$ obtained from Y_i'
σ^2	Variance of real or imaginary parts of $w(t)$

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND

With the increasing need to communicate digital data from one place to another, there is considerable interest in techniques that will allow the data to be transmitted at the highest convenient rates over a wide range of channels [1]. The need for increasing the rate at which data is sent is due to the high cost of using a channel, for example, if the data was transmitted over a satellite link, then, increasing the data rate will reduce the time taken to send a block of data and therefore reduce the total cost of hiring the link. For a given equipment complexity, the maximum rate at which the data can be transmitted depends upon the bandwidth of the channel and the amount of noise it introduces into the transmitted signal [2,3,5].

There are many different media over which the data stream can be transmitted, but the most important of these are the voice-band channels in the form of the telephone network and high frequency (H.F.) radio links [2].

The telephone network offers a convenient medium (due to its worldwide existence) over which digital data can be sent. A telephone channel may be made up of several different media, such as, coaxial cable, optical fibres and satellite links.

H.F. radio links are also important for data communications because they offer the most cost effective means of communicating data over long distances. Unfortunately, because of the unpredictable nature of H.F. radio links, satellite links are generally used for long distance communications, even though the latter are very expensive.

Shannon [4] has shown that for a typical voice-band channel, where the signal-to-noise ratio is approximately 30 dB, the maximum rate at which information can be sent over such a channel is roughly 20000 bits/second.

Various detection processes have been developed to detect the digital data from the received noisy and distorted signal. Their complexity depends upon the rate at which the data is transmitted, as well as the nature of the signal impairments likely to be encountered over the channel. Many promising near-maximum likelihood detectors have been proposed in recent years [6-26,29]. Their performance has been shown to be improved by the placement of a linear feedforward filter placed ahead of the detector [6,8-11,16,27-29] such that the impulse response of the channel and filter is minimum phase [30]. The characteristics of the voice-band channel are likely to be varying (from call to call in the case of a telephone channel, or, continuously for a H.F. radio link) and the filter ahead of the detector must, therefore, be adaptive in nature. A simple, cost effective algorithm has been developed by Clark and Hau [27,29], that will enable the filter to be adjusted both rapidly and accurately to the changes in the channel characteristics.

A large amount of work has been carried out, using computer simulations, to show the performance of various detectors, operating with the filter just mentioned, in the presence of intersymbol interference and additive white Gaussian noise [8-12,15,16,19-25]. The computer simulations have all been carried out using a high degree of numerical precision. Since the detectors are intended to be used in a practical modem where, it is highly likely that they will be run on a limited precision digital signal processor which will also have a limited speed [1], the computer simulations mentioned above do not give a clear idea of whether or not the detection algorithms will run and, if so, how accurately when the numerical precision used is reduced

The aim of this thesis is, therefore, to show if the various algorithms that are found in the detection stages of a digital modem will run on a digital signal processor such as the Texas Instruments TMS320C25.

1.2 OUTLINE OF THE THESIS

Chapter 2 gives a description of a model of a digital data transmission system over which quadrature amplitude modulated signals are sent. Based on this model, an

equivalent baseband model is derived, where data is transmitted over a linear baseband model of the transmission system.

Chapter 3 covers the various types of detection processes, beginning with a description of the decision feedback equaliser, which gives a simple (and easily implementable) means of detecting the data. An outline of the optimum detection process is then given, followed by a brief description of the Viterbi algorithm detector. Two near-maximum likelihood detectors, both based upon the reduced state Viterbi algorithm, are then described. The performance of the near-maximum likelihood detectors is then shown to be improved by placing a filter ahead of the them to give a minimum phase impulse response.

In chapter 4 the transmission of digital data over telephone circuits is studied. First, some of the effects of using reduced precision arithmetic are considered. Descriptions of the linear feedforward channel estimator and a root finding algorithm, that will adaptively adjust the filter ahead of the detector, are given. Results are presented showing the performance of these two algorithms when 16-bit (or less) integer arithmetic is used whilst they are running. Following this, the results of running the root finding algorithm on a TMS320C25 development board are shown. Finally, bit error rate curves showing the performance of the estimator, root finding algorithm and near maximum likelihood detector, operating together, are presented. Ways in which the speed of the various algorithms studied are also discussed.

Chapter 5 describes some of the properties of the H.F. radio link. A model of the H.F. radio link is presented that is suitable for use in computer simulations.

Chapter 6 gives a description of the transmission of a QAM signal over an H.F. radio link and presents a model of an equivalent baseband model which can be used in computer simulation tests. Results of running the algorithms considered in chapter 4 with reduced precision arithmetic and a time varying channel are then presented in the form of bit error rate curves.

Chapter 7 gives some conclusions and some suggestions for further work.

Chapter 2
Model of the Digital Data Transmission System

2.1 Introduction

In this chapter a model of the digital data transmission system is presented, where a binary data stream is transmitted over a voice-band channel. Before the data stream is transmitted, it is first converted into a form so that it can pass over the transmission path with as little distortion and loss of signal power as possible [2,31]. Since the data stream is a baseband signal, i.e. one whose frequency spectrum is centred around zero Hertz, it must be used to modulate a carrier waveform to produce a signal whose frequency spectrum lies in the centre of the passband of the voice-band channel. The most suitable type of modulation scheme for voice-band channels is suppressed carrier quadrature amplitude modulation (QAM). QAM has several advantages over other types of modulation techniques, for example, it makes efficient use of available bandwidth and the actual modulation and demodulation processes are linear, which greatly simplifies the detector [2,3,5,32,33].

In the following section, the band-pass model of the data transmission system, which is shown in figure 2.1, is examined and it is shown that this model can be represented by an equivalent baseband model [5,33,34]. Because of its simplicity, the baseband model is easily simulated on a computer.

2.2 Analysis

In figure 2.1, the digital data stream is first encoded to produce two streams of bipolar impulses, $\Sigma S_{i,0}\delta(t-iT)$ and $\Sigma S_{i,1}\delta(t-iT)$, where 1/T is the rate at which the impulses are generated. For an m-level QAM signal $s_{i,0}$ and $s_{i,1}$ have one of the possible values given by

$$s_{i,0}, s_{i,1} = 2j - \sqrt{m} + 1 \qquad j = 0,1, \ldots, (\sqrt{m} - 1) \qquad ..2.1$$

For example, with a 16-level QAM signal, $s_{i,0}$, $s_{i,1} = \pm 1$ or ± 3. The symbols $s_{i,0}$ and $s_{i,1}$, together, may thus have one of m possible combinations. The two streams of impulses are fed separately into two low-pass filters (filters C_1 and C_2), both of which have the real valued impulse response c(t). The outputs of these two filters are modulated (multiplied) by two carriers that are in phase quadrature, but with the same frequency f_c. The outputs of the two modulators are added together to form a QAM signal, which is fed into the band-pass filter D. Filter D has a real valued impulse response, d(t), and it prevents any unwanted signals that lie outside the band of frequencies, occupied by the QAM signal, entering the transmission path. The carrier frequency is chosen such that the amplitude spectrum of x(t) which is fed into the transmission path, |X(f)|, will fit within the frequency characteristics of the transmission path, whose real valued impulse response is h(t). Figure 2.2a shows the amplitude response of filter C, and figure 2.2b shows the amplitude spectrum of x(t). In figure 2.2b it can be seen that for |X(f)| to have a bandpass shape, the carrier frequency, f_c, must not be less than 1/2T, where |C(f)| is assumed to be bandlimited to -1/2T to 1/2T Hz, i.e. the system is assumed to be operating at the Nyquist rate [2]. A real valued noise signal, n(t), is added to the signal before it passes through filter E. The noise signal is assumed to be a white Gaussian signal, with zero mean and a two sided power spectral density $n_0/2$. At the receiver, filter E, which has a real valued impulse response e(t), removes any spectral components of n(t) that lie outside the frequency band of the QAM signal, without however, causing excessive distortion. Figure 2.2c shows the amplitude response, |E(f)|, of filter E. The output of filter E is coherently demodulated by multiplying it by two carriers in phase quadrature which have same frequency f_c. The two low-pass filters with real valued impulse responses f(t), remove any high frequency components generated by the demodulation process to give two baseband signals, which are fed into the detector. Chapter 3 describes the detection process.

From figure 2.1, it can be seen that x(t) is given by,

$$x(t)=\sqrt{2}\sum_i s_{i,0}c(t-iT)\cos 2\pi f_c t-\sqrt{2}\sum_i s_{i,1}c(t-iT)\sin 2\pi f_c t \qquad ..2.2$$

Now, if we let

$$s_i=s_{i,0}+js_{i,1} \qquad ..2.3$$

then equation 2.2 can be expressed as

$$x(t)=\sqrt{2}\,\mathrm{Re}\left[\sum_i s_i\, c(t-iT)e^{j2\pi f_c t}\right] \qquad ..2.4$$

or equivalently by

$$x(t)=\frac{1}{\sqrt{2}}\sum_i\left(s_i e^{j2\pi f_c t}+s_i^* e^{-j2\pi f_c t}\right)c(t-iT) \qquad ..2.5$$

where $s_i^* e^{-j2\pi f_c t}$ is the complex conjugate of $s_i e^{j2\pi f_c t}$. The input signal to the demodulator is given by

$$z(t)=x(t)*d(t)*h(t)*e(t)+n(t)*e(t) \qquad ..2.6$$

and the outputs of the two demodulator filters are

$$r_1(t)=\left(\sqrt{2}\,z(t)\cos(2\pi f_c t+\theta)\right)*f(t) \qquad ..2.7$$

and

$$r_2(t)=\left(-\sqrt{2}\,z(t)\sin(2\pi f_c t+\theta)\right)*f(t) \qquad ..2.8$$

where it is assumed that the frequency of the two reference carriers is equal to the frequency of the signal carrier, f_c, and that the phase difference between the reference carriers and the signal carrier is zero, i.e. $\theta=0$. The phase difference may be adjusted to be zero using phase locked loop techniques [14], therefore, it will be assumed that it is zero throughout the rest of this analysis. If we let

$$r(t)=r_1(t)+jr_2(t) \qquad ..2.9$$

then

$$r(t)=\sqrt{2}\left(z(t)e^{-j2\pi f_c t}\right)*f(t) \qquad ..2.10$$

substituting 2.5 and 2.6 into 2.10 yields the following expression for the received signal

$$r(t)=\sum_i s_i\left(c(t-iT)*\left((d(t)*h(t)*e(t))e^{-j2\pi f_c t}\right)\right)*f(t)$$

$$+\sum_i s_i^*\left(\left(c(t-iT)e^{-j4\pi f_c t}\right)*\left((d(t)*h(t)*e(t))e^{-j2\pi f_c t}\right)\right)*f(t)$$

$$+\sqrt{2}\left((n(t)*e(t))e^{-j2\pi f_c t}\right)*f(t) \qquad ..2.11$$

The term $c(t-iT)e^{-j4\pi f_c t}$ in the second summation in equation 2.11 represents a band-pass signal whose frequency sprectrum lies outside the frequency band of the low-pass filter F, therefore the second summation can be ignored. Thus equation 2.11

reduces to

$$r(t)=\sum_i s_i y(t-iT) + w(t) \quad ..2.12$$

where

$$y(t)=c(t)*\left((d(t)*h(t)*e(t))e^{-j2\pi f_c t}\right)*f(t) \quad ..2.13$$

and

$$w(t)=\sqrt{2}\left((n(t)*e(t))e^{-j2\pi f_c t}\right)*f(t) \quad ..2.14$$

The term in the the square brackets in equation 2.13 can be expressed alternatively as

$$(d(t)e^{-j2\pi f_c t})*(h(t)e^{-j2\pi f_c t})*(e(t)e^{-j2\pi f_c t}) \quad ..2.15$$

therefore the Fourier transform of equation 2.13 is

$$Y(f)=C(f)D(f+f_c)H(f+f_c)E(f+f_c)F(f) \quad ..2.16$$

If we consider $D(f+f_c)$, since d(t) is a real valued band-pass function then [5]

$$D(f)=D^*(-f) \quad ..2.17$$

Defining $D_0(f-f_c)$ as

$$D_0(f-f_c)=\begin{cases} D(f) & f>0 \\ 0 & f<0 \end{cases} \quad ..2.18$$

then

$$D_0^*(-f-f_c)=\begin{cases}0 & f>0\\ D^*(-f) & f<0\end{cases} \qquad ..2.19$$

thus from equation 2.17

$$D(f)=D_0(f-f_c)+D_0^*(-f-f_c) \qquad ..2.20$$

Taking the inverse Fourier transform of 2.20 gives [5]

$$\begin{aligned} d(t)&=d_0(t)e^{j2\pi f_c t}+d_0^*(t)e^{-j2\pi f_c t}\\ &=2\mathrm{Re}\left[d_0(t)e^{j2\pi f_c t}\right] \end{aligned} \qquad ..2.21$$

where the function $d_0(t)$ is said to be the baseband equivalent of d(t). Multiplying d(t) in 2.21 by $e^{-j2\pi f_c t}$ gives

$$d(t)e^{-j2\pi f_c t}=d_0(t)+d_0^*(t)e^{-j2\pi f_c t} \qquad ..2.22$$

The second term on the right hand side of equation 2.22 has spectral components that lie outside the frequency response of the low-pass filters C and F ,therefore, it can be ignored. From equation 2.22, and from a similar analysis for h(t) and e(t), the impulse response of the channel, y(t), given by equation 2.13, can be written as

$$y(t)=c(t)*d_0(t)*h_0(t)*e_0(t)*f(t) \qquad ..2.23$$

which represents the linear baseband channel formed by the two transmitter filters C, the linear modulators, filter D, the bandpass transmission path, filter E, the linear demodulators and the two receiver filters F. Equation 2.23 is in general a complex valued quantity. The Fourier transform of 2.23 is

$$Y(f)=C(f)D_0(f)H_0(f)E_0(f)F(f) \quad ..2.24$$

where $D_0(f)$, $H_0(f)$ and $E_0(f)$ are the baseband equivalent transfer functions of filter D, the transmission path and filter E, respectively. If we let

$$A(f)=C(f)D_0(f) \quad ..2.25$$

represent the overall filtering carried out at the transmitter, and

$$B(f)=E_0(f)F(f) \quad ..2.26$$

represent the overall filtering carried out at the receiver, then 2.24 becomes

$$Y(f)=A(f)*H_0(f)*B(f) \quad ..2.27$$

which is the transfer function of the linear baseband channel. The transfer functions A(f) and B(f) are adjusted so that $|A(f)|=|B(f)|$ such that if the transmission path introduced no distortion then the receiver filter would be matched to the transmitter filter and the signal-to-noise ratio at the output of filter B would be maximised [2,5].

The noise signal (equation 2.15), w(t), is a low-pass random process and is a complex valued quantity. If n(t) in equation 2.15 is a Gaussian random variable then both the real and imaginary parts of w(t) will also be Gaussian random variables.

Figure 2.3 shows a model of the linear baseband channel represented by equation 2.27, where the overall filtering carried out at the transmitter in figure 2.21 is represented by filter A, and likewise the overall filtering carried out at the receiver is represented by filter B. The sampler in figure 2.3 samples the received signal, r(t), once every T seconds to give the samples $\{r_i\}$, where

$$r_i=\sum_{h=0}^{g} s_{i-h}y_h+w_i \quad ..2.28$$

and $y_h=y(hT)$, $r_i=r(iT)$ and $w_i=w(iT)$. The $\{y_h\}$ represent the impulse response of the channel sampled once every T seconds. It is assumed that there is no delay in transmission, thus $y_0 \neq 0$, also $y_h=0$ for $h<0$ and $h>g$. Since the amplitude reponse of the baseband channel, $|Y(f)|$, is approximately zero for $|f|>1/2T$, the sampling rate of the sampler approximately satisfies Nyquists sampling theorem, and the samples $\{r_i\}$ contain all the necessary information needed by the detection process. From equation 2.28, a more simple model of the system can be derived, which is suitable for use in computer simulations. Figure 2.4 shows such a model, where the noise component in r(t) are generated separately by feeding the complex valued noise signal u(t) into a separate filter identical to the receiver filter B.

Using equation 2.12, the average energy per signal element in r(t) is

$$\mathcal{E}_s=E\left[|s_i|^2\int_{-\infty}^{\infty}|y(t)|^2dt\right] \qquad ..2.29$$

where E[.] is the expected value. If the $\{s_i\}$ are statistically independent and with zero mean then

$$\mathcal{E}_s=\overline{s_i^2}\int_{-\infty}^{\infty}|y(t)|^2dt \qquad ..2.30$$

where s_i^2 is given by

$$\overline{s_i^2}=E\left[|s_i|^2\right] \qquad ..2.31$$

For convenience, the impulse response of the channel y(t) is scaled such that the integral in equation 2.30 has a value equal to unity, therefore

$$\varepsilon_s = \overline{s_i^2} \qquad ..2.32$$

The representation of the QAM system by a linear baseband channel model, shown in figure 2.4, gives a simplified view of the data transmission system. Because of its simplicity, the linear baseband model is very useful for computer simulation tests that measure the performance of the system under different conditions.

2.3 Differential Encoding

At the transmitter (figure 2.1), the binary data stream is encoded using differential encoding. Differential encoding reduces the number of errors occurring in the detected data due to sudden phase changes in the signal carrier relative to the reference carriers at the receiver. Figure 2.5 shows a block diagram of the differential encoder. It is assumed that the binary digits entering the encoder are statistically independent and equally likely to have the values 0 or 1. As the binary digits enter the encoder they are split up into groups of four or six digits, $\{\alpha_{i,h}\}$, for h=1,2,...,b, where, depending on whether 16 or 64-level QAM is used, b=4 or 6, respectively. The first two binary digits, $\alpha_{i,1}$ and $\alpha_{i,2}$ in any one group are now recoded according to table 2.1, to give the coded binary digits $ß_{i,1}$ and $ß_{i,2}$, where, in table 2.1, $ß_{i-1,1}$ and $ß_{i-1,2}$ are the two previously differentially encoded digits. The remaining binary digits, $\alpha_{i,3}$ to $\alpha_{i,b}$ are left as they are, giving the digits $ß_{i,3}$ to $ß_{i,b}$ at the encoder output. Initially, when the first group of digits are encoded (when i=0), $ß_{-1,1}$ and $ß_{-1,2}$ are fixed to some arbitrary value, say 0. The resulting group of b digits now determine the corresponding coded data symbol s_i. Figures 2.6 and 2.7 show the signal constellations of the coded symbols for a 16 or 64-level QAM signal, respectively, where the binary numbers below each point are the values of the encoded binary digits, $ß_{i,1}, ß_{i,2}, \ldots ß_{i,b}$. The first two binary digits, $ß_{i,1}$ and $ß_{i,2}$ determine which quadrant in figure 2.6 or 2.7 s_i lies in and the remaining digits, $ß_{i,3}$ to $ß_{i,b}$ determine the position of s_i within a particular quadrant. To further help reduce the probability of errors occurring, due to noise, in the detected

data, Gray coding has been used within each of the quadrants as can be seen in figures 2.6 and 2.7. Exact Gray coding is not possible over the entire signal constellation.

At the receiver, after the data symbol s_i has been detected, the corresponding values of the encoded binary digits, $ß_{i,1}, ß_{i,2}, \ldots ß_{i,b}$, are found using figure 2.6 or 2.7 The values of $\alpha_{i,1}$ and $\alpha_{i,2}$ are then determined using table 2.1, where $ß_{i,1}$ and $ß_{i,2}$ are now the detected coded digits, and $ß_{i-1,1}$ and $ß_{i-1,2}$ are the previously detected coded digits.

It can be seen from table 2.1 and figure 2.6 or figure 2.7 that a phase shift of $\pi/2$ radians (or any multiple) in the phase relationship between the reference carriers in the coherent demodulators and the received signal carrier, giving the corresponding rotation in the phase angle of a received sample r_i, doesn't change the detected values of $\alpha_{i,3}$ to $\alpha_{i,b}$, corresponding to any given value of s_i, nor can it lead to any prolonged burst of errors in the detected values of $\alpha_{i,1}$ and $\alpha_{i,2}$.

Table 2.1 Differential Encoding of the 16 or 64-Level QAM Signal

$\alpha_{i,1}$	$\alpha_{i,2}$	$\beta_{i-1,1}$	$\beta_{i-1,2}$	$\beta_{i,1}$	$\beta_{i,2}$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

Figure 2.1 Model of the Digital Data Transmission System

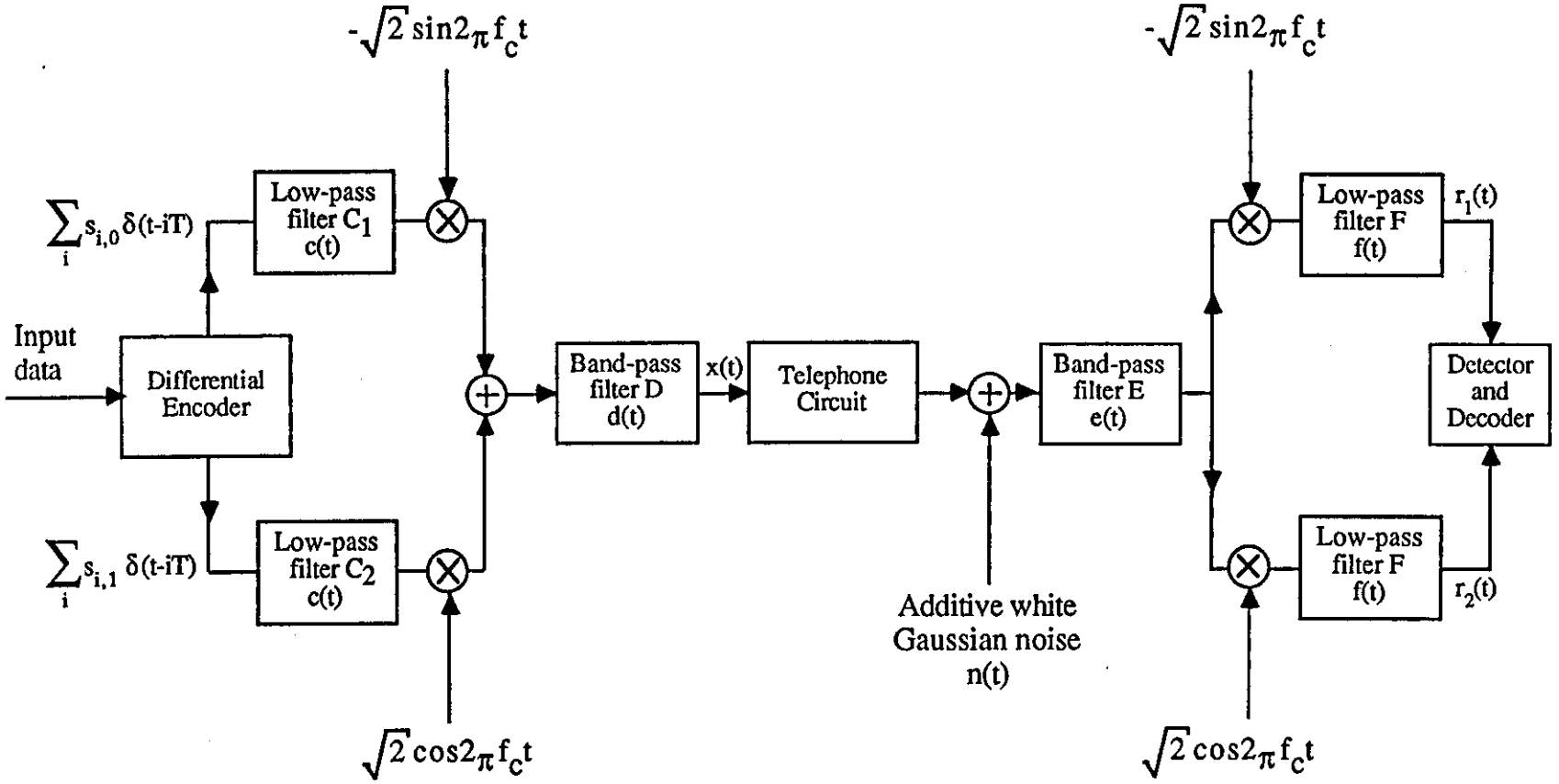


Figure 2.2a Amplitude Response of Filter C

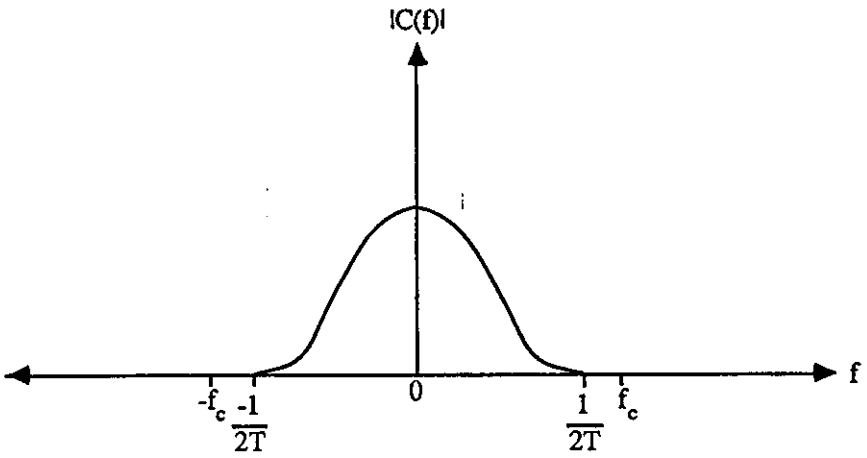


Figure 2.2b Frequency Spectrum of x(t)

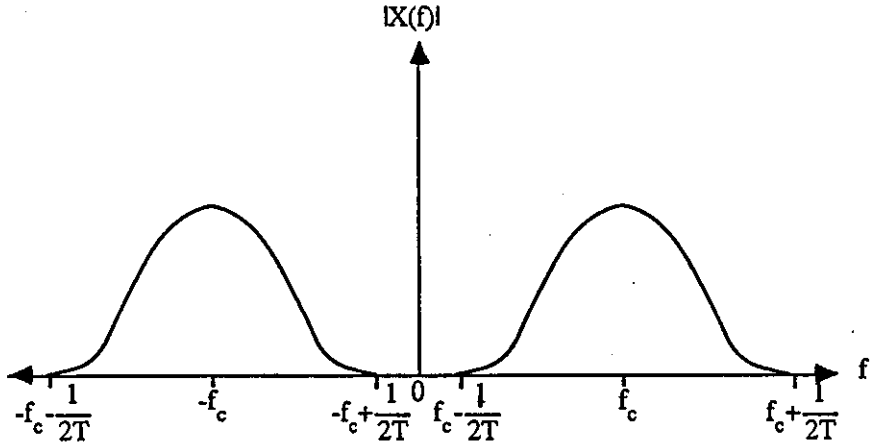


Figure 2.2c Amplitude Response of Filter E

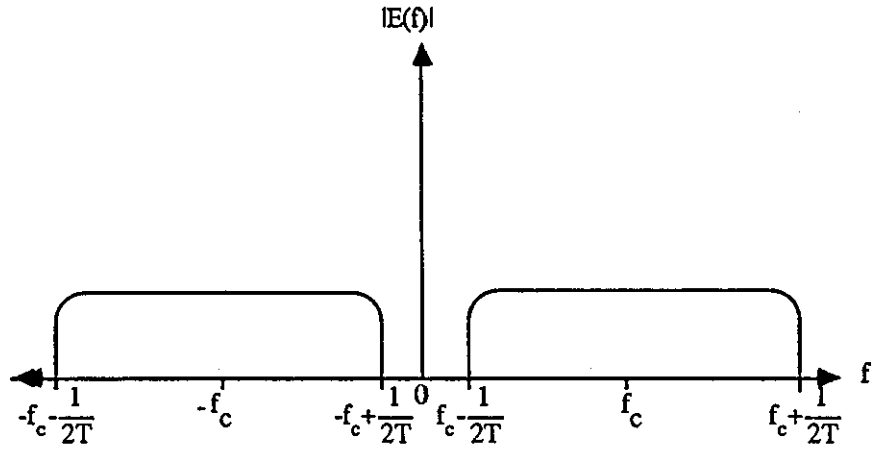


Figure 2.3 Equivalent Baseband Model of Data Transmission System

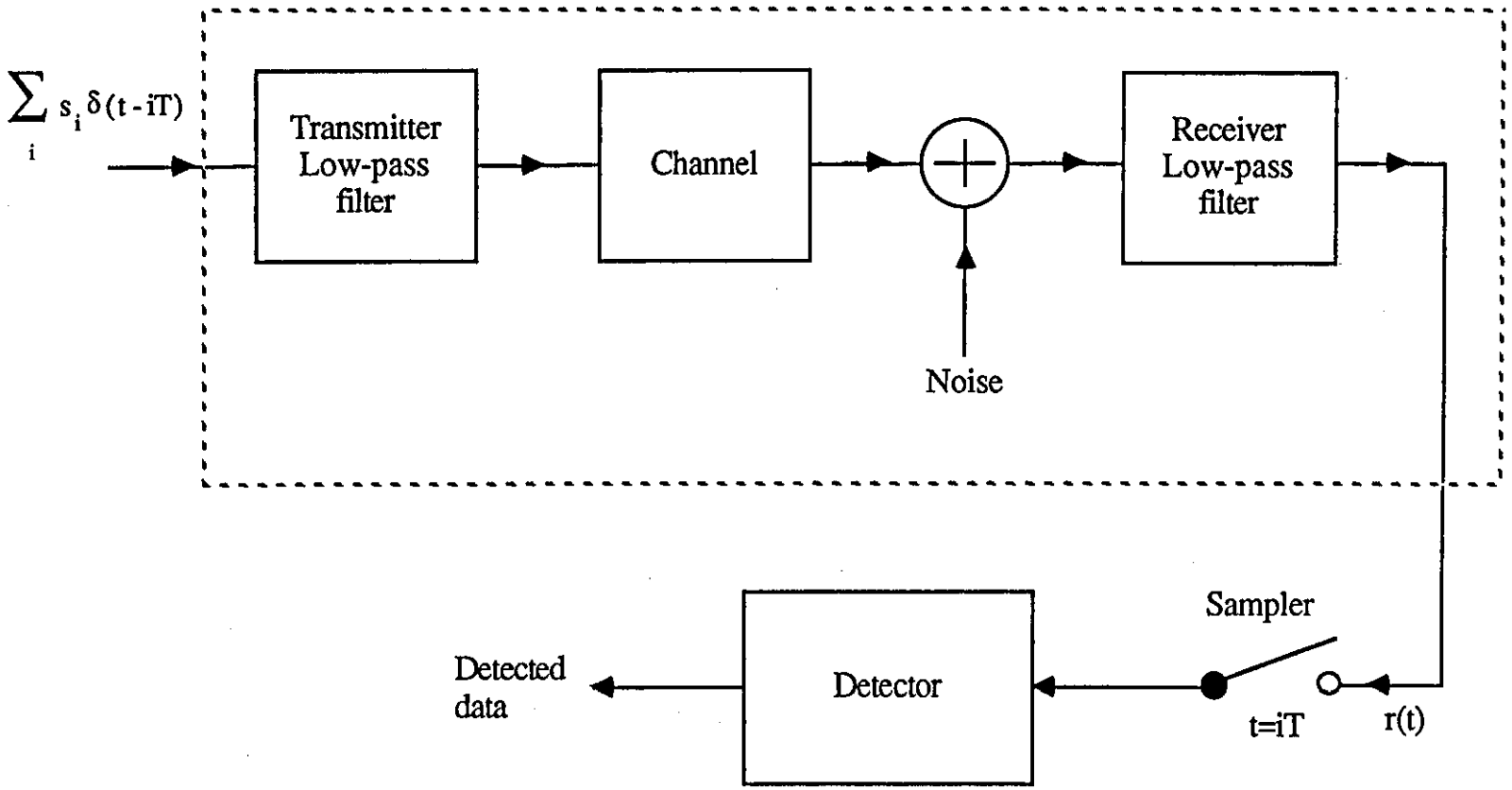


Figure 2.4 Simplified Baseband Model of Data Transmission System used in the Computer Simulations

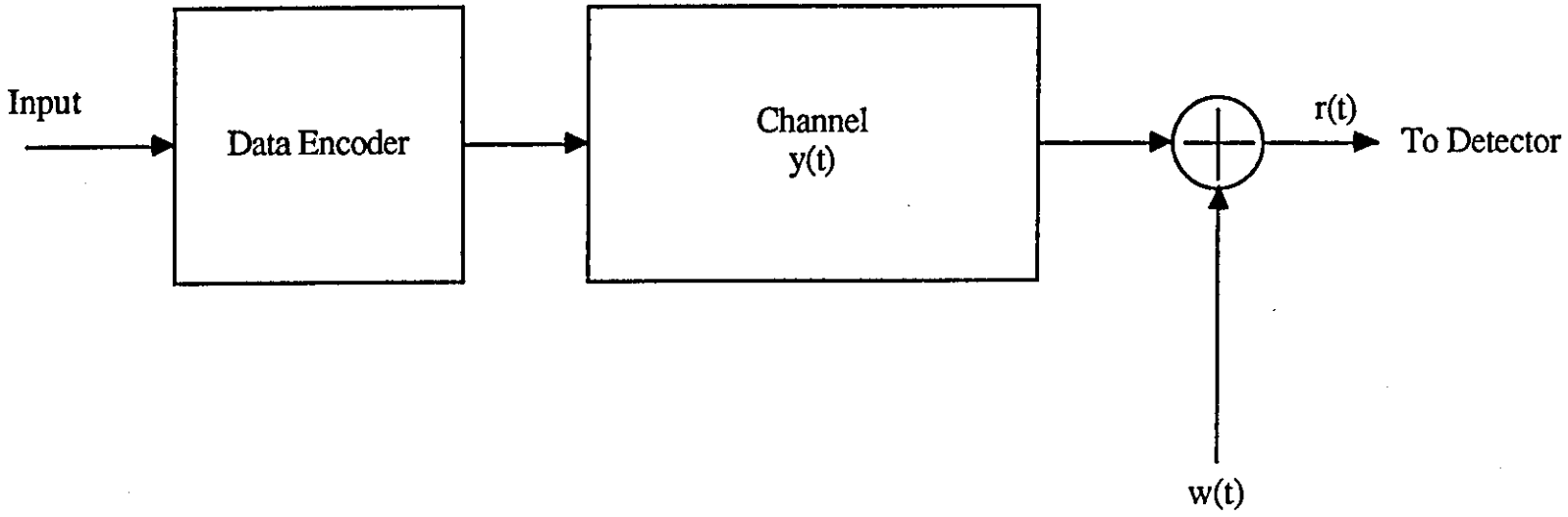
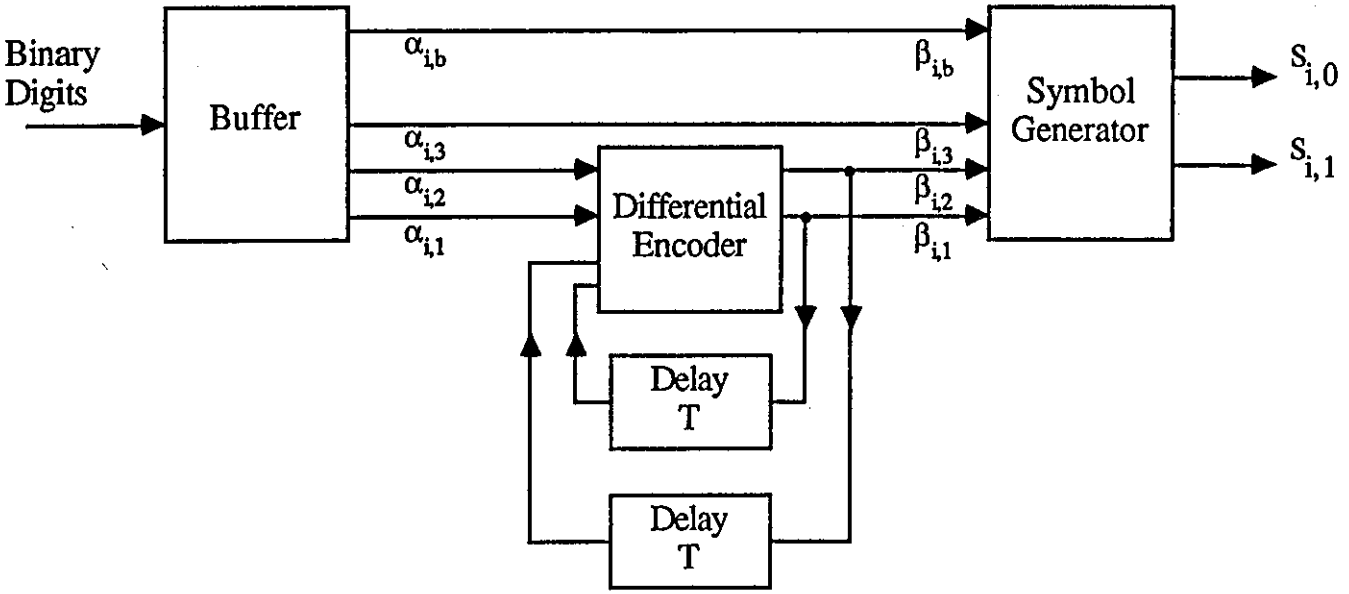


Figure 2.5 Block Diagram of the Differential Encoder



Note: b=4 or 6 for 16 or 64-level QAM respectively

Figure 2.6 Signal Constellation for a 16-level Differentially Encoded QAM Signal

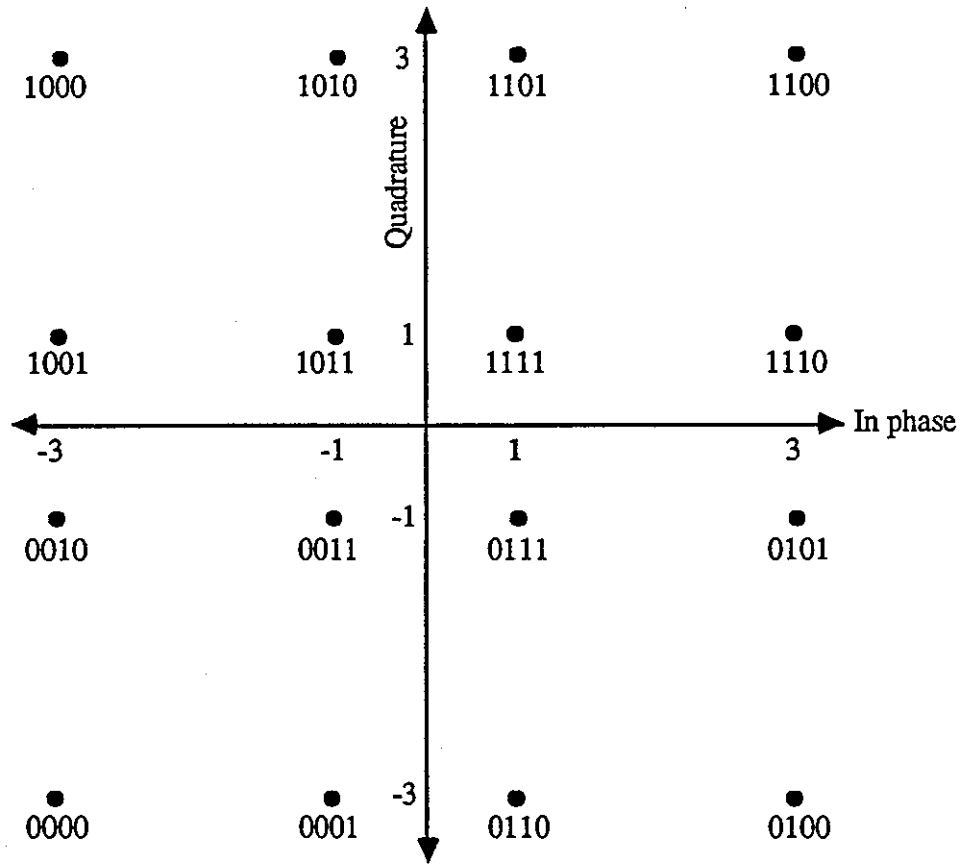
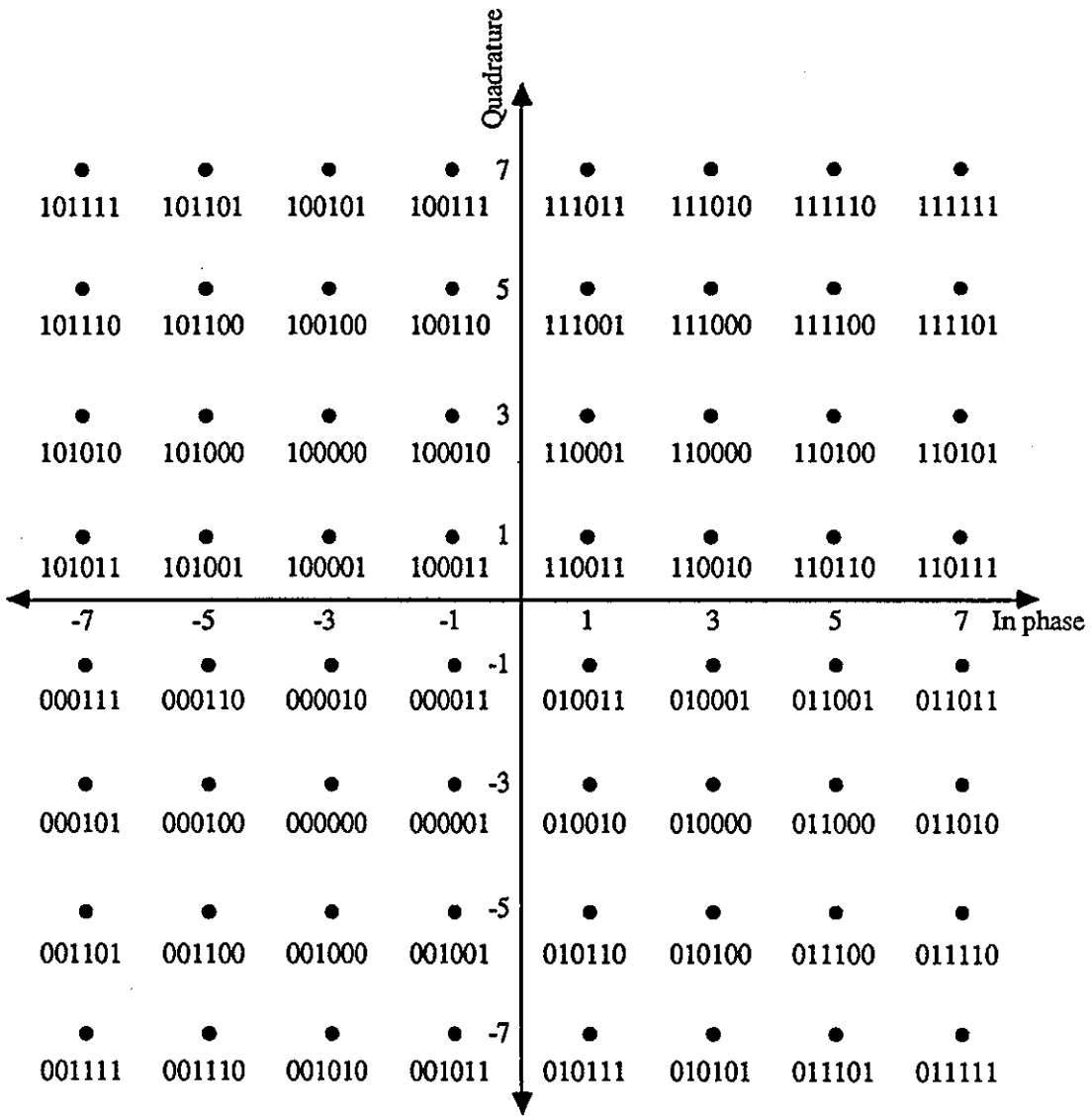


Figure 2.7 Signal Constellation for a 64-level Differentially Encoded Signal



CHAPTER 3
DETECTION PROCESSES

3.1 Introduction

In a digital data transmission system, where a signal is transmitted over a channel that introduces severe distortion, such as a telephone channel, the distortion will be too great for a simple threshold detector to be of any use as a means of detecting the data from the received signal. A more effective means of detecting the data from the received distorted signal is therefore needed.

Let the model of the data transmission system be as shown in figure 3.1. The information is carried by the data symbols $\{s_i\}$, where

$$s_i=s_{i,0}+js_{i,1} \qquad ..3.1$$

and the possible values of $s_{i,0}$ and $s_{i,1}$ are given by

$$s_{i,0}, s_{i,1}=2j-\sqrt{m}+1 \qquad j=0,1,\ldots,(\sqrt{m}-1) \qquad ..3.2$$

and where m is the number of possible values that s_i can take on. It is assumed that all of the possible values of s_i are equally likely to occur and that they are statistically independent. In figure 3.1, the data symbols are fed into the input of the transmitter filter in the from of a stream of impulses, $\sum s_i\delta(t-iT)$. These impulses are fed into the system at a rate of 1/T symbols per second, which is approximately equal to the Nyquist rate [2], since it is assumed that the amplitude response the linear base-band channel is close to zero for frequencies $|f|>1/2T$. The output of the receiver filter is given by

$$r(t)=\sum_i s_i y(t-iT) + w(t) \qquad ..3.3$$

where y(t) is the impulse response of the linear base-band channel and w(t) is the complex valued base-band noise signal. The waveform r(t) is sampled once per data symbol period, T, at times t=iT, to give the received samples $\{r_i\}$, where r_i is given by

$$r_i=\sum_{h=0}^{g} s_{i-h} y_h + w_i \qquad ..3.4$$

and $r_i=r(iT)$, $y_h=y(hT)$ and $w_i=w(iT)$. It is assumed that the duration of the impulse response lasts for only (g+1)T seconds, where g is a positive integer. Thus, the sampled impulse response of the channel is given by the (g+1) component row vector

$$Y=[y_0\ y_1\ \ldots\ y_g] \qquad ..3.5$$

Now, equation 3.4 can be expressed alternatively as

$$r_i=s_i y_0 + \sum_{h=1}^{g} s_{i-h} y_h + w_i \qquad ..3.6$$

The term $s_i y_0$ on the right hand side of equation 3.6 is the wanted component of r_i, from which the value of s_i can be determined using a simple threshold detector. The second term on the right hand side of equation 3.6 represents the intersymbol interference in r_i, caused by the channel.

Many methods for detecting the data symbols $\{s_i\}$ in the prescence of intersymbol interference and noise have been developed and these may be broadly classified into two distinct groups. The first type of detectors make use of a filter, called an equalizer, which is usually placed ahead of the detector, and attempts to remove the distortion

introduced by the channel into the received samples $\{r_i\}$. In the second group of detectors the decision process itself is modified to take into account the intersymbol interference present in the received samples $\{r_i\}$, in fact no attempt need be made to reduce the distortion present in the received signal prior to detection. Generally the second group of detection processes out-perform the detectors making use of equalization techniques.

3.2 The Decision Feedback Equaliser [6]

As mentioned previously, an equaliser can be used to remove all or part of the distortion present in the received signal. There three main types of equaliser, known as, the linear equaliser, the pure non-linear equaliser and the decision feedback equaliser (DFE). The DFE is, in fact, a combination of the linear and pure non-linear equaliser and generally gives a superior performance over the two former types of equaliser when used for the detection of data sent over typical voice-band channels. The decision feedback equaliser will now be described.

The basic structure of the decision feedback equaliser is shown in figure 3.2. The received signal, r(t), is sampled once per data symbol period to give the sequence of samples $\{r_i\}$ at the input of the transversal filter D. From equation 3.4,

$$r_i = \sum_{h=0}^{g} s_{i-h} y_h + w_i \qquad ..3.7$$

which can alternatively be expressed as

$$r_i = s_i y_0 + \sum_{h=1}^{g} s_{i-h} y_h + w_i \qquad ..3.8$$

where $s_i y_0$ is the desired component of the sample r_i from which s_i can be detected, and $\Sigma s_{i-h} y_h$ represents the intersymbol interference contained in the sample r_i.

Let the sampled impulse response of the transversal filter D be represented by the q-component row vector D

$$D=[d_0 d_1 \ldots d_{q-1}] \qquad ..3.9$$

and let the z-transform of this sequence be

$$D(z)=d_0+d_1 z^{-1}+\ldots+d_{n-1} z^{-q+1} \qquad ..3.10$$

Also, let the tap weights of the transversal filter F be represented by the g-component row vector F

$$F=[f_1 f_2 \ldots f_g] \qquad ..3.11$$

and the let the z-transform of the sampled impulse response of the channel be

$$Y(z)=y_0+y_1 z^{-1}+\ldots+y_g z^{-g} \qquad ..3.12$$

Y(z) can alternatively be expressed as the product of two polynomials [6], i.e.,

$$Y(z)=Y_1(z)Y_2(z) \qquad ..3.13$$

where

$$Y_1(z)=1+a_0 z^{-1}+\ldots+a_{g-m} z^{-m+g} \qquad ..3.14$$

and

$$Y_2(z)=b_0+b_1 z^{-1}+\ldots+b_m z^{-m} \qquad ..3.15$$

where $Y_1(z)$ is a polynomial with roots that lie inside the unit circle in the complex z-plane and $Y_2(z)$ is a polynomial with roots outside the unit circle and m is an integer that has a value between 0 and g and is the number of roots of Y(z) that lie outside the unit circle in the complex z-plane. Let $Y_3(z)$ be a polynomial given by

$$Y_3(z)=b_m^*+b_{m-1}^* z^{-1}+\ldots+b_0^* z^{-m} \qquad ..3.16$$

where * denotes the complex conjugate. It can be seen that the sequence with z-transform $Y_3(z)$ is the reverse of the sequence with z-transform $Y_2(z)$, hence $Y_3(z)$ has its roots at the reciprocal of the conjugate positions of those of $Y_2(z)$ in the complex z-plane.

Now, there are two possible combinations of the tap weights of filters D and E that give optimum or near optimum performance [6]. The first attempts to minimise the mean square error in the signal p_i in figure 3.2 and is known as a minimum mean square error (MMSE) equaliser [6,7]. In the second, the channel is accurately equalized, and, subject to this constraint, it minimises the mean square error in the equalized signal. This second type of equaliser is known as a zero forcing equaliser (ZF) and is generally accepted as being the optimum DFE [6]. As the signal-to-noise ratio increases the performance of the MMSE equaliser approaches that of the ZF equaliser. But at very low signal-to-noise ratios the MMSE equaliser may or may not give a better performance [6]. It can be shown that the optimum values of the tap weights (for zero forcing) of filters D and E are

$$\begin{aligned} Y(z)D(z)&=(b_m^*)^{-1}z^{-h}Y_1(z)Y_3(z) \\ &=z^{-h}[1+e_1z^{-1}+\ldots+e_gz^{-g}] \\ &=z^{-h}E(z) \end{aligned} \qquad ..3.17$$

where h is a positive integer, and the sequence with z-transform E(z) is the desired sampled impulse response of the channel in cascade with filter D. Thus, D(z) is given by

$$D(z)=(b_m^*)^{-1}z^{-h}Y_2^{-1}(z)Y_3(z) \qquad ..3.18$$

from which the tap weights of filter D can be calculated. In the optimum DFE the linear filter D, when correctly adjusted, removes the zeros of Y(z) that lie outside the unit circle in the complex z-plane and replaces them with zeros at their reciprocal of their conjugate positions in the complex z-plane.

The sequence $\{p_i\}$ at the output of filter D are given by

$$p_i = \sum_{j=0}^{g} s_{i-j}\, e_j + u_i \qquad ..3.19$$

where u_i represents the filtered noise samples. From equation 3.17, it can be seen that the first component of the row vector E will be unity, therefore equation 3.19 becomes

$$p_i = s_i + \sum_{j=1}^{g} s_{i-j}\, e_j + u_i \qquad ..3.20$$

The second term in equation 3.20, which represents the intersymbol interference, can now be removed by setting the tap gains of filter F (figure 3.2) such that

$$F = [\, e_1\, e_2 \ldots e_g \,] \qquad ..3.21$$

and hence q_i will be

$$q_i = \sum_{j=1}^{g} s'_{i-j}\, e_j \qquad ..3.22$$

and thus, the signal at the output of the detector (figure 3.4) will be

$$x_i = p_i - q_i$$

$$= s_i + u_i \qquad ..3.23$$

Where, in equation 3.22, the $\{s_i'\}$ represent the detected values of s_i. If the values of s_i' have been correctly detected (i.e. $s_i' = s_i$) then the value of q_i will be equal the second term of equation 3.20. In the abscence of noise and with correct adjustment of filters D

and F, $x_i=s_i$.

Since the characteristics of the telephone channel may vary considerably from call to call, or during the duration of a single call, the two filters, D and F, must be adaptively adjusted for each call before actual transmission of data can start. Figures 3.3 and 3.4 show the structure of an adaptive DFE, which is based upon the steepest descents method to adaptively adjust the tap gains of the two filters [26].

The signal at the input of the detector is

$$x_i=s_i'+u_i \qquad ..3.24$$

The error in x_i is $e_i=x_i-s_i'$ and when s_i' is correctly detected then $s_i'=s_i$ and $e_i=x_i-s_i$. The mean square error in x_i can be taken to be

$$\bar{\varepsilon}_i=\overline{(x_i-s_i')^2} \qquad ..3.25$$

It can be shown that provided $\{s_i\}$ are statistically independent, the mean square error is minimised when the tap gains of filter D are incremented according to [6]

$$d_j=d_{j-1}+\delta d_j \qquad ..3.26$$

where

$$\delta d_j=-a\varepsilon_i(r_{i-j})^* \qquad \text{for } j=0,1,\ldots,(n-1) \qquad ..3.27$$

and the tap gains of filter F are incremented using

$$f_j=f_{j-1}+\delta f_j \qquad ..3.28$$

where

$$\delta f_j = +a\varepsilon_i (s_{i-j})^* \qquad \text{for } j=0,1,\ldots,(n-1) \qquad ..3.29$$

and a is a suitable small positive constant. The smaller the value of a the smaller the effects of noise on $\{d_i\}$, but, at the same time, the slower the rate of convergence. At high signal to noise ratios the tap gains of filters D and F are adjusted to minimise the error in s_i' and the adaptive equaliser approaches the optimum DFE [6].

3.3 The Optimum Detection Process

The main weakness of using a decision feedback equaliser is that only a portion of the received signal element is used in the detection of that element, the remaining part of that element being removed by cancellation and not involved in the actual detection process. If the detection process is to have the best tolerance to additive white Gaussian noise, i.e. the detector minimises the probability of an error in the detection of the complete message, then the whole of the received signal must be used by the detector. This leads to two definitions of the optimum detector [6], the first type minimises the probability of error in the detection of the whole of the received message, and the second type minimises the error probability in the detection of an individual data symbol and hence minimises the average bit error rate over all the received data symbols, provided they are statistically independent.

If a sequence of data symbols $\{s_i\}$, are transmitted, as shown in figure 3.1, then the received samples at times t=iT, will be given by

$$r_i = z_i + w_i \qquad ..3.30$$

where

$$z_i=\sum_{h=0}^{g} s_{i-j}y_h \qquad ..3.31$$

It is assumed that $\{w_i\}$ are statistically independent Gaussian random variables that represent the noise components in the received samples $\{r_i\}$. The $\{r_i\}$, $\{z_i\}$, $\{w_i\}$, $\{s_i\}$ and $\{y_i\}$ are, in general, complex valued quantities. The real and imaginary parts of w_i are Gaussian random variables with zero mean and varience σ^2.

If the total number of transmitted symbols is N, i.e. i=1, 2,. . . ,N , then the samples $\{r_i\}$, $\{z_i\}$, $\{w_i\}$ and $\{s_i\}$ can be represented by the N-component row vectors R_N, Z_N, W_N and S_N, where

$$R_N=[r_1 r_2 \ldots r_N] \qquad ..3.32$$

$$Z_N=[z_1 z_2 \ldots z_N] \qquad ..3.33$$

$$W_N=[w_1 w_2 \ldots w_N] \qquad ..3.34$$

and

$$S_N=[s_1 s_2 \ldots s_N] \qquad ..3.35$$

From equations 3.30, 3.32, 3.33 and 3.34

$$R_N=Z_N+W_N \qquad ..3.36$$

Once all of the data symbols have been transmitted, the vector R_N will be known to the detector at the receiver and the detection process can start. Initially, the detector computes the m^N a-posteriori probabilities [3,33,35]

$$P(S_N=X_{N,i}|R_N) \qquad \text{for } i=1,2,\ldots,m^N \qquad ..3.37$$

where $X_{N,i}$ is one of the m^N possible values of the vector S_N. The detector accepts the hypothesis $S_N=X_{N,j}$ if [6,33-35]

$$P(S_N=X_{N,j}|R_N)>P(S_N=X_{N,i}|R_N) \qquad i=1,2,\ldots,m^N,\ i\neq j \qquad ..3.38$$

Applying Bayes theorem [6,35] to equation 3.38 leads to the receiver making the decision $S_N=X_{N,j}$ when,

$$p(R_N|S_N=X_{N,j})P(S_N=X_{N,j}|R_N)>P(S_N=X_{N,i})p(R_N|S_N=X_{N,i}) \qquad ..3.39$$

$$\text{for } i=1,2,\ldots,m^N,\ i\neq j$$

where $P(S_N=X_{N,i})$ is the a-priori probability that $S_N=X_{N,i}$, $p(R_N|S_N=X_{N,i})$ is the conditional probability density function of R_N given $X_{N,i}$ has been transmitted. For the special case where all of possible values of s_i are equally likely to occur and hence all m^N values of S_N are equally likely to occur, equation 3.39 reduces to

$$p(R_N|S_N=X_{N,j})>p(R_N|S_N=X_{N,i}) \qquad \text{for } i=1,2,\ldots,m^N,\ i\neq j \qquad ..3.40$$

where $X_{N,j}$ is the value of $X_{N,i}$ that maximises $p(R_N|S_N=X_{N,i})$, which is known as the likelihood function of $X_{N,i}$, and the detector that selects the value of i for which this function is maximum is known as a maximum likelihood detector [6,26,31,34,35].

Now, since there is a one-to-one mapping between S_N and Z_N (equations 3.30 and 3.31) then

$$p(R_N|S_N=X_{N,i})=p(R_N|Z_N) \qquad ..3.41$$

where $p(R_N|Z_N)$ is the conditional probability density function of R_N given Z_N. Since

the samples $\{r_i\}$ and $\{z_i\}$ are statistically independent then $p(R_N|Z_N)$ can be expressed as

$$p(R_N|Z_N)=p(r_1,r_2,\ldots,r_N|z_1,z_2,\ldots,z_N) \qquad ..3.42$$

Let the complex values r_i and z_i be written as

$$r_i=r_{i,1}+jr_{i,2} \qquad ..3.43$$

and

$$z_i=z_{i,1}+jz_{i,2} \qquad ..3.44$$

for i=1,2,. . . ,N. Since $\{w_i\}$ are statistically independent Gaussian random variables with zero mean and variance σ_w^2, then for a given transmitted vector S_N and hence for a given vector Z_N, the received samples, $\{r_i\}$, are statistically independent random variables with mean z_i and variance σ_w^2, thus [3]

$$p(R_N|Z_N)=p(r_{i,1}|z_{i,1})p(r_{i,2}|z_{i,2})\ldots p(r_{N,1}|z_{N,1})p(r_{N,2}|z_{N,2}) \qquad ..3.45$$

so for each $r_{i,h}$

$$p(r_{i,h}|z_{i,h})=\frac{1}{\sqrt{2\pi\sigma_w^2}}\exp\left(\frac{-(r_{i,h}-z_{i,h})^2}{2\sigma_w^2}\right) \qquad ..3.46$$

for h=1,2 and i=1,2,. . . ,N.

Substituting 3.46 into 3.45 gives

$$p(R_N|Z_N)=\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi\sigma_w^2}}\exp\left(\frac{-(r_{i,1}-z_{i,1})^2}{2\sigma_w^2}\right)$$

$$\cdot\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(r_{i,2}-z_{i,2})^2}{2\sigma_w^2}\right)$$

$$=\frac{1}{(2\pi\sigma^2)^{N/2}}\exp\left(\frac{-1}{2\sigma_w^2}\sum_{i=1}^{N}(r_{i,1}-z_{i,1})^2+(r_{i,2}-z_{i,2})^2\right)$$

$$=\frac{1}{(2\pi\sigma^2)^{N/2}}\exp\left(\frac{-|R_N-Z_N|^2}{2\sigma_w^2}\right) \qquad ..3.47$$

where

$$|R_N-Z_N|^2=\sum_{i=1}^{N}|r_i-z_i|^2$$

$$=\sum_{i=1}^{N}((r_{i,1}-z_{i,1})^2+(r_{i,2}-z_{i,2})^2) \qquad ..3.48$$

and $|R_N-Z_N|$ is the length of the vector (R_N-Z_N) and so it is the unitary distance in the

N-dimensional vector space between the vectors R_N and Z_N [26]. It can be seen from equation 3.47 that $p(R_N|Z_N)$ is maximised by minimising $|R_N-Z_N|$, as well as reducing the noise variance. Thus, for the conditions assumed, the maximum likelihood detector that selects a value of $X_{N,j}$ satisfying equation 3.40, will select a possible vector $X_{N,i}$ for which the corresponding signal vector Z_N is closest in the N-dimensional vector space to the received vector R_N. The computation of the m^N a-posteriori probabilities suggested by equation 3.40 can therefore be replaced by the evaluation of the m^N possible values of $|R_N-Z_N|^2$ given by equation 3.48, which are known as 'costs'. The vector $X_{N,i}$ giving the vector (R_N-Z_N) with the smallest cost will be the selected vector.

The above process does not minimise the probability of error in the detection of individual data symbols, but at high signal-to-noise ratios and with white Gaussian noise, the described process will approximately minimise the average probability of an error in the detection of an individual symbol [6]. Since m and N will, in general, be large the maximum likelihood detection process described above will be far too complex to implement since it will involve the evaluation of equation 3.48 m^N times. However this process can be greatly simplified using recursive technique known as the Viterbi algorithm. This makes use of the knowledge that a large number of the vectors $X_{N,i}$ have a very low probability of being equal to the transmitted sequence S_N, and thus not all of the m^N costs (equation 3.48) need to be calculated.

3.3.1 The Viterbi Algorithm

The Viterbi algorithm [18] is a recursive method of obtaining the maximum likelihood estimate of the transmitted sequence of data symbols, S_N, without the need for computing all of the m^N costs given by equation 3.48. Its operation will now be described.

Let X_N be a possible value of the transmitted sequence of data symbols given by the components of the vector S_N and let the vector X_i (known as a survivor) be an initial segment of the vector X_N. Also let the g-component vector Q_i' be a vector whose

components are equal to the last g components of X_i, i.e.

$$X_N=[x_1\ x_2, \ldots, x_N] \qquad ..3.49$$

$$X_i=[x_1\ x_2, \ldots, x_i] \qquad ..3.50$$

and

$$Q_i'=[x_{i-g+1}, x_{i-g+2}, \ldots, x_i] \qquad ..3.51$$

Now, for any time, say t=iT, the detector can only hold in memory m^g vectors (survivors) $\{X_i\}$. Associated with each of these stored vectors is a cost given by

$$|R_i-X_i|^2=\sum_{j=1}^{i}|r_j-x_j|^2$$

$$=\sum_{j=1}^{i}((r_{j,1}-x_{j,1})^2+(r_{j,2}-x_{j,2})^2) \qquad ..3.52$$

where the vector R_i is the i component vector

$$R_i=[r_1 r_2, \ldots, r_i] \qquad ..3.53$$

The m^g stored vectors have the m^g smallest possible costs associated with them. For all of the survivors their last g components which form the vectors $\{Q_i'\}$ take on one of the m^g possible combinations of data symbols. Now, the detector does not know which of the vectors Q_i' will form part of the maximum likelihood vector X_N giving the smallest cost, but it definately knows that Q_i' can only have m^g different values. Therefore, for any value of i, it can be decided that there are only m^g possible vectors

$\{X_i\}$ which may be candidates to form part of the vector X_N giving the smallest cost.

Now, at time t=(i+1)T, the detector, in turn, takes each of the vectors $\{X_i\}$ and adds an extra component, x_{i+1}, to the end, where x_{i+1} takes on one of the m possible values of s_i. Thus from each stored vector X_i, m expanded vectors $\{X_{i+1}\}$ are created. Expanding all m^g stored vectors will give m^{g+1} expanded vectors $\{X_{i+1}\}$ and their associated costs. But the detector can only hold in store m^g vectors, and so it selects, from the m^{g+1} expanded vectors, the m^g vectors whose last g components take on all m^g possible combinations of component values and have the smallest costs.

This recursive process continues until t=NT, when the detector has in store the m^g vectors $\{X_N\}$ along with their associated costs. It then selects from these vectors the vector with the smallest cost associated with it and gives this as the maximum likelihood estimate of the transmitted sequence S_N and the detection process will have been completed.

Using the Viterbi algorithm, a very large saving can be made in the number of numerical and sorting operations needed to give a maximum likelihood estimate of the transmitted sequence of data symbols. For each detection instant the Viterbi algorithm computes m^{g+1} costs (equation 3.48), so for a complete message of N data symbols, it computes Nm^{g+1} costs. This is to be compared with m^N cost evaluations using the detection process suggested by equation 3.40.

Unfortunately, the Viterbi algorithm is still too complex for it to be implemented in practice. This is because as i increases the lengths of the survivors $\{X_i\}$ increase, and hence the amount of storage required by the detector will increase linearly with time. This problem can be overcome (with a loss of the detectors optimality) by assuming that the selected survivors $\{X_i\}$ at time t=iT are all very close to the actual transmitted vector, S_i, and that it can be assumed the first i-n components of each of the survivors do not change after each sampling instant, where n is a suitable integer (usually n>g). The lengths of the stored vectors $\{X_i\}$ can therefore be fixed to n components. Now, at time t=iT, the detector operates by first expanding each of the m^g n-component

survivors $\{X_i\}$ to give m^{g+1} (n+1)-component vectors, from which are selected the m^g different expanded vectors with the smallest costs, the detector then discards the first component of each of the selected vectors since they all should (provided n is large enough) be identical to give the m^g n-component vectors $\{X_{i+1}\}$ ready for the next sampling instant at time t=(i+2)T. The value of the discarded component will, in fact be part of the maximum likelihood estimate of S_N and hence there is no need to wait for the arrival of the complete message before the values of the components of S_N can be given, a delay of only n sampling instants is needed before a detected data symbol is given.

Many derivations of the above process have been developed to further reduce the complexity and the amount of storage needed, whilst maintaining a performance that comes close to that of the optimum detector. These simplified detectors are known as near maximum likelihood detectors.

3.3.2 Near Maximum Likelihood Detectors

The detection process mentioned in the previous section represents a large saving in equipment complexity, but the number of stored vectors is still too large. This number can be drastically reduced by noticing that a large number of the stored vectors do not play an important role in the selection process and hence can be ignored. The following two detectors to be described have in store k (where $k \ll m^g$) vectors.

Detector A [6,16]

Just prior to the receipt of the sample r_i at time t=iT, the detector hold in store the k n-component vectors $\{Q_{i-1}\}$, where

$$Q_{i-1}=[x_{i-n}x_{i-n+1}\cdots x_{i-1}] \qquad ..3.54$$

and x_{i-h}, for h=1,2, . . . ,n, takes on one of the possible values of s_{i-h}, the k vectors

$\{Q_{i-1}\}$ being all different. Each of the stored vectors Q_{i-1} represent a possible sequence of transmitted data symbols $s_{i-n}, s_{i-n+1}, \ldots, x_i$. On receipt of r_i at t=iT the detector expands each of the stored vectors Q_{i-1}, by adding an extra component to the end of each vector to give mk (n+1)-component sequences $\{P_i\}$,

$$P_i=[x_{i-n}x_{i-n+1}\ \ldots\ x_i] \qquad ..3.55$$

and x_i has one of the m possible values of s_i. In each group of vectors $\{P_i\}$, derived from one of the vectors Q_{i-1}, the first n components are the same as the n components in the original vector, and the last component, x_i, takes on one of the m possible values of s_i. Associated with each of the stored vectors Q_{i-1} is the quantity

$$C_{i-1}=\sum_{j=1}^{i-1}|r_j-\sum_{h=0}^{g}x_{j-h}y_h|^2 \qquad ..3.56$$

where C_{i-1} is the cost of the vector Q_{i-1}. The smaller the cost of one of the vectors Q_{i-1}, the more likely it forms part of the maximum likelihood estimate of the transmitted sequence S_N. The costs of the expanded vectors $\{P_i\}$ are now computed by the detector using the following expression

$$C_i=\sum_{j=1}^{i}|r_j-\sum_{h=0}^{g}x_{j-h}y_h|^2$$

$$=C_{i-1}+|r_i-\sum_{h=0}^{g}x_{i-h}y_h|^2 \qquad ..3.57$$

The detector now selects from the mk expanded vectors $\{P_i\}$ the vector which has the smallest cost associated with. The detected value of s_{i-n}' is now taken to be the first component x_{i-n} of the vector P_i with the smallest cost. Now, any expanded vector whose first component x_{i-n} does not equal the detected value s_{i-n}' is discarded, and from

the remaining vectors are selected the k-1 vectors having the next smallest costs $\{C_i\}$. The first component of each of the k selected vectors $\{P_i\}$ is then omitted (without changing its cost) to give the corresponding vectors $\{Q_i\}$, which are stored, together with their costs $\{C_i\}$, ready for the detection of the next data symbol s_{i-n+1}' at time t=(i+1)T. The discarding of the vectors whose first components differ from the detected data symbol s_{i-n}' ensures that the k stored vectors $\{Q_i\}$ are always different, provided that they were all different at the first detection process, which can easily be arranged.

The above detector involves the calculation of mk costs and the searching of k costs per sampling instant, this is a large reduction in complexity when compared with the Viterbi algorithm which involves the calculation of Nm^{g+1} costs. Further reductions in complexity can be achieved with some loss of performance as will now be described.

Detector B [10]

As with detector A, just prior to the receipt of the signal sample ri, the detector holds in store the k n-component vectors $\{Q_{i-1}\}$,

$$Q_{i-1}=[x_{i-n}x_{i-n+1}\cdots x_{i-1}] \qquad ..3.58$$

Associated with each of these vectors is a cost, given by

$$C_{i-1}=\sum_{j=1}^{i-1}|r_j-\sum_{h=0}^{g}x_{j-h}y_h|^2 \qquad ..3.59$$

Now, upon receipt of the sample r_i, each of the stored vectors Q_{i-1} is expanded into two (n+1)-component vectors $\{P_i\}$

$$P_i=[x_{i-n}x_{i-n+1}\cdots x_i] \qquad ..3.60$$

The first n components of P_i are the same as the n components of the corresponding Q_{i-1}

from which it was determined, and the last components $\{x_i\}$ of the two vectors $\{P_i\}$ take on two of their m possible values for which the cost $\{C_i\}$ of the expanded vectors $\{P_i\}$ are smallest. From equation 3.57, these costs are calculated using

$$C_i = C_{i-1} + |r_i - \sum_{h=0}^{g} x_{i-h} y_h|^2$$

$$= C_{i-1} + |r_i - f_i - x_i|^2 \qquad ..3.61$$

where

$$f_i = \sum_{h=1}^{g} x_{i-h} y_h \qquad ..3.62$$

and the value of y_0 is assumed to be equal to unity by suitably scaling the sampled impulse, Y, response of the channel. Clearly the two vectors $\{P_i\}$ which have been derived from one of the vectors Q_{i-1}, have same value of C_{i-1} but different values of $|r_i - f_i - x_i|^2$.

To determine the value of x_i for which $|r_i - f_i - x_i|^2$, and therefore C_i, is smallest a very simple threshold comparison technique can be used. Let

$$d_i = r_i - f_i - x_i \qquad ..3.68$$

minimising both the real and imaginary parts of d_i simultaneously will give the minimum value of $|r_i - f_i - x_i|^2$. The detector first computes the real and imaginary parts of the quantity $(r_i - f_i)$, then by means of a simple threshold comparison, it determines the value of x_i for which $|d_i|^2$, and hence C_i is minimum. The detector now selects a second value of x_i that will give the next smallest value for $|r_i - f_i - x_i|^2$, and hence the next smallest cost. It achieves this by first determining whether the real or imaginary part of d_i has

the greater magnitude, the second selected value of x_i is now found by changing the real or imaginary part, respectively, of x_i to the next adjacent possible value, in the direction given by the sign of the corresponding real or imaginary part of d_i. If there is no possible value of x_i in the location given by this selection process (i.e. when the real or imaginary part of x_i is already at its most positive or negative value) then, instead, the detector carries out the process using the real or imaginary part of d_i that has the smaller magnitude. If, again, the selection process fails to yield a valid value of x_i, the process is repeated for the real or imaginary part of d_i having the smaller magnitude, but this time the corresponding real or imaginary part of x_i is incremented in the direction opposite to that of the sign of the real or imaginary part of d_i. The value of x_i found using this procedure will always give the second smallest value of $|r_i-f_i-x_i|^2$ and hence the second smallest cost, C_i.

Once all k stored vectors $\{Q_{i-1}\}$ have been expanded this way the detector has in store 2k expanded vectors along with 2k costs. It then selects from these the vector with the smallest cost to give the detected data symbol x_{i-n}. The detector then follows the same procedure as described for detector A to select the next (k-1) vectors and their associated costs to give the new set of k vectors $\{Q_i\}$ and their associated costs $\{C_i\}$ ready for the detection of the next data symbol x_{i-n+1}.

3.3.3 Near Maximum Likelihood Detection with a Desired Sampled Impulse Response

In equation 3.6 it is shown that the received signal is

$$r_i = s_i y_0 + \sum_{h=1}^{g} s_{i-h} y_h + w_i$$

where $s_i y_o$ is the wanted component and $\Sigma s_{i-h} y_h$ is the intersymbol interference. Now,

the selection of the k maximum likelihood estimates of the transmitted sequence, involves the computation of the costs of each of the expanded vectors $\{P_i\}$, using

$$C_i = C_{i-1} + |r_i - \sum_{h=0}^{g} x_{i-h} y_h|^2 \qquad ..3.64$$

If, however, the first few values of the sampled impulse response of the channel have a small magnitude, and the sample with the largest magnitude is y_f, where $0 \leq f \leq g$, then the value that x_i takes on during the expansion of the vectors $\{Q_{i-1}\}$ will have little effect upon the costs of the expanded vectors $\{P_i\}$. This means that the vectors are selected almost arbitrarily. To prevents this occurring the detection of the signal element x_{i-n} is delayed by f sampling intervals [22]. The problem with this method is that the value of f must be determined and, if the channel varies considerably with time, the value of f must be continually updated. Also since the complexity of the near maximum likelihood detector increases as the number of components in the sampled impulse response of the channel with 'significant' magnitude (g+1) increases it would be desirable to modify the sampled impulse response of the channel such that there are fewer components and such that its first component has a significant magnitude relative to the other component magnitudes.

To obtain this 'desired impulse response' a filter (known as the pre-filter) is placed ahead of the detector, as shown in figure 3.6, such that the sampled impulse response of the channel and filter is minimum phase [6,27,28,30]. A minimum phase sampled impulse response has the important property that most of its energy is concentrated towards its first few components, with the later components rapidly decaying in magnitude, therefore the magnitude of its first component should be relatively large. Figure 3.5 shows an example of a minimum phase impulse response. It turns out that the linear filter needed to make the sampled impulse response of the channel minimum phase is identical to the linear filter in the optimum decision feedback filter.

