# Communications Research Centre

# A BLOCK DEMODULATOR FOR FREQUENCY DIVISION MULTIPLEXED NCFSK SIGNALS

by

M.D. Shaw, J.L. Pearce and J.S. Wight

This document was prepared for and is the property of the Department of National Defence, Research and Development Branch under Project No. 041LG14.



Government of Canada
Department of Communications

Gouvernement du Canada
Ministère des Communications

CRC REPORT NO. 1427
OTTAWA, JULY 1987

Canada

COMMUNICATIONS RESEARCH CENTRE

DEPARTMENT OF COMMUNICATIONS
CANADA

# A BLOCK DEMODULATOR FOR FREQUENCY DIVISION MULTIPLEXED NCFSK SIGNALS

by

M.D. Shaw, J.L. Pearce and J.S. Wight

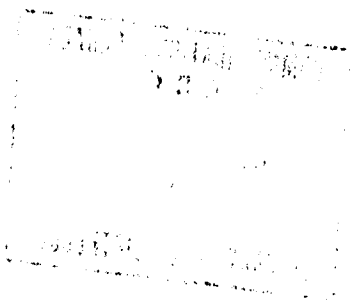
*(Communications Technologies Research Branch)*



**CRC REPORT NO. 1427**

July 1987
OTTAWA

This document was prepared for and is the property of the Department of National Defence, Research and Development Branch under Project No.041LG14.



ABSTRACT

A block demodulator for the noncoherent demodulation of M-ary FSK signals multiplexed in frequency has been built and tested. The various users' signals are separated and demodulated with the use of the discrete Fourier transform. The demodulator is capable of handling 16/M users. The bit error performance in the presence of system noise has been measured for M = 2-, 4-, 8-, and 16-ary FSK with a symbol period of 50 $\mu$s. The measured implementation loss depends on the specific channel location within the demodulator's bandwidth, varying from 0.2 dB for the best channel allocations to 2.6 dB for the worst. The effects of frequency offsets and adjacent channel interference are characterized with both a rectangular and a Kaiser-Bessel ($\alpha$ = 1.4) window. For small frequency errors the rectangular window still performs well, but for larger offsets or large differences in power levels between the users the Kaiser-Bessel window is better.

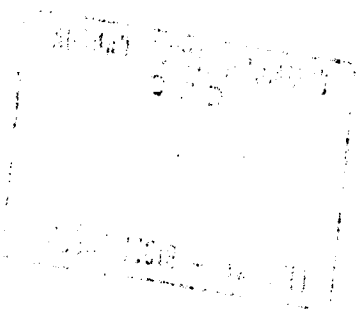


TABLE OF CONTENTS

# GLOSSARY OF TERMS

$a_m$ transmitted data
A A/D input scaling factor
b digital register length excluding sign bit
$e_k$ cumulative truncation error at $k^{th}$ DFT output
e(n) quantization or truncation error sequence
E{•} expected value of
$E_s$ transmitted symbol energy
$E_{sr}$ received symbol energy
ENBW equivalent noise bandwidth
$f_c$ center frequency of IF signal
$f_k$ center frequency of the $k^{th}$ DFT output
$f_m$ FSK tone frequencies
$f_s$ sampling rate
F{•} Fourier transform
g(t) baseband pulse shape
G(f) Fourier transform of the baseband pulse shape
$I_o${•} modified zeroth order Bessel function of the first kind
L number of frequency hops per transmitted symbol
$m_e$ mean value of the truncation error
M number of unique FSK symbols for a single user
N number of samples per symbol
$N_o$ noise spectral density
$P_b$ bit error probability
$P_s$ symbol error probability
PL window processing loss
r code rate of error correction encoder
r(t) complex envelop of the received signal
r(n) sampled version of the complex envelop of the received signal
$R(t_1,t_2)$, $R(\tau)$, R(n) autocorrelation function
$R_b$ bit rate
$R_c$ coded bit rate
$R_h$ hopping rate
$R_s$ symbol rate
R(k) discrete Fourier transform of r(n)
$s_m(t)$ transmitted symbol
S(f) power spectral density
SNR signal-to-noise ratio
T sampling period
$T_b$ bit period
$T_c$ coded bit period
$T_h$ hopping period
$T_s$ symbol period

$u_m(t)$ complex envelop of the transmitted symbol
$u(n)$ unit step sequence
$U_m$ decision variables
$v(t)$ received IF signal
$w(t)$ continuous-time window function
$w(n)$ discrete-time window function
$W(f)$ frequency response of the window sequence $w(n)$
$z(t)$ complex envelop of the received noise process
$\alpha$ channel attenuation
$\delta(n)$ unit impulse sequence
$\Delta f$ half of the FSK tone spacing
$\zeta(n)$ discrete-time noise process
$\Theta$ random phase angle
$\sigma^2$ variance

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 INTRODUCTION

Block demodulation, in which several signals are demodulated by a common processor is a requirement for some multiple access communication systems. One application where block demodulation is needed is for satellite systems which use spread spectrum communications and processing onboard the satellite to provide protection against interference. These systems would use frequency shift keying (FSK) for data modulation and the various users' signals are multiplexed in frequency. In this report, a block demodulator designed to provide demodulation for several FSK signals multiplexed in frequency is described. The demodulator makes use of digital signal processing techniques and is implemented with commercial microprocessors and digital electronics.

Frequency-hopping spread spectrum is often used to provide interference rejection capabilities for data transmision systems [1]-[3]. It is currently being studied for use in satellite communication systems which may be required to maintain reliable communications even under adverse conditions with intentional interference present. For such a situation, rather than having the satellite simply act as a repeater and hence retransmit the interference as well as the desired signals, it is desirable for the satellite to recover the transmitted data onboard and retransmit a "clean" signal. The system should be capable of handling several signals at once, and to keep the size, weight and power consumption to a minimum, block demodulation is necessary.

Block demodulation generally makes use of Fourier transform techniques to separate signals multiplexed in frequency. Also, for the noncoherent demodulation of FSK signals, the Fourier transform is equivalent to the optimum matched filter and therefore the various signals can be both separated and demodulated with one Fourier transforming block. This requires that the symbol timing of each of the received FSK signals be synchronized with each other. As well, frequency synchronization between each of the users and the receiver is required since the receiver cannot separately synchronize itself with each of the individual users.

The Fourier transform can be performed with a variety of technologies. One implementation of a block demodulator makes use of a surface acoustic wave (SAW) device for the Fourier transforming operation. Charge coupled devices have also been considered for the implementation of a block demodulator. Use of the discrete Fourier transform has not previously been seriously considered due to the limited processing speeds of digital electronics of the day, but with current digital technology, digital signal processing and the discrete Fourier transform should now allow the implementation of a digital block demodulator which can handle a reasonable

number of users with data symbol rates on the order of tens of kilosymbols per second.

## 1.2 PROJECT OBJECTIVES

For the onboard-the-satellite processing of frequency division multiplexed signals, block demodulation is required. Digital block demodulation which uses the discrete Fourier transform is studied in this project with the intention of demonstrating that the current digital technology is capable of the processing speeds required for onboard processing of frequency-hopping signals. Thus a digital block demodulator for FSK signals is to be demonstrated, and the performance characterized. Because in a real system, maintaining perfect frequency synchronization between the various users would be very difficult, the effect of frequency misalignments between the users and the receiver will be considered. Nonuniform receiver windowing can be used to lessen the effect of frequency errors and interference, and thus the choice of a receiver window is to be addressed. As well, the demodulator design should address the effects of quantization and truncation inherent in the digital processing.

## 1.3 OUTLINE

Chapter 2 begins with an introduction of spread spectrum techniques and applications, with an emphasis on frequency-hopping systems and their interference rejection capabilities. A typical frequency-hopping system for satellite communications which employs onboard processing is then discussed in more detail to show the need for block demodulation. Following this the report focuses on noncoherent demodulation of FSK signals, and an analysis of the discrete Fourier transform-based demodulator is presented to show that the performance is equivalent to that of the optimum analog matched filter demodulator. Receiver windowing is then introduced and its effects characterized.

In chapter 3, the design and implementation of a digital block demodulator is presented. The implications of the sampling rate are discussed. The algorithm for calculating the DFT is analysed and simulation of the effect of both windowing and the finite register length of the microprocessor on the frequency response of the DFT is presented for several window functions.

In chapter 4, the performance measurements for the block demodulator implementation are presented. The performance with two different windows is measured in the presence of system noise, and frequency errors. The effect of co-channel and adjacent channel interference is measured.

The results of the measurements are summarized in chapter 5. A comparison with a SAW-based demodulator previously tested [4] is then given, followed by suggestions for further studies related to the work presented in this report.

# CHAPTER 2

# FREQUENCY-HOPPING SYSTEM CONCEPTS

## 2.1 INTRODUCTION

In this chapter, spread spectrum systems are discussed with an emphasis on techniques for the reception of frequency-hopping spread spectrum signals. The material presented is meant to give a theoretical base for the block demodulator discussed in chapter 3, designed to be part of an onboard-the-satellite signal processing system for spread spectrum satellite communications.

## 2.2 SPREAD SPECTRUM TECHNIQUES

In spread spectrum communications, the transmitted signal bandwidth is made much greater than the minimum required information bandwidth by modulating a conventional information-carrying narrowband signal with a broadband encoding or spreading waveform as shown in Figure 2.1. The receiver multiplies the incoming signal with a replica of the spreading waveform, which collapses the transmitted signal back into its original bandwidth and allows conventional demodulation to recover the transmitted information. Spread spectrum is used in applications which may include interference rejection, secure communications, covert communications, or multiple-access systems.

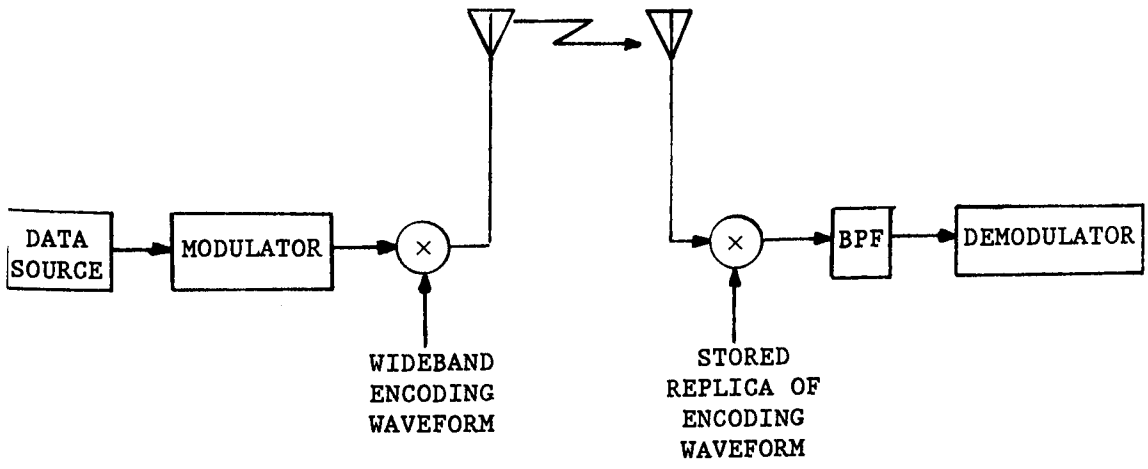



Figure 2.1 : A general spread spectrum transmission system.

The various forms of spread spectrum are distinguished by the characteristics of the encoding waveform used for spectrum spreading. The two most common spread spectrum techniques are frequency-hopping and direct sequence.

In a frequency-hopping system, the spreading waveform is generated by a frequency agile oscillator whose output rapidly switches frequencies in a known pattern. Figure 2.2 illustrates the spectrum use over time with a frequency-hopping system. Effectively, the total transmission bandwidth is divided into several channels, each with a bandwidth equal to the bandwidth of the signal before spreading. The transmitted signal occupies only one of these channels at any given time and "hops" from channel to channel in the pattern of the agile oscillator.

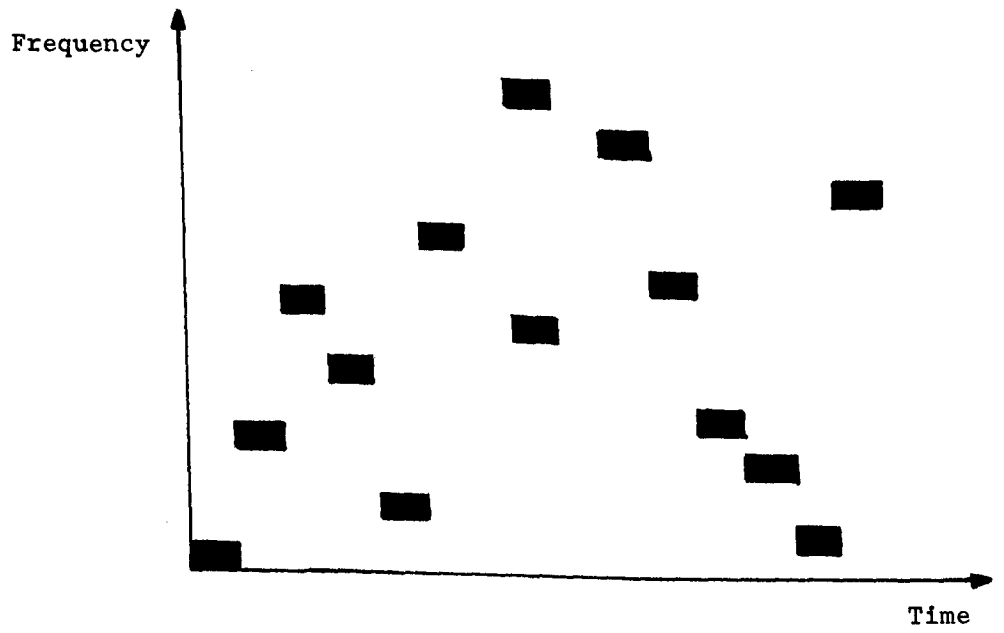



Figure 2.2 : Spectrum use over time for a frequency-hopping system.

With a direct sequence system, the spectrum spreading is achieved by modulating the information-carrying signal with a pseudo-noise (PN) sequence whose bit rate is much greater than the data symbol rate. The transmitted bandwidth is then determined by the bandwidth of the PN sequence. A PN sequence is a deterministic binary sequence which has properties very similar to those of a random binary sequence. Thus the sequence is apparently random, but may be duplicated by others who know the key (the method used to generate the sequence).

The subject of this report is a demodulator suitable for a frequency-hopping spread spectrum receiver. Before discussing the details of frequency-hopping systems, it is worthwhile to note the motivation for using frequency-hopping in digital communication systems.

## 2.3 FREQUENCY-HOPPING APPLICATIONS

Frequency-hopping spread spectrum systems were initially used in military applications to provide protection against interference produced by an enemy trying to disrupt radio communications. As well, frequency-hopping may be used to provide covert communications which avoids enemy interference by making it difficult for the enemy to detect the transmission. When data security is required, frequency-hopping may be used to provide some degree of message privacy. The multiple access capabilities of spread spectrum have prompted the consideration of frequency-hopping systems for civilian applications of mobile radio and satellite communications.

### 2.3.1 Interference Rejection

Consider the environment depicted in Figure 2.3 in which an enemy (jammer) intercepts the transmitted signal and attempts to interfere with the information recovery at the receiver. If the communicator is using conventional narrowband digital communications, the jammer may then be able to mimic the communicator's signal and confuse the receiver. This can be done if the jammer has knowledge of the channel frequency and bandwidth and the type of modulation being used by the communicator. Even with knowledge only of the channel frequency and bandwidth the jammer can severely disrupt a narrowband transmission.

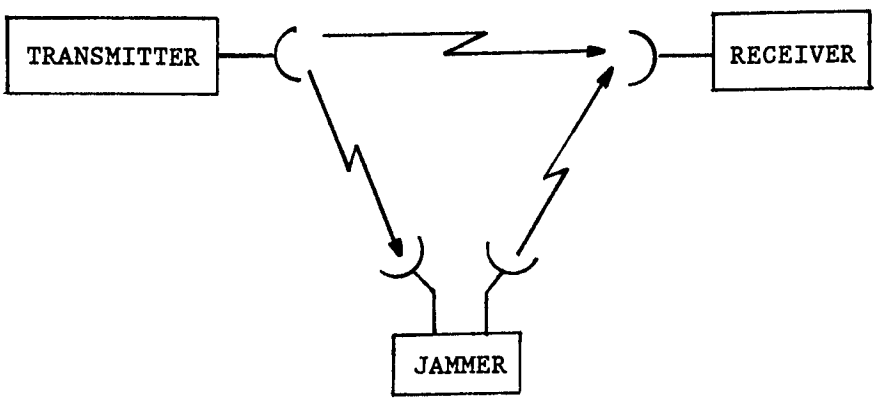



Figure 2.3 : A hostile communications environment.

Knowing this, the communicator may resort to frequency-hopping techniques in which the carrier frequency of the transmitted signal is rapidly changed over a wide bandwidth in an apparently random, or pseudo-random pattern known only to the transmitter and intended receiver. A narrowband jamming signal will only affect the recovered information during the instants in which the transmitted signal hops into one of the few channels occupied by the jammer. As long as the hopping bandwidth is made large enough then the percentage of time for which the transmitter occupies a jammed channel will be low, and if there is suitable encoding or redundancy introduced into the data transmission to correct errors made when they do occupy the same channel, the integrity of the recovered data can be maintained. This forces the jammer to spread his signal energy over a wider bandwidth to increase the probability that at least some of his energy is interfering with the data transmission. Assuming the jammer has an average power constraint on his jamming signal, as the signal is spread to cover more channels, the jamming energy in each of those channels will be less, so that although the percentage of time for which the jammer interferes increases, the effect of that interference is less. Thus introducing an element of apparent randomness into the method of data transmission results in a lower susceptibility to interference.

### 2.3.2 Covert Communications

A system designed to make the detection of the transmitted signal difficult for an enemy makes use of low probability of intercept (LPI) signals. Because the transmission rapidly hops from channel to channel in a frequency-hopping system, over time the signal energy is spread over a very large bandwidth and the power level will be low in any given narrow bandwidth, making detection less likely. Therefore frequency-hopping spread spectrum may in some situations be suitable for covert communications. There is a conflict in trying to design a system which has both jamming rejection capabilities and LPI properties, because high transmission power is desired for the former and low power is required for the latter. Thus systems are designed to provide one capability or the other, but not usually both at the same time.

### 2.3.3 Secure Communications

Spread spectrum systems have an inherent degree of data security as a result of the pseudo-random spreading waveform. An eavesdropper must have knowledge of the code sequence used to generate the spreading waveform in order to intercept the transmitted data. If a sophisticated eavesdropper has the capability of determining the pseudo-random code sequence by observing the communicator's signal for a suitable length of time, the communicator and intended receiver may agree to change the code pattern at set intervals. As well, they may use codes which are not periodic, making it very difficult for the eavesdropper to predict the sequence without knowledge of how the code is produced. The transmission is truly secure to the extent that the code sequence is secure.

### 2.3.4 Multiple Access Systems

With a frequency-hopping multiple access system, many users may share the same entire frequency band at the same time because the encoding provided by the spreading process allows the desired signal to be distinguished from all other users and other interference. The advantage this type of system has over the conventional forms of multiple access such as frequency division multiple access (FDMA) and time division multiple access (TDMA) is the inherent frequency diversity of a frequency-hopping system. This is desirable in a selective frequency-fading environment typical of mobile radio channels.

The system considered in this project uses frequency-hopping to provide protection against jamming for satellite communications. Rather than acting as a simple repeater, the satellites will have onboard processing to recover the transmitted data and remodulate the recovered data for the downlink transmission. This avoids retransmitting any jamming signals and allows multiple-users' signals to be separated and routed to different downlink antenna beams. Thus the frequency-hopping systems further discussed in this report are suitable for such satellite communications with onboard processing.

## 2.4 FREQUENCY-HOPPING SYSTEMS

Frequency-hopping systems may be distinguished as either fast hopping or slow hopping systems, depending on the rate at which the carrier frequency hops. Fast hopping is the case when there is one or more frequency hop for each transmitted data symbol and slow hopping is the case when two or more data symbols are transmitted per frequency hop. In a hostile environment, fast hopping is used to avoid follower jamming. A follower jammer intercepts the transmitted signal, determines which channel is currently in use and jams that particular channel. To be effective, the jamming energy must reach the receiver before the transmission hops to a new channel. A fast hopping system is studied in this project. Because it would be difficult to maintain phase references at the receiver for each of the possible channels in a fast frequency-hopping system, noncoherent demodulation is generally used. Typically, noncoherent M-ary frequency shift keying (NCFSK) is used.

Figure 2.4 shows a block diagram of a frequency-hopped transmitter in which M-ary FSK is used for data modulation. The binary data from the information source have a period $T_b$ s and a rate $R_b$ bits/s. Error-correction (EC) encoding may be used which produces data at a rate $R_c = R_b/r$ bits/s, where r is the code-rate of the EC encoder. The encoder output is converted to M-ary symbols which determine which one of M tones is generated by the M-ary FSK modulator. The carrier frequency is hopped at a rate $R_h = LR_s$, where L is an integer greater than or equal to one for a fast hopping system. When L is greater than one, there is time diversity built into the system because each symbol is repeated over L different hops.

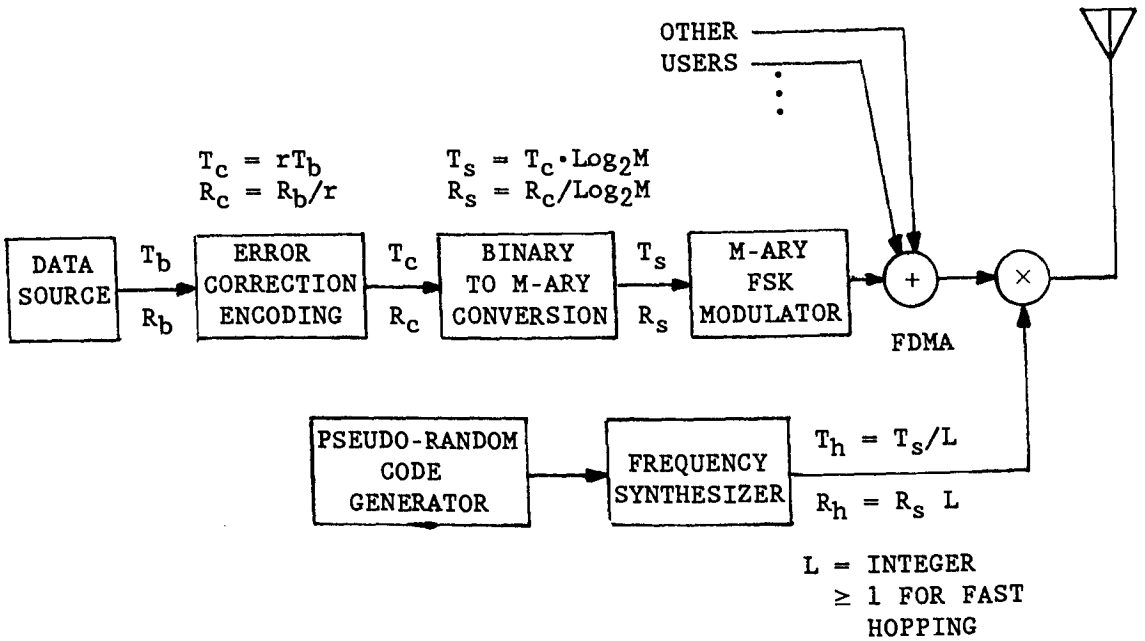



Figure 2.4 : A frequency-hopping transmitter with M-ary FSK modulation.

In the system shown, different users are separated by conventional FDMA before the carrier frequency is hopped. For practical reasons, this would be the case for a satellite uplink when the satellite employs onboard processing because each user then has the same hopping pattern. In such a case there is only one dehopping synthesizer needed on the satellite, whereas if frequency-hopping multiple access is used, each user has a different hopping pattern and the satellite would require a dehopping synthesizer for each user.

At the receiver, shown in Figure 2.5, the signal is dehopped using a locally generated hopping waveform. The various users, multiplexed in frequency, will each have a noncoherent FSK demodulator and decoding to recover the transmitted data. Furthermore, diversity combining is used for the case of $L > 1$. The remainder of this chapter deals with the processing of the dehopped waveform to produce demodulated data for each of the users.

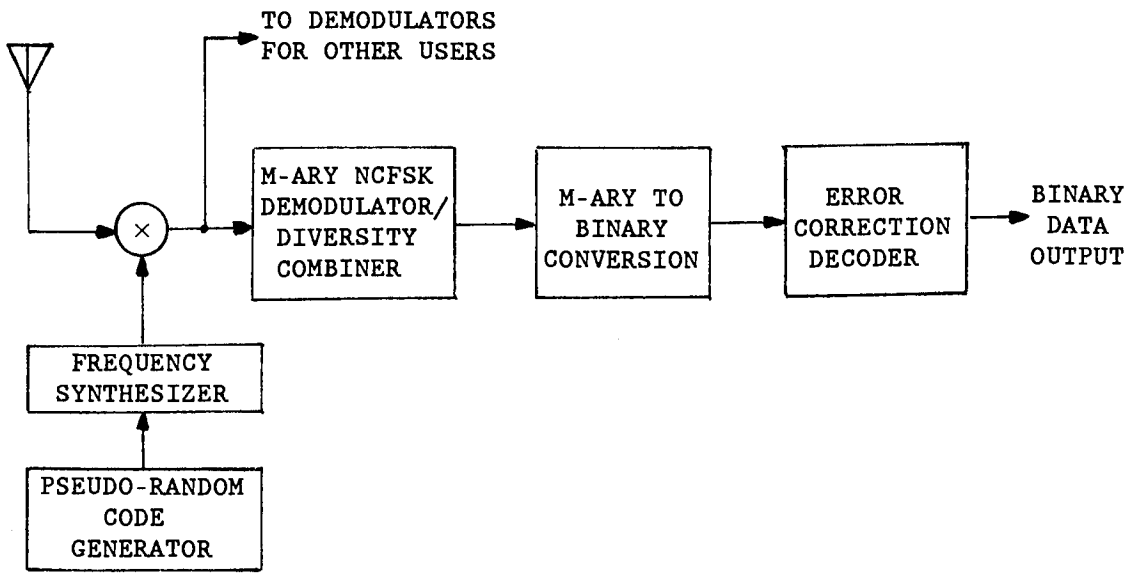



Figure 2.5 : A receiver for a frequency-hopping system with M-ary FSK.