Figure 3.1 Model of Digital Data Transmission System

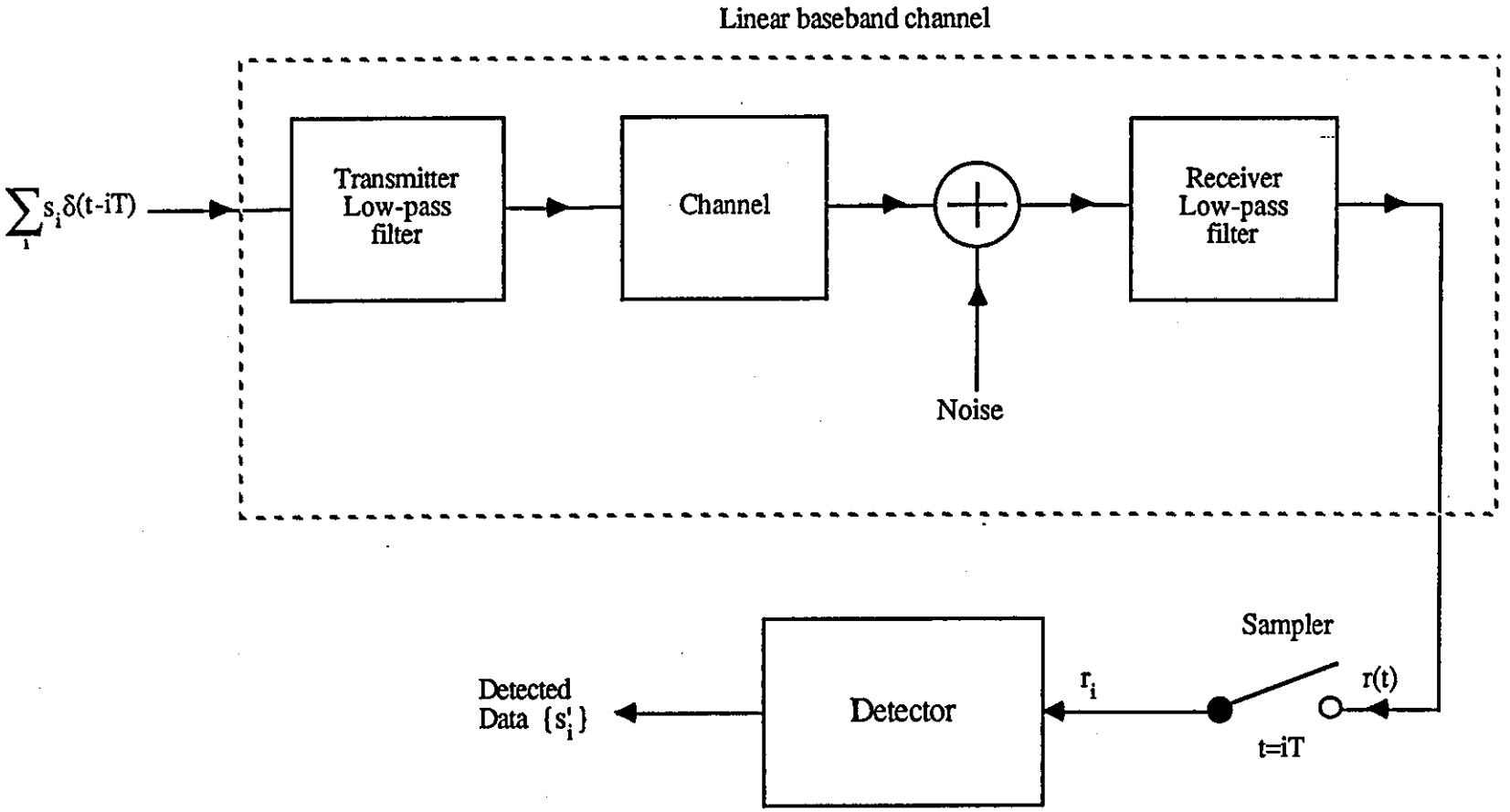


Figure 3.2 Basic Structure of the Decision Feedback Equaliser

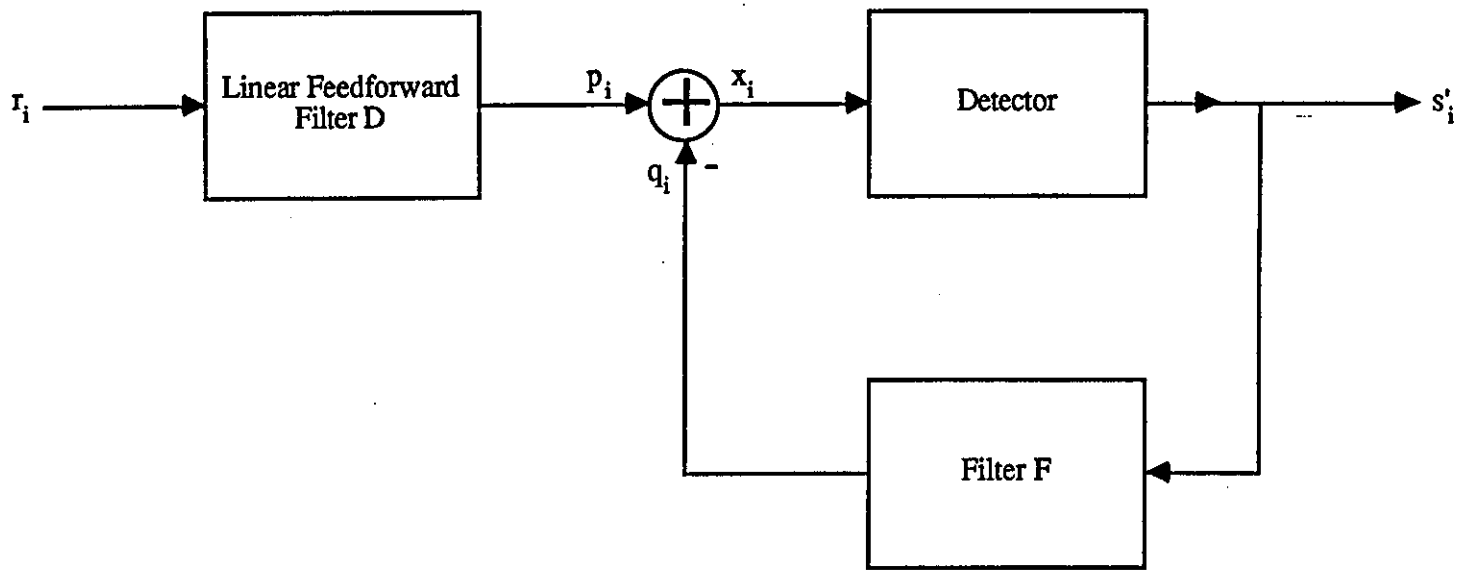


Figure 3.3 Detailed Structure of the Decision Feedback Equaliser - Linear Filter Section (Filter D)

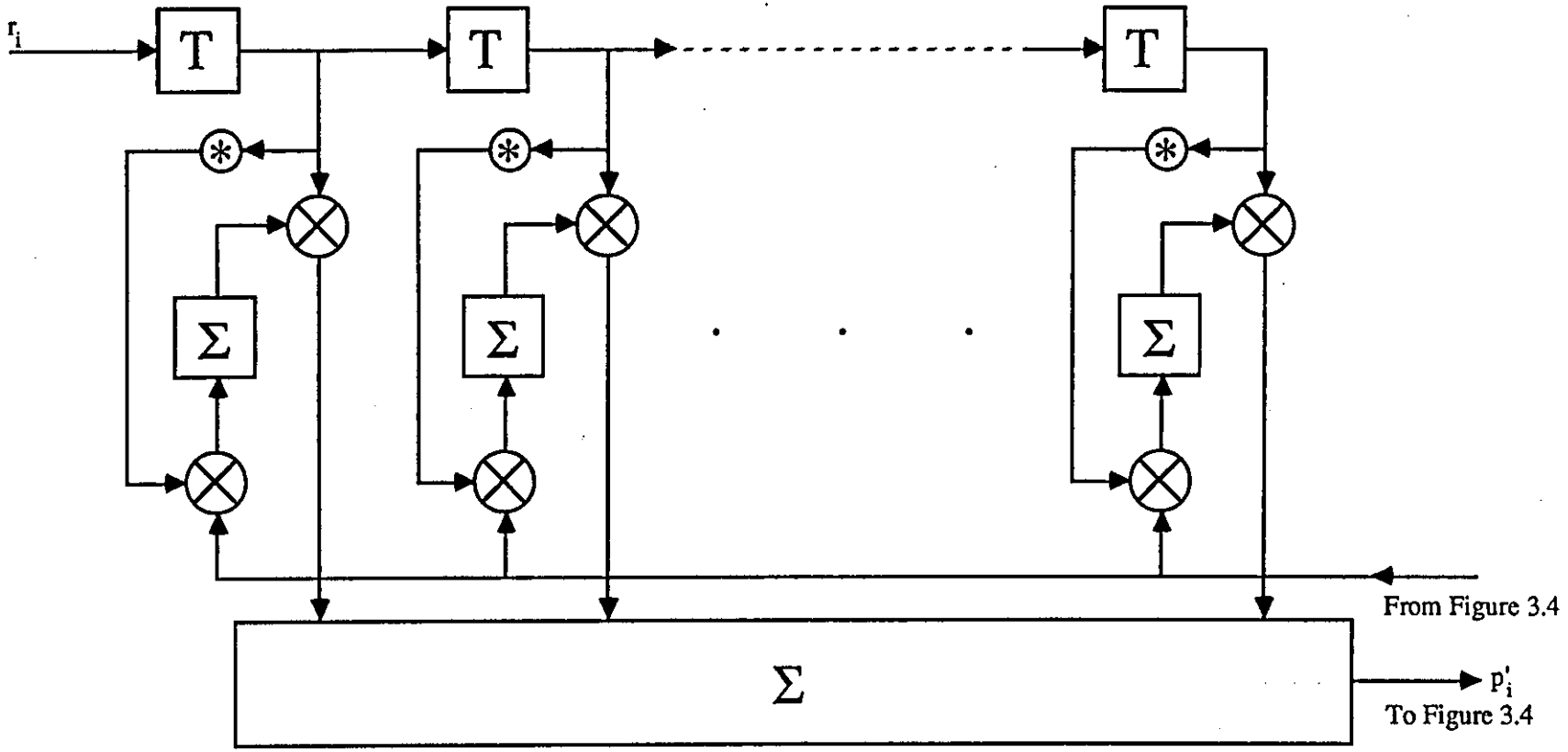


Figure 3.4 Decision Feedback Equaliser - Non-Linear Section

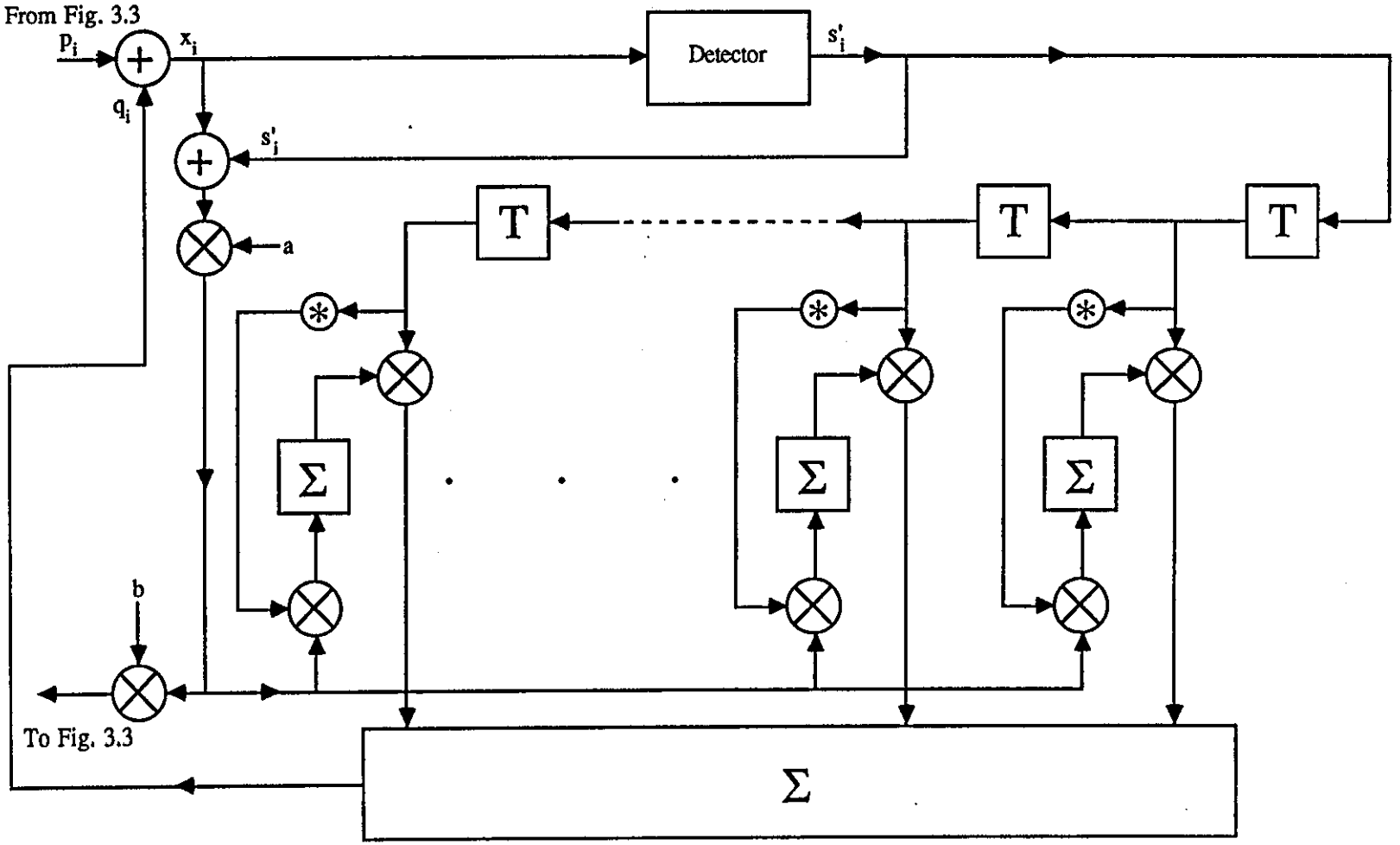


Figure 3.5 Example of a Non-Minimum and Minimum Phase Sampled Impulse Response

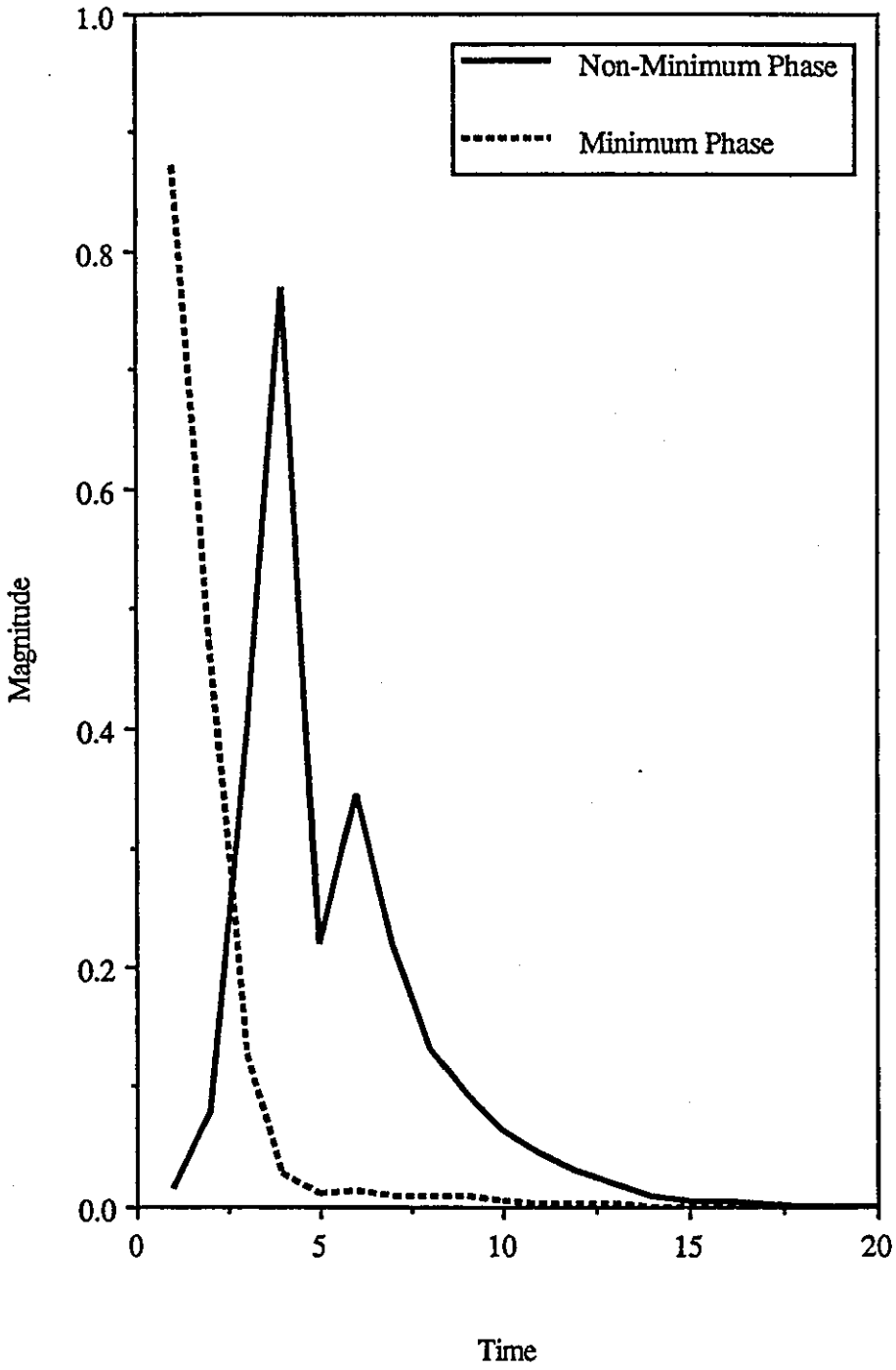
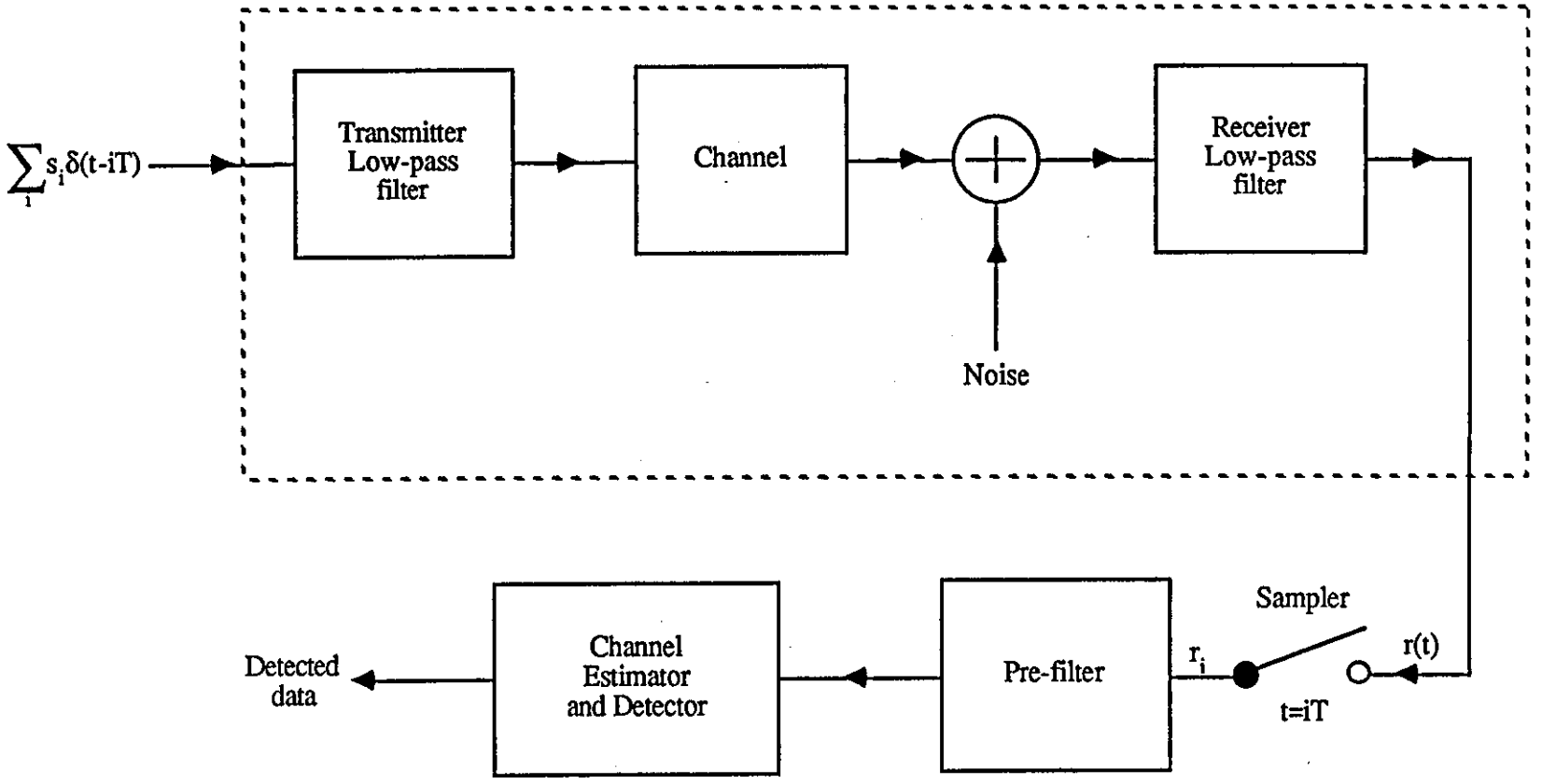


Figure 3.6 Model of Data Transmission System with Desired Sampled Impulse Response



Chapter 4
Transmission of Digital Data over a Telephone Channel

4.1 Introduction

Telephone channels are now widely used as a medium for transmitting digital data. There is an extensive World-wide telephone network, and hence, the use of telephone channels is an obvious choice as a means of communicating data. However, there are disadvantages in using the telephone network, these include the cost per unit time of using the network, especially over a satellite link. The cost can be reduced by increasing the symbol rate, unfortunately, the telephone channel has a limited bandwidth, the attenuation ideally being zero in the frequency range 200 to 3200 Hz. The bandwidth of the channel therefore limits the maximum transmission rate to no more than 3000 symbols per second, which is approximately equal to the Nyquist rate of the channel [2,3]. Increasing the number of possible values the data symbols can have is another way of increasing the bit rate, but this has the disadvantage of reducing the tolerance of the system to additive noise since, for a given signal power, the distance between neighbouring symbols will decrease. Typically, data symbols with between 16 and 256 levels are used for transmission of digital data over a telephone channel, i.e. between 4 and 8 bits are encoded to produce one data symbol.

For a channel, whose bandwidth is limited to 0 to 2W Hertz, the Nyquist rate is 2W symbols per second. Shannon has shown that when the channel introduces no distortion in the frequency range -W to W Hertz, and when the data is fed into it at a rate of 2W signal elements per second from a Gaussian data source, the maximum number of bits per second that can be transmitted in the presence of white Gaussian noise is [4]

$$C = W\log_2(1+SNR) \quad \text{bits/s} \qquad ..4.1$$

where SNR is the signal-to-noise ratio. For example, when W=3000 Hz and

SNR=30 dB, the maximum bit rate is 29900 bits/s. When a non ideal channel is used (i.e. one that has some distortion in the frequency band 200 Hz to 3200 Hz) and when the data source has a uniform probability density function this maximum data rate drops to 19700 bits/s. The highest standard rate in use for data transmission over the public switched telephone network is 19.2 Kbits/s

There are two distinct types of noise [2] that are picked up by a signal sent over a telephone circuit, these are additive noise and multiplicative noise. Additive noise is, as its name suggests, a random waveform added to the signal, and, since it is added to the signal the error rate can be reduced by increasing the power of the transmitted signal which increases the distance between adjacent symbols. When the signal is multiplied (modulated) by a random process then this is termed multiplicative noise, increasing the signal power, will of course, have no effect upon the performance.

In this work it will be assumed that only additive white Gaussian noise is present in the received signal. It has been shown that the tolerance of the system to this type of noise gives a good idea of the performance of the same system to other types on noise that may be present in an actual telephone channel [2].

The telephone channels found in practice do not have ideal characteristics, they introduce severe levels of distortion within the frequency band 200 Hz to 3200 Hz, and hence, their bandwidth is effectively reduced. For reliable transmission at rates of greater than 9.6 Kbits/s over such channels, the receiver must take into account the distortion introduced by the channel. For the lower data rates, a decision feedback equaliser (DFE) gives a good performance. At higher data rates the DFE gives a poor performance, and more complex detection processes are needed for reliable operation.

The Viterbi algorithm detector (described in chapter 3) will, when the noise samples at its input are Gaussian and the data symbols are statistically independent and equally likely to have one of their m possible values, give an optimum performance. Under these assumed conditions, it minimises the probability of error and is therefore known as a maximum likelihood detector. The Viterbi algorithm is, however, too complex to be used in any real time applications. This excessive complexity has led to many derivations of the basic Viterbi algorithm, these use far less storage and are much less

complex, but can give a performance that comes close to the optimum detector. These simpler algorithms are called near maximum likelihood detectors, and come much closer to a practical solution for the detection problem.

The performance of near-maximum likelihood detectors can be seriously affected by phase distortion introduced by the channel into the transmitted signal, this is because the first few components of the sampled impulse response of the baseband channel will have a small magnitude [27]. By placing an adaptive linear feedforward filter (known as the pre-filter) ahead of the detector, the performance of the near maximum likelihood detector can be greatly improved. The linear filter ahead of the detector converts the sampled impulse response of the channel into a minimum phase sequence, without introducing any amplitude distortion into the signal. A minimum phase sampled impulse response has the very important property that most of its energy is concentrated towards its first few components, and therefore its first few components [30] will have a significant magnitude relative to latter components.

The characteristics of the channel are likely to be varying from call to call or continuously with time and must therefore be adaptive in nature. An algorithm has been developed that will adaptively adjust the pre-filter taps, such that the channel becomes minimum phase [27,29]. It uses a recursive technique similar to the Newton-Raphson method and has been shown to provide a cost effective means of adjusting the pre-filter.

Much work has been carried out to show the performance of a wide range of near maximum likelihood detectors, operating with the pre-filter, when data is transmitted over a telephone channel. However, all this work has been carried out by computer simulation using a very high degree of arithmetical accuracy. In a practical modem, that is intended to be used for transmission of digital data over a telephone channel, such a high degree of precision is, at the present time, unlikely to be available due to the present state of microprocessor technology. More likely, a much lower level of arithmetical precision would be used, such as that obtained using the latest 16-bit digital signal processors [1,39-45,48-50].

The aim of this chapter is, therefore, to show the effects of reducing the level of

arithmetical precision used to run the various algorithms in the modem receiver. Specifically, the detector, the channel estimator, and the algorithm used to adjust the taps of the pre-filter, are examined. Computer simulation tests have been used throughout this work, in these tests the algorithms just mentioned, are run using integer arithmetic instead of high precision floating point arithmetic.

4.2 Model of System

Figure 4.1 shows a model of the baseband system incorporating a telephone circuit. Filters A and B represent the overall filtering carried out at the transmitter and receiver respectively. White Gaussian noise is added to the signal at the output of the telephone circuit before it is fed into filter B.

In the model, the transmitted signal consists of a stream of impulses which are modulated by the values of the data symbols $\{s_i\}$. The impulses are a mathematical tool, used to make the analysis of the model easier. In reality a stream of square pulses would be used and the transfer function of filter A appropriately modified with sinx/x correction.

In a practical digital modem, all operations would be carried out using digital techniques. Digital filters would be used to shape the transmitted pulses at the transmitter and at the receiver they would be used to remove noise [36-48]. The quadrature amplitude modulation/demodulation processes would also be carried out digitally using sine and cosine look-up tables to generate the carriers [36-45]. After the quadrature amplitude modulated signal has been generated (digitally) it is then conveted to an analogue signal using a D/A converter. At the receiver, an A/D converter would be used to interface the output of the channel with the modem input, the rest of the signal processing would then be carried out in the digital domain.

To achieve carrier phase synchronisation at the receiver, a digital phase locked loop [51] could be used [2,16,36-38,40-45,48,52,69]. The symbol timing recovery would also be recovered using a digital phase locked loop which would be adjusted such that the demodulated symbols are sampled at the optimum points in time.

At the start of transmission an initial sequence of data symbols, known to the receiver, is sent so that the channel estimator can converge to an accurate estimate of the sampled impulse response of the linear baseband channel. If the channel characteristics don't change for the duration of the call then after the startup sequence has been sent the estimator can be kept fixed for the rest of the call.

4.3 Limited Precision Arithmetic

As mentioned earlier, a large amount of work has been carried out, using computer simulation, to show the performance of various detection algorithms in the pres ence of intersymbol interference and additive white Gaussian noise. These simulations have all been carried out using a high degree of numerical precision (64 or 128-bit floating point arithmetic). However, with the present state of microprocessor technology, a practical modem is likely to be implemented using a lower arithmetical precision with, say, 16-bit integers. In this work it is assumed that a high speed microprocessor, such as the Texas Instruments TMS320C25 digital signal processor, is used.

The TMS320C25 has been designed specifically for the high speed processing of digital signals [54]. It has two arithmetic and logic units, a hardware multiplier, a timer, internal random access memory (RAM), several auxillary registers, a 32-bit accumulator and control signals for communication with slower external devices and multiprocessing applications.

Of the two arithmetic and logic units, the Central Arithmetic and Logic Unit (CALU) is used to carry out most numerical operations. It supports two's complement integer arithmetic, has a 32-bit accumulator into which the results of all numerical operations can be stored. An overflow protection mode is provided so that if the result of a numerical operation has a magnitude that is too large to be stored into the accumulator, then the later is loaded with the most positive or negative number it can hold. The contents of the accumulator can be stored in memory via a shift register which can be used to scaling. Another important feature of the CALU is the hardware multiplier. This will multiply two 16-bit two's complement numbers to give a 32-bit result, which is held in a temporary register known as the P-register. The contents of the P-register

can either be stored via a shift register into memory, or, it can be added to, or subtracted from the contents of the accumulator.

The auxillary registers may be used for temporary storage or for indirect addressing. The contents of any one of the auxillary registers can be manipulated (i.e. incremented, decremented, have a constant added or have logical operations carried out on them) using the Auxillary Register Arithmetic and Logic Unit (ARALU). The ARALU can operate independently from the CALU, thus freeing the later from the the tasks just mentioned above and therefore increasing throughput.

The built-in high speed memory can be used as either data memory (for frequently used variables), or, it can be configured as high speed program memory. Most of the TMS320C25 instructions can be pipelined and when programs are run from this internal memory maximum running speeds are obtained.

With the various features of the TMS320C25 described above in mind, computer simulation programs were written in C to simulate the transmission of data over voice-band channels. In the simulations, the actual transmission of the data over various channels was simulated using 64-bit floating point arithmetic. The received complex valued analogue signal (represented by 64-bit floating variables) was converted into a digital signal by, first scaling it, and then converting its real and imaginary parts into 16-bit integers.

In one of the algorithms used in the receiver, that will be described later, a complex valued division has to be carried out. This can be split up into two real divisions by rationalisation. Unfortunately, the TMS320C25 doesn't have a hardware divider and so, division has to be performed in software. Integer division can be performed using repeated subtraction by making use of the assembly language command 'SUBC'. A simple integer division program, that will run on the TMS320C25, will now be described. It assumes that the both the numerator and the denominator are positive 16-bit integers and that the numerator is greater than the denominator. The complete division process takes fifteen machine cycles to perform, excluding any extra operations needed to check for divide by zero, calculation of the sign of the division and a check to see if the numerator is greater than the denominator.

The high byte of the accumulator is first loaded with the numerator and the low byte is zeroed. The denominator is then multiplied by 2^{15} and conditionally subtracted from the contents of the accumulator (using the SUBC command). If the subtraction gives a negative result then the contents of the accumulator is left as it was before the subtraction and then shifted one place to the left with the LSB filled with a '0'. If the result of the subtraction is positive then the accumulator is loaded with the difference, it is then shifted to the left one place and a '1' loaded into the LSB. This subtraction process is repeated fifteen times and, after the last subtraction, the low byte of the accumulator will hold the result of the division scaled up by 2^{15} and its high byte will hold the remainder scaled up by 2^{15}.

4.4 Telephone Channels used in the Computer Simulations

The sampled impulse responses of eight telephone channels were used in the computer simulations. Tables 4.1 and 4.2 give the sampled impulse responses of all eight channels, where each channel consists of the actual bandpass telephone circuit and the equipment filters (the combined filters A and B in figure 4.1). The impulse responses of channels 1 to 4 have been sampled at a rate of 2400 samples/s. The telephone circuits used in channels 5 to 8 are identical to those used in channels 1 to 4, but a sampling rate of 3200 samples/s has been used. Figures 4.2 to 4.5 show the attenuation and group delay characteristics of the four telephone circuits used, and figure 4.6 shows the attenuation and group delay characteristics of the combined equipment filters [9]. The amount of distortion introduced by the telephone circuits increases from circuits 1 to 4. Telephone circuit 1 and telephone circuit 2 introduce negligible and typical levels of distortion, respectively, whereas telephone circuits 3 and 4 are close to the typical worst circuits normally considered for the transmission of data at 9600 bits/s. Telephone circuit 3 introduces severe group delay distortion as well as considerable attenuation distortion, and telephone circuit 4 introduces very severe attenuation distortion [55].

4.5 Channel Estimation

The near maximum likelihood detector needs to have accurate knowledge of the sampled impulse response of the linear baseband channel if it is to operate correctly. Let the sampled impulse of the channel be represented by the (g+1)-component row vector Y

$$Y=[y_0 y_1 \ldots y_g] \qquad ..4.2$$

For telephone channels, the impulse response varies only very slowly with time, but the channel characteristics may differ considerably from one call to another. The sampled impulse response of the channel may, therefore, be continuously estimated, or if it doesn't change for the duration of the call, it could be estimated at the start of transmission, the estimated components then remain fixed for the rest of the call.

The simplest of all channel estimators is based on an adaptive linear feedforward transversal filter [5,56,57] that models the baseband sampled impulse response Y. The structure of the linear filter is shown in figure 4.7. The filter is adaptive in nature, and makes use of the received samples $\{r_i\}$ and the detected data $\{s_i'\}$ in such a way as to minimise the mean square error between the received samples $\{r_i\}$ and the corresponding estimates of the received samples $\{r_i'\}$. The tap weights are adjusted using the steepest descents method. It is assumed that the data symbols $\{s_i'\}$ are detected correctly (which will be the case at high signal-to-noise ratios), and therefore, $s_i'=s_i$ for all i. Each of the squares marked T in figure 4.7 represent a store that introduces a delay of one sampling period equal to T seconds. At time t=iT the detector gives the detected data symbol s_i', which is fed into the input of the linear filter. Each of the stored values, $\{s_{i-h}'\}$, is multiplied by the corresponding components $\{y_{i-1,h}'\}$, where $y_{i-1,h}'$ is an estimate of the components y_h of the sampled impulse response at time t=(i-1)T to give r_i'

$$r_i' = \sum_{h=0}^{g} s_{i-h}' \; y_{i-1,h}' \qquad ..4.3$$

Now, r_i' is compared with the actual received sample r_i to give an error e_i,

$$e_i = r_i - r_i' \qquad ..4.4$$

which is multiplied by a small positive constant Δ. The resulting signal Δe_i is then multiplied by $(s_{i-h}')^*$, the complex conjugate of s_{i-h}', and the product added to the corresponding components of Y_{i-1}' (the estimate of Y at t=(i-1)T) to give the new stored estimate of the impulse response Y_i'. Thus the $(h+1)^{th}$ component of Y_i' is given by

$$y_{i,h}' = y_{i-1,h}' + \Delta e_i (s_{i-h}')^* \qquad ..4.5$$

for h=0,1, . . .,g. The smaller the value of Δ, the smaller the effects of additive noise on Y_i', but the slower the rate of response of Y_i' to changes in Y_i.

Initially, the vector Y_0' can be set to zero, and a sequence of data symbols known to the receiver, transmitted such that the estimator converges to an accurate estimate of the sampled impulse response before any actual data is sent.

4.5.1 Computer Simulation Tests

Computer simulation tests have been carried out to show the performance of the channel estimator when all of the variables (Δ, $\{r_i'\}$, $\{y_{i,h}'\}$ and r_i) were represented in discrete form by n-bit integers. At the start of each test a known sequence of data symbols was transmitted to allow the estimator to converge to an accurate estimate. The accuracy of the estimate was measured by comparing the estimate with the actual sampled impulse response of the channel. Equation 4.6 gives the difference (expressed in decibels) between the estimate and the actual values of the components of the

sampled impulse response of the channel

$$\text{Error (dB)} = 10\log_{10}\left(\sum_{h=0}^{g}\left|y_h^{(est)} - y_h^{(actual)}\right|^2\right) \qquad ..4.6$$

where $\{y_h^{(est)}\}$ are the components of the estimate of the sampled impulse response and $\{y_h^{(actual)}\}$ are the actual components of the sampled impulse response of the linear baseband channel (given in tables 4.1 and 4.2).

Figure 4.8 gives an example of the error in the estimate (found using equation 4.6) versus the number of data symbols transmitted (iterations) over telephone channel 1. A 64-level QAM signal was used at a baud rate of 2400 and a signal-to-noise ratio of 30 dB. The performance of the estimator, when less than 16-bit integer arithmetic was used, is also shown to see if a dedicated channel estimator integrated circuit could be fabricated using a lower arithmetical precision. For a comparison, the performance of the estimator when 64-bit floating point arithmetic is used, is shown as well. It can be seen that for a SNR of 30 dB the difference in performance between the 16-bit integer and 64-bit floating point versions of the channel estimator is small. The plots showing the error in the estimate for channels 2 to 8 are all very similar and have therefore not been included.

Figures 4.9 to 4.12 show plots of the average error in the estimate of the sampled impulse response, after the estimator has been allowed to converged, against the SNR. The average error was found by allowing the estimator to first converge to a steady estimate and then measuring the average value of equation 4.6 over 5000 sampling instants. In figures 4.9 and 4.10 the performance of the estimator is shown where a 16-level QAM signal was sent over channels 1 and 4, respectively. Figures 4.11 and 4.12 show the performance when a 64-level QAM signal was used. Comparing all four figures, it can be seen that the number of signal levels and the severity of the distortion introduced by the channel has little effect upon the estimators performance. For a signal-to-noise ratio, typically found over telephone channels, of 30 dB, figures 4.9 to 4.12 all show that there is little difference in the accuracy of the estimate when 16-bit integer or when 64-bit floating point arithmetic is used. In figures 4.9 to 4.12 it

can be noticed that the accuracy of the estimate ceases to improve with the signal-to-noise ratio when its value is high enough, in fact, at a signal-to-noise ratio of 50 dB, there seems to be a difference of 12 dB in the accuracy of the estimate when the number of bits used is reduced from 16 to 14 and from 14 to 12, etc. This can be explained as follows.

If the largest amplitude in the received signal was, say, A and n-bit integers were used to represent the received samples $\{r_i\}$, then the quantisation noise power in the samples $\{r_i\}$ would be

$$P_{n1}=20\log_{10}\left(\frac{A}{2x2^{n}}\right) \qquad ..4.7$$

and if (n-2)-bit integers were used, then the average quantisation noise power in the received samples $\{r_i\}$ would be

$$P_{n2}=20\log_{10}\left(\frac{A}{2x2^{n-2}}\right) \qquad ..4.8$$

The increase in quantisation noise power due to the reduced level of accuracy introduced by using 2 bits less to represent the variables in the channel estimator will be

$$P_{n}=20\log_{10}\left(\frac{A}{2x2^{n-2}}\right)-20\log_{10}\left(\frac{A}{2x2^{n}}\right)$$

$$=20\log_{10}4$$

$$=12 \text{ dB} \qquad ..4.9$$

In figures 4.9 to 4.12, the error in the estimate stops decreasing with the signal-to-noise

ratio because the level of quantization noise 'drowns' out the effects of the additive white Gaussian noise.

4.6 Adaptive Adjustment of the Pre-Filter for a Time Invariant Channel

The performance of the near maximum likelihood detector can be improved by placing a linear feedforward filter, known as the pre-filter ahead of it, such that the sampled impulse response of the channel in cascade with the linear filter is minimum phase [27,29,30] without, however, introducing any amplitude distortion. Since the characteristics of the telephone circuit are likely to change considerably from call to call, the pre-filter tap weights must adapt to these changes, and therefore, the adjustment of the pre-filter depends upon an accurate estimate of the channel being given by the channel estimator.

Let the estimate of the channel sampled impulse response be represented by the (g+1)-component row vector Y,

$$Y=[y_0 y_1 \ldots y_g] \qquad ..4.10$$

which has a z-transform given by

$$Y(z)=y_0+y_1z^{-1}+\ldots+y_gz^{-g} \qquad ..4.11$$

If

$$Y(z)=Y_1(z)Y_2(z) \qquad ..4.12$$

where

$$Y_1(z)=\eta(1+\alpha_1z^{-1})(1+\alpha_2z^{-1})\ldots(1+\alpha_{g-m}z^{-1}) \qquad ..4.13$$

and

$$Y_2(z)=z^{-m}(1+\beta_1 z)(1+\beta_2 z)\ldots(1+\beta_m z) \qquad ..4.14$$

$Y_1(z)$ and $Y_2(z)$ are complex polynomials with roots (zeros) inside and outside the unit circle in the complex z-plane respectively, also $|\alpha_i|<1$ and $|\beta_i|<1$, where α_i is negative of a root of Y(z) and β_i is the negative of the reciprocal of one of the roots of Y(z). The quantity η is a complex valued constant needed such that equations 4.11 and 4.12 are satisfied, and m is the number of roots of Y(z) that lie outside the unit circle in the complex z-plane. It is assumed that no roots lie on the unit circle. The adaptive filter D in figure 4.1 has (q+1) taps, where q is typically between 30 and 50.

To make the channel in cascade with the pre-filter minimum phase, the roots of Y(z) that lie outside the unit circle ($-1/\beta_i$) must be replaced by the reciprocal of their conjugate values, as shown in the example in figure 4.13 [47]. Thus the z-Transform of the sampled impulse response of the filter D is now approximately

$$D(z)=z^{-q}Y_2^{-1}(z)Y_3(z)$$

$$=d_0+d_1z^{-1}+\ldots+d_qz^{-q} \qquad ..4.15$$

where

$$Y_3(z)=(1+\beta_1^*z^{-1})(1+\beta_2^*z^{-1})\ldots(1+\beta_m^*z^{-1}) \qquad ..4.16$$

and β_i^* is the complex conjugate of β_i. The z-transform of the channel and linear filter in cascade will approximately be

$$F(z)=Y(z)D(z)$$

$$=z^{-q}Y_1(z)Y_3(z) \qquad ..4.17$$

Equations 4.15 and 4.17 are only approximate because the polynomial $Y_2^{-1}(z)Y_3(z)$ has positive powers of z and would therefore represent a non-causal system, multiplying by z^{-q} makes filter D realisable. The value of q should, ideally, be be infinite (introducing an infinite delay), but a very good approximation is usually obtained without an unduly large value of q. All the roots of the polynomial F(z) lie inside the unit circle in the complex z-plane, which means that the channel and linear filter together have an impulse response that is minimum phase. The roots of $Y_3(z)$ are the complex conjugates of the reciprocals of the roots of $Y_2(z)$.

The determination of the roots of Y(z) that lie outside the unit circle, with no restrictions in time or complexity, could be achieved by any conventional root finding algorithm. The speed of present day microprocessors does, however, require that the root finding algorithm be as fast as possible and, at the same time, accurate such that it may be used in real time applications. One such algorithm, which is suited to real time applications, is the Clark-Hau algorithm [27,29]. This algorithm uses an iterative technique (similar to the Newton-Raphson method) to locate the roots of Y(z) that lie outside the unit circle and, at the same time, finds the tap weight of the pre-filter D and the components of the minimum phase sampled impulse response that are needed by the near maximum likelihood detector.

4.6.1 The Clark-Hau Algorithm [27,29]

The adjustment of the adaptive pre-filter proceeds as follows. The algorithm first forms a filter, with z-transform

$$A_i(z)=\frac{1}{1+\lambda_i z} \qquad ..4.18$$

for i=0,1, . . . ,k in turn, where λ_i is an estimate of one of the roots of Y(z) that lie outside the unit circle in the z-plane. The sequence Y is now fed into this filter, and an iterative process used to adjust λ_i such that $\lambda_i \rightarrow \beta_i$, where β_i is the negative of the reciprocal of the first root of Y(z) to be processed by the algorithm and, of course, $|\beta_i|<1$. The filter with z-transform $A_i(z)$ is not realisable, therefore it is impossible to operate on a signal in real time. To achieve the effect of passing the sequence Y through a filter with z-transform $A_i(z)$, the sequence Y is first reversed in order, starting with y_g at time t=0 and ending with y_0 at time t=gT, to give the sequence

$$Y^R=[y_g y_{g-1} \cdots y_0] \qquad ..4.19$$

which is fed into a filter with z-transform

$$A_i'(z)=\frac{1}{1+\lambda_i z^{-1}} \qquad ..4.20$$

which can be realised as a one tap feedback filter, as shown in figure 4.14. The reversed sequence Y^R, will appear to be moving backwards in time, starting with y_g at time t=0. Thus, a delay of one sampling interval, T, in the feedback filter becomes an advance of T seconds, with z-transform z. The effective z-transform of the filter in figure 4.14, when used in the way described, will be given by equation 4.20 and the output of the filter will be the sequence $\{e_{i,h}\}$, for h=0,1, . . . ,g which has a z-transform $Y(z)A_i(z)$, given by

$$Y(z)A_i(z)=e_{i,0}+e_{i,1}z^{-1}+\ldots+e_{i,g}z^{-g} \qquad ..4.21$$

If Y(z) is factorised such that

$$Y(z)=(1+\beta_1 z)(u_0 z^{-1}+u_1 z^{-2}+\ldots+u_{g-1}z^{-g}) \qquad ..4.22$$

then it can be shown [27]

$$e_{i,0}=(\beta_1-\lambda_i)(u_0-u_1\lambda_i+u_2\lambda_i^2-\ldots+u_{g-1}(-\lambda_i)^{g-1}) \qquad ..4.23$$

If λ_i is close to β_1 then $e_{i,h}\approx u_{h-1}$, for h=1,2, . . . ,g and so equation 4.23 becomes [27]

$$e_{i,0}\approx(\beta_1-\lambda_i)(e_{i,1}-e_{i,2}\lambda_i+e_{i,3}\lambda_i^2-\ldots+e_{i,g}(-\lambda_i)^{g-1})$$

$$\approx(\beta_1-\lambda_i)\varepsilon_i\approx 0 \qquad ..4.24$$

where

$$\varepsilon_i=e_{i,1}-e_{i,2}\lambda_i+\ldots+e_{i,g}(-\lambda_i)^{g-1} \qquad ..4.25$$

From 4.24 it can be seen that

$$\beta_1\approx\lambda_i+\frac{e_{i,0}}{\varepsilon_i} \qquad ..4.26$$

which suggests the following algorithm for finding β_1,

$$\lambda_{i+1}=\lambda_i+\frac{ce_{i,0}}{\varepsilon_i} \qquad ..4.27$$

where λ_{i+1} is an improved estimate of β_1 and c is a small positive constant in the range 0 to 1. As $\lambda_i\rightarrow\beta_1$, $e_{i,0}\rightarrow 0$ which suggests the following stopping criterion for convergence [27]

$$\left|\frac{e_{i,0}}{\varepsilon_i}\right|^2 < d \qquad ..4.28$$

where d is some suitably small threshold, say 10^{-7} [27]. At the end of the iterative process, when 4.28 is satisfied for, say when i=k, the z-transform of the one tap feedback filter is

$$A_k(z)=\frac{1}{1+\beta_1 z} \qquad ..4.29$$

and one of the roots of Y(z) that lies outside the unit circle in the z-plane will have been found. Since $|\beta_1|<1$, the search for the values of β_i is limited to values that lie within the unit circle with $|\lambda_i|<1$.

The receiver now forms a two tap linear feedforward filter, as shown in figure 4.15, with z-transform

$$B_k(z)=1+\lambda_k^* z^{-1}$$

$$\approx 1+\beta_1^* z^{-1} \qquad ..4.30$$

The output of the one tap feedback filter, $\{e_{i,h}\}$, for h=0,1, . . . ,g, is now fed into the feedforward filter, in the correct order, to give the (g+2)-component sequence, with z-transform

$$F'(z)=f_{1,-1}+f_{1,0}z^{-1}+f_{1,1}z^{-2}+\ldots+f_{1,g}z^{-g-1} \qquad ..4.31$$

which is approximately equal to $Y(z)A_k(z)B_k(z)$ and where $f_{1,-1}$ is approximately equal to zero. The overall effect of passing the sequence Y through the one tap feedback filter A, and the two tap feedforward filter B, is equivalent to passing it through a single filter

with z-transform

$$C_1(z)=A_k(z)B_k(z)$$

$$=\frac{1+\beta_1^* z^{-1}}{1+\beta_1 z} \qquad ..4.32$$

Finally, the sequence at the output of filter B is advanced one place to the right, which is equivalent to multiplying it by z, and the first component $f_{1,-1}$ discarded to give a (g+1)-component sequence F, with z-transform

$$F_1(z)=f_{1,0}+f_{1,1}z^{-1}+\ldots+f_{1,g}z^{-g}$$

$$=zC_1(z)Y(z) \qquad ..4.33$$

Thus the linear factor $(1+\beta_1 z)$ in equation 4.14 is replaced by the factor $(1+\beta_1^* z^{-1})$, the first factor in equation 4.35, which effectively replaces the root of Y(z) at $-1/\beta_1$ by the root $-\beta_1^*$ which is the complex conjugate of its reciprocal, and lies inside the unit circle in the z-plane. The sequence F_1 (with z-transform $F_1(z)$) is an estimate of the z-transform of the channel and adaptive filter in cascade, the latter having a z-transform

$$D_1(z)=zC_1(z) \qquad ..4.34$$

Initially, before the algorithm is started, all the taps of the pre-filter are set to zero, except the $(q+1)^{th}$ tap, d_q, whose value is set to unity. Thus, the initial z-transform of the filter will be

$$D_0(z)=z^{-q} \qquad ..4.35$$

which is a delayed unit impulse and the initial z-transform of the channel and filter will be $z^{-n}Y(z)$. After the algorithm has converged, to give a value for β_1, the sequence D_0 is reversed in order and fed into the one tap feedback filter A, the first q+1 components at its output are then fed into the two-tap feedforward filter B. The first q+2 components from the output of filter B are then advanced by one place to give the sequence D_1, with z-transform

$$D_1(z)=zC_1(z)D_0(z)$$

$$=z^{-q+1}C_1(z) \qquad ..4.36$$

The coefficients of $D_1(z)$ are the required tap weights of the pre-filter.

The whole of the above procedure is now repeated, but with $F_1(z)$ in place of $Y(z)$, and $D_1(z)$ in place of $D_0(z)$. At the end of the second iterative process, the value of β_2 will have been found, and the values of β_2, the coefficients of $F_1(z)$ and $D_1(z)$ determine $F_2(z)$ and $D_2(z)$ which are then used in place of $F_1(z)$ and $D_1(z)$ for the location of β_3.

When all of the values of β_r, for r=1,2, . . . ,m, have been determined, the z-transform of the adaptive filter, $D_m(z)$, will approximately be

$$D_m(z)=z^{-q+m}C_1(z)C_2(z)\ldots C_m(z)$$

$$=D(z) \qquad ..4.37$$

and the z-transform of the channel and adaptive filter is given by

$$F(z)=Y(z)D(z)$$

$$\approx f_0+f_1z^{-1}+\ldots+f_{q+g}z^{-q-g} \qquad ..4.38$$

where $f_h\approx 0$ for $h=0,1,\ldots,(q-1)$. The estimate of the sampled impulse response of the combined channel and adaptive filter is the sequence F_m, with z-transform

$$F_m(z)=f_{m,0}+f_{m,1}z^{-1}+\ldots+f_{m,g}z^{-g}$$

$$=z^qF(z)$$

$$=Y_1(z)Y_3(z) \qquad ..4.39$$

which is needed by the near maximum likelihood detector. The delay of q sampling intervals introduced by the adaptive filter is, for convenience ignored here, but must obviously be taken into account when comparing $Y(z)D(z)=F(z)$ and $F_m(z)$.

To start the algorithm off, some initial guess of the value of λ_0 in equation 4.18, is needed, figures 4.16 and 4.17 show two possible sets of starting points that were used in the tests [27]. Whilst the algorithm is running, every time λ_{i+1} (equation 4.27) is calculated a check is made to see if $|\lambda_{i+1}|$ is greater than unity, i.e. the algorithm starts to converge towards a root inside the unit circle. If $|\lambda_{i+1}|$ is greater than unity then the algorithm is restarted with a different starting point. Another check that is made whilst the algorithm is running, is to see if the number of iterations (the value of i) exceeds an upper limit, say, when i>40 [27]. If too many iterations are required to locate a root then the algorithm is stopped and another starting point tried. If all of the starting points have been tried and the algorithm still fails to locate a root then it is assumed that all of the roots of Y(z) that lie outside the unit circle in the z-plane have been found. In figure 4.18 a flow diagram of the Clark-Hau algorithm is shown.

4.6.2 Computer Simulation Tests

The performance of the Clark-Hau algorithm can be measured using three parameters, ψ_1, ψ_2 and ψ_3 (expressed in dB), given by

$$\psi_1=10\log_{10}\left(\sum_{h=0}^{q-1}|f_h|^2\right) \qquad ..4.40$$

$$\psi_2=10\log_{10}\left(\sum_{h=0}^{g}|f_{m,h}-f_{h+q}|^2\right) \qquad ..4.41$$

and

$$\psi_3=10\log_{10}\left(\sum_{h=0}^{g}|y_h^{(min)}-y_{m,h}|^2\right) \qquad ..4.42$$

where $\{f_h\}$ and $\{f_{m,h}\}$ are from equations 4.38 and 4.39 respectively and $\{y_h^{(min)}\}$ are the g+1 components of the minimum phase sampled impulse response of the channel found using the NAG library. The quantity, ψ_1,is a measure of how close to zero the first q components of the sampled impulse response of the channel, convolved with the sampled impulse response of the pre-filter are (with a z-transform given by equation 4.38). The value of ψ_2 is a measure of how close the minimum phase sequence with a z-transform given by equation 4.39 is to the last g+1 components of the sequence with

a z-transform given by equation 4.38. The last figure, ψ_3, is a measure of the difference between the minimum phase sequence $\{f_{m,h}\}$, and the actual minimum phase sequence found using the NAG library.

To test the algorithm with limited precision arithmetic, channels 1 to 8 were used, their sampled impulses are given in tables 4.1 and 4.2. In tables 4.3 and 4.4 the minimum phase sampled impulse responses of channels 1 to 8 are given ($\{y_h^{(min)}\}$ in equation 4.42), and in figures 4.19 to 4.26 the magnitudes of the components of the minimum and non-minimum phase sampled impulse responses are shown plotted against time. It can be seen that for channel 4 (figure 4.20), where the first component of the minimum phase sequence is small, a near maximum likelihood detector would be more sensitive to noise because of the small magnitude of the first component relative to the magnitudes of latter components. However, the performance of the detector would still be better than if no pre-filter was used. In figures 4.27 to 4.34 the positions of the roots that lie outside the unit circle in the z-plane are shown for channels 1 to 8. It can be seen that for the channels that introduce severe amplitude distortion, like channel 4, there are several roots lying very close to the unit circle which are the cause of the amplitude distortion [6,47]. The reciprocal of the positions of the roots that lie only just outside the unit circle (the values of $-\beta$) will lie just within the unit circle and, therefore, the Clark-Hau algorithm will be more prone to having to restart with a different starting position because of the greater possibility of $|\lambda_{i+1}|$ (equation 4.27) becoming greater than unity. This will cause the algorithm to take longer to locate the roots lying close to the unit circle because of the increased possibility of it having to try several starting positions before it successfully converges. Reducing the value of c, which controls the rate of convergence and the accuracy of the algorithm, will have the effect of reducing the probability of $|\lambda_{i+1}|$ becoming greater than unity.

For channels 1 to 4 the starting points shown in figure 4.16 were used and with channels 5 to 8 the starting points in figure 4.17 were used [27].

In equation 4.27, a complex valued division has to be carried out which has to be repeated for every iteration of the Clark-Hau algorithm. The complex division can be split up into two real divisions, i.e.

$$\lambda_{i+1}=\lambda_i+\frac{cRe[e_{i,0}\varepsilon_i^*]}{|\varepsilon_i|^2}+\frac{cIm[e_{i,0}\varepsilon_i^*]}{|\varepsilon_i|^2} \qquad ..4.43$$

For the Clark-Hau algorithm to run on a TMS320C25 microprocessor the integer division routine, described in section 4.4, will have to be used. The division routine takes a minimum of 16 instruction cycles to divide two real positive 16-bit integers, a few extra operations are required to find the sign of the dividend and to check for special cases such as divide by zero.

Tables 4.5 to 4.12 show the values of ψ_1, ψ_2 and ψ_3 when the algorithm was run using channels 1 to 8. For a comparison, the results of running the algorithm with 64-bit floating point arithmetic are also shown. The superscripts in some of the columns indicate that an incorrect number of roots lying outside the unit circle were found. For 64, 16 and 14 bit arithmetic, d (equation 4.28) was set to 10^{-7}, and for 12, 10 and 8 bit arithmetic a value of 10^{-6}, 10^{-5} and 10^{-4} was used for d, respectively. Making the values of d less than those given gave no useful improvement in the accuracy of the algorithm. The number of taps in the pre-filter (q+1) was kept as small as possible so that the amount of processing required to calculate them was kept to a minimum. For channels 1 to 3 the length of the pre-filter was set to (q+1)=30, for channel 4, (q+1)=40, and for channels 5 to 8 (q+1)=50. These values are the minimum required to maintain accuracy [27].

In tables 4.5 to 4.12 it can be seen that, in general, there is a worsening in the values of ψ_1,ψ_2 and ψ_3 as the number of bits used decreases. Table 4.5 shows that in the case of a channel that introduces little distortion, such as channel 1, the algorithm succeeded in locating all roots that lie outside the unit circle, even when 10-bit integer arithmetic was used. For channels 2 and 3 (tables 4.6 and 4.7) the algorithm successfully located all roots when integer arithmetic was used, but it failed to locate all roots when 10-bits or less were used. With channel 4, the algorithm failed to locate all roots when the arithmetical precision was reduced to 12 bits or less. For channels 5 and 6, which were sampled at 3200 samples/s, it can be seen (tables 4.9 and 4.12) that for the algorithm to

operate satisfactorily at least 12-bit integer arithmetic had to be used. With channels 7 and 8 the algorithm failed to locate all of the roots outside the unit circle. The reason for this failure is because both of these channels have many roots lying very close to the unit circle.

The magnitude of the change in the value of ψ_2, when converting from floating point to integer, is very large, as can be seen from tables 4.5 to 4.12. This is due to the quantization process which prevents very small numbers being represented accurately (ψ_2 is a measure of how close the minimum phase impulse response generated by the algorithm is to the minimum phase impulse response produced by convolving the channel impulse response with the pre-filter). The values of ψ_1 and ψ_3 do not change by such a large amount when changing from floating point to integer arithmetic.

For channels 1 to 3, varying the value of c from 0.5 to 1.0 has little effect upon the accuracy of the algorithm when integer arithmetic is used, but for channel 4 the value of ψ_3 is affected when c is reduced from 0.7 to 0.5. These results suggest that the value of c can be made equal to 1.0, thus allowing the algorithm to converge at its maximum rate without affecting its overall accuracy. For channels 5 and 6 a smaller value of c is needed if the accuracy of the algorithm is to be maintained.

Whilst finding the values of ψ_1, ψ_2 and ψ_3 in tables 4.5 to 4.12, a check was also made to see how many times equation 4.27 had to be used to give an indication of how long the algorithm took to find the minimum phase sequence. Tables 4.13 to 4.15 show the number of iterations taken for different values of c in equation 4.27. The tables show that the number of iterations required to locate all of the roots outside the unit circle increases as the value of c decreases. There is also a general increase in the number of iterations as the number of roots that lie outside the unit circle increases. It can be seen that the algorithm is quickest when c has a value of unity. There is no noticeable increase in the number of iterations performed when converting from floating point to 16-bit integer arithmetic.

The results in this section suggest that the algorithm will work satisfactorily with channels sampled at 2400 samples/s and with 16-bit or, possibly, 14-bit integer

arithmetic. With channels sampled at 3200 samples/s it is unlikely that a 16-bit microprocessor would give sufficient numerical precision for the Clark-Hau algorithm to operate reliably.

4.6.3 Running the Clark-Hau Algorithm on the TMS320C25

The performance of the Clark-Hau algorithm, when running on an actual 16-bit microprocessor, was tested using a Loughborough Sound Images TMS320C25 development board. The development board has a TMS320C25 processor, running at 40 MHz, 32K of RAM that can be used as program memory and 32K of RAM that can be used as data memory.

Initially, the computer simulation programs, which had been written in C, were recompiled so that they could be run on the development board. These programs worked but they were slow because of the inefficient way in which the C-compiler had produced the machine code. To obtain greater speeds, an assembly language program was written using some of the special features that the TMS320C25 has, such as the on chip memory, shift registers and the auxillary registers.

A further increase in speed was found by noting that the equation for ε_i (equation 4.25) can be written alternatively as [27]

$$\varepsilon_i = y_1 - 2\lambda_i y_2 + 3\lambda_i^2 y_3 - \ldots + g(-\lambda_i)^{g-1} y_g \qquad ..4.44$$

Now, it can easily be shown that the components in the sequence $\{e_{i,h}\}$, for i=0,1, ...,g, can be given by [27]

$$e_{i,0}=y_0-\lambda_i y_1+\lambda_i^2 y_2-\ldots+(-\lambda_i)^g y_g$$
$$e_{i,1}=y_1-\lambda_i y_2+\lambda_i^2 y_3-\ldots+(-\lambda_i)^{g-1} y_{g-1}$$
$$e_{i,2}=y_2-\lambda_i y_3+\lambda_i^2 y_4-\ldots+(-\lambda_i)^{g-2} y_{g-2}$$
$$\vdots$$
$$e_{i,g-2}=y_{g-2}-\lambda_i y_{g-1}+\lambda_i^2 y_g$$
$$e_{i,g-1}=y_{g-1}-\lambda_i y_g$$
$$e_{i,g}=y_g$$

If the sequence $\{e_{i,h}\}$ is now reversed in order, starting with $e_{i,g}$ and finishing with $e_{i,1}$, and passed through a filter identical to filter A, then the samples appearing at its output are given by

$$o_{i,1}=y_1-2\lambda_i y_2+3\lambda_i^2 y_3-4\lambda_i^3 y_4+\ldots+g(-\lambda_i)^{g-1} y_g$$
$$o_{i,2}=y_2-2\lambda_i+3\lambda_i^2 y_4+\ldots+(g-1)(-\lambda_i)^{g-2} y_g$$
$$\vdots$$
$$o_{i,g-1}=y_{g-1}-2\lambda_i y_g$$
$$o_{i,g}=y_g$$

and noticing that $o_{i,1}$ is identical to the polynomial in equation 4.45, giving ε_i. This suggest an alternative method for obtaining ε_i during each of the cycles that updates λ_i to give λ_{i+1} using equation 4.27. Instead of using a loop to directly evaluate equation 4.45, the $\{e_{i,h}\}$ which have been generated as described earlier, are reversed in order and fed into a filter identical to filter A to give the sequence $\{o_{i,h}\}$, from which the component $o_{i,1}$ can be extracted to give ε_i. The difference equation needed to generate

$e_{i,h}$, for h=0,1, . . .,g is

$$e_{i,j} = y_j - \lambda_i e_{i,j+1} \qquad ..4.45$$

for j=g,g-1, . . . ,0. The sequence $o_{i,h}$, for h=1,2, . . . ,g can then be generated using the difference equation

$$o_{i,j} = e_{i,j} - \lambda_i o_{i,j+1} \qquad ..4.46$$

for j=g,g-1, . . . ,1. Since $\{e_{i,h}\}$ have already been obtained, (to be further processed once a root has been found) the only extra operations needed to give ε_i are those involved with equation 4.46. To calculate $o_{i,1}$ only g iterations of equation 4.46 are needed, which involves g-1 complex multiplications and g-1 complex subtractions. Whereas, if equation 4.25 was used to give ε_i, 2g-3 complex multiplications and g-1 complex additions would have to be performed. Since each complex multiplication involves four real multiplications and two real additions, and each complex addition involves two real additions, and noting that on the TMS320C25 an addition or a multiplication takes the the same amount of time to perform, then 8g-8 numerical operations have to be carried out using equation 4.46, compared with 14g-20 numerical operations using equation 4.25. Thus, there a noticeable reduction in the number of operations carried out when equation 4.46 is used.

The speed of the resulting assembly language program that was written, using equation 4.46, was such that λ_i could be updated using equation 4.27 approximately 100 times in four sampling periods (one sampling period=1/2400 seconds) when the number of components (g+1) in the sampled impulse response of the channel was 20 and the number of taps in the pre-filter was 30 (it has been found that, for an H.F. radio link, the pre-filter taps need only be adjusted once every four sampling periods without having a noticeable effect upon the performance of the system [16]). When the Clark-Hau algorithm eventually converges to a root that lies outside the unit circle it then proceeds to pass the $\{e_{k,h}\}$, the output of the feedback filter, into filter B (the two tap feedforward filter) and it also updates the taps of the pre-filter. Considering the

extra time taken to pass the $\{e_{i,h}\}$ into filter B and to update the pre-filter taps, after a root has been found, and assuming that approximately five iterations of equation 4.27 are required to locate a root, the Clark-Hau algorithm, when running on the TMS320C25, can process approximately four roots (and update the pre-filter taps) in four sampling periods.

In table 4.16, the roots positions found by the Clark-Hau algorithm are shown, along with the roots positions obtained using the NAG routine C02ADF; also shown are the number of iterations taken to process each of the roots. The threshold level, d (equation 4.28), was set to 10^{-7} and the value of c was set to 1.0. Tables 4.17 to 4.19 show the scaled minimum phase sampled impulse responses of channels 1,2 and 3; also shown are the scaled NAG values for comparison.

The results show that the Clark-Hau algorithm will run on the TMS320C25 and that its speed is such that it could possibly be used to adjust the pre-filter taps in an H.F. radio system. It is certainly fast enough for it to be used in a telephone system (at baud rates of upto 2400) and, if the channel is time invariant, it only has to be run once at the start of transmission during the start-up period.

4.6.4 Clark-Hau Algorithm Operating with a Noisy Estimate of the Sampled Impulse Response

In this section the performance of the Clark-Hau algorithm is examined when a noisy estimate of the sampled impulse response of the channel is used. The channel estimator is first allowed to converge to an accurate estimate of the impulse response Y, by sending a known sequence of data symbols at the start of transmission. Once the estimator has settled down the noisy estimate is fed into the Clark-Hau algorithm. To measure the performance of the algorithm, using estimates of the channel at different signal-to-noise ratios, the value of ψ_3 was found (equation 4.42) which compares the minimum phase sequence produced by the algorithm with the NAG values.

Figures 4.35 to 4.38 show the variation of ψ_3 with SNR for channels 1 to 4 with the

algorithm using reduced precision arithmetic, the numbers placed next to some of the points show how many roots outside the unit circle were found for that particular SNR. It can be seen that there is a worsening in the accuracy of the estimate of the minimum phase sequence as the numerical precision is reduced, but at a signal-to-noise ratio of 30 dB (typical for telephone channels) there is little difference between the accuracy of the floating point program and that of the 16-bit integer program. In fact, as the signal-to-noise ratio decreases, the accuracy of the minimum phase sequence tends to become similar no matter what the arithmetical precision is, this is due to the additive noise drowning out the effects of the quantization noise.

4.7 The Complete System

The channel estimator and Clark-Hau algorithm are now operated together with the near maximum likelihood detector to make up the core of a digital modem receiver. A model of this system is shown in figure 4.39. At the start of transmission a training signal, known to the receiver, is sent. The training sequence is long enough for the channel estimator to converge to an accurate estimate of the sampled impulse response. In the tests carried out, it is assumed that the channel is time invariant and therefore, once the estimator has converged, the estimate of the sampled impulse response is then fixed for the rest of the transmission. The estimate is fed into the Clark-Hau algorithm, which produces two sequences, $\{f_{m,h}\}$ and $\{d_h\}$, which are the estimate of the minimum phase sequence and the estimate of the pre-filter tap weights, respectively. The detector then uses the sequence $\{f_{m,h}\}$ to carry out the detection process.

In chapter 3 two near maximum likelihood detectors were described (detectors A and B). In detector A, the costs of the expanded vectors $\{P_i\}$, are given by

$$C_i = C_{i-1} + \left| r_i - \sum_{h=0}^{g} x_{i-h} y_h \right|^2 \qquad ..4.47$$

which can be expressed alternatively as

$$C_i = C_{i-1} + |z_i - y_0 x_i|^2 \qquad ..4.48$$

where

$$z_i = r_i - \sum_{h=1}^{g} x_{i-h} y_h \qquad ..4.49$$

Now, to reduce the number of operations carried out by the detector, rather than calculating the mk costs (where m is the number of possible values each data symbol can have and k is the number of stored vectors) of the mk expanded vectors $\{P_i\}$using equation 4.47, it first computes the k values of z_i (using equation 4.49) for each of the stored vectors Q_i. The value of z_i for m expanded vectors $\{P_i\}$ derived from one of the vectors Q_i will be the same, and therefore, equation 4.49 need only be used k times for each detection instant. After calculating z_i for one of the stored vectors Q_i,the detector calculates m costs (by substituting the m possible values of the term $y_0 x_i$ into equation 4.48) corresponding to the m expanded vectors $\{P_i\}$ and it stores these costs in memory. The position in the memory in which each cost is stored is determined by the value of x_i and which of the stored vectors Q_i was used to calculate it. Thus, when all of the costs have been calculated, the detector will have in store a list of mk costs, each one of them being calculated without actually expanding the stored vectors $\{Q_i\}$. The detector now searches through the list stored costs to find the smallest cost. The position of the smallest cost in the memory tells the detector which of the stored vectors Q_i and the value of x_i were used to calculate it. The detector now knows the maximum likelihood vector Q_i, the first component of which, is the detected data symbol s_{i-n}'. The selected vector Q_i is now copied over into a temporary store along with the value of the smallest cost. Also, the component x_i (which can be determined by the position of the smallest cost in memory) used in the calculation of the smallest cost is added to the end of the selected vector Q_i in the temporary memory to give an expanded vector P_i. The selected smallest cost in the list is now set to some arbitrarily high value to prevent it from being selected again. The detector now compares the first components of each of the stored vectors Q_i (including the selected vector with smallest cost) with the

detected data symbol s_{i-n}' and discards those vectors whose first components differ. The discarding process is achieved by setting the m costs (in the list of costs) derived from the stored vector whose first component differs from s_{i-n}' to some arbitrarily large value. The detector now repeats the above process to find the remaining k-1 expanded vectors $\{P_i\}$ and their costs, which are held in the temporary store. The first component of each of the expanded vectors P_i is now discarded (without changing their costs) and the resulting vectors and costs are used to replace the original stored vectors $\{Q_i\}$ and their costs.

The process just described cuts out a large number of operations that would otherwise be unneccesarily repeated if the stored vectors $\{Q_i\}$ were actually expanded and the costs of the expanded vectors $\{P_i\}$ calculated using equation 4.47.

In the description of detector B it was assumed that the first component of the estimate of the minimum phase sampled impulse response of the channel had been adjusted to a value of 1+j0 [10]. To obtain such a response every, component of the minimum phase sequence produced by the Clark-Hau algorithm would have to be divided by the value of the first in that sequence, y_0. This, however, would be a computationaly intensive procedure because it would involve complex divisions. An alternative to this is to modify detector B such that the first component of the minimum phase sequence does not have to be made equal to unity. A method for achieving this will now be described.

Prior to the receipt of the sample r_i, the detector holds in store the k vectors $\{Q_i\}$,

$$Q_i=[x_{i-n}\, x_{i-n+1} \cdots x_{i-1}] \qquad ..4.50$$

and it also holds in store the k costs $\{C_{i-1}\}$ associated with each of the stored vectors Q_i. Now, if the stored vectors were to be expanded to give the vectors $\{P_i\}$, then the costs associated with each one of these expanded vectors will be given by

$$C_i = C_{i-1} + \left| r_i - \sum_{h=0}^{g} x_{i-h} y_h \right|^2$$

$$= C_{i-1} + \left| r_i - f_i - y_0 x_i \right|^2 \qquad ..4.51$$

where

$$f_i = \sum_{h=1}^{g} x_{i-h} y_h \qquad ..4.52$$

and y_0 doesn't necessarily have to be equal to 1+j0. Since the first n components of a group of m expanded vectors P_i, derived from any vector Q_i, will be the same, the detector only has to compute the quantity f_i once for each group of expanded vectors, giving a saving in the number of operations carried out to give the costs of the expanded vectors. If we let

$$z_i = r_i - f_i \qquad ..4.53$$

then

$$C_i = C_{i-1} + |z_i - y_0 x_i|^2 \qquad ..4.54$$

and also let

$$d_i = z_i - y_0 x_i \qquad ..4.55$$

then selecting the real and imaginary parts of x_i that minimise the real and imaginary parts of d_i will lead to the smallest cost C_i. However, the simple threshold level method, described in chapter 3, can't be used here to give the smallest value of $|d_i|^2$ because y_0 may not be equal to unity. This problem can be overcome by multiplying

both sides of equation 4.55 by y_0^*, the complex conjugate of y_0, to give

$$y_0^* d_i = y_0^* z_i - |y_0|^2 x_i \qquad ..4.56$$

The detector now operates by first calculating the value of $y_0^* z_i$ associated with one of the stored vectors Q_i, it then finds the value of $|y_0|^2$ and uses the threshold level method to find the real and imaginary parts of $|y_0|^2 x_i$ that come closest to the real and imaginary parts of $y_0^* z_i$ and hence the real and imaginary parts of x_i. For a 16-level QAM signal, the possible values of x_i will be -3, -1, 1 and 3, therefore the detector only has to compute the values of $|y_0|^2$ and $4|y_0|^2$ for it to know all m possible values of $|y_0|^2 x_i$ in equation 4.56. The detector carries on in a similar way as described in chapter 3, but using equation 4.56 as the basis for the threshold comparison.

4.7.1 Computer Simulation Tests

Computer simulation tests were carried out to show the performance (bit error rate) of the complete system, operating with reduced precision arithmetic at the receiver. Data rates of 9.6, 14.4 and 19.2 Kbits/s were used in the tests, at baud rates of 2400 over channels 1 to 4 and 3200 over channel 5 to 8. To achieve data rates of 9.6 and 14.4 Kbits/s, 16 and 64-level QAM signal were used at a baud rate of 2400, and for 19.2 Kbits/s, a 64-level QAM signal was used at a baud rate of 3200. In figures 4.40 and 4.41 the error rate curves for an ideal communications channel that introduces no distortion and only additive white Gaussian noise are shown. Since the channel introduces no distortion, its impulse response will be a scaled and delayed impulse and, under the assumed conditions, the optimum detector will be a simple threshold detector. The results of the computer simulation tests of the complete system operating at 9.6, 14.4 and 19.2 Kbits/s will now be presented.

9.6 KBits/s

Figures 4.42 to 4.45 show the performance of the complete system operating at 9.6 Kbits/s, with detector A. The performance of the system at a lower arithmetical precision is close to the performance of the system when 64-bit floating point arithmetic is used, which is encouraging but not surprising, since the results obtained in the previous sections all suggest that at signal-to-noise ratios of less than 30 dB there is little difference between the 64-bit and 16-bit integer programs. When 10-bit integer arithmetic was used the system failed to operate reliably when the data was sent over channels 1,2 and 3. With channel 4, which introduces very severe amplitude distortion, the system failed to operate reliably when 12-bits or less were used. A delay in detection of eight sampling intervals was chosen (i.e. n=8), increasing the delay has little effect upon the error rate of the system [55]. The results suggest that a dedicated processor using 14-bit integer arithmetic could be used at the receiver when data rates of upto 2400 baud are used. However, to take into account any incorrect adjustment of the gain circuits ahead of the A/D converter at the receiver (which would normally be adjusted so that full use of the output number range from the A/D converter was achieved) and because of the convenience of currently available 16-bit microprocessors, the use of 16-bit arithmetic is a better choice. Also, it must be remembered that the results presented here are upper limits since the system performance would also be affected by the accuracy of the carrier phase synchronisation, the symbol timing recovery and other types of noise.

An example of the reduction in the tolerance of the system (with the receiver using 64-bit floating point arithmetic) to additive noise when no pre-filter is used ahead of the detector is given in figure 4.46, where channel 1, which introduces the least amount of distortion, has been used in the test. It can be seen that there is a large drop in the overall performance of the system. The loss of performance is due to the first few components of the non-minimum phase impulse response having a relatively small magnitude compared with some of the latter components. The small magnitude of the first component makes the selection of the expanded vector (in the near maximum likelihood detector) with the smallest cost almost arbitrary.

To further simplify the complexity of the receiver, the number of taps in the channel estimator could be reduced. This would result in a shorter estimate of the sampled impulse response of the channel which leads to a reduction in the number of computations performed by the Clark-Hau algorithm. Figures 4.47 to 4.50 show the performance of the system using integer arithmetic and where the channel estimate had only 10 components for channels 1 and 2 and 15 components for channels 3 and 4 [72]. Comparing these figures with figures 4.42 to 4.45 it can be seen that there is no noticeable reduction in performance.

The performance of the complete system when detector B is used in place of detector A and when the shorter estimates of the channel estimate have been used, are shown in figures 4.51 to 4.54. The lengths of the stored vectors and the number of stored vectors used by detector B are the same as those used by detector A to generate figures 4.42 to 4.45. Figures 4.51 and 4.52 show that there is a 1 dB drop in performance (at an error rate of 10^{-4}) compared with figures 4.42 and 4.43. For channel 3 there is a 2 dB drop in performance compared with figure 4.44 and for channel 4 there is a 3.5 dB drop compared with figure 4.45.

The complexity of detector B is much lower than that of detector A; this can be illustrated as follows. For a m-level signal, detector A has to compute mk costs (where k is the number of stored vectors) and perform k searches through mk costs, whereas detector B has to calculate 2k costs and perform k searches through 2k costs. So if m=16 and k=8, then, detector A will have to compute 128 costs and detector B would have to compute 16 costs which is a great saving in complexity. The very much lower complexity of detector B makes it very attractive for use in a practical modem.

Figures 4.55 and 4.56 show the error rate curves when a simple adaptive decision feedback equaliser was used with channels 1 and 2. The results were obtained with the equaliser using 16-bit integer arithmetic, and it can be seen that there is a 3.5 dB drop in performance compared with the performance of detector A operating with the same channels. For channels 3 and 4, and when integer arithmetic was used, the equaliser failed to operate satisfactorily.

14.4 Kbits/s

Figures 4.57 to 4.60 show the performance of the system with channels 1 to 4 and detector A being used. Again the drop in the performance of the system when changing from 64-bit floating point arithmetic to 16-bit integer arithmetic is small.

Figures 4.61 to 4.64 show the performance of the system with detector B and it can be seen that there is no real loss in performance when compared with detector A. With a 64-level signal and 4 stored vectors, detector A would have to compute 256 costs and with detector B using 8 stored vectors, only 16 costs have to be evaluated. There is thus a very great saving in equipment complexity if detector B is used.

Figures 4.65 and 4.66 show the effects on the bit error rate when an inaccuracy in the estimate of the carrier phase occurs over channels 1 and 4, respectively. The results were obtained by introducing a phase error in the received samples after the channel estimator had converged and the system had settled. In the tests, although the phase error was fixed, the results give an idea of the effects of small and random changes in the received signal phase. The results show that there is little increase in the sensitivity of the system to phase errors when the numerical accuracy is reduced from 64-bit floating point to 16-bit integer arithmetic.

19.2 Kbits/s

The results presented in section 4.6 indicated that with sampled impulse responses sampled at 3200 samples/s, the Clark-Hau algorithm only worked reliably with channels 5 and 6 when integer arithmetic was used. These results suggest that the performance of the complete system operating at 3200 baud would be seriously affected by the channel characteristics when integer arithmetic is used.

It was found that the system only worked with channels 5 and 6, as expected. Figures 4.65 and 4.66 show the performance of the system, operating with detector B.

4.8 Design Considerations for a Modem Using the TMS320C25

Overall, the results presented in this chapter suggest that for a modem operating at 2400 baud over a telephone channel, 16-bit integer arithmetic should provide enough precision for reliable operation.

For a time invariant channel a single TMS320C25 could be used to carry out all of the operations needed in the detection process. At the start of transmission the channel estimator program could be run, after it has converged the Clark-Hau algorithm can be run once to give the minimum phase sampled impulse response of the channel needed by the near maximum likelihood detector and the tap gains of the pre-filter. The same TMS320C25 can then be used to run the near maximum likelihood detector program. For carrier phase tracking and timing recovery another TMS320C25 would be needed.

If the channel is varying very slowly with time then one TMS320C25 could be used to continuously run the channel estimator program and Clark-Hau algorithm, whilst the other could be used to run the near maximum likelihood detector.