## 2.5 DEMODULATION OF DEHOPPED MULTIPLE-USER M-ARY NCFSK SIGNALS

For each symbol period, the transmitted FSK signal before hopping will have the form

$$s_m(t) = \sqrt{2E_s/T_s}\cos[2\pi(f_c + a_m\Delta f)t] \qquad (2.5\text{-}1)$$

where $a_m$ is data dependent and takes on the values $\pm 1, \pm 3, \cdots, \pm(M-1)$ determined by

$$a_m = 2m - M - 1, \qquad (2.5\text{-}2)$$

$E_s$ is the transmitted symbol energy, $T_s$ is the symbol period, and $2\Delta f$ is the frequency spacing of the FSK tones. For orthogonal NCFSK this spacing is an integer multiple of $1/T_s$. The signal $s_m(t)$ can be expressed as

$$s_m(t) = \mathrm{Re}[u_m(t)\exp(j2\pi f_c t)] \qquad (2.5\text{-}3)$$

where $u_m(t)$ is the complex envelop and is given by

$$u_m(t) = \sqrt{2E_s/T_s}\exp(j2\pi a_m \Delta f t). \qquad (2.5\text{-}4)$$

The symbol energy is given by

$$E_s = \int_0^{T_s} s_m^2(t)\, dt$$

$$= 1/2 \int_0^{T_s} |u_m(t)|^2\, dt. \qquad (2.5\text{-}5)$$

The power spectral density of the NCFSK signal before hopping is of interest and is derived by first finding the autocorrelation function. The long term transmitted signal is

$$s(t) = \sum_{n=-\infty}^{\infty} g(t-nT_s)\mathrm{Re}\{u_{mn}(t-nT_s)\exp(j2\pi f_c t + j\theta_n)\} \qquad (2.5\text{-}6)$$

where g(t) is given by

$$g(t) = \begin{cases} 1, & 0 \le t \le T_s \\ 0, & \text{otherwise} \end{cases} \qquad (2.5\text{-}7)$$

and $\theta_n$ is a random phase uniformly distributed over $(0,2\pi)$ and independent from symbol to symbol. The autocorrelation function is [5]

$$R_s(t,t+\tau) = E\{s(t)s(t+\tau)\}$$

$$= \frac{E_s}{MT_s} \sum_{m=1}^{M} \cos\{2\pi[(M-2m+1)\Delta f + f_c]\tau\} \sum_{n=-\infty}^{\infty} g(t-nT_s)g(t+\tau-nT_s) \qquad (2.5\text{-}8)$$

where $E\{\cdot\}$ denotes expectation. Since $R_s(t,t+\tau)$ is a function of t and in fact is periodic in t with period $T_s$, s(t) is a cyclostationary process and the autocorrelation function must be averaged over one period before proceeding to find the power spectral density of s(t). The average autocorrelation function is

$$R_s(\tau) = \frac{1}{T_s} \int_0^{T_s} R_s(t,t+\tau)\, dt$$

$$= \frac{E_s}{MT_s^2} \sum_{m=1}^{M} g(\tau)*g(-\tau)\cos\{2\pi[(M-2m+1)\Delta f + f_c]\tau\} \qquad (2.5\text{-}9)$$

where * denotes the convolution operation. The power spectral density is determined by the Fourier transform of equation (2.5-9), the result being

$$S_s(f) = \frac{E_s}{2MT_s^2} \sum_{m=1}^{M} |G[f - (M-2i+1)\Delta f - f_c]|^2 + |G[f + (M-2i+1)\Delta f + f_c]|^2 \qquad (2.5\text{-}10)$$

where G(f) is the Fourier transform of g(t). From the definition of g(t) in equation (2.5-7), $|G(f)|^2$ is

$$|G(f)|^2 = T_s^2 \text{sinc}^2(T_s f) \ , \qquad (2.5\text{-}11)$$

which is shown in Figure 2.6. Thus the power spectral density of an NCFSK signal is the superposition of the squared magnitude of the spectrum of the baseband pulse shape centered on the FSK tone frequencies.

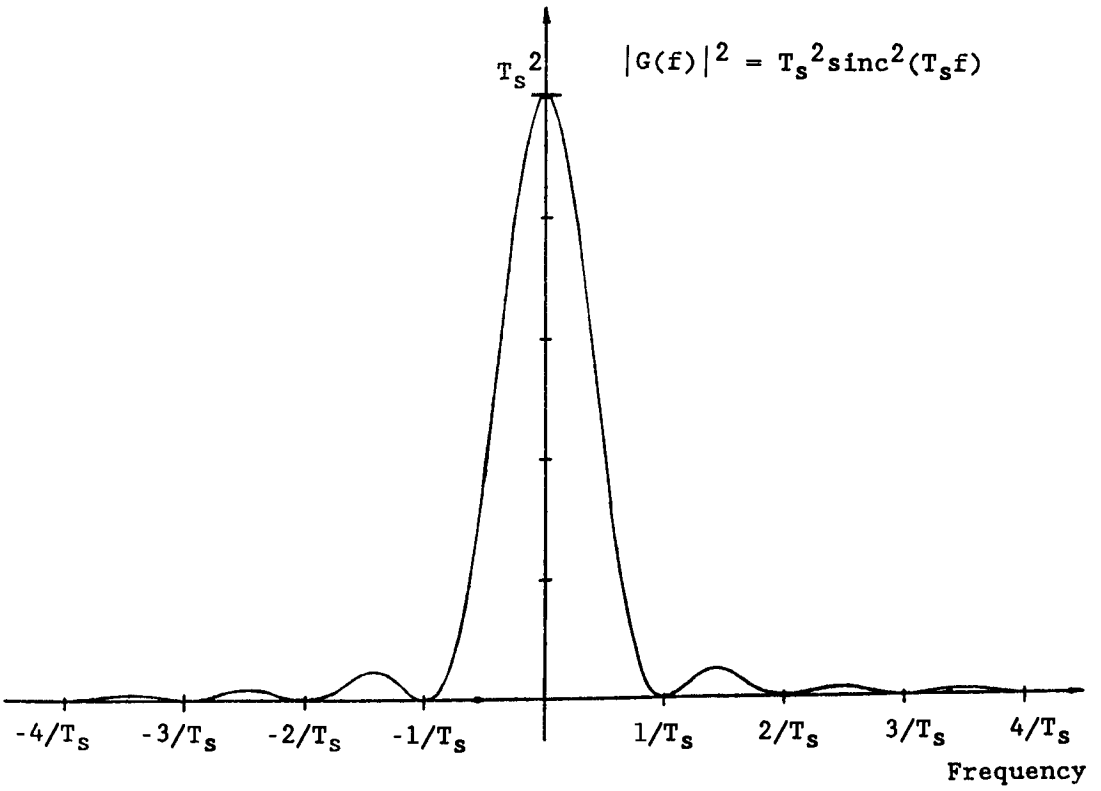



Figure 2.6 : The squared magnitude of the Fourier transform of the rectangular baseband pulse g(t).

When the received signal is simply a uniformly delayed and attenuated version of the transmitted signal corrupted only by additive Gaussian noise, an optimum processing algorithm can be derived which minimizes the

probability that the recovered data is in error. The form of this optimum processing is discussed in the next section.

### 2.5.1 The Optimum Receiver

When there is no time diversity (i.e. $L = 1$ and $R_h = R_s$) and the only degradation is additive Gaussian noise, over a single symbol period the received signal after dehopping will be

$$v(t) = \mathrm{Re}\{[u_m(t)\alpha\exp(j\theta) + z(t)]\exp(j2\pi f_c t)\}$$

$$= \mathrm{Re}\{r(t)\exp(j2\pi f_c t)\} \qquad (2.5\text{-}12)$$

where $\alpha$ is the channel attenuation, $\theta$ is a random phase uniformly distributed over $(0,2\pi)$ resulting from the noncoherent hopping and dehopping, and $z(t)$ represents the equivalent lowpass additive noise with zero mean and variance $N_o$. The optimum receiver [6] computes the M decision variables

$$U_m = \left|\int_0^{T_s} r(t)u_m^*(t)\,dt\right| \qquad (2.5\text{-}13)$$

or equivalently,

$$U_m = \left|\int_0^{T_s} v(t)u_m^*(t)\exp(-j2\pi f_c t)\,dt\right|$$

$$= \left|\int_0^{T_s} v(t)\sqrt{2E_s/T_s}\exp(-j2\pi f_m t)\,dt\right| \qquad (2.5\text{-}14)$$

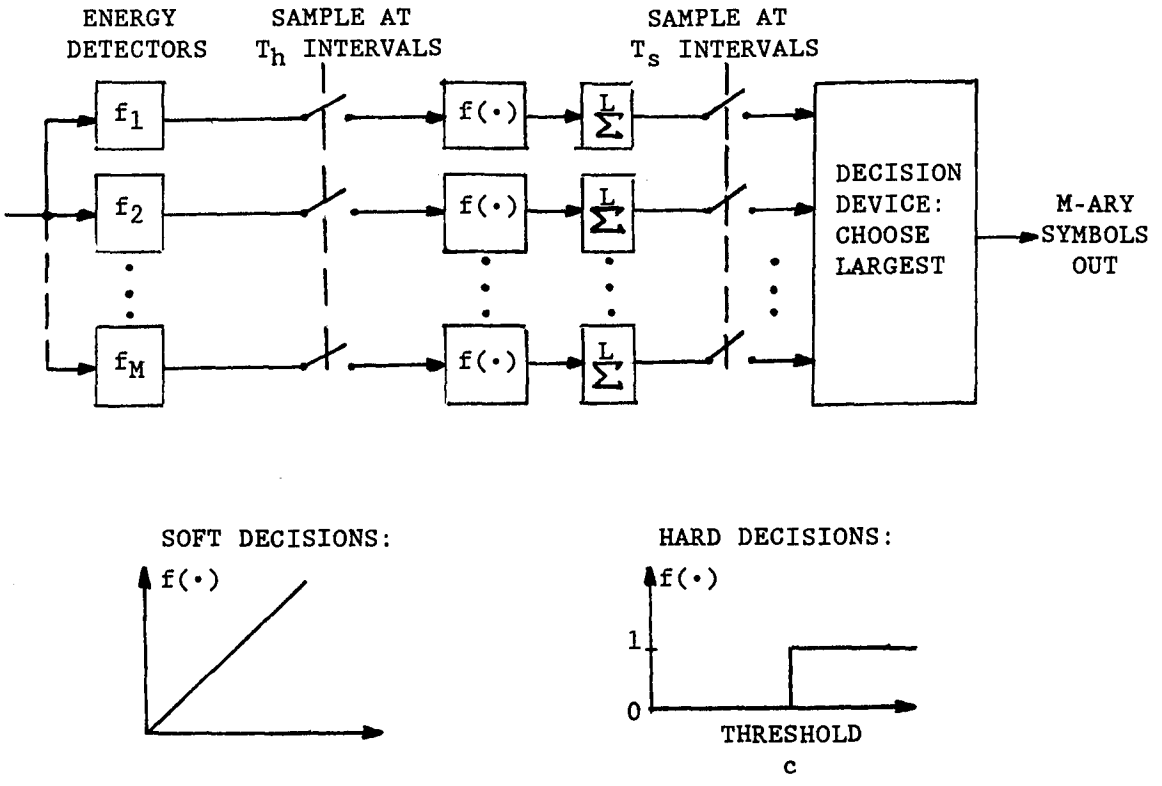
where $f_m$ is the $m^{th}$ FSK tone frequency given by

$$f_m = f_c - (M + 1 - 2m)\Delta f, \quad m = 1, 2, \cdots, M. \qquad (2.5\text{-}15)$$

A noncoherent demodulator for dehopped M-ary FSK signals for the more general case in which there is time diversity is shown in Figure 2.7. The dehopped signal is passed to a bank of M energy detectors tuned to the FSK signaling frequencies. The outputs of the energy detectors are sampled at the end of each hopping period and combined over a symbol period. The decision device chooses the symbol corresponding to the combining output with the largest amplitude at the end of a symbol period.

The diversity combining may make use of soft or hard decisions, depending on the function $f(\cdot)$. If soft decision combining is used, symbol decisions are based on

$$U_m = \sum_{k=1}^{L} U_{mk}\ , \qquad (2.5\text{-}16)$$




ENERGY DETECTORS:

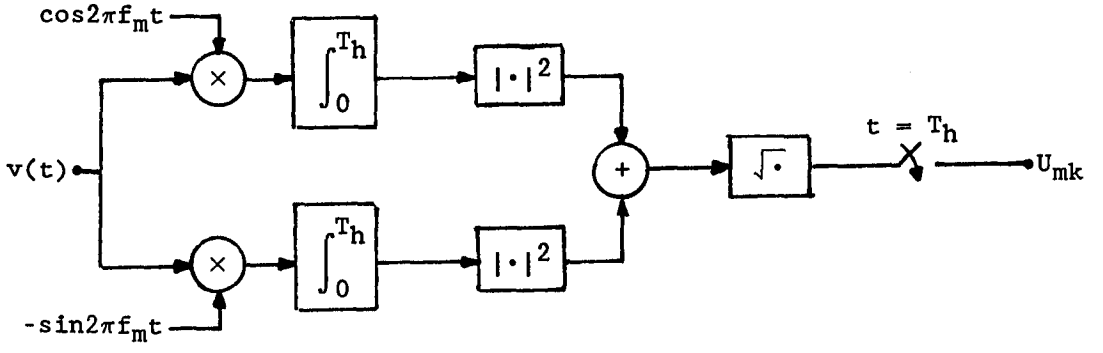



Figure 2.7 : A noncoherent M-ary FSK demodulator with diversity combining.

where $U_{mk}$ is the sampled output of the $m^{th}$ energy detector after the $k^{th}$ hop.

With hard decision combining, after each hop a decision is made for each energy detector output as follows:

$$U_{mk}' = \begin{cases} 1, & U_{mk} > c \\ 0, & U_{mk} \leq c, \end{cases} \tag{2.5-17}$$

where c is an appropriate threshold level. The final symbol decision is based on

$$U_m = \sum_{k=1}^{L} U_{mk}' \ . \tag{2.5-18}$$

When the received signal is degraded only by additive white Gaussian noise, the optimum demodulator makes use of soft decision combining, but when there is intentional jamming present at the receiver input, soft decisions may no longer be optimum and the performance with hard decision combining may be closer to the optimum. Indeed the optimum combining function f(•) changes as the form of the jamming signal changes, and there is no single f(•) which is optimum for all jamming signals.

The implementation of the energy detectors in Figure 2.7 can be achieved directly as shown in the figure, using 2M analog mixers and integrators, and M summers and local oscillators for each user. For block demodulation in which several signals separated by FDMA are demodulated at the same time, if either M or the number of users is large the direct implementation is cumbersome because of the number of components required, and alternatives are studied which make use of spectral analysis techniques. The spectral analysis receiver is equivalent to the direct implementation but when the number of frequencies to be processed is large the spectral analysis receiver will be much less complex than the direct implementation. A spectral analysis system makes use of surface acoustic wave (SAW) devices or digital signal processing to generate the decision variables.

### 2.5.2 The Spectral Analysis Receiver

With a spectral analysis receiver, the decision variables are obtained by inspecting the spectrum of the received signal at the appropriate frequencies. The decision variables given in equation (2.5-14) may be rewritten as

$$U_m = \left| \int_0^{T_s} v(t)\sqrt{2E_s/T_s}\exp(-j2\pi f_m t)\ dt \right|$$

$$= \left| \int_{-\infty}^{\infty} v(t)w(t)\sqrt{2E_s/T_s}\exp(-j2\pi f_m t)\ dt \right|$$

$$= \sqrt{2E_s/T_s} \left| F\{v(t)w(t)\} \right|_{f = f_m} \qquad (2.5\text{-}19)$$

where $w(t)$ is a rectangular time function with unit amplitude on the time interval $0 \le t \le T_s$ and zero amplitude otherwise, and $F\{\cdot\}$ denotes the Fourier transform operation. Thus, the decision variables are samples of the magnitude of the Fourier transform of the input windowed over the symbol period.

SAW devices can be used in the implementation of a spectral analysis receiver. SAW devices can be used to perform a chirp transform in which output time represents input frequency, and the frequency domain samples or decision variables are obtained by sampling the chirp transformer output at the correct instants in time [4]. A block demodulator implemented with a SAW chirp transformer has been investigated [4] and the results indicated this approach gives very good performance.

An alternative implementation of a spectral analysis receiver is to use digital signal processing techniques and the discrete Fourier transform. With the ever increasing processing speeds available for digital components, a discrete Fourier transform-based block demodulator is a viable alternative to the analog implementations. A digital implementation of a Fourier transforming block demodulator is the focus of this project.

### 2.5.3 Digital Demodulation Techniques

With digital processing, it is convenient to generate a low pass equivalent of the received signal. A complex baseband representation can be obtained as shown in Figure 2.8 [7]. The received bandpass signal is split into two arms and one arm is multiplied by $2\cos(2\pi f_c t)$ while the other is multiplied by $-2\sin(2\pi f_c t)$. The inphase and quadrature components of $r(t)$ are obtained by filtering the products with identical low pass filters. The complex baseband signal is sampled over the symbol period with a sampling rate $f_s$. The sampling rate must be at least twice the highest possible signaling tone frequency $(M\text{-}1)\Delta f$. Sampling will produce $N = f_s T_s$ samples in one symbol period, and the sequence of complex samples is represented as

$$r(n) = r(t)\Big|_{t = nT}, \quad n = 0, 1, \cdots, N\text{-}1 \qquad (2.5\text{-}20)$$

where $T = 1/f_s$ is the sampling period.

Replacing the integral with a summation in equation (2.5-13) results in the decision variables

$$U_m = \left| \sum_{n=0}^{N-1} r(n) u_m^*(nT) \right|$$

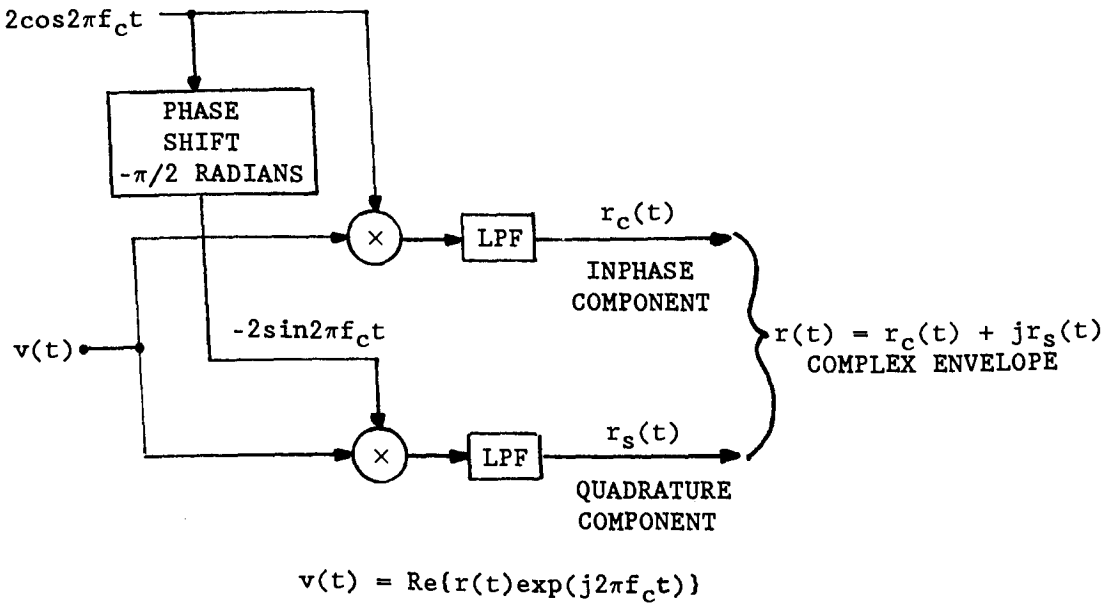



Figure 2.8 : Generation of a complex baseband signal representation.

$$= \left| \sum_{n=0}^{N-1} r(n)\sqrt{2E_s/T_s}\exp(-j2\pi a_m \Delta f n T) \right|$$

$$= \left| \sum_{n=0}^{N-1} r(n)\sqrt{2E_s/T_s}\exp(-j2\pi a_m \Delta f N T n/N) \right|$$

$$= \left| \sum_{n=0}^{N-1} r(n)\sqrt{2E_s/T_s}\exp(-j2\pi nk/N) \right| \qquad (2.5\text{-}21)$$

where $k = Na_m\Delta fT = Na_m\Delta f/f_s$. If the sampling rate and tone spacings are judiciously chosen such that k is an integer for all $a_m$, then the decision variables are scaled samples of the magnitude of the discrete Fourier transform of the equivalent lowpass received signal, and equation (2.5-21) can be rewritten as

$$U_m = \sqrt{2E_s/T_s}\ |R(k)|, \quad k = Na_m\Delta f/f_s \tag{2.5-22}$$

where R(k), k = 0, 1, •••, N-1 denotes the N-point discrete Fourier transform of the sequence r(n) and is defined as [8]

$$R(k) = \sum_{n=0}^{N-1} r(n)\exp(-j2\pi nk/N). \tag{2.5-23}$$

The discrete Fourier transform produces N frequency domain samples evenly spaced from $-f_s/2$ to $f_s/2$ at the locations $0, \pm f_s/N, \pm 2f_s/N, \bullet\bullet\bullet, \pm(N-1)f_s/2N, f_s/2$. Because the spectrum of the sampled sequence is periodic with period $f_s$, any spectral component loccated at $-f_s/2$ will be identical to that at $+f_s/2$.

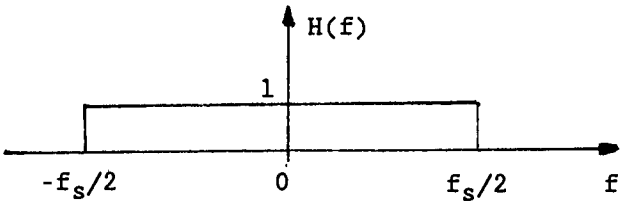



Figure 2.9 : Frequency response of ideal anti-aliasing filters.

If the low pass filters used in generating the complex baseband signal are ideal anti-aliasing filters with frequency response shown in Figure 2.9, the performance of this digital demodulator will be identical to the performance of the optimum analog demodulator described by equation (2.5-14). The performance is expressed in terms of the probability that the receiver decides in favour of the incorrect symbol. The probability of a decision error is derived by assuming that $u_1(t)$ is the transmitted symbol [6]. Then the sequence obtained by sampling the equivalent lowpass received signal is

$$r(n) = u_1(nT)\alpha\exp(j\Theta) + \xi(n) \tag{2.5-24}$$

where $\xi(n)$ are samples of the noise process z(t). If z(t) is obtained by passing white Gaussian noise through the filters with frequency response shown in Figure 2.9, then it has a power spectral density and autocorrelation given respectively by

$$S_z(f) = N_o\,\mathrm{rect}(f/f_s) \tag{2.5-25}$$

and

$$R_z(\tau) = f_s N_o \frac{\sin(\pi\tau f_s)}{\pi\tau f_s} \tag{2.5-26}$$

and the sampled sequence $\xi(n)$ will have an autocorrelation and power

spectral density given by

$$R_\xi(n_1,n_2) = f_s N_o \delta(n_1-n_2) \tag{2.5-27}$$

and

$$S_\xi(f) = f_s N_o. \tag{2.5-28}$$

Thus the noise samples will be mutually uncorrelated with zero mean and variance equal to $N_o f_s = N_o/T$.

The receiver calculates the decision variables

$$U_m = \left| \sum_{n=0}^{N-1} [u_1(nT)\alpha \exp(j\theta) + \xi(n)]\sqrt{2E_s/T_s}\exp(-j2\pi a_m \Delta f n T) \right|$$

$$= \left| \frac{2\alpha E_s}{T_s} \sum_{n=0}^{N-1} \exp[j2\pi(a_1-a_m)\Delta f n T] + \sum_{n=0}^{N-1} \sqrt{2E_s/T_s}\xi(n)\exp(-j2\pi a_m \Delta f n T) \right|. \tag{2.5-29}$$

The phase factor $\exp(j\theta)$ is neglected because it is irrelevent due to the noncoherent detection. For orthogonal tone spacings, using equation (2.5-2) and the facts that $T = T_s/N$ and for orthogonal tone spacing $2\Delta f = i/T_s$ where i is an integer, the first summation in equation (2.5-29) simplifies to

$$\sum_{n=0}^{N-1} \exp[j2\pi(a_1-a_m)\Delta f n T] = \sum_{n=0}^{N-1} \exp[-j2\pi 2(m-1)\Delta f T_s n/N]$$

$$= \sum_{n=0}^{N-1} \exp[-j2\pi(m-1)in/N]$$

$$= \begin{cases} N, & m = 1 \\ 0, & m \neq 1 \end{cases} \tag{2.5-30}$$

Then the decision variables can be expressed as

$$U_1 = \left| \frac{2N\alpha E_s}{T_s} + N_1 \right|$$

$$U_m = |N_m| \quad m = 2, 3, \cdots, M \tag{2.5-31}$$

where the noise terms $\{N_m\}$ given by

$$N_m = \sum_{n=0}^{N-1} \sqrt{2E_s/T_s}\,\xi(n)\exp[-j2\pi(2m-M-1)in/2N] \qquad (2.5\text{-}32)$$

are mutually uncorrelated and identically distributed Gaussian random variables with zero mean and variance given by

$$\sigma_n^2 = 2NE_s f_s N_o/T_s. \qquad (2.5\text{-}33)$$

If the symbol and noise energies are redefined as

$$\begin{aligned} E_s' &= NE_s/T_s \\ N_o' &= f_s N_o \end{aligned} \qquad (2.5\text{-}34)$$

then the decision variables become

$$\begin{aligned} U_1 &= |2\alpha E_s' + N_1| \\ U_m &= |N_m| \quad m = 2, 3, \cdots, M \end{aligned} \qquad (2.5\text{-}35)$$

and the variance of the $\{N_m\}$ terms becomes

$$\sigma_n^2 = 2E_s'N_o'. \qquad (2.5\text{-}36)$$

With these definitions the derivation of the probability of a decision error is identical to that for the optimum analog receiver found in [6] and the final result is

$$P_s = \sum_{n=0}^{M-1} (-1)^{n+1} \frac{(M-1)!}{(n+1)!(M-n-1)!} \exp\left[\frac{-\alpha^2 E_s'}{N_o'} \frac{n}{n+1}\right]. \qquad (2.5\text{-}37)$$

Noting that

$$\frac{E_s'}{N_o'} = \frac{NE_s}{T_s f_s N_o} = \frac{E_s}{N_o} \quad , \qquad (2.5\text{-}38)$$

the error performance of the digital receiver is thus identical to that of the optimum analog receiver. It is convenient to express the error probability in terms of bit errors instead of symbol errors because it is easier to measure the bit error probability $P_b$. For the optimum orthogonal NCFSK receiver, the bit error probability is [6]

$$\begin{aligned} P_b &= \frac{M}{2(M-1)} P_s \\ &= \frac{\exp(-E_{sr}/N_o)}{2(M-1)} \sum_{m=2}^{M} (-1)^m \frac{M!}{m!(M-m)!} \exp(E_{sr}/mN_o) \end{aligned} \qquad (2.5\text{-}39)$$

where $E_{sr} = \alpha^2 E_s$ is the received energy per symbol.

A digital implementation of a multiple-user NCFSK demodulator with diversity combining is shown in Figure 2.10a. In this example the received signal is sampled at an IF frequency and a complex baseband representation of the received signal is generated with digital components after sampling by multiplying the samples with the sequence $\exp(-j2\pi f_c n/f_s')$ where $f_s'$ is the IF sampling rate, and filtering the resulting real and imaginary sequences with identical digital lowpass filters. After filtering, the complex sequence is resampled at a lower rate $f_s$ in order to reduce the computational load of the discrete Fourier transform which follows. Before the discrete Fourier transform is performed, the complex sequence may be multiplied by a time varying window function w(n). The motivation for this is discussed in the next section. The diversity combining and decision sections are the same as for the analog receiver discussed in section 2.5.1.

Figures 2.10(b-g) illustrate the process of generating the complex baseband equivalent signal with sketches of the frequency spectra at various points in the demodulator of Figure 2.10a. Figure 2.10b is an example of the frequency spectrum of a narrowband signal centered on a frequency $f_c$. After sampling, the spectrum is periodic with a period equal to the sampling frequency as shown in Figure 2.10c. Multiplication by $\exp(-j2\pi f_c n/f_s')$ results in a frequency shift of the spectrum by the amount $f_c$ (Figure 2.10d). If the digital lowpass filters both have the frequency response shown in Figure 2.10e, then the their outputs will be the inphase and quadrature components of the complex baseband signal with the combined spectrum shown in Figure 2.10f. The windowing and discrete Fourier transform could be performed directly on the filter outputs, but it is clear from Figure 2.10f that there is a large frequency band which contains no relevent information about the received signal. The number of terms needed in calculating the decision variables expressed in equations (2.5-22) and (2.5-23) is equal to N, the number of samples in the sequence used to represent the received signal. Figure 2.10g shows that N can be reduced with no loss of information by resampling at a lower rate, thus reducing the amount of computation needed to calculate each decision variable.

The complex baseband signal can be generated before sampling using analog components as shown in Figure 2.8. This eases the digital processing load, which is important when the symbol duration (or hopping period for the case with time diversity) is short and the amount of digital processing available for each symbol (hopping) period is limited. As well, there is less demand on the analog-to-digital converters, as the required sampling rate is lower because the conversion is done on the baseband signal. The disadvantage is that phase matching of the lowpass filters and other analog devices used in generating the inphase and quadrature signal components is much more difficult than for the equivalent digital implementation. Phase matching is required to maintain the 90° phase quadrature between the two channels.

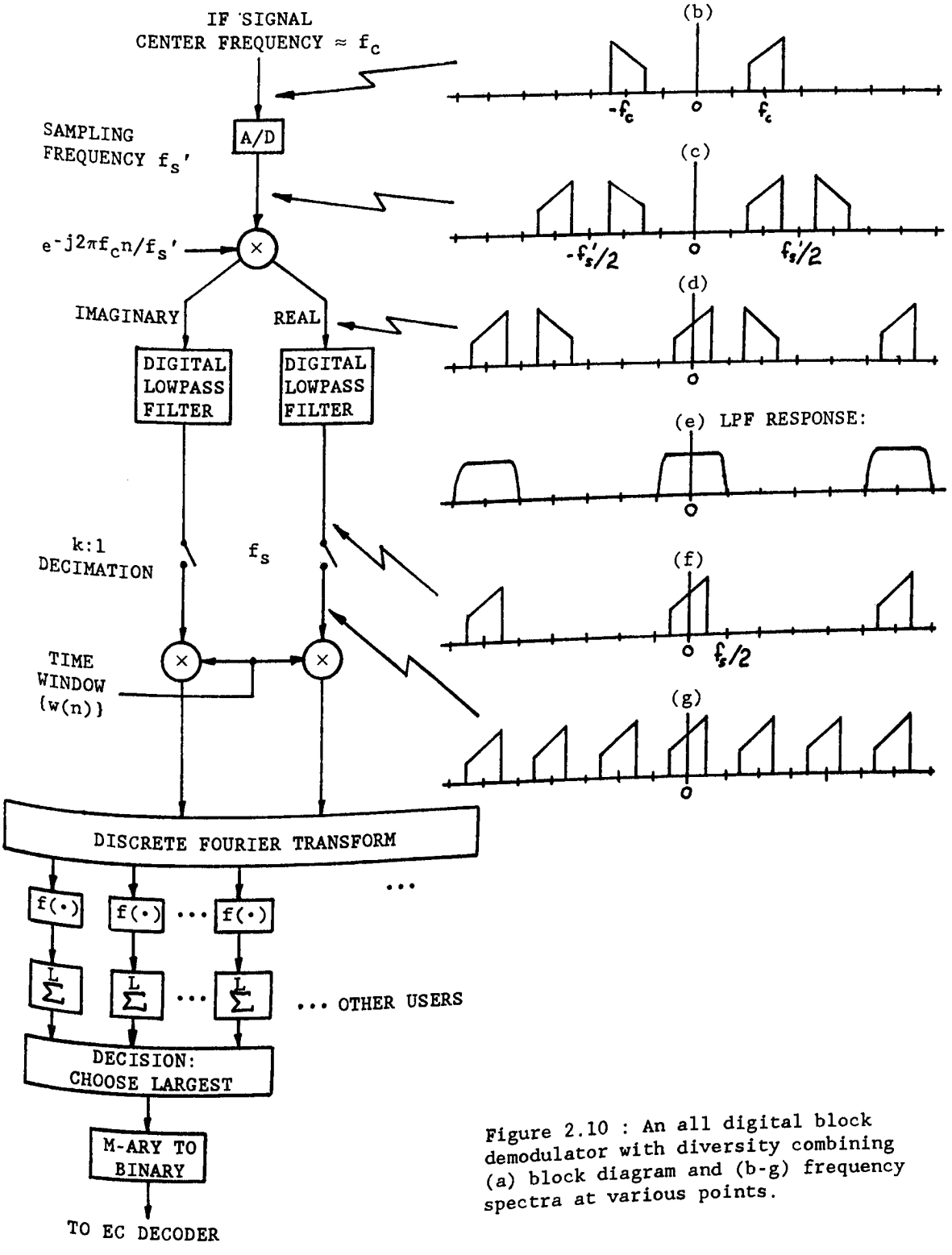



Figure 2.10 : An all digital block demodulator with diversity combining (a) block diagram and (b-g) frequency spectra at various points.

## 2.6 RECEIVER WINDOWING

Under some circumstances it may be desirable to multiply the received signal by a time varying window rather than the implied rectangular window with the duration of the hopping period [9]. Windowing, which can be used in both the analog receiver of section 2.5.2 and the digital receiver of section 2.5.3, is discussed in this section in terms of the digital receiver, but in either case the effect of windowing is the same. To see the effect of windowing on the response of the discrete Fourier transform (DFT) defined by equation (2.5-23), it is convenient to think of the DFT as a bank of N contiguous filters with unit sample responses given

$$h_k(n) = [u(n) - u(n-N)]\exp(-j2\pi nk/N)$$

$$= w(n)\exp(-j2\pi f_k nT), \quad k = 0, 1, \cdots, N-1 \qquad (2.6\text{-}1)$$

where $u(n)$ is the unit step sequence, $w(n)$ is a rectangular window function of length N, and $f_k = k/T_s = k/NT$ is the center frequency of the $k^{th}$ DFT filter. The $k^{th}$ filter's frequency response is then

$$H_k(f) = W(f - f_k) \qquad (2.6\text{-}2)$$

where $W(f)$ is the frequency response of the window sequence $w(n)$ and is given by

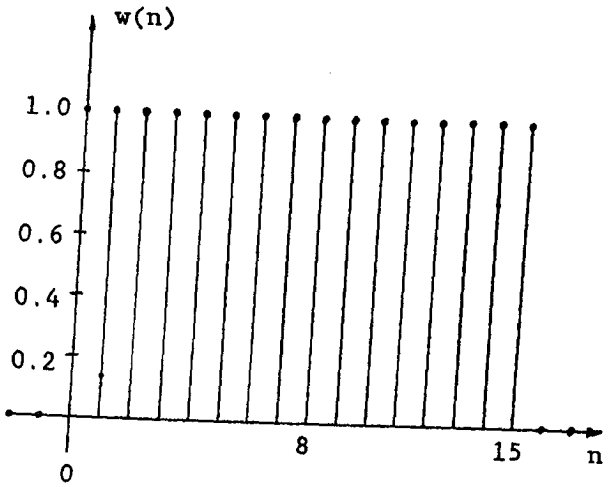
$$W(f) = \sum_{n=0}^{N-1} w(n)\exp(-j2\pi fnT). \qquad (2.6\text{-}3)$$

Thus the shape of the DFT filters is determined by the Fourier transform of the window.

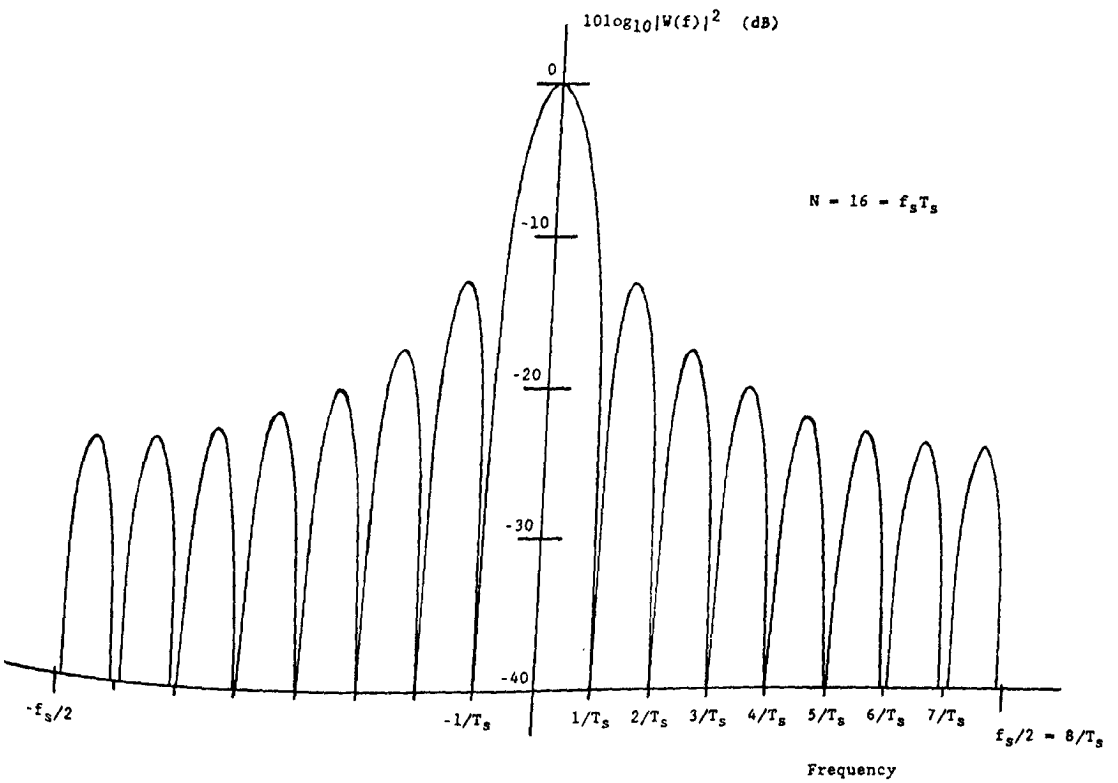
When there is no explicit windowing, the DFT filter response is that of the rectangular window, and the magnitude is [10]

$$W(f) = \left|\frac{\sin(\pi fNT)}{\sin(\pi fT)}\right| . \qquad (2.6\text{-}4)$$

Figure 2.11 shows the rectangular window with N = 16 and the log magnitude of its transform. This illustrates the criteria for orthogonality of the NCFSK signal tones. For orthogonal signaling, the received signal energy will be detected only in the DFT filter associated with the transmitted frequency. From the plot of the DFT frequency response for the rectangular window, there are nulls in the response located at integer multiples of $1/T_s$ from the center of the main lobe, which is therefore the required tone spacing for orthogonal NCFSK signals. When various users are separated by FDMA, the signaling frequencies for each user are orthogonally spaced from those of other users. For a block demodulator, crosstalk, in which signal energy is detected by DFT filters other than the correct ones, results when there




(a)




(b)

Figure 2.11 : (a) The rectangular window (N = 16) and (b) the log magnitude of its transform.

is not perfect frequency and timing synchronization between each of the users and the dehopping synthesizer of the receiver. This interchannel interference can be significant when there are large differences in received power levels between the users. Because it is unlikely that there will be perfect synchronization between the users, nonrectangular windowing is used at the receiver to improve the tolerance to crosstalk.

There are several figures of merit used in characterizing windows, three of which are the 3 dB bandwidth of the main lobe, the level of the highest sidelobe, and the processing loss or equivalent noise bandwidth of the window. With a time varying window, the DFT filter shape is no longer the optimum for detection of sinusoids in the presence of white noise, and the processing loss is a measure of the loss in signal detectability with the time varying window with respect to that for the optimum rectangular window. The processing loss (PL) is the ratio of input signal-to-noise ratio to output signal-to-noise ratio with a complex sinusoidal sequence at the input with frequency $f_k$ matched to one of the DFT filters [10]. This ratio is also equal to the window's equivalent noise bandwidth (ENBW) given by

$$\text{ENBW} = \text{PL} = \frac{N \sum_{n=0}^{N-1} w^2(n)}{\left[\sum_{n=0}^{N-1} w(n)\right]^2} . \qquad (2.6\text{-}5)$$

The processing loss then is the increase in transmission power required with windowing above that required with rectangular windowing to maintain the same bit error probability under ideal conditions (perfect synchronization). For the rectangular window, the processing loss is by definition 0 dB.

The maximum sidelobe level determines the amount of interchannel interference that can be tolerated with a given window. For the rectangular window with N = 32, the highest sidelobe level is 13.2 dB below the peak of the main lobe. By tapering the edges of the window, the sidelobe levels drop at the expense of an increased width of the main lobe and a higher processing loss.

For a given set of assumptions on the number of users, the number of signaling tones per user, the tone spacing, and the probability density functions characterizing the time and frequency misalignments between the various users and the dehopping synthesizer onboard the satellite, there may be an optimum window that minimizes the average crosstalk-to-signal ratio [9], which is for a given user the average total crosstalk energy detected by one of the DFT filters divided by the average signal energy detected by the DFT filter corresponding to the transmitted symbol. Of course this window would no longer be optimum for a slightly different set of

assumptions, making the choice of a window difficult. The Kaiser-Bessel family of windows defined as

$$w(n) = \frac{I_o\left[\alpha\pi\sqrt{1 - (1 - 2n/N)^2}\right]}{I_o[\alpha\pi]} \qquad n = 0, 1, \cdots, N-1 \qquad (2.6\text{-}6)$$

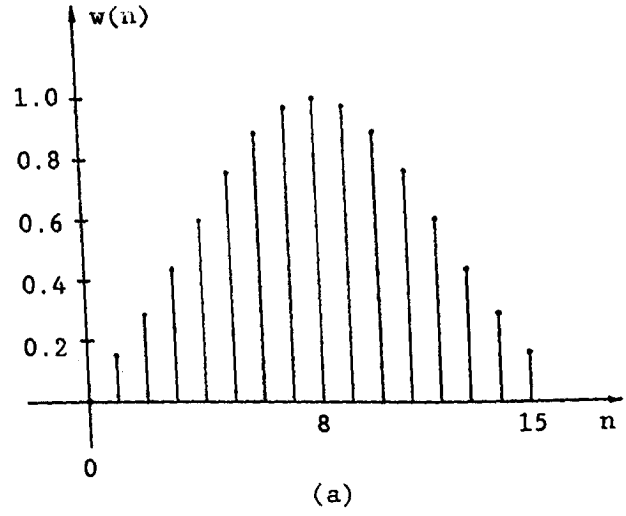
has good sidelobe properties and the tradeoff of sidelobe level versus main-lobe width can be directly manipulated with the parameter $\alpha$. As $\alpha$ is increased, the sidelobe levels drop and the main lobe widens. Figure 2.12 shows a Kaiser Bessel window and its transform with N = 16 and $\alpha$ = 1.4. The highest sidelobe level is 29.9 dB down from the main lobe peak, and the processing loss is 1.17 dB. Table 2.1 summarizes the properties of the Kaiser-Bessel window for N = 32 as $\alpha$ is varied from 1.2 to 1.8. The characteristics for the rectangular window are included for comparison. Because the frequency spectrum of the the continuous time version of the window w(t) is not strictly bandlimited the sidelobe levels of the discrete time version w(n) will be higher due to aliasing. As the number of samples N is increased the sampling rate is effectively increased and the amount if aliasing decreases, lowering the sidelobe levels. As N becomes very large the sidelobe levels approach those for the continuous time version of the window. Comparing the maximum sidelobe level for N = 32 listed in Table 2.1 with that from the plot for N = 16 for the Kaiser-Bessel window, the maximum level is indeed 0.4 dB lower for N = 32.

Thus in choosing a window there is a tradeoff between the amount of crosstalk that can be rejected determined by the sidelobe levels, and the amount of processing loss that can be tolerated. If processing loss is not a major factor, then the FSK tone spacing limits how wide the main lobe can be expanded and hence how much the sidelobe levels can be lowered.

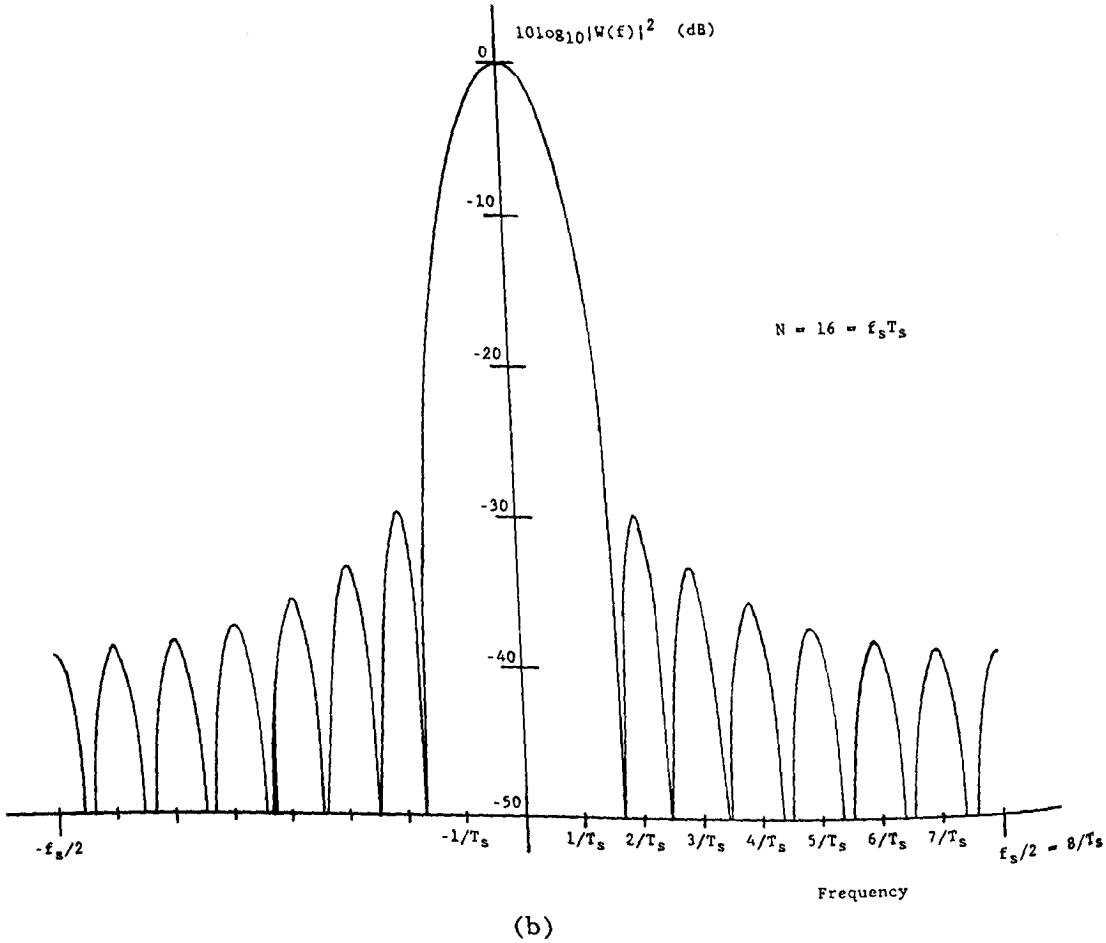
Now that receiver windowing has been discussed, all of the major theoretical aspects of NCFSK block demodulation have been addressed, and the report can focus on the implementation of a digital block demodulator as presented in the next chapter. Before proceeding it is worthwhile to summarize the ideas presented in this chapter.

## 2.7 SUMMARY

In this chapter frequency-hopping systems have been discussed in general to show the motivation for this project. In section 2.3 several suitable applications have been presented which illustrate the need or desire for using frequency-hopping in a communication system. After presenting a typical frequency-hopping system in section 2.4, the focus of the discussion narrowed to demodulation techniques for frequency-hopping receivers in section 2.5 and 2.6. The spectral analysis receiver was emphasized because of its practicality for systems with several users multiplexed in frequency and receiver windowing was introduced as a means of reducing possible interference between the various users. Two competing technologies were presented for implementing a Fourier transforming




(a)




(b)

Figure 2.12 : (a) The Kaiser-Bessel window ($\alpha$ = 1.4, N = 16) and (b) the log magnitude of its transform.

| WINDOW | | HIGHEST SIDELOBE LEVEL(dB) | PROCESSING LOSS (dB) | 3 dB BANDWIDTH ($\div 1/T_s$ Hz) |
|---|---|---|---|---|
| Rectangular | | -13.2 | 0 | 0.88 |
| Kaiser-Bessel | $\alpha = 1.2$ | -27.3 | 0.91 | 1.19 |
| | $\alpha = 1.4$ | -31.3 | 1.17 | 1.25 |
| | $\alpha = 1.6$ | -35.6 | 1.36 | 1.31 |
| | $\alpha = 1.8$ | -40.4 | 1.57 | 1.37 |

Table 2.1 : Comparison of window parameters.

demodulator, the first being a SAW-based analog demodulator. It was shown that the performance of a digital implementation using the discrete Fourier transform in theory is equivalent to the optimum analog demodulator and as a SAW-based demodulator has been built and tested [4], it would be worthwhile to do the same for a digital demodulator. The implementation of a digital block demodulator is presented in the next chapter.

# CHAPTER 3

# AN NCFSK BLOCK DEMODULATOR

## 3.1 INTRODUCTION

In the past, NCFSK demodulators for jamming-resistant frequency-hopping systems have been implemented primarily with analog technologies as described in section 2.5.2 because the hopping rates required for such systems have severely limited the time available for processing during each frequency hop. With VLSI technology it is now conceivable to build a digital block demodulator which is able to process a reasonable number of dehopped multiple-user NCFSK signals in real time. The objective of this project is to demonstrate the feasability and to determine the performance of the digital approach. The design of a digital block demodulator is presented in this chapter and measurements of it's performance are presented in chapter 4.

## 3.2 DEMODULATOR CONFIGURATION

To facilitate comparison with other demodulator implementations, no diversity combining is included in the demodulator design, and therefore the hopping rate $R_h$ is treated as the symbol rate $R_s$. The hopping rate is set to 16 k hops/s, which might be a typical value for a jamming-resistant system which is fast enough to avoid follower jammers yet slow enough to allow synchronization at the receiver. In the design of the demodulator, perfect time synchronization between the various users and the receiver is assumed. The effect of frequency misalignment is considered and measured in section 4.5 of the next chapter.

The symbol period is 62.5 $\mu$s, but to allow for finite switching times of the hopping and dehopping synthesizers, the demodulator operates only on the last 50 $\mu$s of the received symbol, leaving the first 12.5 $\mu$s for the symbol frequencies and phases to settle to their final values. Thus the effective symbol period is $T_s$ = 50 $\mu$s and the minimum orthogonal tone spacing is $(50\ \mu s)^{-1}$ = 20 kHz. To allow for receiver windowing as described in the previous chapter, the FSK tone spacing is set at twice the minimum value or 40 kHz. The input to the demodulator is thus assumed to be synchronized FSK signals multiplexed in frequency with random phases and tone spacings of multiples of 40 kHz.

For the purpose of this study, the most convenient implementation of the digital demodulator is with readily available high speed microprocessors. Although a custom VLSI implementation on a single chip would be more advantagous in terms of size, weight and power consumption which are all very important when considering deployment onboard a satellite, the intention here is not to necessarily produce a digital demodulator suitable for final deployment but to show that such a goal is feasible. Thus the design

makes use of TMS32020 digital signal processors produced by Texas Instruments. These microprocessors are designed for high-speed and numeric-intensive digital signal processing applications.

Each of the M possible tone locations for all of the users will be refered to as a signaling bin. If there are D users then the demodulator must process C = DM bins to produce C decision variables as described by equation (2.5-21). The 62.5 $\mu$s symbol period and the instruction time of the TMS32020 set a limit on the number of calculations that can be performed during each symbol period and hence limit the number of bins which can be processed which in turn limits the number of users which can be accomodated by the block demodulator.

The instruction cycle time of the TMS32020 is 195 ns, which limits the processing to 320 instructions per symbol period for a single microprocessor. If the received signal is sampled at an IF frequency as shown in Figure 2.10, the 320 instructions must be used to perform several functions including sample retrieval, multiplication by the sequence $\exp(-j2\pi f_c n/f_s')$, low pass filtering of the two resulting sequences, multiplication of each sequence by a time window, calculation of C decision variables, D(M-1) comparisons, and D symbol outputs. If a single microprocessor cannot handle all of the required processing, there are several options available to meet the processing requirements, the simplest of which is to use analog components to generate the equivalent complex baseband signal. Alternatively, the processing can be broken down into steps, each carried out by a separate microprocessor in a pipeline structure. For example one microprocessor could be dedicated to generating the complex baseband representation while a second could perform the actual demodulation. A third option is to use parallel processing in which N microprocessors are used, each in turn operating on every $N^{th}$ symbol period with a maximum processing time of $NT_s$. Finally, any combination of the above three techniques may be employed.

In order to keep the implementation fairly simple, analog components are used to generate the baseband signal and only two TMS32020s are used in parallel to perform the demodulation functions. This means that for each symbol period there is a resource of 640 instruction cycles available for sample retrieval, windowing, DFT calculation, and decision making. As will be shown in this chapter, this is enough instructions to process 16 signal bins, which means the demodulator can produce data for up to eight binary FSK users, four 4-ary FSK users, two 8-ary FSK users, or one 16-ary FSK user.

The demodulator input is an IF signal centered on 10.7 MHz with a bandwidth of approximately 16x40kHz = 640kHz. Figure 3.1 illustrates the locations of the 16 signaling bins at the demodulator input. Also shown in Figure 3.1 is the tone allocation for four 4-ary FSK users. For reasons which will become apparent in section 3.4, for the purposes of generating the complex baseband signal it is better to label the center frequency $f_c$ such that it corresponds to a center frequency of one of the signaling bins, which in this case will not correspond to the true center frequency of the IF signal. Thus in generating the inphase and quadrature components the local oscillator frequency is set to $f_c$ = 10.68 MHz. The equivalent lowpass

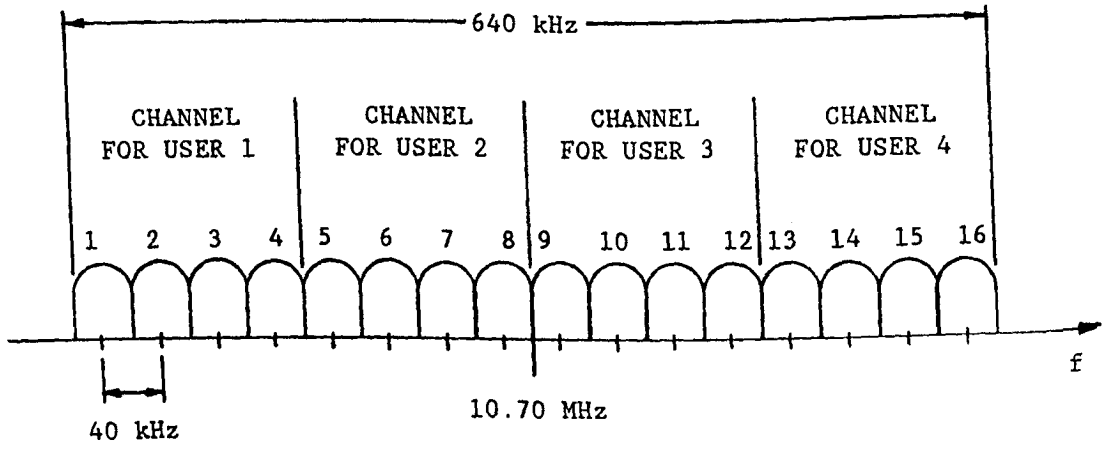



Figure 3.1 : IF signaling bin locations at demodulator input, and bin allocation for 4-ary FSK users.

bins are then located as shown in Figure 3.2. This sets the requirements for the sampling described in the next section.

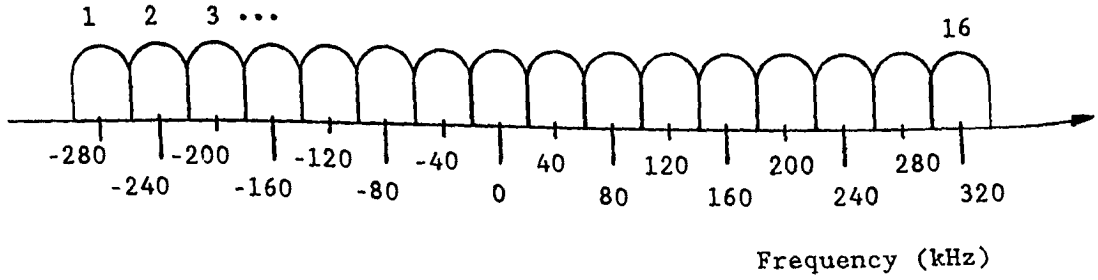



Figure 3.2 : Equivalent lowpass bin locations with a local oscillator frequency of 10.68 MHz.