Table 4.1 Sampled Impulse Responses of Channels 1 to 4

Channel 1		Channel 2		Channel 3		Channel 4	
Real	Imaginary	Real	Imaginary	Real	Imaginary	Real	Imaginary
-0.0291	-0.0373	0.0145	-0.0006	0.0176	-0.0175	-0.0038	-0.0049
0.2290	-0.2244	0.0750	0.0176	0.1381	-0.1252	0.0077	-0.0044
0.7612	0.1817	0.3951	0.0033	0.4547	-0.1885	0.0094	0.0207
0.2988	0.3050	0.7491	-0.1718	0.5078	0.1622	-0.0884	0.0355
-0.0338	-0.2915	0.1951	0.0972	-0.1966	0.3505	-0.1138	-0.2869
-0.0789	0.0616	-0.2856	0.1894	-0.2223	-0.2276	0.5546	-0.2255
0.0291	0.0287	0.0575	-0.2096	0.2797	-0.0158	0.1903	0.5813
-0.0137	-0.0352	0.0655	0.1139	-0.1636	0.1352	-0.2861	-0.0892
0.0020	0.0204	-0.0825	-0.0424	0.0594	-0.1400	0.2332	-0.0384
0.0004	-0.0108	0.0623	0.0085	-0.0084	0.1111	-0.0652	0.0428
0.0028	0.0065	-0.0438	0.0034	-0.0105	-0.0817	0.0335	-0.0519
-0.0027	-0.0014	0.0294	-0.0049	0.0152	0.0572	-0.0323	0.0170
0.0000	-0.0013	-0.0181	0.0032	-0.0131	-0.0406	0.0044	-0.0023
0.0003	0.0006	0.0091	0.0003	0.0060	0.0255	0.0054	0.0076
-0.0002	0.0001	-0.0038	-0.0023	0.0003	-0.0190	0.0008	-0.0051
-0.0009	-0.0006	0.0019	0.0027	-0.0035	0.0116	-0.0056	0.0001
0.0005	-0.0000	-0.0018	-0.0014	0.0041	-0.0078	0.0018	0.0032
0.0003	0.0004	0.0006	0.0003	-0.0031	0.0038	-0.0009	-0.0015
0.0001	-0.0011	0.0005	0.0000	0.0018	-0.0005	-0.0022	-0.0026
0.0004	0.0001	-0.0008	-0.0001	-0.0018	-0.0005	0.0029	0.0019
				0.0007	0.0007	-0.0008	0.0009
				0.0004	0.0001	-0.0014	-0.0003
				-0.0004	0.0001	0.0019	-0.0002
				-0.0001	0.0010	-0.0003	0.0005
				0.0000	-0.0007	0.0007	0.0005
				0.0004	0.0008	-0.0007	-0.0001
						0.0002	-0.0008

Table 4.2 Sampled Impulse Responses of Channels 5 to 6

Channel 5		Channel 6		Channel 7		Channel 8	
Real	Imaginary	Real	Imaginary	Real	Imaginary	Real	Imaginary
-0.004807	0.001975	0.001966	0.000544	0.001155	-0.001275	0.000139	-0.000657
-0.023279	-0.005598	0.016707	0.003912	0.011734	-0.016035	0.002122	-0.001400
-0.003958	-0.095155	0.067284	0.008166	0.057043	-0.059210	0.005858	0.003295
0.249745	-0.169609	0.221772	0.000227	0.170236	-0.103845	-0.005860	0.020025
0.541549	0.119610	0.502139	-0.056595	0.289941	-0.041411	-0.058710	0.003587
0.297847	0.342398	0.548069	-0.110307	0.203587	0.140088	-0.055260	-0.122651
0.011628	-0.093299	0.046425	0.064073	-0.106333	0.175431	0.165603	-0.166552
0.069883	-0.227266	-0.294318	0.210038	-0.188929	-0.059493	0.264580	0.134973
-0.116629	0.107089	0.006498	-0.101885	0.098031	-0.135501	-0.084000	0.238554
-0.051507	0.055005	0.145968	-0.128412	0.116619	0.098686	-0.122758	-0.114973
0.120645	-0.051534	-0.070121	0.184022	-0.141349	0.062653	0.163238	-0.040380
-0.080646	-0.017534	-0.051347	-0.055243	0.004742	-0.126885	-0.027620	0.118527
-0.001307	0.048080	0.084641	-0.087113	0.115057	0.052773	-0.049954	-0.093587
0.054696	-0.017118	-0.017438	0.127455	-0.106427	0.058291	0.076453	0.006640
-0.042190	-0.025187	-0.048800	-0.070772	0.013733	-0.100340	-0.054343	0.029049
0.007644	0.042773	0.059209	-0.008137	0.067016	0.061483	0.008210	-0.035339
0.016672	-0.034243	-0.029553	0.055785	-0.085331	0.007381	0.007390	0.029327
-0.023740	0.016745	-0.007553	-0.058468	0.049681	-0.055936	-0.008383	-0.010218
0.013562	-0.003006	0.028324	0.035002	0.000236	0.061144	0.010418	0.002475
0.001807	-0.004991	-0.031788	-0.007764	-0.035259	-0.039069	-0.007445	-0.003145
-0.014160	0.007369	0.025419	-0.010526	0.045572	0.006828	0.002625	0.001647
0.017820	-0.008654	-0.014780	0.018476	-0.036460	0.016117	-0.003203	0.001599
-0.014543	0.010301	0.004397	-0.018811	0.019015	-0.027514	0.003120	-0.000777
0.007007	-0.012037	0.002551	0.014924	-0.001935	0.027598	-0.003116	-0.001415
0.001402	0.010854	-0.006499	-0.010226	-0.009838	-0.020545	0.001378	-0.000502
-0.007189	-0.007094	0.007904	0.005818	0.015003	0.011764	-0.000249	0.001776
0.009463	0.000670	-0.007652	-0.002228	-0.015829	-0.003476	0.001308	-0.000529
-0.006936	0.004000	0.006315	-0.000400	0.013445	-0.002168		
0.002667	-0.006328	-0.004811	0.001611	-0.009728	0.005539		
0.001578	0.005983			0.006697	-0.006761		
-0.004066	-0.003360			-0.004013	0.007442		
				0.001671	-0.007064		
				0.000127	0.006670		
				-0.001612	-0.005538		
				0.002745	0.004606		
				-0.003295	-0.003236		
				0.003444	0.002208		
				-0.003439	-0.001632		
				0.003402	0.000765		
				-0.003335	-0.000310		
				0.003124	-0.000448		
				-0.002731	0.000974		

Table 4.3 Minimum Phase Sampled Impulse Responses of Channels 1 to 4

Channel 1		Channel 2		Channel 3		Channel 4	
Real	Imaginary	Real	Imaginary	Real	Imaginary	Real	Imaginary
0.5349	0.6551	0.7320	-0.4731	0.5587	-0.2035	-0.3188	-0.0729
0.0452	0.5097	0.4632	-0.0910	0.4814	0.5210	0.0662	-0.6489
-0.1082	-0.0948	-0.1098	0.0624	-0.3165	0.1429	0.5328	0.1901
0.0350	-0.0093	0.0173	-0.0212	0.0526	-0.1286	-0.2726	0.2031
-0.0183	0.0081	0.0105	0.0012	0.0079	0.0523	0.0255	-0.1639
0.0008	-0.0166	-0.0117	0.0035	-0.0051	-0.0104	0.0261	0.0546
0.0035	0.0058	0.0076	-0.0043	0.0005	0.0050	-0.0065	-0.0144
-0.0023	-0.0033	-0.0085	0.0041	-0.0043	-0.0031	0.0025	0.0129
0.0034	0.0040	0.0071	-0.0022	0.0056	0.0013	-0.0038	-0.0069
-0.0034	-0.0005	-0.0041	0.0005	-0.0028	-0.0011	0.0046	0.0014
0.0006	-0.0019	0.0011	0.0017	0.0002	0.0008	-0.0001	0.0008
0.0007	0.0005	0.0001	-0.0021	0.0010	-0.0038	-0.0009	0.0011
-0.0001	0.0000	0.0004	0.0017	0.0000	0.0016	0.0001	0.0000
-0.0008	-0.0007	-0.0007	-0.0003	0.0007	-0.0019	0.0003	0.0025
0.0007	0.0001	-0.0002	0.0000	-0.0004	-0.0004	-0.0017	-0.0020
0.0000	0.0005	0.0006	0.0000	0.0011	0.0007	0.0012	-0.0013
0.0003	-0.0004	-0.0003	0.0000	-0.0011	-0.0004	0.0017	0.0015
0.0005	0.0002	-0.0002	0.0000	-0.0001	0.0000	-0.0019	0.0000
0.0000	0.0002	0.0000	0.0000	0.0000	0.0000	0.0006	0.0010
0.0000	0.0000	0.0000	0.0000	0.0002	0.0000	0.0003	0.0000
				-0.0001	0.0007	0.0004	0.0002
				-0.0005	0.0000	-0.0002	0.0003
				0.0000	0.0000	0.0000	0.0001
				0.0001	0.0004	0.0000	0.0000
				0.0000	0.0002	0.0000	0.0000
				0.0000	0.0000	0.0000	0.0000
						0.0000	0.0000

Table 4.4 Minimum Phase Sampled Impulse Responses of Channels 5 to 8

Channel 5		Channel 6		Channel 7		Channel 8	
Real	Imaginary	Real	Imaginary	Real	Imaginary	Real	Imaginary
-0.176657	0.450177	0.285649	-0.507757	0.245356	-0.032166	0.044694	-0.063505
-0.399730	0.542591	0.456534	-0.561966	0.342642	0.289367	0.239493	0.034044
-0.122502	-0.125097	0.012011	0.113587	-0.173279	0.294660	0.034113	0.327801
0.078662	-0.084614	-0.070731	0.064164	-0.131818	-0.144464	-0.288963	0.016585
-0.036314	0.089678	0.074859	-0.097823	0.119692	-0.006269	0.049581	-0.197055
0.008401	-0.069168	-0.030075	0.061777	-0.056124	0.060425	0.092456	0.088525
0.013867	-0.009683	-0.006970	0.001947	-0.006390	-0.049473	-0.077014	0.010535
-0.001927	0.020163	0.016624	-0.022139	0.026545	0.019287	0.027475	-0.039722
-0.007863	-0.006377	-0.005322	0.011138	-0.017021	0.004397	0.007395	0.029157
0.005561	-0.000853	0.004933	-0.004554	0.005391	-0.006058	-0.015325	-0.005654
-0.001619	0.011047	-0.003877	0.000286	-0.000058	0.004033	0.006906	-0.003861
-0.004959	-0.012396	0.002008	0.001193	-0.001746	-0.000675	0.000220	0.004039
0.007910	0.008348	0.001272	0.000674	0.001391	-0.000434	-0.001896	-0.000971
-0.010122	-0.003998	-0.003264	-0.000717	-0.001014	0.000034	0.000519	-0.001212
0.011565	-0.003163	0.002494	0.001114	0.000797	-0.001353	-0.000322	0.000558
-0.007692	0.008869	-0.001090	-0.000537	0.000718	0.000505	-0.000888	-0.000520
0.002620	-0.011546	0.000506	-0.000336	-0.000546	-0.000697	-0.000105	0.000323
0.003273	0.010234	0.000577	0.000712	0.001687	-0.000817	-0.000436	-0.000275
-0.007178	-0.005669	-0.001467	-0.000323	-0.000707	0.000733	0.001069	-0.000818
0.008546	-0.000367	0.001110	-0.000183	0.000674	-0.000876	0.000751	0.000584
-0.006399	0.005453	-0.000319	0.000245	-0.000133	0.000688	-0.000163	0.000613
0.002282	-0.006319	-0.000792	0.000471	-0.000300	-0.000296	0.000043	0.000214
0.002500	0.004701	0.000990	-0.000962	0.000028	0.000014	-0.000067	0.000032
-0.004552	-0.001078	-0.000093	-0.000125	-0.000586	-0.000010	-0.000100	0.000013
0.003247	-0.000924	-0.000603	0.001356	0.000733	-0.000096	-0.000022	-0.000025
-0.000182	0.002550	-0.000244	0.001014	-0.000640	0.000515	0.000008	0.000015
-0.001210	-0.002311	-0.000044	0.000305	0.000255	-0.000280	0.000012	0.000000
0.001877	-0.001446	-0.000005	0.000106	0.000095	0.000304		
0.000747	0.000543	0.000001	0.000018	-0.000768	0.000082		
-0.000061	0.000180			0.000619	-0.000283		
-0.000056	0.000007			-0.000306	0.000386		
				-0.000531	0.000121		
				0.000636	-0.000180		
				-0.000120	0.000086		
				-0.000565	0.000086		
				0.000650	-0.000202		
				0.000439	0.000067		
				-0.000338	-0.000018		
				-0.000749	-0.000286		
				-0.000398	-0.000276		
				-0.000157	-0.000109		
				-0.000019	-0.000007		

Table 4.5 Values of ψ_1, ψ_2 and ψ_3 for channel 1

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-75.5 -322.2 -59.2	-77.9 -322.2 -59.7	-83.9 -320.0 -59.9
16	-50.0 -54.9 -57.3	-50.0 -54.7 -57.9	-50.0 -55.0 -57.9
14	-38.6 -43.8 -50.6	-38.6 -43.7 -50.6	-38.6 -43.7 -50.6
12	-28.8 -32.9 -39.6	-28.7 -33.2 -40.4	-28.6 -33.5 -39.5
10	-24.4 -25.0 -29.6	-24.3 -25.2 -30.0	-25.1 -25.4 -29.7
8	-8.4[2] -15.4 -2.9	-9.8[2] -14.9 -4.7	-7.7 -15.6 -20.6

Table 4.6 Values of ψ_1, ψ_2 and ψ_3 for channel 2

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-73.4 -311.1 -61.2	-72.4 -309.1 -62.2	-91.6 -317.3 -62.7
16	-51.5 -54.0 -58.2	-51.5 -54.0 -59.0	-51.4 -54.1 -59.3
14	-41.1 -41.8 -49.4	-40.8 -41.8 -49.5	-40.9 -42.0 -49.3
12	-30.2 -30.8 -39.1	-30.4 -30.5 -38.1	-28.0 -30.6 -37.0
10	-18.7[3] -21.0 -2.3	-18.6[3] -21.0 -2.4	-19.1 -20.5 -26.4
8	-9.0 -11.6 -20.3	-9.7 -10.1 -16.1	-4.4[2] -13.4 9.6

Table 4.7 Values of ψ_1, ψ_2 and ψ_3 for channel 3

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-70.2 -315.4 -62.8	-70.0 -318.2 -73.4	-71.1 -317.6 -89.2
16	-49.2 -50.6 -54.1	-48.6 -50.9 -54.6	-48.2 -51.2 -52.9
14	-38.2 -39.5 -42.6	-37.4 -39.7 -44.2	-37.5 -39.9 -44.1
12	-25.5 -28.2 -31.7	-25.9 -27.9 -30.8	-25.3 -28.1 -33.0
10	-14.7^{3} -19.6 -0.8	-24.4^{3} -20.0 -1.4	-18.2 -19.2 -24.2
8	-5.3^{3} -9.0 -1.0	-4.9 -8.3 -10.1	-5.7 -8.8 -13.1

Table 4.8 Values of ψ_1, ψ_2 and ψ_3 for channel 4

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-40.6 -313.1 -45.9	-40.8 -311.1 -52.6	-40.7 -310.3 -52.3
16	-43.3 -53.3 -33.7	-45.1 -52.3 -47.5	-45.1 -53.7 -46.6
14	-33.7 -40.5 -30.7	-33.9 -40.9 -34.2	-42.1^{1} -41.1 44.5
12	-22.7 -31.2 -22.7	-22.1^{7} -28.7 -19.5	-22.2 -30.9 -23.2
10	-13.9^{11} -22.2 -14.0	-16.5^{6} -20.9 -13.2	-13.2^{10} -23.7 -4.6
8	-5.2 -18.6 -2.8	-4.9^{7} -21.4 -6.6	No Roots Found

Table 4.9 Values of ψ_1, ψ_2 and ψ_3 for Channel 5

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-42.5 -312.5 -66.5	-43.2 -312.4 -62.1	-43.2 -312.0 -58.0
16	-40.6 -49.9 -43.1	-37.4 -50.3 -36.0	-42.7 -50.8 -21.6
14	-31.6 -37.6 -21.1	-31.0^{7} -37.7 -16.4	-29.8 -37.5 -21.0
12	-19.9 -24.7 -16.1	-21.4^{6} -25.3 -23.0	-20.4^{7} -25.4 -15.3
10	-8.5^{7} -14.6 -9.5	-8.9^{7} -14.6 -9.7	-8.2 -14.1 -10.0

Table 4.10 Values of ψ_1, ψ_2 and ψ_3 for Channel 6

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-43.4 -311.5 -59.7	-43.5 -309.5 -60.3	-43.4 -309.9 -77.2
16	-42.3 -47.4 -39.1	-39.2 -47.5 -36.8	-44.8^{8} -48.2 -7.2
14	-33.3 -36.3 -30.1	-31.8 -35.5 -30.5	-32.7 -35.3 -28.1
12	-21.8 -25.0 -25.3	-22.0^{7} -24.8 -9.6	-21.2^{6} -25.4 1.7
10	-10.0^{7} -13.3 -13.1	-10.3^{7} -13.4 -16.0	-10.0^{7} -13.7 -9.8

Table 4.11 Values of ψ_1, ψ_2 and ψ_3 for Channel 7

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-28.2 -305.5 -60.3	-28.9 -305.8 4.9	-28.8 -309.5 4.9
16	-31.9 -44.7 -9.0	-31.1 -44.4 -8.8	-30.0 -44.8 -2.2
14	-29.7 -38.6 -24.5	-28.0 -33.1 -18.9	-28.0 -33.8 11.9
12	-18.7 -21.3 8.2	-19.0 -22.9 9.7	-18.3 -21.3 -5.6

Table 4.12 Values of ψ_1, ψ_2 and ψ_3 for Channel 8

Number of Bits Used	c=0.5	c=0.7	c=1.0
64	-39.8 -312.0 -33.9	-37.8^{12} -309.7 12.4	-39.1^{11} -309.3 8.6
16	-40.5^{10} -49.8 -3.0	-40.5^{10} -51.1 6.7	-44.3^{8} -51.7 17.8
14	-32.5^{11} -37.7 2.3	-33.4^{10} -39.1 5.5	-33.2^{9} -40.6 13.7
12	-22.0^{8} -29.0 9.6	-21.2^{11} -26.1 11.6	-21.8^{20} -28.9 15.2

Table 4.13 Number of Iterations Taken to Converge, with c=0.5

Number of bits used	Channel							
	1	2	3	4	5	6	7	8
64	45	58	72	144	234	217	266	397
16	48	61	68	120	239	162	294	174
14	47	54	64	104	651	192	1330	298
12	41	48	56	110	189	147	511	312
10	75	221	80	190	863	875	-	-
8	209	46	75	176	-	-	-	-

Table 4.14 Number of Iterations Taken to Converge, with c=0.7

Number of bits used	Channel							
	1	2	3	4	5	6	7	8
64	29	41	53	112	168	152	175	262
16	33	45	56	94	186	134	146	222
14	30	42	48	90	104	127	410	136
12	29	39	42	95	89	92	102	170
10	64	217	81	47	290	208	-	-
8	210	66	32	97	-	-	-	-

Table 4.15 Number of Iterations Taken to Converge, with c=1.0

Number of bits used	Channel							
	1	2	3	4	5	6	7	8
64	19	32	36	104	127	102	189	190
16	20	35	35	63	116	87	157	109
14	20	35	37	60	106	102	245	121
12	19	27	36	103	302	292	105	289
10	19	32	52	88	200	283	-	-
8	57	211	30	-	-	-	-	-

Table 4.16 Comparison of root positions found by the Clark-Hau Algorithm with those obtained with the NAG routines

	NAG		Clark-Hau Algorithm		
Channel	Real	Imaginary	Real	Imaginary	Iterations
1	1.5816	1.8716	1.5798	1.8723	6
	1.3527	0.4453	1.3519	0.4449	5
	-1.8112	-4.7915	-1.8106	-4.8089	5
2	2.0621	-0.5661	2.0761	-0.5727	3
	0.6153	-3.5564	0.6326	-3.5010	5
	1.7858	5.1965	1.8283	5.1968	4
	1.4073	-0.1481	1.4048	-0.1492	4
3	2.1505	-0.4796	2.6600	-0.3798	2
	1.2298	1.3614	1.3490	1.2958	7
	4.0645	-1.5207	3.2297	-0.8483	8
	1.3431	0.2367	1.3957	0.2190	5

Table 4.17 Comparison of sampled impulse responses for channel 1

NAG		Algorithm	
Real	Imaginary	Real	Imaginary
1.0000	0.0000	1.0000	0.0000
0.5006	0.3397	0.5004	0.3397
-0.1678	0.0282	-0.1679	0.0279
0.0176	-0.0391	0.0176	-0.0392
-0.0062	0.0229	-0.0064	0.0228
-0.0146	-0.0132	-0.0147	-0.0132
0.0080	0.0012	0.0079	0.0011
-0.0049	-0.0004	-0.0050	-0.0004
0.0063	-0.0001	0.0061	-0.0002
-0.0031	0.0027	-0.0032	0.0027
-0.0013	-0.0020	-0.0014	-0.0021
0.0012	-0.0003	0.0009	-0.0003
-0.0003	0.0001	-0.0003	-0.0001
-0.0011	0.0002	-0.0015	0.0001
0.0004	-0.0008	0.0005	-0.0007
0.0008	0.0007	0.0004	0.0003
-0.0002	-0.0011	-0.0004	-0.0007
0.0004	-0.0002	0.0005	-0.0004
-0.0002	0.0003	0.0001	0.0002
0.0004	0.0000	-0.0001	0.0000

Table 4.18 Comparison of sampled impulse responses for channel 2

NAG		Algorithm	
Real	Imaginary	Real	Imaginary
1.0000	0.0000	1.0000	0.0000
0.5031	0.2008	0.5026	0.2008
-0.1447	-0.0083	-0.1449	-0.0083
0.0299	-0.0097	0.0299	-0.0100
0.0094	0.0077	0.0092	0.0075
-0.0136	-0.0039	-0.0135	-0.0042
0.0102	0.0006	0.0100	0.0005
-0.0108	-0.0013	-0.0109	-0.0015
0.0083	0.0024	0.0082	0.0022
-0.0044	-0.0020	-0.0044	-0.0023
0.0001	0.0024	0.0000	0.0021
0.0013	-0.0021	0.0014	-0.0023
-0.0005	0.0019	-0.0006	0.0018
-0.0007	-0.0008	-0.0007	-0.0009
0.0000	-0.0002	-0.0003	-0.0003
0.0004	0.0006	0.0005	0.0002
-0.0002	-0.0004	-0.0003	-0.0005
-0.0002	-0.0003	-0.0004	-0.0002
-0.0002	0.0004	-0.0001	-0.0001
0.0000	0.0003	0.0000	0.0000

Table 4.19 Comparison of sampled impulse responses for channel 3

NAG		Algorithm	
Real	Imaginary	Real	Imaginary
1.0000	0.0000	1.0000	0.0000
0.4608	1.1004	0.4490	1.1037
-0.5824	0.0436	-0.5744	0.0274
0.1573	-0.1729	0.1447	-0.1545
-0.0175	0.0872	0.0021	0.0721
-0.0021	-0.0194	-0.0319	-0.0145
-0.0021	0.0083	0.0298	0.0154
-0.0051	-0.0075	-0.0100	-0.0102
0.0080	0.0054	-0.0202	-0.0352
-0.0039	-0.0035	-0.0080	-0.0344
-0.0001	0.0014	-0.0081	-0.0145
0.0038	-0.0056	0.0030	-0.0053
-0.0009	0.0026	-0.0004	0.0020
0.0023	-0.0027	0.0016	-0.0028
-0.0004	-0.0009	0.0000	-0.0012
0.0013	0.0018	0.0008	0.0014
-0.0016	-0.0013	-0.0013	-0.0014
-0.0003	0.0000	-0.0008	-0.0004
-0.0001	0.0001	0.0002	0.0002
0.0003	0.0002	0.0001	-0.0004

Figure 4.1 Model of the System

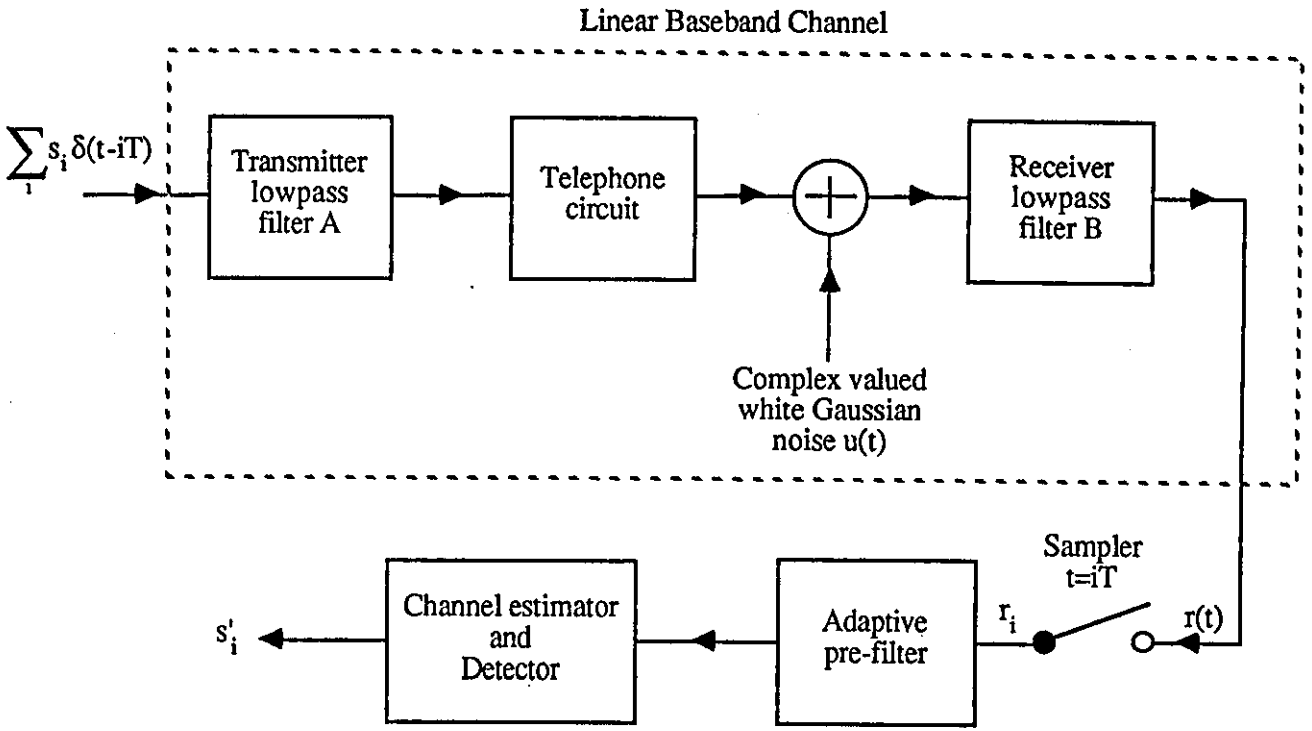


Figure 4.3 Attenuation and Group Delay Characteristics of Telephone Circuit 2

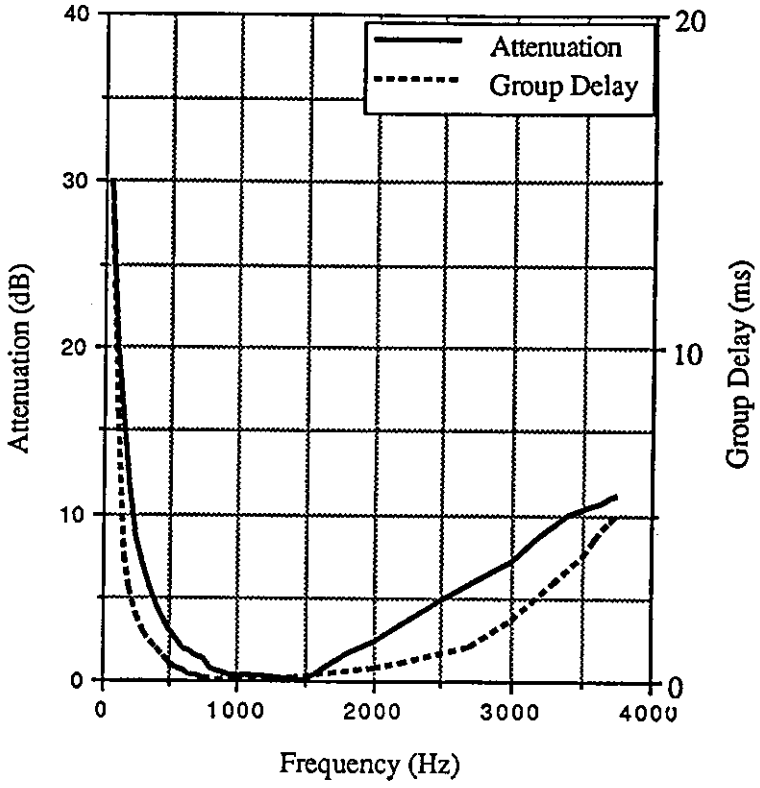


Figure 4.2 Attenuation and Group Delay Characteristics of Telephone Circuit 1

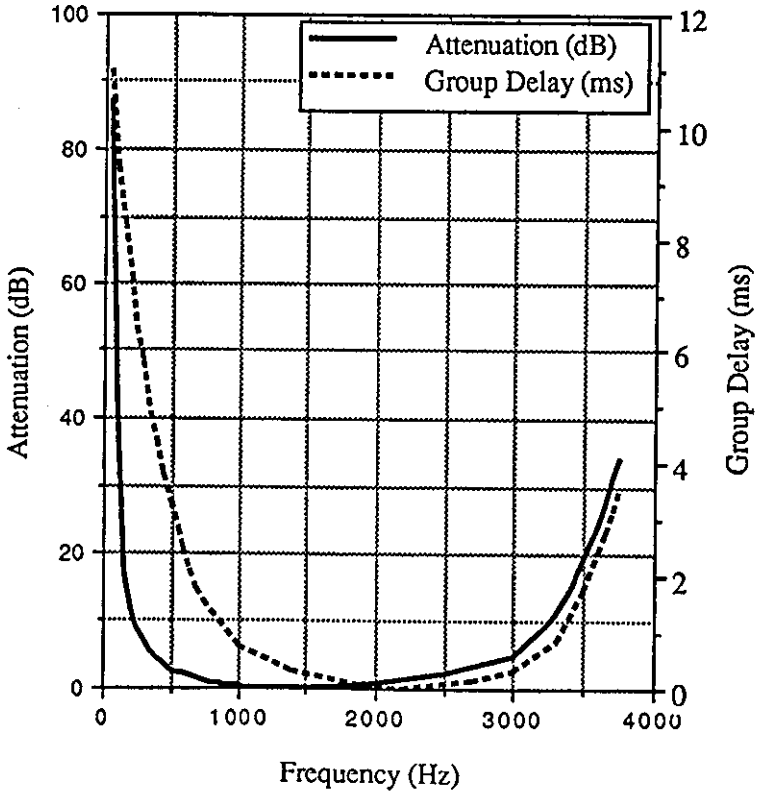


Figure 4.5 Attenuation and Group Delay Characteristics of Telephone Circuit 4

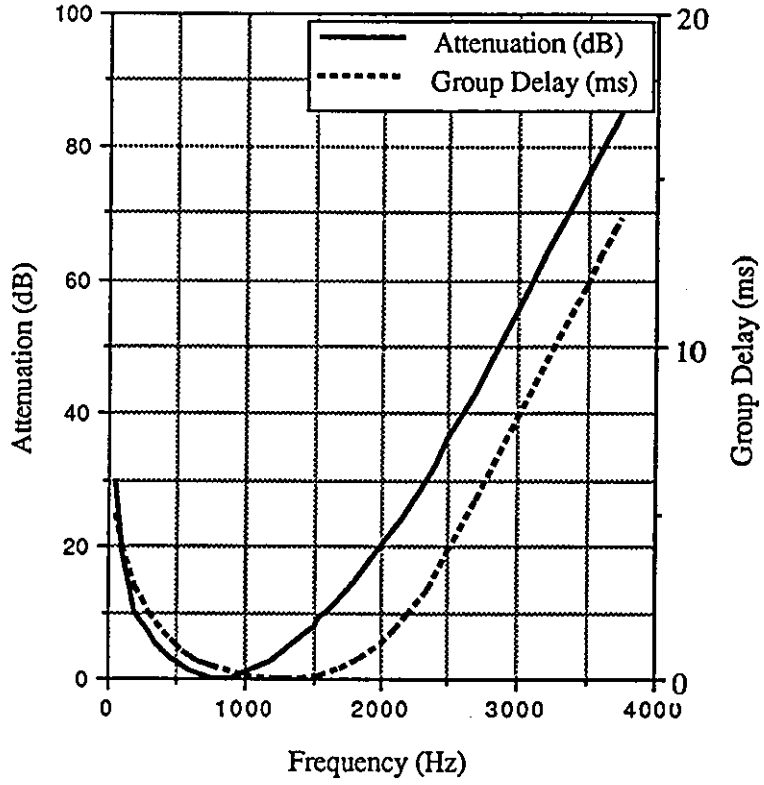


Figure 4.4 Attenuation and Group Delay Characteristics of Telephone Circuit 3

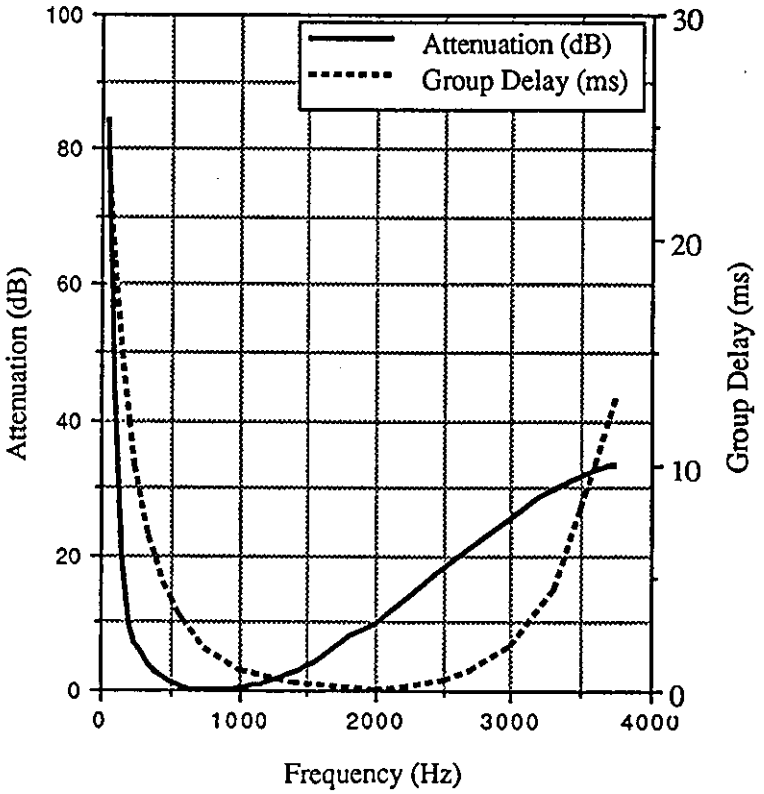


Figure 4.6 Attenuation and Group Delay Characteristics of Equipment Filters

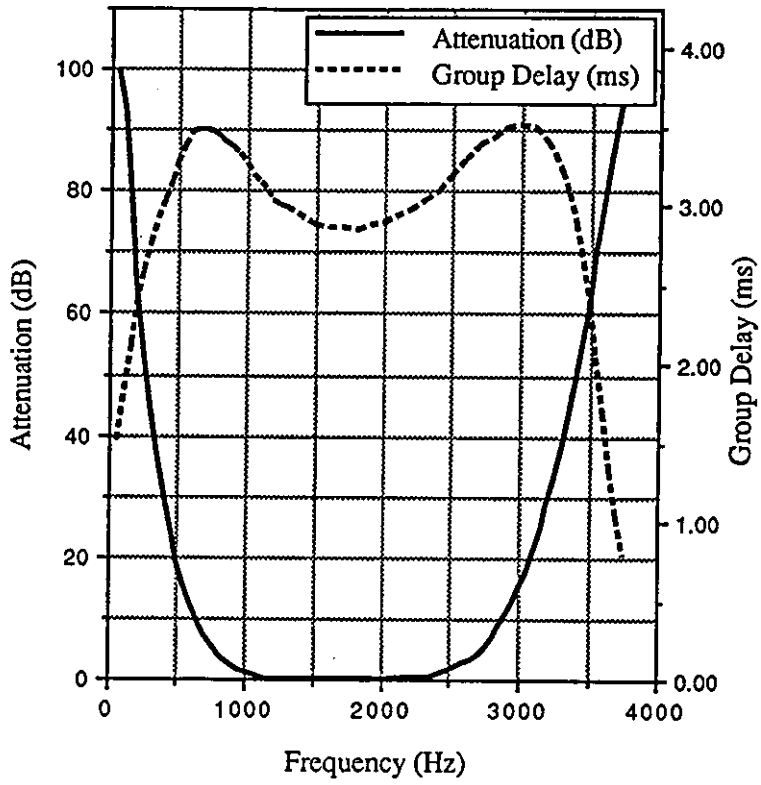


Figure 4.7 Block Diagram of the Linear Feedforward Channel Estimator.

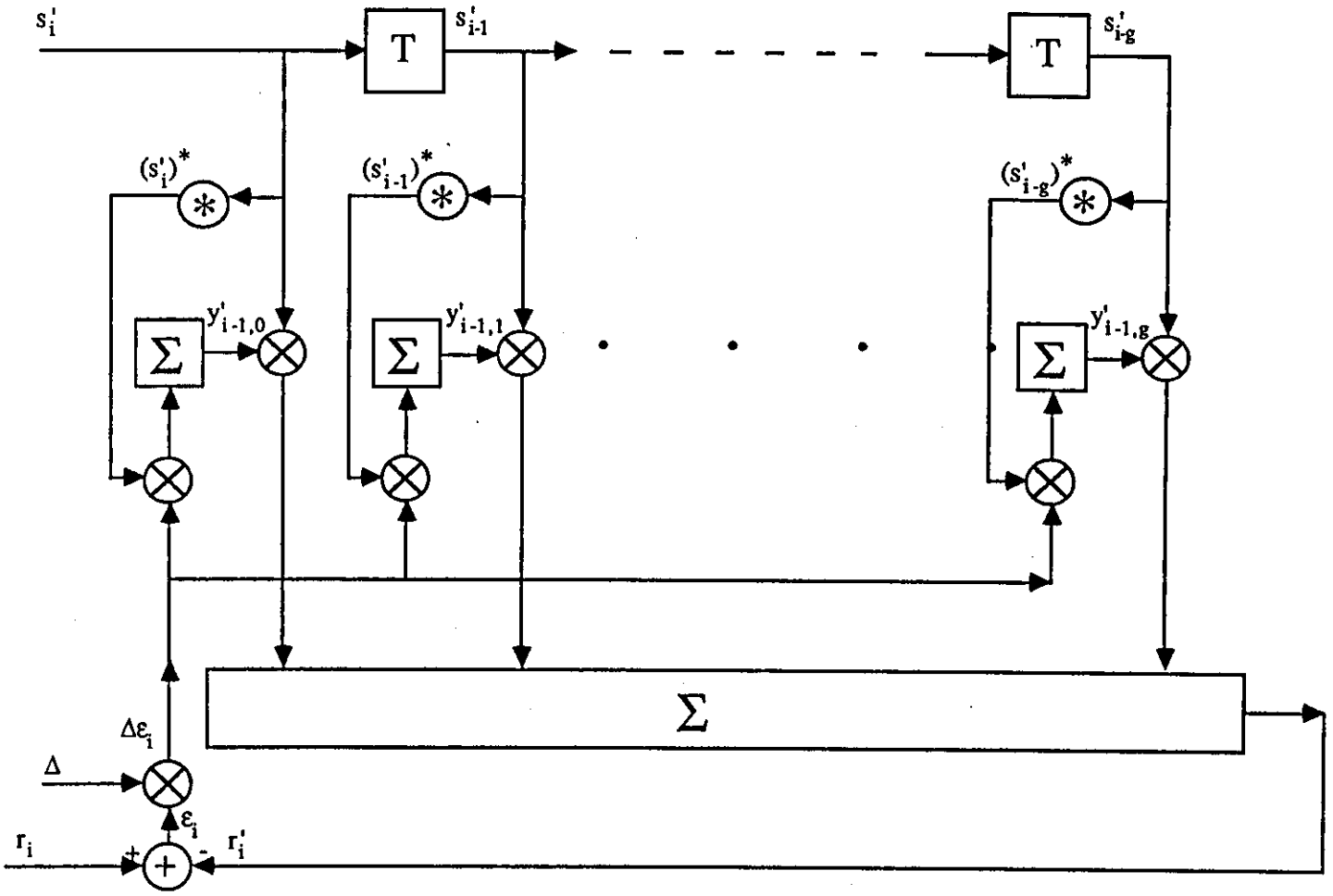


Figure 4.8 Plot Showing the Convergence of the Channel Estimator (SNR=30 dB)

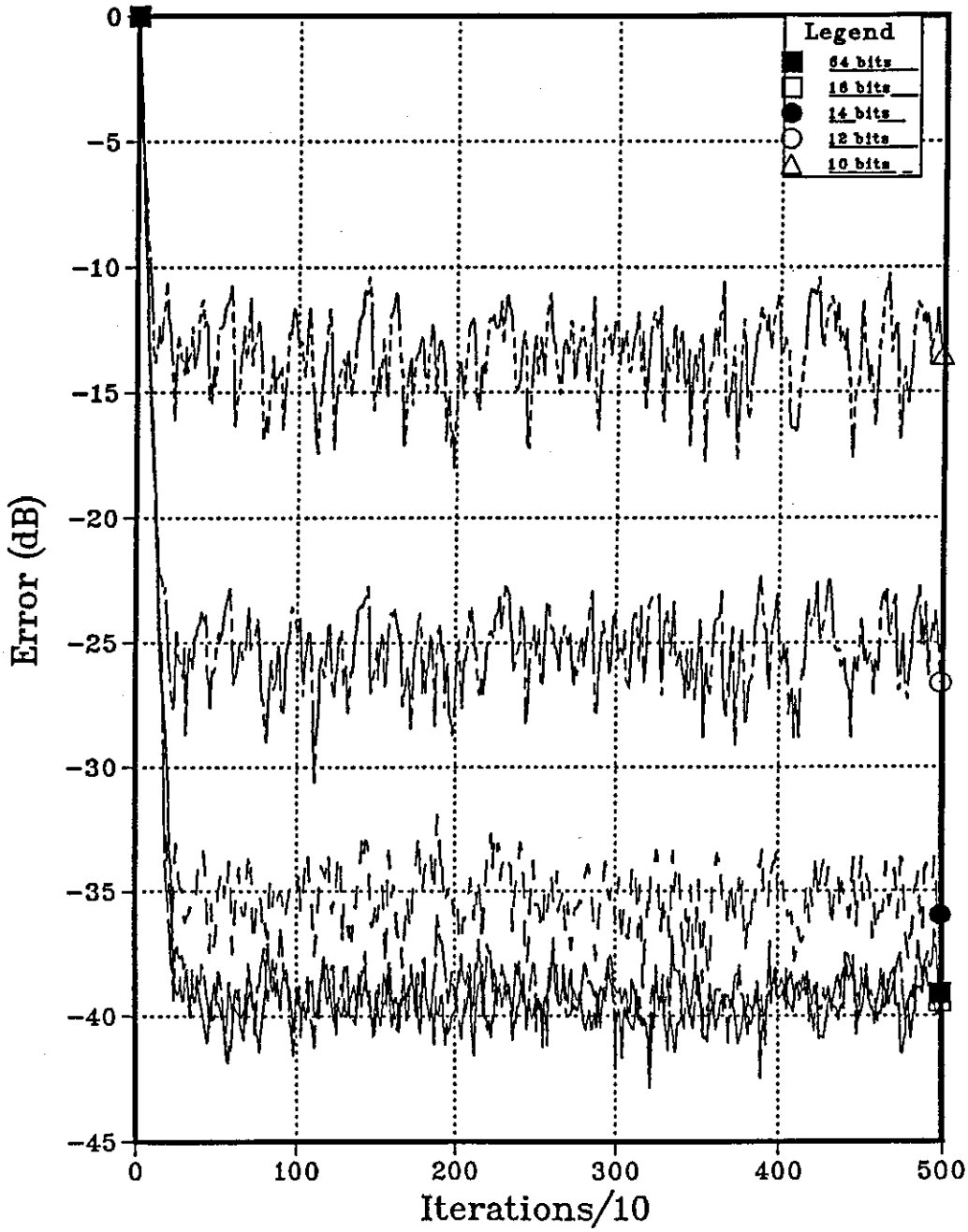


Figure 4.9 Plot of Error in the Estimate of the Sampled impulse response of Channel 1 with a 16-Level QAM signal

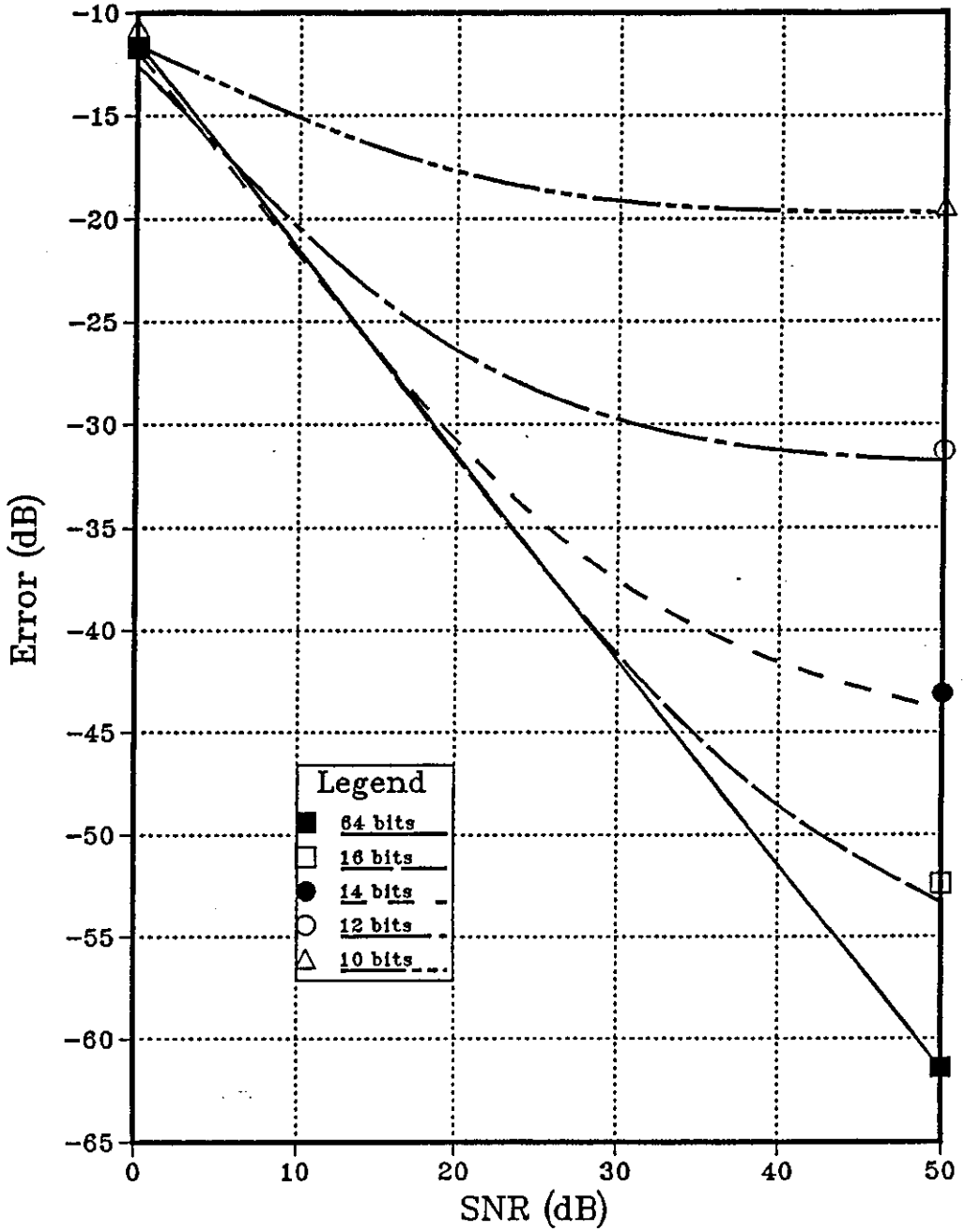


Figure 4.10 Plot of Error in the Estimate of the Sampled impulse response of Channel 4 with a 16-Level QAM signal

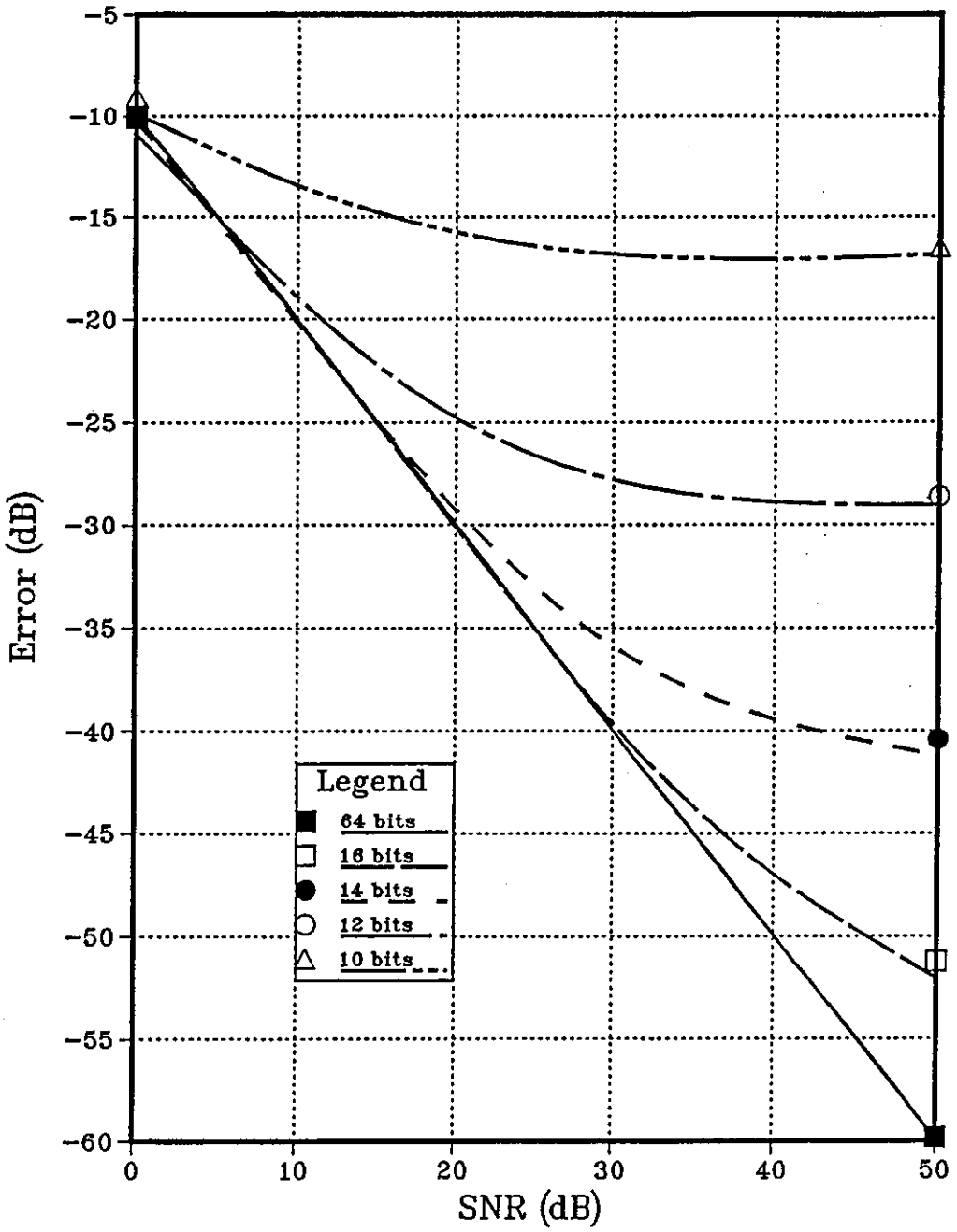


Figure 4.11 Plot of Error in the Estimate of the Sampled impulse response of Channel 1 with a 64-Level QAM signal

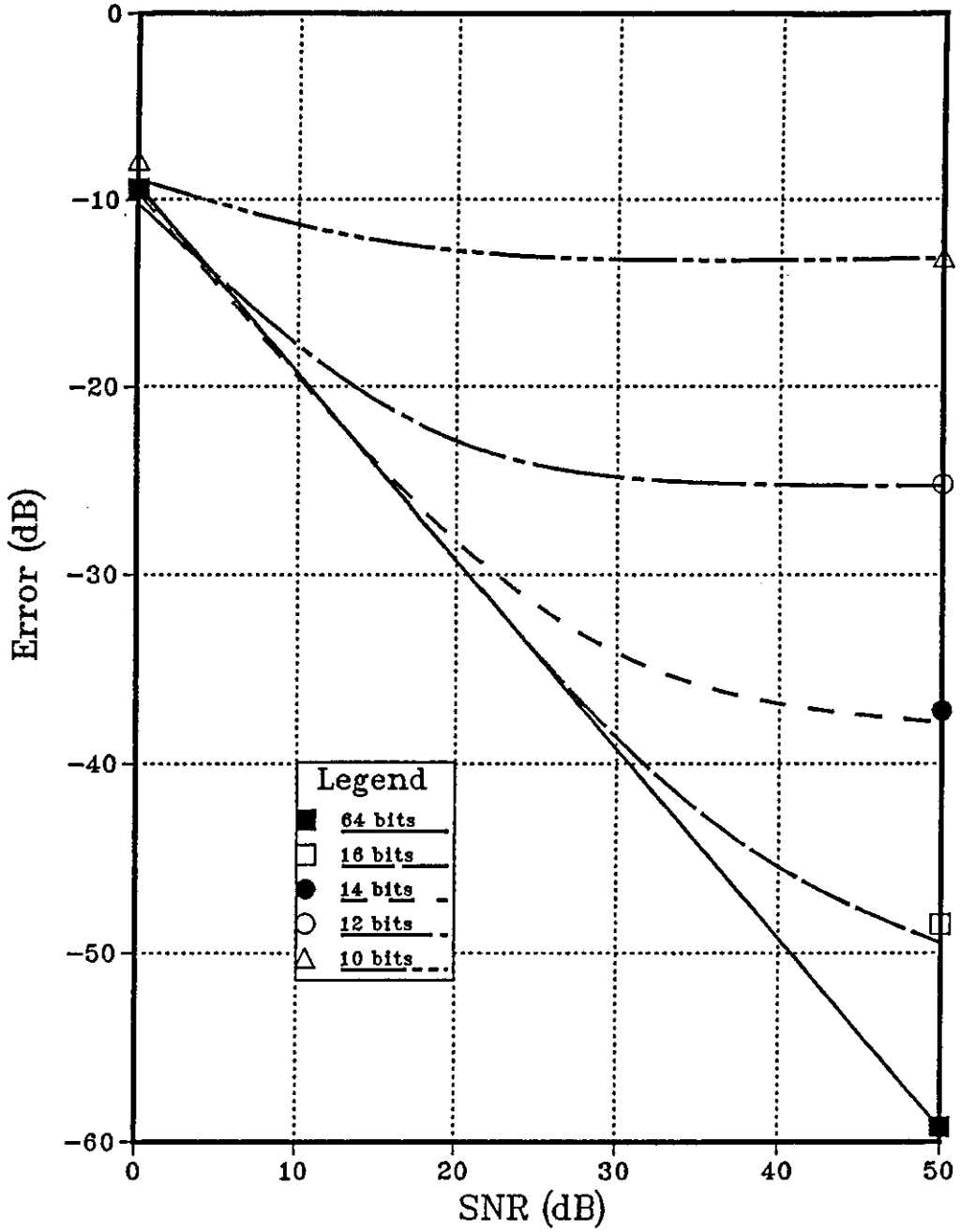


Figure 4.12 Plot of Error in the Estimate of the Sampled impulse response of Channel 4 with a 64-Level QAM signal

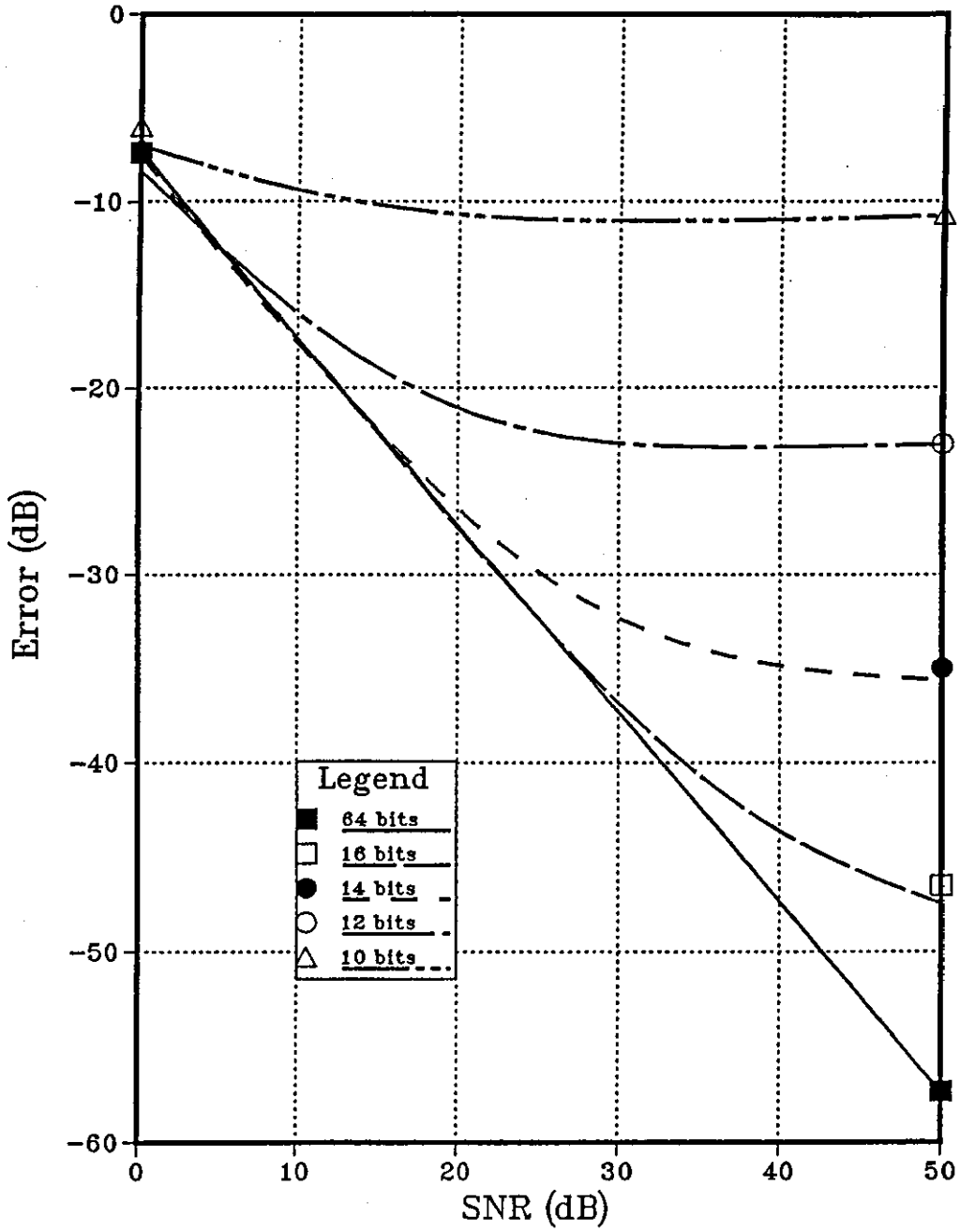


Figure 4.13 Example of moving zeros of Y(z) from outside the unit circle to the reciprocal of their conjugate positions

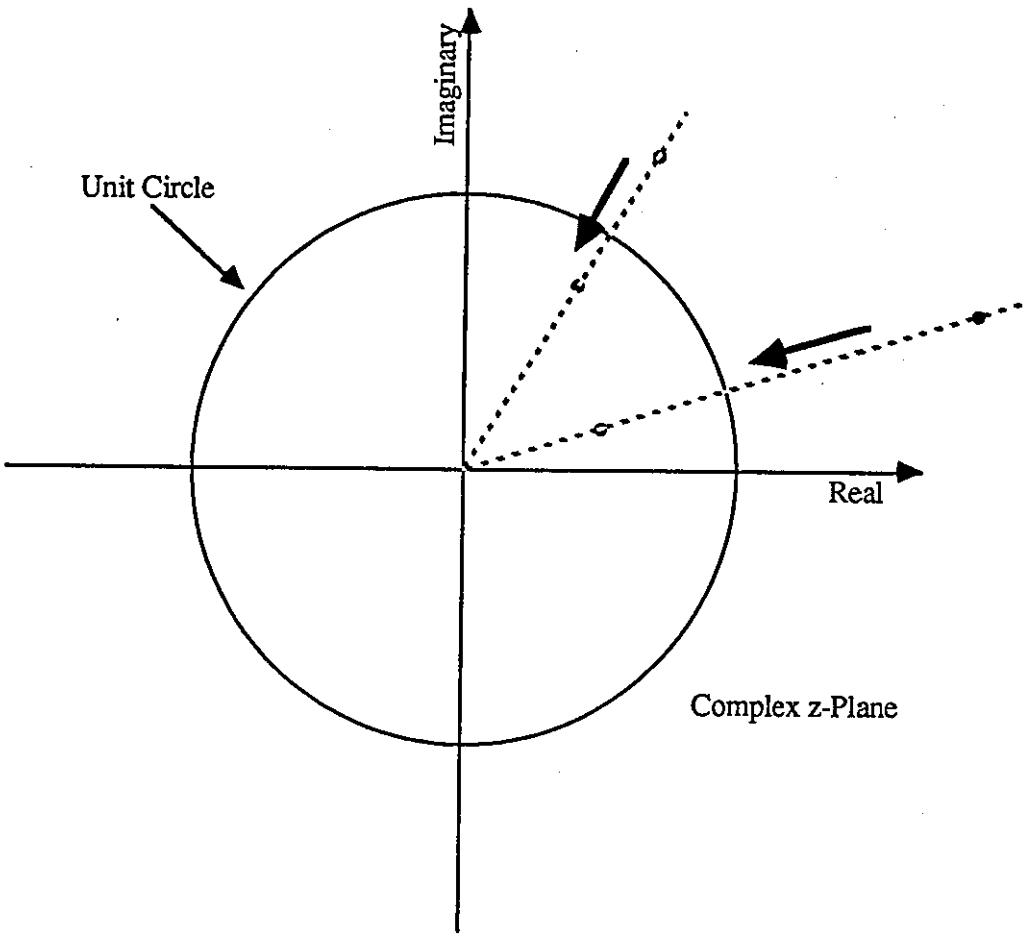


Figure 4.14 One Tap Feedback Filter

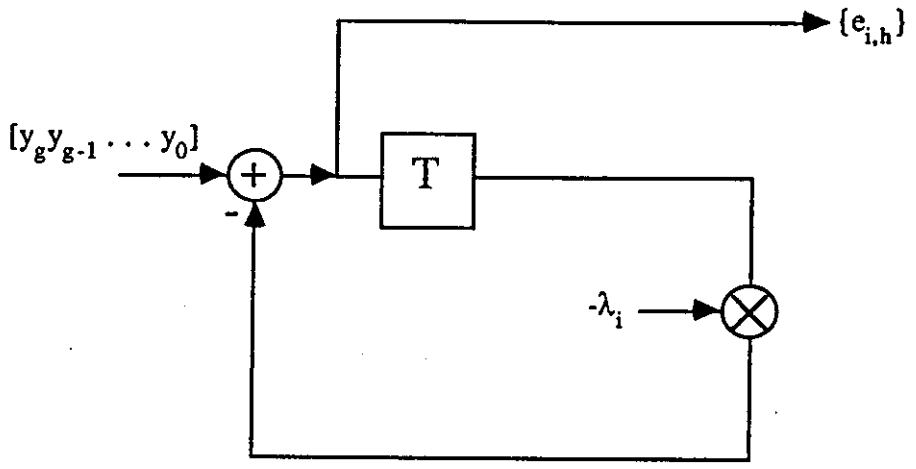


Figure 4.15 Two Tap Feedforward Filter

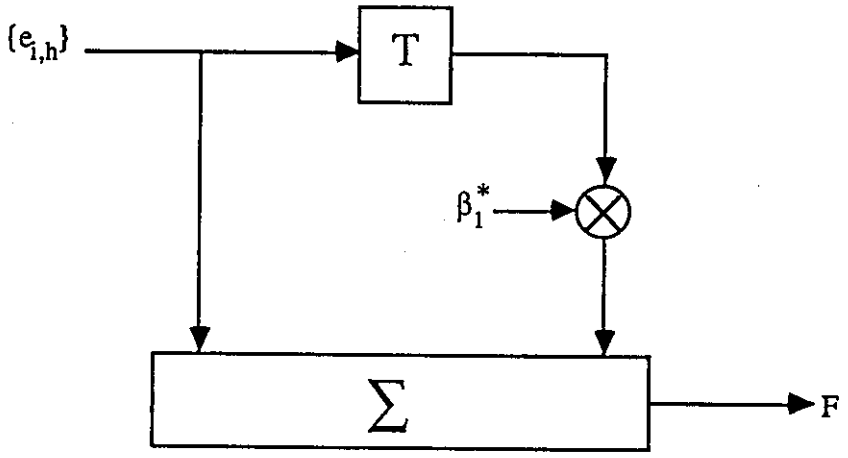


Figure 4.16 Starting positions used for Channels 1 to 4

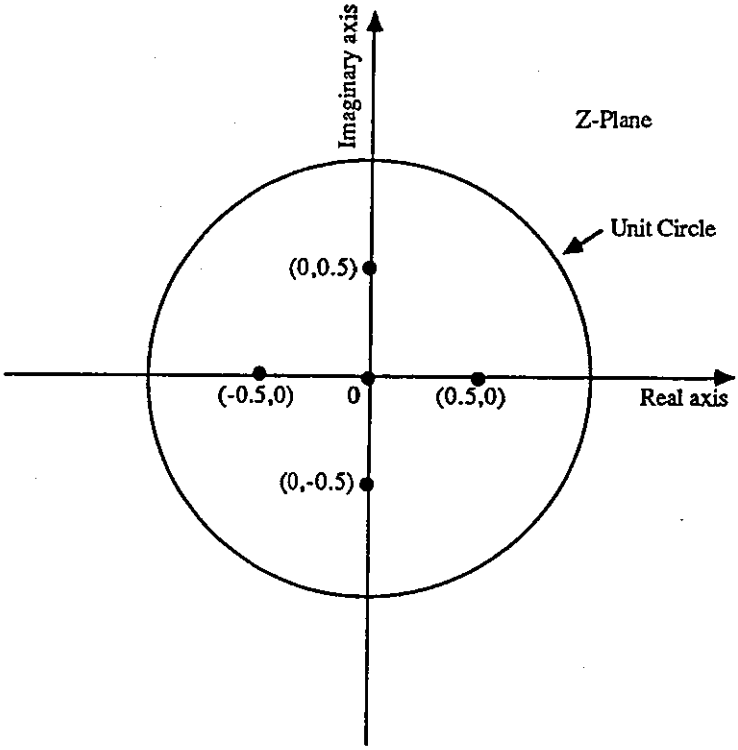


Figure 4.17 Starting positions used for Channels 5 to 8

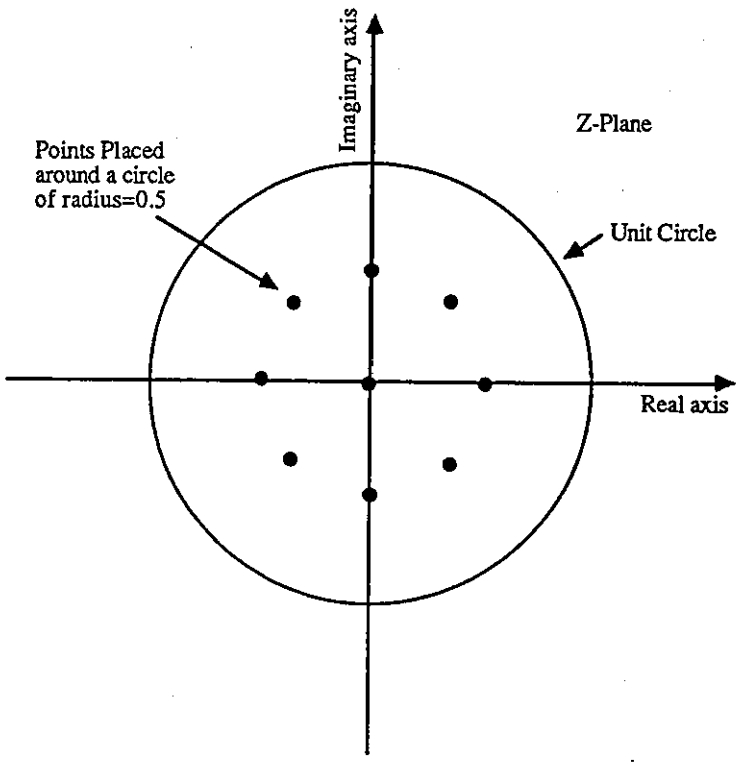


Figure 4.18 Flow Diagram of the Clark-Hau Algorithm

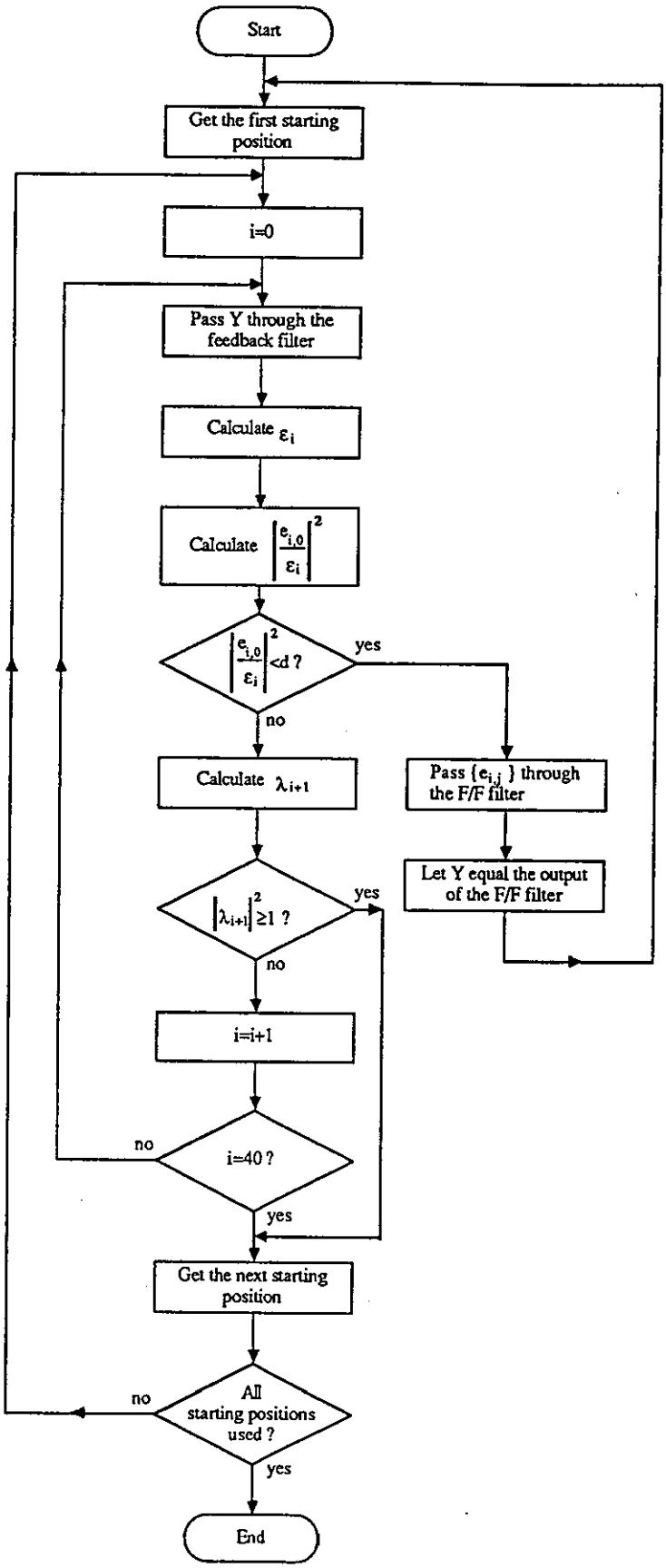


Figure 4.19 Magnitudes of the Components in the Sampled Impulse Response of Channel 1

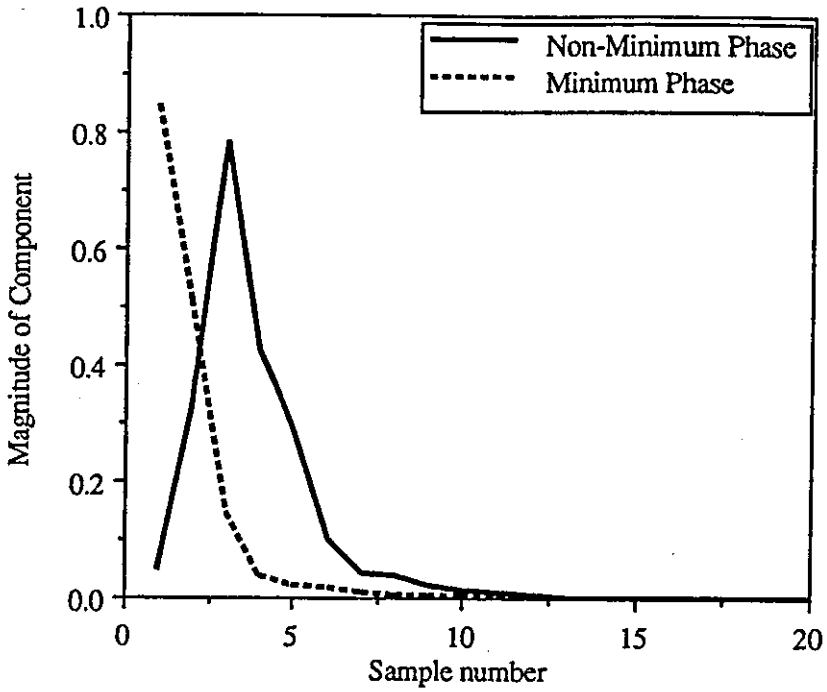


Figure 4.20 Magnitude of the Components in the Sampled Impulse Response of Channel 2

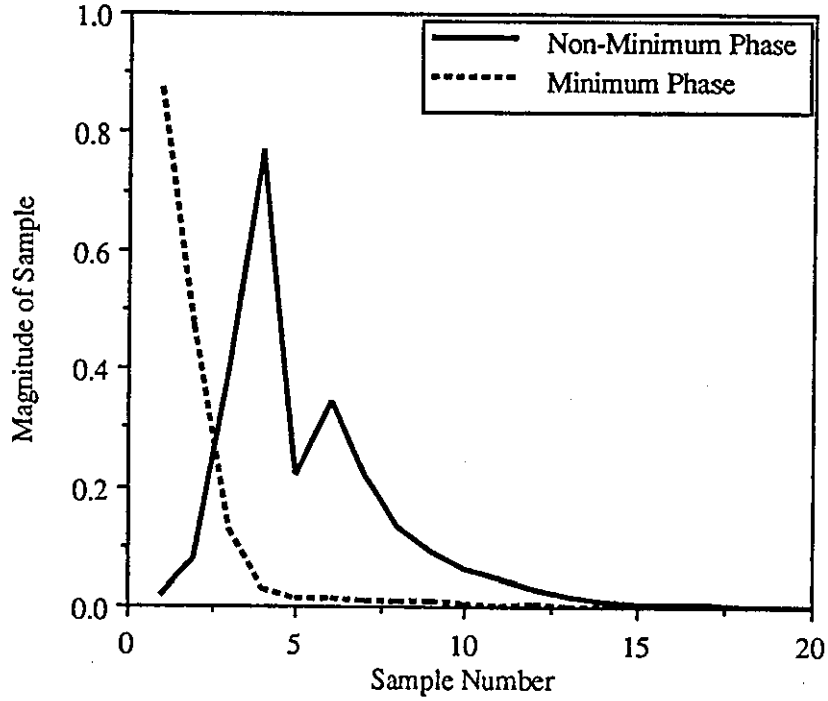


Figure 4.21 Magnitudes of the Components of the Sampled Impulse Response of Channel 3

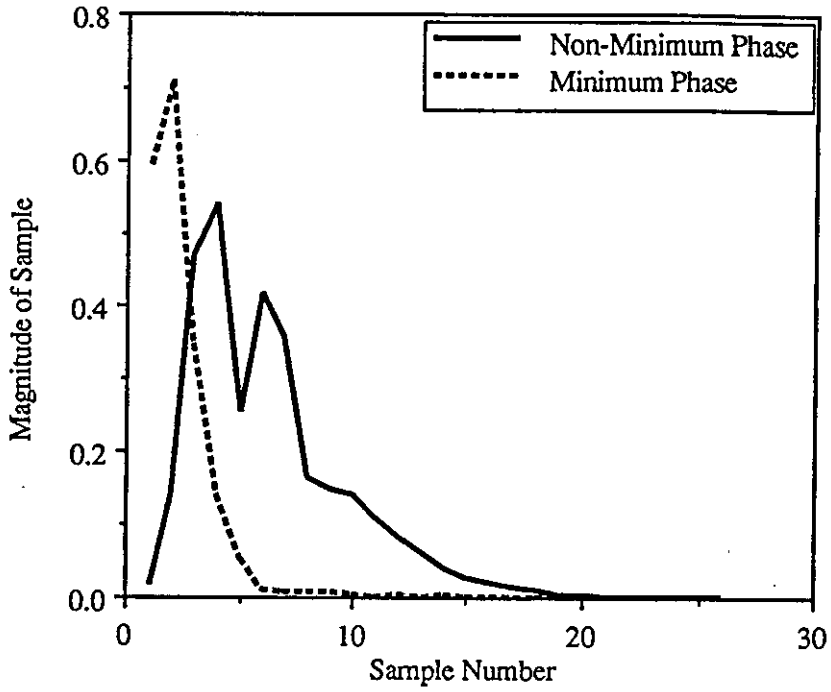


Figure 4.22 Magnitudes of the Components of the Sampled Impulse Response of Channel 4

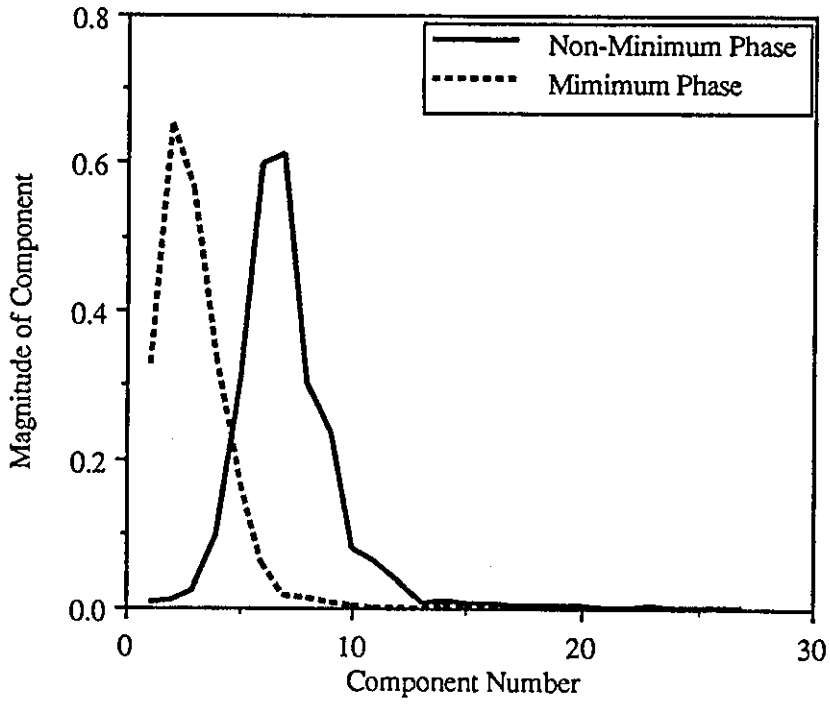


Figure 4.23 Magnitudes of the Components of the Sampled Impulse Response of Channel 5