## 3.3 SAMPLE GENERATION

From Figure 3.2, the highest possible tone frequency in the complex baseband signal is 320 kHz so the minimum sampling frequency for the inphase and quadrature arms is 640 kHz. Sampling at the minimum rate over 50 $\mu s$ results in 32 complex samples. A 32-point DFT produces frequency domain samples spaced by 640/32 = 20 kHz. Thus the set of decision variables would consist of every second DFT output, corresponding to the even k's in the DFT given by equation (2.5-23).

When sampling an analog signal it is important to know the spectral characteristics of the signal so that appropriate lowpass filtering can be done before sampling without affecting the signal but limiting the noise spectrum to frequencies less than $f_s/2$ to avoid noise aliasing. In section 2.5 the spectrum of an NCFSK signal was shown to be a superposition of $\text{sinc}^2(T_s f)$ shapes centered on the FSK tone frequencies. If the bandwidth of the $\text{sinc}^2(T_s f)$ shapes is taken as the bandwidth between the first nulls on either side of the main lobe (see Figure 2.6) then their bandwidth is $2/T_s = 32$ kHz, the IF signal has a bandwidth of 632 kHz, and the inphase and quadrature components of the equivalent baseband signal each have a bandwidth of 336 kHz. This suggests that the minimum sampling rate is 672 kHz. This sampling rate would be required if the receiver has no knowledge other than the bandwidth of the incoming signal and sampling is continuous through the symbol switching transients but for this demodulator, switching instants are known and there is no need for sampling to continue through the transitions. Since there is no difference between sampling the received signal over one symbol period and sampling a sum of pure sinusoids with the symbol frequencies, the minimum sampling rate is twice the frequency of the highest possible tone frequency, as previously stated.

Sampling at the minimum rate creates unresolvable requirements on the lowpass filters in the inphase and quadrature arms. In order to pass the desired signal to the samplers without removing significant signal energy, the filters should have a bandwidth of at least 336 kHz, but to avoid aliasing the noise spectrum and hence increasing the noise power in the sampled sequence the filters should have a bandwidth no greater than 320 kHz. Thus in sampling at the minimum rate one is forced to compromise the system's error performance by using filters that either remove signal energy or increase the noise energy. There are two alternatives that may be used to avoid this problem. The obvious solution is to increase the sampling rate to at least 672 kHz. The lowest convenient sampling rate above 672 kHz which produces an integer number of samples when the sampling is carried out over exactly 50 $\mu$s is 680 kHz. This rate produces 34 samples and the decision variables would be samples of a 34-point DFT. Unfortunately determining 16 outputs of a 34-point DFT requires far more calculations than needed to determine 16 outputs of a 32-point DFT, as will be shown in the next section.

The other option is to set the local oscillator frequency to 10.7 MHz resulting in the equivalent baseband signal bins being located as shown in Figure 3.3. In this situation, the alternate 16 samples of the output of the DFT corresponding to the odd k's in equation (2.5-23) are of interest. The bandwidth of the inphase and quadrature arm signals would be 316 kHz and there is no longer conflicting requirements on the filters. As alluded to previously though and as will be shown in the next section, determining the odd outputs of the DFT requires more calculations than determining the even outputs. The only way to process the 16 signaling bins with the 640 available instructions is to use the even outputs of the DFT and accept the performance degradation brought about by filters which do not meet both the bandwidth requirements for passing most of the signal energy and limiting the noise spectrum so that there is no noise aliasing.

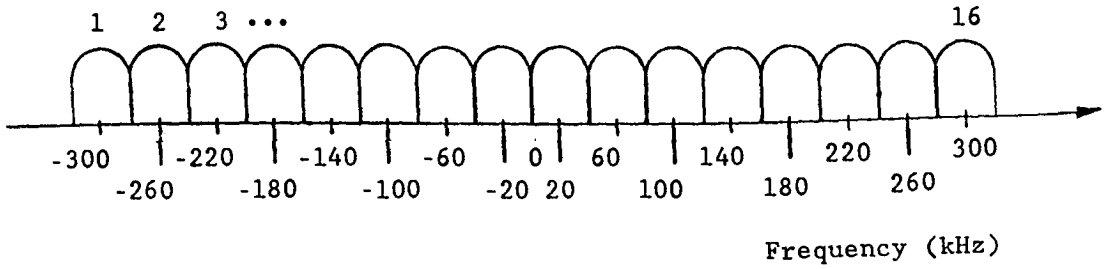



Figure 3.3 : Equivalent lowpass bin locations with a local oscillator frequency of 10.70 MHz.

Another degradation associated with sampling is that due to quantization errors resulting from the finite number of bits used to represent each sample. Before discussing quantization errors, it is useful to know the manner in which signed numbers are represented for processing. The TMS32020 uses fixed-point arithmetic with two's-complement number representation in which all numbers are treated as fractions. With two's complement representation [8] the the sign is determined by the leading binary digit and the rest of the digits represent the magnitude in a manner depending on the sign. When the leading bit is a 0 the number is positive and the magnitude is given directly by the other bits. When the leading bit is a 1 the number is negative and its magnitude is determined by subtracting from two the magnitude of the bits following the leading bit. Table 3.1 shows the two's complement number system for four bit numbers.

When the samples are quantized by the analog-to-digital (A/D) converters the amplitude of the analog inputs to the A/D converters must be suitably scaled such that the unquantized samples do not exceed the range of the numbers which can be expressed with the number of bits used in quantization. If the quantized samples are represented with (b+1) bits (i.e. one bit for the sign and b bits for the magnitude) then the analog input must be scaled such that the unquantized samples are within the range

$$-(1 + 2^{-b}/2) < r(nT) < (1 - 2^{-b}/2). \qquad (3.3\text{-}1)$$

If the analog signal contains a Gaussian noise process, equation (3.3-1) cannot be met unconditionally. The scaling factor should be set such that equation (3.3-1) is met with a high probability so that the input signal only rarely exceeds the dynamic range of the A/D converter yet on average the signal amplitude still covers a fair range of bits. This suggests the need for automatic gain control (AGC) to keep the analog signal level of the A/D inputs constant over long time periods as the received signal strength varies. For the demodulator measurements described in the next chapter, AGC was achieved by monitoring the A/D converter input voltage levels on an oscilloscope and adjusting a variable attenuator at the demodulator input so that the A/D converters were rarely saturated.

| FRACTIONAL NUMBER | TWO'S COMPLEMENT BINARY EQUIVALENT |
|---|---|
| 7/8 | 0.111 |
| 6/8 | 0.110 |
| 5/8 | 0.101 |
| 4/8 | 0.100 |
| 3/8 | 0.011 |
| 2/8 | 0.010 |
| 1/8 | 0.001 |
| 0 | 0.000 |
| -1/8 | 1.111 |
| -2/8 | 1.110 |
| -3/8 | 1.101 |
| -4/8 | 1.100 |
| -5/8 | 1.011 |
| -6/8 | 1.010 |
| -7/8 | 1.001 |
| -1 | 1.000 |

Table 3.1 : Two's complement number representation.

Assuming the A/D converters round the signal to the nearest quantization level and the analog signal does not exceed the dynamic range of the A/D converters, the quantization error e(n) for each sample will be in the range

$$-2^{-(b+1)} < e(n) < 2^{-(b+1)} \quad . \tag{3.3-2}$$

In analysing the effect of quantization the error sequence e(n) is commonly treated as additive noise [8] and the magnitude of the error samples are characterized by a uniform distribution over the range of equation (3.3-2) as shown in Figure 3.4. The error samples have zero mean and variance $\sigma_e^2$ given by

$$\sigma_e^2 = \frac{2^{-2b}}{12} \quad . \tag{3.3-3}$$

It is assumed that the error samples are uncorrelated from sample to sample and uncorrelated with the sequence of exact samples r(nT). Then the error sequence has an autocorrelation given by

$$R_e(n) = \sigma_e^2 \delta(n) \tag{3.3-4}$$

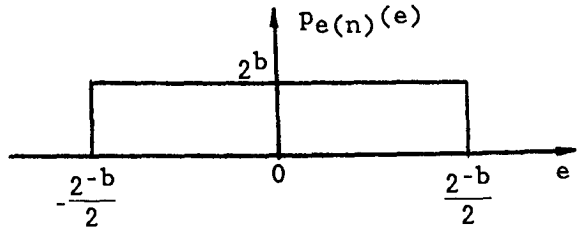



Figure 3.4 : Probability density function for amplitude of error samples.

and the signal-to-noise ratio due to quantization at the demodulator input is [8]

$$SNR_q = 10 \cdot \log_{10}(A^2 \sigma_r^2 / \sigma_e^2)$$

$$= 6b + 10 \cdot \log_{10}(12) + 10 \cdot \log_{10}(\sigma_r^2) + 20 \cdot \log_{10}(A) \qquad (3.3\text{-}5)$$

where $\sigma_r^2$ is the variance of the signal without any system noise at the A/D converter inputs and A is the scaling factor introduced to keep the signal plus system noise within the dynamic range of the A/D converters. The variance of an FSK signal with an amplitude normalized to unity (i.e. $E_s/T_s = 1/2$) is found from the autocorrelation function given by equation (2.5-8) with $\tau = 0$. The result is $\sigma_r^2 = 1/2$. If there are eight binary FSK signals with equal amplitudes at the demodulator input, then A must be less than 1/8 to keep the composite signal magnitude always less than one. If there is an additional factor of 1/3 included because of the possible signal variations due to system noise, then A = 1/24 and for a given user the signal-to-noise ratio due to quantization is

$$SNR_q = 6b - 19.8 \quad (dB). \qquad (3.3\text{-}6)$$

where $\sigma_r^2$ is equal to 1/2 for an FSK signal with an peak amplitude of 1. If there are only two FSK signals at the input then with the same assumptions

$$SNR_q = 6b - 7.8 \quad (dB). \qquad (3.3\text{-}7)$$

A convenient quantization level for which A/D converters are readily available is eight bits, for which case b = 7 and $SNR_q$ = 22.2 dB with eight users and $SNR_q$ = 34.2 dB with two users. Assuming the the total noise power due to both quantization and system noise is just the sum of the two individual powers the effective loss in signal-to-noise ratio resulting from quantization is

$$\text{loss (dB)} = 10 \cdot \log_{10}\left[\frac{SNR_i + SNR_q}{SNR_q}\right] \qquad (3.3\text{-}8)$$

where $SNR_i$ is the A/D converter input signal-to-system noise ratio. For example if $SNR_i$ is 16 dB for a given user then the eight bit quantization results in a loss in signal-to-noise ratio of 0.9 dB with eight users present or a loss of 0.07 dB with two users present. Measurements of the demodulator performance will be carried out with only two test signals present at the input because there is only enough equipment available to generate two FSK signals. Therefore eight bit quantization will impose a very small loss on the performance, but if the demodulator is put into use with the possibility of having the maximum eight input signals, smaller quantization levels would have to be considered if the eight bit quantization loss is higher than can be tolerated.

If in the future the number of signal bins and hence the number of users can be increased due to faster electronics the number of quantization bits needed to meet a given $SNR_q$ requirement would be larger not only due to the increase in the number of users but also due to the increase in noise power at the A/D inputs. For example if the demodulator processes 32 signal bins or up to 16 binary users, the scaling factor A would be 1/16 due to the number of users alone. Because the bandwidth of the IF and baseband filters must be doubled the noise variance at the A/D inputs also doubles and the factor of 1/3 used to allow for noise fluctuations in the previous examples would be changed to $1/(3\sqrt{2})$ resulting in $A = 1/(48\sqrt{2})$. For eight-bit quantization the signal-to-noise ratio would be $SNR_q = 13.1$ dB and the loss in SNR with a 16 dB signal-to-system noise ratio would be 4.7 dB. This illustrates how the effect of quantization on an individual user varies with the filter bandwidths and the number of other users present at the demodulator input.

Figure 3.5 is a block diagram of the demodulator input illustrating the processing used to generate the complex samples of the equivalent baseband representation of the input IF signal. There is a bandpass filter at the input to limit the amount of noise power passing to the mixers. This is necessary to make the noise process a narrowband process. The signal is then split into two channels, each being mixed down to baseband followed by lowpass filtering and A/D conversion. The lowpass filters have a 1 dB bandwidth of 336 kHz and a 3 dB bandwidth of 355 kHz and are phase matched $\pm 2°$ over the 1 dB bandwidth. The power splitter, 90 degree hybrid, mixers and lowpass filters have been selected with care to keep the phase difference between the two channels close to the ideal 90 degrees. The A/D converters are TDC1048E1C circuit boards made by TRW. These circuit boards have a wideband operational amplifier at the analog input with a potentiometer to control the d.c. offset of the amplifier output which allows any d.c. offset resulting from the analog mixers to be cancelled. This is important because one of the signaling tones is located directly on 0 Hz in the baseband signal and any d.c. offset resulting from the electronics would be equivalent to a co-channel interfering tone at the IF input. The complex samples are passed to a circuit board which contains all of the digital electronics for the demodulator.

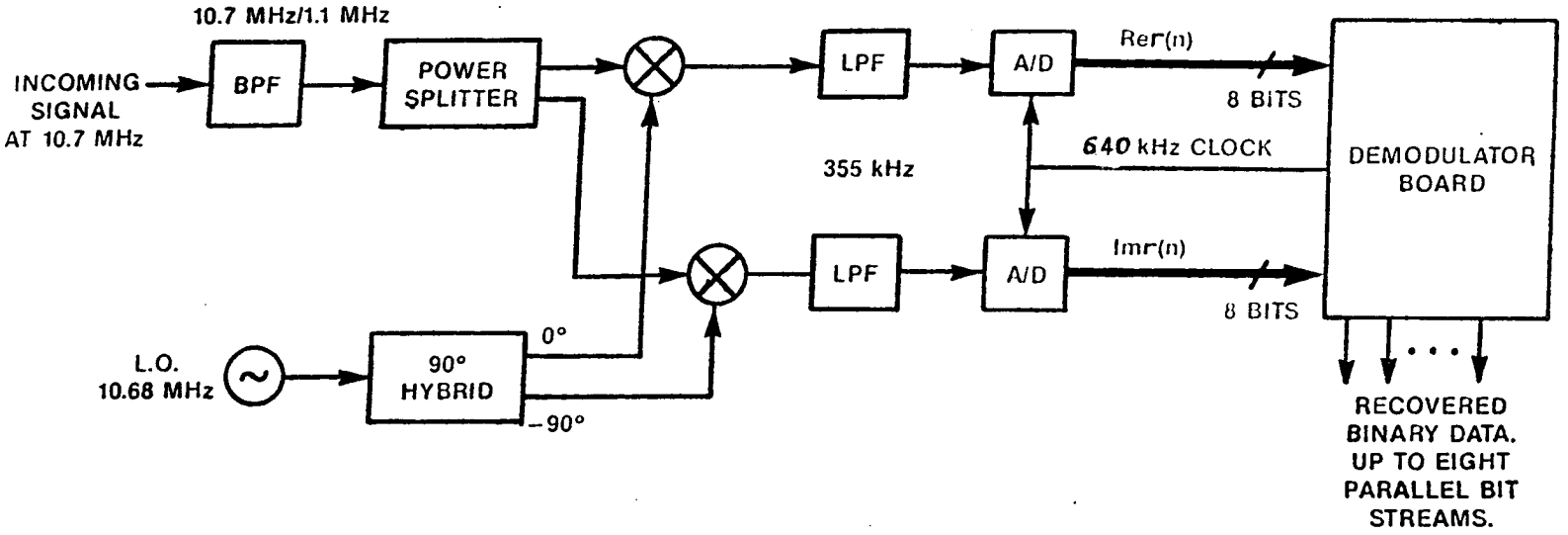



Figure 3.5: Block diagram of circuit used for generating the complex baseband samples.

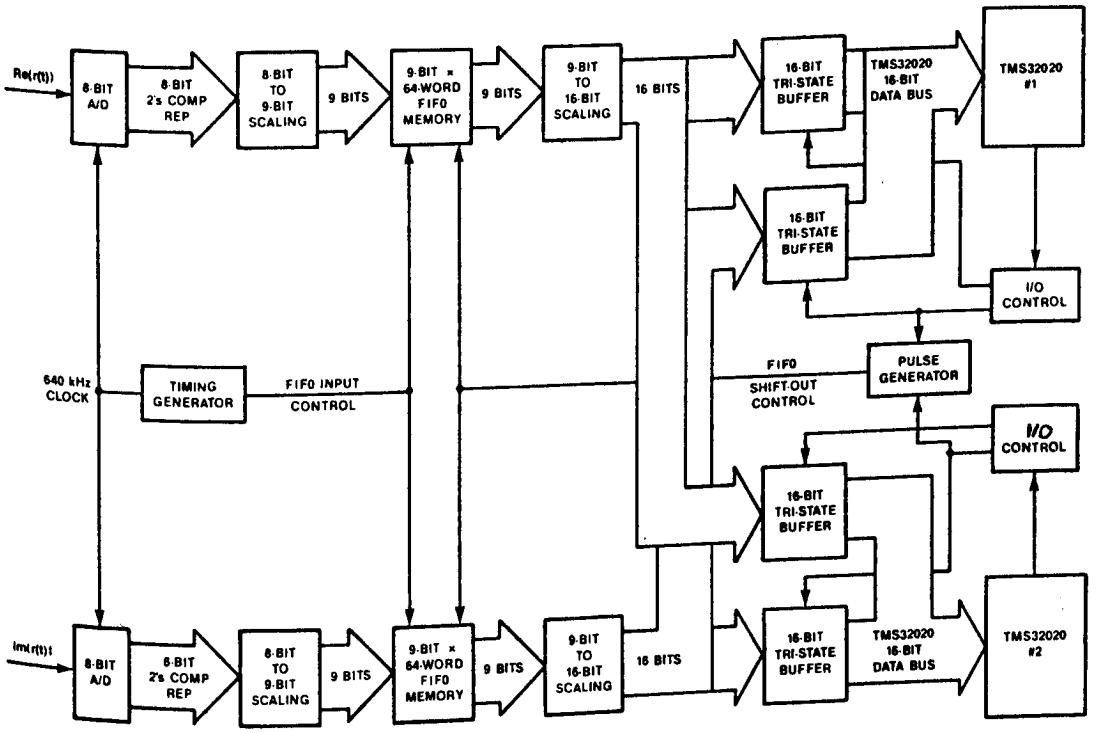
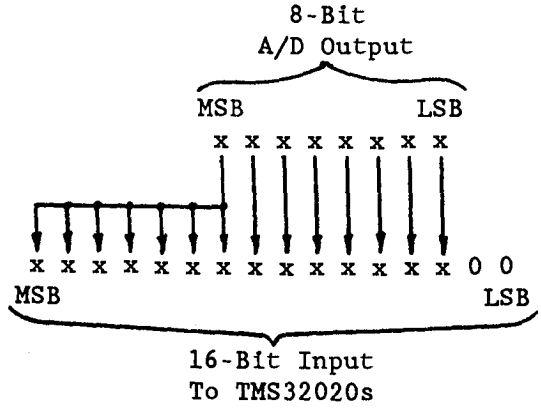



Figure 3.6 : Block diagram of the circuit interfacing the A/D converters and the TMS32020s.

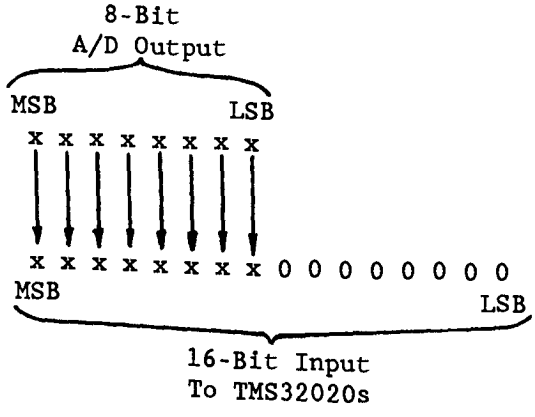
Figure 3.6 is a block diagram of the circuit used for passing the complex samples to the TMS32020s. The A/D converters generate eight bit two's complement samples which are stored over a symbol period in first-in first-out (FIFO) memories. An alternative to this configuration would be to have the microprocessors obtain the samples directly from the A/D converters, but the FIFO interface saves valuable processing time. Without the FIFOs 50 $\mu$s would be required for sample retrieval because the microprocessors would have to monitor the A/D converters and retrieve the samples as they are produced over the symbol period. There is only 1.56 $\mu$s between sampling instants which is not enough time to successfully use interrupt programming in which the TMS32020s carry out other tasks in the demodulation process such as windowing the samples which have already been retrieved while waiting for the next sample input. With the FIFOs all 64 real samples (i.e. 32 complex samples) are obtained by the TMS32020s with 64 successive instructions which takes only 12.5 $\mu$s.

Figure 3.6 also shows hardware scaling is required to convert the eight-bit two's complement samples produced by the A/D converters to the 16-bit form used by the TMS32020s. The simplest way to do this is to add

eight zeros onto the end of the eight-bit samples as shown in Figure 3.7(a). This is referred to here as the direct conversion because the 2's complement value of the sample remains the same after conversion. This method is not used for reasons to do with the fixed point arithmetic used by the TMS32020 and the possible growth in magnitude of the processed data while computing the DFT.




(a)




(b)

Figure 3.7 : Hardware scaling for converting 8-bit samples to a 16-bit format (a) direct and (b) divide by 64.

Parseval's theorem states that

$$\sum_{k=0}^{N-1} |R(k)|^2 = N \sum_{n=0}^{N-1} |r(n)|^2 \quad . \qquad (3.3\text{-}9)$$

Using equation (3.3-9) to compute an upper bound on the magnitude of a single output of 32-point DFT shows that if the real and imaginary parts of input sequence each have a maximum magnitude of one then the maximum squared magnitude of any particular DFT output is bounded by 2048. With 16-bit fixed point two's complement arithmetic, all numbers must have a magnitude less than $1 - 2^{-16} \approx 1$. One method that can be used to avoid overflow is to divide each of the input samples by $\sqrt{2048} = 45.25$. With binary represention it is more convenient to divide by powers of 2, or in this case divide each sample by 64 which can be achieved with simple hardware shifts. This can be accomplished while converting the eight-bit A/D outputs to the 16-bit format as shown Figure 3.7(b). Figures 3.8 and 3.9 are circuit diagrams of the data interface between the A/D converter boards and the two TMS32020s. Also shown in Figure 3.9 are circuits for generating a 640 kHz clock signal for the A/D converters and determining which of the resulting samples represent one 50 $\mu$s symbol period.

Once the microprocessors have obtained all 32 complex samples, windowing and the DFT must be performed. There are many algorithms available which greatly reduce the number of calculations required for determining the DFT. These algorithms are collectively referred to as the fast Fourier transform (FFT) and the choice of which algorithm to use depends on several factors, as discussed in the next section.

## 3.4 DFT ALGORITHM

This section deals with the algorithm used to compute the 16 even order outputs of the DFT given by

$$R(k) = \sum_{n=0}^{31} r(n)\exp(-j2\pi nk/32), \quad k = 0, 2, \cdots, 30. \qquad (3.4\text{-}1)$$

If the R(k)s are computed directly with equation (3.4-1) then 256 nontrivial complex multiplications and 496 complex additions would be required where multiplication by $e^{-ji\pi/2}$ with i an integer is considered trivial since no real operations are required. Since each complex multiply requires four real multiplications and two real additions and each complex add requires two real additions, the direct evaluation of equation (3.4-1) requires 1024 real multiplications and 1504 real additions. Obviously this number of operations cannot be carried out with only the 576 available instructions (64 of the total available 640 instructions must be used to obtain the 32 complex samples) and an FFT algorithm must be used which considerably reduces the required number of operations.

A common FFT algorithm which is very useful in this case is the decimation-in-frequency (DIF) algorithm. The first stage of the DIF algorithm [8] and [11] is obtained by dividing the input sequence {r(n)} into two subsequences so that

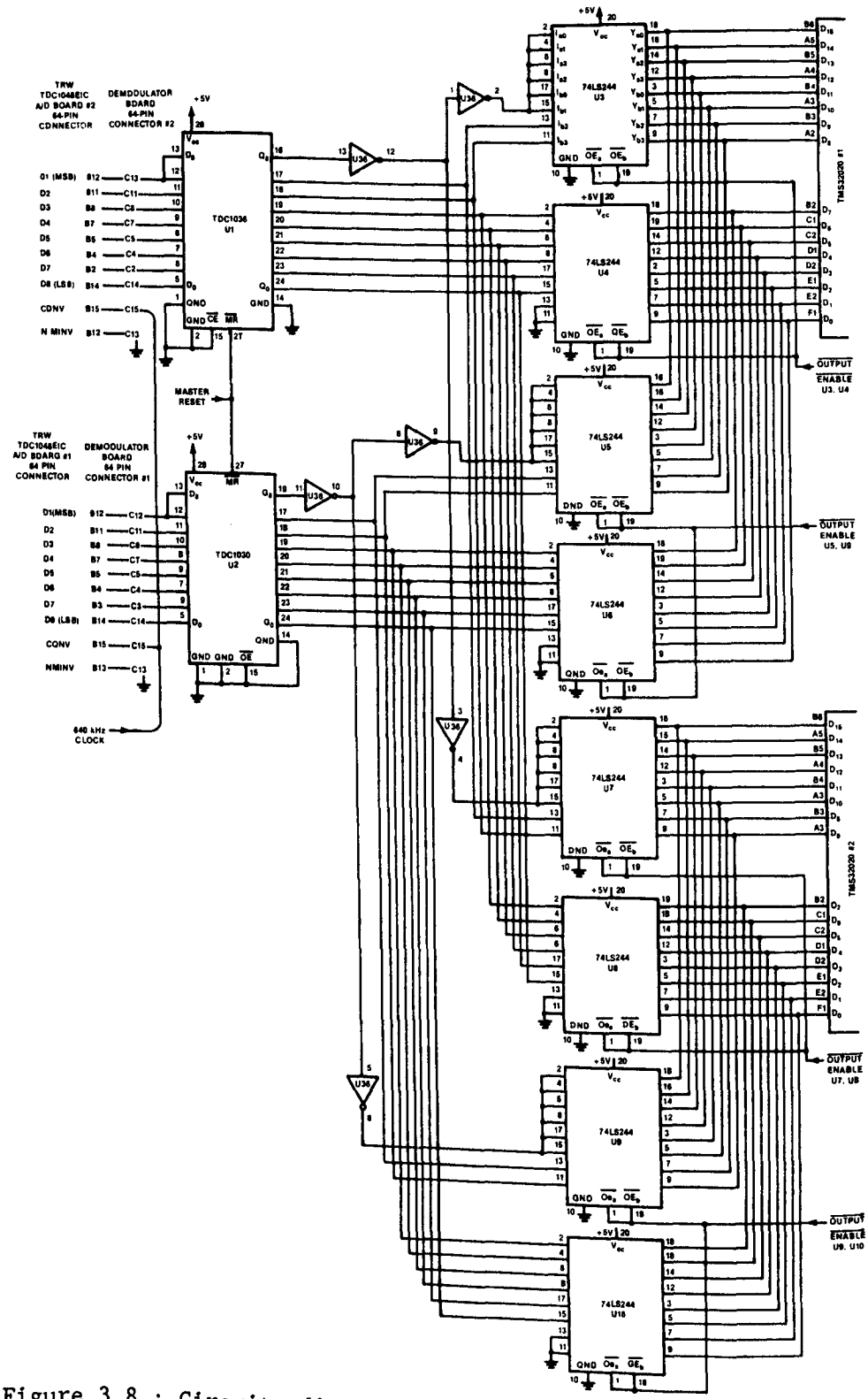



Figure 3.8 : Circuit diagram for the data interface between the A/D converters and the TMS32020s.

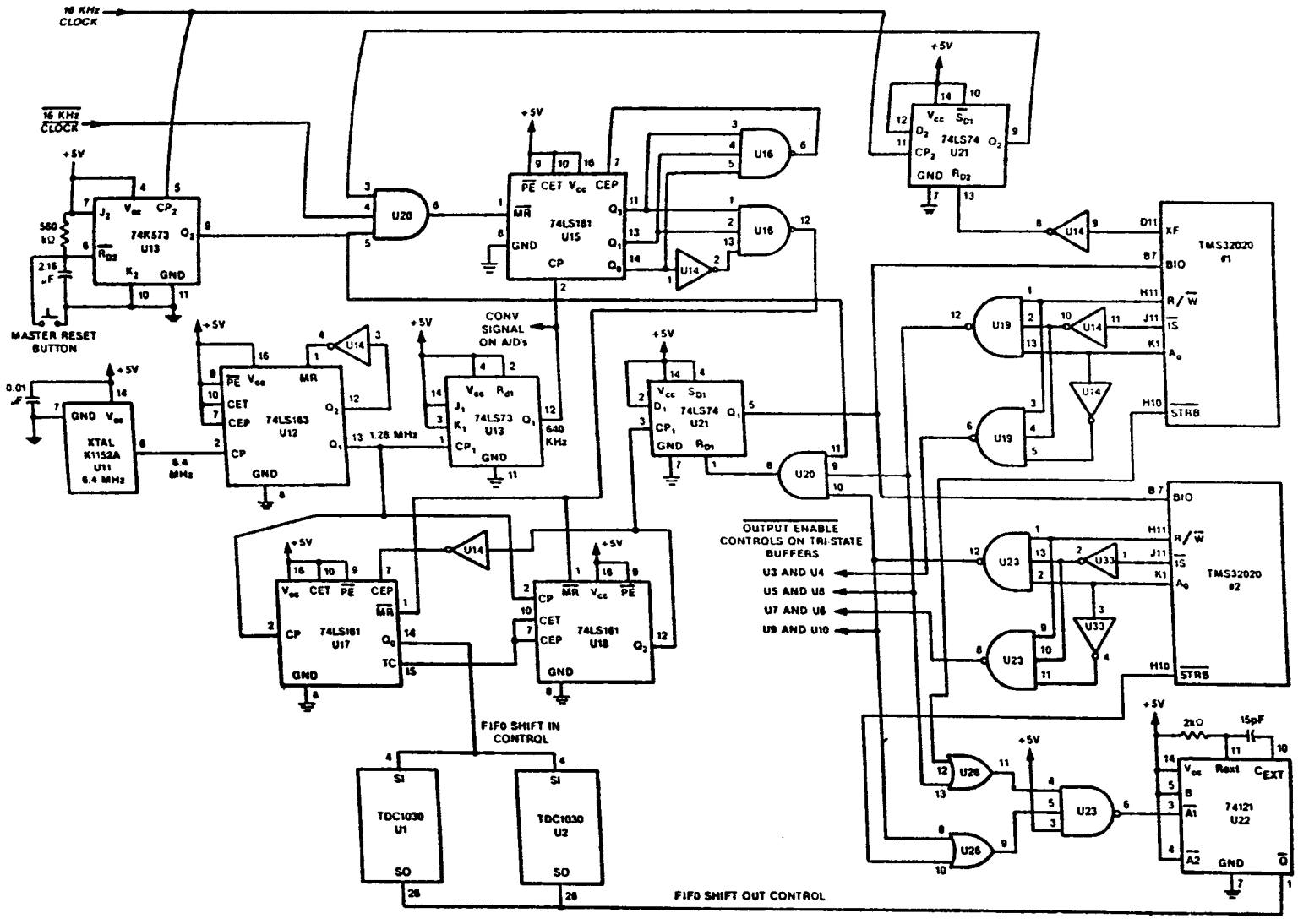



Figure 3.9: Circuit diagram of the clock generation and input control for the TMS32020s.

$$R(k) = \sum_{n=0}^{15} r(n)\exp(-j2\pi nk/32) + \sum_{n=16}^{31} r(n)\exp(-j2\pi nk/32)$$

$$= \sum_{n=0}^{15} r(n)\exp(-j2\pi nk/32) + \exp(-jk\pi)\sum_{n=0}^{15} r(n+16)\exp(-j2\pi nk/32)$$

$$= \sum_{n=0}^{15} [r(n) + (-1)^k r(n+16)]\exp(-j2\pi nk/32) \quad . \qquad (3.4\text{-}2)$$

Since k is even and can be expressed as k = 2p, equation (3.4-2) becomes

$$R(2p) = \sum_{n=0}^{15} [r(n) + r(n+16)]\exp(-j2\pi np/16)$$

$$= \sum_{n=0}^{15} x(n)\exp(-j2\pi np/16)$$

$$= X(p) \quad , \quad p = 0, 1, \cdots, 15 \qquad (3.4\text{-}3)$$

and the R(k)s can be obtained from the 16-point DFT of the sequence $\{x(n) = r(n) + r(n+16)\}$, n = 0, •••, 15. If the 16-point DFT is evaluated directly, computing the R(k)s requires 128 complex multiplies and 256 complex adds or 512 real multiplications and 768 real additions, or roughly half of the computations required for the direct evaluation of equation (3.4-1).

If the odd values of k are of interest then k = 2p+1 and equation (3.4-2) becomes

$$R(2p+1) = \sum_{n=0}^{15} \{[r(n) - r(n+16)]\exp(-j2\pi n/32)\}\exp(-j2\pi np/16) \quad ,$$

$$p = 0, 1, \cdots, 15 \qquad (3.4\text{-}4)$$

so the R(k)s for odd values of k can be obtained from the 16-point DFT of the sequence $\{[r(n) - r(n+16)]\exp(-j2\pi n/32)]\}$, n = 0, •••, 15. It was pointed out in section 3.3 that using the odd ordered DFT outputs would improve the demodulator error performance because the bandwidth of the inphase and quadrature channel filters could then be selected so that almost all of the signal energy would be passed to the A/D converters while the avoiding the aliasing of the noise spectrum. It is apparent from equation (3.4-4) that computing the DFT outputs for the odd values of k requires 14 more complex multipies or 56 more real multiplies and 28 more real additions than

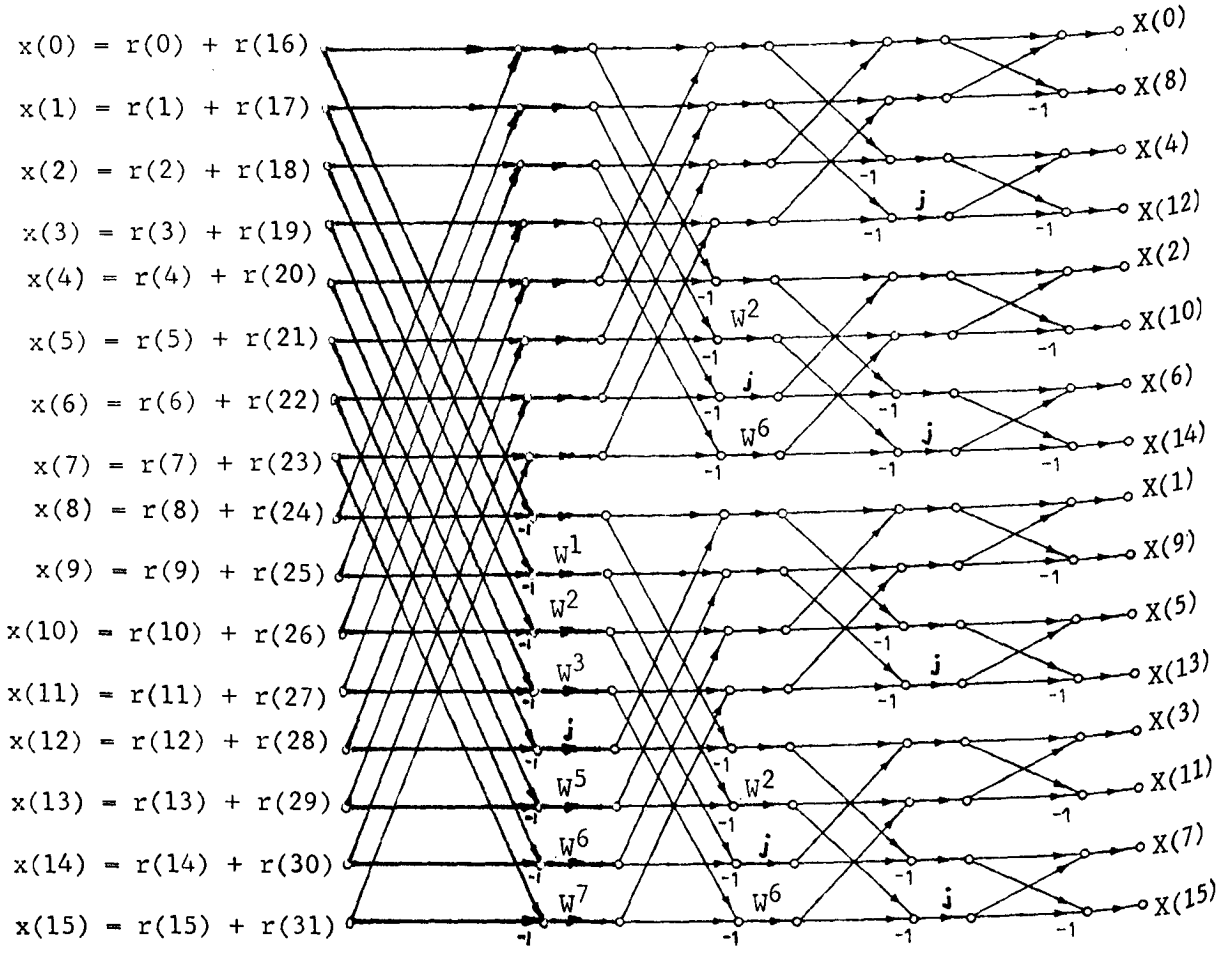
required for the even values of k, due to the $\exp(-j2\pi n/32)$ factors. Unfortunately these extra operations cannot be accomodated with the available instructions and the degradation associated with using the even ordered DFT outputs must be accepted.

Equations (3.4-3) and (3.4-4) illustrate how the DIF algorithm breaks the 32-point DFT down into two 16-point DFTs resulting in a reduction of required computations. Using similar steps the desired 16-point DFT can be broken down into two eight-point DFTs each of which are then broken down into two four-point DFTs which are further broken down into two-point DFTs. Figure 3.10 is a signal flow graph of the 16-point DIF algorithm for computing the even ordered DFT outputs given by equation (3.4-3). This algorithm requires only 10 nontrivial complex multiplications and 80 complex additions or 40 real multiplications and 180 real additions which is a considerable reduction in computation when compared to the direct calculation of equation (3.4-1).

There are many other algorithms which result in similar computational savings [8], [10], and [12]. The greatest savings always result when N is a power of two. If for example N is 34, using the DIF technique allows the 34-point DFT to be broken down into two 17-point DFTs, but no further and other algorithms must be used to compute the 17-point DFTs. There are algorithms which do reduce the computations required for a 17-point DFT but they do not produce nearly enough computational savings to meet the processing requirements of this demodulator. Thus the idea of sampling at a higher rate to avoid noise aliasing as described in section 3.3 cannot be used because of the increase in computation required for N greater than 32. The sampling rate will be left at 640 kHz and N at 32, and the first stage of the DIF algorithm will be used. We are then left with finding an algorithm which efficiently computes the 16-point DFT given by equation (3.4-3).

The DIF algorithm described above is referred to as a radix 2 algorithm because the DFT is successively broken down into several basic operations called butterflies (radix 2 butterflies in this case) which consist of a two-point DFT and a complex multiplication, as illustrated in Figure 3.10. One alternative algorithm is a radix 4 DIF algorithm with a signal flow graph representation as shown in Figure 3.11 in which the basic calculation is a radix 4 butterfly which consists of a four-point DFT and three complex multiplications. Using a radix 4 algorithm reduces the number of nontrivial multiplies at the expense of increasing the number of additions when compared to radix 2 algorithms. From Figure 3.11 the total number of calculations including the first radix 2 DIF stage is 8 complex multiplies and 112 complex additions or 32 real multiplies and 240 real additions. In some cases reducing the number of multiplies greatly reduces the processing time but since the TMS32020 has a hardware multiplier which allows multiplies to be done with a single instruction cycle, reducing multiplies while increasing the number of additions does not improve the processing speed.

From the total number of real operations it appears that the radix 2 algorithm should take less time to compute than the radix 4 algorithm, but one must consider more than just the total number of mathematical operations required. The number of memory references must be considered because each




$W = \exp(-j2\pi/16)$

Radix 2 Butterfly:

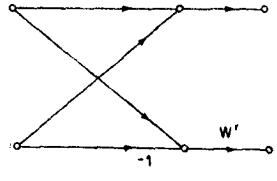
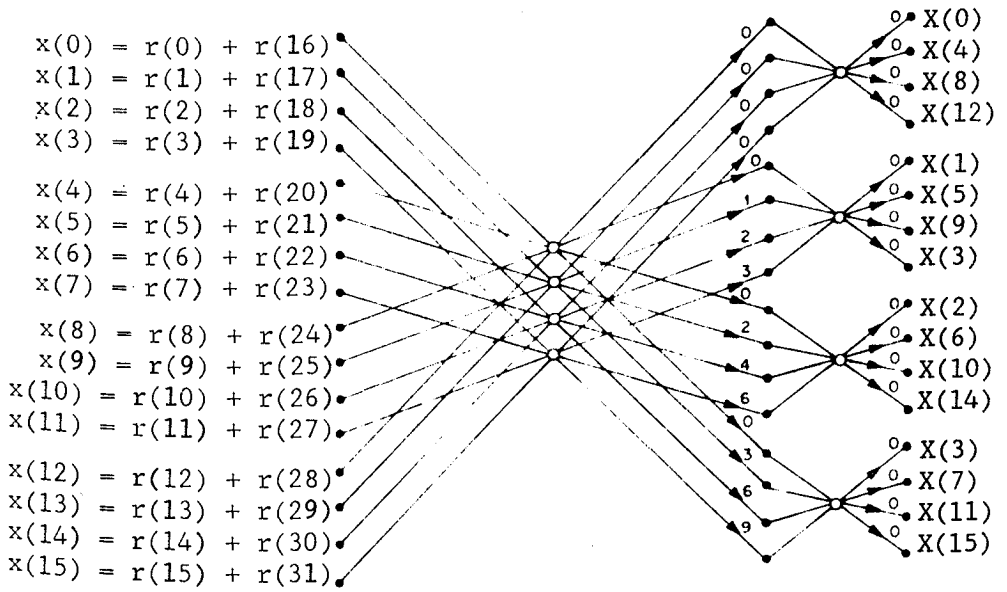
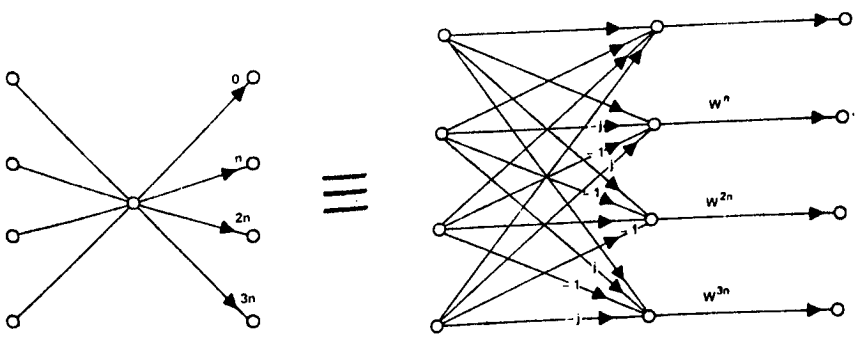



Figure 3.10 : Signal flow graph for the DIF algorithm.




Radix 4 Butterfly:




$W = \exp(-j2\pi/16)$

Figure 3.11 : Signal flow graph for the radix 4 DIF algorithm.

memory reference requires an instruction cycle with the TMS32020. The total number of memory references is the sum of the number of intermediate and final results which must be stored in memory and the number of calculations which begin with a load from memory. For example the calculation of $(x + y)\times z$ where x, y, and z are real numbers requires one instruction to load x from memory into the accumulator, a second instruction to add y to the accumulator, and a third to store the intermediate result $x' = x + y$ in memory before the multiplication can be carried out since there is no direct path from the output of the accumulator to the input of the multiplier on the TMS32020.

An upper limit on the number of stores required for an algorithm is obtained by simply counting the number of nodes or intermediate and final variables on the signal flow graph where any two nodes connected directly with a single line and a multiplier of $\pm 1$ or $\pm j$ are considered as a single node. An upper limit on the number of calculations which begin with a load from memory is similarly obtained. These limits would give a reasonable estimate of of the total number of memory references required when the size of the DFT is large (i.e. N is large) and the algorithm is implemented with a program which repeatedly loops through the same subroutine to perform each butterfly, but when processing time is of prime importance each butterfly is explicitly included as a seperate section of code and many of the memory references can be avoided because several of the intermediate variables do not need to be stored in and and reloaded from memory if they are used immediately in the next calculation.

From Figure 3.10 the radix 2 algorithm requires no more than 90 complex stores and 90 complex loads or a total of 360 real memory references while from Figure 3.11 the radix 4 algorithm requires no more than 56 complex intermediate stores and 56 complex loads or 224 real memory references. As mentioned these are only upper bounds and the real number of memory references may be considerably less. Thus to see which algorithm can be implemented with the fewest number of instruction cycles, the algorithm must be expressed as a program or flow chart so that each operation, be it a mathematical operation or a memory reference can be counted. Preliminary programs were written for both the radix 2 and radix 4 DIF algorithms with the windowing of the input sequence included. With these programs it was estimated that 463 instruction cycles would be required for the radix 2 algorithm and 478 instructions cycles would be required for the radix 4 algorithm and hence the radix 2 algorithm should take up less processing time. Other forms of radix 2 and radix 4 algorithms were considered and they all resulted in roughly the same instruction requirements as for the DIF algorithms. Unfortunately allocating 64 instructions for the input and 463 instructions for the windowing and the DFT does not leave enough instructions for the decision section of the demodulator software which may require up to 125 instruction cycles and a more efficient algorithm must be found.

There exist DFT algorithms derived with the use of number theory which reduce the required number of multiplies while keeping the number of additions about the same as required for a radix 2 algorithm [12] and [13]. An algorithm for $N = 16$ originally described in [14] and reproduced in Figure 3.12 from [12] requires 20 nontrivial real multiplies and 180 real additions

$N = 16$: $u = \frac{2}{16}\pi$.

$$t_1 = x(0) + x(8), \quad t_2 = x(4) + x(12), \quad t_3 = x(2) + x(10)$$
$$t_4 = x(2) - x(10), \quad t_5 = x(6) + x(14), \quad t_6 = x(6) - x(14)$$
$$t_7 = x(1) + x(9), \quad t_8 = x(1) - x(9), \quad t_9 = x(3) + x(11)$$
$$t_{10} = x(3) - x(11), \quad t_{11} = x(5) + x(13), \quad t_{12} = x(5) - x(13)$$
$$t_{13} = x(7) + x(15), \quad t_{14} = x(7) - x(15), \quad t_{15} = t_1 + t_2$$
$$t_{16} = t_3 + t_5, \quad t_{17} = t_{15} + t_{16}, \quad t_{18} = t_7 + t_{11}$$
$$t_{19} = t_7 - t_{11}, \quad t_{20} = t_9 + t_{13}, \quad t_{21} = t_9 - t_{13}$$
$$t_{22} = t_{18} + t_{20}, \quad t_{23} = t_8 + t_{14}, \quad t_{24} = t_8 - t_{14}$$
$$t_{25} = t_{10} + t_{12}, \quad t_{26} = t_{12} - t_{10}$$
$$m_0 = 1 \times (t_{17} + t_{22}), \quad m_1 = 1 \times (t_{17} - t_{22})$$
$$m_2 = 1 \times (t_{15} - t_{16}), \quad m_3 = 1 \times (t_1 - t_2)$$
$$m_4 = 1 \times [x(0) - x(8)], \quad m_5 = (\cos 2u) \times (t_{19} - t_{21})$$
$$m_6 = (\cos 2u) \times (t_4 - t_6), \quad m_7 = (\cos 3u) \times (t_{24} + t_{26})$$
$$m_8 = (\cos u + \cos 3u) \times t_{24}, \quad m_9 = (\cos 3u - \cos u) \times t_{26}$$
$$m_{10} = j \times (t_{20} - t_{18}), \quad m_{11} = j \times (t_5 - t_3)$$
$$m_{12} = j \times (x(12) - x(4)), \quad m_{13} = (-j \sin 2u) \times (t_{19} + t_{21})$$
$$m_{14} = (-j \sin 2u) \times (t_4 + t_6), \quad m_{15} = (-j \sin 3u) \times (t_{23} + t_{25})$$
$$m_{16} = j(\sin 3u - \sin u) \times t_{23}, \quad m_{17} = -j(\sin u + \sin 3u) \times t_{25}$$
$$s_1 = m_3 + m_5, \quad s_2 = m_3 - m_5, \quad s_3 = m_{11} + m_{13}$$
$$s_4 = m_{13} - m_{11}, \quad s_5 = m_4 + m_6, \quad s_6 = m_4 - m_6$$
$$s_7 = m_8 - m_7, \quad s_8 = m_9 - m_7, \quad s_9 = s_5 + s_7$$
$$s_{10} = s_5 - s_7, \quad s_{11} = s_6 + s_8, \quad s_{12} = s_6 - s_8$$
$$s_{13} = m_{12} + m_{14}, \quad s_{14} = m_{12} - m_{14}, \quad s_{15} = m_{15} + m_{16}$$
$$s_{16} = m_{15} - m_{17}, \quad s_{17} = s_{13} + s_{15}, \quad s_{18} = s_{13} - s_{15}$$
$$s_{19} = s_{14} + s_{16}, \quad s_{20} = s_{14} - s_{16}$$
$$X(0) = m_0, \quad X(1) = s_9 + s_{17}, \quad X(2) = s_1 + s_3$$
$$X(3) = s_{12} - s_{20}, \quad X(4) = m_2 + m_{10}, \quad X(5) = s_{11} + s_{19}$$
$$X(6) = s_2 + s_4, \quad X(7) = s_{10} - s_{18}, \quad X(8) = m_1$$
$$X(9) = s_{10} + s_{18}, \quad X(10) = s_2 - s_4, \quad X(11) = s_{11} - s_{19}$$
$$X(12) = m_2 - m_{10}, \quad X(13) = s_{12} + s_{20}, \quad X(14) = s_1 - s_3$$
$$X(15) = s_9 - s_{17}$$

Figure 3.12 : Small N DFT algorithm for N = 16 [12].

Figure 3.12 : Small N DFT algorithm for N = 16 [12].

(including the additions required for the first DIF stage). The reduction in computation provided by this algorithm is enough to meet the requirements for the demodulator. The algorithm is implemented in a TMS32020 assembler language program listed in the appendix. The windowing is combined with the DFT in this program, and the total number of instruction cycles required for the windowing and the DFT is 424, which leaves 152 instruction cycles for the decisions and outputs described in section 3.6. The window function can be changed by simply changing the windowing coefficients stored in program memory. The DFT program has been tested with various window functions as described in the next section.

## 3.5 DFT PERFORMANCE AND RECEIVER WINDOWING

A software simulator was used to test the DFT section of the program in the appendix. The simulator uses 16-bit fixed point arithmetic and acts exactly as the TMS32020 would except the speed is much slower. Before presenting the simulation results it is worthwhile to discuss the effect of the fixed point arithmetic on the DFT calculations. With fixed point calculations, errors occur in additions or subtractions only when there is overflow in which the magnitude of the result exceeds the range of fractions which can be represented (approximately one) with the number of bits used. As mentioned previously, the input samples are effectively scaled by 1/64 when converted from eight-bit samples to the 16-bit representation used by the TMS32020. This prevents the occurence of any overflow in the DFT calculations and the only errors in the calculations result from the multiplications.

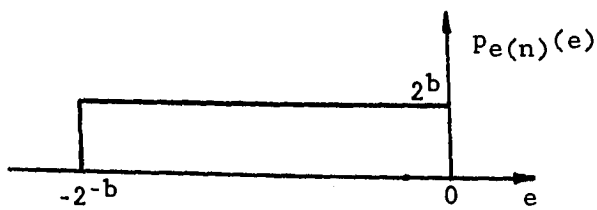



Figure 3.13 : Probability density function for two's complement truncation error.

When two 16-bit numbers are multiplied the resulting product is a 32-bit number. With the TMS32020 the 32-bit product is eventually truncated to 16 bits when it is stored in memory. As with the quantization noise at the input the error due to truncation can be treated as additive noise with error samples characterized by the probability density function shown in Figure 3.13 [8]. The difference between the input rounding quantization and this truncation noise is that for two's complement truncation the resulting errors have nonzero mean. The truncation error after each multiplication will have a mean $m_e = -2^{-b}/2$ and variance $\sigma_e^2 = 2^{-2b}/12$ with b = 15 in this

case. This variance is for real multiplications, and since for this particular algorithm each multiplication shown in the algorithm corresponds to two real multiplications the variance of the complex signal after multiplication will be $2^{-2b}/6$, where it has been assumed that the truncation errors for the real and imaginary parts of the product are uncorrelated.

For the algorithm given in Figure 3.12, since there are no multiplies in the calculations leading to the DFT outputs of X(0), X(4), X(8) and X(12) these particular outputs have no additive noise due to truncation. The outputs X(2), X(6), X(10) and X(14) each have two multiplies associated with their calculation, while the odd ordered outputs each have six multiplications associated with their calculation. The variance of the complex truncation noise at the DFT outputs is then 0, $2^{-2b}/3$ or $2^{-2b}$ depending which output is being considered. The truncation noise at each output will have a mean value depending on how the various contributing multiplication outputs are added or subtracted. From Figure 3.12 the worst possibility is that the real and imaginary parts of the truncation noise at a particular output will each be $4m_e$, and therefore the worst case mean square value of the truncation noise is

$$E\left\{|e_k|^2\right\}_{max} = 6\times 2\sigma_e^2 + 2(4m_e)^2$$

$$= 9\times 2^{-2b} \qquad (3.5\text{-}1)$$

where $e_k$ is the cumulative error due to truncation at the $k^{th}$ DFT output. If the DFT input sequence is $(6\times 64)^{-1}\cos(2\pi f_k nT)$ where the factor 1/6 is the scaling before sampling to avoid A/D converter saturation and 1/64 is the DFT input scaling to avoid overflow in the calculations then the $k^{th}$ DFT output with no quantization or truncation errors will have a mean square value of $[32(6\times 64)^{-1}]^2 = (1/12)^2$ and if the worst case signal-to-truncation noise ratio is taken as the ratio of the mean square signal level over the worst case mean square truncation noise level then

$$SNR_t = 10\cdot\log_{10}\left[\frac{2^{2b}}{9(12)^2}\right]$$

$$= 6b - 31.1 \text{ dB} . \qquad (3.5\text{-}2)$$

For $b = 15$ the signal-to-noise ratio due to truncation is 58.9 dB. If the scaling factor of 1/6 is changed to 1/24 to allow for the possibility of eight users' signals at the demodulator input the signal-to-noise ratio works out to be 46.8 dB, and in either case the effect of truncation in the calculation of the DFT is negligible compared to the effect of the eight-bit quantization of the A/D converters (recall that $SNR_q = 34.2$ dB with two FSK signals at the demodulator input and $SNR_q = 22.2$ dB with eight FSK signals at the demodulator input).