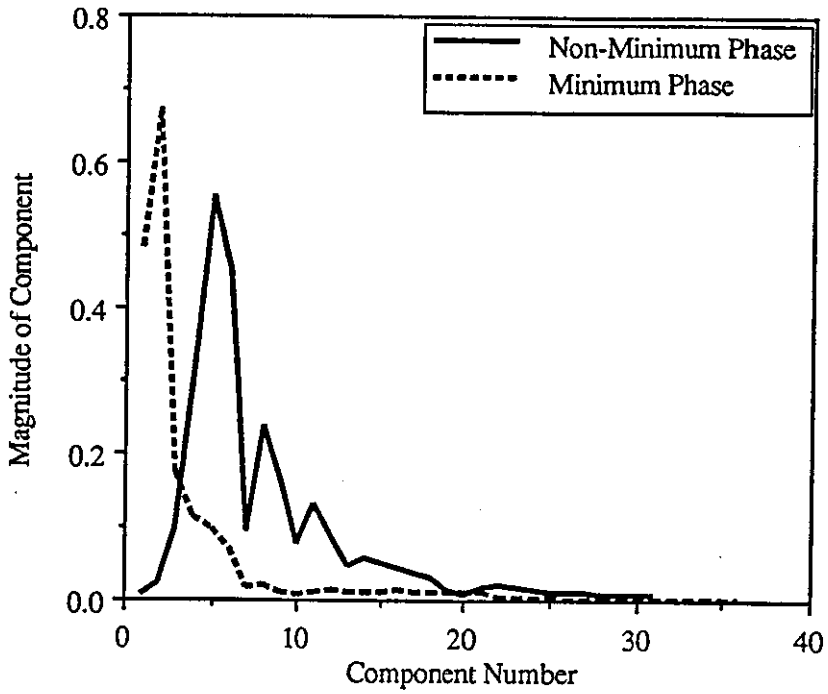


Figure 4.24 Non-Minimum and Minimum Phase Sampled Impulse Responses of Channel 6

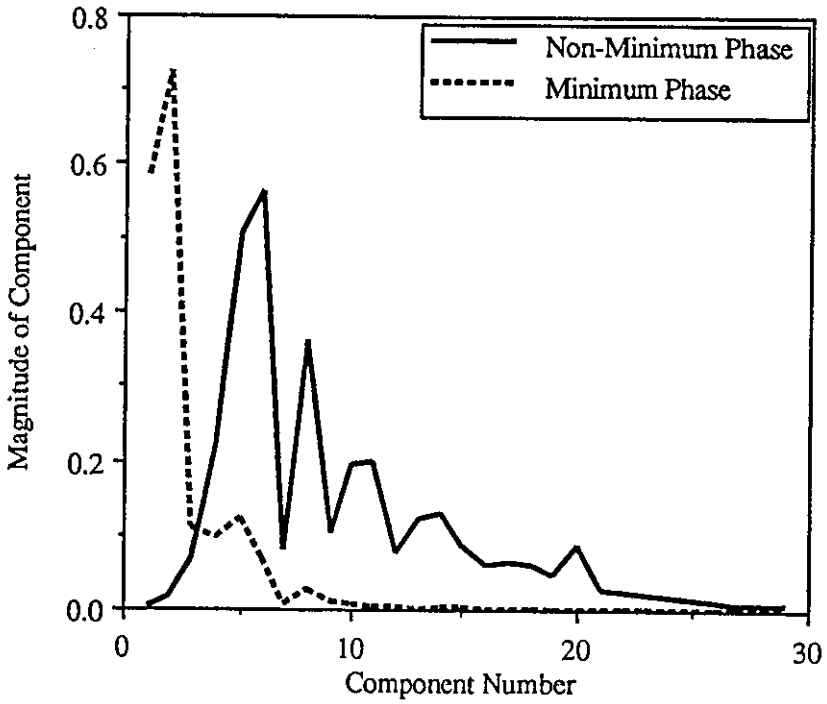


Figure 4.25 Non-Minimum and Minimum Phase Sampled Impulse responses of Channel 7

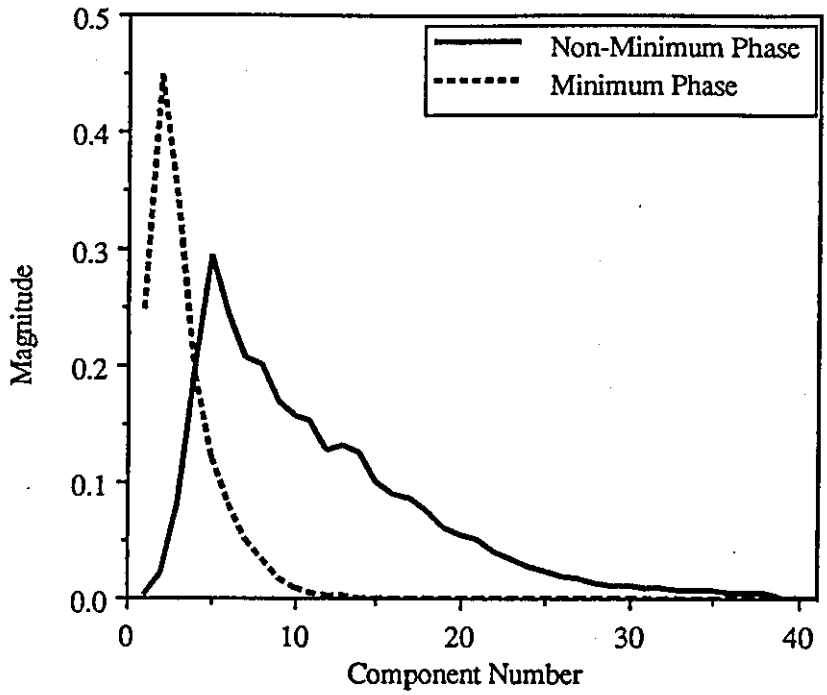


Figure 4.26 Non-Mimimum and Mimimum Phase Sampled impulse Responses of Channel 8

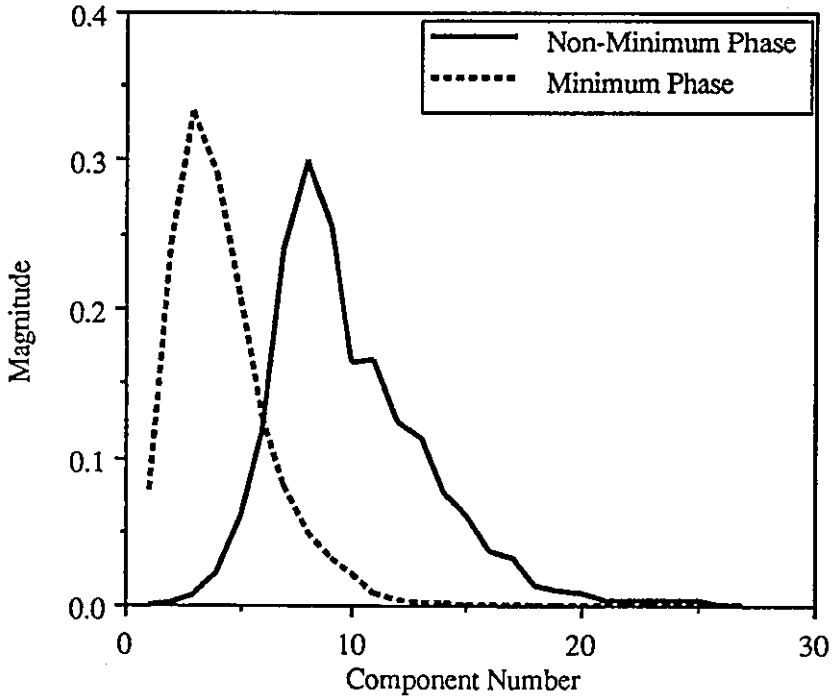


Figure 4.27 Roots positions of channel 1

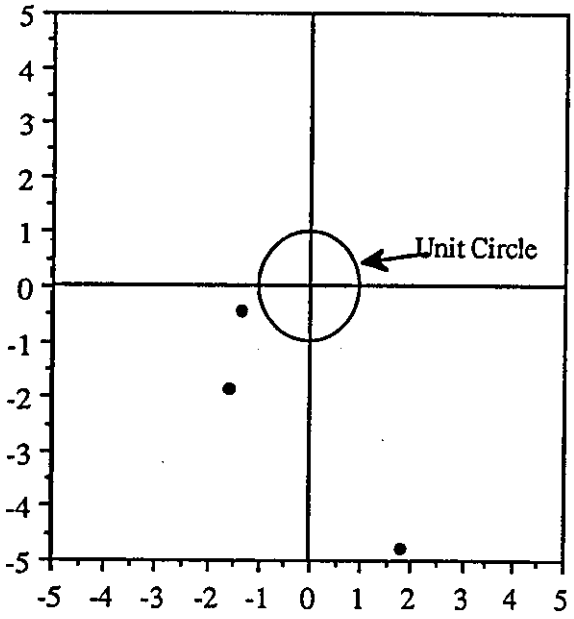


Figure 4.28 Root positions of channel 2

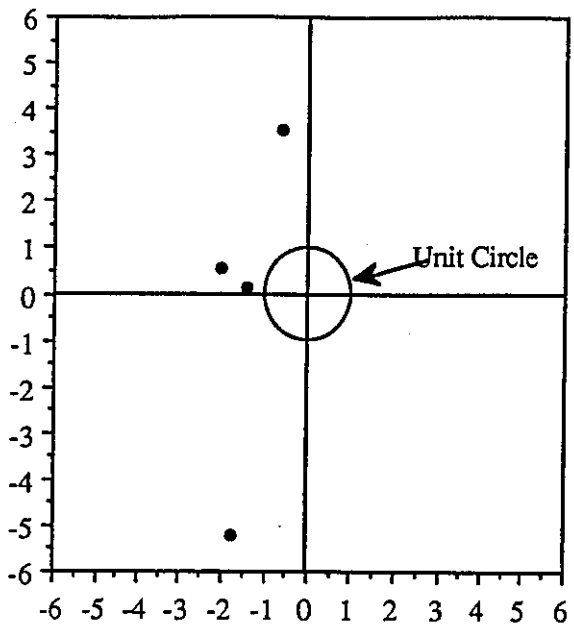


Figure 4.29 Root positions of channel 3

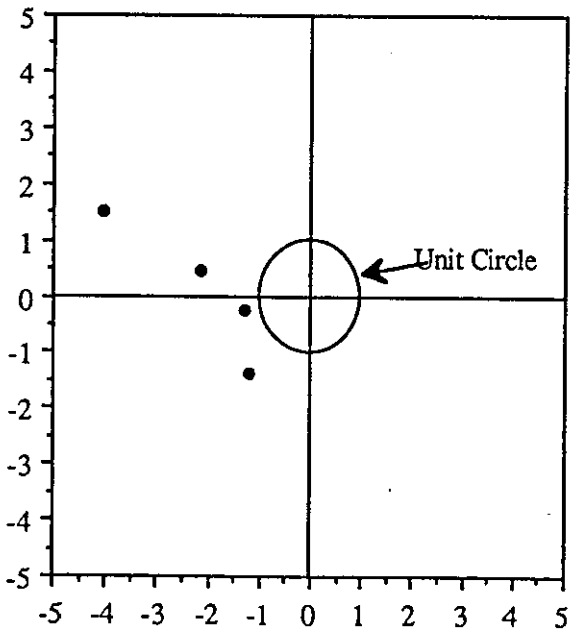


Figure 4.30 Root positions of channel 4

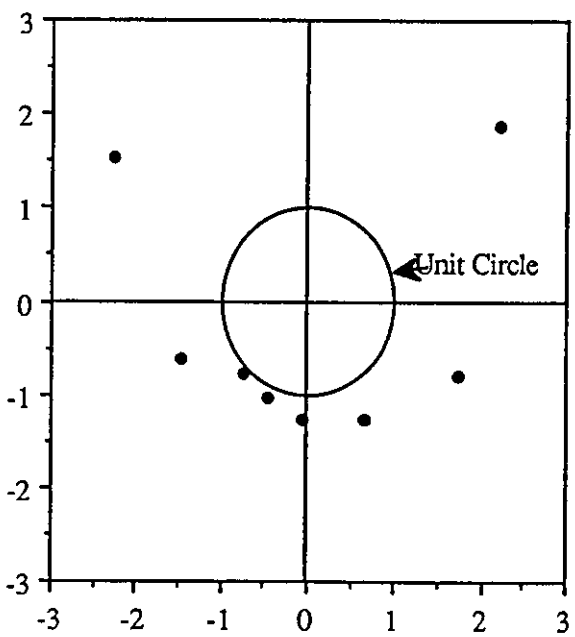


Figure 4.31 Root positions of channel 5

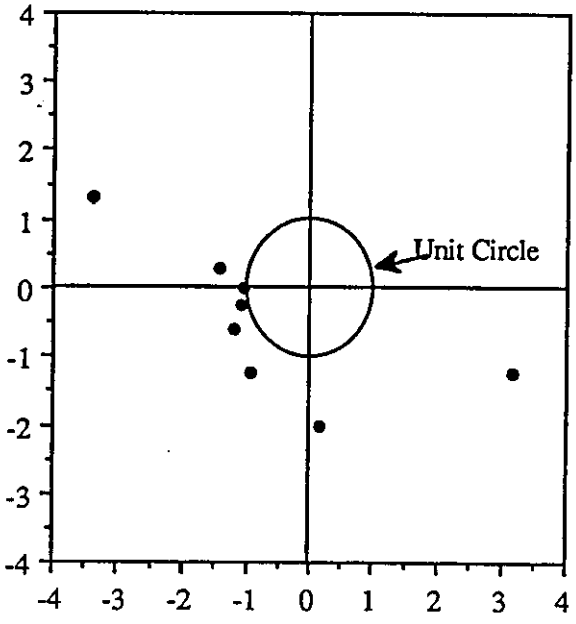


Figure 4.32 Root position of channel 6

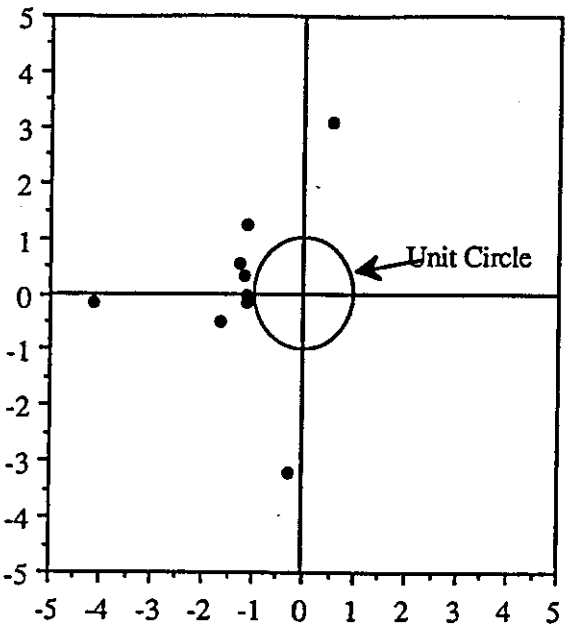


Figure 4.33 Root positions of channel 7

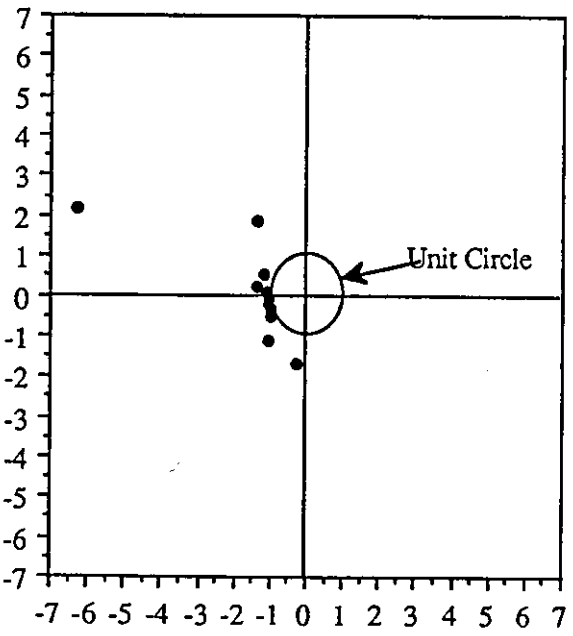


Figure 4.34 Root positions of channel 8

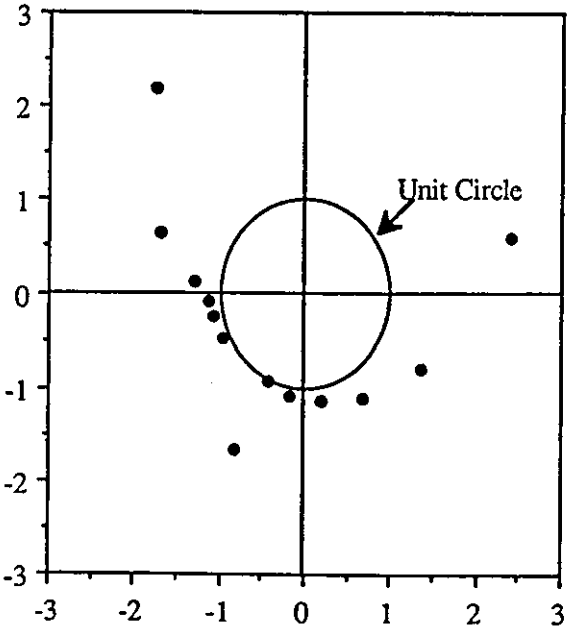


Figure 4.35 Variation of ψ_3 (equation 4.42) with Signal-to-noise ratio for Channel 1

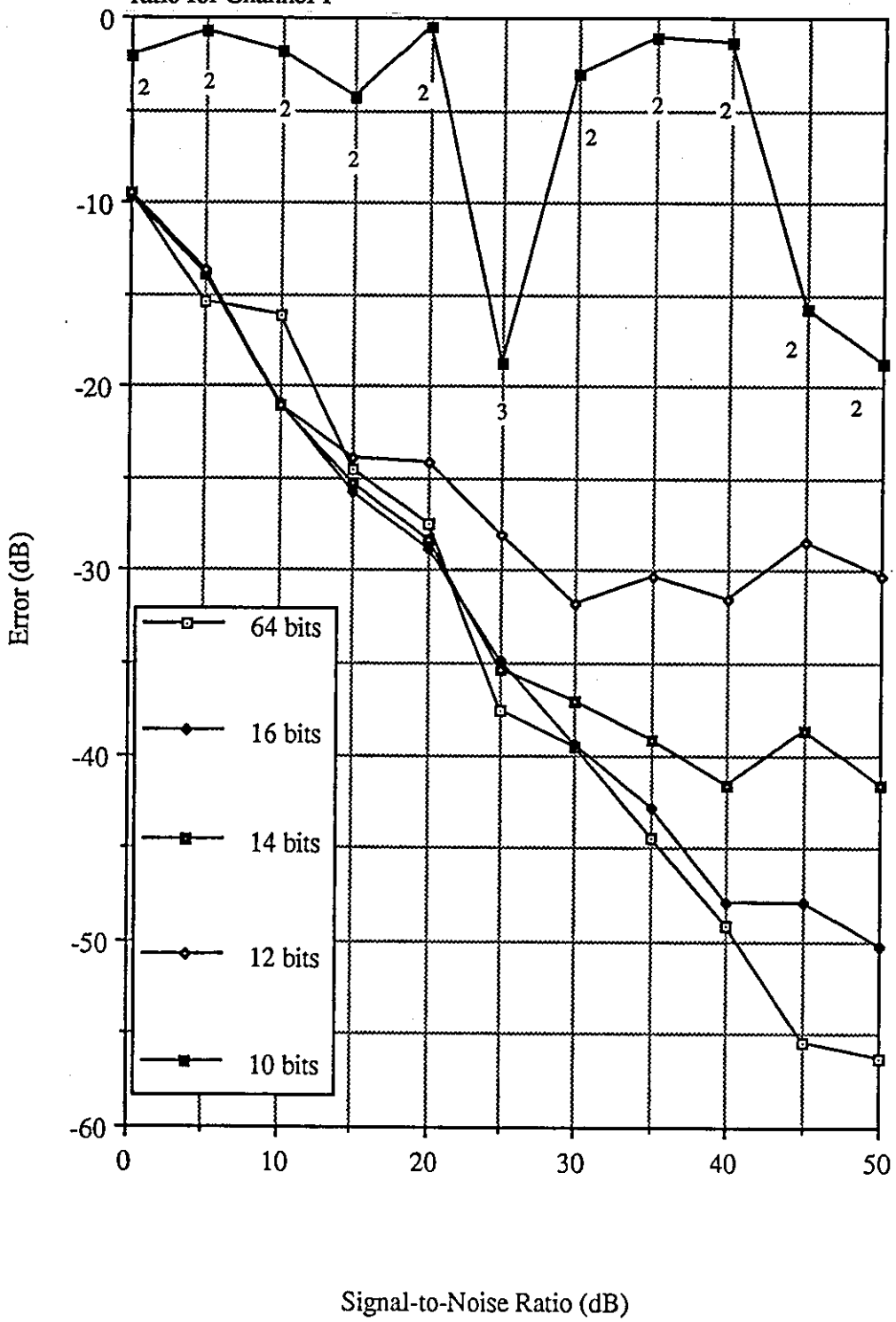


Figure 4.36 Variation of ψ_3 (equation 4.42) with Signal-to-noise ratio for Channel 2

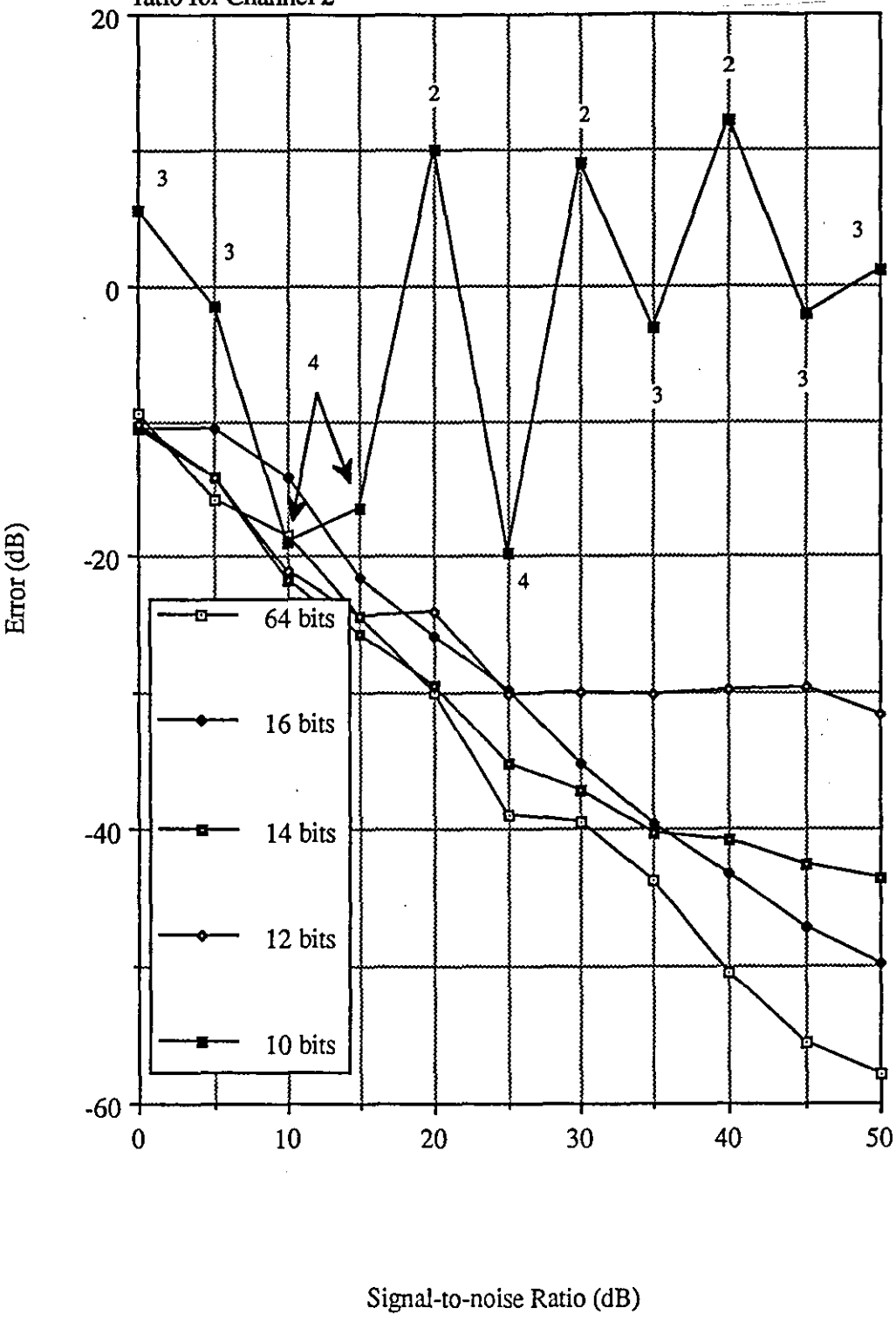


Figure 4.37 Variation of ψ_3 (equation 4.42) with Signal-to-noise ratio for Channel 3

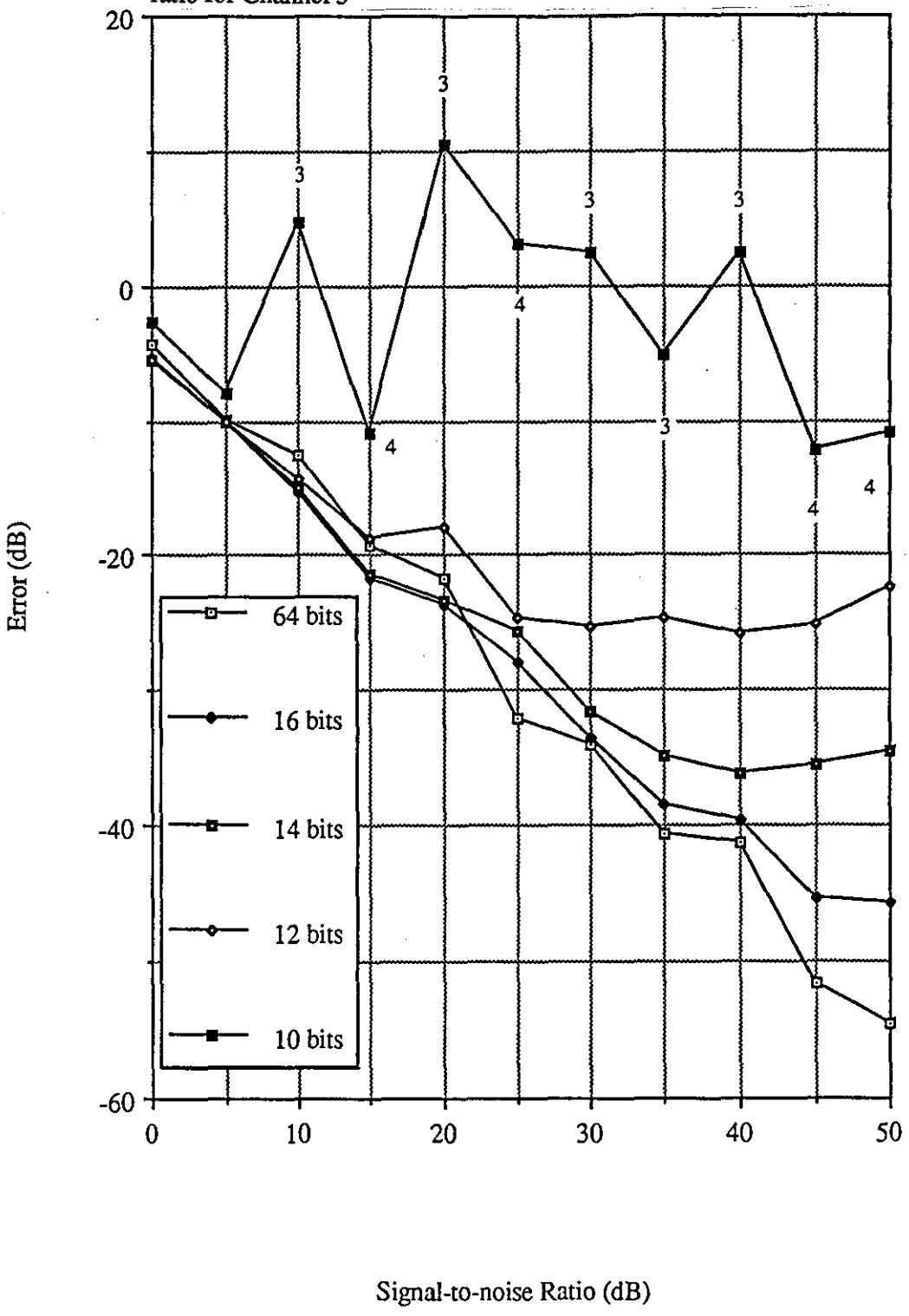


Figure 4.38 Variation of ψ_3 (equation 4.42) with Signal-to-noise ratio for Channel 4

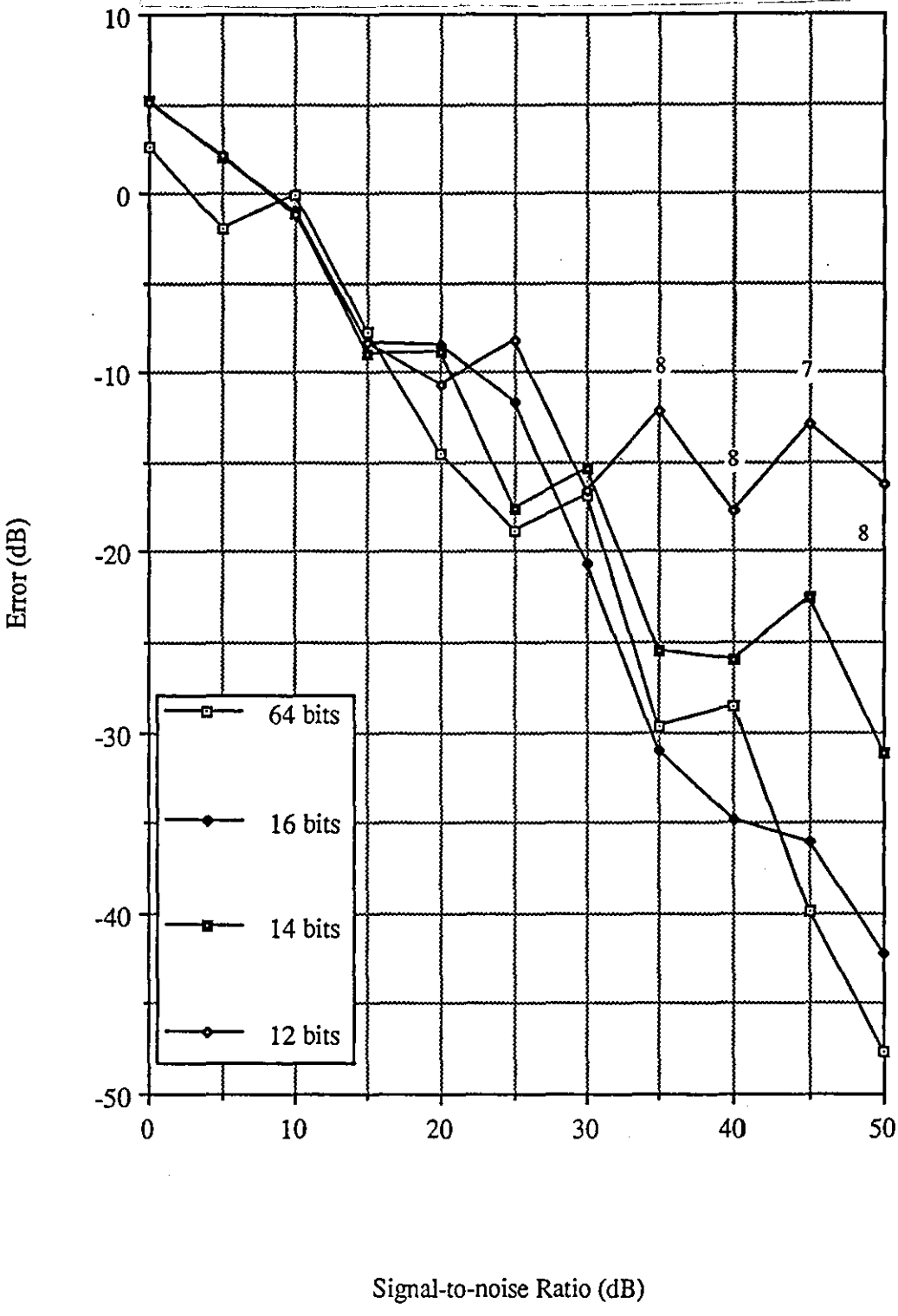


Figure 4.39 Model of the System

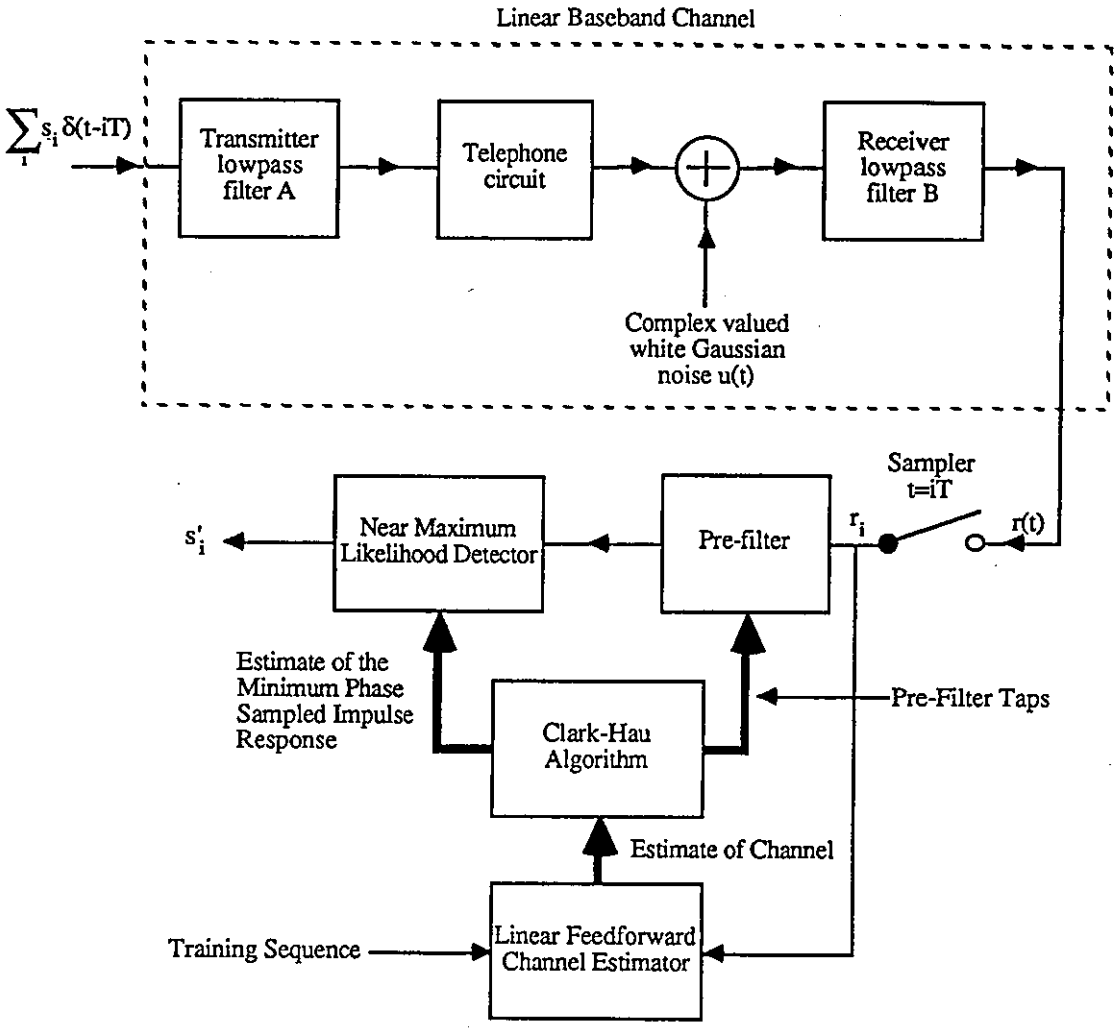


Figure 4.40 Ideal Bit Error Rate Curve for a 16-Level QAM Signal

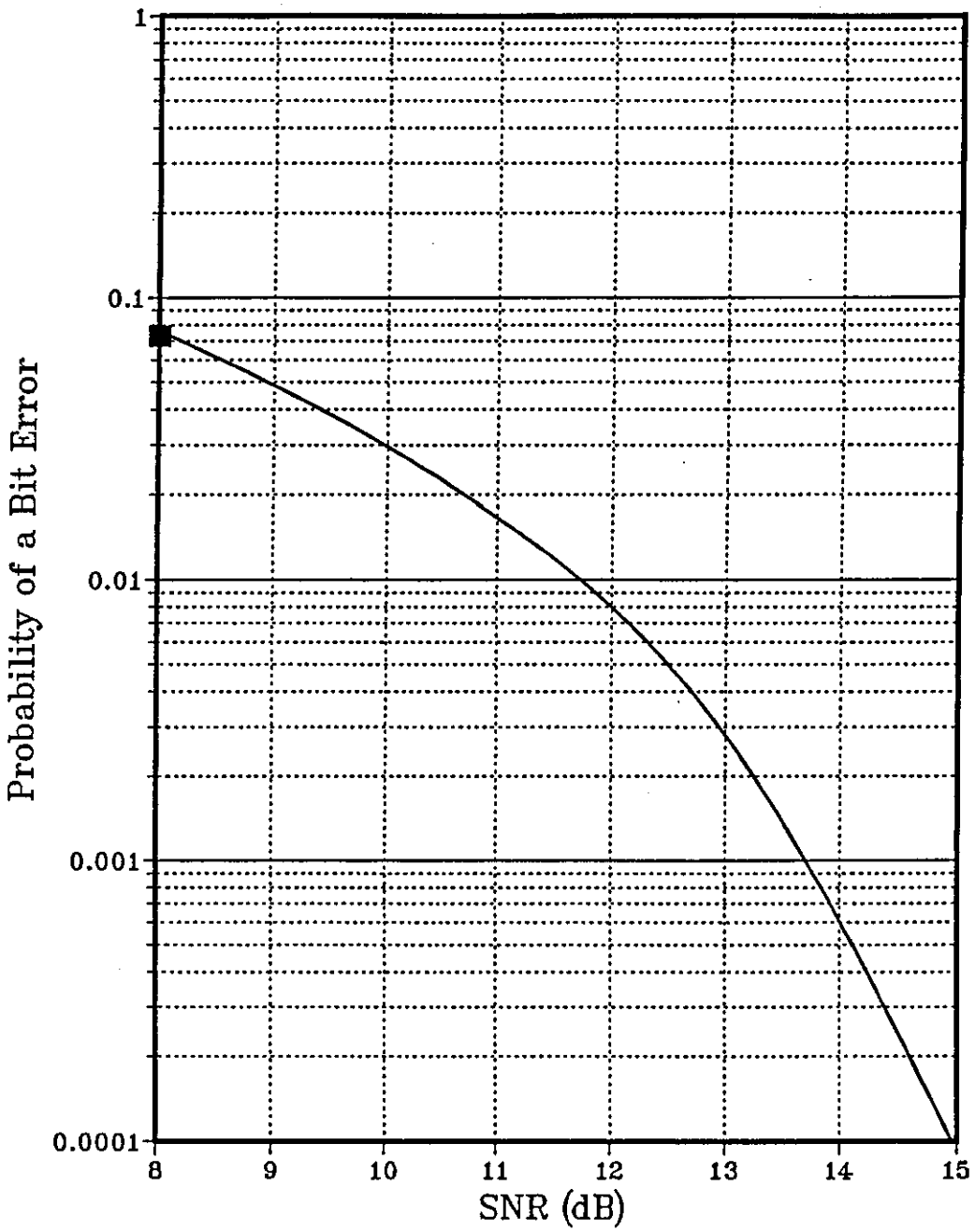


Figure 4.41 Ideal Bit Error Rate Curve for a 64-Level QAM Signal

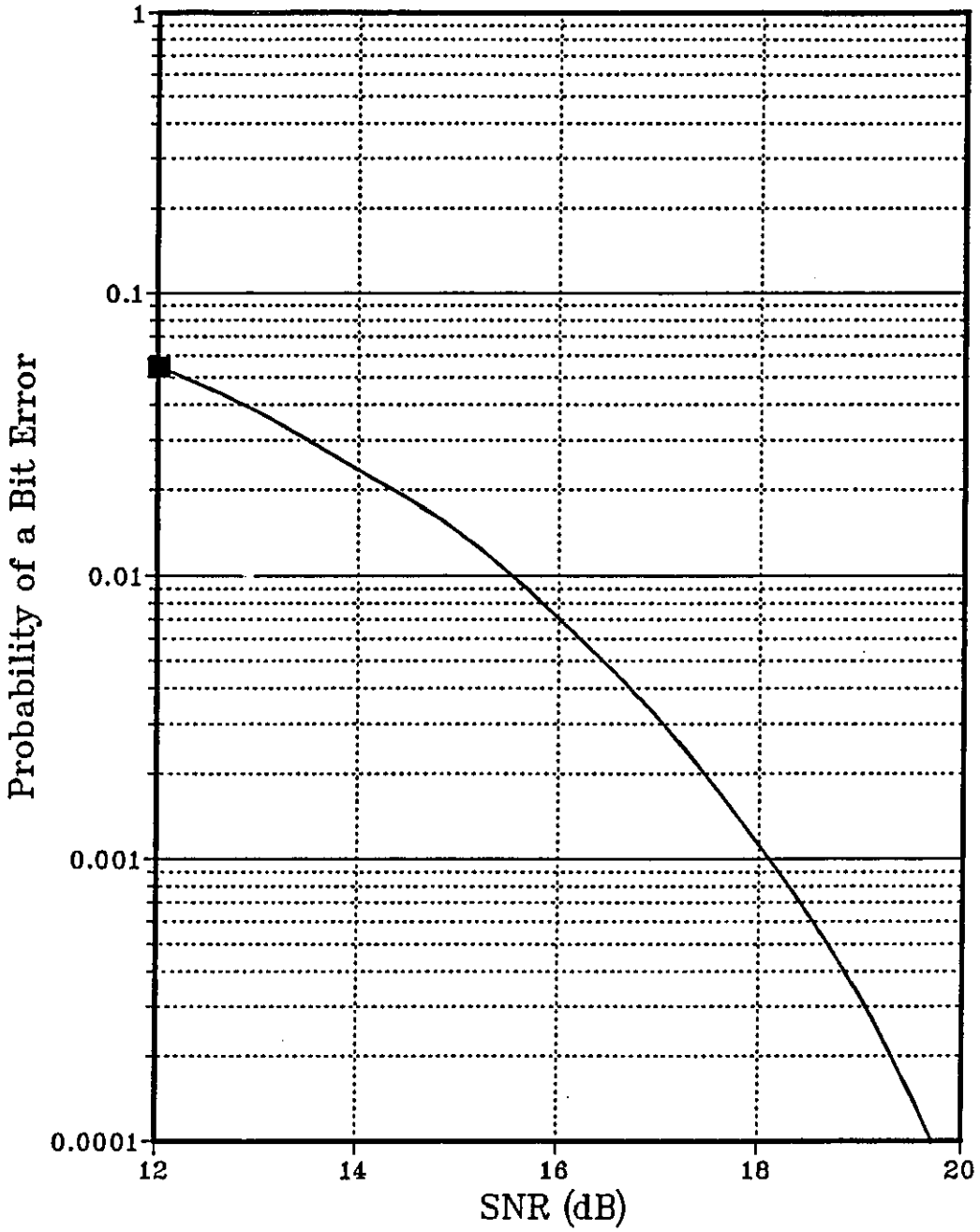


Figure 4.42 Performance of Detector A over Channel 1 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8)

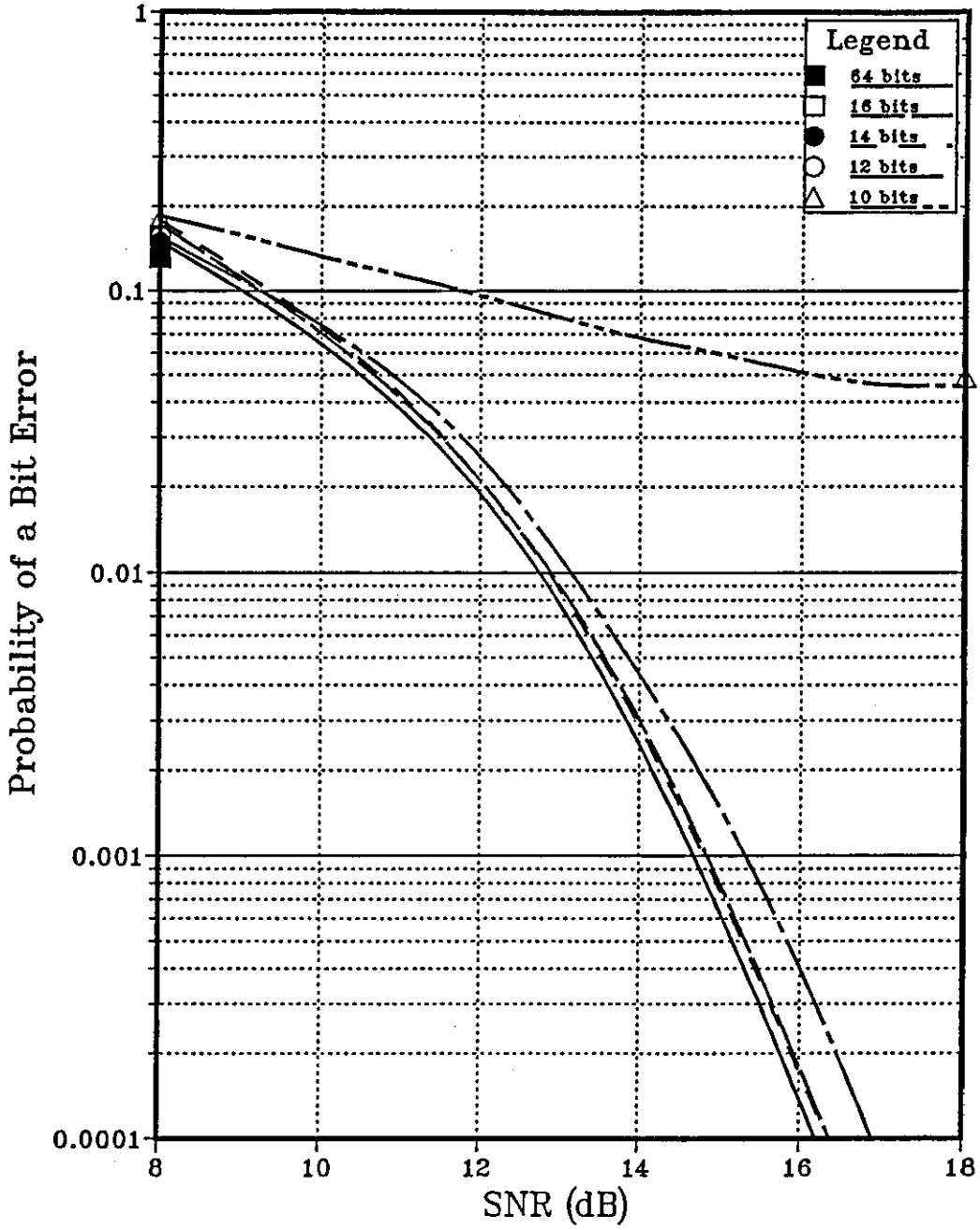


Figure 4.43 Performance of Detector A over Channel 2 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8)

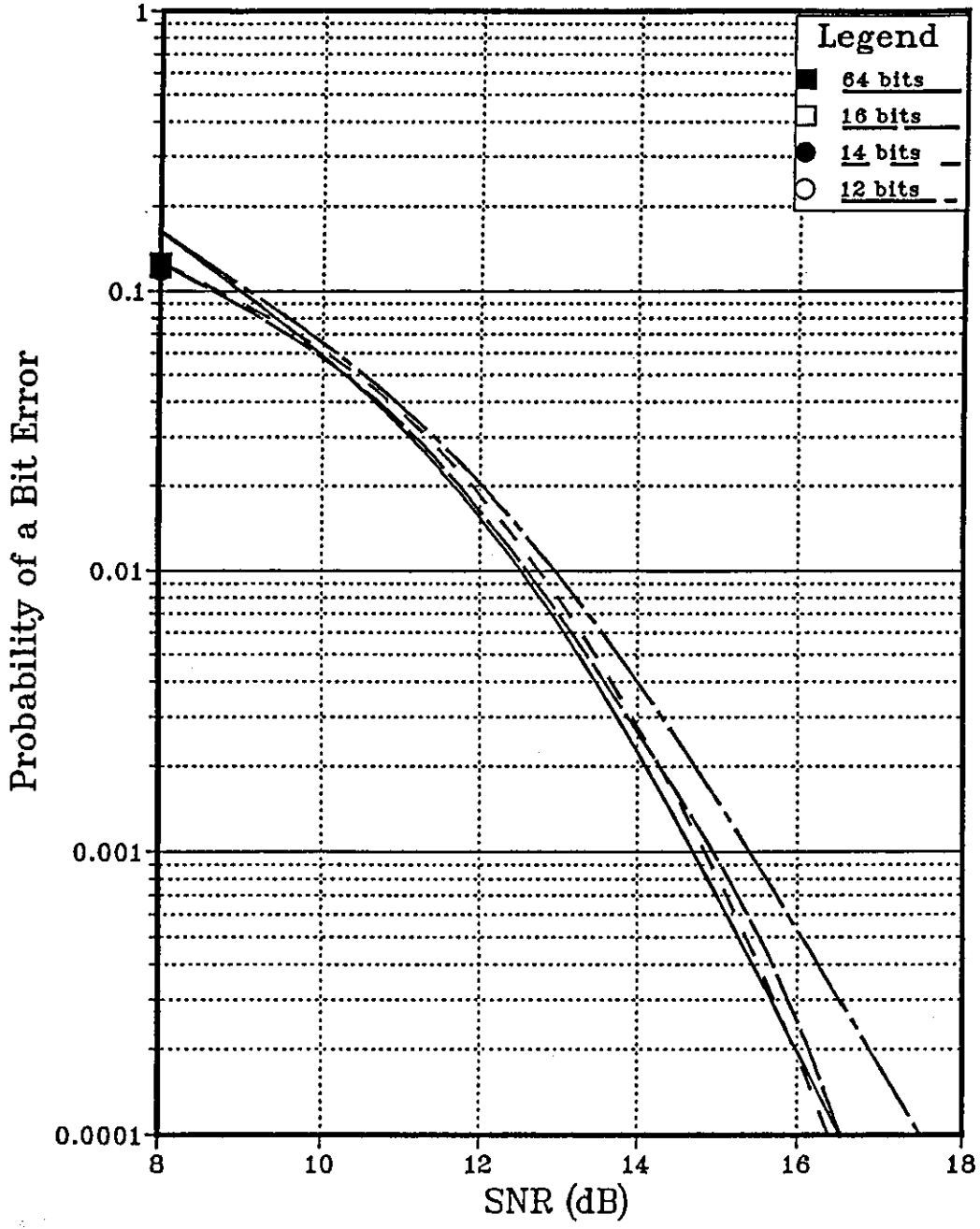


Figure 4.44 Performance of Detector A over Channel 3 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8)

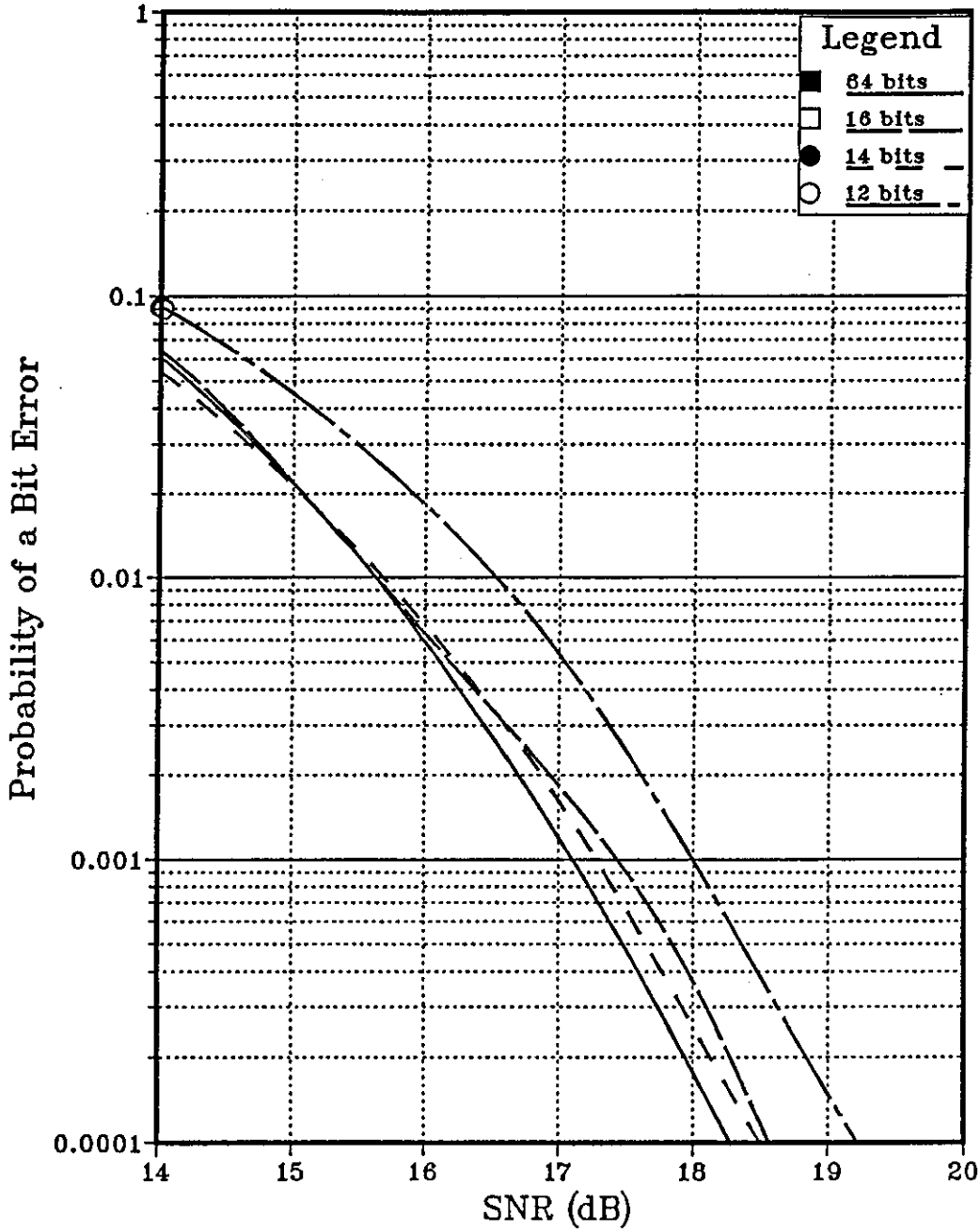


Figure 4.45 Performance of Detector A over Channel 4 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8)

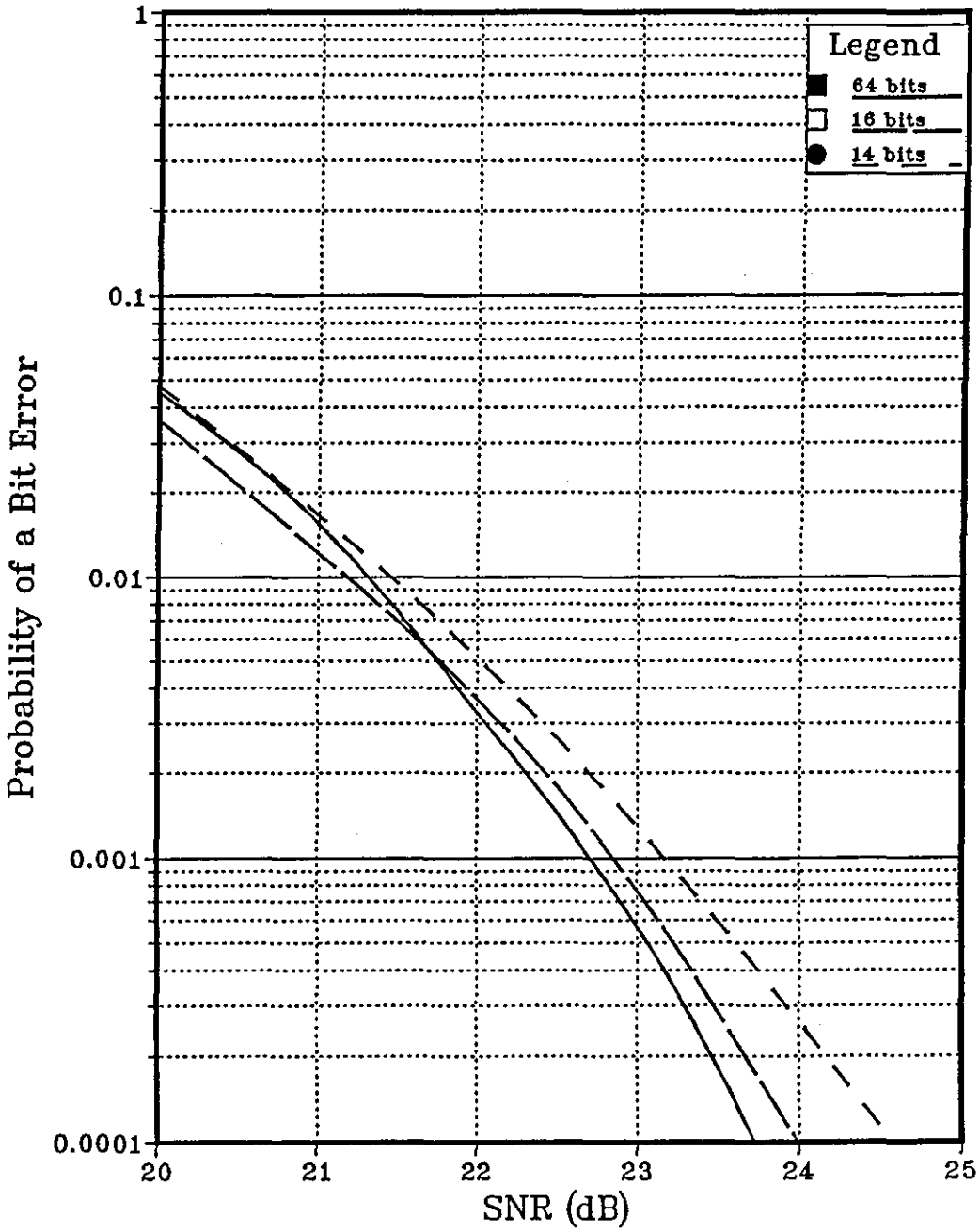


Figure 4.46 Performance of System without Pre-filter and using 64-Bit Floating Point Arithmetic

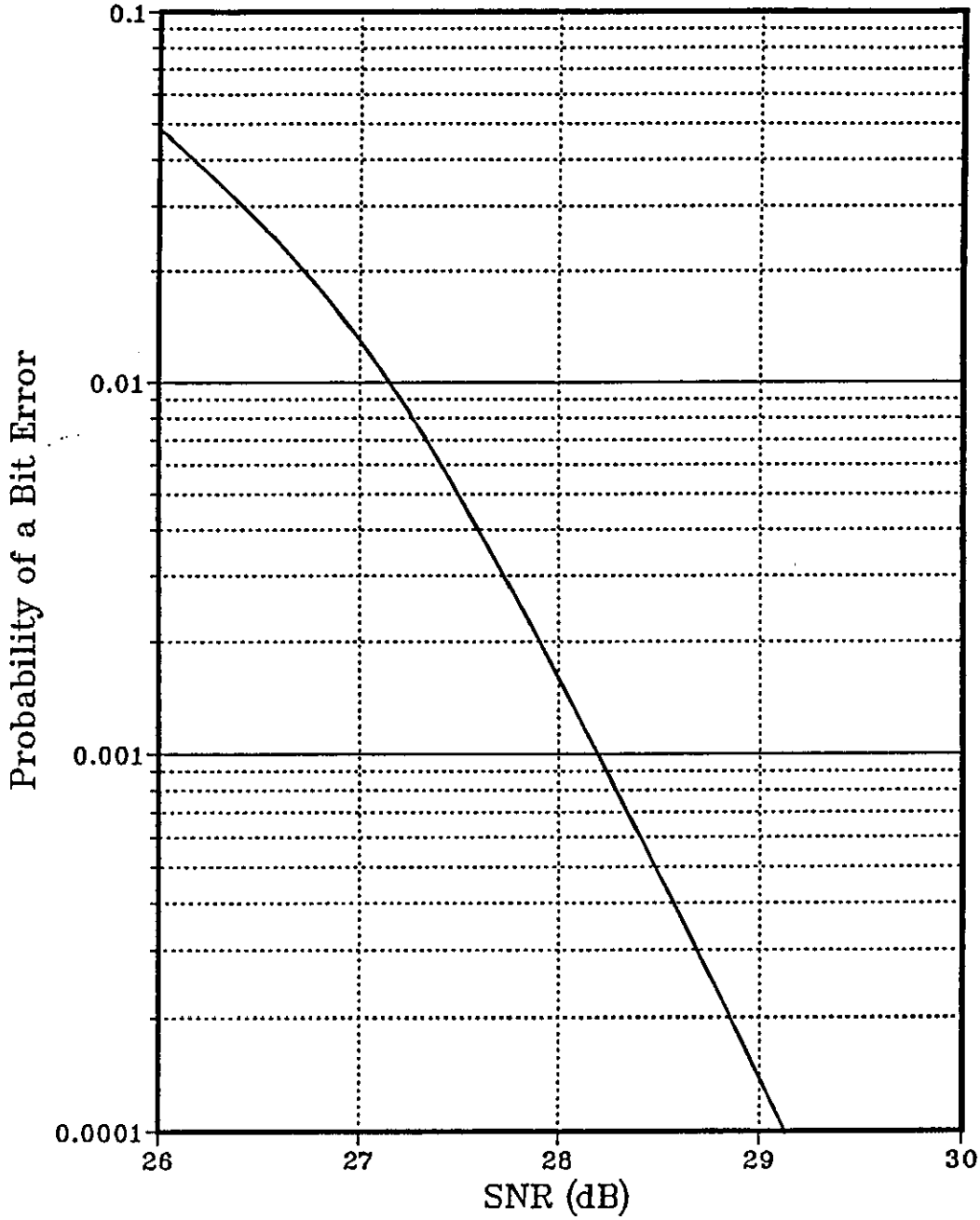


Figure 4.47 Performance of Detector A over Channel 1 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=9)

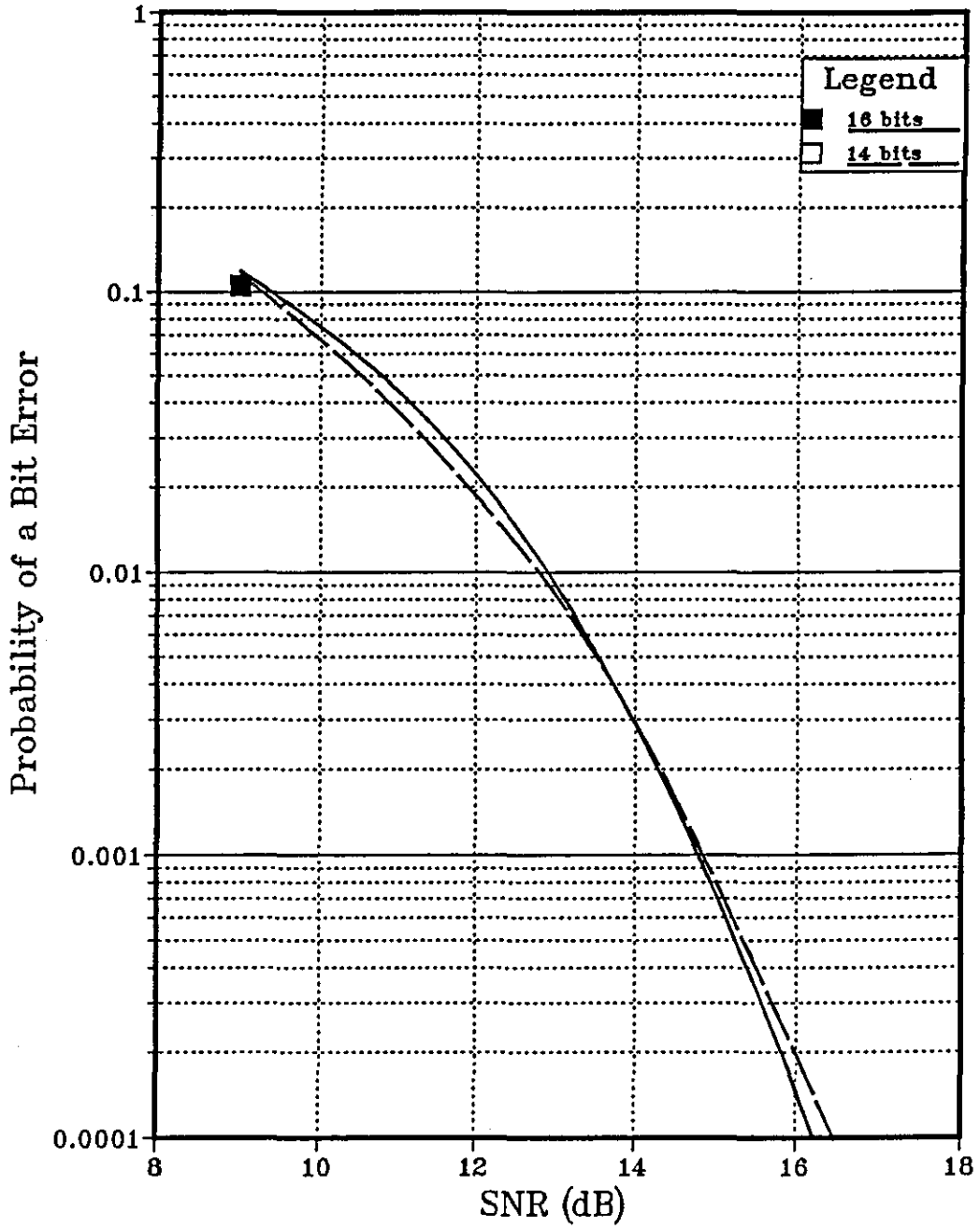


Figure 4.48 Performance of Detector A over Channel 2 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=9)

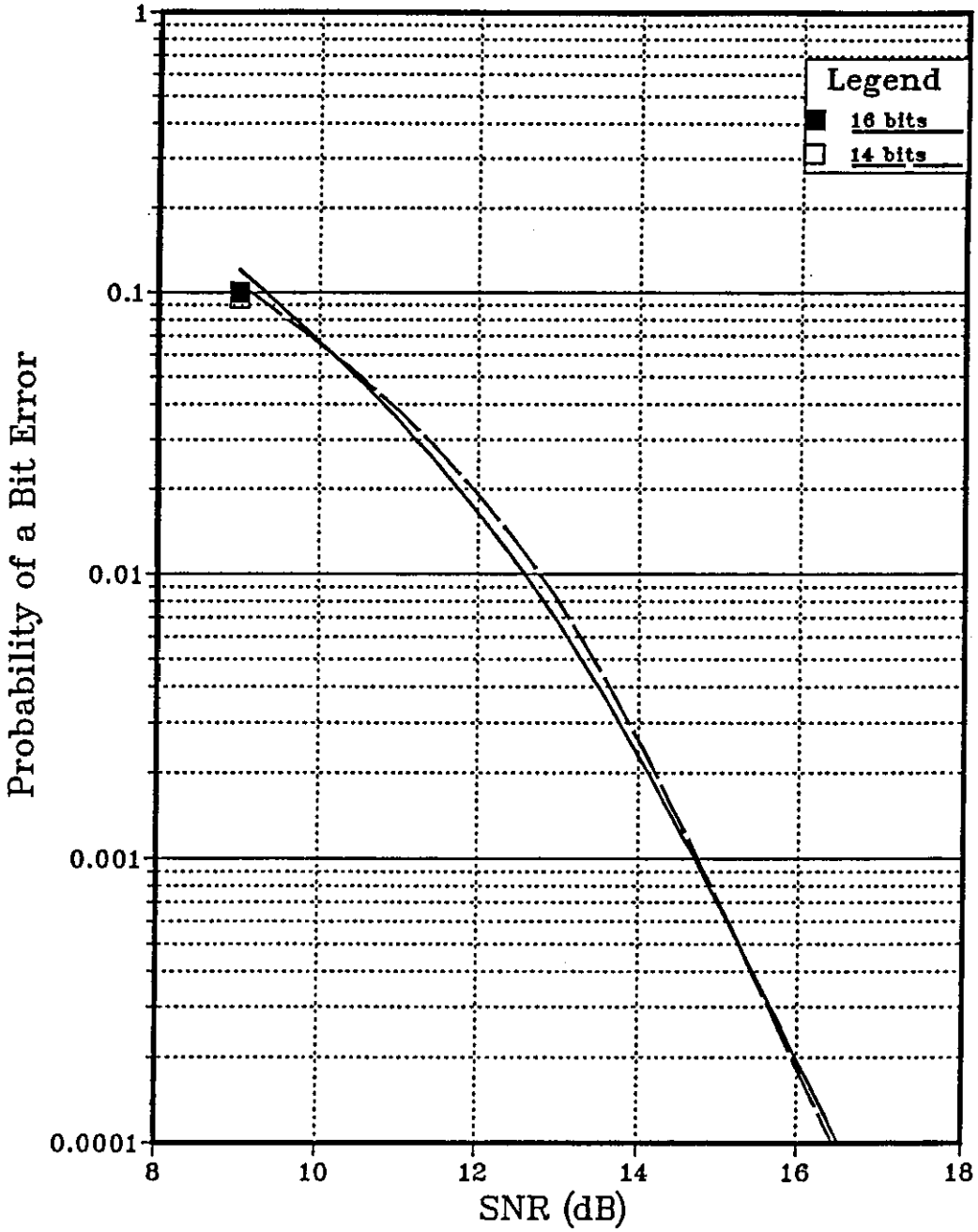


Figure 4.49 Performance of Detector A over Channel 3 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=14)

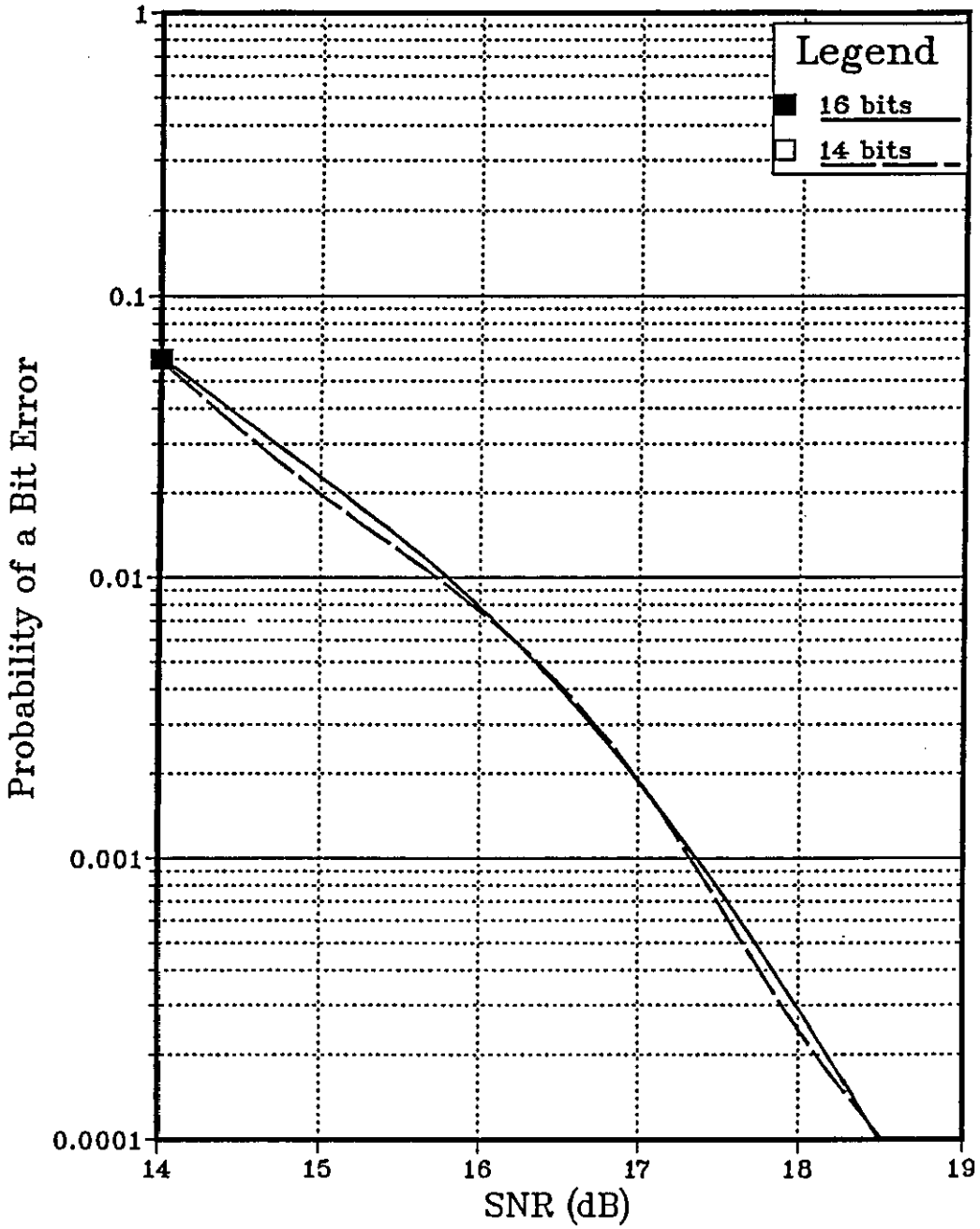


Figure 4.50 Performance of Detector A over Channel 4 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =16, g=14)

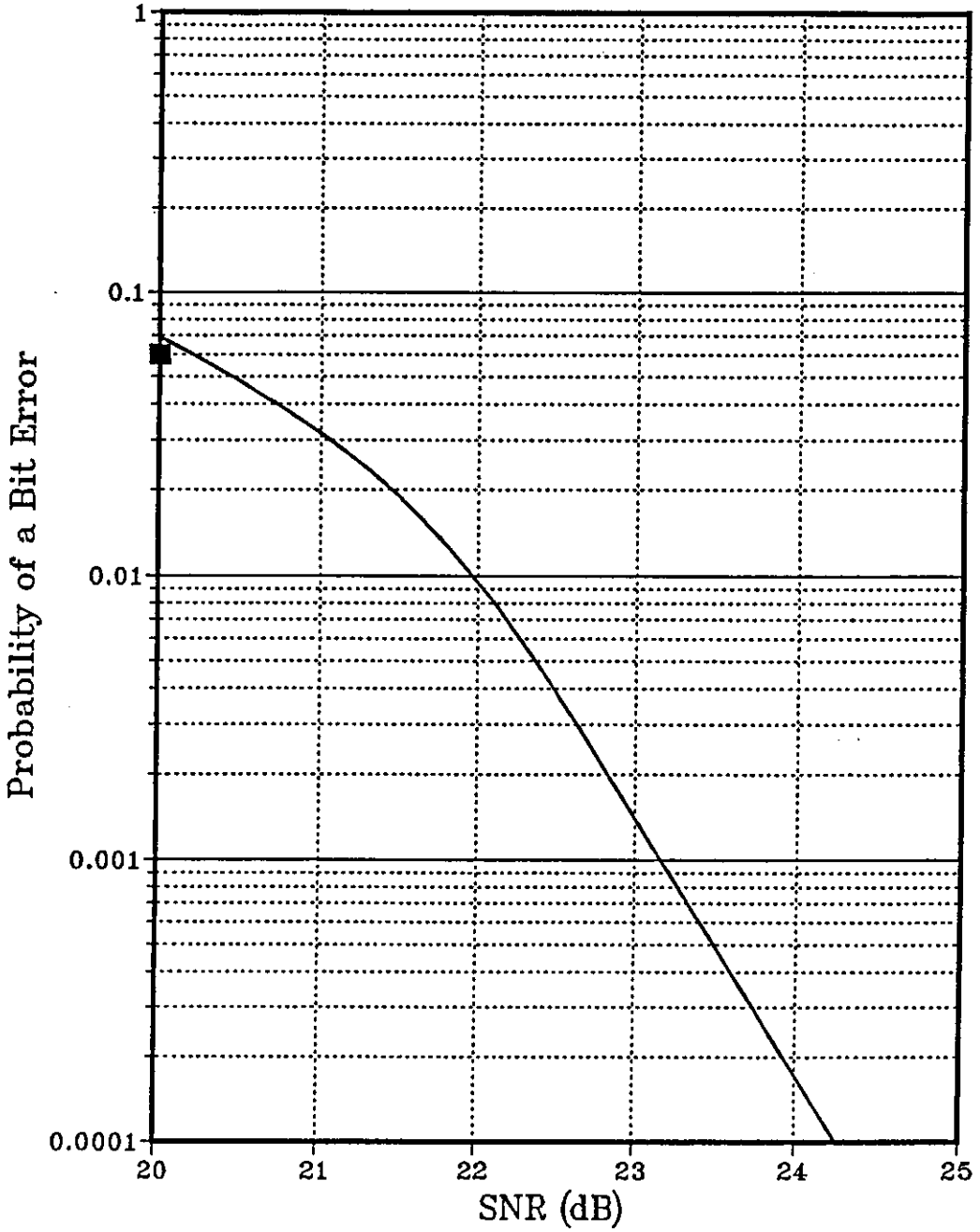


Figure 4.51 Performance of Detector B over Channel 1 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=9)

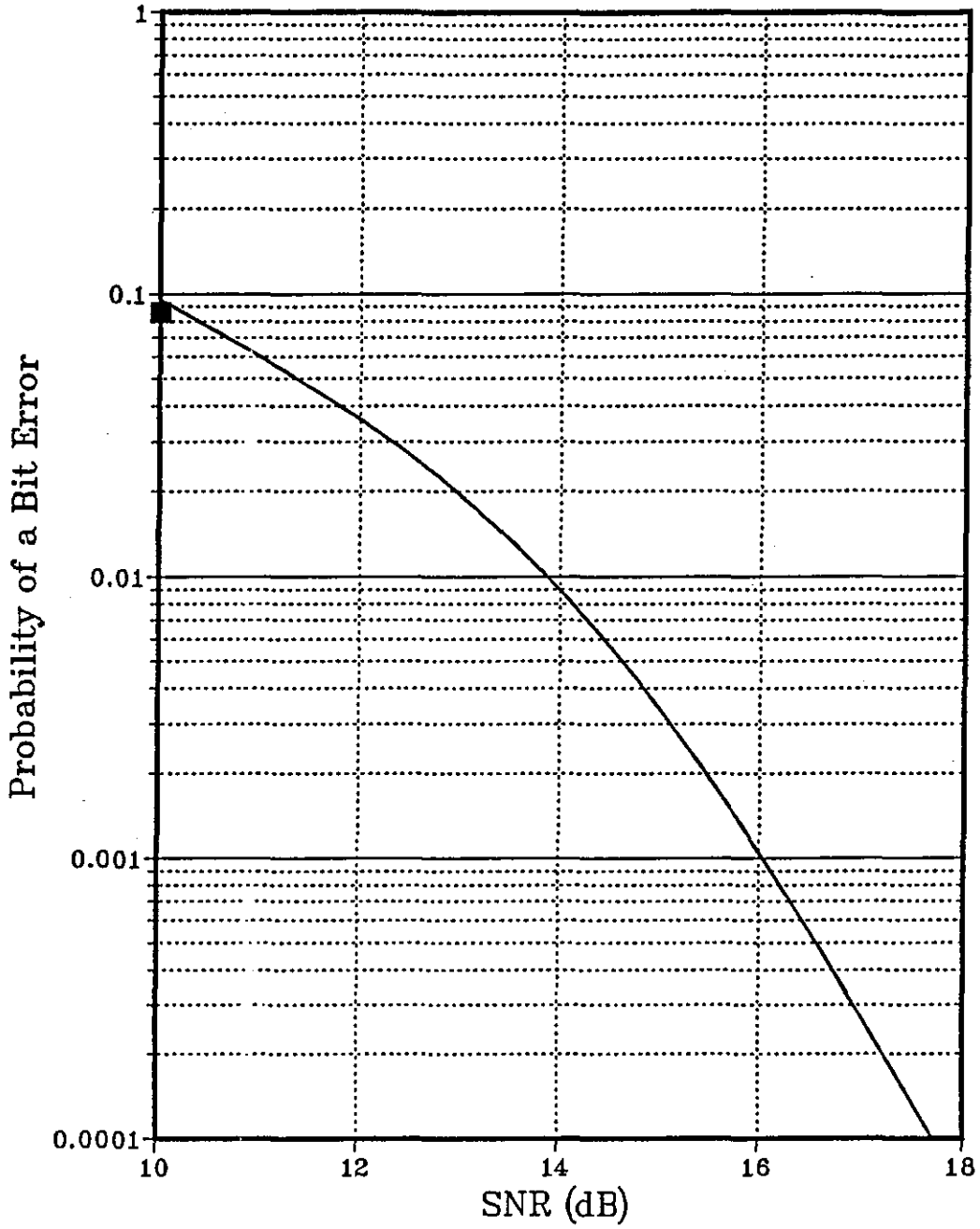


Figure 4.52 Performance of Detector B over Channel 2 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=9)

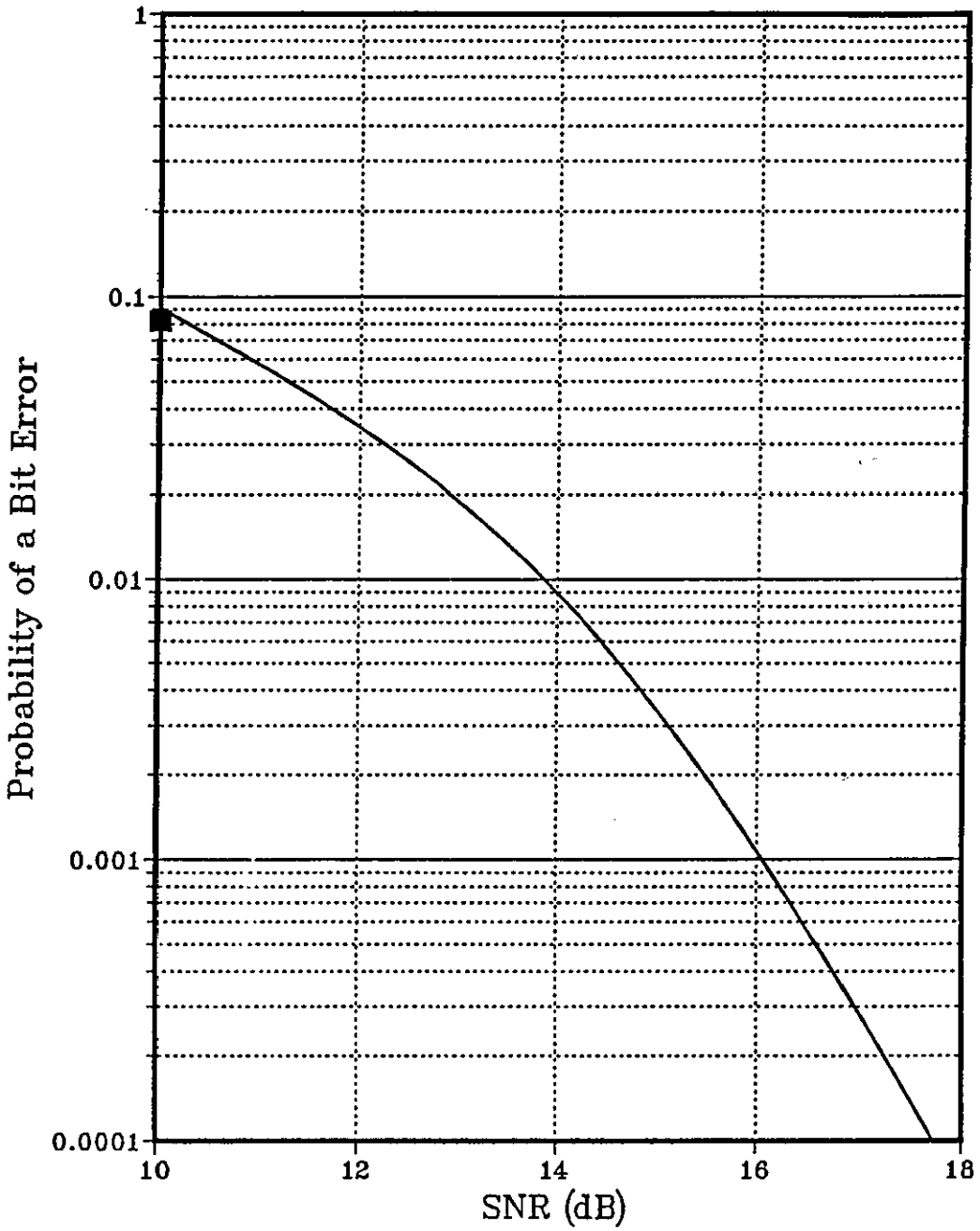


Figure 4.53 Performance of Detector B over Channel 3 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =8, g=14)