The previous analysis is not valid for nonrectangular windowing. If the window coefficients are not all one then the multiplication by the window sequence at the input to the DFT will result in additional truncation

errors. Then each output will have at least 31 and possibly up to 37 multiplications associated with the calculation of the real and imaginary parts. The worst error will be associated with X(0) because the mean value of the real and imaginary parts of the output error due to truncation will each be $31m_e$. Then the worst case mean square value of the truncation noise ratio is

$$E\left\{|e_k|^2\right\}_{max} = 31\times 2\sigma_e^2 + 2(31m_e)^2$$

$$= 485.7\times 2^{-2b} \qquad (3.5\text{-}3)$$

and the worst case signal-to-truncation noise ratio with two FSK input signals is

$$SNR_t = 6b - 48.4 \text{ dB} . \qquad (3.5\text{-}4)$$

For $b = 15$ and nonrectagular windowing equation (3.5-4) results in $SNR_t = 41.6$ dB. Using equation (3.3-8) with an input signal-to-system noise ratio of 16 dB and $SNR_q$ replaced by $SNR_t$, the loss in signal-to-noise ratio due to truncation noise is 0.01 dB which is still small when compared to the effect of the input quantization noise.

In order to test the DFT algorithm a BASIC program has been written which generates 32 complex samples of a sinusoid with phase and frequency as input variables. The samples are quantized to eight bits and then converted to 16-bit samples so that the output samples are identical to those that would be presented to the microprocessors by the A/D converters with a pure sinusoid at the demodulator input. A few extra instructions were added to the DFT section of the program in the appendix such that the square magnitude of the DFT output X(0) is calculated and stored as an output by the simulator. Using the TMS32020 simulator $|X(0)|^2$ was calculated with the modified DFT program with the frequency of the sampled input sinusoidal sequence varying from -320 kHz to +320 kHz. The results for various windowing functions are presented in Figures 3.14 - 3.18 and Table 3.2.

A comparison of the theoretical window parameters given in Table 2.1 and the simulation measurements shows that the eight-bit A/D quantization, the 16-bit register length of the TMS32020 and the 16-bit representation of the window coefficients have very little noticable effect on the window parameters and the measured DFT filter shapes. The measured 3 dB bandwidth of the main lobe is identical to that predicted in Table 2.1 for all of the windows. The sidelobe levels are as predicted in theory for peak levels less than 30 dB down from the peak of the mainlobe, but as the predicted levels drop below 30 dB, differences begin to arise due to the quantization and truncation effects. For the Kaiser-Bessel window with $\alpha = 1.8$ the difference between the predicted and measured highest sidelobe level is only 0.9 dB which is the largest difference for all of the windows listed in Table 3.2.

Now that the sampling, windowing and DFT sections of the demodulator implementation have been discussed and their nonideal effects characterized

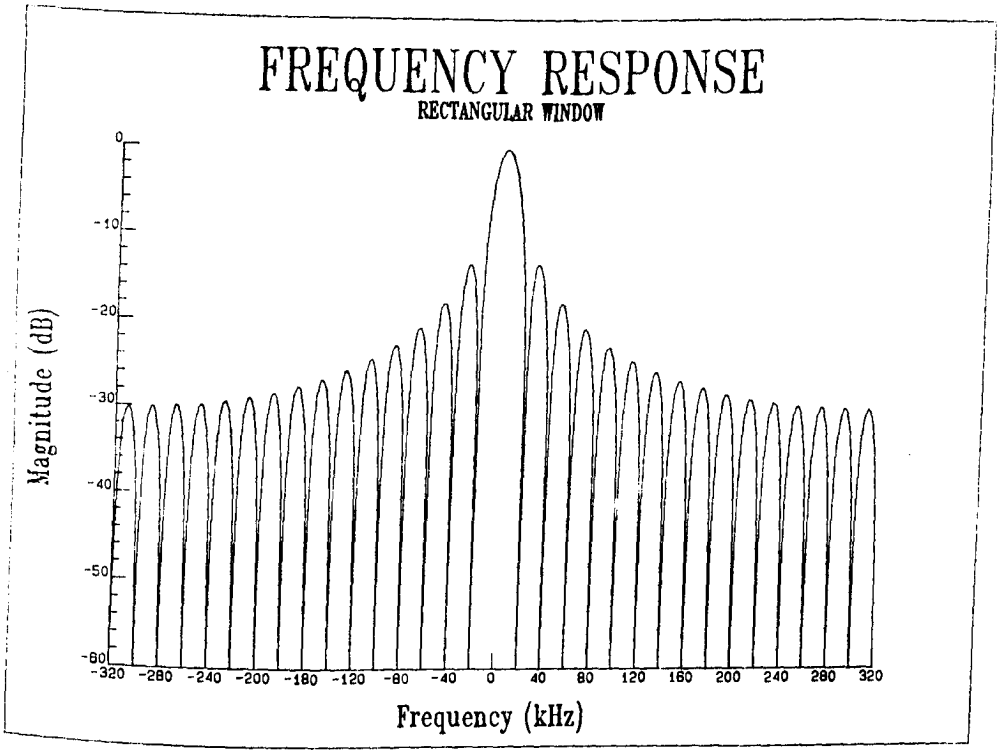



Figure 3.14 : TMS32020 DFT frequency response with a rectangular window.

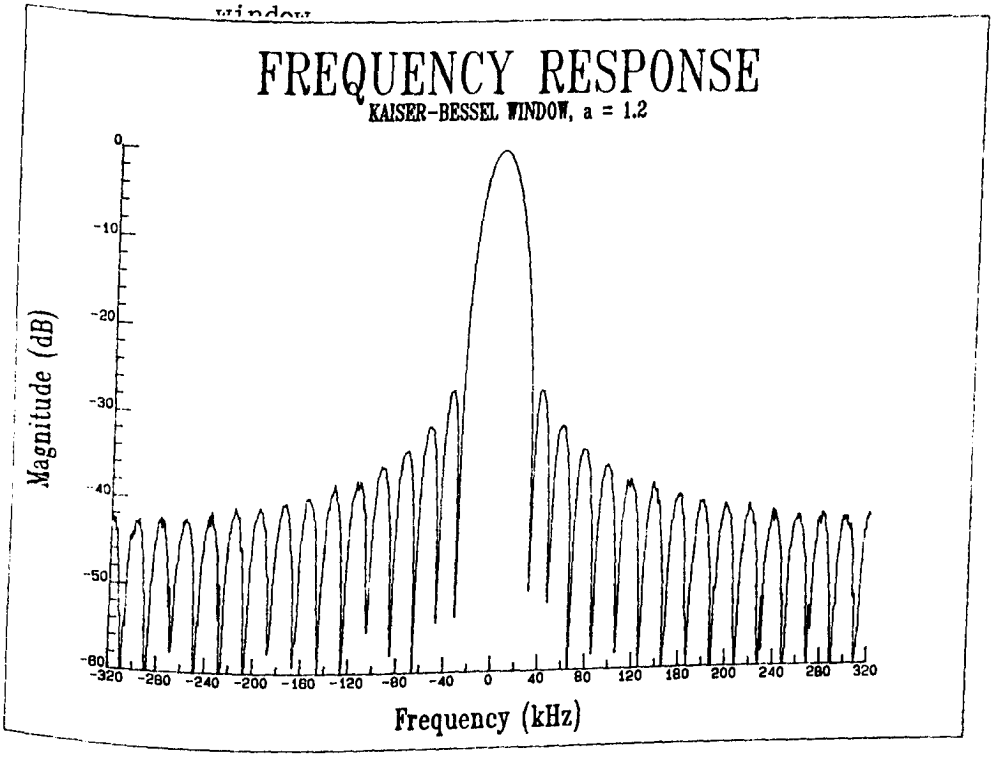



Figure 3.15 : TMS32020 DFT frequency response with a Kaiser-Bessel window, $\alpha$ = 1.2.

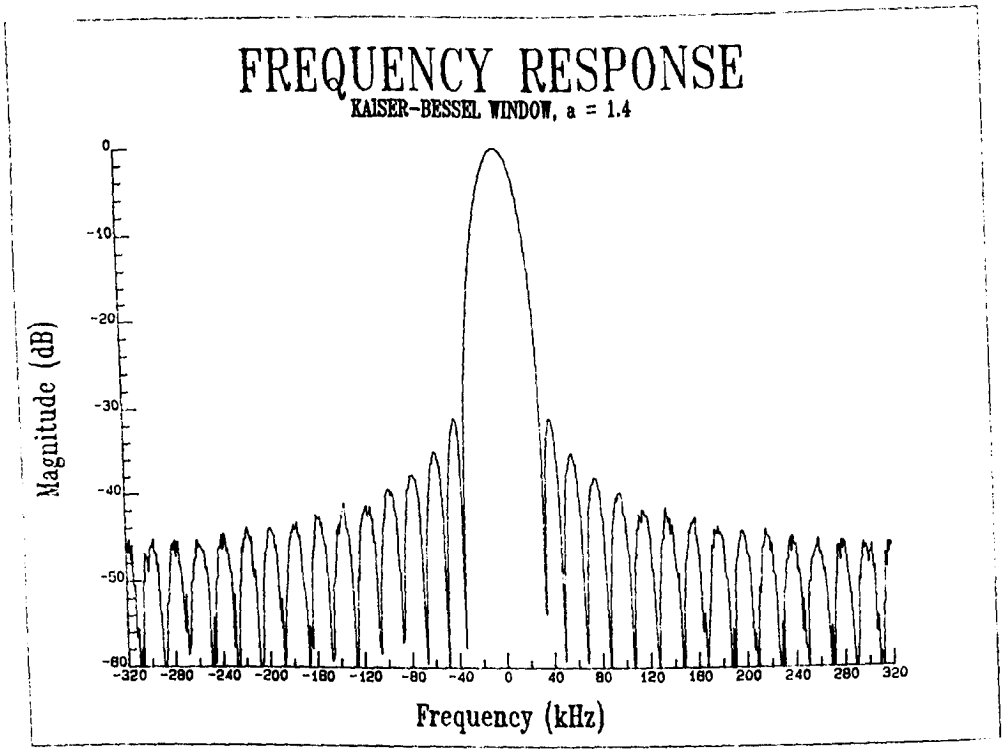



Figure 3.16 : TMS32020 DFT frequency response with a Kaiser-Bessel window, $\alpha = 1.4$.

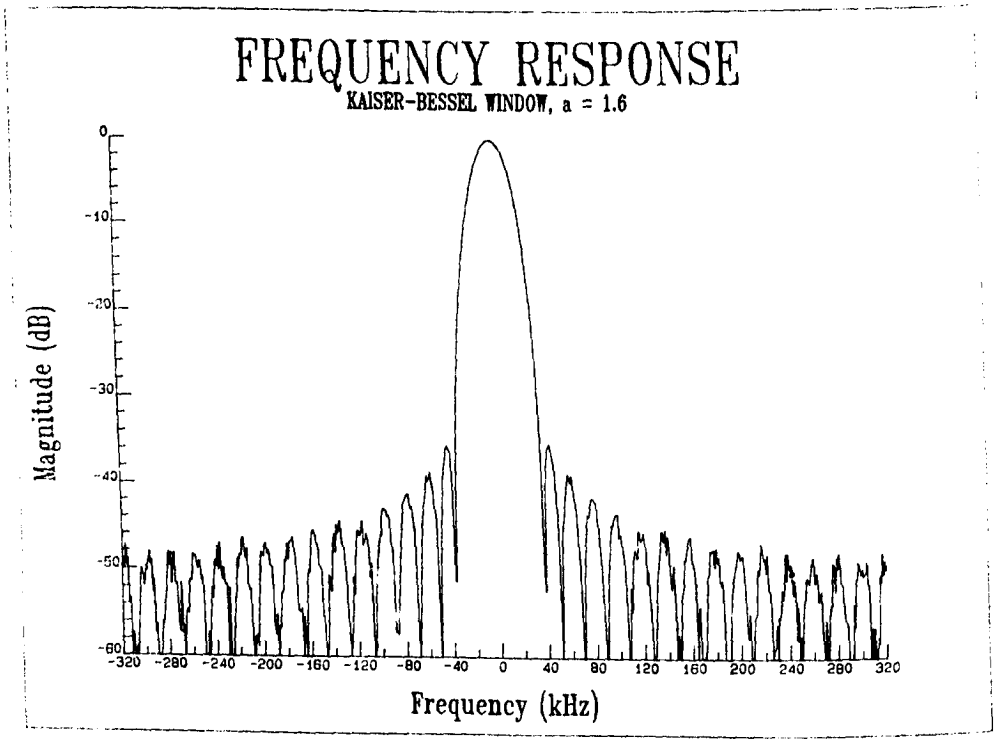



Figure 3.17 : TMS32020 DFT frequency response with a Kaiser-Bessel window, $\alpha = 1.6$.

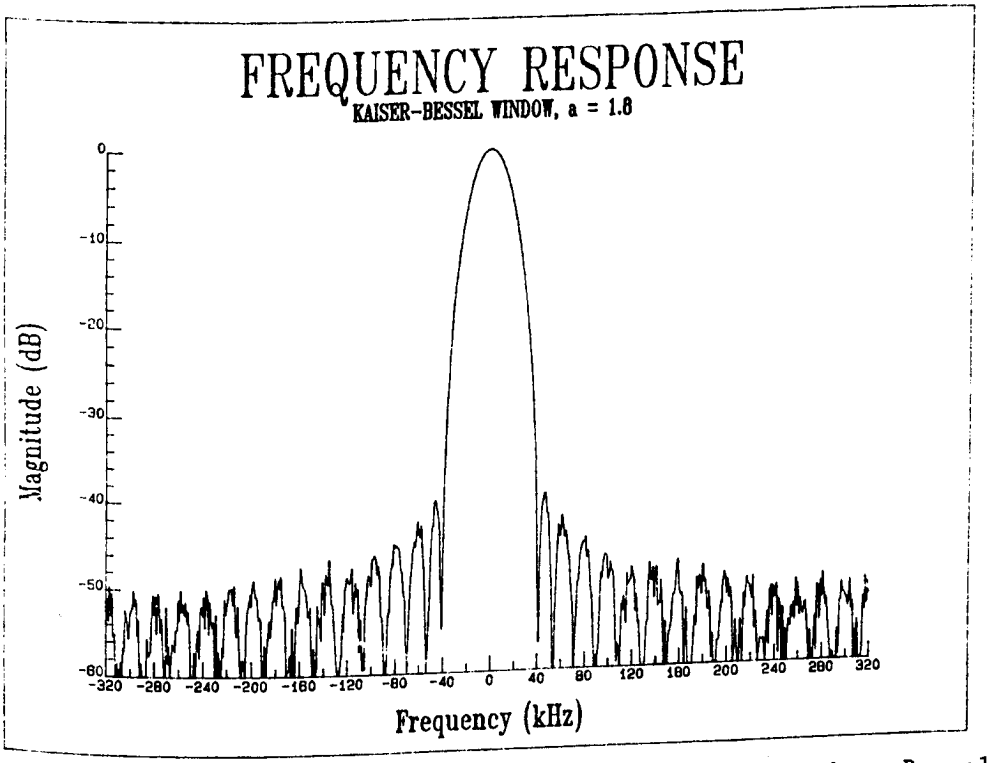



Figure 3.18 : TMS32020 DFT frequency response with a Kaiser-Bessel window, $\alpha$ = 1.8.

| WINDOW | | HIGHEST SIDELOBE (dB) | 3 dB BANDWIDTH (÷ $1/T_s$ Hz) |
|---|---|---|---|
| Rectangular | | -13.2 | 0.88 |
| Kaiser-Bessel | $\alpha$ = 1.2 | -27.3 | 1.19 |
| | $\alpha$ = 1.4 | -31.1 | 1.25 |
| | $\alpha$ = 1.6 | -35.2 | 1.31 |
| | $\alpha$ = 1.8 | -39.5 | 1.37 |

Table 3.2 : Comparison of measured window parameters.

the only function left to describe is the decision and output section of this particular implementation. This is the subject of the next section.

## 3.6 DECISIONS AND DATA REGENERATION

From equation (2.5-22) the decision variables are just scaled versions of the DFT outputs. Because the scaling is the same for all of the decision variables the factors $\sqrt{2E_s/T_s}$ can be ignored and the decisions will be made based directly on the DFT outputs. The demodulator then must determine for each user which of M DFT outputs has the largest magnitude. Since the decision variable with the largest magnitude will also have the largest squared magnitude, it makes no difference in the error performance whether the demodulator compares the M magnitudes or squared magnitudes. Therefore the process of finding the square roots can be avoided and the decision section of the program will make comparisons of the squared magnitudes of the DFT outputs.

To simplify the decision section it has been assumed that M is the same for each user and separate programs have been written for M = 2, 4, 8, and 16. For binary decisions the demodulator simply computes for each user the difference in the squared magnitudes of the two DFT outputs of interest. From the sign of the difference the demodulator determines for each user whether the transmitted symbol was a one or a zero. For M greater than 2 the demodulator compares two of the decision variables, makes an intermediate decision and then compares the larger of the two with a third decision variable, repeating the process until all of the decision variables have been exhausted for that particular user. In this manner the demodulator determines which frequency was transmitted for each user and hence which symbol was transmitted.

For each symbol period, if there are eight binary users a total of eight bits are required to represent the recovered data for all of the users. Similarly for four 4-ary users eight bits are required to represent the two bits per symbol for all of the users, for two 8-ary users six bits are required and for a single 16-ary user four bits are needed for the demodulator output. The demodulator will use an eight bit code to present the recovered M-ary symbols in parallel as a single output. The TMS32020 assembler language program listed in the appendix contains the section of code used for decisions and generating the output code for 4-ary FSK users. The decision sections of the demodulator software are similar for the cases of M = 2, 8 or 16. The maximum number of instructions required is for M = 16 in which case the decision and output section may require up to 125 instruction cycles to execute. The maximum number in instruction cycles required for the demodulator software is made up of 64 cycles for input, 424 cycles for windowing and the DFT, 125 cycles for decisions and output, and 6 cycles for loop and branch control for a total 619 instruction cycles. This leaves 21 unused cycles in the 125 $\mu$s processing period, which is not enough cycles to eliminate the noise aliasing problem as discussed in section 3.2.

Figure 3.19 is a block diagram of the circuit which takes the outputs from the two TMS32020s and presents the output codes to the individual

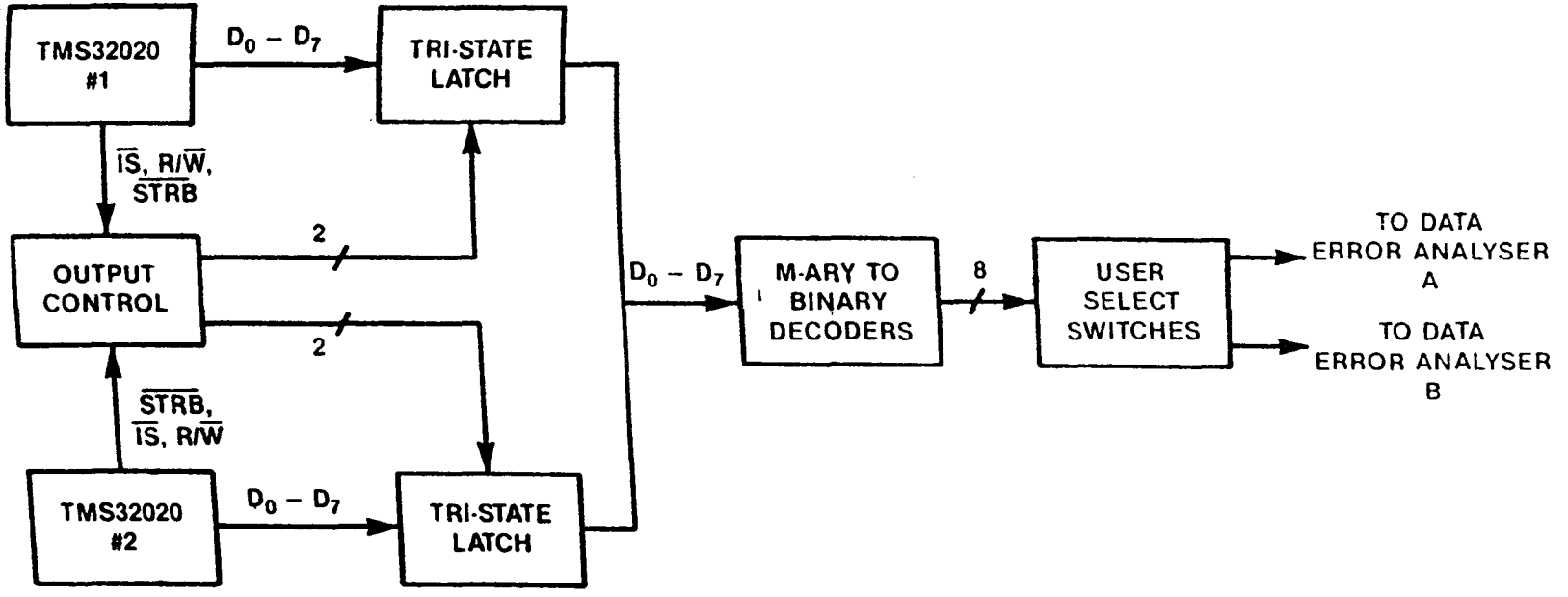



Figure 3.9: Block diagram of circuit at the TMS32020 outputs for data regeneration.

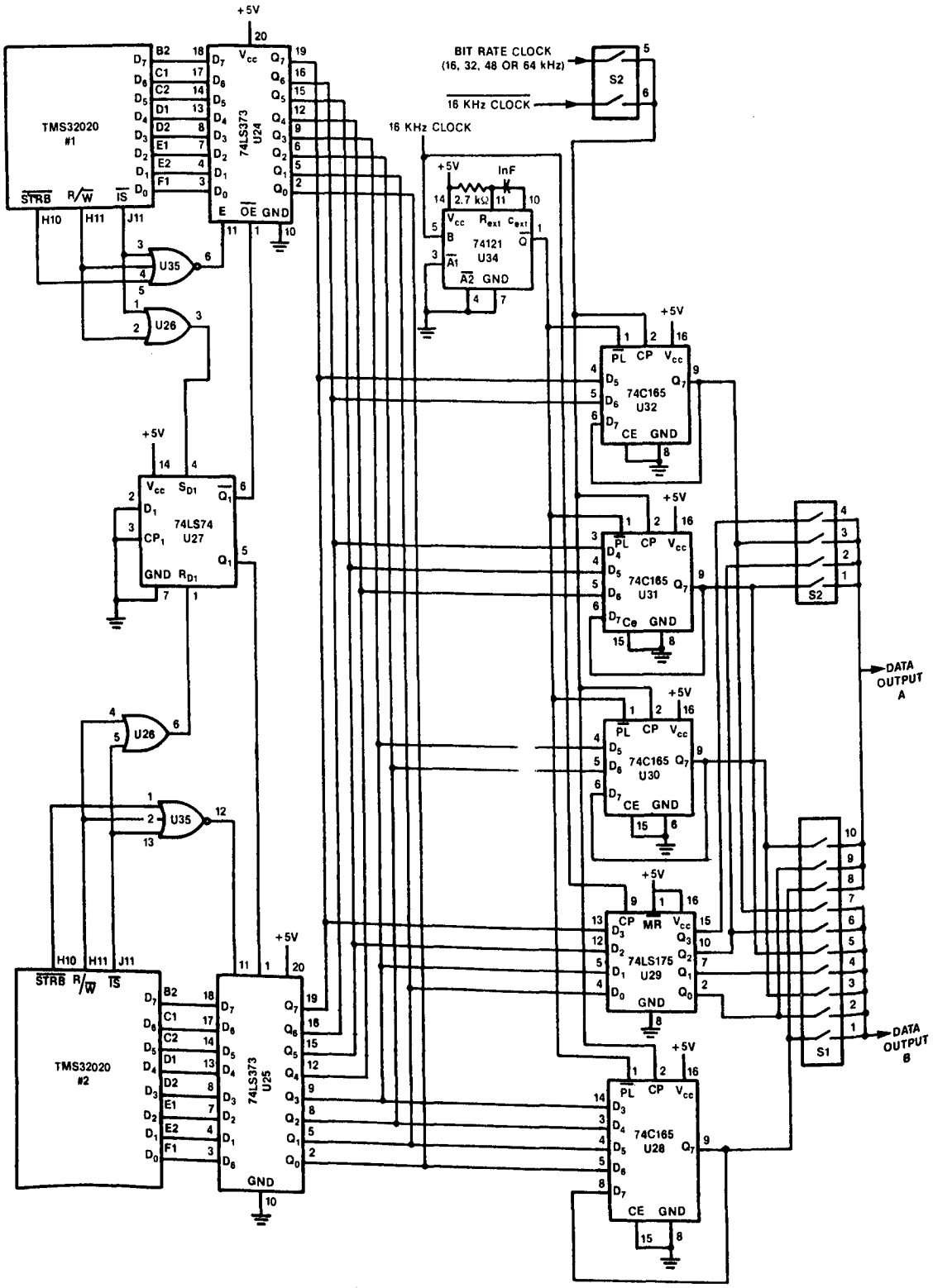



Figure 3.20 : Circuit diagram of the data regeneration circuits.

decoders for each user. Each user will have a $\log_2 M$ bit shift register which if necessary converts the M-ary symbols to binary data and synchronizes the data with a local data rate clock. Figure 3.20 is a circuit diagram of the electronics used to implement the block diagram of Figure 3.19. For testing purposes switches have been included which allow a given user's output to be routed to one of the two output connectors on the demodulator circuit board.

The majority of the hardware used for the demodulator implementation has been presented in the circuit diagrams of Figures 3.5, 3.8, 3.9, and 3.20. The only hardware that has not yet been mentioned is the simple interface between program read-only memory (ROM) and the two TMS32020s which is shown in Figure 3.21. The ROMs used are erasable so that the windowing function can be changed simply by reprogramming the ROMs with the new window coefficients. Thus the description of the demodulator implementation is complete.

## 3.7 SUMMARY

The design and implementation of a digital block demodulator has been discussed in this chapter, drawing on the theory presented in chapter 2. The choice of a symbol rate and other parameters for the demodulator was discussed in section 3.2. The sampling and quantization effects were characterized in section 3.3 followed by a discussion of DFT algorithms in section 3.4. Simulations of the DFT response with various windows were presented in section 3.5 and the demodulator decision making and output have been discussed in this final section, completing chapter 3. The demodulator has been tested and the results of various measurements are presented in the next chapter.