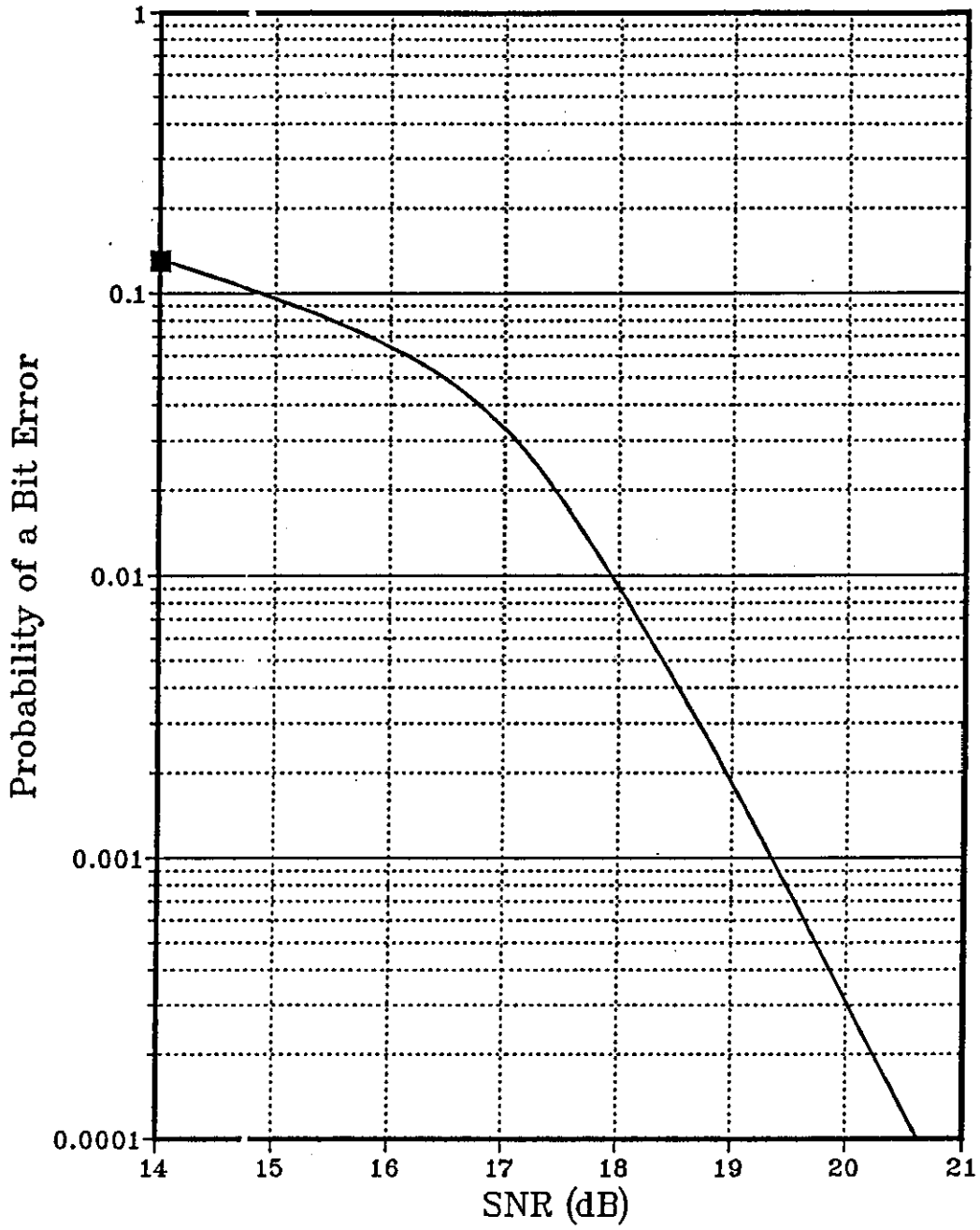


Figure 4.54 Performance of Detector B over Channel 4 at 9600 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =16, g=14)

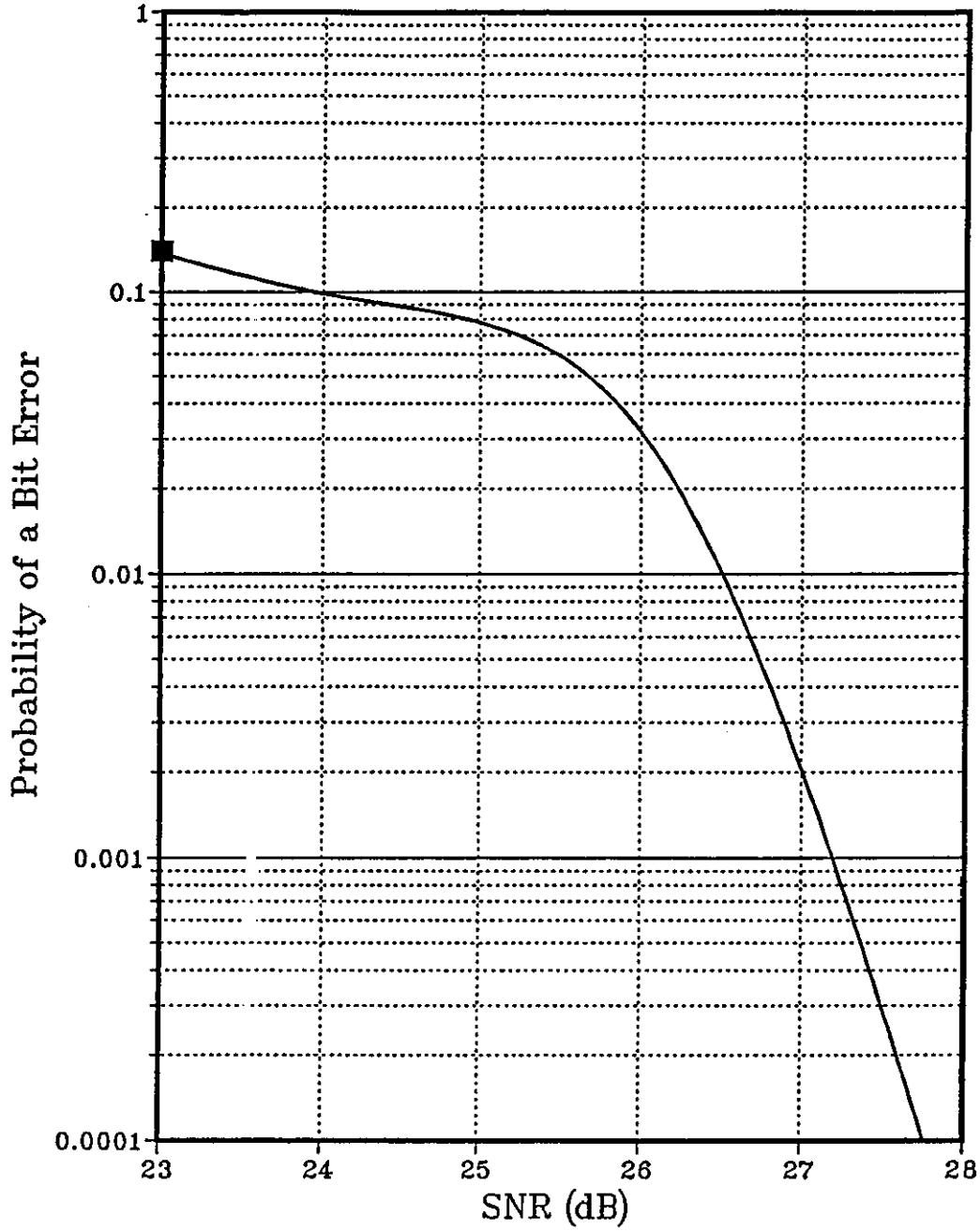


Figure 4.55 Performance of Decision Feedback Equaliser over Channel 1 at 9600 Bits/s using 16-bit Integer Arithmetic

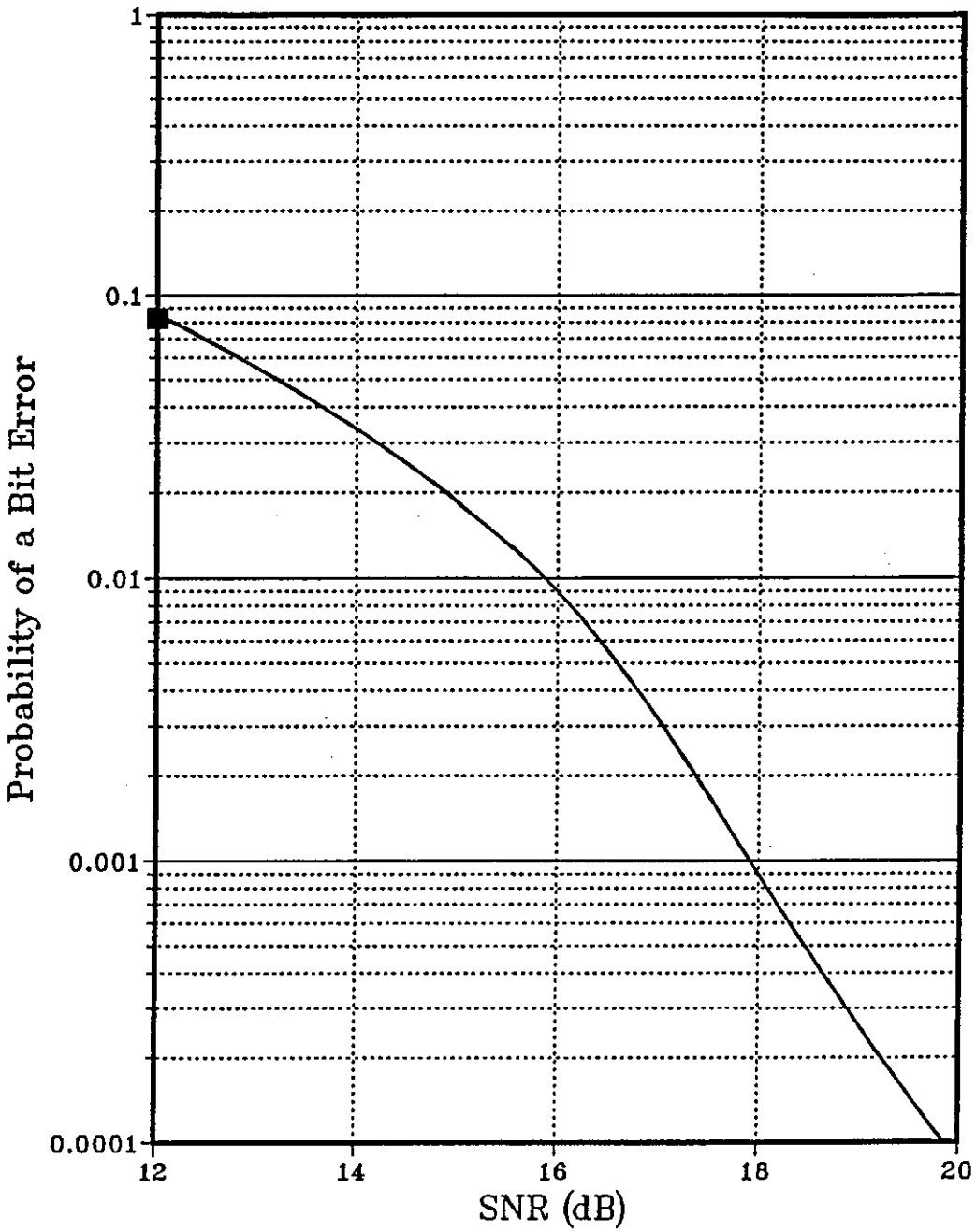


Figure 4.56 Performance of Decision Feedback Equaliser over Channel 2 at 9600 Bits/s using 16-bit Integer Arithmetic

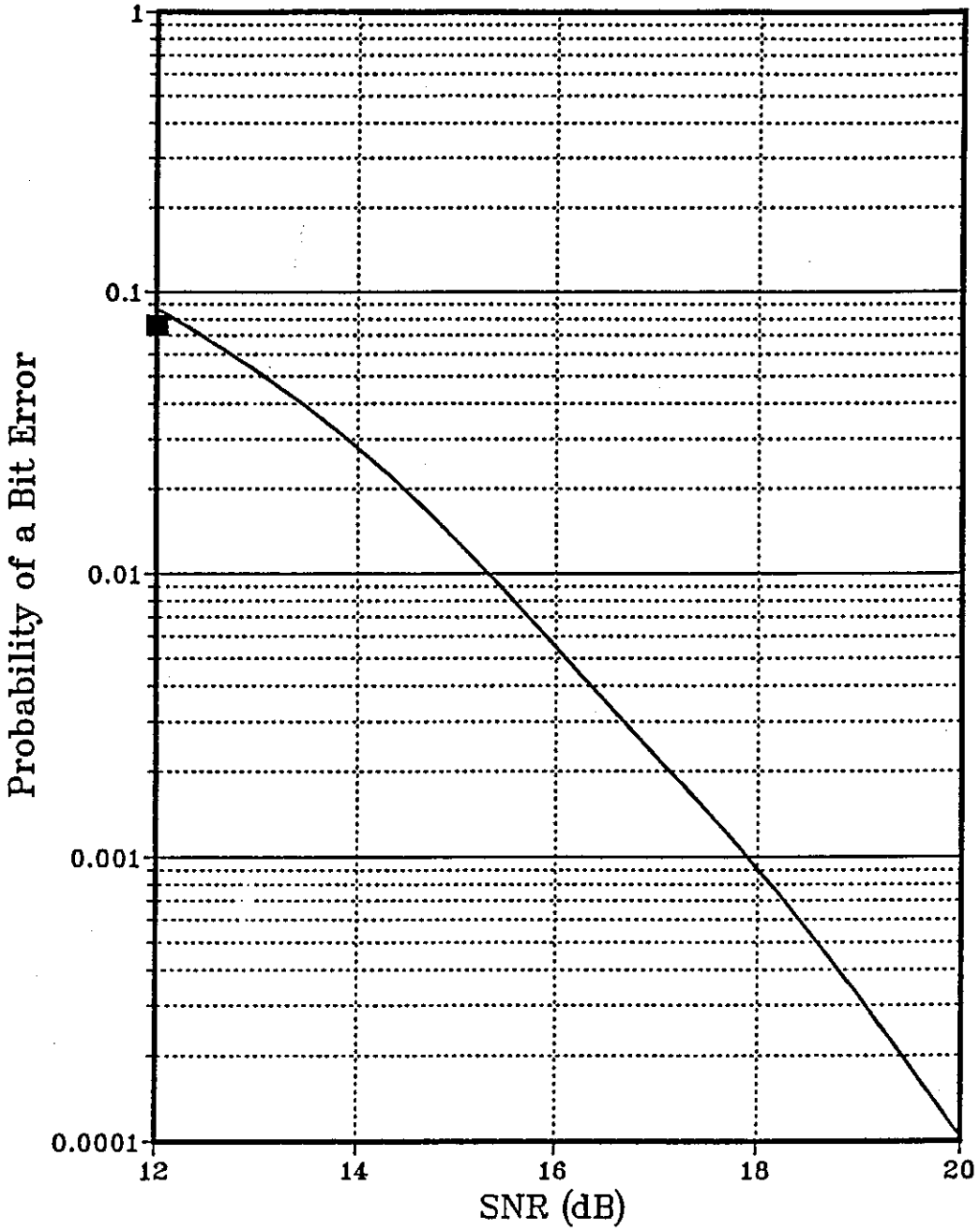


Figure 4.57 Performance of Detector A over Channel 1 at 14400 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =10)

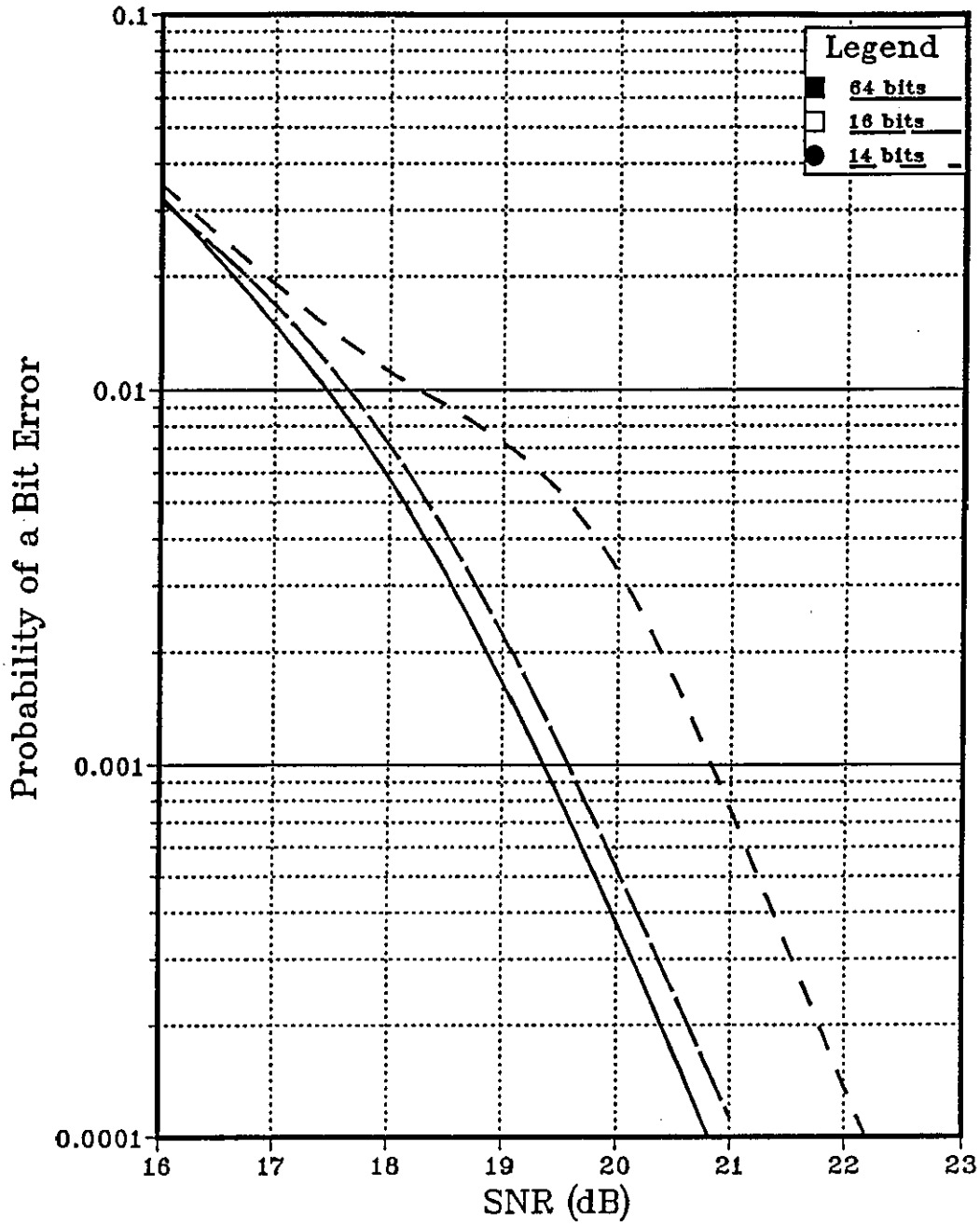


Figure 4.58 Performance of Detector A over Channel 2 at 14400 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =10)

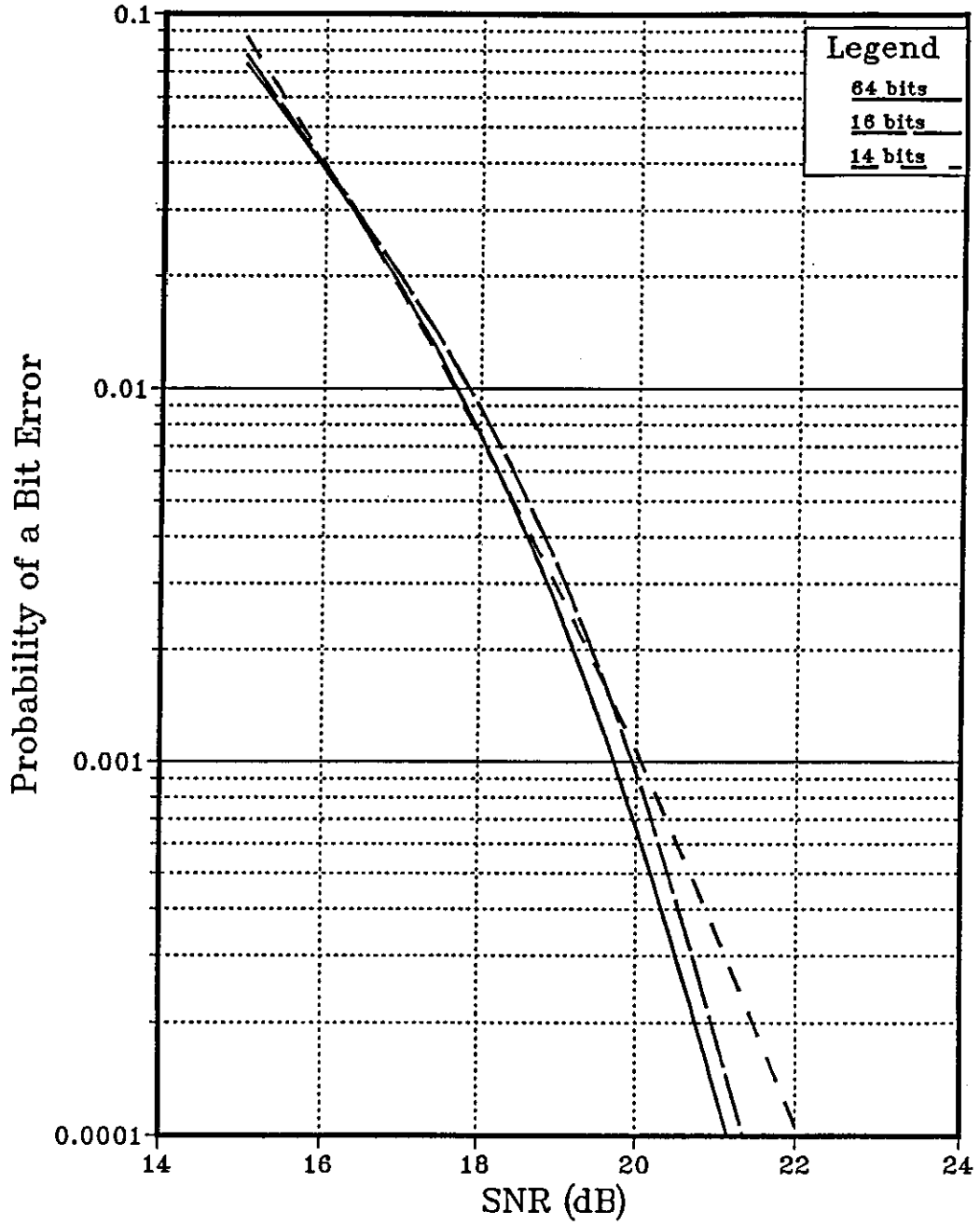


Figure 4.59 Performance of Detector A over Channel 3 at 14400 Bits/s (Number of Stored Vectors=4, Length of each Stored Vectors =10)

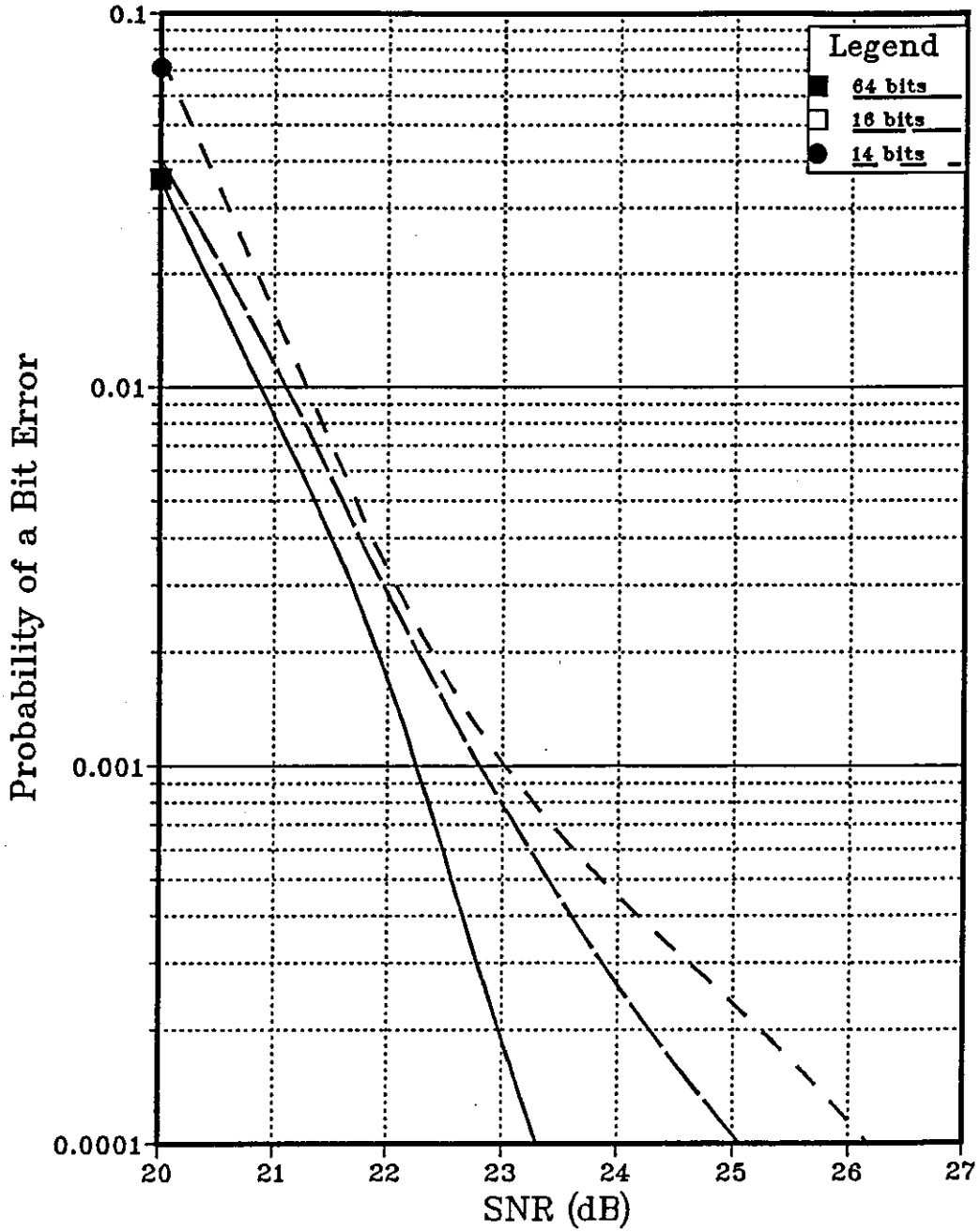


Figure 4.60 Performance of Detector A over Channel 4 at 14400 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =20)

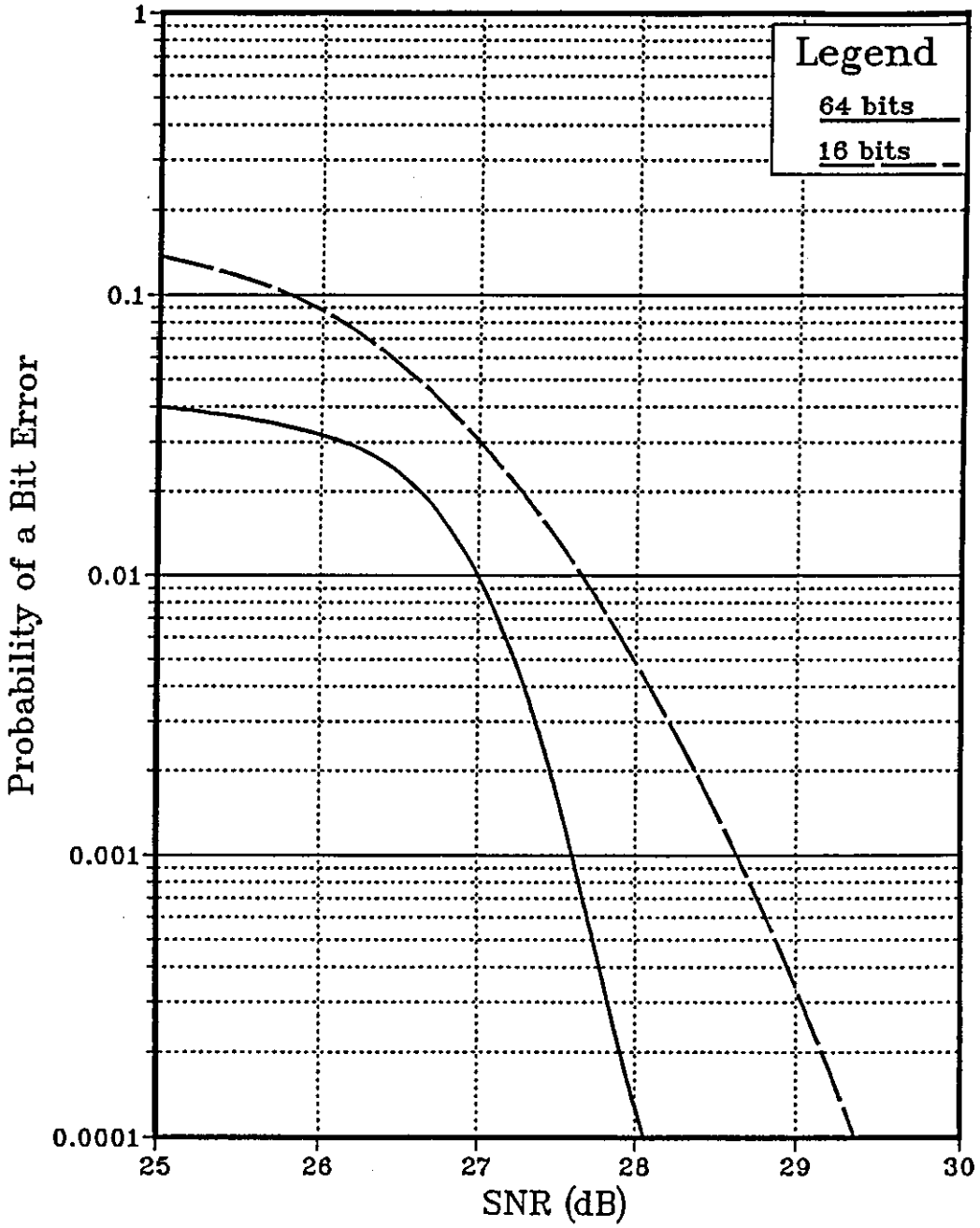


Figure 4.61 Performance of Detector B over Channel 1 at 14400 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =10)

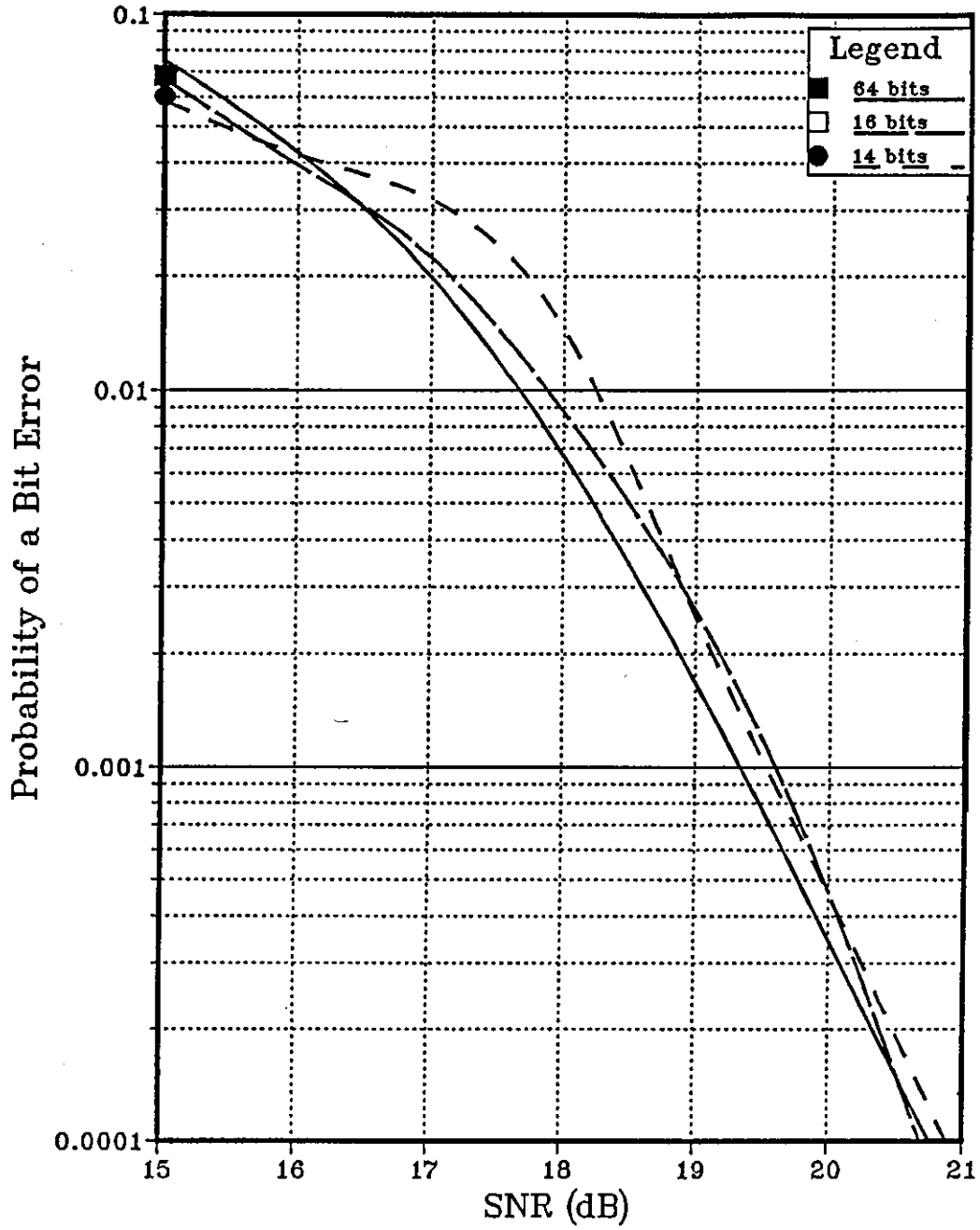


Figure 4.62 Performance of Detector B over Channel 2 at 14400 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =10)

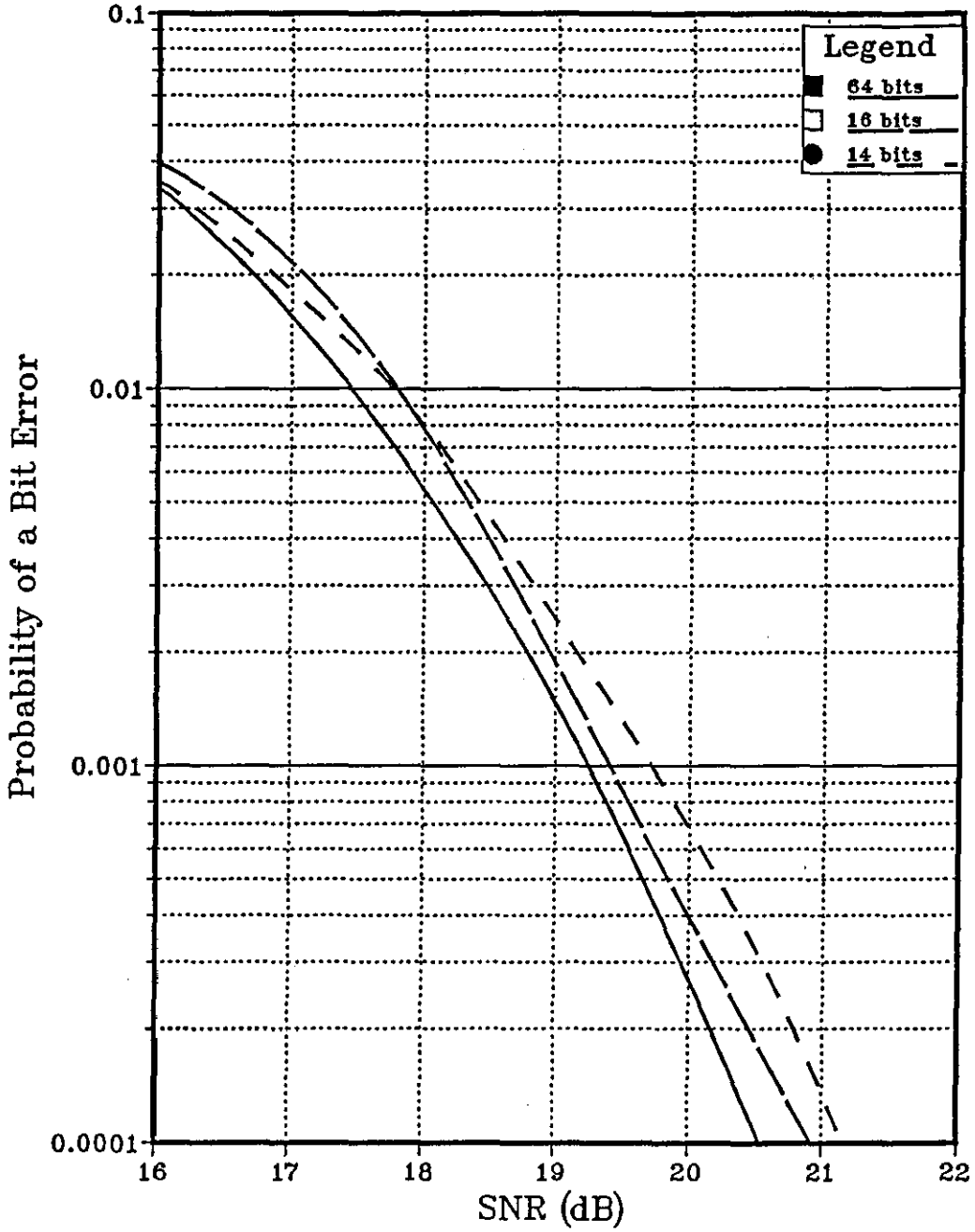


Figure 4.63 Performance of Detector B over Channel 3 at 14400 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =10)

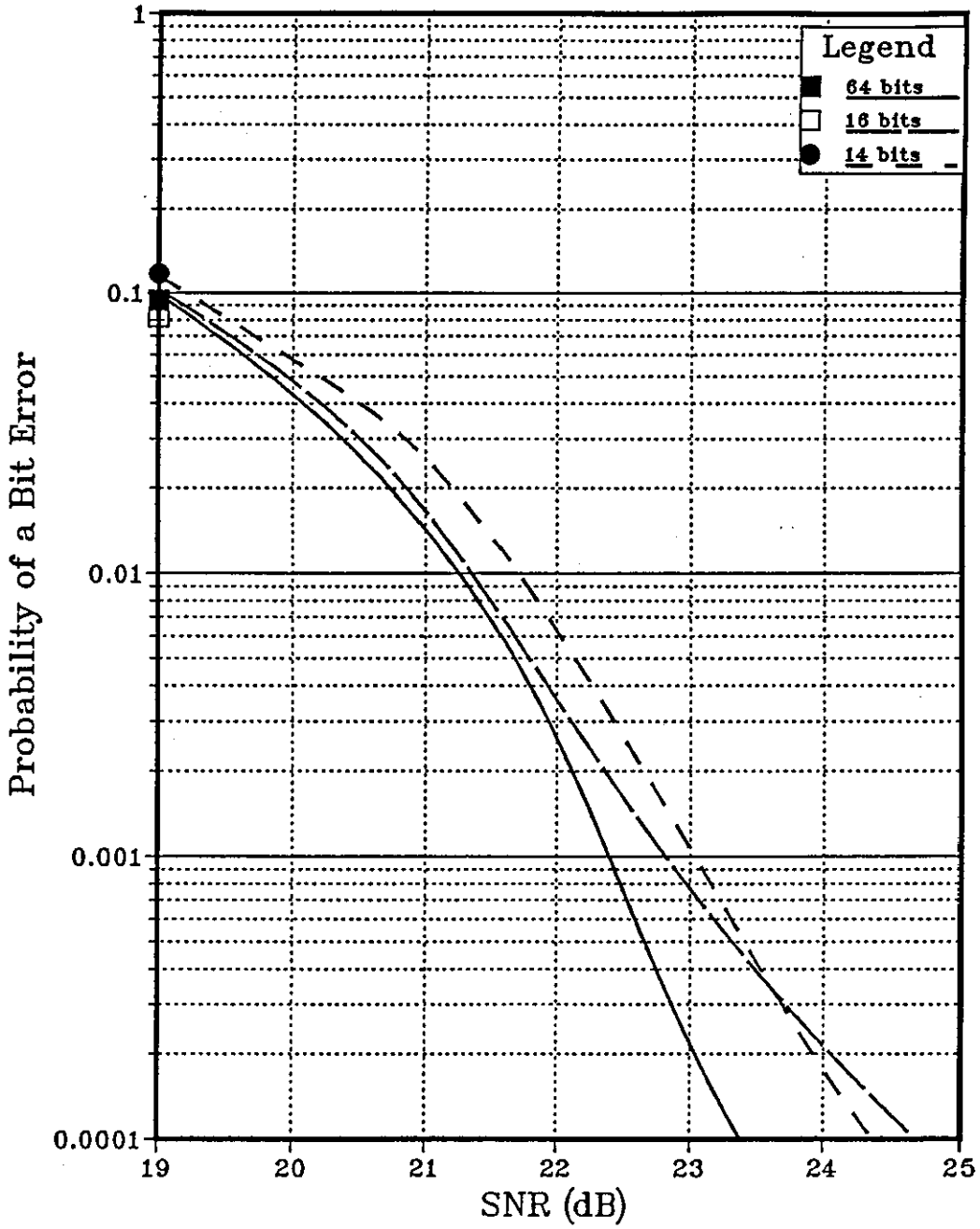


Figure 4.64 Performance of Detector B over Channel 4 at 14400 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =10)

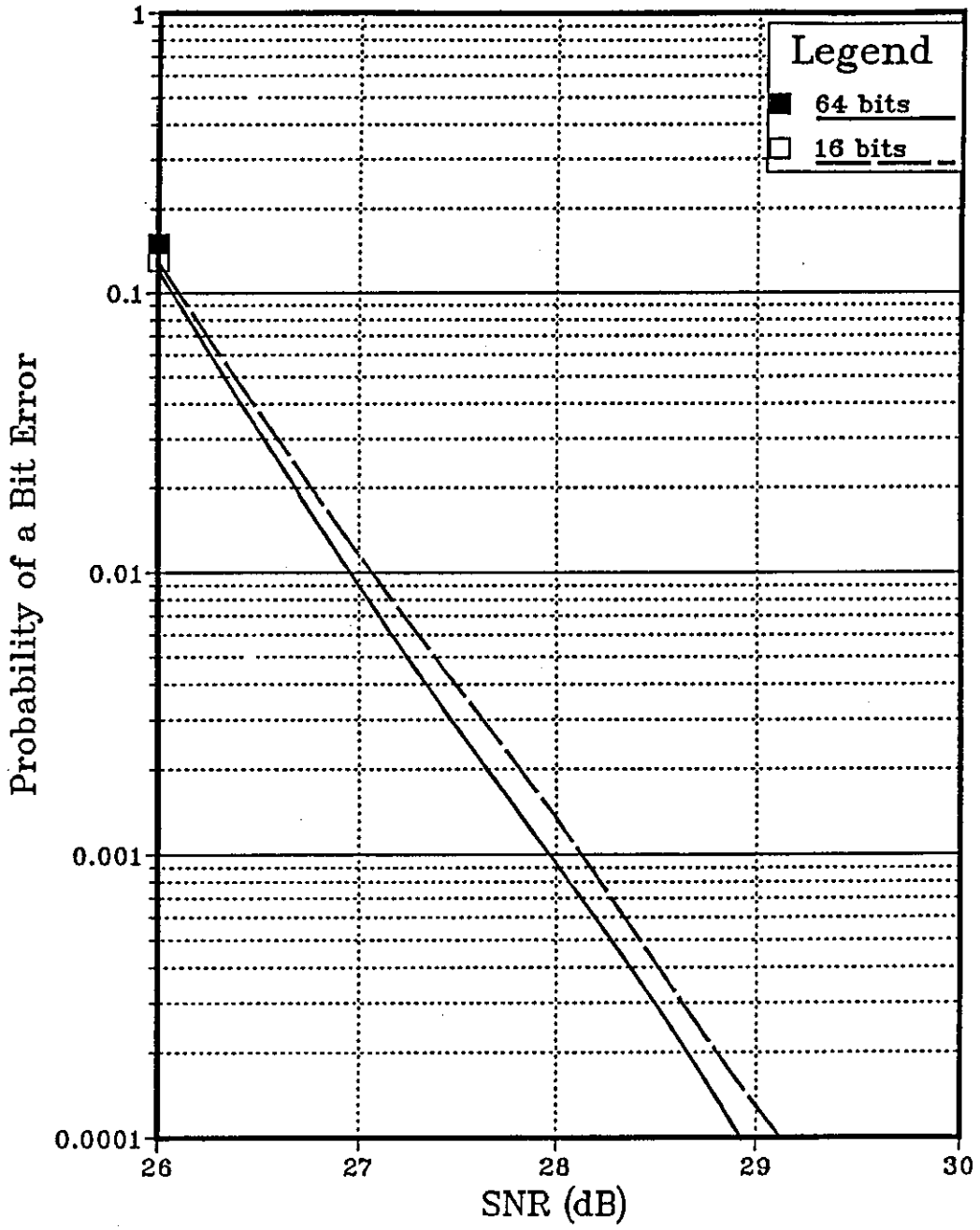


Figure 4.65 Performance of Detector B over Channel 1 with a Carrier Phase Error

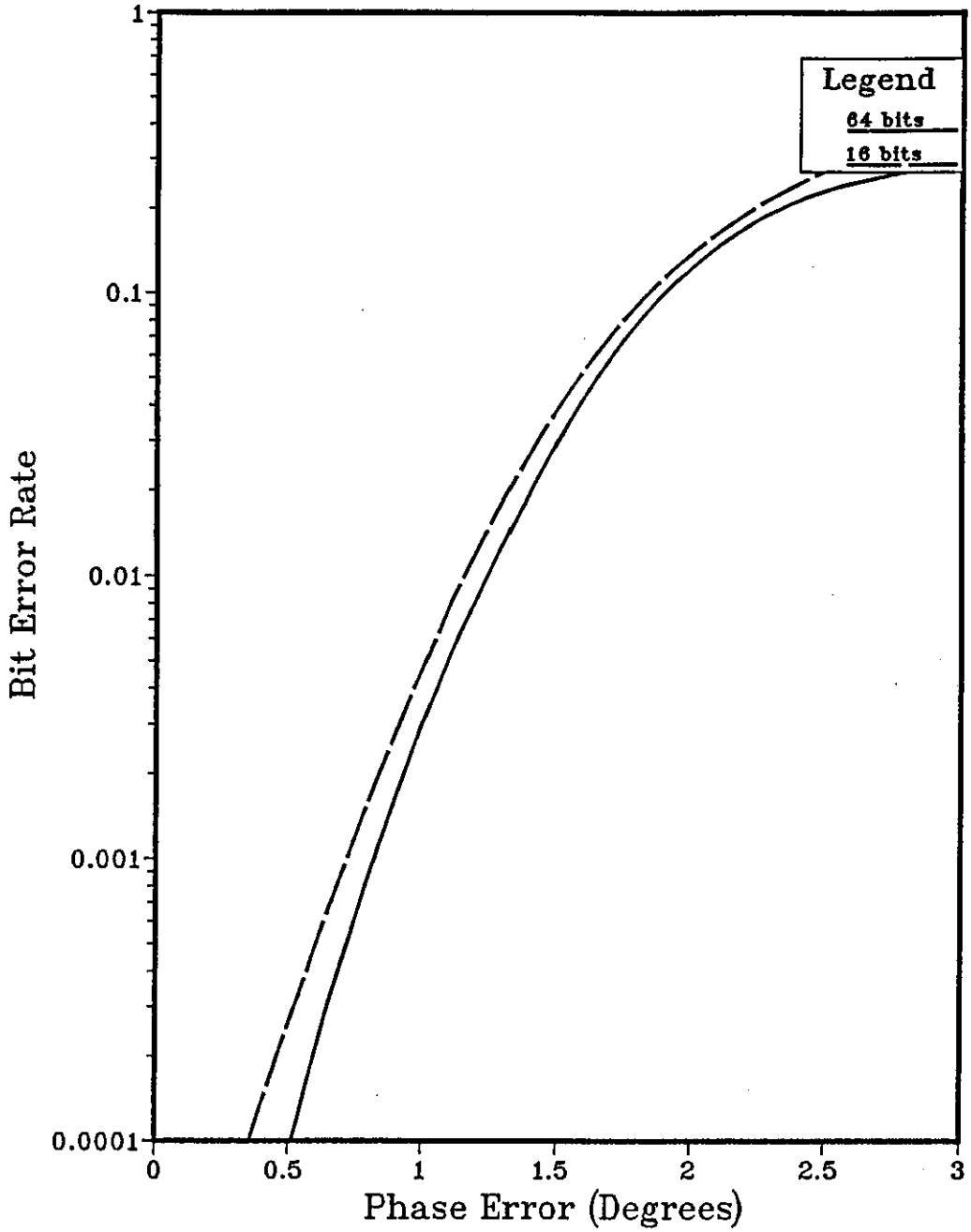


Figure 4.66 Performance of Detector B over Channel 4 with a Carrier Phase Error

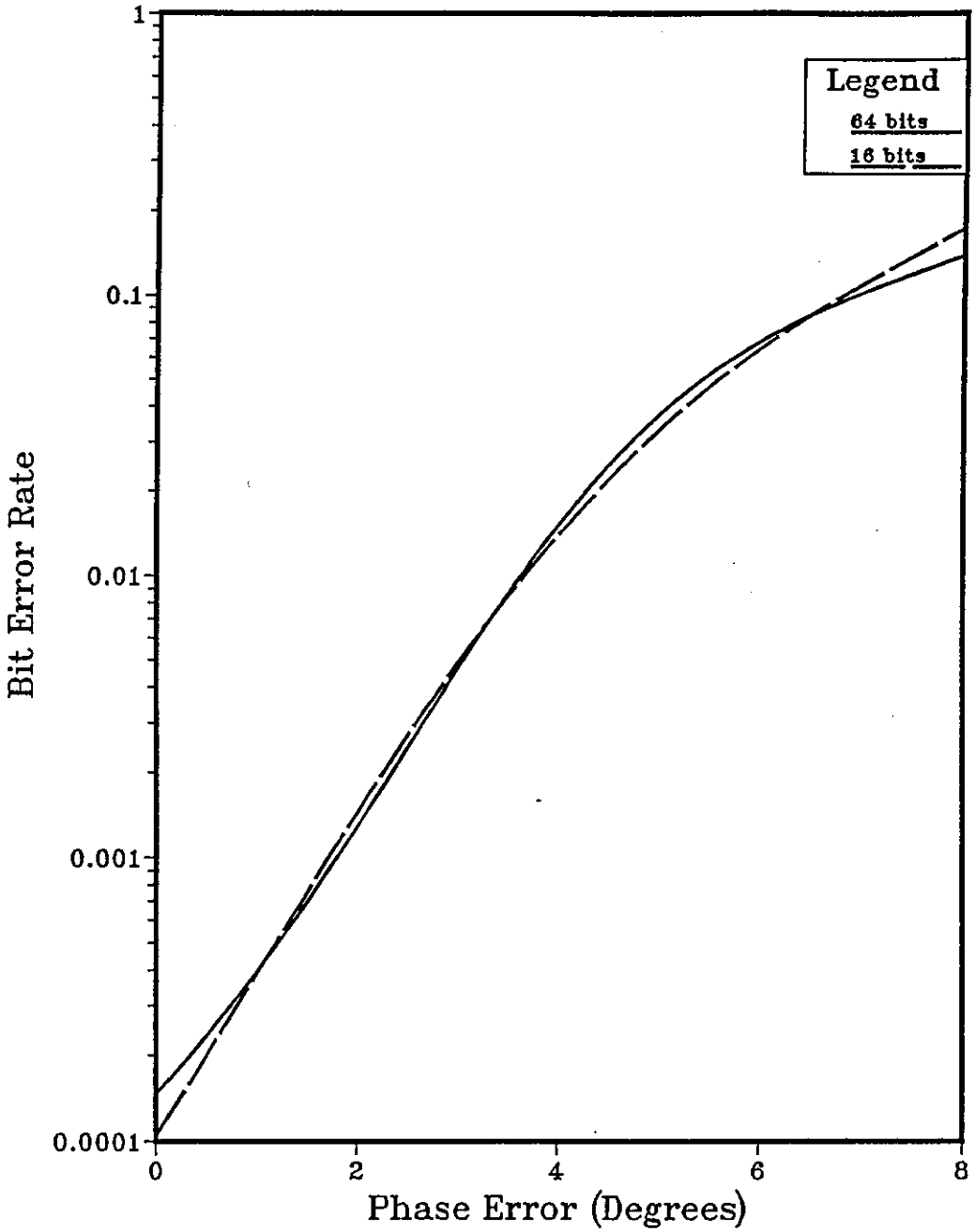


Figure 4.67 Performance of Detector B over Channel 5 at 19200 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =20)

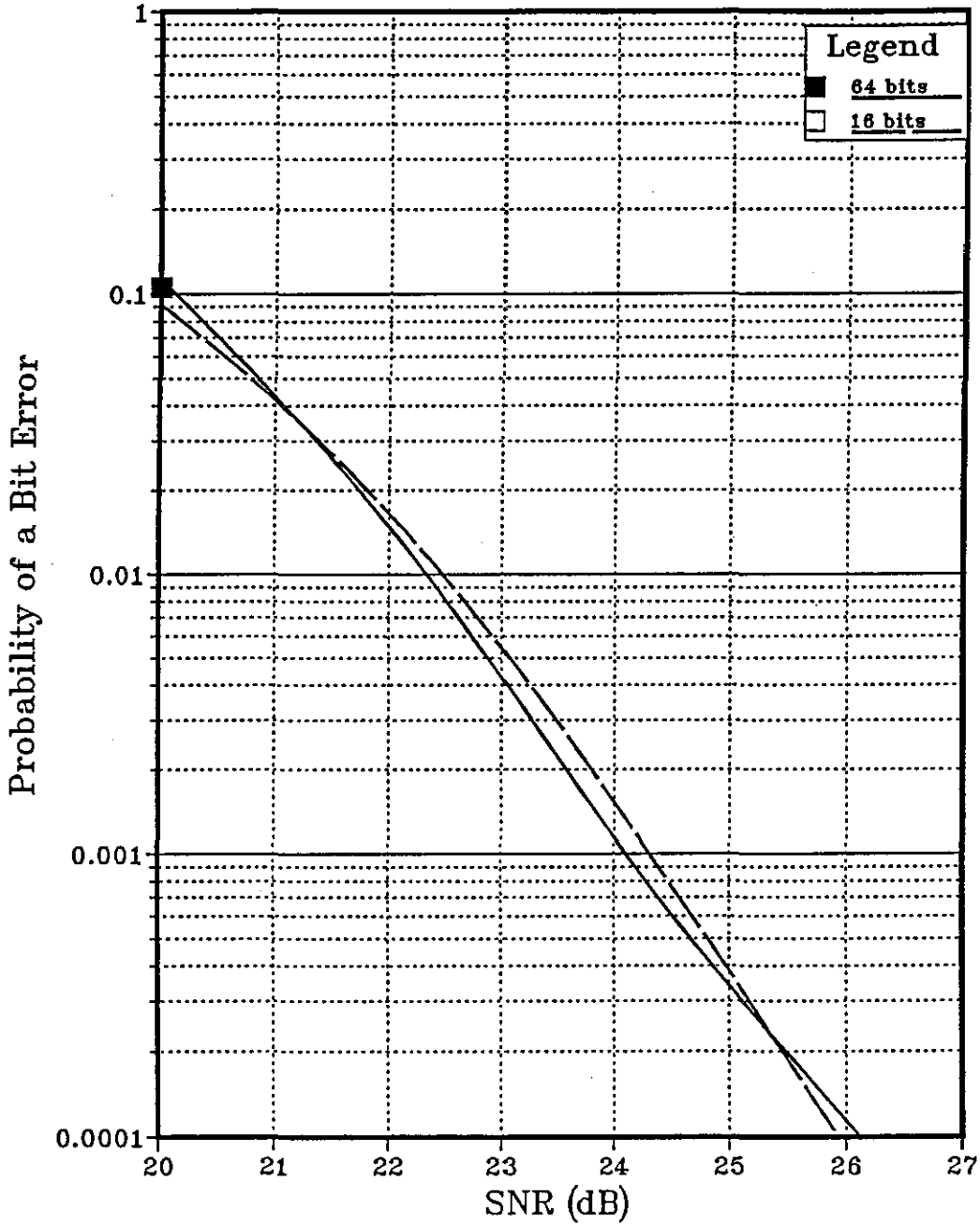
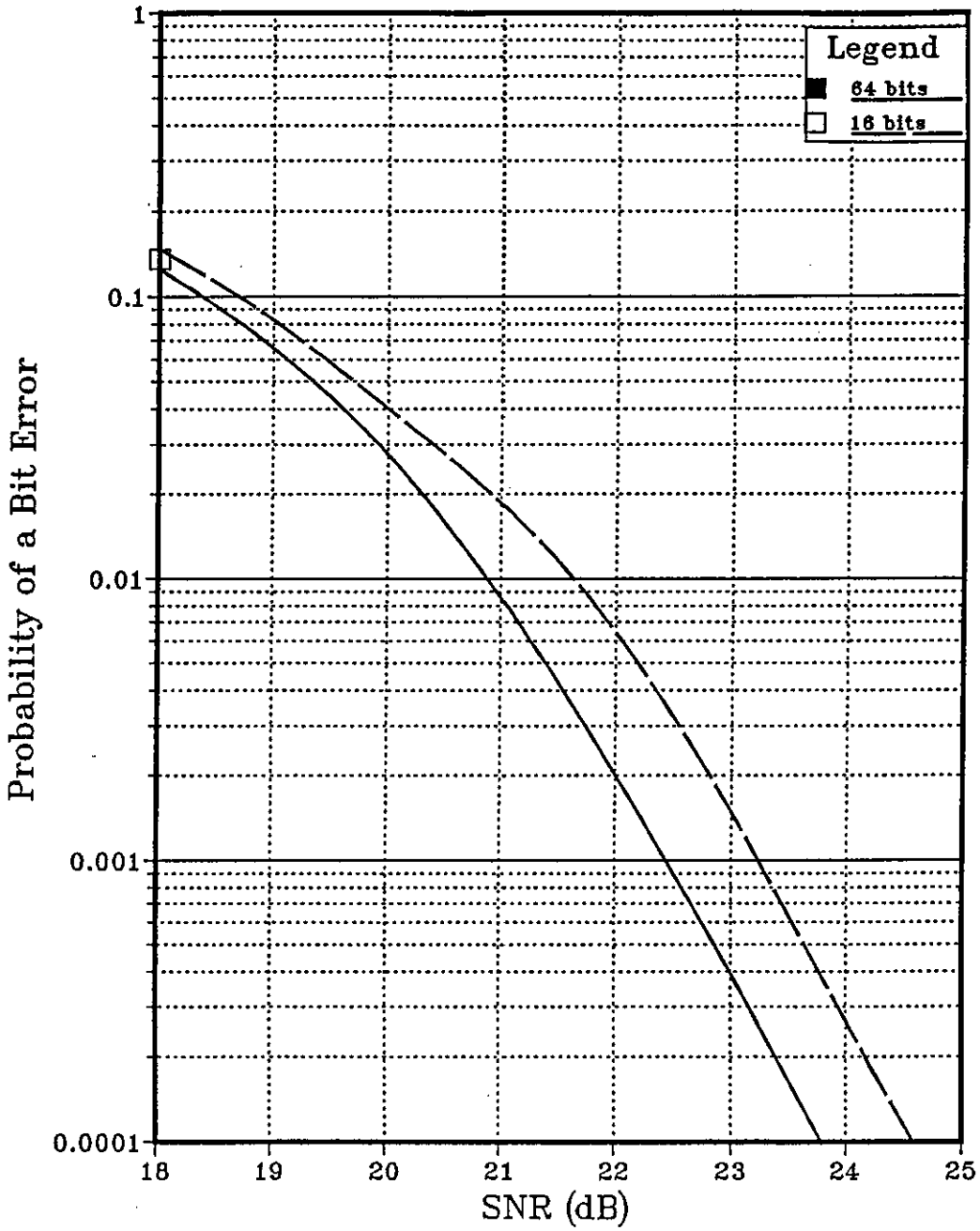


Figure 4.68 Performance of Detector B over Channel 6 at 19200 Bits/s (Number of Stored Vectors=8, Length of each Stored Vectors =20)



Chapter 5
The H.F. Channel

5.1 Introduction

The H.F. radio link is an important medium for long distance communications, particularly in millitary applications. H.F. radio waves are easily propagated over long distances due to the Earths ionospheric layers in the upper parts of its atmosphere. The ionosphere consists of free electrons and ions [58] that exist in sufficient abundance to allow H.F. radio waves that lie in the frequency range of 3 to 30 MHz to be reflected off them. Telecommunications that rely upon the ionosphere as a medium for propagation are subjected to the ionospheric changes that exist, and will therefore, be seriously affected by these changes [58].

5.2 Propagation of H.F. waves in the Ionosphere

The free electrons within the ionosphere are produced as a result of the electromagnetic radiation emitted by the sun passing through the upper regions of the Earths atmosphere. The ionosphere extends from around 50 km to over 700 km above the Earths surface, its structure varies with height and it may be divided vertically into three main regions, known as the D, E and F layers. These layers increase in altitude and electron density, the F layer having the highest altitude and electron density.

The D region extends from 50 to 90 km above the surface of the Earth. Its electron density varies from sunrise to sunset, reaching a maximum just after midday and dropping to almost zero at night. The critical frequency (the highest frequency of a vertically incident carrier that will be reflected back towards the Earths surface) of the D layer is around 100 to 700 kHz. Since H.F. radio waves have typical frequencies greater than 3 MHz, the D layer mainly acts as an attenuator and is of no use for propagation.

The E layer lies between 90 and 130 km above the Earths surface, and has a critical frequency of about 4 MHz. As with the D layer, ionization begins at sunrise, reaching a maximum at around noon and after sunset the layer gradually breaks down. Also, within the E layer are additional narrow layers, known as sporadic E layers, that may have a local electron density exceeding that of the rest of the E layer and which varies rapidly with time.

The F layer is a thick region that extends from 130 km upwards above the Earths surface. The lower part of this layer shows different variations compared with its upper parts and it may be sub-divided into two layers, named the F_1 and F_2 layers. The F_1 layer, which extends from 130 km to 220 km has similar variations to those of the E layer. At night or at certain times during the winter months the F_1 layer merges with the F_2 layer and the two together are termed simply as the F layer. The F_2 layer is the highest ionospheric layer, and extends from 225 km upwards. Its critical frequency is between 5 and 10 MHz. During the night time and sometimes during the day time, especially in the winter months, the F_1 layer merges with the F_2 layer and the critical frequency drops to 3 to 5 MHz. The F_2 layer has the highest electron density, both during the day and at night and is therefore the most important layer for H.F. radio communications.

The high electron density in the F layer makes it a good reflector of H.F. radio waves, and because of the great altitude of this layer, it can support a single hop propagation of over 4000 km as shown in figure 5.1.

When designing an H.F. radio link many factors need to be considered [58], for example, the length of the link, the optimum operating frequency and the probability of interference from another link, the projected lifetime of a link and the percentage of time a system is likely to be inoperable due to extreme ionospheric conditions.

5.3 Theory of Propagation [59]

The H.F. radio waves are returned back to the Earth from the ionosphere through a process known as refractive bending. The refractive index, ε, of the ionosphere depends upon the electron density, which varies with height. The refractive index of an ionised medium is given by

$$\varepsilon=\left(1-\frac{81N_e}{f^2}\right)^{1/2} \qquad ..5.1$$

where f is the frequency of the radio waves and N_e is the number of free electrons per cubic meter (m^3). In figure 5.2 the refractive bending process is illustrated. The ionosphere can be divided up into many very narrow strips, each with a constant refractive index. For a given angle of incidence of a radio wave meeting a reflecting layer, total internal reflection occurs when

$$\sin\theta_i=\left(1-\frac{81N_e}{f^2}\right)^{1/2} \qquad ..5.2$$

where θ_i is the critical angle of the radio wave measured from the normal. If a radio wave is incident upon the layer at an angle less than θ_i then it will pass through the layer and will be lost. For reflection to occur when the angle of incidence θ_i is zero (vertical incidence) a large electron density is required. For a given layer the highest frequency that will be reflected back for vertical incidence is given by

$$f_{cr}=9\sqrt{N_{em}} \qquad ..5.3$$

where N_{em} is the maximum number of free electrons per m^3 in the layer.

The critical frequency f_{cr} for any layer represents the highest frequency that will be reflected back from the layer. The highest frequency that will be reflected also depends upon the angle of incidence, and hence, for a given layer height, upon the distance between the transmitter and receiver. This maximum frequency is known as the maximum usable frequency (M.U.F.) and is given by

$$M.U.F.=f_{cr}\sec\theta_i \qquad ..5.4$$

Because the maximum usable frequency may vary with time of day and the season a somewhat lower frequency is used [60,61].

5.4 Types of Distortion Introduced by an H.F. Radio Link

Multipath Propagation and Time Dispersion

A radio wave may be propagated to the receiver along one or more paths, the propagation along each path taking a different period of time. This effect is known as multipath propagation [2]. Many different paths (known as modes) are possible, especially for long haul propagation, however, the number of effective modes is small. For example, the transmitted radio signal may travel from the transmitter to the receiver via two skywaves which are reflected from two different layers as shown in figure 5.3a. Or alternatively, the transmitted signal may travel from the transmitter to the receiver via both one and two hops as shown in figure 5.3b. The radio waves will arrive at the receiver at different times, thus if a single pulse is transmitted it will arrive at the receiver via several routes and will have an amplitude versus time profile as shown in figure 5.4 [61]. The time between reception of the first and last pulse in figure 5.4 is known as the time spread or time dispersion of the received signal. Measurements have shown that, usually, two to four paths are present for most of the time [62] and that the time dispersion is order of 1 ms [2,5,62,63].

Time dispersion gives rise to intersymbol interference, when the symbol period

becomes comparable with the relative multipath delay neighbouring symbols will overlap and make correct detection of the transmitted data difficult.

Frequency Dispersion

Frequency dispersion is caused by the variations in altitude of the ionospheric layers with time, which introduce a Doppler shift into the receiver signal. Figure 5.5 shows the variations in altitude for a 24 hour period [59]. It can be seen that between the hours of 5 am to 8 am and 5 pm to 7 pm that the F_2 layer is moving approximately 50 km/hour in the vertical direction. The magnitude of the Doppler shift depends upon the frequency of the carrier used in the H.F. link, for example, a 15 MHz operating frequency would give rise to a shift of around 1 Hz [2,5,63], although it may be as high as 10 Hz.

Fading

Fading is the variation with time of the received signal strength, and may be caused by several different processes. The ionosphere really acts as a large number of different reflectors at different heights and introducing different attenuations, to give a large number of reflected waves with different levels and carrier phases. Hence the signal at the receiver antenna will be the resultant of all these waves given by the phasor sum, and, if the relative heights and attenuations of these layers vary randomly with time, the resultant field strength at the receiver antenna will also vary in a random manner. The envelope of the received random signal will have a Rayleigh probabilty density function and its phase will have a uniform probability density function.

The random variations in the altitudes of the ionospheric layers cause slight changes in the frequency of the received waves to occur due to the Doppler effect. These changes cause the bandwidth of the signal to be very slightly increased, typically around 0.1 Hz under mild conditions and 0.5 Hz for more severe links.

Absorption fading is caused by the variation in the absorption characteristics of the ionosphere with time. The attenuation characteristic of the D layer slowly changes with

the time of day, reaching maximum absorption at sunrise and sunset. The depth of fading can be as high as 10 dB below the mean signal level.

Skip fading is caused by changes in the M.U.F. If the operating frequency is kept fixed then at one instant it may be very much below the M.U.F., and at another instant it may be above, causing complete signal loss for a short period of time.

5.5 Statistics of the received signal [5]

Consider a transmitted signal, which may be represented in general as

$$s(t)=\mathrm{Re}\left[u(t)e^{j2\pi f_c t}\right] \qquad ..5.5$$

where f_c is the carrier frequency and u(t) is the low-pas modulating signal, given by

$$u(t)=a(t)e^{j\theta} \qquad ..5.6$$

where a(t) is the amplitude (envelope) of s(t), and θ is the phase of s(t). Assuming that there are multipath propagations, and that associated with each path is a propagation delay and an attenuation factor then the received bandpass signal, r(t), will be

$$x(t)=\sum_n \alpha_n(t)s\left(t-\tau_n(t)\right) \qquad ..5.7$$

where $a_n(t)$ is the amplitude of the signal received via the n^{th} path at time t and $\tau_n(t)$ is the propagation delay for the n^{th} path. Substituting 5.5 into 5.7 gives

$$x(t)=\mathrm{Re}\left[\left\{\sum_n \alpha_n(t)u\left(t-\tau_n(t)\right)e^{-j2\pi f_c\tau_n(t)}\right\}e^{j2\pi f_c t}\right] \qquad ..5.8$$

It can is apparent from equation 5.8 that the equivalent low-pass received signal is

$$r(t)=\sum_n \alpha_n(t)e^{-j2\pi f_c \tau_n(t)}u\left(t-\tau_n(t)\right) \qquad ..5.9$$

Since r(t) is the response of an equivalent low-pass channel to the equivalent low-pass signal u(t), it follows that the equivalent low-pass channel is described by the time variant impulse response

$$h(t,\tau)=\sum_n \alpha_n(t)e^{-j2\pi f_c \tau_n(t)}\delta\left(t-\tau_n(t)\right) \qquad ..5.10$$

If the received signal is considered to be a continuum of multipath components, the summation in equation 5.7 may be expressed in the integral form

$$x(t)=\int_{-\infty}^{\infty}\alpha(\tau,t)s(t-\tau)d\tau \qquad ..5.11$$

where $a(\tau,t)$ denotes the attenuation of the signal components at delay t and at time instant t. Substituting 5.5 into 5.11 yields

$$x(t)=Re\left[\left\{\int_{-\infty}^{\infty}\alpha(\tau,t)e^{-j2\pi f_c\tau}u(t-\tau)\,d\tau\right\}e^{j2\pi f_c t}\right] \qquad 5.12$$

Since the integral in 5.12 represents the convolution of u(t) with the equivalent low-pass time varient impulse response $h(\tau,t)$ it follows that

$$h(\tau,t)=\alpha(\tau,t)e^{-j2\pi f_c\tau} \qquad ..5.12$$

Consider now, the case when an unmodulated carrier is transmitted, at a frequency f_c. Then u(t)=1 for all t and, hence the received signal for the case of many discrete paths given by equation 5.9, reduces to

$$r(t)=\sum_n \alpha_n(t)e^{-j2\pi f_c \tau_n(t)}$$

$$=\sum_n \alpha_n(t)e^{-j\theta_n(t)} \qquad ..5.13$$

where $\theta_n(t)=2\pi f_c t_n(t)$. The signal consistes of a number of time varient phasors having amplitudes $a_n(t)$ and phases $\theta_n(t)$. It is only when there is a large change in the structure of the reflecting medium before $a_n(t)$ changes sufficiently for a significant variation in the received signal level. On the other hand, $\theta_n(t)$ will change by 2π radians whenever $t_n(t)$ changes by $1/f_c$. But $1/f_c$ is a very small quantity, hence θ_n can change by 2π radians with relatively small motions of the reflecting medium. The random variations of the ionospheric reflecting layers causes $\theta_n(t)$ and hence $\tau_n(t)$ to vary in a random manner. This implies that the received signal, r(t), in equation 5.13 can be modelled as a random process. When there are a large number of paths, i.e. for a large value of n, then according to the central limit theorem, the real and imaginary parts of r(t) in equation 5.13 can be considered as two independant Gaussian random processes. The two quadrature components of the resultant signal will both have a zero mean and equal variances. The envelope of r(t), that is |r(t)| will have a Rayleigh distribution and the phase of r(t) will be uniformly distributed between 0 and 2π radians.

A single non-fading component may also be present in r(t) due to ground wave propagation, r(t) will then have a Nakagami-Rice or Ricean distribution. The majority of ionospheric media exhibit Rayleigh fading, and thus based on the present knowledge of ionospheric conditions, it appears that the Raleigh fading model best describes most H.F. channels.

The envelope of the received signal at any time, can be taken to be Rayleigh distributed. It is a continuous random variable, derived from the two indendent Gaussian random variables X and Y, and its probability density function is given by

$$p(r)=\begin{cases}\frac{r}{\sigma^2}\exp\left(\frac{-r^2}{2\sigma^2}\right) & 0\le r\le\infty \\ 0 & r<0\end{cases} \qquad ..5.14$$

where σ^2 is the varience of X and Y, and the mean value of X and Y are zero. Also

$$\sigma_X^2=\sigma_Y^2=\sigma^2 \qquad ..5.15$$

The envelope, $R=\sqrt{(X^2+Y^2)}$, has a Rayleigh distribution and has a probability distribution given by equation 5.14.

Since R can't be non-negative then its mean value must be non-zero, even though X and Y have zero means. Figure 5.6 shows the plot of the Rayleigh probability density function of R as a function of r. The curve reaches a maximum value of $1/\sigma\sqrt{e}$ at $r=\sigma$. The cumulative density function of the Rayleigh distribution, is give by

$$f(r)=\int_{-\infty}^{r}\frac{u}{\sigma^2}\exp\left(\frac{-u^2}{2\sigma^2}\right)du$$

$$=1-\exp\left(\frac{-r^2}{2\sigma^2}\right) \qquad \text{for } r\ge 0 \qquad ..5.16$$

The mean value of R is given by

$$\bar{r}=\int_0^\infty rf(r)dr=\sqrt{\frac{\pi}{2}}\sigma \qquad ..5.17$$

and the second moment of R (the mean square value) is given by

$$E[R^2]=E[X^2+Y^2]$$

$$=E[X^2]+E[Y^2] \qquad ..5.18$$

where $E[X^2]$ and $E[Y^2]$ are given by

$$E[X^2]=E[Y^2]=\sigma^2 \qquad ..5.19$$

Substituting 5.19 into 5.18 gives the mean square value of R to be

$$\bar{r}^2=2\sigma^2 \qquad ..5.20$$

The variance of R will be

$$\sigma_r^2=\bar{r}^2-\bar{r}^2 \qquad ..5.21$$

$$=\left(2-\frac{\pi}{2}\right)\sigma^2$$

The median value of the Rayleigh distribution curve occurs at $r=r_m$ at the point where the cumulative density function, f(r), is equal to 0.5. Therefore, from equation 5.16

$$r_m=\sigma\sqrt{2\log_e 2} \qquad ..5.22$$

5.6 Simulation of the H.F. channel

The performance of two or more transmission systems for use with H.F. channels can be compared by constructing the equipment and then simultaneously testing the systems over an actual H.F. channel. Testing must be simultaneous because the channels characteristics randomly vary with time and hence the tests can't be repeated under exactly the same conditions at other times. This method is therefore not a very satisfactory arrangement due to the cost and time involved in constructing the equipment and repeatedly testing it.

An alternative to the above method is to use a channel simulator, which, when correctly set up, has the advantages of accuracy, repeatability and the possibility of generating a wide range of channel conditions in a controlled manner. It is possible to simulate the H.F. channel both in hardware and in software. The software simulator is the easiest to implement and gives a greater control over the simulated conditions, it has therefore been used throughout this work.

The software simulator makes use of a tapped delay line to model the Rayleigh fading H.F. channel; this model is shown in figure 5.7 [64-66]. It is assumed that a complex valued baseband input signal is used. The taped delay line is used to give signals arriving at the receiver with different propagation delays. Each of the delayed signals is multiplied by a suitable tap gain function, $Q_h(t)$, which introduces Rayleigh fading to the delayed signal. The resulting delayed Rayleigh fading signals are added together with an additional noise interference term, $u_N(t)$, to give the received fading signal. Usually, $u_N(t)$, is Gaussian noise noise, although other types of noise are present, such as man made noise, thermal noise and atmospheric noise. Now, to the i^{th} output of the tapped delay line Rayleigh fading must be introduced by modulating the output with the signal $Q_i(t)$. Figure 5.8 shows this process in more detail. The signals, $q_1(t)$ and $q_2(t)$, are two narrow bandwidth statistically independent Gaussian random processes with zero mean and the same variance, and they both have Gaussian shaped power spectra with the same r.m.s. frequency (f_{rms}) as shown in figure 5.9 [11]. The frequency spread, f_{sp}, introduced into an unmodulated carrier by $q_1(t)$ and $q_2(t)$ is defined as the width of the power spectrum and is given by

$$f_{sp}=2f_{rms} \qquad 5.23$$

The fading rate, f_r, of the received signal, which is defined as the average number of times the envelope of the received signal crosses its median value, is given by

$$f_r=1.475f_{rms} \qquad 5.24$$

Control of the fading rate is achieved by changing the bandwidth of the power spectrum of the Gaussian variables $q_1(t)$ and $q_2(t)$. From equation 5.23 and 5.24, f_{sp} is related to f_e by

$$f_{sp}=1.356f_e \qquad 5.25$$

which means that a 1 Hz frequency spread is equivalent to 44 fades per minute.

The random process $q_1(t)$ (or $q_2(t)$) is generated by filtering a zero mean white noise signal $n_1(t)$ as shown in figure 5.10. The power spectrum of $q_1(t)$ has a Gaussian shape, hence the filter should have a Gaussian shaped frequency response and a Gaussian shaped impulse response $f_1(t)$. The theoretical power spectrum of $q_1(t)$ is

$$|Q_1(f)|=|Q_2(f)|=\exp\left(\frac{-f^2}{2f_{rms}^2}\right) \qquad ..5.26$$

and its frequency response given by

$$\Gamma(f)=\exp\left(\frac{-f^2}{4f_{rms}^2}\right) \qquad ..5.27$$

A fifth order Bessel filter is used in the channel simulator to provide the necessary shaping to give the random process $q_1(t)$. When only one propagation path is present the depth of fade may cause a total loss of the received signal. With two propagation paths present the probability of such a deep fade occuring is less. In a typical H.F. radio link there are 2 or 3 skywaves present [58,61,62]. In the computer simulations a two sky wave H.F. channel is used and the model of this is shown in figure 5.11. It requires four random processes $q_i(t)$, i=1,2,3,4. Each random process is generated exactly as described above, the variances of all four are equal to 0.25. This value for the variances ensures that the total variance of the two skywave channel is unity. Each of the $q_i(t)$ is generated from an independent Gaussian source, thus ensuring that all four random processes $q_1(t)$ to $q_4(t)$ are uncorrelated. The Bessel filter may be implemented digitally, appendix A gives the design details of the filter, such that the $q_i(t)$ have a frequency spread of 0.5 Hz or 1 Hz. As the fading signals $q_i(t)$ have a Gaussian spectra, they contain all frequencies, but for practical purposes, a fading signal of (say) 1 Hz frequency spread could be represented adequately at a sampling rate of 10 samples/second without any significant aliasing occurring. For reasons connected with the processing of the transmitted signal, the actual sampling rate used is 4800 samples/second. Unfortunately at this rate the filter poles are very close the the unit circle in the complex z-plane, and to obtain the desired filter characteristics these poles must be specified to a very high degree of accuracy, otherwise instability of the filter can occur [67]. This problem can be overcome by sampling $q_i(t)$ at a much lower rate and then using interpolation to give the required sampling rate of 4800 samples/second. A sampling rate of 50 Hz was chosen so that it satisfies Nyquists sampling theorem for the $q_i(t)$, but, is not so low as to introduce inaccuracies in the fading samples obtained using interpolation to increase the sampling rate. Linear interpolation has been used, this has been shown to be adequate [68].

5.7 Testing the H.F. Channel Simulator

Figure 5.12 shows an example of the probability density of one of the $q_i(t)$, which should ideally cut the y-axis at 0.798 corresponding to a variance of 0.25. In figure 5.13 a plot of |Y| versus time for a two skywave link, that introduces a 0.5 Hz frequency spread and a 0.5 ms delay, is shown.

Figure 5.1 A Single Hop Propagation

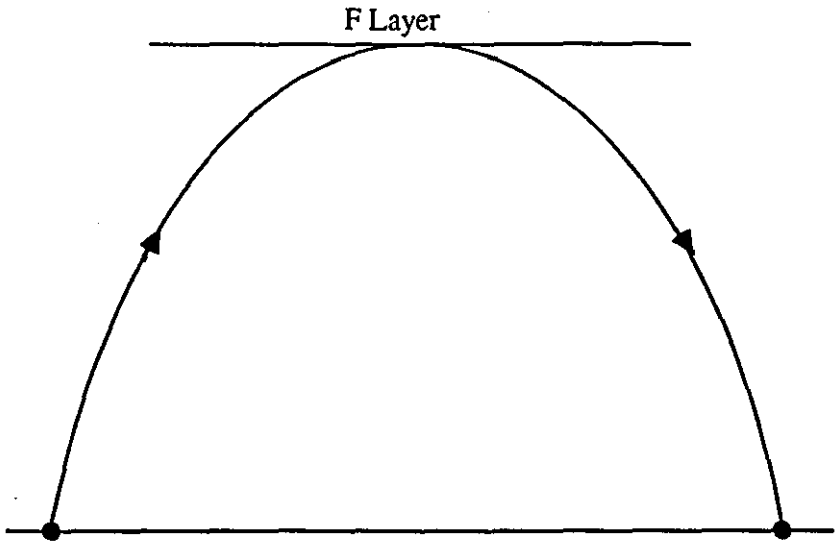


Figure 5.2 Example of Refractive Bending

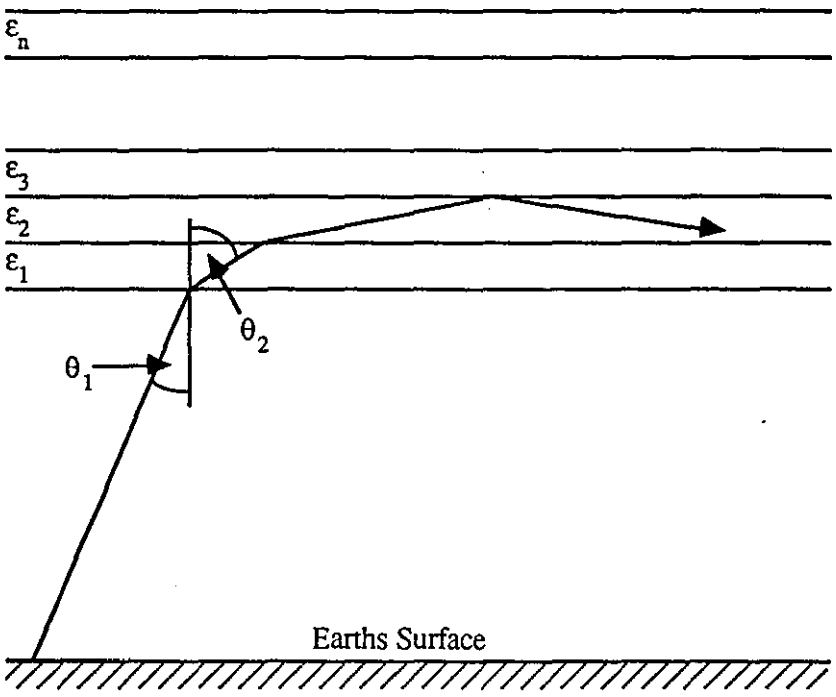


Figure 5.3a Reflection from different layers

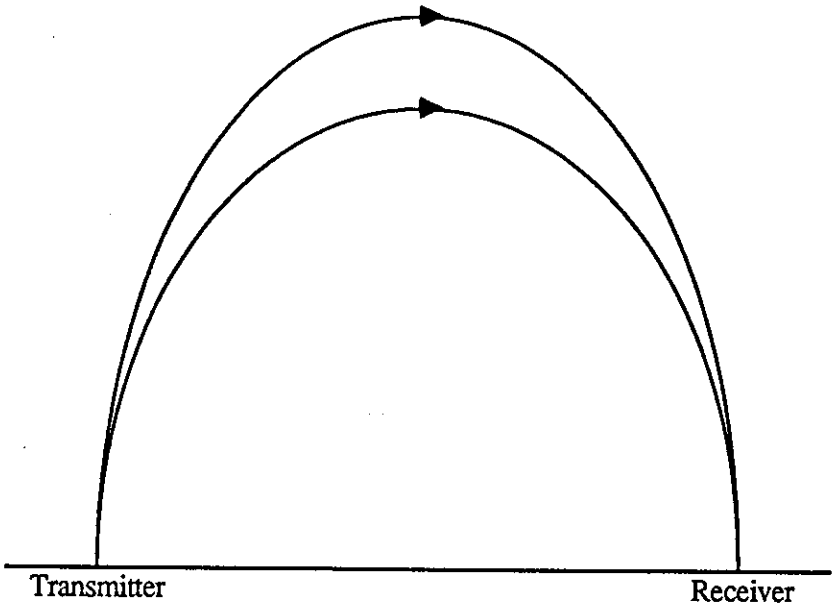


Figure 5.3b Different number of hops

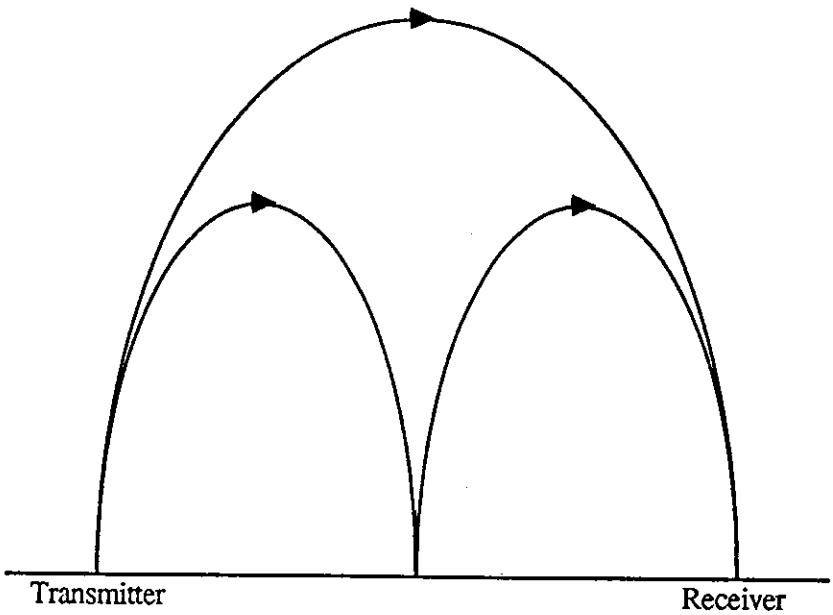


Figure 5.4 Typical Response of a Multipath Channel

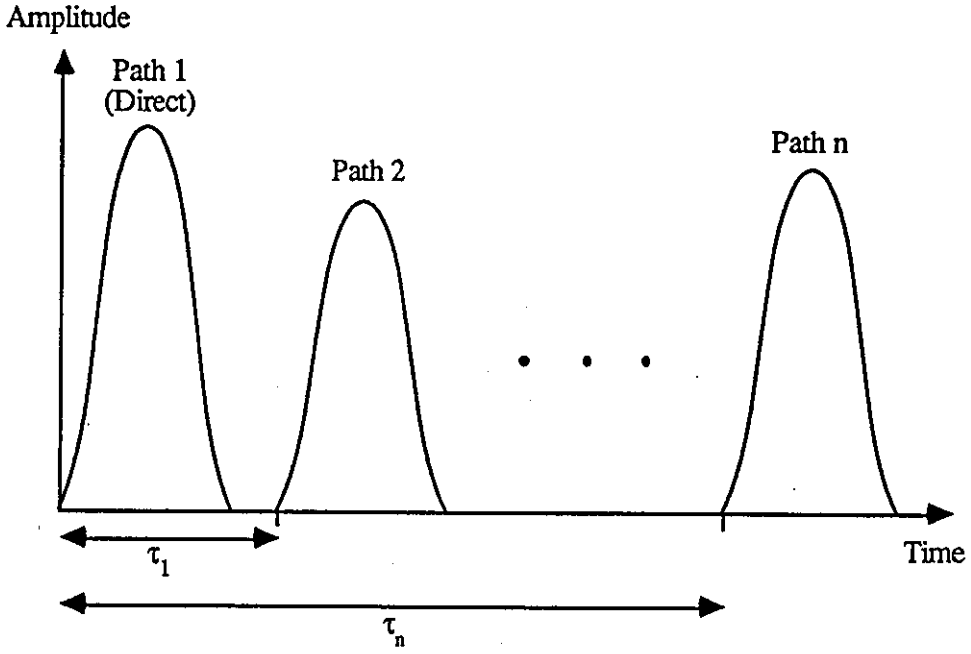


Figure 5.5 Variation of Layer Altitude with Time

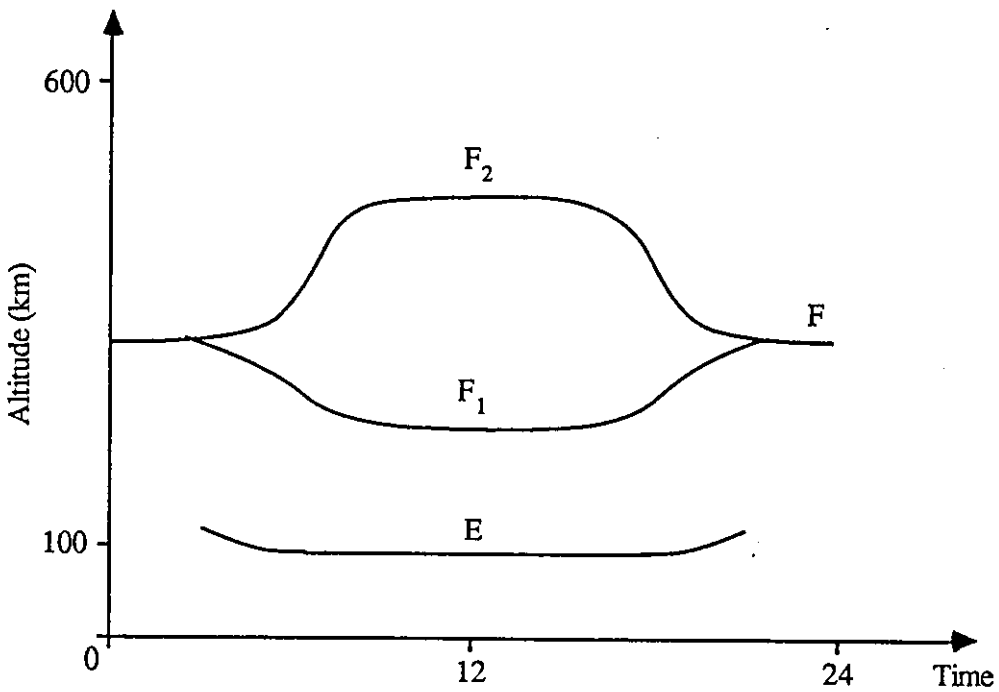


Figure 5.6 Rayleigh Distribution

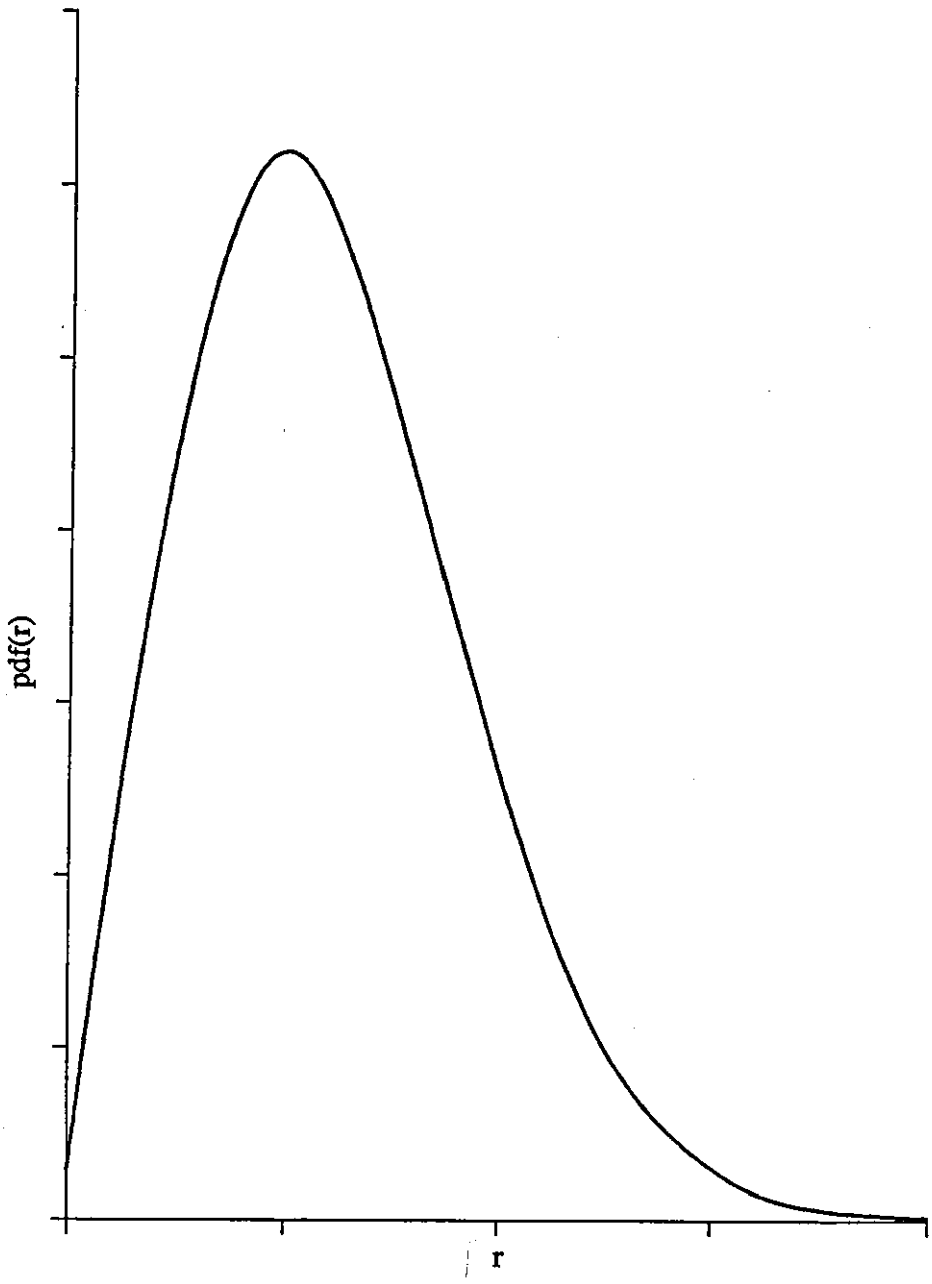


Figure 5.7 Block Diagram of an n path H.F. Radio link Channel Simulator

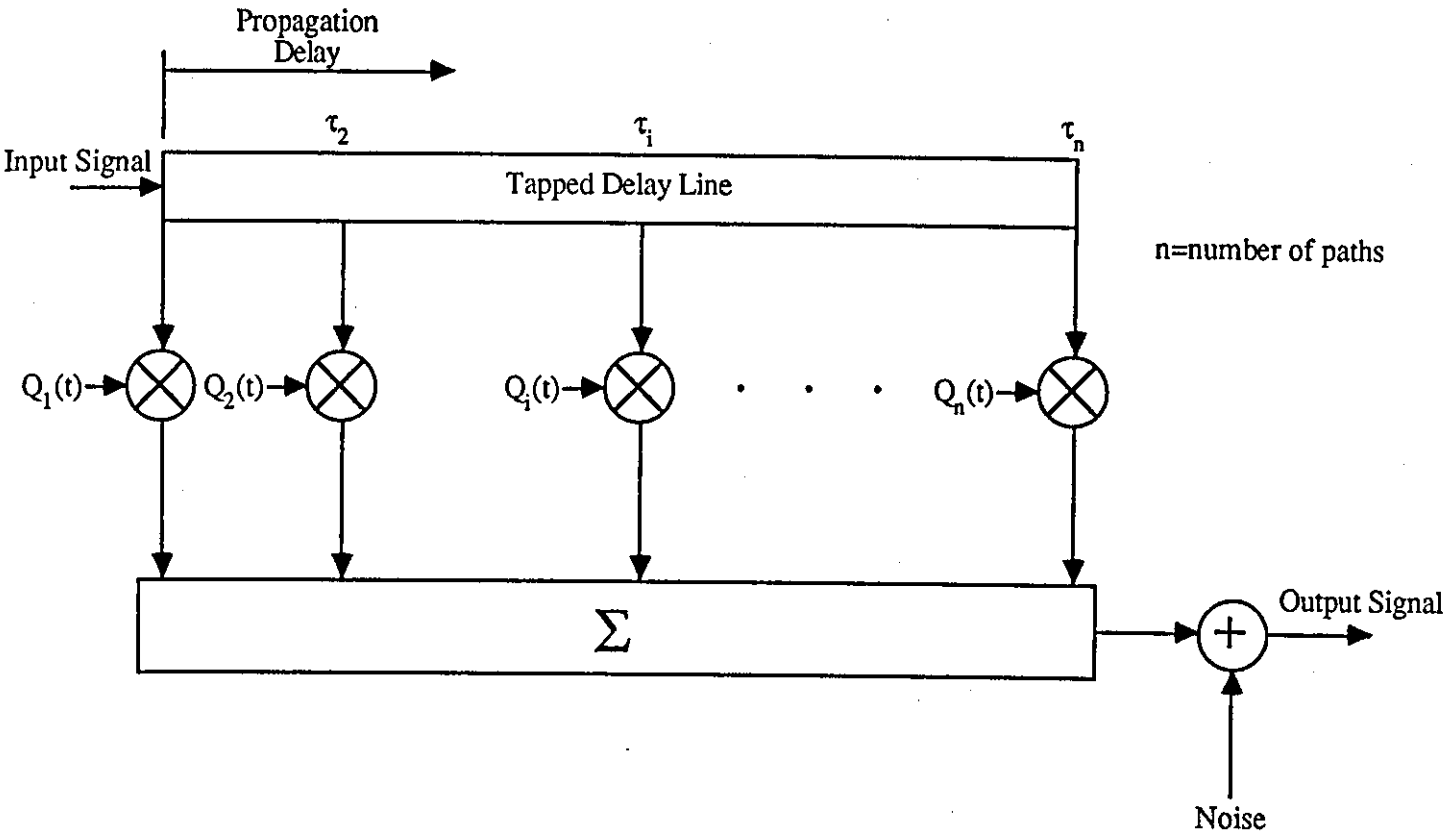


Figure 5.8 Model of a Single Rayleigh Fading Path

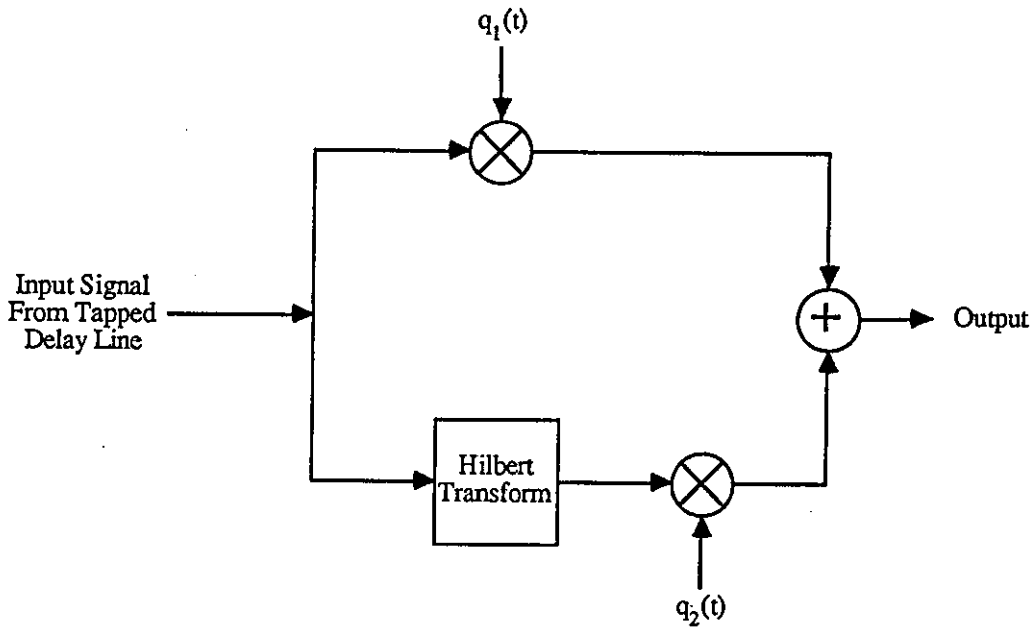


Figure 5.9 Power Spectrum of $q_i(t)$

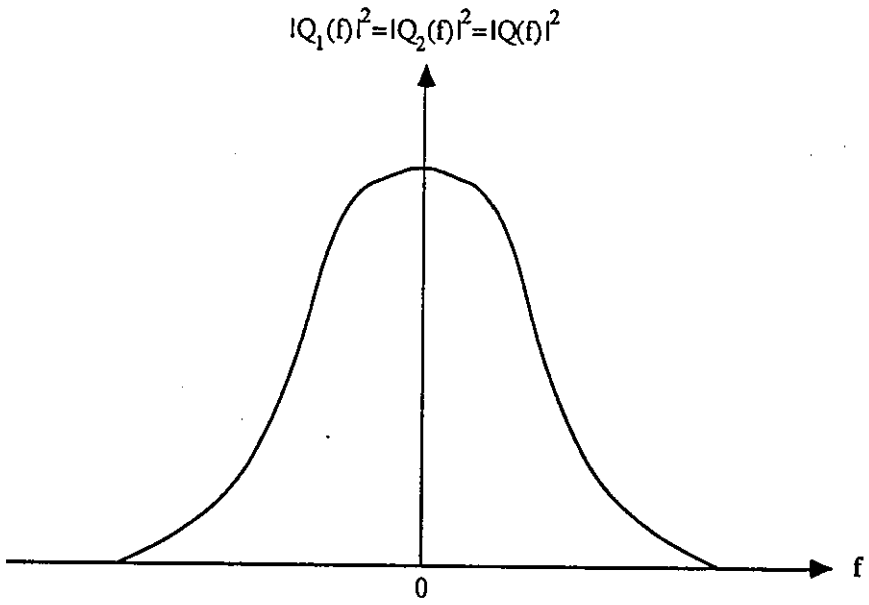


Figure 5.10 Characteristics of the Signals $q_i(t)$

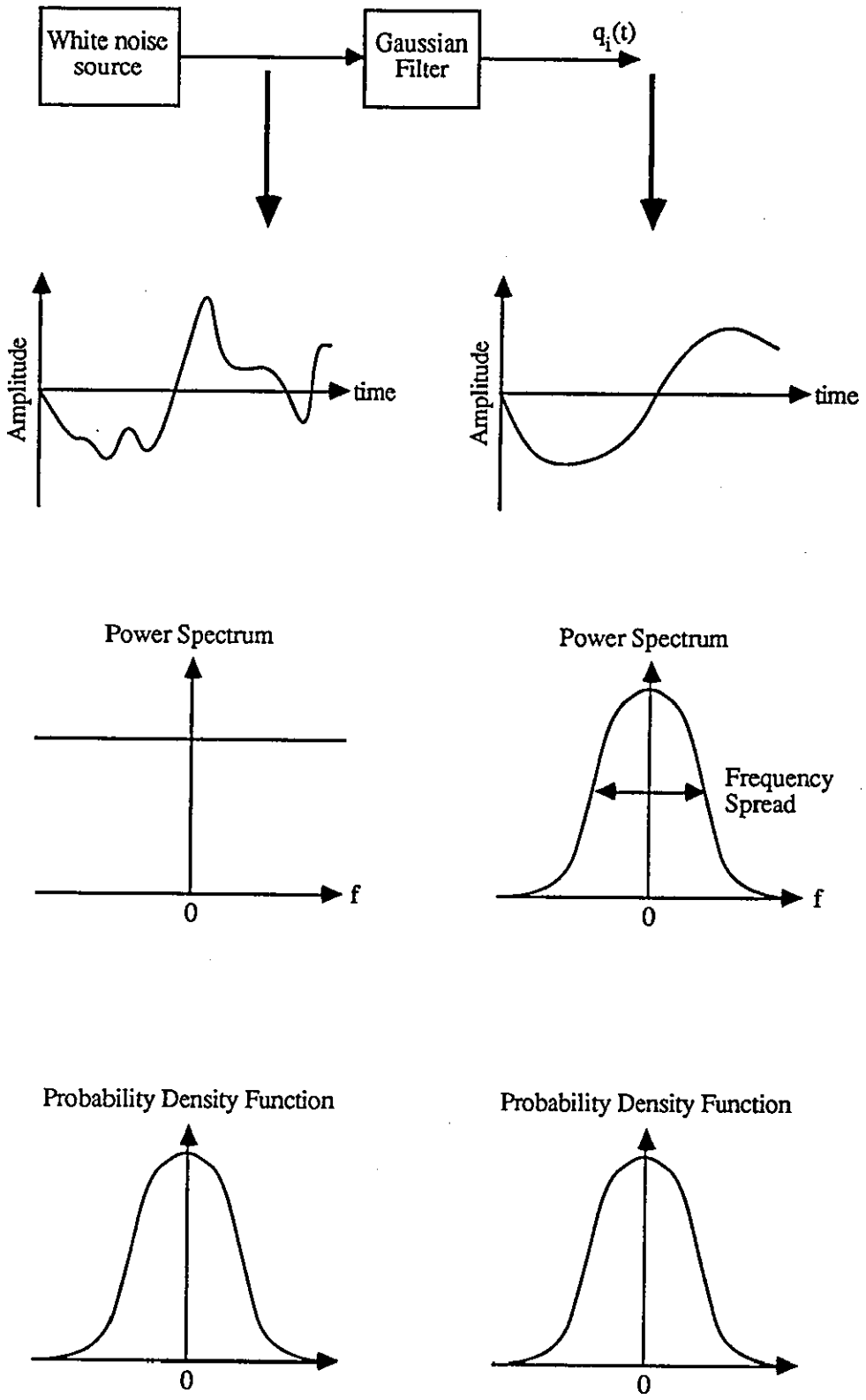


Figure 5.11 Model of a Two Skywave H.F. Radio Link

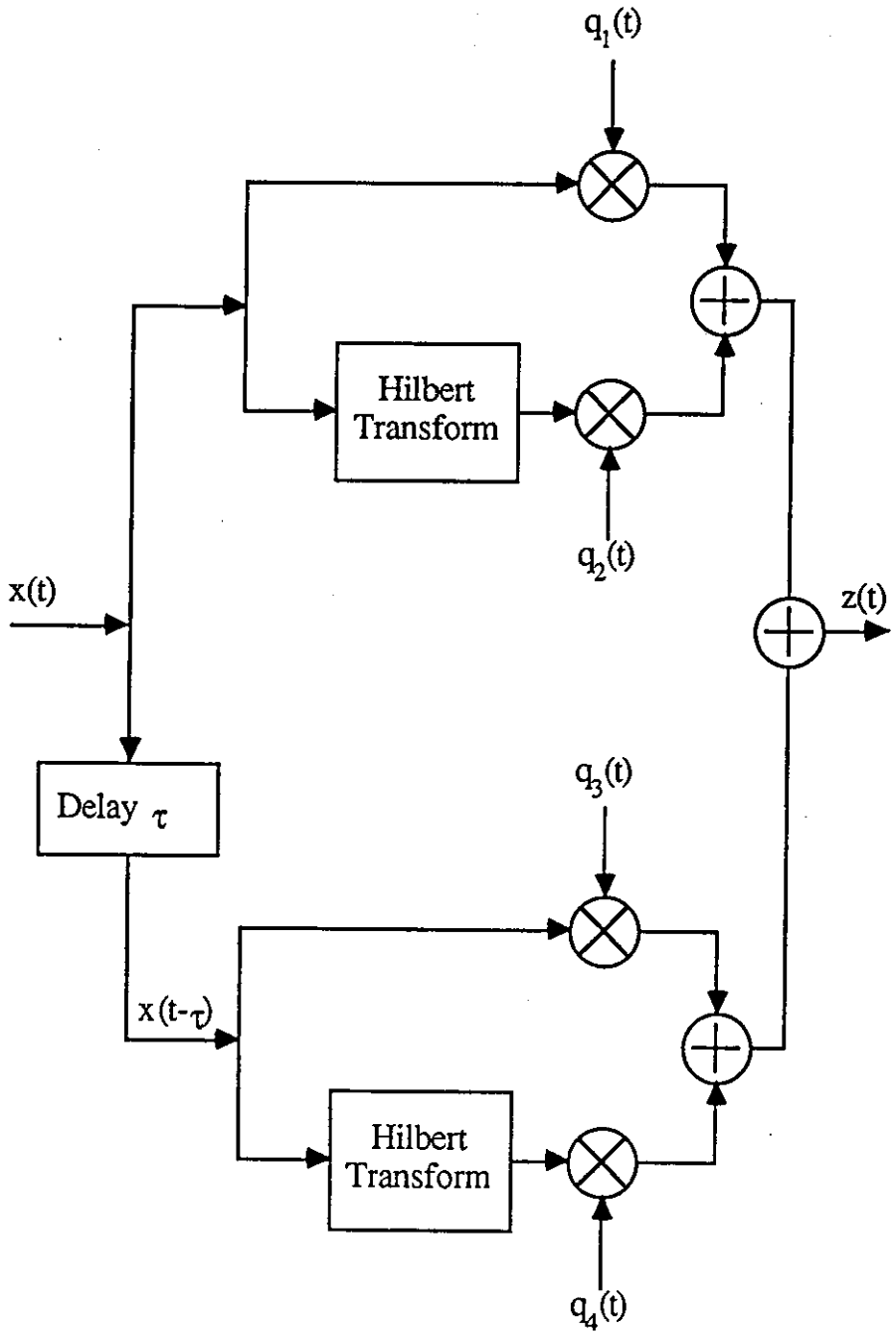


Figure 5.12 Measured Probability Density Function of one of the $q_i(t)$

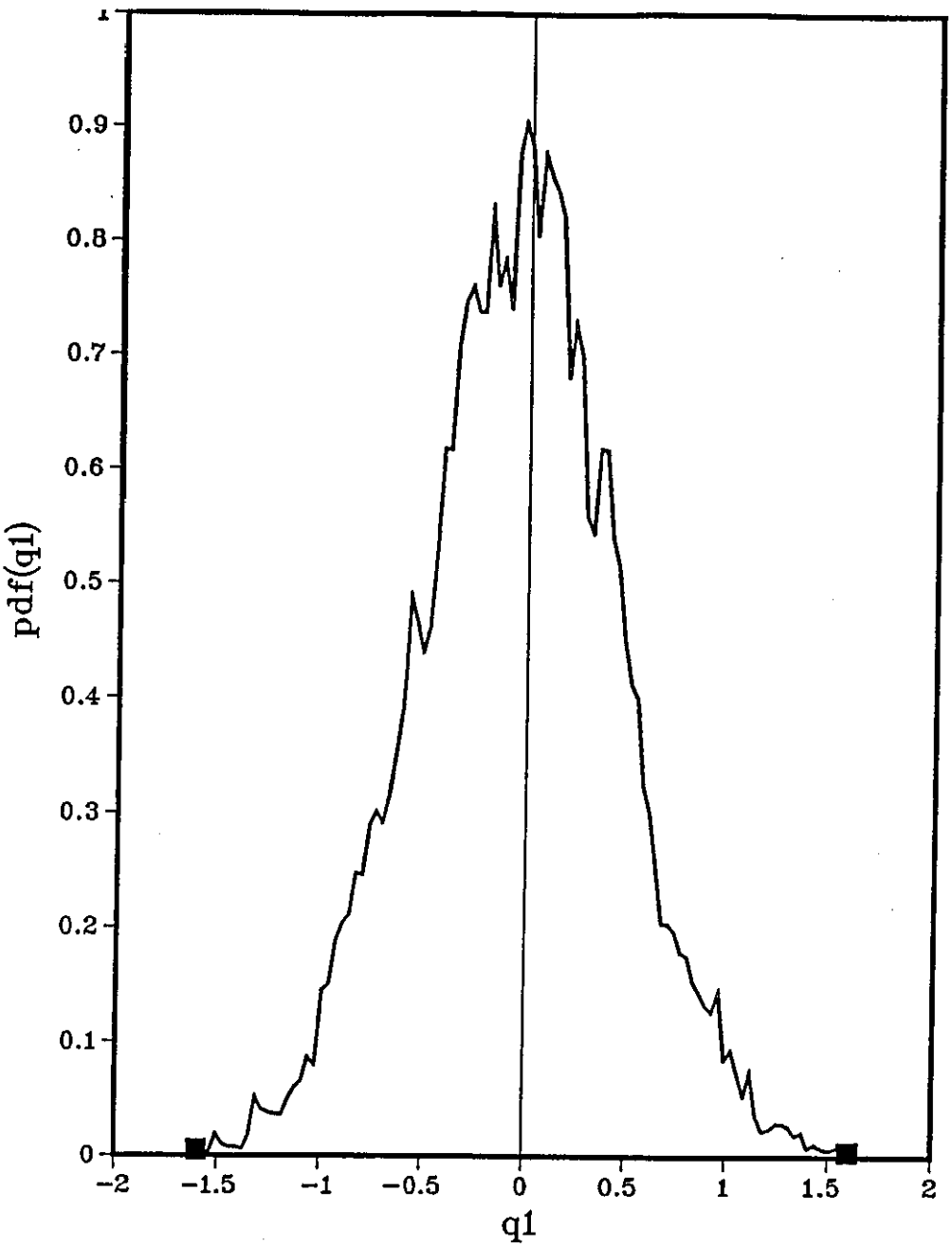
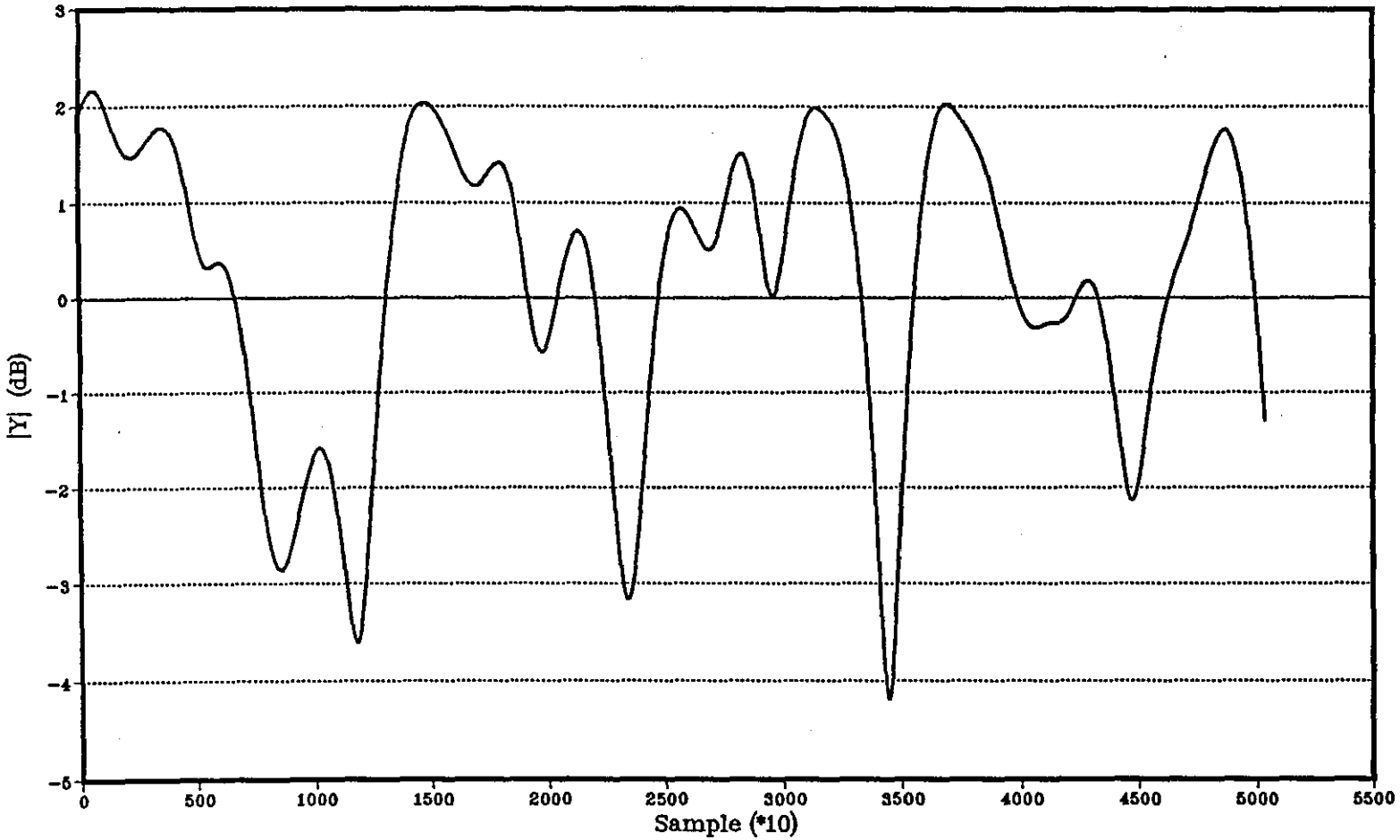


Figure 5.13 Plot Showing the Variation of |Y| with Time



Chapter 6
Transmission over an H.F. radio Link

The main difference between a voice band H.F. radio link and a telephone channel is that the characteristics of the former vary rapidly with time compared with the latter. These rapid changes cause the received signal power to change with time with, sometimes, a complete loss of signal.

Let the model of the QAM transmission system, that includes an H.F. radio link, be as shown in figure 6.1. The binary data stream is first encoded to produce two streams of bipolar impulses $\sum s_{i,0}\delta(t-iT)$ and $\sum s_{i,1}\delta(t-iT)$ which are fed into the two filters C_1 and C_2, both of which have the same real valued impulse response c(t). Filters C_1 and C_2 shape the stream of impulses such that the spectrum of the QAM signal lies within the centre of the voice frequency band (200 to 3200 Hz). The transfer function of filters C_1 and C_2 is given by

$$C(f)=0 \qquad \text{for} \quad f<(-f_c+kf_{sp}) \text{ and } f>(f_c-kf_{sp}) \qquad ..6.1$$

where f_{sp} is the largest frequency spread expected to be introduced by the H.F. channel into the transmitted QAM signal, k is an integer, whose value will be discussed later, and f_c is the carrier frequency of the QAM signal which, in this work, is assumed to have a value of 1800 Hz. The two signals at the output of filters C_1 and C_2 are multiplied (modulated) by two carrier signals that have the same frequency but are in phase quadrature. The outputs of the two modulators are summed to give a QAM signal. This signal is then fed into filter D, which has a real valued impulse response d(t) and prevents any out of band signals entering the transmission path, to give the transmitted QAM signal x(t). Filter D includes any extra filtering that may be carried out when the QAM signal is shifted up to the H.F. frequency range using a single side-band (SSB) modulator. The latter is not shown since it is assumed that the process of SSB modulation and demodulation is linear and therefore has no effect on the system performance. The H.F. radio link introduces Rayleigh fading into the

transmitted signal. At the output of the channel, noise is added and the noisy Rayleigh fading signal is fed into the receiver filter E. It is assumed that white Gaussian noise with zero mean and a two sided power spectral density of $n_0/2$ is added to the signal at the output of the H.F. radio link. Filter E removes any the noise frequencies that lie outside the QAM signal band without excessively distorting it. The real valued impulse response of filter E is e(t). The output of filter E is now coherently demodulated by multiplying it by two reference carriers that have the same frequency f_c', but are in phase quadrature. It is assumed that the average value of f_c' is equal to f_c, thus eliminating any constant frequency offset which may be present in the QAM signal. The two demodulated signals are then lowpass filtered using filters F_1 and F_2, which remove any high frequency components present in the demodulated signal. Filters F_1 and F_2 have the same impulse response f(t), which has a frequency response given by

$$F(f)=0 \qquad \text{for } |f|>f_c \qquad ..6.2$$

From figure 6.1, it can be seen that the transmitted QAM signal x(t) is given by

$$x(t)=\left(\sqrt{2}\sum_i s_{i,0}c(t-iT)\cos 2\pi f_c t - \sqrt{2}\sum_i s_{i,1}c(t-iT)\sin 2\pi f_c t \right) * d(t) \qquad ..6.3$$

where $1/T$ is the signal element rate in baud. If we let

$$s_i = s_{i,0} + j s_{i,1} \qquad ..6.4$$

where s_i may have any one of a finite number of complex values then

$$x(t)=\mathrm{Re}\left[\sqrt{2}\sum_i s_i c(t-iT)e^{j2\pi f_c t} \right] * d(t) \qquad ..6.5$$

or equivalently

$$x(t)=\frac{1}{\sqrt{2}}\left(\sum_i s_i c(t-iT)e^{j2\pi f_c t}+s_i^* c^*(t-iT)e^{-j2\pi f_c t}\right)*d(t) \qquad ..6.6$$

Letting

$$a(t)=c(t)*(d(t)e^{-j2\pi f_c t}) \qquad ..6.8$$

which represents the overall filtering carried out at the transmitter, we get

$$x(t)=\frac{1}{\sqrt{2}}\left(\sum_i s_i a(t-iT)e^{j2\pi f_c t}+s_i^* a^*(t-iT)e^{-j2\pi f_c t}\right) \qquad ..6.9$$

Figure 6.2 shows a model of the H.F. link used in the channel simulator. The Hilbert transform of the transmitted signal x(t) is given by

$$\hat{x}(t)=x(t)*g(t) \qquad ..6.10$$

where $\hat{x}(t)$ is the Hilbert transform of x(t) and the function g(t) is the impulse response of the Hilbert transformer whose Fourier transform is

$$G(f)=\begin{cases} j & f<0 \\ 0 & f=0 \\ -j & f>0 \end{cases} \qquad ..6.11$$

From equations 6.9 and 6.10, the Hilbert transform of x(t) is

$$\hat{x}(t)=\frac{1}{\sqrt{2}}\sum_i s_i\left\{a(t-iT)e^{j2\pi f_c t}\right\}*g(t)+s_i^*\left\{a^*(t-iT)e^{-j2\pi f_c t}\right\}*g(t) \qquad ..6.12$$

which may be expressed alternatively as

$$\hat{x}(t)=\frac{1}{\sqrt{2}}\sum_i s_i\left\{a(t-iT)*g(t)e^{-j2\pi f_c t}\right\}e^{j2\pi f_c t}+$$

$$s_i^*\left\{a^*(t-iT)*g(t)e^{j2\pi f_c t}\right\}e^{-j2\pi f_c t} \qquad ..6.13$$

Now, according to equation 6.8, A(f), the Fourier transform of a(t) is bandlimited to the frequency band occupied by C(f) (equation 6.1). But the Fourier transform of $g(t)e^{-j2\pi f_c t}$(i.e. $G(f+f_c)$) has the value -j over the frequency band $-f_c$ to f_c, whereas the Fourier transform of $g(t)e^{j2\pi f_c t}$(i.e. $G(f-f_c)$) has the value j over the same frequency band, as shown in figure 6.3. Consequently, by taking the Fourier transform of x(t), replacing $G(f+f_c)$ and $G(f-f_c)$ by their corresponding values over the frequency band of A(f) ($-f_c$ to f_c) and taking the inverse Fourier transform, equation 6.13 becomes

$$\hat{x}(t)=\frac{1}{\sqrt{2}}\sum_i -js_i\,a(t-iT)e^{j2\pi f_c t}+js_i^*a^*(t-iT)e^{-j2\pi f_c t} \qquad ..6.14$$

Form figure 6.2, the output signal from the two skywave Rayleigh fading channel is given by

$$z(t)=x(t)q_1(t)+\hat{x}(t)q_2(t)+x(t-\tau)q_3(t)+\hat{x}(t-\tau)q_4(t) \qquad ..6.15$$

where x(t) and $\hat{x}(t)$ are given by equations 6.9 and 6.14 respectively. Thus, the signal at the input of filter E is

$$z(t)=\frac{1}{\sqrt{2}}\sum_i s_i\,a(t-iT)(q_1(t)-jq_2(t))e^{j2\pi f_c t}+$$

$$s_i^* a^*(t-iT)(q_1(t)+jq_2(t))e^{-j2\pi f_c t}+$$

$$s_i a(t-iT)(q_3(t)-jq_4(t))e^{j2\pi f_c(t-\tau)}+$$

$$s_i^* a^*(t-\tau-iT)(q_3(t)+jq_4(t))e^{-j2\pi f_c(t-\tau)} \quad ..6.16$$

Letting

$$h_i(t-iT)=a(t-iT)(q_1(t)-jq_2(t))+a(t-\tau-iT)(q_3(t)-jq_4(t))e^{-j2\pi f_c\tau} \quad ..6.17$$

gives z(t) to be written as

$$z(t)=\frac{1}{\sqrt{2}}\sum_i s_i h_i(t-iT)e^{j2\pi f_c t}+s_i^* h_i^*(t-iT)e^{-j2\pi f_c t} \quad ..6.18$$

If the delay τ is assumed to be constant, and since $q_3(t)$ and $q_4(t)$ are statistically indepenent with zero mean, the factor $e^{-j2\pi f_c\tau}$, which has a magnitude of one, will have no effect on the statistical properties of $(q_3(t)-q_4(t))e^{-j2\pi f_c\tau}$nor will it affect the power spectrum of this signal. Consequently, $h_i(t-iT)$ may be written as

$$h_i(t-iT)=a(t-iT)(q_1(q)-jq_2(t))+a(t-\tau-iT)(q_3(t)-q_4(t)) \quad ..6.19$$

The signal at the output of the receiver filter F in figure 6.1 is

$$r(t)=\sqrt{2}\left\{(z(t)*e(t))e^{-j2\pi f_c t}\right\}*f(t)+\sqrt{2}\left\{(n(t)*e(t))e^{-j2\pi f_c t}\right\}*f(t) \quad ..6.20$$

If we let

$$b(t)=(e(t)e^{-j2\pi f_c' t})*f(t) \qquad ..6.21$$

and

$$w(t)=\sqrt{2}\left\{(n(t)*e(t))e^{-j2\pi f_c' t}\right\}*f(t) \qquad ..6.22$$

then r(t) can be written as

$$r(t)=\sqrt{2}(z(t)e^{-j2\pi f_c' t})*b(t)+w(t) \qquad ..6.23$$

where b(t) represents the overall filtering carried out on the signal at the receiver considered operating on the demodulated baseband signal and w(t) represents the filtered noise signal reaching the detector and is a baseband signal. Assuming that the reference carrier frequency f_c' is equal to the transmitted signal carrier frequency f_c, substituting 6.18 into 6.23 gives

$$r(t)=\sum_i\left\{s_i h_i(t-iT)+s_i^* h_i^*(t-iT)e^{-j4\pi f_c t}\right\}*b(t)+w(t) \qquad ..6.24$$

Now, $h_i(t-iT)$ (equation 6.19) consists of the time-invariant impulse response a(t) and the random components $q_1(t)$ to $q_4(t)$. Each of these random components has a Gaussian-shaped power spectral density which extends over all frequencies and has an r.m.s. frequency of only a few Hertz. But since this power density decreases very sharply as f increases, the frequency response of $h_i(t-iT)$ may be considered to be strictly bandlimited and given by the Fourier transform of a(t), dispersed both upwards and downwards by less than $10f_{rms}$. From equation 6.8, the Fourier transform of a(t) is bandlimited to the frequency band of C(f), the transfer function of filter C, which is defined by equation 6.1. Thus, if the constant k in equation 6.1 is equal to 5 ($f_{sp}=2f_{rms}$) then the frequency response of $h_i(t-iT)$ will be strictly bandlimited such that

$$|H(f)|=0 \quad \text{for } |f|>f_c \qquad ..6.25$$

Hence, the Fourier transform of $(h_i(t-iT))*e^{-j4\pi f_c t}$ in equation 6.24 will be outside the passband of the lowpass filter whose impulse response is b(t) (the passband of the filter whose impulse response is b(t) is zero for $|f|>f_c$ according to equation 6.2 and 6.21) and is therefore blocked by this filter. Thus, equation 6.24 becomes

$$r(t)=\sum_i s_i y_i(t-iT)+w(t) \qquad ..6.26$$

where

$$y_i(t-iT)=h_i(t-iT)*b(t) \qquad ..6.27$$

or (from equation 6.19)

$$y_i(t-iT)=\{a(t-iT)(q_1(t)-jq_2(t))+a(t-\tau-iT)(q_3(t)-jq_4(t))\}*b(t) \qquad ..6.28$$

which is the time varying impulse response of the linear baseband channel.

Using equation 6.28, the equivalent baseband model incorporating a 2-skywave time varying H.F. radio link, can be drawn, as shown in figure 6.4. The impulse responses a(t) and b(t) represent the overall filtering carried out on the transmitted baseband signal at the transmitter and receiver respectively. The signal u(t) is a complex valued Gaussian random variable which, after being fed into filter B, gives the noise component w(t) in the received signal r(t).

The average transmitted energy per signal element at the input to the sampler is [6]

$$\overline{\mathcal{E}_s}=E\left[\int_{-\infty}^{\infty}\left|s_i\left[a(t-iT)(q_1(t)-jq_2(t))+a(t-\tau-iT)(q_3(t)-jq_4(t))\right]*b(t)\right|^2 dt\right]$$

$$=\overline{s_i^2}\left(\overline{q_1^2(t)}+\overline{q_2^2(t)}+\overline{q_3^2(t)}+\overline{q_4^2(t)}\right)\int_{-\infty}^{\infty}|A(f)|^2|B(f)|^2 df \qquad ..6.29$$

where Parseval's theorem has been used, E[.] denotes expected value and

$$\overline{s_i^2}=E\left[|s_i|^2\right] \qquad ..6.30$$

The quantities $q_1(t)$, $q_2(t)$, $q_3(t)$ and $q_4(t)$ are the variances of $q_1(t)$, $q_2(t)$, $q_3(t)$ and $q_4(t)$ respectively. In the H.F. model, these four variances are equal and have a value of 0.25. This value was chosen to simplify the signal-to-noise ratio calculations (Appendix B), it also means that, on average, the two skywaves don't introduce any gain or attenuation into the transmitted signal. If the transmitter and receiver filters are chosen such that

$$\int_{-\infty}^{\infty}|A(f)|^2|B(f)|^2 df=1 \qquad ..6.31$$

then

$$\overline{\mathcal{E}_s}=\overline{s_i^2} \qquad ..6.32$$

Thus for a 16-level QAM signal, where the real and imaginary parts of s_i take on the values ±1 or ±3, the value of s_i^2 is 10, likewise for a 4-level QAM signal $s_i^2=2$. The value of $\overline{\mathcal{E}_s}$ is used in the signal-to-noise ratio calculations in Appendix B.

6.1 Model of the H.F. Data Transmission System used in Tests

The data transmission system is a synchronous serial system, with a 4 or 16-level QAM signal and an adaptive detection process. Figure 6.5 shows the model used in the tests. The data that is to be transmitted is in the form of binary digits, that are statistically independent and equally likely to have either of their binary value. These binary digits are encoded, using differential encoding, to produce a stream of impulses $\sum s_i \delta(t-iT)$, that are fed into the linear baseband channel. The data symbols, $\{s_i\}$, are statistically independent and equally likely to have any one of their m possible values (where m=4 or 16). The received signal at the output of filter B is

$$r(t)=\sum_i s_i y_i(t-iT)+w(t) \qquad ..6.33$$

where $y_i(t-iT)$ is the time varying impulse response of the linear baseband channel, given by

$$y_i(t-iT)=\left\{a(t-iT)(q_1(t)-jq_2(t))+a(t-\tau-iT)(q_3(t)-jq_4(t))\right\}*b(t) \qquad ..6.34$$

and a(t) and b(t) are the combined impulse responses of the transmitter and receiver filters respectively, where

$$a(t)=c(t)*(d(t)e^{-j2\pi f_c t}) \qquad ..6.35$$

and

$$b(t)=(e(t)e^{-j2\pi f_c t})*f(t) \qquad ..6.36$$

The performance of the near maximum likelihood detector is improved by making the impulse response of the linear baseband channel minimum phase. This is achieved by placing the pre-filter ahead of the detector and the filter adjusted such that the combined

impulse response of the baseband channel and the pre-filter is minimum phase. The pre-filter moves the roots of the z-transform of the channel, Y(z), that lie outside the unit circle to the reciprocal of their conjugate positions. To reduce the amount of processing time involved in calculating the taps weights of the pre-filter, filters A and B in figure 6.5 are made minimum phase by processing their roots outside the unit circle, as described above. Let the minimum phase versions of filters A and B have the impulse responses a'(t) and b'(t) and substituting these for a(t) and b(t) in equation 6.34 gives

$$y_i(t-iT)=\left\{a'(t-iT)(q_1(t)-jq_2(t))+a'(t-\tau-iT)(q_3(t)-jq_4(t))\right\}*b(t) \qquad ..6.37$$

At the receiver, the demodulated baseband signal, r(t), comprises the stream of signal elements $\{s_i'y_i(t-iT)\}$ to which is added the stationary zero mean complex valued Gaussian noise signal w(t), the filtered version of u(t). The waveform r(t) is sampled once per data symbol period T, at times t=iT. It is assumed, for convenience, that the delay in transmission is zero, such that the first potentially non-zero sample of a received signal element arrives with no delay. Thus, the received sample, at time t=iT, is

$$r_i=\sum_{h=0}^{g} s_{i-h}y_{i,h}+w_i \qquad ..6.38$$

where

$$y_{i,h}=y_{i-h}(hT) \qquad ..6.39$$

$$w_i=w(iT) \qquad ..6.40$$

and $y_{i,h}=0$ for $h<0$ and $h>g$, for practical purposes. The sequence of complex values given by the row vector

$$Y_i=[y_{i,0}y_{i,1}\cdots y_{i,g}] \qquad ..6.41$$

is taken to be the sampled impulse response of the linear baseband channel at time t=iT. The pre-filter in figure 6.5, moves the roots of the z-transform of the sequence Y_i that lie outside the unit circle to the reciprocal of their conjugate positions without, however, introducing any amplitude distortion. The early detected data symbol s_i" is obtained, with no delay in detection, from the last component in the expanded vector which has the smallest cost giving the value of the detected data symbol s_i'. The early detected data symbols {s_i"} are used along with the delayed samples {r_i}, to give an estimate of the time varying sampled impulse response Y_{i+1}', at time t=(i+1)T.

The Clark-Hau algorithm (described in chapter 4) makes use of the estimate of the sampled impulse response Y_{i+1}' to calculate the tap gains of the pre-filter and, also, provide an estimate of the minimum phase sampled impulse response for the detector. For time varying channels the Clark-Hau algorithm is modified. Firstly, the number of iterations that are allowed before a new starting position is tried is increased to 100, also the number of starting points tried is increased to nine, as shown in figure 6.6 [16]. Once the locations of the roots outside the unit circle are found, they are used as the starting positions the next time the Clark-Hau algorithm is run. Using the previously found roots in this way reduces the number of iterations the algorithm goes through to locate a root, i.e. it effectively tracks the roots outside the unit circle. Previous work has shown that the pre-filter taps can be updated only once every four sampling intervals without unduly affecting the system performance.

Before any data is transmitted, the channel estimator is allowed to converge to an accurate estimate of the baseband channel by sending a known sequence of data symbols to the receiver. After this initial startup period, the actual transmission of data can start and the channel estimator uses the early detected data {s_i"} to track any changes in the sampled impulse response. Usually, at high signal-to-noise ratios, s_i"=s_i. However, if a deep fade occurs, particularly at lower signal-to-noise ratios, errors can occur in the early detected data {s_i"}. These errors cause the channel estimator to loose track and hence the adaptive filter to be adjusted incorrectly and the

estimate of the minimum phase impulse response used by the detector to be erroneous. The detector will therefore fail to give the correctly detected data symbols $\{s_i'\}$ and $\{s_i''\}$, which will lead to the channel estimator to become even less accurate. Eventually this feedback process will lead to the whole system to collapse.

To prevent this collapse from occurring a training sequence is periodically sent [16], approximately once every 0.5 seconds, as shown in figure 6.7. Sending this training sequence regularly allows the estimator to correct itself if it has lost track, the price to pay for this is around 10% of the symbol rate is used for retraining. At the end of the retraining period, the vectors of the near maximum likelihood detector are set to their correct values and the costs of one of these vectors set to zero, while the costs of the others is set to a high value so that they are discarded during the next detection process.

In the computer simulation tests, the simple linear feedforward channel estimator is used. The reason for choosing this estimator is due to its simplicity and the ease with which it could be implemented in real time on a TMS320C25. More complex channel estimators have been described [16,68], that make use of some prior knowledge of the basic structure of the H.F. radio link in the estimation process. Unfortunately, these estimators involve a large number of computations, including division, for them to be updated once every sampling period

In the tests it was found that when the H.F. channel characteristics were varying rapidly with time, that the accuracy of the linear feedforward estimator could be improved by adding degree 1 simple fading-memory prediction [57]. The linear feedforward estimator with prediction operates as follows. At time t=iT the estimator holds in store the estimate of the sampled impulse response Y_i denoted by Y_i'. A prediction of the sampled impulse response of the channel at time t=(i+1)T, Y_{i+1}, is now made using

$$Y'_{i+1,i}=Y_i'+\Delta_{1,i} \qquad ..6.42$$

where $Y'_{i+1,i}$ is the prediction of Y_{i+1} based upon the previous estimate Y_i' at time t=iT and $\Delta_{1,i}$ is a vector given by

$$\Delta_{1,i}=\Delta_{1,i-1}+c(Y'_i-Y'_{i-1}-\Delta_{1,i-1}) \qquad ..6.43$$

where c is a small positive real constant. Initially, at the start of the estimation process

$$Y'_0=0 \qquad ..6.44$$

and

$$\Delta_{1,0}=\Delta_{1,-1}=0 \qquad ..6.45$$

Thus, at time t=(i+1)T the prediction of Y_{i+1} (i.e. $Y'_{i+1,i}$) is used by the linear feedforward channel estimator instead of Y_i' to give the new estimate of the sampled impulse response Y'_{i+1}.

The components of the vector Y_i are obtained by sampling the $\{y_i(t-iT)\}$ at a rate of 2400 samples/second. Since the process is performed in the discrete time domain and to avoid aliasing likely to occur when any of the $q_1(t)$, $q_2(t)$, $q_3(t)$ or $q_4(t)$ are changing rapidly, the convolution in equation 6.37 is carried out in the discrete time domain at a sampling rate of 4800 samples/second which is well above the Nyquist rate of filters A and B. The functions $q_1(t)$, $q_2(t)$, $q_3(t)$ or $q_4(t)$ are generated as described in chapter 5. Let the three sequences A_1, A_2 and B represent the three impulse responses a'(t), a'(t-t) and b'(t) respectively, sampled at 4800 samples/second, i.e.

$$A_1=[a'_{1,0}\ a'_{1,1}\ \ldots\ a'_{1,p}]$$

$$A_2=[a'_{2,0}\ a'_{2,1}\ \ldots\ a'_{2,p}] \qquad ..6.46$$

$$B=[b'_0\ b'_1\ \ldots\ b'_p]$$

where

$$a'_{1,h}=a'\left(\frac{hT}{2}\right)$$

$$a'_{2,h}=a'\left(\frac{hT}{2}-\tau\right) \qquad ..6.47$$

$$b'_{h}=b'\left(\frac{hT}{2}\right)$$

and p is an integer, the value of which will become clear later and $1/T$ is the data symbol rate of 2400 samples/second. For practical purposes

$$a'(t)=b'(t)=0 \quad \text{for } t<0 \text{ and } t>\frac{(p-p')T}{2} \qquad ..6.48$$

where p' is an integer such that the delay τ is

$$\tau=p'\frac{T}{2}+\tau' \qquad ..6.49$$

and

$$\tau'<\frac{T}{2} \qquad ..6.50$$

This implies that

$$a'_{1,h}=b'_{h}=0 \qquad \text{for } h<0 \text{ and } h>p-p' \qquad ..6.51$$

$$a'_{2,h}=0 \qquad \text{for } h<p' \text{ and } h>p \qquad ..6.52$$

It can be seen that the multipath propagation delay τ is expressed as a whole number of sampling periods p' plus a fraction of a sampling instant τ'. The four functions $q_1(t)$,

$q_2(t)$, $q_3(t)$ or $q_4(t)$ are sampled at 4800 samples/second and the corresponding samples upto time t=iT are given by the components of the four row vectors

$$Q_{1,i}=[q_{1,0}q_{1,1}\cdots q_{1,2i}]$$

$$Q_{2,i}=[q_{2,0}q_{2,1}\cdots q_{2,2i}]$$

$$Q_{3,i}=[q_{3,0}q_{3,1}\cdots q_{3,2i}]$$

$$Q_{4,i}=[q_{4,0}q_{4,1}\cdots q_{4,2i}] \qquad ..6.53$$

where $q_{j,h}=q_j(hT/2)$, k=1,2,3,4, for h=0,1, . . . ,2i. From equation 6.37 and equations 6.42 and 6.49, the components of the vector Y_i, at time t=iT, are given by the discrete convolution

$$y_{i,h}=\frac{T}{2}\sum_{k=0}^{2h}[a'_{1,k}(q_{1,(2(i-h)+k)}-jq_{2,(2(i-h)+k)})+$$

$$a'_{2,k}(q_{3,(2(i-h)+k)}-jq_{4,(2(i-h)+k)})]b'_{2h-k} \qquad ..6.54$$

for h=0,1, . . . ,g, where from equations 6.47 and 6.48, it can be seen that [16]

$$g=\frac{2p-p'+1}{2} \qquad ..6.55$$

Thus, for a delay of 0.5 ms, p'=1, p=16 and g=16. Notice, also, that the $\{y_{i,h}\}$ are given at a sampling rate of 2400 samples/second.

The frequency characteristics of the combined filters A and B, used in the tests, are shown in figure 6.8 where, for convenience, they have been shifted up in frequency to the pass band of the voice band channel. If the H.F. radio link introduces no fading, attenuation, or multipath propagation, then the impulse response of the linear baseband

channel will be given by a'(t)*b'(t), where a'(t) and b'(t) are the minimum phase versions of the impulse responses a(t) and b(t). Also, for the signal-to-noise ratio at the output of the sampler to be maximum, the transfer functions of filters A and B are adjusted such that |A'(f)|=|B'(f)| [2,34] (i.e. the receiver filter is matched to the transmitter filter), where A'(f) and B'(f) are the Fourier transforms of a' (t) and b'(t) respectively. From the frequency characteristics of the combined filters, shown in figure 6.8, and the property just mentioned, the sequences A_1, A_2 and B in equation 6.42 are calculated using the inverse fast Fourier transform. The derivation of these sequences is given else-where [11,68]. The resulting sequences are shown in tables 6.1 to 6.5, where the delayed sequence A_2, in equation 6.42, is given by tables 6.3, 6.4 or 6.5, depending on whether a multipath propagation delay of 0.5, 1.1 or 3 ms is to be used in the computer simulations.

6.2 Computer Simulation Tests

Computer simulation tests were carried out to show the performance (in terms of the bit error rate) of the combined, channel estimator, Clark-Hau algorithm, and near maximum likelihood detector, operating at 9600 bits/s and 4800 bits/s over an H.F. channel. The simulations were set up such that at the receiver detector A was used (chapter 3), the pre-filter had 30 taps, the channel estimator was retrained approximately once every 0.5 seconds and the Clark-Hau algorithm was run once every four sampling intervals.

To contain the time taken for the simulations to run, only 50000 data symbols were sent for every bit error rate measurement made [16]. Also, for each run, different sequences of data symbols and noise samples were generated by the random number generators.

Figure 6.9 shows the performance of the complete system, operating at 9600 bits/s over a slowly varying 2-skywave H.F. radio link. It can be seen that the system performance using the linear feedforward channel estimator with integer arithmetic is very poor. Even with perfect channel estimation, the performance of the limited precision Clark-Hau algorithm and detector not good enough for reliable communications. These results are disappointing, but, perhaps expected since the

16-bit integers do not provide the necessary dynamic range needed to cope with the fading in the received signal. To cope with these variations in the received signal level it would be better to use floating point arithmetic. At the present time there are several high speed digital signal processors appearing that provide both integer and floating point arithmetic. For example, the Texas Instruments TMS320C30 has 32-bit floating point arithmetic and operates at almost twice the speed of the TMS320C25. Unfortunately, this processor doesn't have a hardware divider (needed in the Clark-Hau algorithm) and so division would still have to be carried out in software. The Intel i860 is one of the latest signal processors to appear, this has 64-bit floating point arithmetic (which can be configured as two 32-bit numbers for complex number manipulation), operates at twice the speed of the TMS320C25 and has a built in hardware divider which, when pipelining is used, will give the result of a division once every machine cycle. In figure 6.9 the performance of the system with perfect channel estimation using either 32-bit or 64-bit floating point arithmetic is shown.

Figures 6.11 to 6.14 show the performance of the complete system, including the channel estimator, operating at 4800 bits/s over various H.F. channels. In the case where the H.F. channel introduced a 2 Hz frequency spread it was found necessary to include the prediction algorithm mentioned in the previous section. Figure 6.10 shows that the accuracy of the linear feedforward estimator is improved when prediction is added. Figure 6.14 shows the overall performance of the system with prediction.

The results of the system operating at 9600 bits/s are not very encouraging, even with such a mild channel as that used in the tests. They suggest that to obtain a reliable performance at this rate floating point arithmetic is needed. At 4800 bits/s, however, the results indicate it may be possible to achieve reliable communications using 16-bit integer arithmetic at the receiver, but only with mild channels. With H.F. channels introducing a frequency spread of 2 Hz or those with a multipath propagation delay of 3 ms the reliability of the system is poor.

Table 6.1 Sampled Impulse Response of the Receiver Filter

Real	Imaginary
-1.9417691	1.3625952
-15.9797864	11.5941040
-35.1417733	27.3342937
-34.4788717	28.0870086
-11.2301982	7.2714615
7.8155160	-9.2602472
7.5124057	-5.0954462
-0.5057505	3.2326498
-3.3707125	1.8975352
-0.6759166	-1.2813604
1.0482656	-0.4830313
0.3621876	0.7614804
-0.3105902	0.1979014
0.0438410	-0.1532672
0.0738947	0.0940330
-0.0646936	-0.0312132

Table 6.2 Sampled Impulse Response of the Transmitter Filter with no Delay

Real	Imaginary
-0.1795896	2.3539405
-3.0773455	20.7590237
-9.9409021	45.5584592
-11.7869473	41.4909978
-3.4618271	8.7045826
4.4438154	-11.7869820
3.0642536	-5.5819054
-1.3596576	3.1582131
-1.4973528	1.7365460
0.2925598	-0.7776891
0.5180829	-0.1292556
-0.1842786	0.2880296
-0.3167778	-0.2324818
0.0021899	-0.2107548
-0.0443806	0.0392056
0.0515533	0.0098505

Table 6.3 Sampled Impulse Response of Transmitter Filter with a Delay of 0.5 ms

Real	Imaginary
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
-0.7630237	7.3452371
-5.6487240	31.9050405
-11.921618	48.7718190
-9.3588701	29.8079459
0.5649945	-3.0207994
4.9376234	-11.4979467
1.0473111	-0.9822748
-1.9765922	3.5053270
-0.7164903	0.3116039
0.5943870	-0.7219063
0.2543641	0.2044829
-0.3636354	0.1084602
-0.1543700	-0.3286970
-0.0228239	-0.0636352
0.0167220	0.0278816
-0.0609485	0.0185930

Table 6.4 Sampled Impulse Response of Transmitter Filter with a Delay of 1.1 ms

Real	Imaginary
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
-1.6694374	13.2372707
-7.8492148	39.6493461
-12.3887079	46.9272219
-6.6023157	19.2346609
2.9408554	-8.8804125
4.3005084	-9.0256163
-0.3368383	1.6284281
-1.9014342	2.8139013
-0.1433592	-0.4311352
0.6242601	-0.4537174
0.0278577	0.3081762
-0.3820071	-0.0772327
-0.0416905	-0.3043271
-0.0439705	0.0085057
0.0749333	0.0093809
-0.0594132	0.0094992

Table 6.5 Sampled Impulse Response of the Transmitter Filter with a Delay of 3 ms

Real	Imaginary
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
-1.3136537	11.0688962
-7.1104051	37.2136597
-12.3469721	47.9575159
-7.5848703	22.8262482
2.2353854	-7.2498590
4.5938614	-10.0026703
0.0931639	0.8695437
-1.9704176	3.1072800
-0.3233694	-0.2261096
0.6313238	-0.5552906
0.1035718	0.2882096
-0.3865939	-0.0156703
-0.0734526	-0.0321577
-0.0386471	-0.0107706
0.0608046	0.0140909
-0.0713496	0.0135711

Figure 6.1 Model of the Digital Data Transmission System

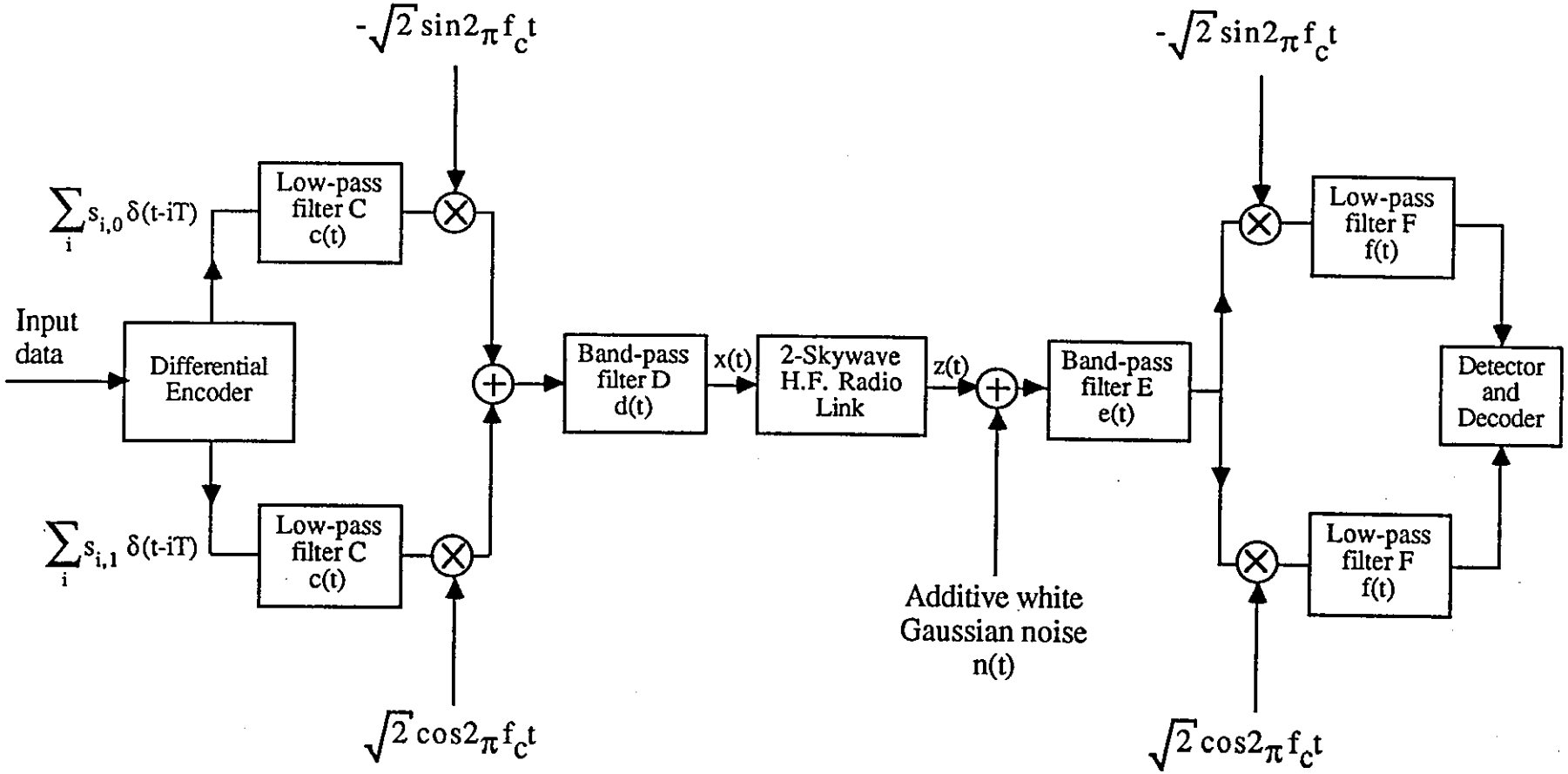


Figure 6.2 Model of a Two Skywave H.F. Radio Link

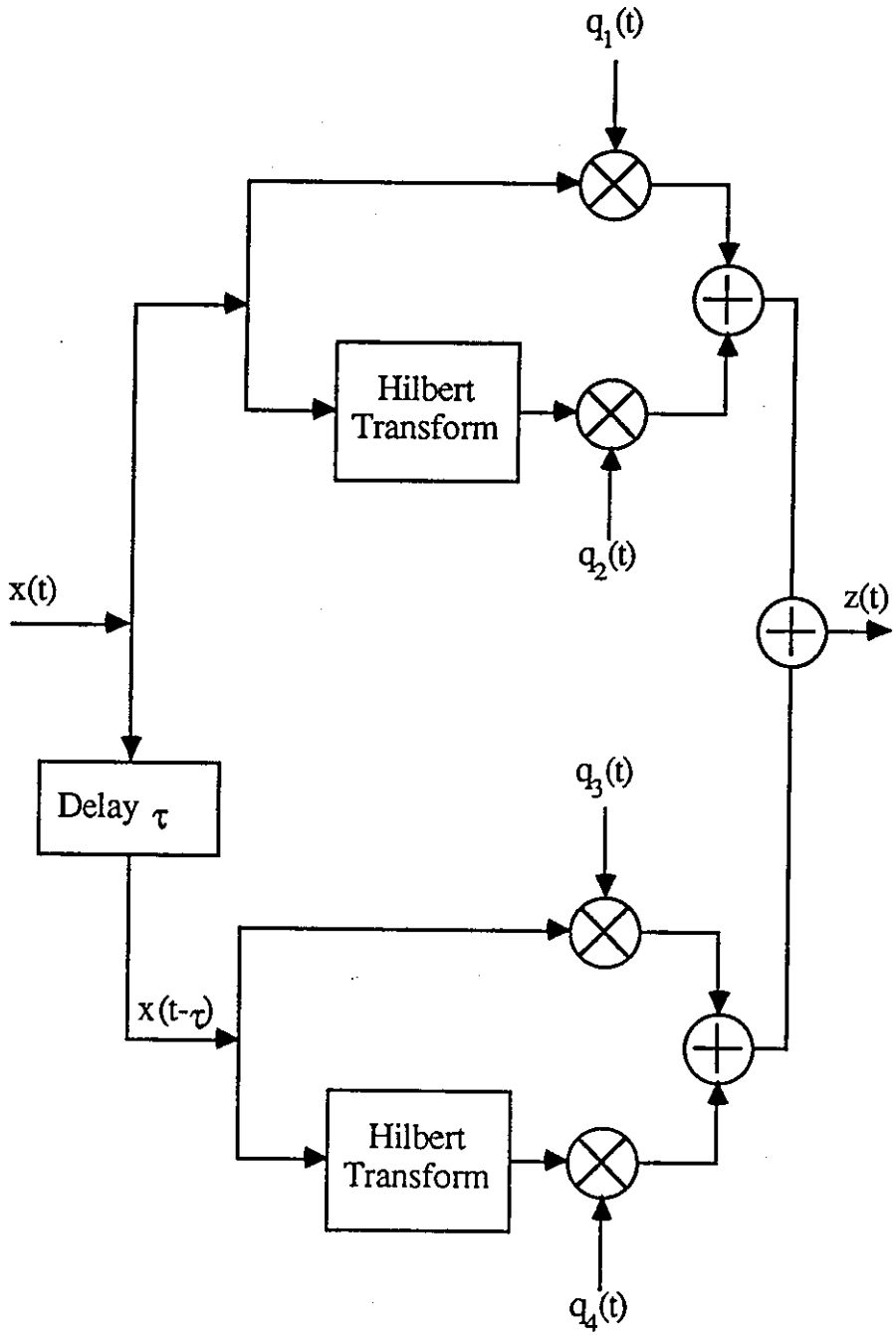


Figure 6.3 Frequency Characteristics of the Hilbert Transform

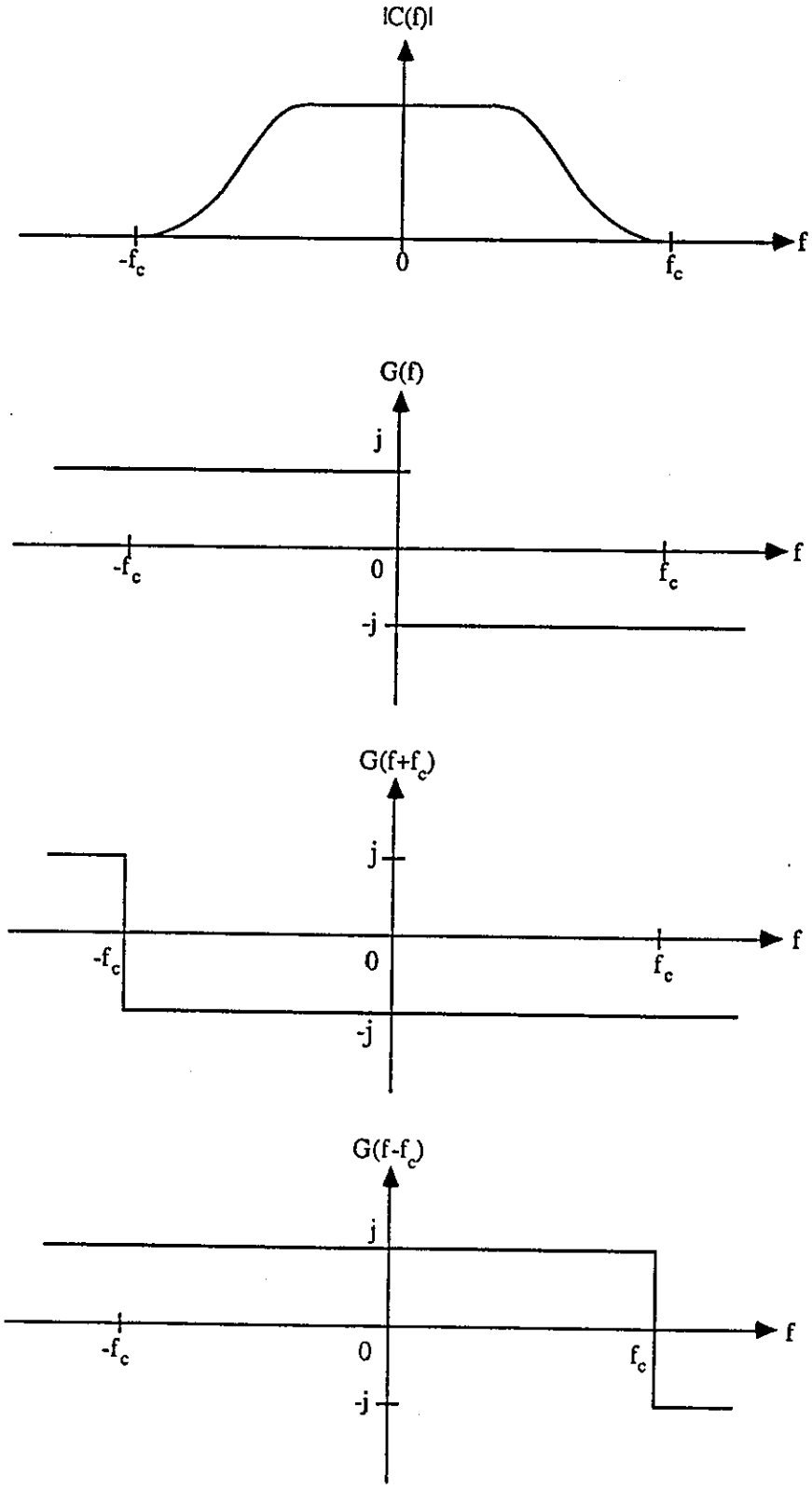


Figure 6.4 Equivalent Baseband Model of the H.F. Radio Link

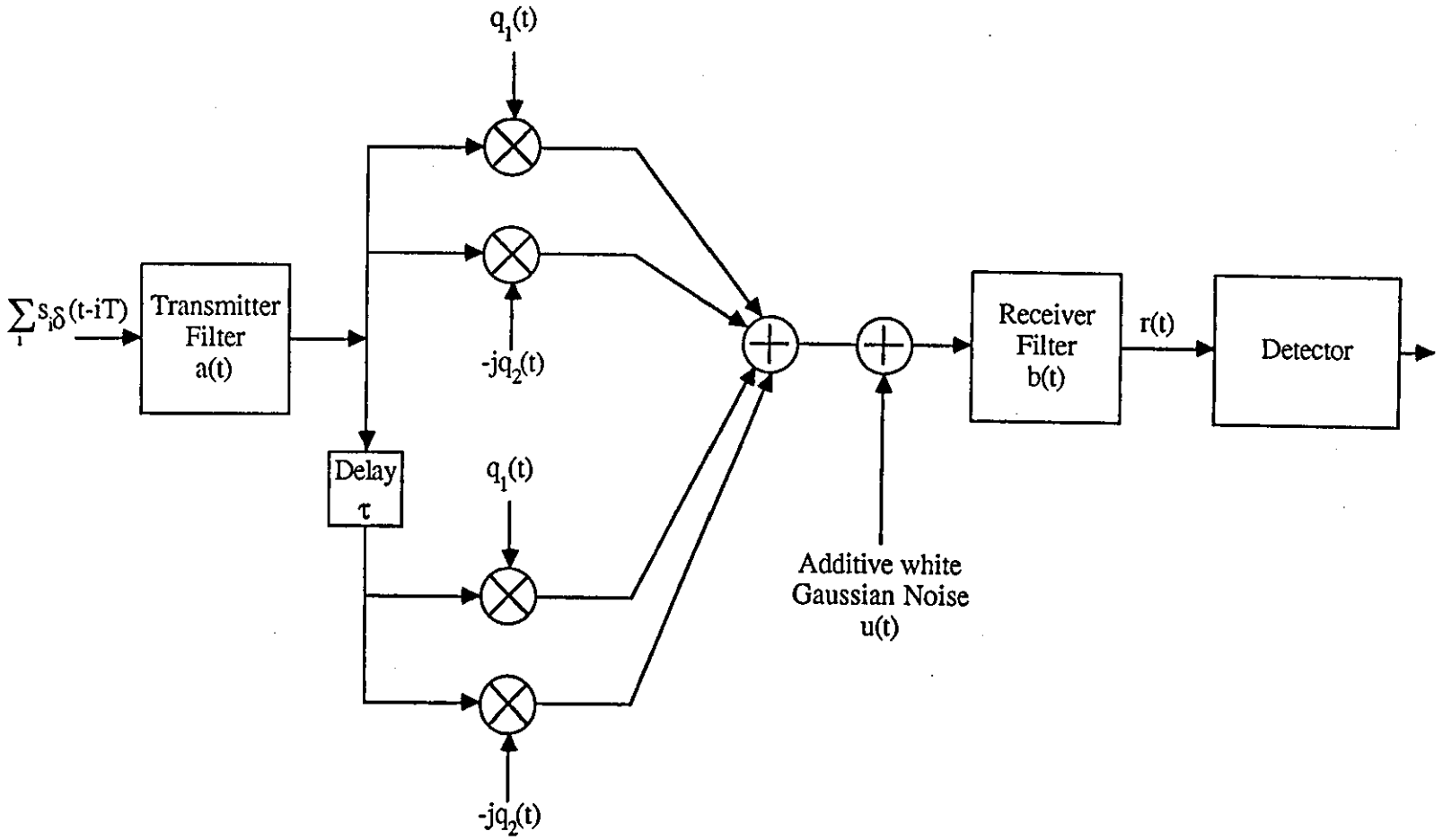


Figure 6.5 Model of the Digital Data Transmission System used in the Computer Simulations.