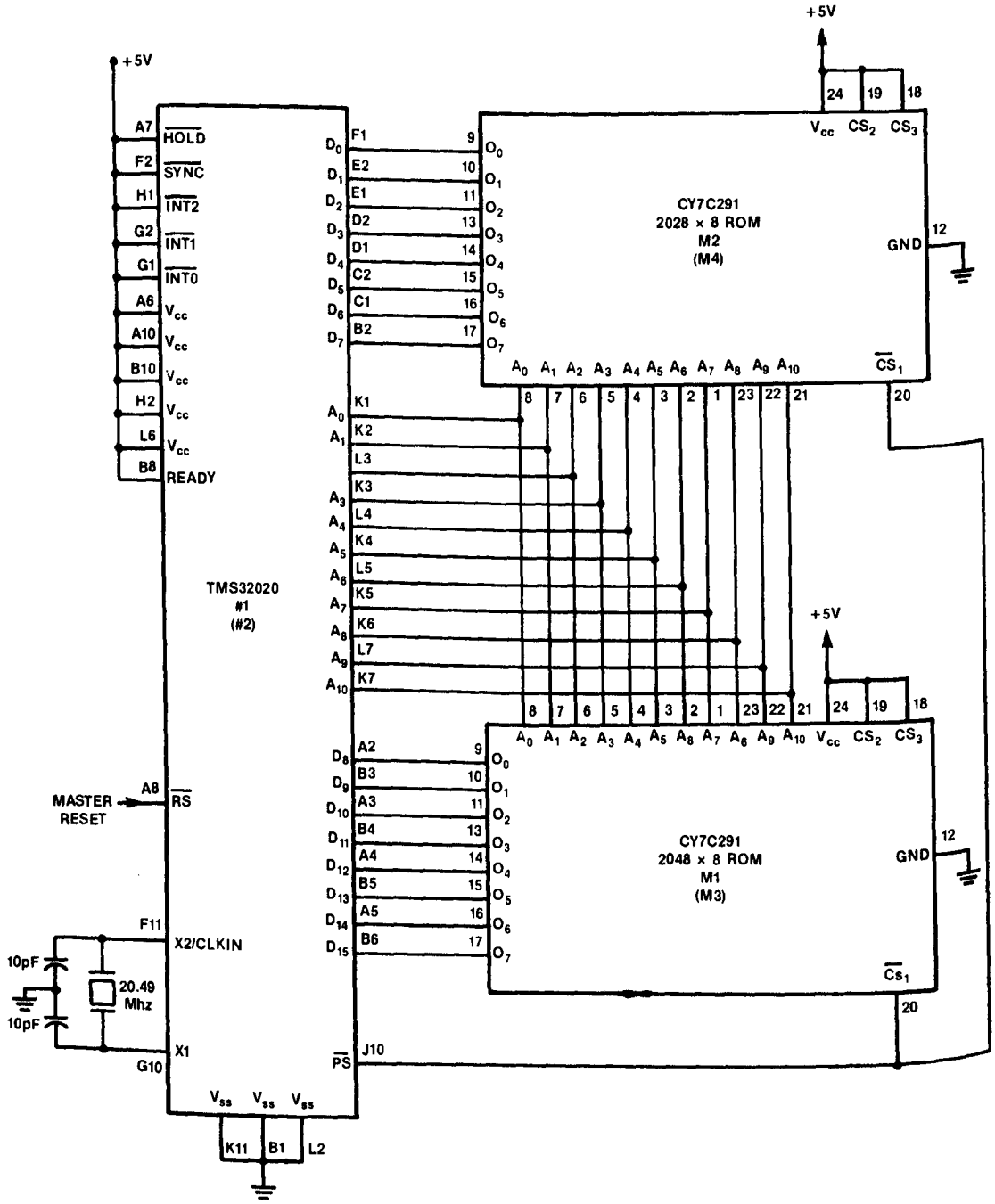



Figure 3.21 : Circuit diagram of program memory for the TMS32020s.

# CHAPTER 4

# EXPERIMENTAL RESULTS

## 4.1 INTRODUCTION

The theory and design of a digital block demodulator has been presented in the previous two chapters. A demodulator has been built based on the TMS32020 microprocessor as described in the chapter 3, with the intention of both demonstrating that the digital approach is a viable alternative to the SAW-based block demodulator and of determining the performance of this approach. In this chapter performance measurements of the digital block demodulator are presented. In most cases the measurements are presented as bit error curves which are plots of the measured probability that a bit re- generated by the demodulator is in error versus the measured ratio of $E_{sr}/N_o$ at the demodulator input. Measurements have been taken with additive Gaus- sian noise at the demodulator input both with and without perfect frequency synchronization between the users and the demodulator. These measurements have been done with a rectangular and a Kaiser-Bessel window to demonstrate the effect of receiver windowing. As well, the effect of an interfering tone has been investigated.

In order to carry out these measurements, suitable FSK signal genera- tors are required. The method of generating the FSK signals is the subject of the next section.

## 4.2 M-ARY FSK SIGNAL GENERATION

The FSK signal generators should be capable of switching between the signaling frequencies in less than the 12.5 $\mu$s allowed for by the demodula- tor, otherwise the characterization of the demodulator performance presented in this chapter would be pessimistic due to the nonideal signals at the input. Two Wavetek model 5155A frequency synthesizers were available for testing the demodulator and they formed the basis for two independent FSK generators. The synthesizers have a quoted switching time of 1 $\mu$s which is more than sufficient for the purpose of testing the demodulator.

Each synthesizer has a parallel interface which allows the output fre- quency to be determined by a binary coded decimal (BCD) input. Figure 4.1 is a block diagram of the circuit used for programming the synthesizers. Binary data is generated by a data error analyser whose output is a pseudo-random bit stream. For $M \neq 2$ this data must be converted to M-ary symbols which consist of $\log_2 M$ bits in parallel. A ROM lookup table is used to take the symbol as an input and generate the BCD code required to program the corresponding symbol frequency at the synthesizer's output. The ROM has several inputs which can be switched between $V_{cc}$ and ground so that the re- quired frequency offsets between the various users for the various values of M can be generated. As well, switches have been included so that there is

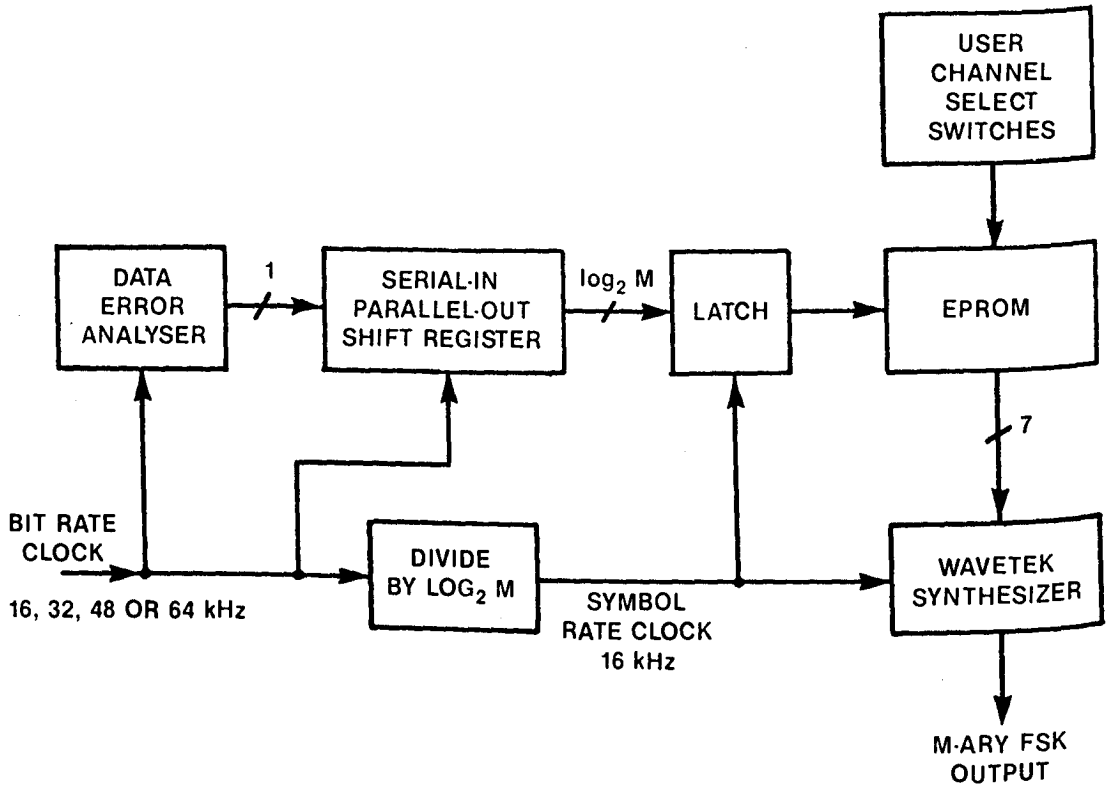



Figure 4.1 : Block diagram of the FSK signal generation circuits.

the option of generating frequency offsets of 4 kHz and 8 kHz corresponding to 10 percent and 20 percent errors in the FSK frequencies, as required for the measurements of section 4.5. Figure 4.2 is a circuit diagram of the electronics for programming the Wavetek synthesizers.

The two FSK signal generators are incorporated as part of the measurement system used to determine the demodulator performance. The overall system is described in the next section.

## 4.3 MEASUREMENT SYSTEM CONFIGURATION

The demodulator performance has been determined with various degradations simulated at the demodulator input. Figure 4.3 is a block diagram of the system used for determining bit error probabilities as a function of the ratio of $E_{sr}/N_o$. The two data generators and synthesizers use the same clock signal so that the symbol switching instants are synchronized between

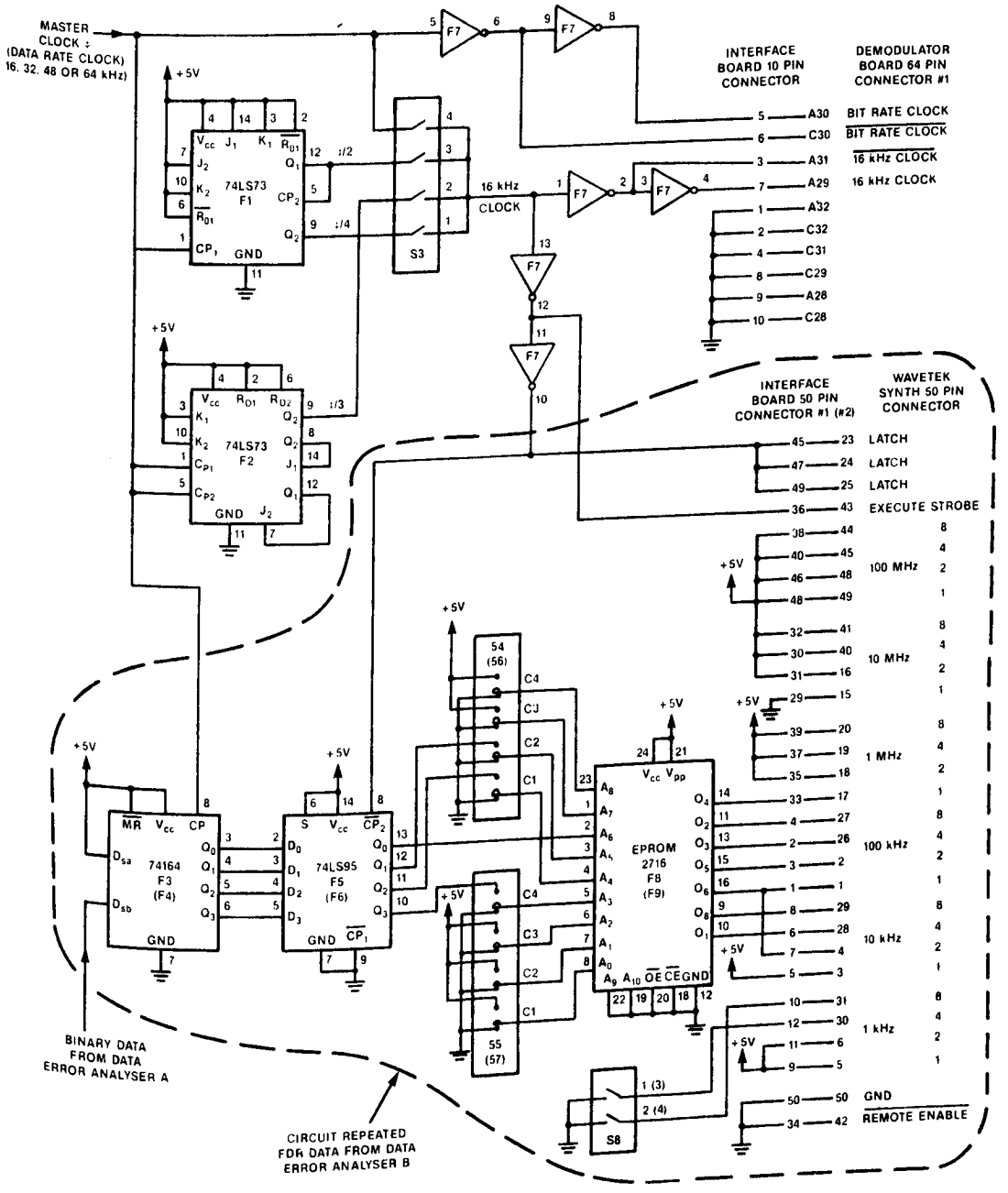



Figure 4.2 : Circuit diagram of the FSK signal generation circuits.

the two resulting FSK signals and the requirement for time synchronization between the users is met. The outputs of the two FSK signal generators designated as user A and user B are added together with system noise to simulate what would be the dehopped IF signal at the receiver. This signal is then operated on by the block demodulator which generates estimates of the transmitted data for each user. The two recovered bit streams corresponding to user A and user B are fed back to the data error analysers which compare the recovered data with the data which was transmitted and generate the bit error rate averaged over a period of time.

For each user the effective received energy per symbol $E_{sr}$ is determined from the measured average power output $C_A$ and $C_B$ of the frequency synthesizers. The energy per symbol is given by

$$E_{sr} = C \cdot T_s \tag{4.3-1}$$

where $T_s$ is the symbol period which is effectively 50 $\mu$s in this case. It was found that the nonzero switching time of the synthesizers has no noticable effect on the measured signal powers.

The system noise is generated with a 50 ohm load connected as the input to several cascaded amplifiers. The output of the final amplifier stage was observed on a spectrum analyser to make sure there were no oscillations and to verify that the noise spectrum is indeed flat over a wide range of frequency. The observed spectrum appeared constant at frequencies up to 20 MHz, beyond which the spectrum dropped off. The noise power spectral density in the region of 10.70 MHz is estimated by passing the noise signal through a bandpass filter with a center frequency of $f_c$ = 10.70 MHz. If the input to the filter with a frequency response H(f) is white Gaussian noise with a power spectral density $N_o/2$ then the average output power is

$$\begin{aligned} N &= N_o \int_0^\infty |H(f)|^2 \, df \\ &= N_o B |H(f_c)|^2 \end{aligned} \tag{4.3-2}$$

where B is the noise equivalent bandwidth of the filter defined as

$$B = \frac{\int_0^\infty |H(f)|^2 \, df}{|H(f_c)|^2} \, . \tag{4.3-3}$$

The noise equivalent bandwidth of the filter has been determined with the use of numerical integration and found to be B = 1.13 MHz, and the midband insertion loss at $f_c$ = 10.70 MHz has been measured as 1.6 dB. Using equations (4.3-1) and (4.3-2) the ratio of $E_{sr}/N_o$ is determined as

$$10 \cdot \log_{10}(E_{sr}/N_o) = C|_{dBm} - N|_{dBm} + 10 \cdot \log_{10}(T_s B) - 20 \cdot \log_{10}|H(f_c)|$$

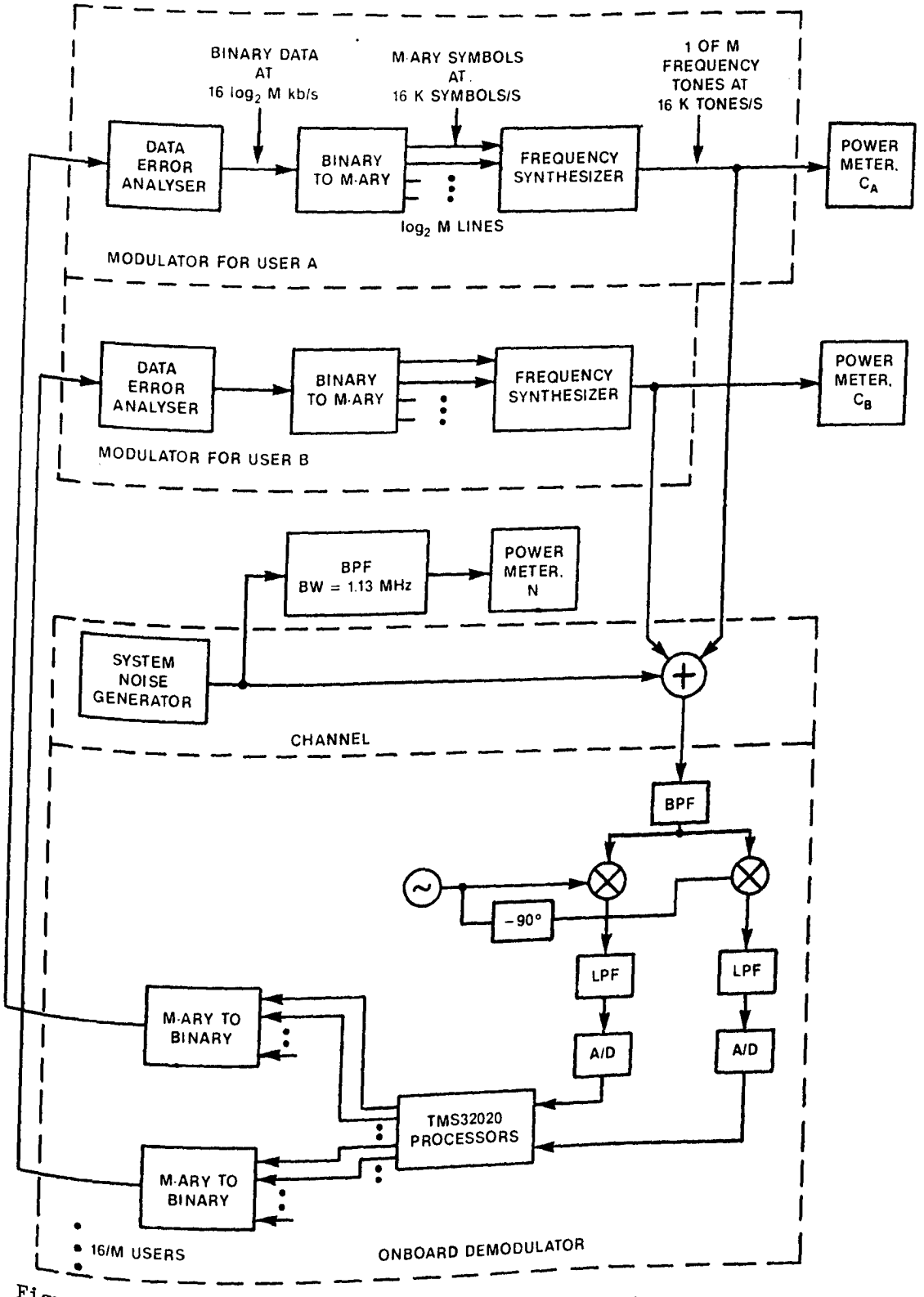



Figure 4.3 : Block diagram of the measurement system.

$$= C|_{dBm} - N|_{dBm} + 15.9 \text{ dB} \qquad (4.3\text{-}4)$$

The bit error probability has been measured with various values of $E_{sr}/N_o$. The results with perfect frequency synchronization are presented in the next section. Before proceeding to the performance characterization, the power consumption of the demodulator is of interest. The demodulator circuit board requires a single 5 volt power supply, and the A/D converters require ±5 volts. The current drawn from each supply has been measured to determine that the demodulator consumes 9.6 watts of power.

## 4.4 PERFORMANCE WITH SYSTEM NOISE

For noncoherent reception of M-ary FSK signals the ideal demodulator performance with system noise and no synchronization errors has been expressed in equation (2.5-39), which is repeated here as equation (4.4-1):

$$P_b = \frac{\exp(-E_{sr}/N_o)}{2(M-1)} \sum_{m=2}^{M} (-1)^m \frac{M!}{m!(M-m)!} \exp(E_{sr}/mN_o) \quad . \qquad (4.4\text{-}1)$$

Figure 4.4 shows the measured bit error probability as a function of the measured ratio of $E_{sr}/N_o$ for 4-ary FSK and rectangular windowing. The theoretical curve given by equation (4.4-1) is included for comparison.

Measurements for each of the four user channel allocations of Figure 3.1 illustrate the effect of the noise aliasing discused in chapter 3. Figure 4.5(a) shows the complex baseband channel allocations for M = 2, 4, or 8. Figure 4.5(b) shows the power spectral density of the analog complex baseband noise process z(t), which is just the square magnitude of the frequency response of the lowpass filters in the inphase and quadrature arms of the demodulator, and Figure 4.5(c) is the spectrum of the sampled version of z(t). From figure 4.5 the noise aliasing should have the most dramatic effect on the channels which include the signaling bins at the edge of the demodulator bandwidth while the noise aliasing should have very little effect on those channels which are made up of only signaling bins close to the center of the demodulator bandwidth. Indeed from Figure 4.4 this is the case as the performance for 4-ary users 1 and 4 which are the channels at either band edge is clearly poorer than that for users 2 and 3 which are the two center channels. The performance for user 4 is the worst of all since the channel for user 4 contains the signaling bin located directly on the sampling frequency, which is the location of the peak in the noise spectrum.

The implementation loss of the demodulator with rectangular windowing is the difference in decibels between the theoretical and measured ratio of $E_{sr}/N_o$ required to maintain a given bit error probability. From Figure 4.4 the implementation loss for 4-ary FSK varies from only 0.2 dB to 1.8 dB, depending on which channel is used. These deviations from the theoretical optimum can be attributed to the quantization and truncation noise associated with the digital processing, the deviation from the ideal 90°

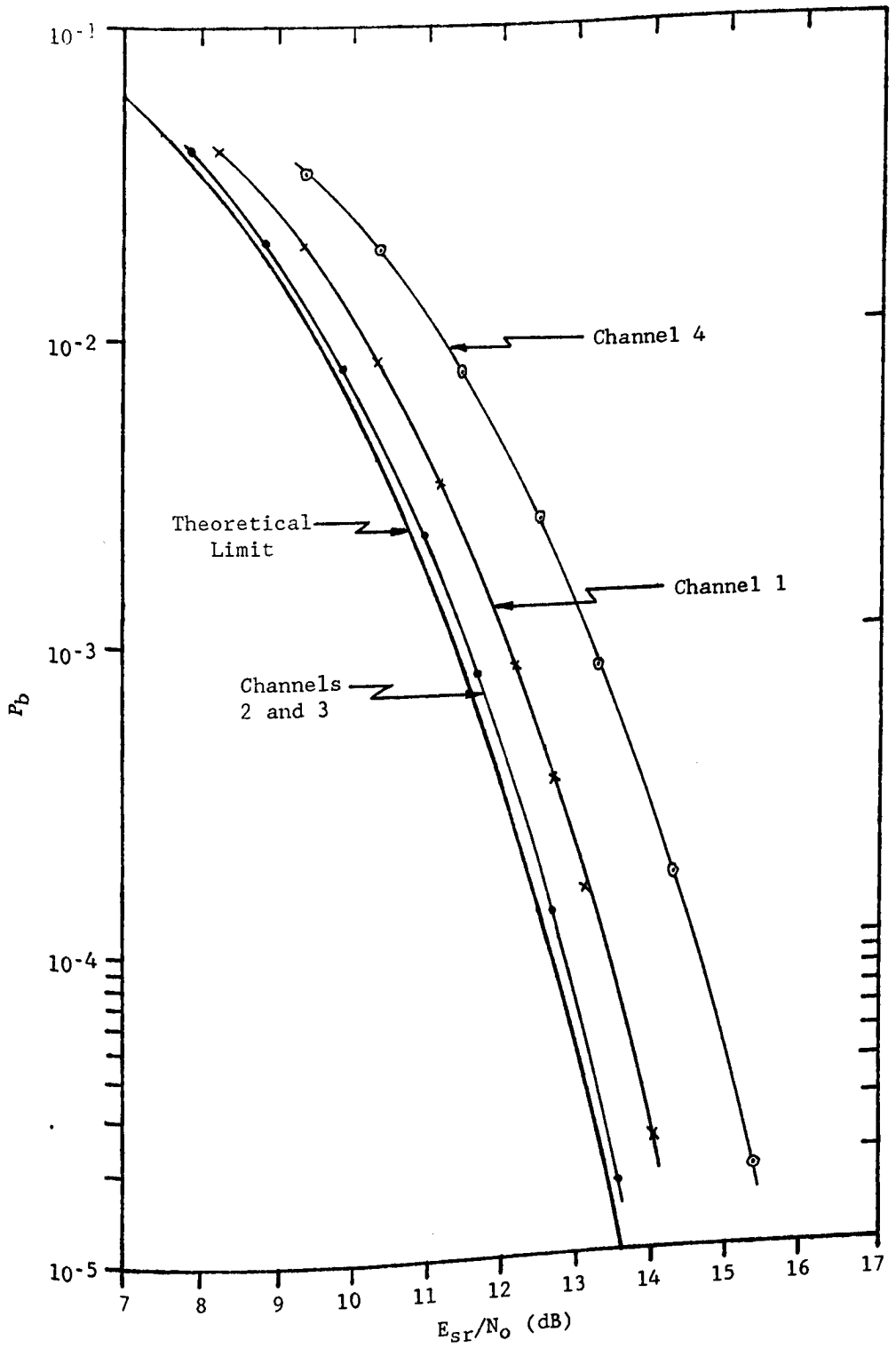



Figure 4.4 : The bit error probability measured for 4-ary FSK for each of the four possible channel locations.

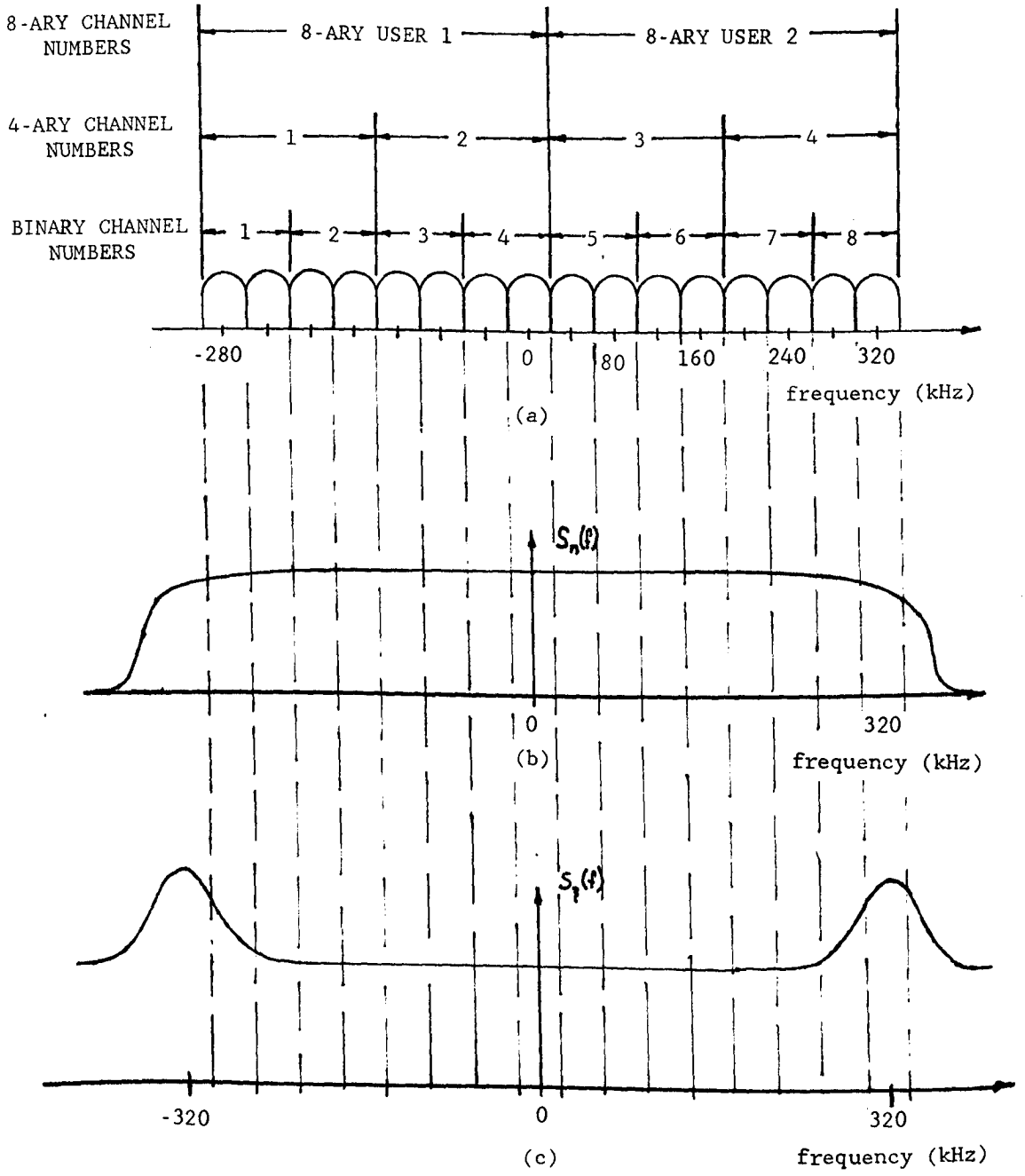



Figure 4.5 : (a) Complex baseband channel locations for M = 2, 4, and 8, (b) complex baseband analog noise spectrum and (c) sampled complex baseband noise spectrum.

phase matching of the analog components of the demodulator's inphase and quadrature arms and any system noise contributions due to the demodulator's electronics. As well, for the channels at the band edges, most of the implementation loss above the 0.2 dB loss associated with the center channels is due to the aliasing of the noise spectrum.