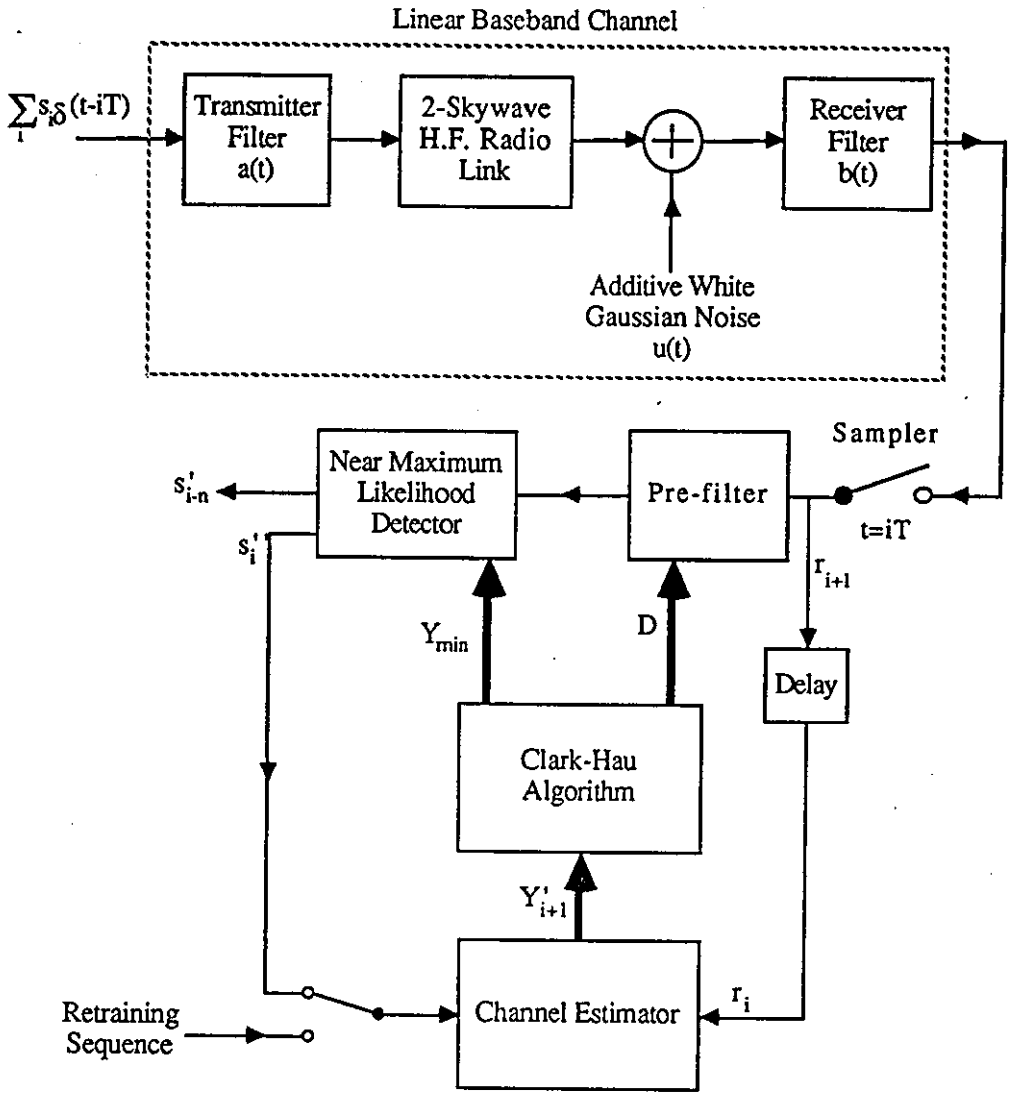


Figure 6.6 Starting Points used with an H.F. Channel

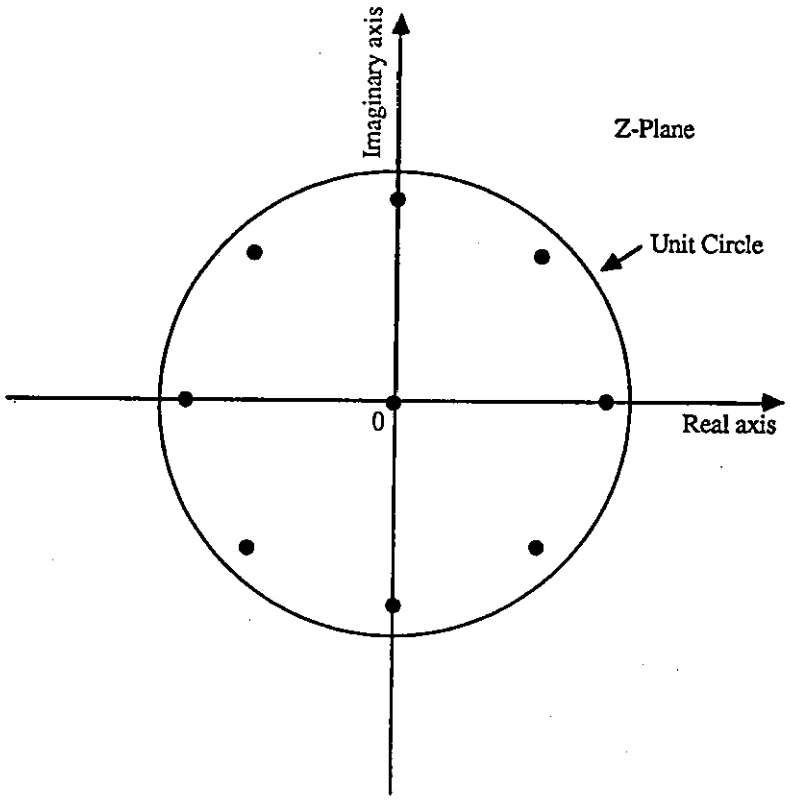


Figure 6.7 Retraining of the Channel Estimator

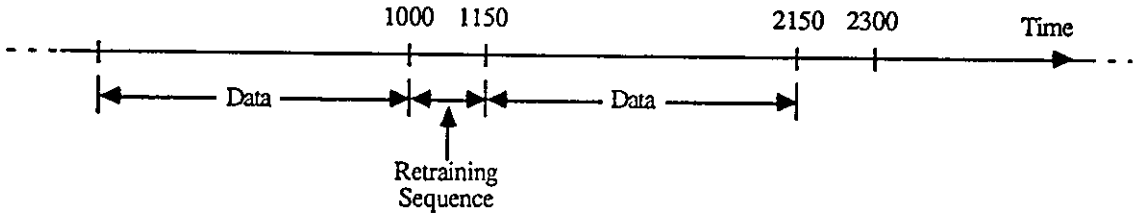


Figure 6.8 Attenuation and Group Delay Characteristics of the Combined Equipment Filters

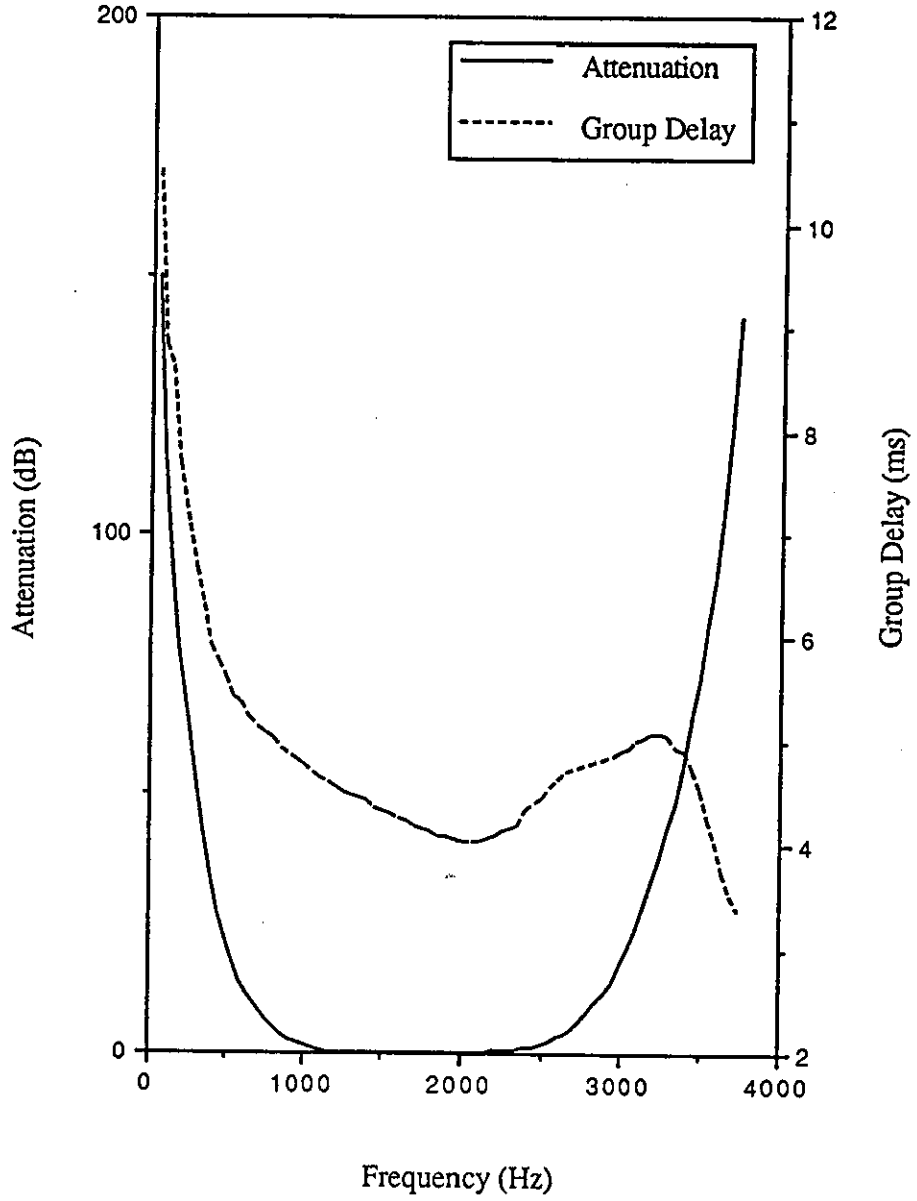


Figure 6.9 Performance of Detector A over a Two Skywave H.F. Channel at 9600 Bits/s (Multipath Propagation Delay=0.5 ms, Frequency Spread=0.5 Hz)

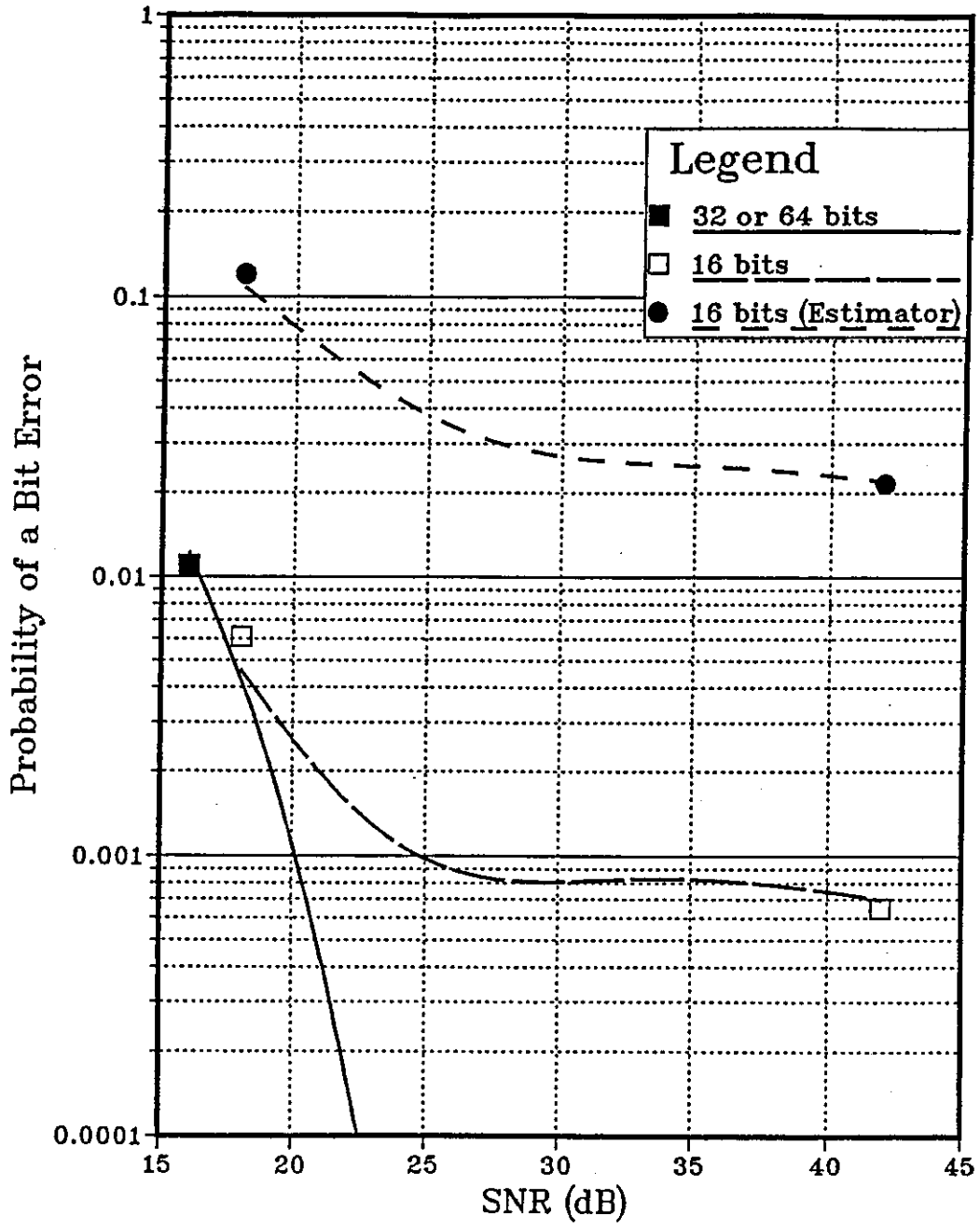


Figure 6.10 Error in the Estimate of the H.F. Channel Sampled Impulse Response (16 or 64-bit arithmetic)

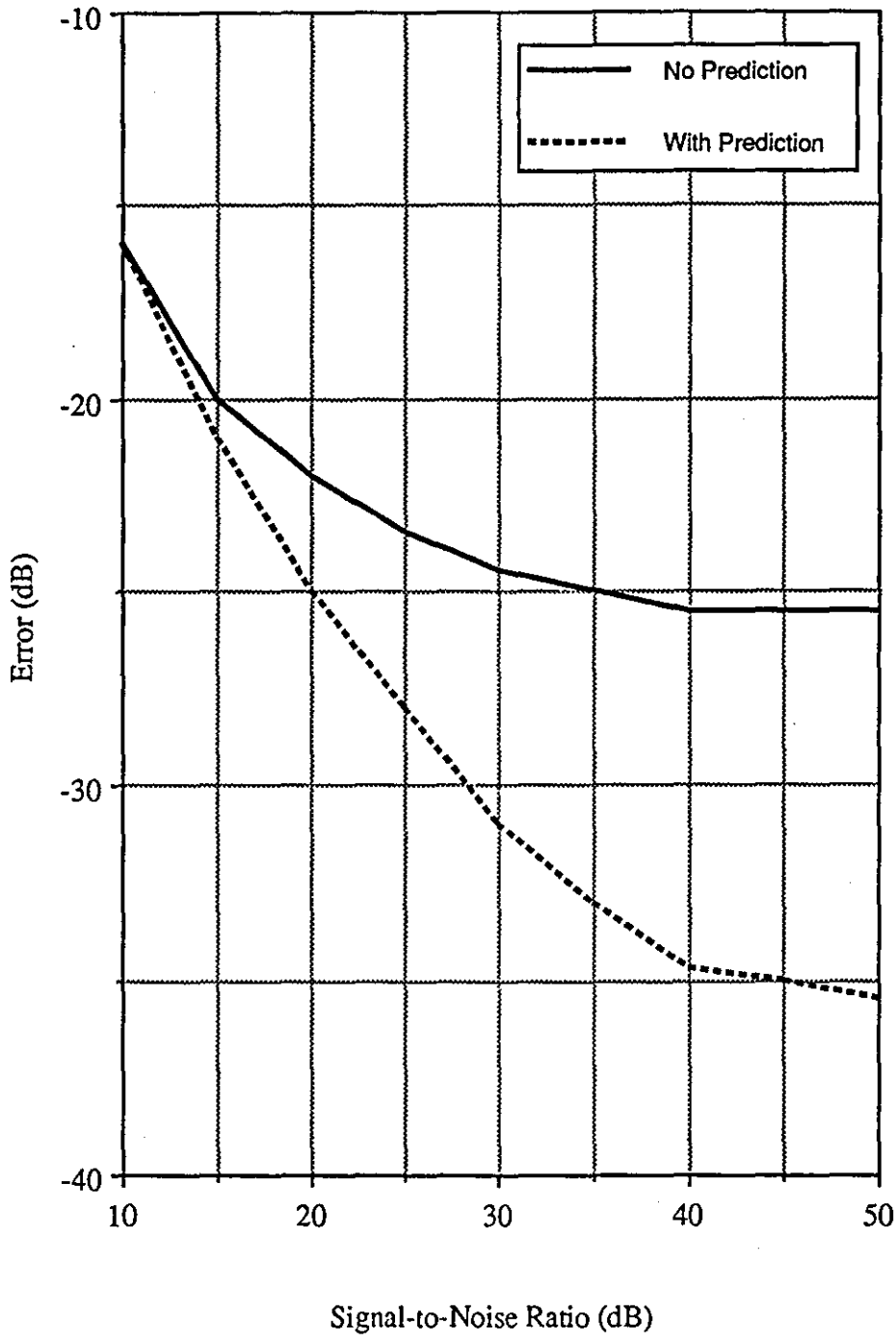


Figure 6.11 Performance of Detector A over a Two Skywave H.F. Channel at 4800 Bits/s using 16-bit Integer Arithmetic (Multipath Propagation Delay=0.5 ms, Frequency Spread=0.5 Hz)

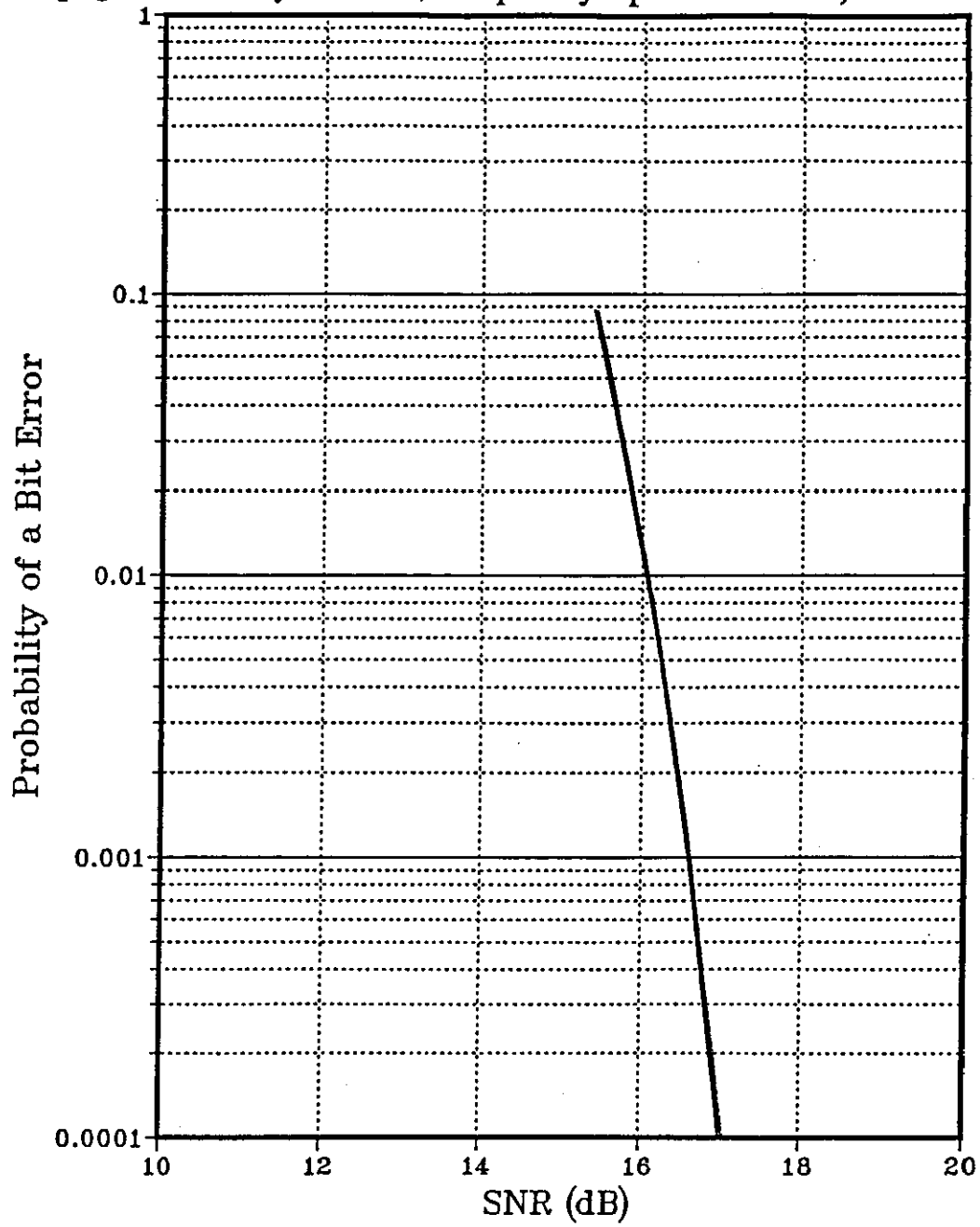


Figure 6.12 Performance of Detector A over a Two Skywave H.F. Channel at 4800 Bits/s using 16-bit Integer Arithmetic (Multipath Propagation Delay=1.1 ms, Frequency Spread=1 Hz)

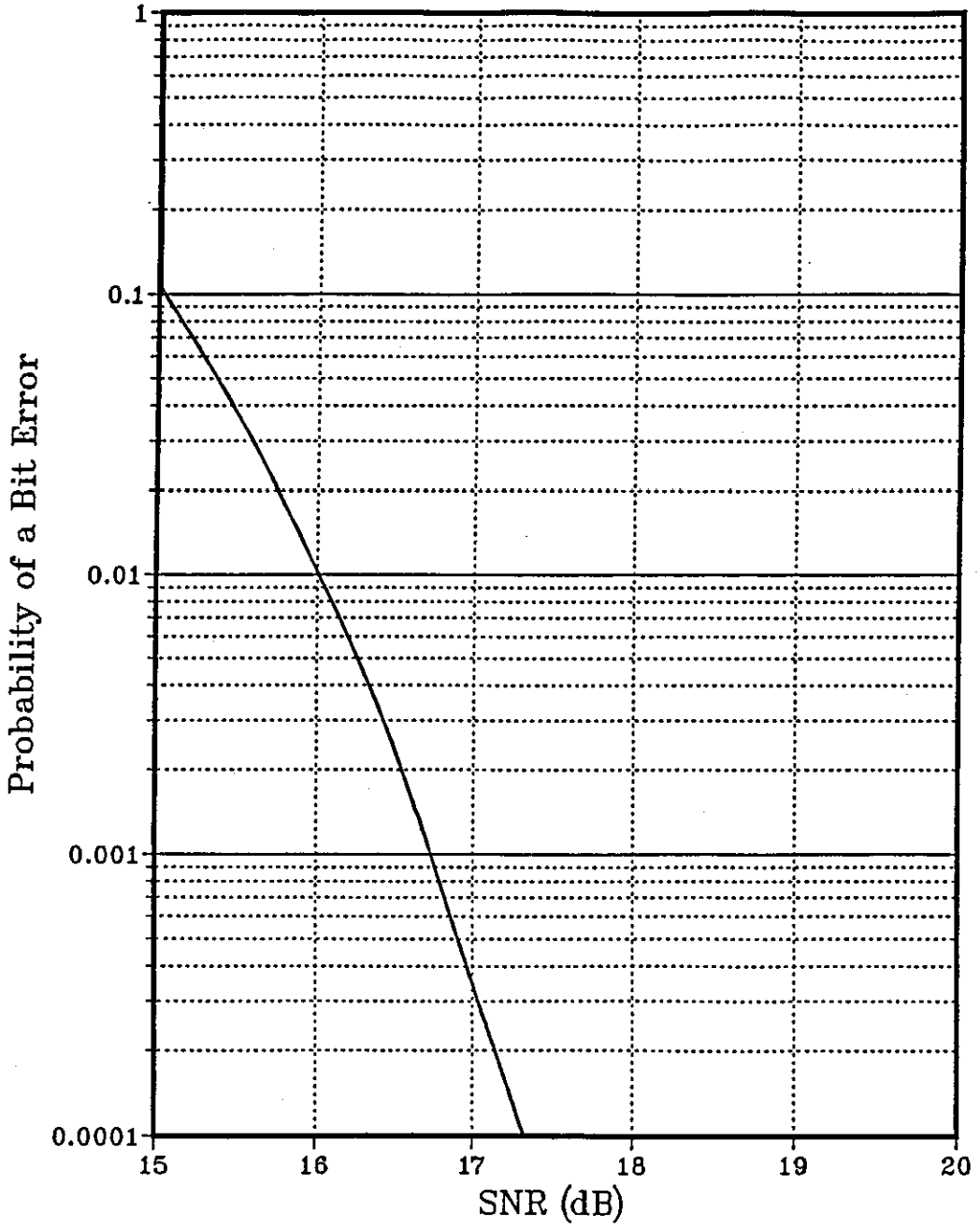


Figure 6.13 Performance of Detector A over a Two Skywave H.F. Channel at 4800 Bits/s using 16-bit Integer Arithmetic (Multipath Propagation Delay=1.1 ms, Frequency Spread=2 Hz)

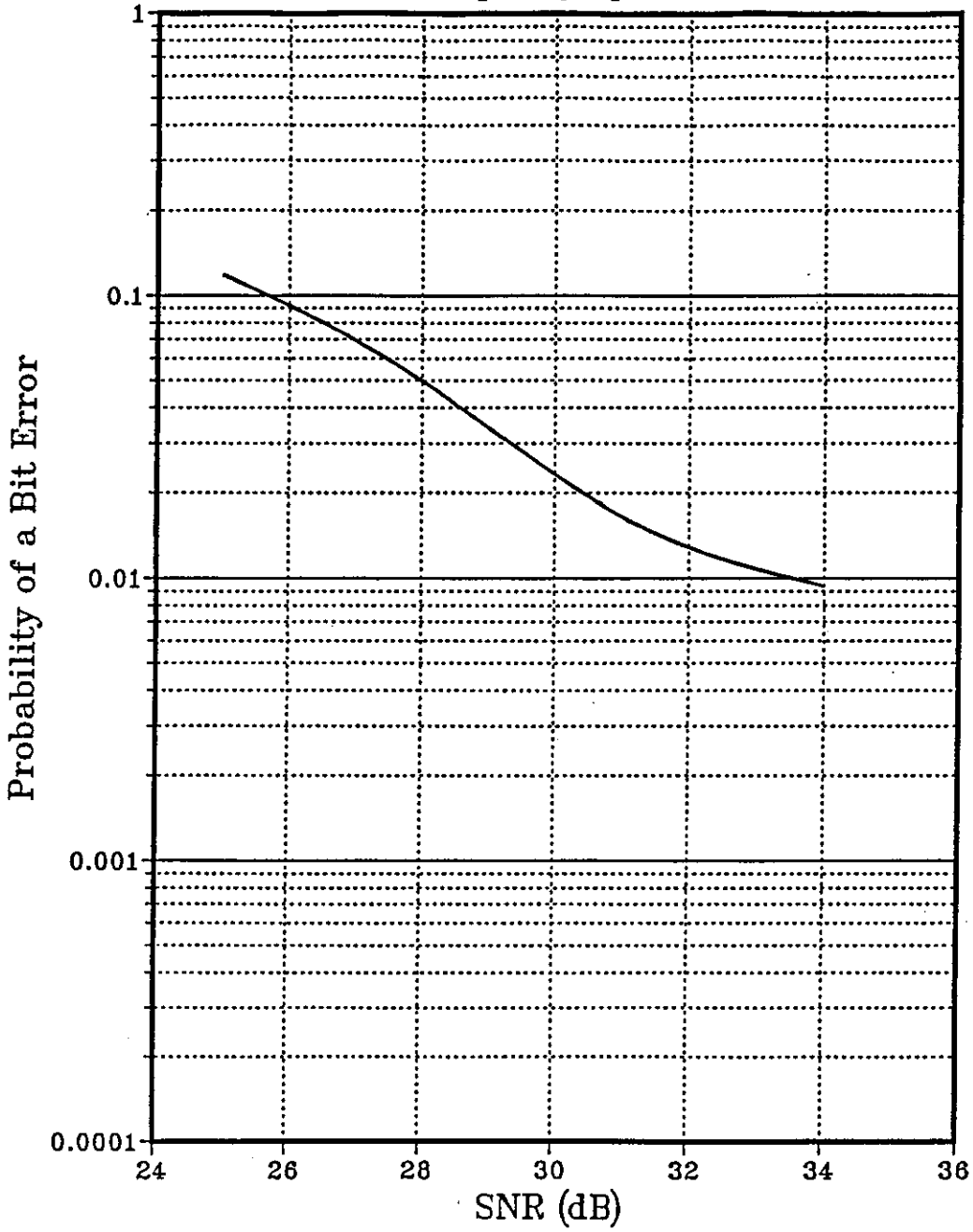
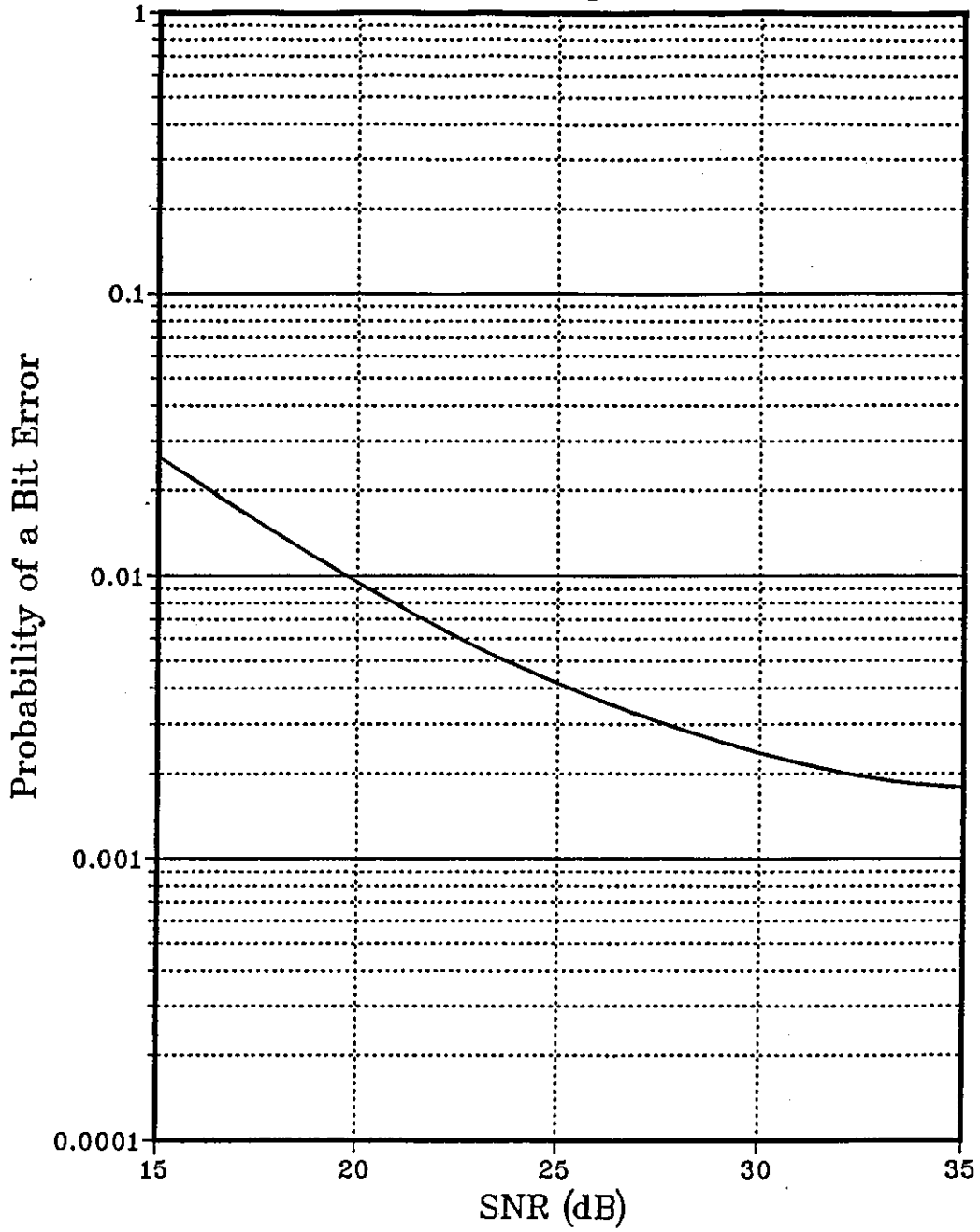


Figure 6.14 Performance of Detector A over a Two Skywave H.F. Channel at 4800 Bits/s using 16-bit Integer Arithmetic (Multipath Propagation Delay=3 ms, Frequency Spread=0.5 Hz)



CHAPTER 7
COMMENTS ON RESEARCH PROJECT

7.1 Discussion

The aim of this research project has been to show the effects of using limited precision integer arithmetic whilst running three different algorithms that, when put together, form the basis of the detector in a digital modem.

In chapter 4, the transmission of data over telephone channels was considered. Simulation results show that, at the typical values of signal-to-noise ratio found over telephone channels, the accuracy of the estimate of the sampled impulse response of the channel produced by the linear feedforward channel estimator is not seriously degraded when the numerical precision is reduced from 64-bit floating point arithmetic to 16-bit integer arithmetic. Also, the results show that increasing the number of levels in the transmitted QAM signal from 16 to 64 has little effect upon the performance of the estimator whilst it was using integer arithmetic.

The results in section 4.6 show that the accuracy of the Clark-Hau algorithm, which is basically a root finding algorithm for high order complex polynomials, is surprisingly good. It can be seen that, for channels sampled at 2400 samples/s, the differences between the minimum phase sampled impulse responses produced by the Clark-Hau algorithm (running with reduced precision arithmetic) and the ideal responses produced by the NAG library are small. At the higher sampling rate of 3200 samples/s the Clark-Hau algorithm performed satisfactorily with channels introducing mild phase distortion, but failed with channels that introduced severe amplitude distortion.

Results of the Clark-Hau algorithm actually running on the TMS320C25 show that it will run on a 16-bit digital signal processor and that it runs fast enough for the pre-filter to be adjusted in only a few sampling intervals.

In the last section of chapter 4, the results showing the bit-error rates of various

systems suggest that a complete modem receiver, operating at 2400 baud, could be constructed using only 16-bit integer arithmetic devices. The tests show that the complexity of the near maximum likelihood detector can be drastically reduced without seriously affecting the system performance. The complexity of this simplified detector is such that it should be possible to implement it in real time on the TMS320C25. In fact, for a system operating at 2400 baud over a time invariant channels, it may be possible to construct a modem detector using only one TMS320C25.

The results presented in chapter 6 show that 16-bit integer arithmetic would not give the required accuracy for the adaptive system to perform reliably at 9600 bits/s. It is shown, however, that 32-bit floating point arithmetic, as used by the TMS320C30, should provide the necessary accuracy for data transmission at this rate.

The results of the system operating at 4800 bits/s show that for mild H.F. channels and with regular retraining of the estimator to prevent collapse, it may be possible to build a modem using 16-bit integer arithmetic.

7.2 SUGGESTIONS FOR FURTHER WORK

Useful future work might include:

1. In Chapter 4, it is shown that the Clark-Hau algorithm will work satisfactorily on the TMS320C25 with telephone channels sampled at 2400 samples/s. The next stage of development would be to write programs for the near maximum likelihood detector and linear feedforward channel estimator in assembly language, making use of the features of the TMS320C25 to maximise speed. A modem receiver could then be constructed using one, or probably two, TMS320C25 signal processors, one for carrier phase synchronisation and timing recovery and the other for channels estimation, the Clark-Hau algorithm and the near maximum likelihood detector.

2. Chapter 6 shows that for mild H.F. channels, a modem receiver could operate with 16-bit integer arithmetic, at a rate of 4800 bits/s with regular retraining of the

channel estimator. A 4800 bits/s modem receiver could be constructed using several TMS320C25 processors operating in parallel and tests carried out over a hardware channel simulator or a real H.F. radio link.

3. A 9600 bits/s H.F. modem could be constructed using the TMS320C30 digital signal processor, or one of the other latest microprocessors to appear on the market that support 32-bit floating point arithmetic.

APPENDICES

APPENDIX A
RAYLEIGH FADING FILTER

The model of a single Rayleigh fading path is shown in figure 5.8, where $q_1(t)$ and $q_2(t)$ are two Gaussian random processes with zero mean and the same variances. The shape of their power spectrum is Gaussian, having the same r.m.s. frequency, f_{rms}. The power spectra of $q_1(t)$ and $q_2(t)$ is given by

$$|Q_1(f)|^2=|Q_2(f)|^2=\exp\left(\frac{-f^2}{2f_{rms}^2}\right) \qquad ..A1$$

As shown in figure 5.10, the random process $q_i(t)$ is generated by filtering a zero mean white Gaussian noise signal $n_i(t)$. Taking the square root of equation A.1 gives the Gaussian shaped frequency response of the filter, $\Gamma(f)$

$$\Gamma(f)=\exp\left(\frac{-f^2}{4f_{rms}^2}\right) \qquad ..A2$$

Taking the inverse Fourier transform gives the impulse response $\gamma(t)$,

$$\gamma(t)=\frac{1}{\sqrt{2\pi}\,t_1}\exp\left(\frac{-t^2}{2t_1^2}\right) \qquad ..A3$$

where

$$t_1=\frac{1}{2\sqrt{2}\,\pi f_{rms}} \qquad ..A4$$

The -3 dB cut-off frequency of the filter can be found from equation A2, and is given by

$$f_{co}=1.17741 f_{rms} \qquad ..A5$$

and the frequency spread f_{sp} is related to f_{rms} by

$$f_{sp}=2f_{rms} \qquad ..A6$$

therefore

$$f_{co}=0.588705 f_{sp} \qquad ..A7$$

The filter characterised by equations A2 and A3 may be approximated by a fifth order Bessel filter [70]. Both the frequency response and the impulse response of a Bessel filter tends towards a Gaussian shape as the order of the filter increases. The transfer function, in the s-plane, of an n^{th} order Bessel filter is

$$H(s)=\frac{d_0}{B_n(s)} \qquad ..A8$$

where $B_n(s)$ is given by

$$B_n(s)=\sum_{k=0}^{n} d_k s^k \qquad ..A9$$

where

$$d_k=\frac{(2n-k)!}{2^{n-k}k!} \qquad ..A10$$

and d_0 is a normalising constant of the form

$$d_0=\frac{(2n)!}{2^n} \qquad ..A11$$

When n=5 (5^{th} order filter) equation A8 becomes

$$H(s)=\frac{945}{s^5+15s^4+105s^3+420s^2+945s+945} \qquad ..A12$$

Factorising the denominator in equation A12 yields

$$H(s)=\frac{945}{\prod_{i=1}^{5}(s-p_i)} \qquad ..A13$$

where $\{p_i\}$ are the poles of H(s) and are given by [11]

$$p_1=-3.64674$$

$$p_2,p_3=-3.35196\pm j1.74266 \qquad ..A14$$

$$p_4,p_5=-2.32467\pm j3.57102$$

Substituting $s=j\Omega$, in equation A13 gives the frequency response of the filter to be

$$H(j\Omega)=\frac{945}{\prod_{i=1}^{5}(j\Omega-p_i)} \qquad ..A15$$

where Ω is the angular frequency in radians/second. The frequency Ω_c, at which the amplitude response of the fifth order Bessel filter drops by 3 dB from its peak is given by

$$\Omega_c=2.4274 \quad \text{radians/second} \qquad ..A16$$

In order to be able to change the cut-off frequency to any desired frequency, let ω be a new angular frequency variable, such that

$$\omega=c_0\Omega \qquad ..A17$$

where

$$c_0=\frac{\omega_c}{\Omega_c}=\frac{2\pi f_c}{\Omega_c} \qquad ..A18$$

and f_c is the desired cut-off frequency. From equations A16 and A18

$$c_0=2.58844f_c \qquad ..A19$$

Substituting the value of in equation A17 into A15 gives

$$H(j\omega)=\frac{945}{\prod_{i=1}^{5}\left(\frac{j\omega}{c_0}-p_i\right)} \qquad ..A20$$

letting $p_i'=c_0 p_i$, then A20 becomes

$$H(j\omega)=\frac{945c_0^5}{\prod_{i=1}^{5}(j\omega-p_i')} \qquad ..A21$$

and from equations A19 and A21,

$$H(j\omega)=\frac{109805.05f_c^5}{\prod_{i=1}^{5}(j\omega-p_i')} \qquad ..A22$$

and

$$H(j\omega)=\frac{d_0'}{\prod_{i=1}^{5}(s-p_i')} \qquad ..A23$$

where in equation A23, s has been redefined such that $s=j\omega$, and, $d_0'=109805.05f_c^5$, also

$$p_i'=2.58844f_c p_i \qquad \text{for } i=1,2,\ldots,5 \qquad ..A24$$

In the computer simulations three values of the frequency spread, f_{sp}, are used, these values are 0.5, 1.0 and 2.0 Hz. From equation A7 the corresponding cut-off frequencies are 0.2943, 0.5887 and 1.1774 Hz, respectively. The parameters of the Bessel filter for all three values of f_c are summarised in table A1.

In the computer simulations the Bessel filter is implemented in a digital form. Using

the impulse invariant transformation method [70] the analogue transfer function in equation A24 becomes, in the z-plane

$$H(z)=\frac{G_0}{\prod_{i=1}^{5}(1-A_i z^{-1})} \qquad ..A25$$

where G_0 is the gain at zero Hertz (D.C. gain) and A_i is given by

$$A_i=e^{p_i'T'} \qquad ..A26$$

The impulse response of the resulting digital filter is a sampled version of the impulse response of the analogue filter. In this technique the poles $\{p_i'\}$ in the s-plane, of equation A23, are transformed to poles at $e^{p_i'T'}$ in the z-plane. However, with a frequency spread of 2 Hz, the 3 dB bandwidth of the analogue filter is 2.35 Hz, as can be seen from equation A5. Therefore, if frequency components of, say 25 Hz, are fed into the filter, the output signal will be negligibly small. Thus according to Nyquists sampling theorem, a sampling rate of 50 Hz will be adequate enough for the accurate representation of $q_i(t)$. The values of G_0 and the $\{A_i\}$ for the three frequency spreads are given in table A2, a sampling rate of 50 Hz is assumed, i.e. T'=20 ms.

Finally, the tap gains of the digital filter may be found by expanding equation A25, thus

$$H(z)=\frac{G_0}{(1-A_1 z^{-1})(1-(A_2+A_3)z^{-1}+A_2A_3z^{-2})(1-(A_4+A_5)z^{-1}+A_4A_5z^{-2})} \qquad ..6.27$$

Equation A27 shows that the digital filter can be split up into three cascaded sections, two 2-pole sections and one single pole section, figure A1 shows a block diagram of the filter. From equation A27 the filter coefficients, $\{C_i\}$, are given by

$$C_1=-A_1$$

$$C_2=-(A_2+A_3)$$

$$C_3=A_2A_3 \qquad \text{..A28}$$

$$C_4=-(A_4+A_5)$$

$$C_5=A_4A_5$$

Table A3 gives the filter coefficients of the filter for the three values of frequency spread. The value of G_0 have been chosen such that the variance of each of the $q_i(t)$ will have a value of 0.25,this ensures that the average length of the vector representing the sampled impulse response of the channel is unity.

Table A1 Poles of the 5th Order Analogue Bessel Filter

f_{sp} (Hz)	d'_0	p'_1	p'_2, p'_3	p'_4, p'_5
0.5	0.2943	-2.7785	-2.554±j1.3277	-1.7712±j2.7208
1.0	0.5887	-5.557	-5.1078±j2.6555	-3.524±j5.44
2.0	1.1774	-11.114	-10.2156±j5.311	-7.0848±j10.883

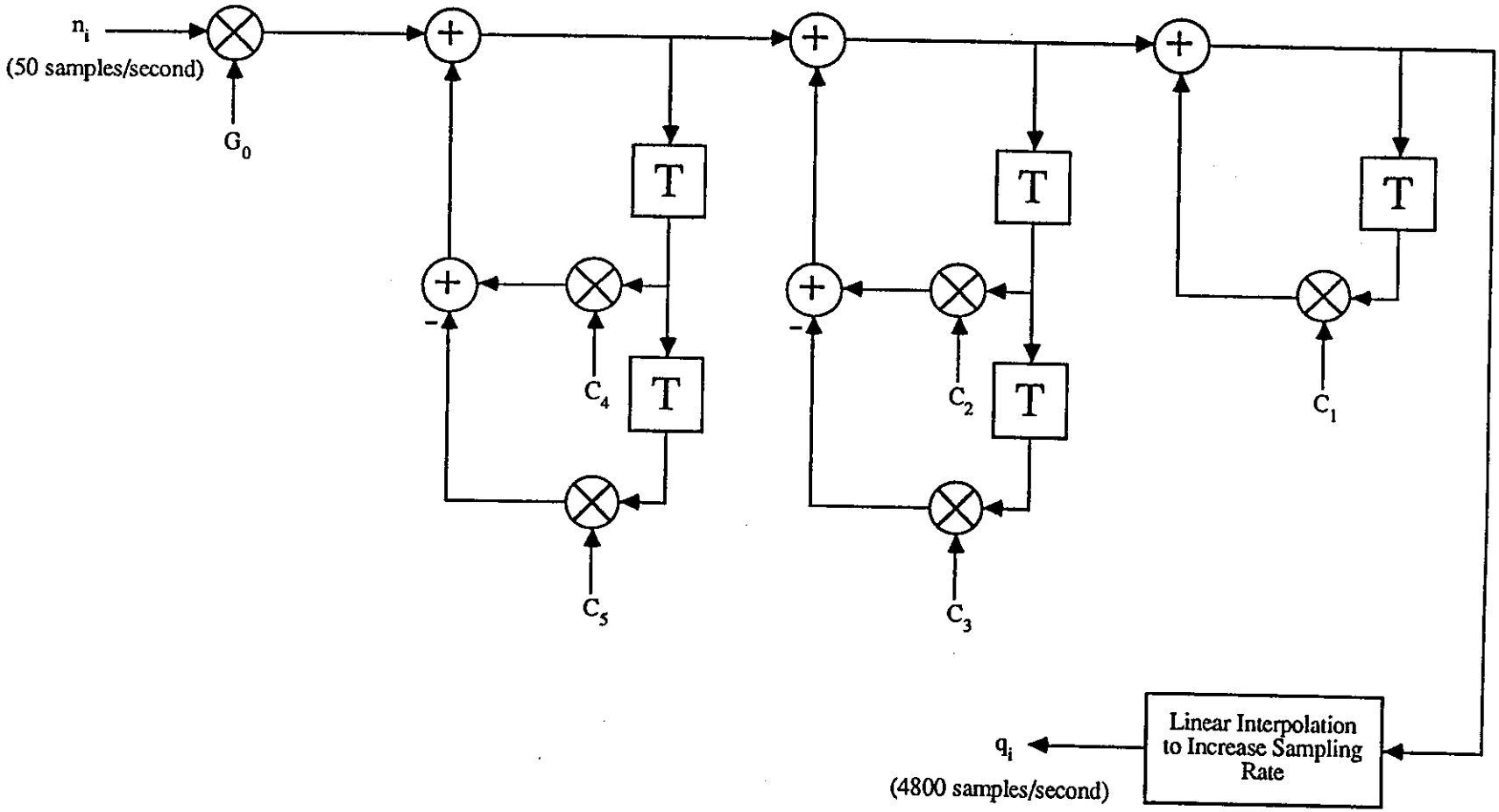
Table A2 Poles of the 5th Order Digital Bessel Filter

f_{sp} (Hz)	G_0^{-1}	A_1	A_2, A_3	A_4, A_5
0.5	325623.4	0.9460	0.9500±j0.0252	0.9638±j0.0524
1.0	16121.6	0.8950	0.9018±0.0479	0.9262±j0.1010
2.0	893.06	0.8010	0.8109±j0.0863	0.8477±0.1872

Table A3 Coeeficients of the 5th Order Digital Bessel Filter

f_{sp} (Hz)	G_0^{-1}	C_1	C_2	C_3	C_4	C_5
0.5	325623.4	0.9460	1.9000	0.9031350	1.9276	0.9316562
1.0	16121.6	0.8950	1.8036	0.8155376	1.8524	0.8680474
2.0	893.06	0.8010	1.6218	0.6650064	1.6954	0.7536391

Figure A1 Block Diagram of the 5th Order Bessel Filter



APPENDIX B
SIGNAL-TO-NOISE RATIO CALCULATIONS

The signal-to-noise ratio in the received samples $\{r_i\}$ is defined as being [2]

$$SNR=10\log_{10}\left(\frac{\varepsilon_b}{\frac{1}{2}n_0}\right) \qquad ..B1$$

where ε_b is the average energy per transmitted bit, and $n_0/2$ is the two sided power spectral density of the additive white Gaussian noise in the bandpass model described in chapter 2.

Let λ be the mean square value of the transmitted data symbols $\{s_i\}$,

$$\lambda=E\left[|s_i|^2\right]$$

$$=E\left[s_{i,0}^2\right]+E\left[s_{i,1}^2\right] \qquad ..B2$$

where E[x] is the expected value of x and $s_{i,0}$ and $s_{i,1}$ are the real and imaginary parts of the complex valued data symbol s_i, respectively. From equation B2 the average transmitted energy per bit is given by

$$\varepsilon_b=\frac{\lambda}{b} \qquad ..B3$$

where b is the number of bits used in the encoding process to produce each of the data symbols s_i.

The variance of the real and imaginary parts of the noise samples $\{w_i\}$ in the received samples $\{r_i\}$ is given by [2]

$$\sigma^2=\frac{1}{2}n_0 \qquad ..B4$$

From equations B1 to B4, the signal-to-noise ratio (measured in dB) is taken to be

$$SNR=10\log_{10}\left(\frac{\lambda}{b\sigma^2}\right) \qquad ..B5$$

For a 16-level QAM signal, the value of λ/b is 2.5 and for a 64-level QAM signal λ/b has a value of 7.

In the computer simulations the real and imaginary parts of the noise components $\{w_i\}$ are generated separately using the Numerical Algorithms Group (NAG) Gaussian distributed random number generator. The random numbers generated have zero mean and a standard deviation σ. The latter is determined by the particular value of the signal-to-noise ratio from the following

$$\sigma^2=\frac{\lambda}{b}10^{\frac{-SNR}{10}} \qquad ..B6$$

To maintain accuracy whilst obtaining the bit error rate performances of the tested systems in chapter 4, the bit error rate is determined only after a sufficient number of independent error events (N_e) occur at a particular signal-to-noise ratio. It has been shown [71], when this number is greater than 30, the 95% confidence limits in the value of the error probability can be approximated by

$$p_e\pm 2p_e\sqrt{N_e} \qquad ..B7$$

where p_e is the probability of error. In the computer simulations it was found that the value of N_c sometimes had to be set to a value greater than 30 (say 80), particularly when the tests were carried at 14.4 kbits/s over channel 4 with integer arithmetic being used at the receiver.

APPENDIX C

PROGRAM LISTINGS

```
/*Program to simulate the complete system*/
/*Transmission is over a telephone channel at a rate of 2400 baud*/
/*Simulates a system using a 64-level QAM signal with detector A*/
/*Integer arithmetic has been used*/
/*3/3/90*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#pragma linkage(g05cbf,OS);
#pragma linkage(g05ddf,OS);

void g05cbf();
double g05ddf();

main()
   {
   FILE *channel;
   FILE *minphase;
   FILE *results;

   int g,h,i,j,k,l,m,n,q1,q2,count,sdr,sdi,err1,tx[80][6];
   int pr[32][32],pi[32][32],qr[32][32],qi[32][32],tag[520][2];
   int levelr[65]={1,-7,-5,-3,-1,1,3,5,7,-7,-5,-3,-1,1,3,5,7,
      -7,-5,-3,-1,1,3,5,7,-7,-5,-3,-1,1,3,5,7,-7,-5,-3,-1,
      1,3,5,7,-7,-5,-3,-1,1,3,5,7,-7,-5,-3,-1,1,3,5,7,-7,-5,-3,
      -1,1,3,5,7};
   int leveli[65]={1,-7,-7,-7,-7,-7,-7,-7,-7,-5,-5,-5,-5,-5,-5,-5,
      -5,-3,-3,-3,-3,-3,-3,-3,-3,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,
      1,1,1,1,1,3,3,3,3,3,3,3,3,5,5,5,5,5,5,5,5,7,7,7,7,7,7,7,7};
   int code1[18]={0,0,1,1,0,1,0,1,1,0,1,0,1,1,0,0};
   int code2[18]={0,1,0,1,1,1,0,0,0,0,1,1,1,0,1,0};
   int decode1[18]={0,0,1,1,1,0,1,0,0,1,0,1,1,1,0,0};
   int decode2[18]={0,1,0,1,0,0,1,1,1,1,0,0,1,0,1,0};
   int s1[65]={-3,-1,-3,-1,-5,-7,-5,-7,-3,-1,-3,-1,-5,-7,-5,-7,
   3,3,1,1,3,3,1,1,5,5,7,7,5,5,7,7,
   -3,-3,-1,-1,-3,-3,-1,-1,-5,-5,-7,-7,-5,-5,-7,-7,
   3,1,3,1,5,7,5,7,3,1,3,1,5,7,5,7};
   int s2[65]={-3,-3,-1,-1,-3,-3,-1,-1,-5,-5,-7,-7,-5,-5,-7,-7,
   -3,-1,-3,-1,-5,-7,-5,-7,-3,-1,-3,-1,-5,-7,-5,-7,
```

```
 3,1,3,1,5,7,5,7,3,1,3,1,5,7,5,7,
 3,3,1,1,3,3,1,1,5,5,7,7,5,5,7,7};
int bits,bit1,bit2,bit3,bit4,bit5,bit6,row,col,start,limit,x;
int bita,bitb,bitc,bitd,bite,bitf;
int bite1,bite2,bite21,bite22,length,vector,vector2,bits_sent;
long max,opr,opi,p1,p2,p3,p4,cr,ci,zr,zi,ad,rdr,rdi,vr,vi;
long mincost,smallest_cost,dr,di,max1,c,lr,li,diff,e1,e2;
long yminr[30],ymini[30],filtr[50],filti[50],or[30],oi[30];
long cost[32 ],coste[515],pcost[32 ],sigr[50],sigi[50];
long fr[50],fi[50],sr[50],si[50],o2r[30],o2i[30];
long w1,w2,d1r[50],d1i[50],symbols;

unsigned long tolerance,d,den,num,div,mag,lambmax;

double yr[30],yi[30],sp[6][2];
double noiser[50],noisei[50],error1,a,b,d1,error2;
double rr,ri,nr,ni,stddev,snr,pe,snrfin,ur,ui,sc;

/*Rx filter impulse response*/
double rxr[]= {-0.4107,-0.205,0.2219,1.4222,11.5059,
 33.8749,46.7973,28.1107,-2.5666,-12.3912,-2.6308,4.3105,
 2.0033,-1.205,-0.7537,0.4731};
double rxi[]= {-0.2882,-0.6548,-2.0303,-3.4222,-1.8432,
 5.1777,11.4399,5.1152,-7.5627,-8.128,0.8993,2.9923,
 -0.873,-1.1213,0.6766,0.3189};

/*Array used for decoding of received sysmbols*/
int detbit[][][]={
{{0,0,1,1,1,1},{0,0,1,1,1,0},{0,0,1,0,1,0},{0,0,1,0,1,1},
 {0,1,0,1,1,1},{0,1,0,1,0,1},{0,1,1,1,0,1},{0,1,1,1,1,1}},
{{0,0,1,1,0,1},{0,0,1,1,0,0},{0,0,1,0,0,0},{0,0,1,0,0,1},
 {0,1,0,1,1,0},{0,1,0,1,0,0},{0,1,1,1,0,0},{0,1,1,1,1,0}},
{{0,0,0,1,0,1},{0,0,0,1,0,0},{0,0,0,0,0,0},{0,0,0,0,0,1},
 {0,1,0,0,1,0},{0,1,0,0,0,0},{0,1,1,0,0,0},{0,1,1,0,1,0}},
{{0,0,0,1,1,1},{0,0,0,1,1,0},{0,0,0,0,1,0},{0,0,0,0,1,1},
 {0,1,0,0,1,1},{0,1,0,0,0,1},{0,1,1,0,0,1},{0,1,1,0,1,1}},
{{1,0,1,0,1,1},{1,0,1,0,0,1},{1,0,0,0,0,1},{1,0,0,0,1,1},
 {1,1,0,0,1,1},{1,1,0,0,1,0},{1,1,0,1,1,0},{1,1,0,1,1,1}},
{{1,0,1,0,1,0},{1,0,1,0,0,0},{1,0,0,0,0,0},{1,0,0,0,1,0},
 {1,1,0,0,0,1},{1,1,0,0,0,0},{1,1,0,1,0,0},{1,1,0,1,0,1}},
{{1,0,1,1,1,0},{1,0,1,1,0,0},{1,0,0,1,0,0},{1,0,0,1,1,0},
```

```
{1,1,1,0,0,1},{1,1,1,0,0,0},{1,1,1,1,0,0},{1,1,1,1,0,1}},
{{1,0,1,1,1,1},{1,0,1,1,0,1},{1,0,0,1,0,1},{1,0,0,1,1,1},
 {1,1,1,0,1,1},{1,1,1,0,1,0},{1,1,1,1,1,0},{1,1,1,1,1,1}}};

  /*Five starting positions for Clark-Hau Algorithm*/
  sp[1][1]=0;
  sp[1][2]=0;
  sp[2][1]=0.5;
  sp[2][2]=0;
  sp[3][1]=0;
  sp[3][2]=0.5;
  sp[4][1]=-0.5;
  sp[4][2]=0;
  sp[5][1]=0;
  sp[5][2]=-0.5;

  g=26;  /*g is the length of the channel*/
  l=64;  /*l is the number of levels*/
  m=8 ;  /*m is the number of stored vectors*/
  n=20;  /*n is the length of the impulse response used by*/
       /*the detector*/
  length=39;  /*Length of the pre-filter*/
  symbols=500000 ; /*number of xmitted symbols*/
  a=0.003;        /*Controls the convergence of Estimator*/
  b=0.5;         /*Controls the convergence of Clark-Hau*/
  d1=0.000001 ;   /*Decision threshold*/

  snr=25  ;      /*Starting value*/
  snrfin=45 ;

  /*Scale the rx filter*/
  sc=0;
  for(i=0;i<=15;i++)
     sc=sc+rxr[i]*rxr[i]+rxi[i]*rxi[i];
  sc=sqrt(sc);

  /*Set up the random number generator*/
  srand(23 ); /*Uniform pdf*/
  g05cbf(341); /*Normal pdf*/

  puts("Enter the number of bits");
```

```
scanf("%d",&bits);

max=1;
max=max<<(bits-1);
max1=max-1;
ad=a*max;
c=b*max;
d=d1*max*max;
lambmax=max>>1;

/*Read in the channel*/
channel=fopen("channel4 data","r");
for(i=0;i<=g;i++)
    {
    fscanf(channel,"%lf,%lf",&yr[i],&yi[i]);
    }
fclose(channel);

/*Set up tags for NMLH detector*/
for(j=1;j<=m;j++)
    {
    for(h=1;h<=l;h++)
        {
        tag[h+l*j-l][1]=j;
        tag[h+l*j-l][2]=h;
        }
    }

/*Store values of Pe in file called results data*/
results=fopen("results data","w");
do
{
/*Calculate std deviation of noise*/
stddev=pow(10,(-snr/20));
stddev=stddev*2.645751311;

/*Set up arrays used to hold stored vectors and expanded vectors*/
for(j=1;j<=m;j++)
    {
    for(i=0;i<=n;i++)
        {
```

```
        qr[j][i]=1;
        qi[j][i]=1;
        pr[j][i]=0;
        pi[j][i]=0;
        }
    }

/*set up the cost array*/
for(j=1;j<=m;j++)
    {
    cost[j]=max1;
    }
cost[1]=0;  /*Let the cost of at least one array be zero*/

/*clear prefilter*/
for(i=0;i<=length;i++)
    {
    filtr[i]=filti[i]=0;
    sigr[i]=sigi[i]=0;
    }
filtr[length]=max1;     /*Set prefilter imp resp=delayed imp*/

for(j=0;j<=5;j++)
{
for(i=0;i<=(n+length);i++)
    tx[i][j]=0;
}

/*Reset estimator*/
for(i=0;i<=g;i++)
    {
    fr[i]=fi[i]=0;
    sr[i]=si[i]=0;
    }

bite1=bite2=0;  /*Bits used for differential encoder*/
bite21=bite22=0;
err1=0;         /*Set the number of bit errors to zero*/
bits_sent=0;
pe=0.5;
/*Start of Transmission*/
```

```
for(k=0;k<=symbols;k++)
    {
    /*Generate random bits*/
    bita=rand();
    bitb=rand();
    bitc=rand();
    bitd=rand();
    bite=rand();
    bitf=rand();
    if(bita<16384)
        bit1=0;
    else
        bit1=1;
    if(bitb<16384)
        bit2=0;
    else
        bit2=1;
    if(bitc<16384)
        bit3=0;
    else
        bit3=1;
    if(bitd<16384)
        bit4=0;
    else
        bit4=1;
    if(bite<16384)
        bit5=0;
    else
        bit5=1;
    if(bitf<16384)
        bit6=0;
    else
        bit6=1;

    /*Differentially encode bits*/
    x=bit1*8+bit2*4+bite1*2+bite2;
    bite1=code1[x];
    bite2=code2[x];

    tx[n+length][0]=bit1;
    tx[n+length][1]=bit2;
```

```
tx[n+length][2]=bit3;
tx[n+length][3]=bit4;
tx[n+length][4]=bit5;
tx[n+length][5]=bit6;

/*Generate symbols for transmission*/
x=bite1*32+bite2*16+bit3*8+bit4*4+bit5*2+bit6;
sr[0]=s1[x];
si[0]=s2[x];

/*Pass si thru the channel*/
rr=ri=0;
for(i=0;i<=g;i++)
    {
    rr=rr+sr[i]*yr[i]-si[i]*yi[i];
    ri=ri+sr[i]*yi[i]+si[i]*yr[i];
    }

noiser[0]=g05ddf(0,stddev);
noisei[0]=g05ddf(0,stddev);
for(i=15;i>=1;i--)
    {
    noiser[i]=noiser[i-1];
    noisei[i]=noisei[i-1];
    }
noiser[0]=g05ddf(0,stddev);
noisei[0]=g05ddf(0,stddev);
/*Pass noise through rx filter*/
nr=ni=0;
for(i=0;i<=15;i++)
    {
    nr=nr+noiser[i]*rxr[i]-noisei[i]*rxi[i];
    ni=ni+noiser[i]*rxi[i]+noisei[i]*rxr[i];
    }

/*Shift noise samples*/
for(i=15;i>=1;i--)
    {
    noiser[i]=noiser[i-1];
    noisei[i]=noisei[i-1];
    }
```

```
nr=nr/sc;
ni=ni/sc;

/*Discretise received signal*/
rr=rr+nr;
ri=ri+ni;
rr=rr/16;  /*Scale received signal before being converted*/
ri=ri/16;
rdr=rr*max;
rdi=ri*max;
if(rdr>max1)    /*Check for overflow*/
   rdr=max1;
if(rdr<-max)
   rdr=-max;
if(rdi>max1)
   rdi=max1;
if(rdi<-max)
   rdi=-max;

/*Feed discrete signal into pre-filter*/
sigr[0]=rdr;
sigi[0]=rdi;

/*Allow estimator to converge at start up*/
/*i.e. during the first 2000 symbols*/
if(k<2000)
   {
   vr=vi=0;
   for(i=0;i<=g;i++)
      {
      p1=sr[i]*fr[i];
      p2=si[i]*fi[i];
      p3=sr[i]*fi[i];
      p4=si[i]*fr[i];
      vr=vr+p1-p2;
      vi=vi+p3+p4;
      /*Could perform the above additions in the ACCU*/
      }
      vr=vr>>4;
      vi=vi>>4;
      /*Check for overflow*/
```

```
if(vr>max1)
    vr=max1;
if(vr<-max)
    vr=-max;
if(vi>max1)
    vi=max1;
if(vi<-max)
    vi=-max;
rdr=rdr*ad;
rdi=rdi*ad;
vr=vr*ad;
vi=vi*ad;
dr=rdr-vr;
di=rdi-vi;
dr=dr>>(bits-1);
di=di>>(bits-1);
/*Check for overflow*/
if(dr>max1)
    dr=max1;
if(dr<-max)
    dr=-max;
if(di>max1)
    di=max1;
if(di<-max)
    di=-max;
/*Update estimator*/
for(i=0;i<=g;i++)
    {
    p1=dr*sr[i];
    p2=di*si[i];
    p3=dr*si[i];
    p4=di*sr[i];
    fr[i]=fr[i]+p1+p2;
    fi[i]=fi[i]+p4-p3;
    /*Check for overflow*/
    if(fr[i]>max1)
        fr[i]=max1;
    if(fr[i]<-max)
        fr[i]=-max;
    if(fi[i]>max1)
        fi[i]=max1;
```

```
            if(fi[i]<-max)
                fi[i]=-max;
            }
    /*End of estimator routine*/

    }
/*Allow the estimator to settle down before switching*/
/*in the detector and Clark-Hau algorithm*/
if(k<2000)
    goto shift;

/*Run the Clark-Hau algorithm once to find prefilter*/
/*and minphase impulse responses*/
if(k==2000  )
    {
    start=0;
    do
        {
        start++;
        lr=max*sp[start][1]; /*lr+jli is the estimate of*/
        li=max*sp[start][2]; /*the reciprocal of a root pos*/
        tolerance=max1;
        count=0;
        /*Start of root finding process*/
        while(tolerance>=d)
            {
            count++;
            /*Pass Y thru F/B filter*/
            or[g]=fr[g];
            oi[g]=fi[g];
            o2r[g]=fr[g];
            o2i[g]=fi[g];
            for(i=(g-1);i>=0;i--)
                {
                p1=lr*or[i+1];
                p2=li*oi[i+1];
                p3=lr*oi[i+1];
                p4=li*or[i+1];
                p1=p1>>(bits-1);/*Scale after multiply*/
                p2=p2>>(bits-1);
                p3=p3>>(bits-1);
```

```
p4=p4>>(bits-1);
or[i]=fr[i]-p1;
oi[i]=fi[i]-p3;
if(or[i]>max1) /*Check for overflow*/
    or[i]=max1;
if(or[i]<-max)
    or[i]=-max;
if(oi[i]>max1)
    oi[i]=max1;
if(oi[i]<-max)
    oi[i]=-max;
or[i]=or[i]+p2;
oi[i]=oi[i]-p4;
if(or[i]>max1) /*Check for overflow*/
    or[i]=max1;
if(or[i]<-max)
    or[i]=-max;
if(oi[i]>max1)
    oi[i]=max1;
if(oi[i]<-max)
    oi[i]=-max;
p1=lr*o2r[i+1]; /*Pass o/p of first F.B.*/
p2=li*o2i[i+1]; /*into second F.B. filter*/
p3=lr*o2i[i+1]; /*to give epsilon*/
p4=li*o2r[i+1];
p1=p1>>(bits-1);
p2=p2>>(bits-1);
p3=p3>>(bits-1);
p4=p4>>(bits-1);
o2r[i]=or[i]-p1;
o2i[i]=oi[i]-p3;
if(o2r[i]>max1)
    o2r[i]=max1;
if(o2r[i]<-max)
    o2r[i]=-max;
if(o2i[i]>max1)
    o2i[i]=max1;
if(o2i[i]<-max)
    o2i[i]=-max;
o2r[i]=o2r[i]+p2;
o2i[i]=o2i[i]-p4;
```

```
    if(o2r[i]>max1)
       o2r[i]=max1;
    if(o2r[i]<-max)
       o2r[i]=-max;
    if(o2i[i]>max1)
       o2i[i]=max1;
    if(o2i[i]<-max)
       o2i[i]=-max;
    }
e1=o2r[1];
e2=o2i[1];
den=e1*e1+e2*e2;  /*Calc. denominator of div.*/
den=den>>(bits-1);
if(den>max1)
   den=max1;
den=den<<(bits-1);

/*Carry out real division to give real part*/
/*of complex division*/
p1=or[0]*e1+oi[0]*e2;
p1=p1>>(bits-1);
if(p1>max1) /*Check for overflow*/
   p1=max1;
if(p1<-max)
   p1=-max;
/*Find absolute value of numerator and*/
/*check for div by zero and if num>den*/
num=abs(p1);
if(num>=den)
   {
   if(p1<0)
      w1=-max;
   else if(p1>0)
      w1=max1;
   else
      w1=0;
   }
else
   {
   num=num<<bits;
```

```
    /*Carry out division*/
    for(j=1;j<=(bits-1);j++)
        {
        diff=(long)(num-den);
        if(diff<0)
            num=num<<1;
        else
            {
            diff=diff<<1;
            diff++;
            num=(unsigned long)(diff);
            }
        }
    w1=(long)(num&max1);
    if(p1<0) /*Change to negative*/
        w1=-w1;
    }

/*Calc imaginary part of complex division*/
p2=oi[0]*e1-or[0]*e2;
p2=p2>>(bits-1);
if(p2>max1)
    p2=max1;
if(p2<-max)
    p2=-max;
num=abs(p2);
/*Check for div by zero*/
if(num>=den)
    {
    if(p2<0)
        w2=-max;
    else if(p2>0)
        w2=max1;
    else
        w2=0;
    }
else

    /*Perform division*/
    {
    num=num<<bits;
```

```
    for(j=1;j<=(bits-1);j++)
        {
        diff=(long)(num-den);
        if(diff<0)
            num=num<<1;
        else
            {
            diff=diff<<1;
            diff++;
            num=(unsigned long)diff;
            }
        }
    w2=(long)(num&max1);
    if(p2<0)
        w2=-w2;
    }
/*Check for convergence*/
tolerance=w1*w1+w2*w2;
if(tolerance>max1)
    tolerance=max1;
if(tolerance<=d)
    break;
w1=w1*c;
w2=w2*c;
w1=w1>>(bits-1);
w2=w2>>(bits-1);

/*Update reciprocal root position*/
lr=lr+w1;
li=li+w2;
if(lr>max1)
    lr=max1;
if(lr<-max)
    lr=-max;
if(li>max1)
    li=max1;
if(li<-max)
    li=-max;
p1=lr*lr;
p2=li*li;
```

```
    /*Check for magnitude of updated position*/
    /*to see if it is greater than unity*/
    mag=(unsigned long)(p1+p2);
    mag=mag>>bits;
    if(mag>max1)
       mag=max1;
    /*Check to see if outside unit circle*/
    if(mag>lambmax)
       break;
    /*Check number of iterations*/
    if(count==40)
       break;
    }
if(mag>lambmax)
    continue;
if(count==40)
    continue;
/*Pass output of F/B filter thru F/F filter*/
or[g+1]=oi[g+1]=0;
for(i=1;i<=(g+1);i++)
    {
    p1=lr*or[i-1];
    p2=li*oi[i-1];
    p3=lr*oi[i-1];
    p4=li*or[i-1];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
    fr[i-1]=or[i]+p1+p2;
    fi[i-1]=oi[i]+p3-p4;
    }
/*Calculate prefilter taps*/
d1r[length]=filtr[length];
d1i[length]=filti[length];
for(i=(length-1);i>=0;i--)
    {
    p1=lr*d1r[i+1];
    p2=li*d1i[i+1];
    p3=lr*d1i[i+1];
    p4=li*d1r[i+1];
```

```
            p1=p1>>(bits-1);
            p2=p2>>(bits-1);
            p3=p3>>(bits-1);
            p4=p4>>(bits-1);
           d1r[i]=filtr[i]-p1+p2;
           d1i[i]=filti[i]-p3-p4;
            }
        d1r[length+1]=d1i[length+1]=0;
         for(i=1;i<=(length+1);i++)
            {
            p1=lr*d1r[i-1];
            p2=li*d1i[i-1];
            p3=lr*d1i[i-1];
            p4=li*d1r[i-1];
            p1=p1>>(bits-1);
            p2=p2>>(bits-1);
            p3=p3>>(bits-1);
            p4=p4>>(bits-1);
            filtr[i-1]=d1r[i]+p1+p2;
            filti[i-1]=d1i[i]+p3-p4;
            }
        start=0;
        }
    while(start<5);

}
/*Pass rx signal thru pre-filter*/
opr=opi=0;
for(i=0;i<=length;i++)
    {
    p1=sigr[i]*filtr[i];
    p2=sigi[i]*filti[i];
    p3=sigr[i]*filti[i];
    p4=sigi[i]*filtr[i];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
    opr=opr+p1-p2;
    opi=opi+p3+p4;
    if(opr>(max-1))
```

```
        opr=max-1;
    if(opr<-max)
        opr=-max;
    if(opi>(max-1))
        opi=max-1;
    if(opi<-max)
        opi=-max;
    }
/*Shift signal*/
for(i=length;i>=1;i--)
    {
    sigr[i]=sigr[i-1];
    sigi[i]=sigi[i-1];
    }

/*Calculate costs of vectors if they were expanded*/
mincost=max1;
for(j=1;j<=m;j++)
    {
    /*Evaluate the cost for m vectors*/
    cr=ci=0;
    for(i=0;i<=(n-1);i++)
        {
        p1=qr[j][i]*fr[n-i];
        p2=qi[j][i]*fi[n-i];
        p3=qr[j][i]*fi[n-i];
        p4=qi[j][i]*fr[n-i];
        cr=cr+((p1-p2)>>4);
        ci=ci+((p3+p4)>>4);
        if(cr>(max-1))
            cr=max-1;
        if(cr<-max)
            cr=-max;
        if(ci>(max-1))
            ci=max-1;
        if(ci<-max)
            ci=-max;
        }
        /*Evaluate the total cost for ml vectors*/
        for(h=1;h<=l;h++)
            {
```

```
                vector2=h+1*j-1;
                p1=levelr[h]*fr[0];
                p2=leveli[h]*fi[0];
                p3=levelr[h]*fi[0];
                p4=leveli[h]*fr[0];
                zr=cr+((p1-p2)>>4); /*Scale cost*/
                zi=ci+((p3+p4)>>4);
                if(zr>(max-1))
                    zr=max-1;
                if(zr<-max)
                    zr=-max;
                if(zi>(max-1))
                    zi=max-1;
                if(zi<-max)
                    zi=-max;
                zr=(zr-opr)>>1;
                zi=(zi-opi)>>1;
                /*Calc. costs of expanded vectors*/
                p1=zr*zr;
                p2=zi*zi;
                p1=p1>>(bits-2); /*Scaling*/
                p2=p2>>(bits-2);
                coste[vector2]=cost[j]+p1+p2;
                if(coste[vector2]>max1)
                   coste[vector2]=max1;
                /*Find the smallest cost*/
                if(coste[vector2]<mincost)
                    {
                    mincost=coste[vector2];
                    q1=j;
                    q2=h;
                    vector=vector2;
                    }
                }
        }

smallest_cost=mincost;

/*Output detected value*/
sdr=qr[q1][0];
sdi=qi[q1][0];
```

```
        col=(sdr+7)/2;
        row=(sdi+7)/2;

        /*Obtain corresponding bits from symbols*/
        bit1=detbit[row][col][0];
        bit2=detbit[row][col][1];
        bit3=detbit[row][col][2];
        bit4=detbit[row][col][3];
        bit5=detbit[row][col][4];
        bit6=detbit[row][col][5];

        /*Differentially decode bits*/
        x=bite21*8+bite22*4+bit1*2+bit2;
        bite21=bit1;
        bite22=bit2;
        bit1=decode1[x];
        bit2=decode2[x];

        /*Count bit errors*/
        if(k>2500)
        {
        bits_sent=bits_sent+6;
        if(bit1!=tx[0][0])
            err1++;
        if(bit2!=tx[0][1])
            err1++;
        if(bit3!=tx[0][2])
            err1++;
        if(bit4!=tx[0][3])
            err1++;
        if(bit5!=tx[0][4])
            err1++;
        if(bit6!=tx[0][5])
            err1++;
        }
if((err1>10000)&&(pe>=.01)) break;
else if((err1>400)&&(pe<0.01)) break;

        /*Store vector with smallest cost in array p*/
        for(i=0;i<=(n-1);i++)
            {
```

```
    pr[1][i]=qr[q1][i];
    pi[1][i]=qi[q1][i];
    }
pr[1][n]=levelr[q2];
pi[1][n]=leveli[q2];
pcost[1]=smallest_cost;
coste[vector]=max1;

/*Discard all costs where first component<>to detected*/
/*value*/
for(j=1;j<=m;j++)
   {
   if((qr[j][0]!=sdr)||(qi[j][0]!=sdi))
      {
      for(i=1;i<=l;i++)
         coste[j*l-l+i]=max1;
      }
   }

/*Select the next m-1 vectors and store in p*/
count=2;
do
   {
   mincost=coste[1];
   vector=1 ;
   for(j=2;j<=(m*l);j++)
      {
      if(coste[j]<mincost)
         {
         mincost=coste[j];
         vector=j;
         }
      }
   pcost[count]=mincost;
   coste[vector]=max1;

   /*Transfer vector to array p*/
   q1=tag[vector][1];
   q2=tag[vector][2];
   for(i=0;i<=(n-1);i++)
      {
```

```
                pr[count][i]=qr[q1][i];
                pi[count][i]=qi[q1][i];
                }
            pr[count][n]=levelr[q2];
            pi[count][n]=leveli[q2];
            count++;
            }
        while(count<=m);

        /*Get new stored vectors*/
        for(j=1;j<=m;j++)
            {
            for(i=1;i<=n;i++)  /*Discard first component in q*/
                {
                qr[j][i-1]=pr[j][i];
                qi[j][i-1]=pi[j][i];
                }
            cost[j]=pcost[j]-smallest_cost;     /*Normalize*/
            }

        /*Shift tx symbols*/
shift:   for(i=g;i>=1;i--)
            {
            sr[i]=sr[i-1];
            si[i]=si[i-1];
            }

        /*Shift tx data*/
        for(i=1;i<=(n+length);i++)
            {
            tx[i-1][0]=tx[i][0];
            tx[i-1][1]=tx[i][1];
            tx[i-1][2]=tx[i][2];
            tx[i-1][3]=tx[i][3];
            tx[i-1][4]=tx[i][4];
            tx[i-1][5]=tx[i][5];
            }
        }
    error1=err1;
    pe=error1/bits_sent;
    printf("snr=%lf  pe=%lf\n",snr,pe);
```

```
    fprintf(results,"%lf,%lf\n",snr,pe);
    snr=snr+1.0;
if(pe<0.0001) break;
    }
    while(snr<=snrfin);
fclose(results);
}
```

```
/*Simulation of complete system using detector B, 64-level QAM signals*/
/*Data is transmitted over telephone channels*/
/*Uses pseudo binary detection*/
/*Integer arithmetic used*/
/*21/3/90*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#pragma linkage(g05cbf,OS);
#pragma linkage(g05ddf,OS);

void g05cbf();
double g05ddf();

main()
    {
    FILE *channel;
    FILE *minphase;
    FILE *results;

    int g,h,i,j,k,l,m,n,q1,q2,count,sdr,sdi,err1,tx[80][6];
    int pr[32][32],pi[32][32],qr[32][32],qi[32][32],tag[520][3];
    int code1[18]={0,0,1,1,0,1,0,1,1,0,1,0,1,1,0,0};
    int code2[18]={0,1,0,1,1,1,0,0,0,0,1,1,1,0,1,0};
    int decode1[18]={0,0,1,1,1,0,1,0,0,1,0,1,1,1,0,0};
    int decode2[18]={0,1,0,1,0,0,1,1,1,1,0,0,1,0,1,0};
    int s1[65]={-3,-1,-3,-1,-5,-7,-5,-7,-3,-1,-3,-1,-5,-7,-5,-7,
     3,3,1,1,3,3,1,1,5,5,7,7,5,5,7,7,
     -3,-3,-1,-1,-3,-3,-1,-1,-5,-5,-7,-7,-5,-5,-7,-7,
     3,1,3,1,5,7,5,7,3,1,3,1,5,7,5,7};
    int s2[65]={-3,-3,-1,-1,-3,-3,-1,-1,-5,-5,-7,-7,-5,-5,-7,-7,
     -3,-1,-3,-1,-5,-7,-5,-7,-3,-1,-3,-1,-5,-7,-5,-7,
     3,1,3,1,5,7,5,7,3,1,3,1,5,7,5,7,
     3,3,1,1,3,3,1,1,5,5,7,7,5,5,7,7};
    int bits,bit1,bit2,bit3,bit4,bit5,bit6,row,col,start,limit,x;
    int bita,bitb,bitc,bitd,bite,bitf;
    int bite1,bite2,bite21,bite22,length,vector,vector2,bits_sent;
    long max,opr,opi,p1,p2,p3,p4,cr,ci,zr,zi,ad,rdr,rdi,vr,vi;
    long mincost,smallest_cost,dr,di,max1,c,lr,li,diff,e1,e2;
```

```
    long yminr[30],ymini[30],filtr[50],filti[50],or[30],oi[30];
    long cost[32 ],coste[260],pcost[40 ],sigr[50],sigi[50];
    long fr[50],fi[50],sr[50],si[50],o2r[30],o2i[30];
    long w1,w2,d1r[50],d1i[50],symbols,mod;
int roots;
    unsigned long tolerance,d,den,num,div,mag,lambmax;

    double yr[30],yi[30],sp[6][2];
    double error1,a,b,d1,error2;
    double rr,ri,nr,ni,stddev,snr,pe,snrfin,ur,ui,sc;

    /*Bits used to decode received symbols*/
    int detbit[][][]={
{{0,0,1,1,1,1},{0,0,1,1,1,0},{0,0,1,0,1,0},{0,0,1,0,1,1},
 {0,1,0,1,1,1},{0,1,0,1,0,1},{0,1,1,1,0,1},{0,1,1,1,1,1}},
{{0,0,1,1,0,1},{0,0,1,1,0,0},{0,0,1,0,0,0},{0,0,1,0,0,1},
 {0,1,0,1,1,0},{0,1,0,1,0,0},{0,1,1,1,0,0},{0,1,1,1,1,0}},
{{0,0,0,1,0,1},{0,0,0,1,0,0},{0,0,0,0,0,0},{0,0,0,0,0,1},
 {0,1,0,0,1,0},{0,1,0,0,0,0},{0,1,1,0,0,0},{0,1,1,0,1,0}},
{{0,0,0,1,1,1},{0,0,0,1,1,0},{0,0,0,0,1,0},{0,0,0,0,1,1},
 {0,1,0,0,1,1},{0,1,0,0,0,1},{0,1,1,0,0,1},{0,1,1,0,1,1}},
{{1,0,1,0,1,1},{1,0,1,0,0,1},{1,0,0,0,0,1},{1,0,0,0,1,1},
 {1,1,0,0,1,1},{1,1,0,0,1,0},{1,1,0,1,1,0},{1,1,0,1,1,1}},
{{1,0,1,0,1,0},{1,0,1,0,0,0},{1,0,0,0,0,0},{1,0,0,0,1,0},
 {1,1,0,0,0,1},{1,1,0,0,0,0},{1,1,0,1,0,0},{1,1,0,1,0,1}},
{{1,0,1,1,1,0},{1,0,1,1,0,0},{1,0,0,1,0,0},{1,0,0,1,1,0},
 {1,1,1,0,0,1},{1,1,1,0,0,0},{1,1,1,1,0,0},{1,1,1,1,0,1}},
{{1,0,1,1,1,1},{1,0,1,1,0,1},{1,0,0,1,0,1},{1,0,0,1,1,1},
 {1,1,1,0,1,1},{1,1,1,0,1,0},{1,1,1,1,1,0},{1,1,1,1,1,1}}};

    /*Starting positions for Clark-Hau algorithm*/
    sp[1][1]=0;
    sp[1][2]=0;
    sp[2][1]=0.5;
    sp[2][2]=0;
    sp[3][1]=0;
    sp[3][2]=0.5;
    sp[4][1]=-0.5;
    sp[4][2]=0;
    sp[5][1]=0;
    sp[5][2]=-0.5;
```

```
g=26;  /*g is the length of the channel*/
l=64;  /*l is the number of levels*/
m=8 ;  /*m is the number of stored vectors*/
n=20;  /*n is the length of the impulse response used by*/
     /*the detector*/
length=39;  /*Length of the pre-filter*/
symbols=500000 ;  /*number of xmitted symbols*/
a=0.003;          /*Controls the convergence of Estimator*/
b=0.5;           /*Controls the convergence of Clark-Hau*/
dl=0.000001 ;     /*Decision threshold*/

snr=25  ;        /*Starting value*/
snrfin=45 ;

/*Set up the random number generator*/
srand(12 );  /*Uniform pdf*/
g05cbf(111); /*Normal pdf*/

puts("Enter the number of bits");
scanf("%d",&bits);
max=1;
max=max<<(bits-1);
max1=max-1;
al=a*max;
c=b*max;
d=dl*max*max;
hmbmax=max>>1;

/*Read in the channel*/
channel=fopen("channel4 data","r");
for(i=0;i<=g;i++)
  {
  fscanf(channel,"%lf,%lf",&yr[i],&yi[i]);
  }
fclose(channel);

/*Store values of Pe in file called results data*/
results=fopen("results data","w");
do
{
```

```
/*Calculate std deviation for noise generation*/
stddev=pow(10,(-snr/20));
stddev=stddev*2.645751311;

/*Set up dectector vectors*/
for(j=1;j<=m;j++)
    {
    for(i=0;i<=n;i++)
        {
        qr[j][i]=1;
        qi[j][i]=1;
        pr[j][i]=0;
        pi[j][i]=0;
        }
    }

/*set up the cost array*/
for(j=1;j<=m;j++)
    {
    cost[j]=max1;
    }
cost[7]=0;

/*clear prefilter*/
for(i=0;i<=length;i++)
    {
    filtr[i]=filti[i]=0;
    sigr[i]=sigi[i]=0;
    }
filtr[length]=max1;     /*Set prefilter imp resp=delayed imp*/

for(j=0;j<=5;j++)
{
for(i=0;i<=(n+length);i++)
    tx[i][j]=0;
}

/*Reset estimator*/
for(i=0;i<=g;i++)
    {
    fr[i]=fi[i]=0;
```

```
    sr[i]=si[i]=0;
    }

bite1=bite2=0;  /*Bits used for differential encoding*/
bite21=bite22=0;
err1=0;
bits_sent=0;
/*Start of Transmission*/
for(k=0;k<=symbols;k++)
    {
    /*Generate random bits*/
    bita=rand();
    bitb=rand();
    bitc=rand();
    bitd=rand();
    bite=rand();
    bitf=rand();
    /*Encode the bits using grey coding*/
    if(bita<16384)
        bit1=0;
    else
        bit1=1;
    if(bitb<16384)
        bit2=0;
    else
        bit2=1;
    if(bitc<16384)
        bit3=0;
    else
        bit3=1;
    if(bitd<16384)
        bit4=0;
    else
        bit4=1;
    if(bite<16384)
        bit5=0;
    else
        bit5=1;
    if(bitf<16384)
        bit6=0;
    else
```

```
    bit6=1;

/*Differentially encode bits*/
x=bit1*8+bit2*4+bite1*2+bite2;
bite1=code1[x];
bite2=code2[x];

tx[n+length][0]=bit1;
tx[n+length][1]=bit2;
tx[n+length][2]=bit3;
tx[n+length][3]=bit4;
tx[n+length][4]=bit5;
tx[n+length][5]=bit6;

/*Generate symbols to be xmitted*/
x=bite1*32+bite2*16+bit3*8+bit4*4+bit5*2+bit6;
sr[0]=s1[x];
si[0]=s2[x];

/*Pass si thru the channel*/
rr=ri=0;
for(i=0;i<=g;i++)
    {
    rr=rr+sr[i]*yr[i]-si[i]*yi[i];
    ri=ri+sr[i]*yi[i]+si[i]*yr[i];
    }

     nr=g05ddf(0,stddev);
     ni=g05ddf(0,stddev);

/*Now discretize everything*/
rr=rr+nr;
ri=ri+ni;
rr=rr/16;  /*Scale signal before being fed into A-D converter*/
ri=ri/16;
rdr=rr*max;
rdi=ri*max;
if(rdr>max1)   /*Check for overflow*/
   rdr=max1;
if(rdr<-max)
   rdr=-max;
```

```
if(rdi>max1)
    rdi=max1;
if(rdi<-max)
    rdi=-max;
sigr[0]=rdr;
sigi[0]=rdi;

/*Allow estimator to converge at start up*/
/*i.e. during the first 1000 symbols*/
if(k<1000)
    {
    vr=vi=0;
    for(i=0;i<=g;i++)
        {
        p1=sr[i]*fr[i];
        p2=si[i]*fi[i];
        p3=sr[i]*fi[i];
        p4=si[i]*fr[i];
        vr=vr+p1-p2;
        vi=vi+p3+p4;
        /*Could perform the above additions in the ACCU*/
        }
        vr=vr>>4;
        vi=vi>>4;
        /*Check for overflow*/
        if(vr>max1)
            vr=max1;
        if(vr<-max)
            vr=-max;
        if(vi>max1)
            vi=max1;
        if(vi<-max)
            vi=-max;
        rdr=rdr*ad;
        rdi=rdi*ad;
        vr=vr*ad;
        vi=vi*ad;
        dr=rdr-vr;
        di=rdi-vi;
        dr=dr>>(bits-1);
        di=di>>(bits-1);
```

```
        /*Check for overflow*/
        if(dr>max1)
           dr=max1;
        if(dr<-max)
           dr=-max;
        if(di>max1)
           di=max1;
        if(di<-max)
           di=-max;
        /*Update estimator*/
        for(i=0;i<=g;i++)
           {
           p1=dr*sr[i];
           p2=di*si[i];
           p3=dr*si[i];
           p4=di*sr[i];
           fr[i]=fr[i]+p1+p2;
           fi[i]=fi[i]+p4-p3;
           /*Check for overflow*/
           if(fr[i]>max1)
              fr[i]=max1;
           if(fr[i]<-max)
              fr[i]=-max;
           if(fi[i]>max1)
              fi[i]=max1;
           if(fi[i]<-max)
              fi[i]=-max;
           }
     /*End of estimator routine*/

     }
/*Allow the estimator to settle down before switching*/
/*in the detector and Clark-Hau algorithm*/
if(k<1000)
     goto shift;

/*Run the Clark-Hau algorithm once to find prefilter*/
/*and minphase impulse responses*/
if(k==1000  )
     {
     start=0;
```

```
roots=0;
        do
          {
          start++;
          lr=max*sp[start][1];
          li=max*sp[start][2];
          tolerance=max1;
          count=0;
          /*Start of root finding process*/
          while(tolerance>=d)
            {
            count++;
            /*Pass Y thru F/B filter*/
            or[g]=fr[g];
            oi[g]=fi[g];
            o2r[g]=fr[g];
            o2i[g]=fi[g];
            for(i=(g-1);i>=0;i--)
              {
              p1=lr*or[i+1];
              p2=li*oi[i+1];
              p3=lr*oi[i+1];
              p4=li*or[i+1];
              p1=p1>>(bits-1);
              p2=p2>>(bits-1);
              p3=p3>>(bits-1);
              p4=p4>>(bits-1);
              or[i]=fr[i]-p1;
              oi[i]=fi[i]-p3;
              if(or[i]>max1)
                or[i]=max1;
              if(or[i]<-max)
                or[i]=-max;
              if(oi[i]>max1)
                oi[i]=max1;
              if(oi[i]<-max)
                oi[i]=-max;
              or[i]=or[i]+p2;
              oi[i]=oi[i]-p4;
              if(or[i]>max1)
                or[i]=max1;
```

```
    if(or[i]<-max)
        or[i]=-max;
    if(oi[i]>max1)
        oi[i]=max1;
    if(oi[i]<-max)
        oi[i]=-max;

    /*Pass sequence thru second F.B. filter*/
    /*to give epsilon*/
    p1=lr*o2r[i+1];
    p2=li*o2i[i+1];
    p3=lr*o2i[i+1];
    p4=li*o2r[i+1];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
    o2r[i]=or[i]-p1;
    o2i[i]=oi[i]-p3;
    if(o2r[i]>max1)
        o2r[i]=max1;
    if(o2r[i]<-max)
        o2r[i]=-max;
    if(o2i[i]>max1)
        o2i[i]=max1;
    if(o2i[i]<-max)
        o2i[i]=-max;
    o2r[i]=o2r[i]+p2;
    o2i[i]=o2i[i]-p4;
    if(o2r[i]>max1)
        o2r[i]=max1;
    if(o2r[i]<-max)
        o2r[i]=-max;
    if(o2i[i]>max1)
        o2i[i]=max1;
    if(o2i[i]<-max)
        o2i[i]=-max;
    }
e1=o2r[1];
e2=o2i[1];
den=e1*e1+e2*e2;
```

```
den=den>>(bits-1);
if(den>max1)
    den=max1;
den=den<<(bits-1);

/*Real part of complex division*/
p1=or[0]*e1+oi[0]*e2;
p1=p1>>(bits-1);
if(p1>max1)
   p1=max1;
if(p1<-max)
   p1=-max;
num=abs(p1);

/*Check for div by zero and if num>den*/
if(num>=den)
   {
   if(p1<0)
      w1=-max;
   else if(p1>0)
      w1=max1;
   else
      w1=0;
   }
else
   {
   /*Carry out integer division*/
   num=num<<bits;
   for(j=1;j<=(bits-1);j++)
      {
      diff=(long)(num-den);
      if(diff<0)
         num=num<<1;
      else
         {
         diff=diff<<1;
         diff++;
         num=(unsigned long)(diff);
         }
      }
   w1=(long)(num&max1);
```

```
    if(p1<0)
        w1=-w1;
    }

/*Imaginary part of complex div*/
p2=oi[0]*e1-or[0]*e2;
p2=p2>>(bits-1);
if(p2>max1)
    p2=max1;
if(p2<-max)
    p2=-max;
num=abs(p2);
if(num>=den) /*Check for div by zero*/
    {
    if(p2<0)
        w2=-max;
    else if(p2>0)
        w2=max1;
    else
        w2=0;
    }
else
    {
    /*Carry out division*/
    num=num<<bits;
    for(j=1;j<=(bits-1);j++)
        {
        diff=(long)(num-den);
        if(diff<0)
            num=num<<1;
        else
            {
            diff=diff<<1;
            diff++;
            num=(unsigned long)diff;
            }
        }
    w2=(long)(num&max1);
    if(p2<0)
        w2=-w2;
    }
```

```
                /*Check for convergence*/
                tolerance=w1*w1+w2*w2;
                if(tolerance>max1)
                    tolerance=max1;
                if(tolerance<=d)
                    break;
                w1=w1*c;
                w2=w2*c;
                w1=w1>>(bits-1);
                w2=w2>>(bits-1);

                /*Update reciprocal root positions*/
                lr=lr+w1;
                li=li+w2;
                if(lr>max1)
                    lr=max1;
                if(lr<-max)
                    lr=-max;
                if(li>max1)
                    li=max1;
                if(li<-max)
                    li=-max;
                p1=lr*lr;
                p2=li*li;
                mag=(unsigned long)(p1+p2);
                mag=mag>>bits;
                if(mag>max1)
                    mag=max1;
                /*Check to see if outside unit circle*/
                if(mag>lambmax)
                    break;
                /*Check number of iterations*/
                if(count==40)
                    break;
                }
            if(mag>lambmax)
                continue;
            if(count==40)
                continue;
roots++;
```

```
/*Pass output of F/B filter thru F/F filter*/
or[g+1]=oi[g+1]=0;
for(i=1;i<=(g+1);i++)
    {
    p1=lr*or[i-1];
    p2=li*oi[i-1];
    p3=lr*oi[i-1];
    p4=li*or[i-1];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
    fr[i-1]=or[i]+p1+p2;
    fi[i-1]=oi[i]+p3-p4;
    }
/*Calculate prefilter taps*/
d1r[length]=filtr[length];
d1i[length]=filti[length];
for(i=(length-1);i>=0;i--)
    {
    p1=lr*d1r[i+1];
    p2=li*d1i[i+1];
    p3=lr*d1i[i+1];
    p4=li*d1r[i+1];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
   d1r[i]=filtr[i]-p1+p2;
   d1i[i]=filti[i]-p3-p4;
    }
d1r[length+1]=d1i[length+1]=0;
for(i=1;i<=(length+1);i++)
    {
    p1=lr*d1r[i-1];
    p2=li*d1i[i-1];
    p3=lr*d1i[i-1];
    p4=li*d1r[i-1];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
```

```
            p4=p4>>(bits-1);
            filtr[i-1]=d1r[i]+p1+p2;
            filti[i-1]=d1i[i]+p3-p4;
            }
        start=0;
        }
    while(start<5);

}
/*Pass rx signal thru pre-filter*/
opr=opi=0;
for(i=0;i<=length;i++)
    {
    p1=sigr[i]*filtr[i];
    p2=sigi[i]*filti[i];
    p3=sigr[i]*filti[i];
    p4=sigi[i]*filtr[i];
    p1=p1>>(bits-1);
    p2=p2>>(bits-1);
    p3=p3>>(bits-1);
    p4=p4>>(bits-1);
    opr=opr+p1-p2;
    opi=opi+p3+p4;
    if(opr>(max-1))
        opr=max-1;
    if(opr<-max)
        opr=-max;
    if(opi>(max-1))
        opi=max-1;
    if(opi<-max)
        opi=-max;
    }

/*Shift signal*/
for(i=length;i>=1;i--)
    {
    sigr[i]=sigr[i-1];
    sigi[i]=sigi[i-1];
    }

/*Expand the m Q vectors to give ml vectors*/
```

```
mincost=max1;
for(j=1;j<=m;j++)
   {
   /*Evaluate the cost for m vectors*/
   cr=ci=0;
   for(i=0;i<=(n-1);i++)
      {
      p1=qr[j][i]*fr[n-i];
      p2=qi[j][i]*fi[n-i];
      p3=qr[j][i]*fi[n-i];
      p4=qi[j][i]*fr[n-i];
      cr=cr+((p1-p2)>>4);
      ci=ci+((p3+p4)>>4);
      if(cr>(max-1))
         cr=max-1;
      if(cr<-max)
         cr=-max;
      if(ci>(max-1))
         ci=max-1;
      if(ci<-max)
         ci=-max;
      }
   zr=(opr-cr)  ;
   zi=(opi-ci)  ;
   if(zr>max1)
      zr=max1;
   if(zr<-max)
      zr=-max;
   if(zi>max1)
      zi=max1;
   if(zi<-max)
      zi=-max;
   p1=fr[0]*zr;
   p2=fi[0]*zi;
   p3=fr[0]*zi;
   p4=fi[0]*zr;
   zr=p1+p2;
   zi=p3-p4;
   zr=zr>>(bits-2);
   zi=zi>>(bits-2);
   if(zr>max1)
```

```
    zr=max1;
if(zr<-max)
    zr=-max;
if(zi>max1)
    zi=max1;
if(zi<-max)
    zi=-max;
p1=fr[0]*fr[0];/*Calc modulus of first component*/
p2=fi[0]*fi[0];/*of sampled impulse response*/
p1=p1+p2;
p1=p1>>(bits-2);
p1=p1>>4;
if(p1>max1) p1=max1;
mod=p1;
p1=6*mod;
p2=4*mod;
p3=2*mod;
/*Threshold comparison*/
if(zr<=-p1)
    q1=-7;
else if((zr>-p1)&&(zr<=-p2))
    q1=-5;
else if((zr>-p2)&&(zr<=-p3))
    q1=-3;
else if((zr>-p3)&&(zr<=0))
    q1=-1;
else if((zr>0)&&(zr<=p3))
    q1=1;
else if((zr>p3)&&(zr<=p2))
     q1=3;
else if((zr>p2)&&(zr<=p1))
    q1=5;
else if(zr>p1)
    q1=7;
if(zi<=-p1)
    q2=-7;
else if((zi>-p1)&&(zi<=-p2))
    q2=-5;
else if((zi>-p2)&&(zi<=-p3))
    q2=-3;
else if((zi>-p3)&&(zi<=0))
```

```
    q2=-1;
else if((zi>0)&&(zi<=p3))
    q2=1;
else if((zi>p3)&&(zi<=p2))
    q2=3;
else if((zi>p2)&&(zi<=p1))
    q2=5;
else if(zi>p1)
    q2=7;
p1=zr-q1*mod;
p2=zi-q2*mod;
if(p1>max1)
    p1=max1;
if(p1<-max)
    p1=-max;
if(p2>max1)
    p2=max1;
if(p2<-max)
    p2=-max;

/*Calculate cost associated with selected vector*/
p3=p1*p1;
p4=p2*p2;
p3=p3>>(bits-4);
p4=p4>>(bits-4);
p3=p3+p4+cost[j];
if(p3>max1)
    p3=max1;
coste[j*2-1]=p3;
tag[j*2-1][0]=q1;
tag[j*2-1][1]=q2;
if(p3<=mincost)
    {
    mincost=p3;
    vector=j;
    vector2=1;
    }
/*Select next data symbol*/
if(abs(p1)>=abs(p2))
    {
    /*magnitude of real part >mag of imag part*/
```

```
    if((p1>=0)&&(q1!=7))
       q1=q1+2;
    else if((p1<0)&&(q1!=-7))
       q1=q1-2;
    /*Real part of xi=most -ve or +ve value*/
    /*Change imag part instead*/
    else if((p2>=0)&&(q2!=7))
       q2=q2+2;
    else if((p2<0)&&(q2!=-7))
       q2=q2-2;
    else if((p2>=0)&&(q2==7))
       q2=q2-2;
    else if((p2<0)&&(q2==-7))
       q2=q2+2;
    }
else
    {
    /*Magnitude of imag part is greater*/
    if((p2>=0)&&(q2!=7))
       q2=q2+2;
    else if((p2<0)&&(q2!=-7))
       q2=q2-2;
    /*Imag part of xi=most -ve or +ve value*/
    else if((p1>=0)&&(q1!=7))
       q1=q1+2;
    else if((p1<0)&&(q1!=-7))
       q1=q1-2;
    else if((p1>=0)&&(q1==7))
       q1=q1-2;
    else if((p1<0)&&(q1==-7))
       q1=q1+2;
    }
p1=zr-q1*mod;
p2=zi-q2*mod;
if(p1>max1)
   p1=max1;
if(p1<-max)
   p1=-max;
if(p2>max1)
   p2=max1;
if(p2<-max)
```

```
        p2=-max;

    /*Calculate cost of vector*/
    p3=p1*p1;
    p4=p2*p2;
    p3=p3>>(bits-4);
    p4=p4>>(bits-4);
    p3=p3+p4+cost[j];
    if(p3>max1)
        p3=max1;
    coste[j*2]=p3;
    tag[j*2][0]=q1;
    tag[j*2][1]=q2;
    if(p3<=mincost)
        {
        mincost=p3;
        vector=j;
        vector2=2;
        }
    }

smallest_cost=mincost;


/*Output detected value*/
sdr=qr[vector][0];
sdi=qi[vector][0];

/*Obtain bits associated with Rx symbol*/
col=(sdr+7)/2;
row=(sdi+7)/2;
bit1=detbit[row][col][0];
bit2=detbit[row][col][1];
bit3=detbit[row][col][2];
bit4=detbit[row][col][3];
bit5=detbit[row][col][4];
bit6=detbit[row][col][5];

/*Differentially decode bits*/
x=bite21*8+bite22*4+bit1*2+bit2;
bite21=bit1;
```

```
        bite22=bit2;
        bit1=decode1[x];
        bit2=decode2[x];

        /*Count errors*/
        if(k>2000)
        {
        bits_sent=bits_sent+6;
        if(bit1!=tx[0][0])
            err1++;
        if(bit2!=tx[0][1])
            err1++;
        if(bit3!=tx[0][2])
            err1++;
        if(bit4!=tx[0][3])
            err1++;
        if(bit5!=tx[0][4])
            err1++;
        if(bit6!=tx[0][5])
            err1++;
        }
if((err1>20000)&&(pe>=.01)) break;
else if((err1>500)&&(pe<0.01)) break;

        /*Store vector with smallest cost in array p*/
        for(i=0;i<=(n-1);i++)
            {
            pr[1][i]=qr[vector][i];
            pi[1][i]=qi[vector][i];
            }
        pr[1][n]=tag[vector*2-2+vector2][0];
        pi[1][n]=tag[vector*2-2+vector2][1];
        pcost[1]=smallest_cost;
        coste[vector*2-2+vector2]=max1;

        /*Discard all costs where first component<>to detected*/
        /*value*/
        for(j=1;j<=m;j++)
            {
            if((qr[j][0]!=sdr)||(qi[j][0]!=sdi))
                {
```

```
        coste[j*2-1]=max1;
        coste[j*2]=max1;
        }
    }

/*Select the next m-1 vectors and store in p*/
count=2;
do
    {
    mincost=coste[1];
    vector=1 ;
    vector2=1;
    for(j=1;j<=m;j++)
        {
        if(coste[j*2-1]< mincost)
            {
            mincost=coste[j*2-1];
            vector=j;
            vector2=1;
            }
        if(coste[j*2]< mincost)
            {
            mincost=coste[j*2];
            vector=j;
            vector2=2;
            }
        }

    pcost[count]=mincost;
    coste[vector*2-2+vector2]=max1;

    /*Transfer vector to array p*/
    for(i=0;i<=(n-1);i++)
        {
        pr[count][i]=qr[vector][i];
        pi[count][i]=qi[vector][i];
        }
    pr[count][n]=tag[vector*2-2+vector2][0];
    pi[count][n]=tag[vector*2-2+vector2][1];
    count++;
    }
```

```
        while(count<=m);

        /*Replace stored vectors by newly selected vectors*/
        for(j=1;j<=m;j++)
            {
            for(i=1;i<=n;i++)  /*Discard first component in q*/
                {
                qr[j][i-1]=pr[j][i];
                qi[j][i-1]=pi[j][i];
                }
            cost[j]=pcost[j]-smallest_cost;     /*Normalize*/
            }

        /*Shift tx symbols*/
shift:   for(i=g;i>=1;i--)
            {
            sr[i]=sr[i-1];
            si[i]=si[i-1];
            }

        /*Shift tx data*/
        for(i=1;i<=(n+length);i++)
            {
            tx[i-1][0]=tx[i][0];
            tx[i-1][1]=tx[i][1];
            tx[i-1][2]=tx[i][2];
            tx[i-1][3]=tx[i][3];
            tx[i-1][4]=tx[i][4];
            tx[i-1][5]=tx[i][5];
            }
        }
    error1=err1;
    pe=error1/bits_sent;
    printf("snr=%lf  pe=%lf \n",snr,pe);
    fprintf(results,"%lf,%lf\n",snr,pe);
    snr=snr+1.0;
if(pe<0.0001) break;
    }
    while(snr<=snrfin);
fclose(results);
}
```

```
/*2 sky wave H.F. channel simulator*/
/*26/9/89*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#pragma linkage(g05cbf,OS);
#pragma linkage(g05ddf,OS);

void g05cbf();
double g05ddf();

main()
   {
   FILE *qres;

   int i,j,g,n,row,k,l1,h,h2,k1,l;
   int step;

   double fsp,z1,z2,z3,c1,c2,c3,c4,c5,q1,q2,q3,q4;
   double grad,br,bi,a1r,a1i,a2r,a2i,yr,yi,p1,p2,p3,p4;
   double f1,f2,f3,s1;
   double tap1[4],tap2[4],tap3[4],tap4[4],tap5[4];
   double qnew[4],qold[4],qi[4][150];
   double mean,var;

   /*Bessel filter coefficients for 1/2, 1 and 2 Hz spread*/
   double coefs[3][6]={
    {0.946,1.9,0.903135,1.9276,0.9316562},
    {0.895,1.8036,0.8155376,1.8524,0.8680474},
    {0.801,1.6218,0.6650064,1.6954,0.7536391}};
   double g0[3]={325623.4,16121.6,893.06};

   /*Receiver filter impulse response*/
   double rxr[]={-1.9417691,-15.9797864,-35.1417733,-34.4788717,
    -11.2301982,7.8155160,7.5124057,-0.5057505,-3.3707125,
    -0.6759166,1.0482656,0.3621876,-0.3105902,0.043841,
    0.0738947,-0.0646936,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
   double rxi[] ={1.3625952,11.5941040,27.3342937,28.0870086,
```

```
 7.2714615,-9.2602472,-5.0954462,3.2326498,1.8975352,
 -1.2813604,-0.4830313,0.7614804,0.1979014,-0.1532672,
 0.094033,-0.0312132,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

 /*Transmitter filter no delay*/
 double txr0[] ={-0.1795896,-3.0773455,-9.9409021,-11.7869473,
 -3.4618271,4.4438154,3.0642536,-1.3596576,-1.4973528,
 0.2925598,0.5180829,-0.1842786,-0.3167778,0.0021899,
 -0.0443806,0.0515533,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 double txi0[] ={2.3539405,20.7590237,45.5584592,41.4909978,
 8.7045826,-11.7869820,-5.5819054,3.1582131,1.7365460,
 -0.7776891,-0.1292556,0.2880296,-0.2324818,-0.2107548,
 0.0392056,0.0098505,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

 /*Xmitter  filter 0.5 ms delay*/
/*  double txr05[] ={0,0,0,-0.7630237,-5.648724,-11.921618,
 -9.3588701,0.5649945,4.9376234,1.0473111,-1.9765922,
 -0.7164903,0.594387,0.2543641,-0.3636354,-0.15437,
 -0.0228239,0.016722,-0.0609485,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 double txi05[] ={0,0,0,7.3452371,31.9050405,48.771819,
 29.8079459,-3.0207994,-11.4979467,-0.9822748,3.505327,
 0.3116039,-0.7219063,0.2044829,0.1084602,-0.328697,
 -0.0636352,0.0278816,0.018593,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
*/
 /*Transmitter filter 1.1ms delay*/
 double txr11[]= {0,0,0,0,0,-1.6694374,-7.8492148,-12.3887079,
 -6.6023157,2.9408554,4.3005084,-0.3368383,-1.9014342,
 -0.1433592,0.6242601,0.0278577,-0.3820071,-0.0416905,
 -0.0439705,0.0749333,-0.0594132,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 double txi11[]= {0,0,0,0,0,13.2372707,39.6493461,46.9272219,
 19.2346609,-8.8804125,-9.0256163,1.6284281,2.8139013,
 -0.4311352,-0.4537174,0.3081762,-0.0772327,-0.3043271,
 0.0085057,0.0093809,0.0094992,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
    /*Transmitter filter 3ms delay*/
/*  double txr3[]= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1.3136537,
    -7.1104051,-12.3469721,-7.5848703,2.2353854,4.5938614,
    0.0931639,-1.9704176,-0.3233694,0.6313238,0.1035718,
    -0.3865939,-0.0734526,-0.0386471,0.0608046,-0.0713496};
    double txi3[]= {0,0,0,0,0,0,0,0,0,0,0,0,0,0,11.0688962,
    37.2136597,47.9575159,22.8262482,-7.249859,-10.0026703,
    0.8695437,3.10728,-0.2261096,-0.5552906,0.2882096,
    -0.0156703,-0.321577,-0.0107706,0.0140909,0.0135711};
*/
    /*Set up the random number generators*/
    g05cbf(49 );

    g=17;
    n=1150 ; /*No. of Yi generated=(n-100)*48*/

    row=1; /*1/2Hzspread*/

    /*Set up the coefficients of the Bessel filter*/
    c1=coefs[row][0];
    c2=coefs[row][1];
    c3=coefs[row][2];
    c4=coefs[row][3];
    c5=coefs[row][4];

    /*Scale equipment filters*/
    f1=f2=f3=0;
    for(i=0;i<=30;i++)
        {
        f1=f1+txr0[i]*txr0[i]+txi0[i]*txi0[i];
        f2=f2+txr11[i]*txr11[i]+txi11[i]*txi11[i];
/*      f2=f2+txr3[i]*txr3[i]+txi3[i]*txi3[i];*/
/*      f2=f2+txr05[i]*txr05[i]+txi05[i]*txi05[i]; */
        f3=f3+rxr[i]*rxr[i]+rxi[i]*rxi[i];
        }
    f1=sqrt(f1);
    f2=sqrt(f2);
    f3=sqrt(f3);

    /*Reset arrays*/
    for(i=0;i<=3;i++)
```

```
    {
    qnew[i]=qold[i]=0;
    tap1[i]=tap2[i]=tap3[i]=tap4[i]=tap5[i]=0;
    }

for(i=0;i<=149;i++)
    qi[0][i]=qi[1][i]=qi[2][i]=qi[3][i]=0;

/*begining of loop to generate Yi*/
qres=fopen("results1 data","w"); /*Store results in results1*/
step=0;
for(j=0;j<=n;j++)
    {
/*Generation of the q1,q2,q3,q4*/
for(l1=0;l1<=3;l1++)
    {
    z1=g05ddf(0.0,1.0); /*Generate samples at 50Hz*/
    z2=z1+c4*tap4[l1]-c5*tap5[l1];
    z3=z2+c2*tap2[l1]-c3*tap3[l1];
    qold[l1]=qnew[l1];
    qnew[l1]=z3+c1*tap1[l1]; /*Output of filter=qnew*/

    /*Shift samples in Bessel filter*/
    tap5[l1]=tap4[l1];
    tap4[l1]=z2;
    tap3[l1]=tap2[l1];
    tap2[l1]=z3;
    tap1[l1]=qnew[l1];

    /*Use linear interpolation to increase the sampling rate*/
    /*to 4800 samples per second*/
    grad=(qnew[l1]-qold[l1])/96; /*Calc grad in y=mx+c*/
    grad=grad/g0[row];  /*Scale output of filter*/
    for(i=0;i<=95;i++)
      {
      qi[l1][i+50]=grad*i+qold[l1]/g0[row];
      }
    }

/*Perform convolution*/
for(i=0;i<=47;i++) /*Generate 48 impulse responses*/
```

```
{
s1=0;
for(h=0;h<=g;h++) /*Generate g+1 samples per impulse resp*/
   {
   h2=2*h;
   yr=yi=0;
      for(k=0;k<=h2;k++) /*generate one sample*/
         {
         a1r=txr0[k]/f1; /*Scale Tx and Rx filter taps*/
         a1i=txi0[k]/f1;
         a2r=txr11[k]/f2;
         a2i=txi11[k]/f2;
         br=rxr[h2-k]/f3;
         bi=rxi[h2-k]/f3;
         k1=50+2*(i-h)+k; /*Set up counter*/
         q1=qi[0][k1];
         q2=qi[1][k1];
         q3=qi[2][k1];
         q4=qi[3][k1];
         p1=a1r*q1+a1i*q2;
         p2=a1i*q1-a1r*q2;
         p3=a2r*q3+a2i*q4;
         p4=a2i*q3-a2r*q4;
         p1=p1+p3;
         p2=p2+p4;
         p3=p1*br-p2*bi;
         p4=p1*bi+p2*br;
         yr=yr+p3;
         yi=yi+p4;
         }

      /*Allow Bessel filter to settle and output sample*/
      if(j>100)
         fprintf(qres,"%lf,%lf\n",yr,yi);

      s1=s1+yr*yr+yi*yi;
      }
/*    step++;
   if((j>100)&&(step>=10))
   {
   s1=sqrt(s1);
```

```
            fprintf(qres,"%lf \n",10*log10(s1));
            step=0;
            }*/
            }

    /*Shift samples ready for next 48 responses*/
     for(i=49;i>=0;i--)
         {
         qi[0][i]=qi[0][i+95];
         qi[1][i]=qi[1][i+95];
         qi[2][i]=qi[2][i+95];
         qi[3][i]=qi[3][i+95];
         }

}
fclose(qres);
}
```

```
/*This program simulates the transmission of data over an H.F.*/
/*radio link.It uses a 4-level QAM signal and data is transmitted*/
/*at 4800 bps.The linear feedforward channel estimator is used with*/
/*degree one predicton.Integer arithmetic is used.
/*7/4/90*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#pragma linkage(g05cbf,OS);
#pragma linkage(g05ddf,OS);

void g05cbf();
double g05ddf();

main()
    {
    FILE *channel;
    FILE *rxfilt;
    FILE *results;

    int i,j,k,g,m,l,num1,num2,n,h,vector,q1,q2,count,sdr,sdi,err1;
    int tag[260][3],tx1[80],tx2[80];
    int pr[18][33],pi[18][33],qr[18][33],qi[18][33];
    int levelr[5 ]={1,-1,-1,1,1};
    int leveli[5 ]={1,-1,1,1,-1};
    int bits,bit1,bit2,start,limit,err2,row,col;
    int length,update;
    int earlyr,earlyi,roots,vector2,retrain;
    long total_bits_sent;
    long max,opr,opi,p1,p2,p3,p4,cr,ci,zr,zi,ad,rdr,rdi,vr,vi;
    long mincost,smallest_cost,dr,di,max1,c,lr,li,diff,e1,e2;
    long yminr[31],ymini[31],filtr[50],filti[50],or[30],oi[30];
    long cost[260],coste[260],pcost[260],sigr[50],sigi[50];
    long er[50],ei[50],fr[50],fi[50],sr[60],si[60],foldr[30];
    long o2r[30],o2i[30],w1,w2,d1r[50],d1i[50],symbols,foldi[30];
    long datr[60],dati[60],deltar2[30],deltai2[30];
    long z1,z2,cd,deltar1,deltai1;

    unsigned long tolerance,d,den,num,div,mag,lambmax,bita,bitb;
```

```
unsigned long bitc,bitd;

double yr[50],yi[50],sp[9][2];
double noiser[50],noisei[50],error1,a,b,d1,error2;
double rr,ri,nr,ni,stddev,snr,pe,snrfin,ur,ui,f1,cc;

/*Reciever filter impulse response*/
/*Sampling rate =2400*/
double rxr[]= {-1.9417691,-35.1417733,-11.2301982,
 7.5124057,-3.3707126,1.0482656,-0.3105902,0.0738947};
double rxi[]= {1.3625952,27.3342937,7.2714615,-5.0954462,
 1.8975352,-0.4830313,0.1979014,0.094033};

/*Starting positions used in the Clark-Hau Algorithm*/
sp[1][1]=0;
sp[1][2]=0;
sp[2][1]=0.9;
sp[2][2]=0;
sp[3][1]=0.6364;
sp[3][2]=0.6364;
sp[4][1]=0;
sp[4][2]=0.9;
sp[5][1]=-0.6364;
sp[5][2]=0.6364;
sp[6][1]=-0.9;
sp[6][2]=0;
sp[7][1]=-0.6364;
sp[7][2]=-0.6364;
sp[8][1]=0;
sp[8][2]=-0.9;
sp[9][1]=0.6364;
sp[9][2]=-0.6364;

g=22;  /*g is the length of the channel*/
l=4 ;  /*l is the number of levels*/
m=16;  /*m is the number of stored vectors*/
n=20 ; /*n is the delay in detection*/
length=29;  /*Length of the pre-filter*/
symbols=50000 ; /*number of xmitted symbols*/
a=0.03 ;       /*Controls the convergence of Estimator*/
b=0.5;         /*Controls the convergence of Clark-Hau*/
```

```
d1=0.000001 ;    /*Decision threshold*/

bits=16;
max=1;
max=max<<(bits-1);
max1=max-1;
ad=a*max;
c=b*max;
if(c>=max)
   c=max1;
d=d1*max*max;
cd=cc*max;
lambmax=max>>1;

snr=25;
snrfin=100;

/*Initialise random number generators*/
srand(54 );
g05cbf(71 );

/*Scale receiver filter impulse response*/
f1=0;
for(i=0;i<=7;i++)
   {
   f1=f1+rxr[i]*rxr[i]+rxi[i]*rxi[i];
   }
f1=sqrt(f1);

/*Set up the tags for the near max likelihood detector*/
for(j=1;j<=m;j++)
   {
   for(h=1;h<=l;h++)
      {
      tag[h+l*j-l][1]=j;
      tag[h+l*j-l][2]=h;
      }
   }

/*Store values of Pe in file called results data*/
results=fopen("results data","w");
```

```
/*Start of main loop that varies SNR*/
do
{
/*Calculate standard deviation*/
stddev=pow(10,(-snr/20));

/*Initialise stored vectors in the detector*/
for(j=1;j<=m;j++)
    {
    for(i=0;i<=n;i++)
        {
        qr[j][i]=1;
        qi[j][i]=1;
        pr[j][i]=0;
        pi[j][i]=0;
        }
    }

/*set up the cost array*/
for(j=1;j<=m;j++)
    {
    cost[j]=max1;
    }
cost[1]=0;
cost[2]=10;
cost[3]=20;
cost[4]=40;
cost[5]=50;
cost[6]=60;
cost[7]=70;
cost[8]=80;
/*Reset estimator*/
for(i=0;i<=31;i++)
    {
    fr[i]=fi[i]=0;
    datr[i]=dati[i]=0;
    }
for(i=0;i<=25;i++)
    ymini[i]=yminr[i]=0;
```

```
/*Set the reference bits for the differential decoder*/
update=0;
retrain=0;
total_bits_sent=0;
err1=0;
channel=fopen("hfch1 data","r");

/*Start of Transmission*/
for(k=1;k<=symbols;k++)
    {
    update++;
    if(k>2000)
        retrain++;
    /*Generate random bits*/
    bita=rand();
    bitb=rand();
    /*Encode the bits using grey coding*/
    if(bita>16383)
        bit1=1;
    else
        bit1=0;
    if(bitb>16383)
        bit2=1;
    else
        bit2=0;

    tx1[n+length ]=bit1;
    tx2[n+length ]=bit2;

    /*Generate transmitted symbols*/
    if((bit1==0)&&(bit2==0))
        {
        sr[0]=-1;
        si[0]=-1;
        }
    else if((bit1==0)&&(bit2==1))
        {
        sr[0]=1;
        si[0]=-1;
        }
    else if((bit1==1)&&(bit2==0))
```

```
    {
    sr[0]=-1;
    si[0]=1;
    }
else
    {
    sr[0]=1;
    si[0]=1;
    }

/*Read in the channel impulse response*/
 for(i=0;i<=g;i++)
    fscanf(channel,"%lf,%lf",&yr[i],&yi[i]);

/*Feed si into the channel*/
rr=ri=0;
for(i=0;i<=g;i++)
    {
    rr=rr+sr[i]*yr[i]-si[i]*yi[i];
    ri=ri+sr[i]*yi[i]+si[i]*yr[i];
    }
noiser[0]=g05ddf(0,stddev);
noisei[0]=g05ddf(0,stddev);

/*Pass noise through rx filter*/
nr=ni=0;
for(i=0;i<=7;i++)
    {
    nr=nr+noiser[i]*rxr[i]-noisei[i]*rxi[i];
    ni=ni+noiser[i]*rxi[i]+noisei[i]*rxr[i];
    }

/*Scale output of reciever*/
nr=nr/f1;
ni=ni/f1;

/*Shift noise samples*/
for(i=7;i>=1;i--)
    {
    noiser[i]=noiser[i-1];
    noisei[i]=noisei[i-1];
```

```
    }

/*Add noise to the received signal*/
rr=rr+nr;
ri=ri+ni;

/*Convert received signal into a digital signal*/
rr=rr/4;
ri=ri/4;
rdr=rr*max;
rdi=ri*max;
if(rdr>max1)
    rdr=max1;
if(rdr<-max)
    rdr=-max;
if(rdi>max1)
    rdi=max1;
if(rdi<-max)
    rdi=-max;

/*Feed received signal into the pre-filter*/
sigr[0]=rdr;
sigi[0]=rdi;

/*Feed in a known sequence into the estimator for startup*/
/*then change over to using the early detected data*/
if((k<2000)||((retrain>=1000)&&(retrain<1200)))
    {
    datr[0]=sr[length+1 ];
    dati[0]=si[length+1 ];
    }
else
    {
    if(retrain==1200) retrain=0;
    datr[0]=earlyr;
    dati[0]=earlyi;
    }

/*Degree 1 prediction*/
for(i=0;i<=g;i++)
    {
```

```
p1=((fr[i]-foldr[i])<<4)-deltar2[i];
p2=((fi[i]-foldi[i])<<4)-deltai2[i];
/*Check for overflow*/
if(p1>max1)
   p1=max1;
if(p1<-max)
   p1=-max;
if(p2>max1)
   p2=max1;
if(p2<-max)
   z2=-max;
p1=p1*(max*.04);
p2=p2*(max*.04);
p1=p1>>(bits-1);
p2=p2>>(bits-1);
deltar1=deltar2[i]+p1;
deltai1=deltai2[i]+p2;
if(deltar1>max1)
   deltar1=max1;
if(deltar1<-max)
   deltar1=-max;
if(deltai1>max1)
   deltai1=max1;
if(deltai1<-max)
   deltai1=-max;
foldr[i]=fr[i];
foldi[i]=fi[i];

/*Update old estimate of the sampled impulse response*/
fr[i]=fr[i]+(deltar1>>4);
fi[i]=fi[i]+(deltai1>>4);
deltar2[i]=deltar1;
deltai2[i]=deltai1;

/*Check for overflow*/
if(fr[i]>max1)
   fr[i]=max1;
if(fr[i]<-max)
   fi[i]=-max;
if(fi[i]>max1)
   fi[i]=max1;
```

```
    if(fi[i]<-max)
        fi[i]=-max;
    }

/*Channel estimator routine*/
    vr=vi=0;
    /*Feed data into estimate of the channel*/
    for(i=0;i<=g;i++)
        {
        p1=fr[i]*datr[i];
        p2=fi[i]*dati[i];
        p3=fi[i]*datr[i];
        p4=fr[i]*dati[i];
        vr=vr+p1-p2;
        vi=vi+p3+p4;
        }
        vr=vr>>1;
        vi=vi>>1;

        /*Check for overflow*/
        if(vr>max1)
            vr=max1;
        if(vr<-max)
            vr=-max;
        if(vi>max1)
            vi=max1;
        if(vi<-max)
            vi=-max;

        dr=sigr[length+1 ];
        di=sigi[length+1 ];
        dr=dr*ad;
        di=di*ad;
        vr=vr*ad;
        vi=vi*ad;

        /*Calculate difference between output of estimator*/
        /*and the actual received sample*/
        dr=dr-vr;
        di=di-vi;
        dr=dr>>7 ;
```

```
di=di>>7 ;

/*Check for overflow*/
if(dr>max1)
   dr=max1;
if(dr<-max)
   dr=-max;
if(di>max1)
   di=max1;
if(di<-max)
   di=-max;

/*Update estimator*/
for(i=0;i<=g;i++)
   {
   p1=dr*datr[i];
   p2=di*dati[i];
   p3=dr*dati[i];
   p4=di*datr[i];
   p1=p1+p2;
   p1=p1>>8;
   p4=p4-p3;
   p4=p4>>8;
   fr[i]=fr[i]+p1   ;
   fi[i]=fi[i]+p4   ;

   /*Check for overflow*/
   if(fr[i]>max1)
      fr[i]=max1;
   if(fr[i]<-max)
      fr[i]=-max;
   if(fi[i]>max1)
      fi[i]=max1;
   if(fi[i]<-max)
      fi[i]=-max;
   }

/*Shift the data that is stored in the shift*/
/*register of the estimator*/
for(i=22;i>=1;i--)
   {
```

```
            datr[i]=datr[i-1];
            dati[i]=dati[i-1];
            }
    /*End of the channel estimator routine*/
    /*Estimate is stored in the vector fr+jfi*/

/*Allow the estimator to settle down before switching*/
/*in the detector and Clark-Hau algorithm*/
if(k<1000)
    goto shift1;

/*Run the Clark-Hau algorithm once every 4 symbols periods*/
if(update>=4 )
    {
    update=0;
    start=0;
    roots=0;

    /*Initialize minimum phase impulse response*/
    for(i=0;i<=g;i++)
        {
        yminr[i]=fr[i];
        ymini[i]=fi[i];
        }

    /*Initialize pre-filter*/
    for(i=0;i<=length;i++)
        filtr[i]=filti[i]=0;
    filtr[length]=max1;

    do
        {
        start++;

        /*Get a starting point from array*/
        lr=max*sp[start][1];
        li=max*sp[start][2];

        tolerance=max1;
        count=0;
```

```
/*Start of root finding process*/
while(tolerance>=d)
    {
    count++;

    /*Pass Y thru F/B filter*/
    or[g]=yminr[g];
    oi[g]=ymini[g];
    o2r[g]=yminr[g];
    o2i[g]=ymini[g];
    for(i=(g-1);i>=0;i--)
        {
        p1=lr*or[i+1];
        p2=li*oi[i+1];
        p3=lr*oi[i+1];
        p4=li*or[i+1];
        p1=p1>>(bits-1);  /*Scaling*/
        p2=p2>>(bits-1);
        p3=p3>>(bits-1);
        p4=p4>>(bits-1);
        or[i]=yminr[i]-p1;
        oi[i]=ymini[i]-p3;

        /*Check for overflow*/
        if(or[i]>max1)
            or[i]=max1;
        if(or[i]<-max)
            or[i]=-max;
        if(oi[i]>max1)
            oi[i]=max1;
        if(oi[i]<-max)
            oi[i]=-max;
        or[i]=or[i]+p2;
        oi[i]=oi[i]-p4;

        /*Check for overflow*/
        if(or[i]>max1)
            or[i]=max1;
        if(or[i]<-max)
            or[i]=-max;
        if(oi[i]>max1)
```

```
    oi[i]=max1;
if(oi[i]<-max)
    oi[i]=-max;
/*Arrays or and oi hold output of F/B*/
/*filter. Now pass this into second*/
/*F/B filter to give epsilon*/

p1=lr*o2r[i+1];
p2=li*o2i[i+1];
p3=lr*o2i[i+1];
p4=li*o2r[i+1];
p1=p1>>(bits-1);
p2=p2>>(bits-1);
p3=p3>>(bits-1);
p4=p4>>(bits-1);
o2r[i]=or[i]-p1;
o2i[i]=oi[i]-p3;

/*Check for overflow*/
if(o2r[i]>max1)
    o2r[i]=max1;
if(o2r[i]<-max)
    o2r[i]=-max;
if(o2i[i]>max1)
    o2i[i]=max1;
if(o2i[i]<-max)
    o2i[i]=-max;
o2r[i]=o2r[i]+p2;
o2i[i]=o2i[i]-p4;

/*Check for overflow*/
if(o2r[i]>max1)
    o2r[i]=max1;
if(o2r[i]<-max)
    o2r[i]=-max;
if(o2i[i]>max1)
    o2i[i]=max1;
if(o2i[i]<-max)
    o2i[i]=-max;
}
```

```
e1=o2r[1]; /*Epsilon*/
e2=o2i[1];

/*Calculate denominator for complex division*/
den=e1*e1+e2*e2;
den=den>>(bits-1);
if(den>max1)
   den=max1;
den=den<<(bits-1);

/*Calculate numerator of real part of div*/
p1=or[0]*e1+oi[0]*e2;
p1=p1>>(bits-1);
if(p1>max1)
   p1=max1;
if(p1<-max)
   p1=-max;

/*Check for special cases, i.e. div by zero*/
/*or if numerator is greater then den*/
num=abs(p1);
if(num>=den)
   {
   if(p1<0)
      w1=-max;
   else if(p1>0)
      w1=max1;
   else
      w1=0;
   }
else
   {
   /*Perform long division*/
   num=num<<bits;
   for(j=1;j<=(bits-1);j++)
      {
      diff=(long)(num-den);
      if(diff<0)
         num=num<<1;
      else
         {
```

```
            diff=diff<<1;
            diff++;
            num=(unsigned long)(diff);
            }
        }
    w1=(long)(num&max1);
    if(p1<0)
        w1=-w1;
    }

/*Calculate numerator of imaginary part of div*/
p2=oi[0]*e1-or[0]*e2;
p2=p2>>(bits-1);
if(p2>max1)
    p2=max1;
if(p2<-max)
    p2=-max;

/*Check for special cases*/
num=abs(p2);
if(num>=den)
    {
    if(p2<0)
        w2=-max;
    else if(p2>0)
        w2=max1;
    else
        w2=0;
    }
else
    {
    num=num<<bits;
    for(j=1;j<=(bits-1);j++)
        {
        /*Perform long division*/
        diff=(long)(num-den);
        if(diff<0)
            num=num<<1;
        else
            {
            diff=diff<<1;
```

```
            diff++;
            num=(unsigned long)diff;
            }
        }
    w2=(long)(num&max1);
    if(p2<0)
        w2=-w2;
    }

tolerance=w1*w1+w2*w2;
if(tolerance>max1)
    tolerance=max1;

/*Check for convergence*/
if(tolerance<=d)
    break;

/*Multiply complex division by constant c*/
w1=w1*c;
w2=w2*c;
w1=w1>>(bits-1);
w2=w2>>(bits-1);

/*Update reciprocal root positions*/
lr=lr+w1;
li=li+w2;

/*Check for overflow*/
if(lr>max1)
    lr=max1;
if(lr<-max)
    lr=-max;
if(li>max1)
    li=max1;
if(li<-max)
    li=-max;

/*Check to see if magnitude of new reciprocal*/
/*is greater than unity*/
p1=lr*lr;
p2=li*li;
```

```
    mag=(unsigned long)(p1+p2);
    mag=mag>>bits;
    if(mag>max1)
       mag=max1;
    /*Check to see if outside unit circle*/
    if(mag>=lambmax)
       break;

    /*Check number of iterations*/
    if(count==100)
       break;
    }

if(mag>lambmax)
   continue;
/*Check number of iterations*/
if(count==100)
   continue;

/*Pass output of F/B filter thru F/F filter*/
or[g+1]=oi[g+1]=0;
for(i=1;i<=(g+1);i++)
   {
   p1=lr*or[i-1];
   p2=li*oi[i-1];
   p3=lr*oi[i-1];
   p4=li*or[i-1];
   p1=p1>>(bits-1);
   p2=p2>>(bits-1);
   p3=p3>>(bits-1);
   p4=p4>>(bits-1);
 /*Update minimum phase response*/
 yminr[i-1]=or[i]+p1+p2;
 ymini[i-1]=oi[i]+p3-p4;

 /*Check for overflow*/
 if(yminr[i-1]>max1)
    yminr[i-1]=max1;
 if(yminr[i-1]<-max)
    yminr[i-1]=-max;
 if(ymini[i-1]>max1)
```

```
        ymini[i-1]=max1;
      if(ymini[i-1]<-max)
        ymini[i-1]=-max;
       }

    /*Calculate prefilter taps*/
    d1r[length]=filtr[length];
    d1i[length]=filti[length];
    j=length-1;
    for(i=j       ;i>=0;i--)
       {
       p1=lr*d1r[i+1];
       p2=li*d1i[i+1];
       p3=lr*d1i[i+1];
       p4=li*d1r[i+1];
       p1=p1>>(bits-1);
       p2=p2>>(bits-1);
       p3=p3>>(bits-1);
       p4=p4>>(bits-1);
       /*Output of feedback filter*/
      d1r[i]=filtr[i]-p1+p2;
      d1i[i]=filti[i]-p3-p4;

       /*Check for overflow*/
       if(d1r[i]>max1)
          d1r[i]=max1;
       if(d1r[i]<-max)
          d1r[i]=-max;
       if(d1i[i]>max1)
          d1i[i]=max1;
       if(d1i[i]<-max)
          d1i[i]=-max;
       }

   d1r[length+1]=d1i[length+1]=0;
    for(i=1;i<=(length+1);i++)
       {
       p1=lr*d1r[i-1];
       p2=li*d1i[i-1];
       p3=lr*d1i[i-1];
       p4=li*d1r[i-1];
```

```
            p1=p1>>(bits-1);
            p2=p2>>(bits-1);
            p3=p3>>(bits-1);
            p4=p4>>(bits-1);

            /*Update pre-filter taps*/
            filtr[i-1]=d1r[i]+p1+p2;
            filti[i-1]=d1i[i]+p3-p4;
            if(filtr[i-1]>max1)
               filtr[i-1]=max1;
            if(filtr[i-1]<-max)
               filtr[i-1]=-max;
            if(filti[i-1]>max1)
               filti[i-1]=max1;
            if(filti[i-1]<-max)
               filti[i-1]=-max;
            }
         start=0;
         }
      while(start<9); /*Nine satring points used*/
}

/*End of the Clark-Hau algorithm, minimum phase impulse*/
/*response of the channel is stored in the vector ymin*/
/*Pass rx signal into pre-filter*/
opr=opi=0;
for(i=0;i<=length;i++)
   {
   p1=sigr[i]*filtr[i];
   p2=sigi[i]*filti[i];
   p3=sigr[i]*filti[i];
   p4=sigi[i]*filtr[i];
   p1=p1>>(bits-1);
   p2=p2>>(bits-1);
   p3=p3>>(bits-1);
   p4=p4>>(bits-1);
   opr=opr+p1-p2;
   opi=opi+p3+p4;

   /*Check for overflow*/
   if(opr>(max-1))
```

```
            opr=max-1;
          if(opr<-max)
            opr=-max;
          if(opi>(max-1))
            opi=max-1;
          if(opi<-max)
            opi=-max;
          }
        /*Output of the prefilter is called opr+jopi*/

        /*Shift signal*/
shift1:  j=length+1 ;
        for(i=j    ;i>=1;i--)
          {
          sigr[i]=sigr[i-1];
          sigi[i]=sigi[i-1];
          }

        /*Allow estimator to settle before starting detection*/
        if(k<1000)
          goto shift2;

        /*At the end of the retraining period reset the stored*/
        /*vectors to their correct values and set costs to large*/
        /*value except one of them*/
        if((k>3000)&&(retrain==1199))
          {
          cost[1]=0;
          for(j=2;j<=m;j++)
            cost[j]=max1;
          for(i=(n-1);i>=0;i--)
            {
            qr[1][i]=datr[n-i];
            qi[1][i]=dati[n-i];
            }
          }

        /*Near maximum likelihood dectector A*/
        mincost=max1;
        for(j=1;j<=m;j++)
          {
```

```
/*Evaluate the cost for m vectors*/
cr=ci=0;
for(i=0;i<=(n-1);i++)
    {
    p1=qr[j][i]*yminr[n-i];
    p2=qi[j][i]*ymini[n-i];
    p3=qr[j][i]*ymini[n-i];
    p4=qi[j][i]*yminr[n-i];
    cr=cr+((p1-p2)>>1);
    ci=ci+((p3+p4)>>1);

    /*Check for overflow*/
    if(cr>(max-1))
        cr=max-1;
    if(cr<-max)
        cr=-max;
    if(ci>(max-1))
        ci=max-1;
    if(ci<-max)
        ci=-max;
    }

    /*Evaluate the total cost for ml vectors*/
    /*by adding the extra cost due to expansion*/
    for(h=1;h<=l;h++)
        {
        vector2=h+l*j-l;
        p1=levelr[h]*yminr[0];
        p2=leveli[h]*ymini[0];
        p3=levelr[h]*ymini[0];
        p4=leveli[h]*yminr[0];
        zr=cr+((p1-p2)>>1);
        zi=ci+((p3+p4)>>1);

        /*Check for overflow*/
        if(zr>(max-1))
            zr=max-1;
        if(zr<-max)
            zr=-max;
        if(zi>(max-1))
            zi=max-1;
```

```
            if(zi<-max)
                zi=-max;
            zr=(zr-opr)>>1;
            zi=(zi-opi)>>1;

            /*Calc. costs of expanded vectors*/
            p1=zr*zr;
            p2=zi*zi;
            /*Scaling*/
            p1=p1>>(bits-1);
            p2=p2>>(bits-1);
            coste[vector2]=cost[j]+p1+p2;
            if(coste[vector2]>max1)
               coste[vector2]=max1;
            /*Find smallest cost*/
            if(coste[vector2]<mincost)
                {
                mincost=coste[vector2];
                q1=j;
                q2=h;
                vector=vector2;
                }
            }
        }
    smallest_cost=mincost;  /*Remember the smallest cost*/

    /*Obtain detected symbol and decode it*/
    sdr=qr[q1][0];
    sdi=qi[q1][0];
    if((sdr==1)&&(sdi==1))
        {
        bit1=1;
        bit2=1;
        }
    else if((sdr==-1)&&(sdi==1))
        {
        bit1=1;
        bit2=0;
        }
    else if((sdr==-1)&&(sdi==-1))
        {
```

```
    bit1=bit2=0;
    }
else
    {
    bit1=0;
    bit2=1;
    }

/*Check for bit errors*/
if((k>3000)&&(retrain<1000))
{
total_bits_sent=total_bits_sent+2;
if(bit1!=tx1[0])
    {
    err1++;
    }
if(bit2!=tx2[0])
    {
    err1++;
    }
}

/*Store vector with smallest cost in array p*/
/*store in the first row*/
for(i=0;i<=(n-1);i++)
    {
    pr[1][i]=qr[q1][i];
    pi[1][i]=qi[q1][i];
    }
pr[1][n]=levelr[q2];
pi[1][n]=leveli[q2];

/*Use early detected data in channel estimator*/
earlyr=levelr[q2];
earlyi=leveli[q2];

pcost[1]=smallest_cost; /*Store cost*/
coste[vector]=max1; /*Discard vector by setting cost to a */
            /*high value*/

/*Discard all costs where first component<>to detected*/
```

```
/*value*/
for(j=1;j<=m;j++)
    {
    if((qr[j][0]!=sdr)||(qi[j][0]!=sdi))
        {
        for(i=1;i<=l;i++)
            coste[j*l-l+i]=max1;
        }
    }

/*Select the next m-1 vectors and store in p*/
count=2;
do
    {
    mincost=coste[1];
    vector=1;
    for(j=2;j<=(m*l);j++)
        {
        if(coste[j]<mincost)
            {
            mincost=coste[j];
            vector=j;
            }
        }
    pcost[count]=mincost;
    coste[vector]=max1;

    /*Transfer vector to array p*/
    q1=tag[vector][1];
    q2=tag[vector][2];
    for(i=0;i<=(n-1);i++)
        {
        pr[count][i]=qr[q1][i];
        pi[count][i]=qi[q1][i];
        }
    pr[count][n]=levelr[q2];
    pi[count][n]=leveli[q2];
    count++;
    }
while(count<=m);
```

```
        /*Replace q by p*/
        for(j=1;j<=m;j++)
            {
            for(i=1;i<=n;i++)  /*Discard first component in q*/
                {
                qr[j][i-1]=pr[j][i];
                qi[j][i-1]=pi[j][i];
                }
            cost[j]=pcost[j]-smallest_cost;     /*Normalize*/
            }

        /*Shift tx symbols*/
shift2:  j=length+1 ;
        for(i=j;i>=1;i--)
            {
            sr[i]=sr[i-1];
            si[i]=si[i-1];
            }

        /*Shift tx data*/
        j=length+n;
        for(i=1;i<=j       ;i++)
            {
            tx1[i-1]=tx1[i];
            tx2[i-1]=tx2[i];
            }
        }
    /*End of transmission*/

    /*Close the channel file*/
    fclose(channel);

    /*Calculate the probability of a bit error*/
    error1=err1;
    pe=error1/total_bits_sent;
    printf("snr=%lf  pe=%lf \n",snr,pe);
    fprintf(results,"%lf,%lf\n",snr,pe);

    snr=snr+1;
    }
    while(snr<=snrfin);  /*Continue to give complete pe curve*/
```

```
fclose(results);
}
```

REFERENCES

1. Pahlavan, K. and Holsinger, J.L., "Voiceband Data Communication Modems - A Historical Review: 1919 - 1988", IEEE Comm. Magazine, Vol. 26, No. 1, pp 16 - 27, January 1988.

2. Clark, A.P., "Principles of Digital Data Transmission", Pentech Press, 1980.

3. Schwartz, M., "Information Transmission, Modulation, and Noise", McGraw Hill Book Company, 1980.

4. Shannon, C.E., "A Mathematical Theory of Communications", Bell Systems, Tech. J., Vol. 27, pp 379 - 423 and pp 565 - 623, 1948.

5. Proakis, J.G., "Digital Communications", McGraw Hill Book Company, 1983.

6. Clark, A.P. "Equalisers for Digital Modems", Pentech Press, 1985.

7. Clark, A.P., Abdullah, S.N. and Ameen, S.Y., "A Comparison of Decision Feedback Equalisers for a 9600 Bit/s Modem", JIERE, Vol. 58, No. 2, pp 74 - 83, March/April 1988.

8. Clark, A.P. and Claydon, M., "Pseudobinary Viterbi Detector", IEE Proc., Vol. 131, Pt. F, No. 2, pp 208 - 218, April 1984.

9. Clark, A.P. and Fairfield, M.J., "Detection Processes for a 9600 Bit/s Modem", IERE, Vol. 51, No. 9, pp 455 - 465, September 1981.

10. Clark, A.P., Abdullah, S.N., Jayasinghe, S.G. and Sun, K.H., "Pseudobinary and Pseudoquarternary Detection Processes for Linearly Distorted Multilevel QAM Signals", IEEE Transactions of Communs. Vol., Com - 33 No. 7, July 1985.

11. Najdi, H.Y., "Digital Data Transmission over Voiceband Channels", Ph.D. Thesis, Loughborough University of Technology, 1982.

12. Clark, A.P., "Near Maximum Likelihood Detection Processes", Jordan International Electrical and Electronic Eng. Conf., Amman, Jordan, April 28 - May 1, 1985.

13. Viterbi, A.J., "Convolutional Codes and Their Performance in Communication Systems", IEEE Trans. Commun., COM - 19, pp 751 - 772, 1971.

14. Harvey, J.D., "Adaptive Detection of Digital Suppressed Carrier A.M. Signals", Ph.D. Thesis, Loughborough University of Technology, 1979.

15. Clark, A.P. and McVerry, F., "Performance of 2400 Bit/s Serial and Parallel Modems over an H.F. Channel Simulator", IERE Conf. on Digitl Processing of Signals in Communications, Proc. No. 49, pp 167 - 179, Loughborough University of Technology, 1981.

16. Abdullah, S.N., "Data Transmission at 9600 Bit/sec over an H.F. Radio Link", Ph.D. Thesis, Loughborough University of Technology, 1986.

17. Bateman, S.C., "Data Transmission at 19200 Bit/s over Telephone Channels", Ph.D. Thesis, Loughborough University of Technology, 1985.

18. Forney, G.D., Jr., "The Viterbi Algorithm", IEEE Proc., Vol. 61, pp 268 - 278, March 1973.

19. Clark, A.P. and Abdullah, S.N., "Near - Maximum Likelihood Detectors for Voiceband Channels", IEE Proc., Vol. 134, Pt. F, No. 3, pp 217 - 226, June 1987.

20. Aftelak, S.B. and Clark, A.P., "Adaptive Reduced-State Viterbi Algorithm Detector", J.IERE, Vol. 56, No. 5, pp 197 - 206, May 1986.

21. Clark, A.P. and Najdi, H.Y., "Detection Process of a 9600 Bit/s Serial Modem for H.F. Radio Links", IEE Proc., Vol. 130, No. 5, pp 368 - 376, August 1983.

22. Clark, A.P., Najdi, H.Y. and McVerry, F., "Performance of a 9600 Bit/s Serial Modem over a Model of an H.F. Radio Link", IEE Conf. Publication No. 224, Radio Spectrum Conservation Techniques", pp 151 - 155, September 1983.

23. Alhahim, S.S. and Abdullah, S.N., "Flexible Pseudobinary and Pseudoquarternary Detectors for 9600 Bits/s Modem", IEE Proc., Vol. 137, Pt. I, No. 2, April 1990.

24. Clark, A.P. and Ashgar, S.M., "Detection of Digital Signals Transmitted over a Known Time - Varying Channel", IEE Proc., Vol. 128, Pt. F, No. 3, June 1981.

25. Fagan, A.D. and O'Keane, F.D., "Performance Comparison of Detection Methods Derived form Maximum Likelihood Sequence Estimation", IEE Proc., Vol. 133, Pt. F, No. 6, pp 535 - 542, October 1986.

26. Clark, A.P., "Advanced Data Transmission Techniques", Pentech Press, 1977.

27. Hau, S.F., "Adaptive Adjustment of Receiver for Distorted Digital Signals", Ph.D. Thesis, Loughborough University of Technology, 1986.

28. Beare, C.T., "The Choice of the Desired Impulse Response in Combined Linear Viterbi Algorithm Equaliser", IEEE Trans. on Commun., Vol. COM - 26, No. 8, pp 1301 - 1307, August 1978.

29. Clark, A.P. and Hau, S.F., "Adaptive Adjustment of Receiver for Distorted Signals", IEE Proc., Pt. F, Vol. 131, pp 526 - 536, August 1984.

30. Cadzow, J.A., "Foundations of Digital Signal Processing and Data Analysis", MacMillan Book Company, 1987.

31. Lucky, R.W., Salz, J. and Weldon, E.J., Jr., "Principles of Data Communication", McGraw Hill Book Company, 1968.

32. Sharpe, J.T.L., "Techniques for High Speed Data Transmission over Voiceband Channels", Electrical Communication, Vol. 46, No. 1, pp 24 - 31, 1971.

33. Lathi, B.P., "Modern Digital and Analogue Communication Systems", Holt, Reinhart and Winston, 1983.

34. Haykin, S., "Communication Systems", John Wiley and Sons Inc. 1978.

35. Mohanty, N., "Signal Processing", Van Nostrand Reinhold Company, 1987.

36. Perl, J.M., Bar, A. and Cohen, J., "TMS - 320 Implementation of a 2400 bps V.26 Modem", Signal Processing: Theories and Applications (EURASIP 1986, Conf., Netherlands Congress Centre, The Hague, Sept. 2 - 5, 1986), Elesevier Science Publishers B.V. 1986.

37. Corden, I.R. and Carrasco, R.A., "An Efficient DSP QPSK Modem Algorithm Extended to a Parallel Structure", Int. J. Elect. Eng. Educ., Vol. 27, pp 119 - 131, Manchester Uni. Publishers, 1990.

38. Allen, R.D., Bramwell, J.R., Saunders, D.A. and Tomlinson, M., "A High Performance Satellite Data Modem Using Real Time Signal Processing Techniques", J.IERE, Vol. 58, No. 3, May 1988.

39. Harmer, D. and Hillam, B., "Digital Implementation of High Speed H.F. Modems", Digital Processing of Signals in Communications, IERE Conf., Proc. No. 62, pp 139 - 155, April 1985.

40. Lu, H.H., Hedberg, D. and Fraenkel, B., "Implementation of High Speed Voiceband Data Modems using the TMS320C25", Proc. IEEE IC ASSP - 87 4, 44.6.1 - 14 1987.

41. Van Gerwen, P.J., Verhoeckx, N.A.M., Van Essen, H.A. and Snijders, F.A.M., "Microprocessor Implementation of High Speed Data Modems", IEEE Trans. on Commun. COMM - 25, No. 2, pp 238 - 250, 1977.

42. Warburton, K. and Carrasco, R.A., "Design and Implementation of a Simple PSK/DPSK Modem", Int. J. Elec. Enging. Educ. 24, pp 215 - 225, 1987.

43. Tomlinson, M. and Bramwell, J.R., "Real Time Digital Signal Processing in Satellite Modems", Conf. on Commun., No. 262, Plymouth Polytech., England, pp 89 - 93, May 1986.

44. Hodgkiss, W., "DSP Modem Techniques applied to Digital Radio", IEEE Collqm. Digitally Implemented Radios, Digest No. 40, 2/1 - 5, 1987.

45. Westall, F.A., "Efficient Digital Signal Processing Realisations for PSK Modem Receivers", Conf. on Digital Processing of Signals in Communications, Loughborough University of Technology, April 7 - 10, 1981.

46. Openheim, A.V. and Shafer, R.W., "Digital Signal Processing", Prentice Hall, 1975.

47. Lynn, P.A., "An Introduction to the Analysis and Processing of Signals", MacMillan, 1982.

48. "Theory and Implementation of a Splitband Modem using the TMS32010", Texas Instruments, U.S.A., 1986.

49. Horvath, S., "Digital Signal Processing in Communication and Transmission", Signal Processing III, Theories and Applications, Proc. EUSIPCO - 83, Sept. 12 - 16, 1983.

50. Stahl, J., Meyr, H. and Oerder, M. "Implementation of a High Speed Viterbi Decoder", Signal Processing III: Theories and Applications, EURASIP 1986.

51. Linsey, W.C. and Simon, M.K., "Phase Locked Loops and Their Application", IEEE Press, 1978.

52. "Carrier Synchronisation and Detection of Polyphase Signals", IEEE Trans. on Commun., pp 441 - 454, June 1972.

53. Kalayil, B., "Timing Phase Synchronisation of a QPSK Modem in the Presence of AWGN and Doppler Shift", Internal Report, Dept. Electronic and Electrical Eng., Loughborough University of Technology, 1990.

54. "TMS320C25 Users Guide", Texas Instruments.

55. Kadhim, A - K, A - R, "Detection of Coded and Distorted QAM Signals", Ph.D. Thesis, Loughborough University of Technology, 1989.

56. Clark, A.P., Kwong, C.P. and McVerry, F., "Estimation of the Sampled Impulse Response of a Channel", Signal Processing, Vol. 2, No. 1, pp 39 - 53, January 1980.

57. Clark, A.P. and McVerry, F., "Channel Estimation for an H.F. Radio Link", IEE Proc., Vol. 128, Pt. F, No. 1, pp 33 - 42, February 1981.

58. Rush, C.M., "H.F. Propagation: What we know and what we need to know", Second Int. Conf. on Antennas and Prop., Proc. No. 195, Pt. 2, pp 229 - 234, April 13 - 16, 1981.

59. Jordan, E.C. and Balmain, K.G., "Elecromagnetic Waves and Radiating Systems", Prentice-Hall, 1968.

60. Williams, H.P., "H.F. Communications at Frequencies above the Predicted M.U.F.", IEE Conf. on Antennas and Prop., Proc. No. 195, Pt. 2, pp 245 - 248, April 1981.

61. Darnell, M., "The Characteristics of H.F. Propagation and their Implications in Digital Communication System Design", IEE Conf. on Antennas and Prop., Proc. No. 195, Pt. 2, pp 254 - 257, April 1981.

62. Hodgkiss, W., Turner, L.F. and Pennington, J., "Serial Data Transmission over H.F. Radio Links", IEE Proc., Vol. 131, Pt. F, No. 2, pp 107 - 116, April 1984.

63. Dai, Y.S. and Zhu, Z.H., "Statistical Characteristics of a Channel Parameter for H.F. Communication Circuit", Sixth Int. Conf. on Antennas and Prop. ICAP 89, Proc. No. 301, pp 63 - 67, April 1989.

64. CCIR: "H.F. Ionospheric Channel Simulator", XIII Plenary Assembly CCIR, Vol. III, Rep. No. 549, Geneva, 1974.

65. Watterson, C.C., Juroshek, J.R. and Bensema, W.D., "Experimental Confirmation of an H.F. Channel Model", IEEE Trans. Commun. Technol., COM - 18, pp 792 - 803, 1970.

66. Ralphs, J.D. and Sladen, F.M.E., "An H.F. Channel Simulator using a new Raleigh fading Method", The Radio and Electronic Engineer, Vol. 46, pp 579 - 587, 1976.

67. Openheim, A.V. and Shafer, R.W., "Digital Signal Processing", Prentice Hall, 1975.

68. Hari, H., "H.F. Channel Estimators", PhD Thesis, Loughborough University of Technology, 1989.

69. Linsey, I.A.B. and Dripps, J.H., "A Novel Phase-Locked Loop for PSK Carrier Recovery", IERE Conf. on Digital Processing of Signals in Communications, Proc. No. 49, pp 173 - 177, Loughborough University of Technology, 1981.

70. Kuo, F.F., "Network Analysis and Synthesis", Wiley, 1966.

71. Clark, A.P., "Modelling of Digital Communication Systems", International Seminar on Mathematical Modelling and Applications, Arab School of Science and Technology, Damascus, Syria, April 1986.

72. Bateman, S.C. and Ameen, S.Y., "Comparison of Algorithms of use in Adaptive Adjustment of Digital Data Receiver", IEE Proc., Vol. 137, Pt. 1, No. 2, April 1990.