Figure 4.6 shows the bit error curves for binary FSK users with rectangular windowing. Again the implementation loss varies depending on the channel location. The loss for channels 2 to 6 is 0.2 dB which agrees with the measurements for 4-ary FSK. The worst implementation loss of 2.6 dB is for channel 8, which should be the worst loss for any of the possible demodulator configurations. From Figure 4.5(c), both of the signaling bins for binary channel 8 are affected by the aliased noise spectrum, while for 4-ary channel 4, the worst channel for 4-ary users, only two of the four signaling bins are affected by the aliased noise spectrum, and as expected the worst case performance degradation due to the aliasing is worse for binary users than for 4-ary users.

Figures 4.7 and 4.8 show the measured and theoretical bit error curves for 8-ary FSK and 16-ary FSK, both with rectangular windowing. For 8-ary and 16-ary FSK, the measured implementation loss varies with the measured ratio of $E_{sr}/N_o$. Both channels for 8-ary users are affected by the noise aliasing which is apparent from the implementation losses varying from 0.3 dB at low values of $E_{sr}/N_o$ to 0.6 dB at higher values of $E_{sr}/N_o$ for channel 1 and 1.1 dB at low $E_{sr}/N_o$ to 1.6 dB at higher values of $E_{sr}/N_o$ for channel 2. The implementation loss for 16-ary FSK varies from 0.7 to 1.2 dB.

As discussed in chapter 2, due to the possibility of frequency misalignment between the various users and the dehopping synthesizer at the receiver, nonrectangular windowing is of interest. The effect of frequency offsets with both the rectangular window and a Kaiser-Bessel window is investigated in the next section. The Kaiser-Bessel window used in these measurements has the parameter $\alpha = 1.4$. Before investigating the effect of frequency offsets, measurements were made with the Kaiser-Bessel window with no offsets for 4-ary FSK channel 3. The results are presented in Figure 4.9. With nonrectangular receiver windowing the deviation from the theoretical curve is attributed to both the demodulator implementation loss and the processing loss asociated with the window. For 4-ary user 3 the demodulator implementation loss is 0.2 dB. Since from Figure 4.9, the total loss with the Kaiser-Bessel window is 1.4 dB, the measured processing loss for the window is 1.2 dB. This is in agreement with the theoretical processing loss of 1.17 dB listed in Table 2.1 for the Kaiser-Bessel window with $\alpha = 1.4$.

## 4.5 PERFORMANCE WITH FREQUENCY ERRORS

In this section performance measurements are presented with frequency offsets of 4 and 8 kHz corresponding to errors of 10 and 20 percent of the signaling bin spacing. Both the rectangular and Kaiser-Bessel ($\alpha$ =1.4) windows are used. The measurements are all carried out for 4-ary FSK, channel 3.

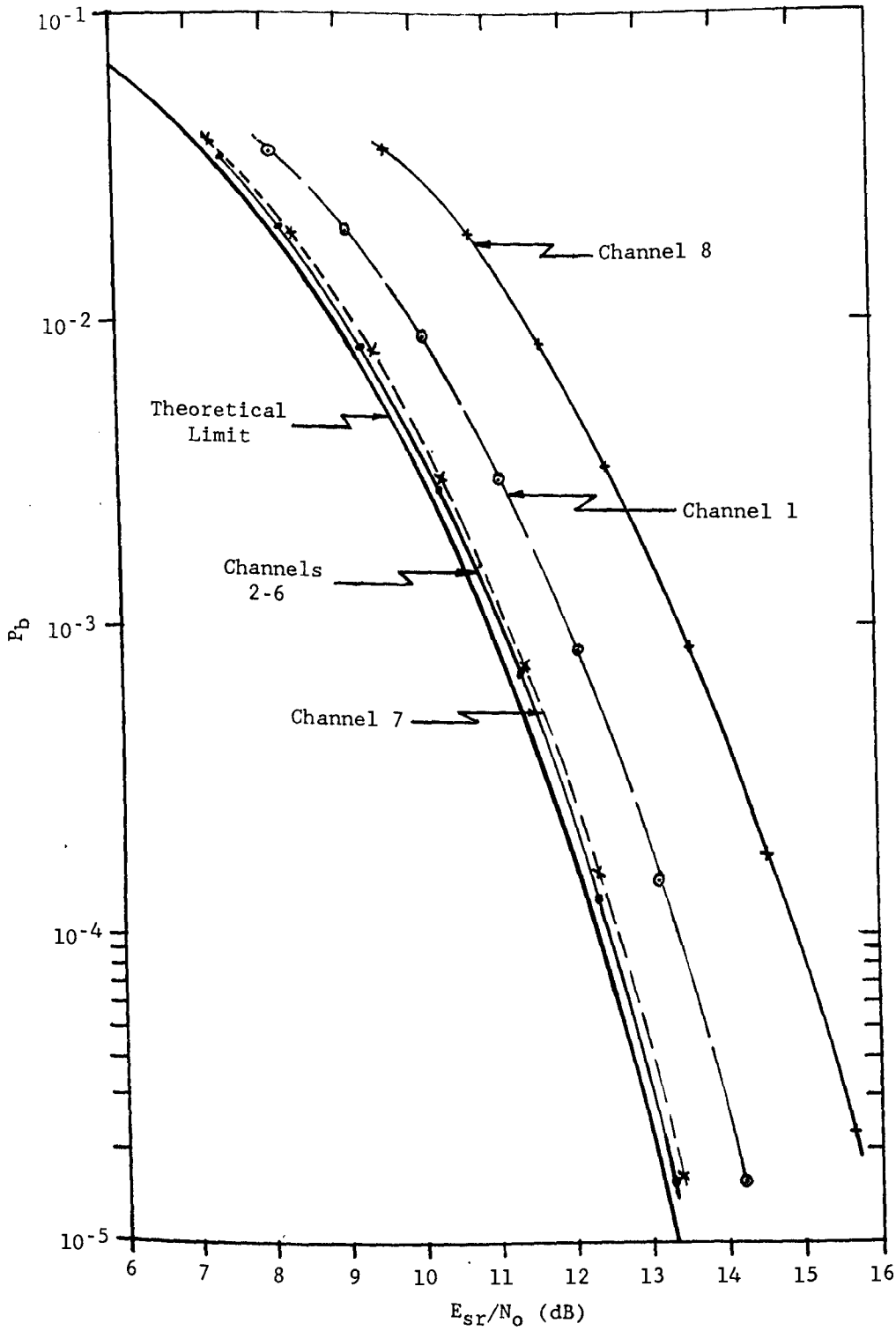



Figure 4.6 : The bit error probability measured for binary FSK for each of the eight possible channel locations.

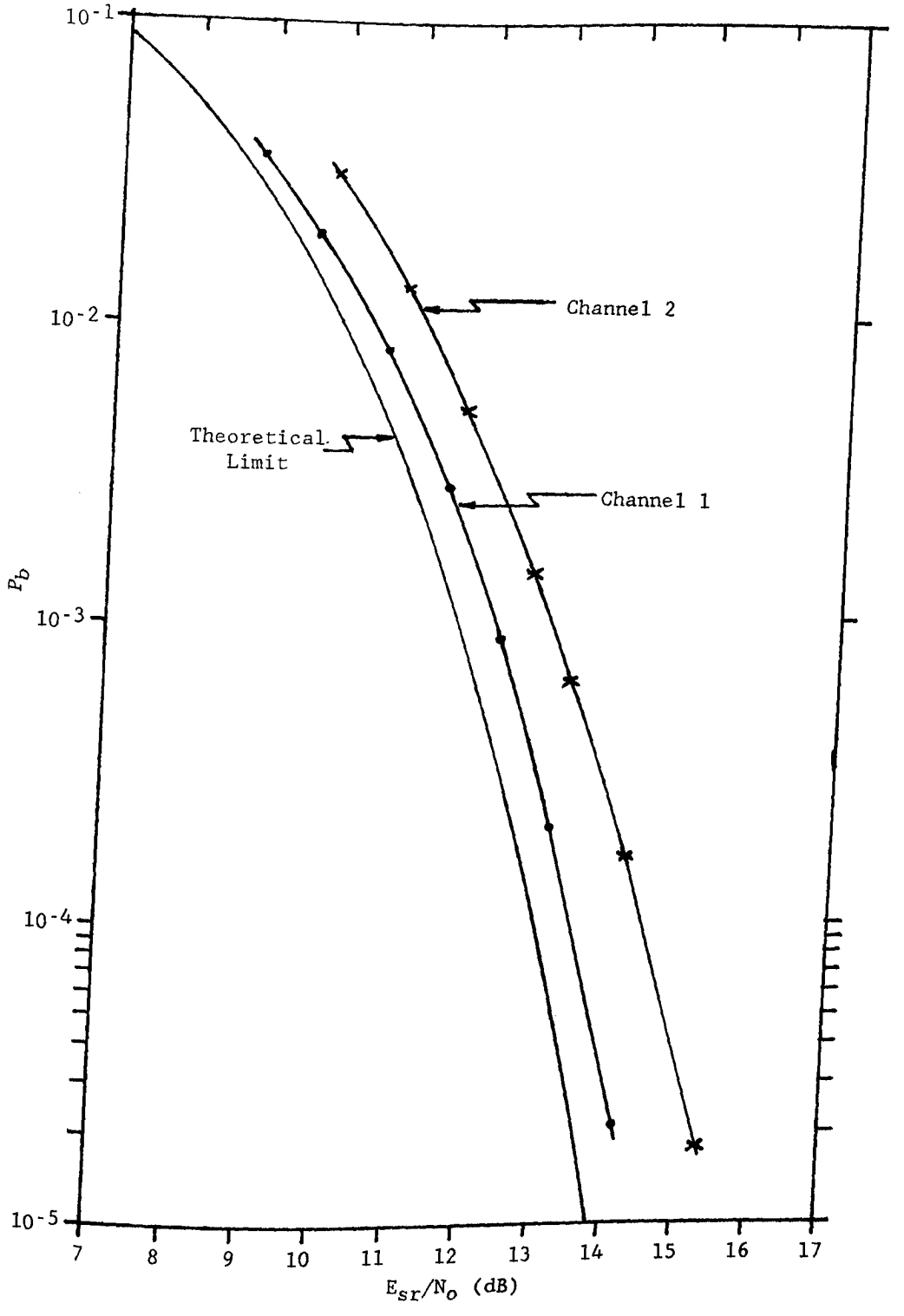



Figure 4.7 : The bit error probability measured for 8-ary FSK for both possible channel locations.

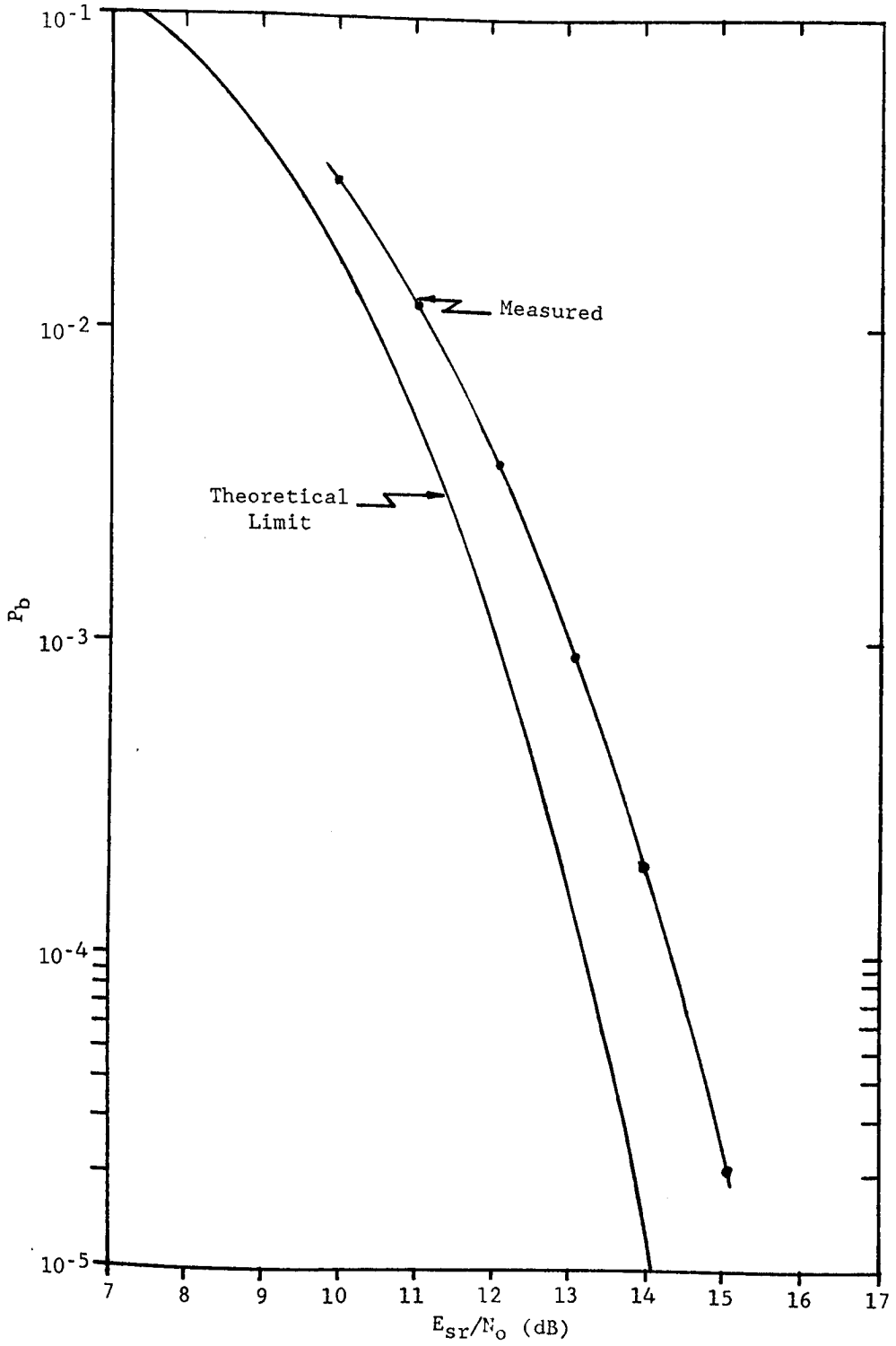



Figure 4.8 : The bit error probability measured for 16-ary FSK.

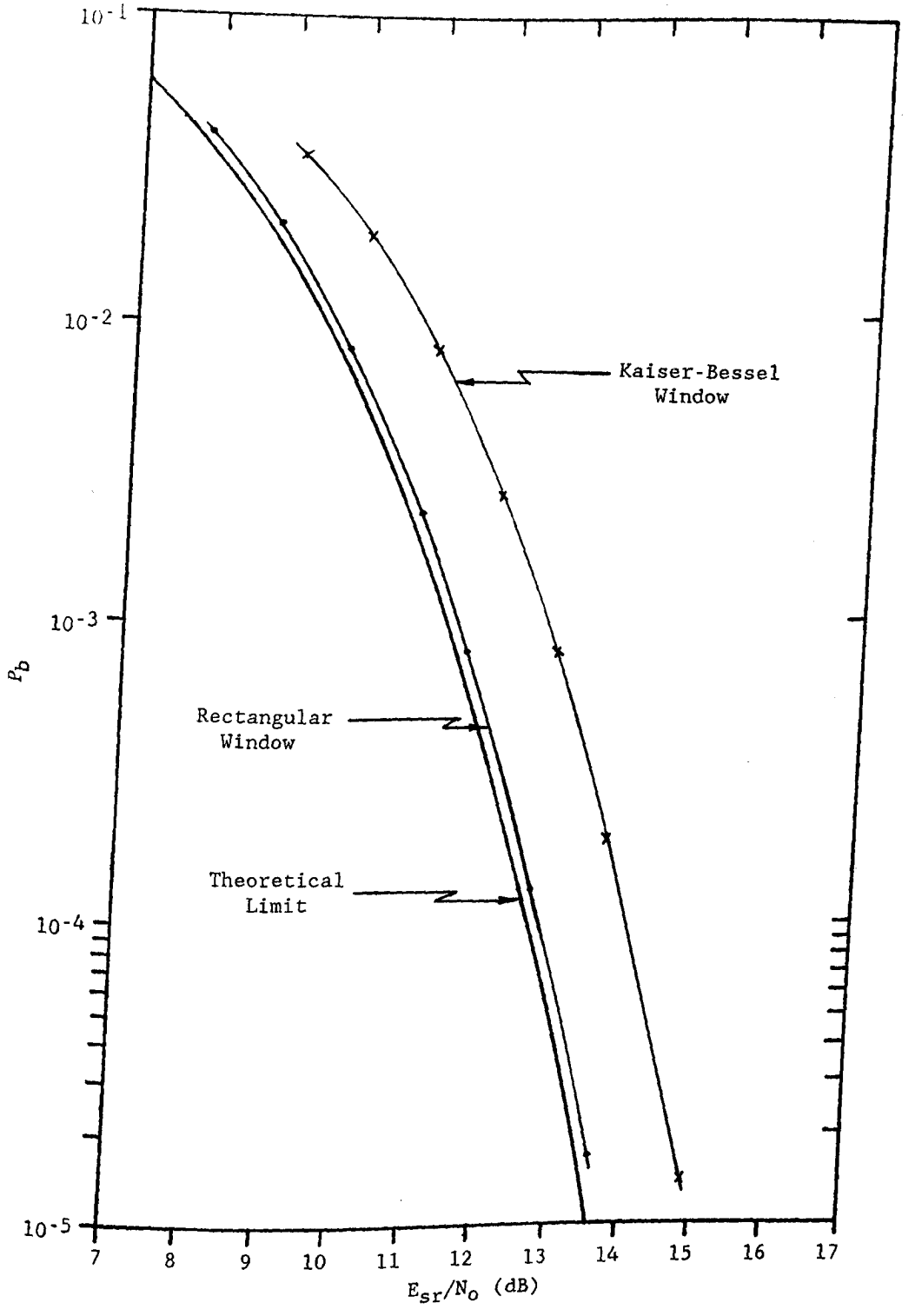



Figure 4.9 : The bit error probability measured for 4-ary channel 3 with rectangular and Kaiser-Bessel ($\alpha$ = 1.4) windows.

The results are shown in Figure 4.10. For rectangular windowing the loss due to the frequency offsets (for a given $P_b$, the difference in dB between the measurements with offsets and those with no offsets) is not constant over the range of measured $E_{sr}/N_o$ but for an offset of 10 percent varies from 0.7 dB at $P_b = 0.04$ to 1.1 dB at $P_b = 10^{-5}$. The loss asociated with the more severe offset of 20 percent varies from 3 dB at $P_b = 0.04$ to 5.3 dB at $P_b = 5 \times 10^{-5}$.

With the Kaiser-Bessel window the loss due to the frequency offsets appears constant for the measured values of $E_{sr}/N_o$, the results being 0.3 dB for offsets of 10 percent and 1.2 dB for offsets of 20 percent. Hence the Kaiser-Bessel window greatly reduces the effects of frequency offsets, but for offsets of 10 percent the additional processing loss asociated with the Kaiser-Bessel window makes the rectangular window still the better of the two. Thus for small frequency errors the rectangular window still performs well, but for larger offsets the Kaiser-Bessel window is better. As well, for small frequency offsets the rectangular window may not be the best choice if the power levels received from the various users can vary greatly due to the high sidelobe levels as discussed in section 2.6. The demodulator may be able to tolerate higher levels of adjacent channel interference with the nonrectangular window. The effect of interference is investigated in the next section.

## 4.6 EFFECT OF CO-CHANNEL AND ADJACENT CHANNEL INTERFERENCE

Since the demodulator is being considered for use in a frequency-hopping system with the possibility of intentional interfering signals present at the receiver input, the effect of interfence is of interest. In this section an interfering tone is placed in or near the channel of interest and the tone amplitude is increased until bit errors are detected. As discussed in the previous section, with frequency offsets, interference from users of adjacent channels may be significant. Measurements are made with two users present, and the difference in power levels between the two users is increased until errors are detected for the lower power user. Again, the measurements in this section are carried out for 4-ary FSK, channel 3. There is no system noise added to the received signal for any of the measurements of this section.

Before proceeding with the measurements with an interfering tone, the difference in average power level has been measured for a single user when the signal drives the A/D converters to the point of saturation, and when the signal is so small that the quantization noise begins to cause errors. This will be an estimate of the dynamic range of the A/Ds. With the rectangular window the measured swing in input power level was 44 dB, and with the Kaiser-Bessel window the measured swing was 42 dB. As expected, the result is slightly smaller for the nonrectangular window due to the window's processing loss.

An interfering tone has been placed exactly on one of the FSK signaling frequencies for user 3. For both window functions, the power of the interfering tone can be raised to 0.4 dB below that of the signal before bit

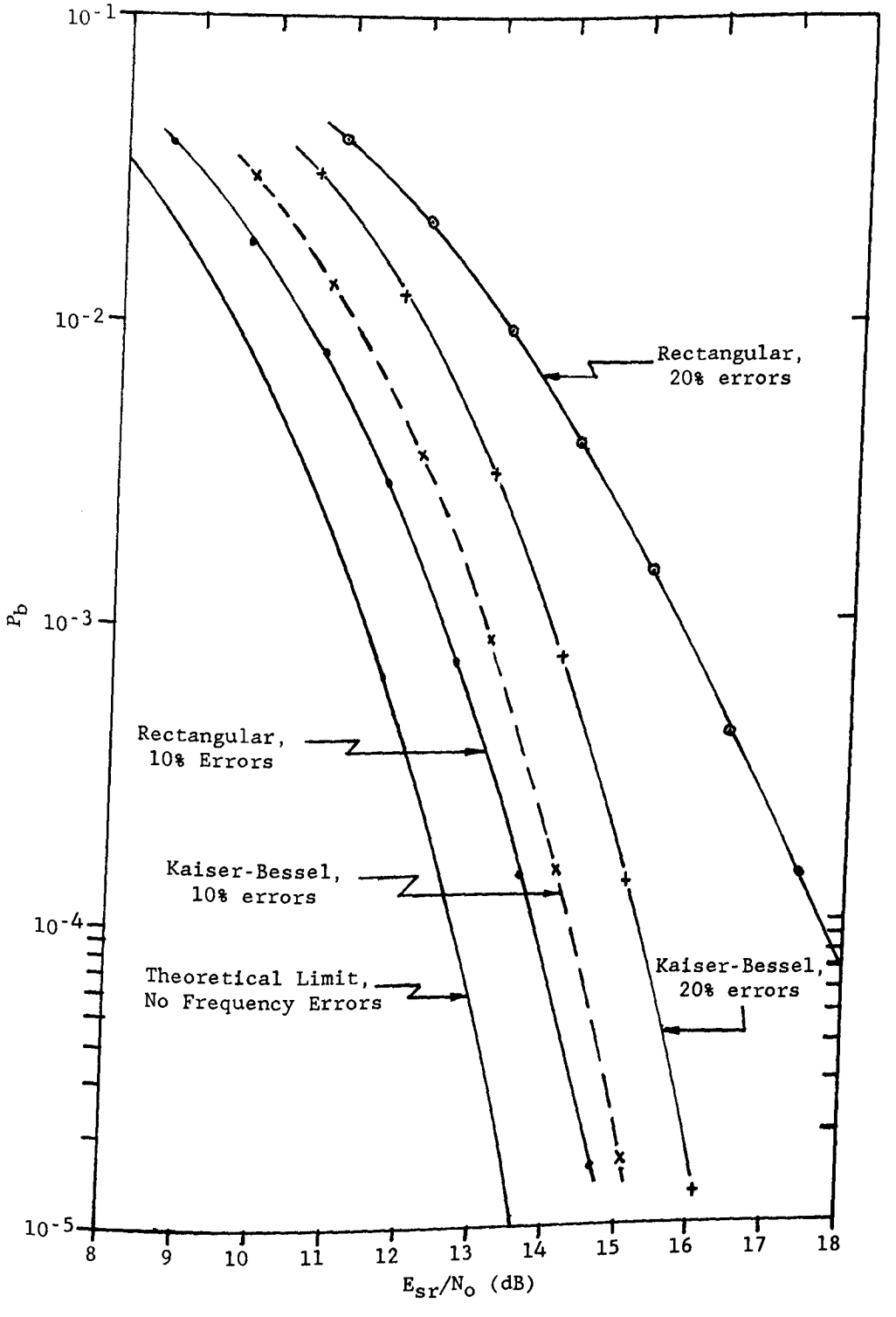



Figure 4.10 : The bit error probability measured for 4-ary FSK with rectangular and Kaiser-Bessel ($\alpha$ = 1.4) windows and frequency offsets of 10 and 20 percent.

| | Interfering tone power to signal power ratio (dB) for tone located: | | |
|---|---|---|---|
| | exactly on a signal frequency | exactly between two signaling frequencies | on nearest adjacent channel signal frequency |
| Rectangular Window | -0.4 dB | 42 dB | 44 dB |
| Kaiser-Bessel Window, $\alpha = 1.4$ | -0.4 dB | 1.9 dB | 27 dB |

Table 4.1 : Interfering tone power required to cause bit errors.

errors begin to occur for the data recovered for user 3. With a perfect system, bit errors would not occur until the power of the interfering tone is equal to the signal power, but because of the demodulator's system and quantization noise, errors occur with an interfering tone power slightly less than the signal power.

The frequency of the interfering tone was then changed so that the tone was exactly half way between two signaling frequencies. With the rectangular window, bit errors begin to occur as the signal power is lowered more than 42 dB below the power of the interfering tone. For the 50 $\mu$s symbol period, the orthogonal tone spacing are multiples of 20 kHz, and since the interfering tone is located 20 kHz from the closest two signaling frequencies and 60 kHz from the other two signaling frequencies, the interfering tone has little effect on the signal demodulation. The signal power must be lowered to almost to the point of dynamic range limit of the A/Ds before errors occur. For the Kaiser-Bessel window, errors begin to occur when the signal power is only 1.9 dB below the power of the interfering tone. This result is surprising since from the simulated DFT frequency response of Figure 3.16 for the Kaiser-Bessel window, the DFT filter response at frequencies of $\pm$20 kHz is 8 dB below the peak of the main lobe. Hence for an interfering tone located half way between two signaling frequencies, the interfering tone should not cause bit errors until its power approaches a level 8 dB above the signal power level.

Finally, the interfering tone was placed exactly on the nearest signaling frequency of the next channel. Again because this corresponds to an orthogonal frequency, the interfering tone has little effect when the rectangular window is used. The signal power could be dropped to 44 dB below the power of the interfering tone, which is the point at which the signal is

no longer within the dynamic range of the A/D converters. With the Kaiser-Bessel window, when the signal power drops more than 27 dB below the interference power, errors begin to occur. The amount of interference that can be tolerated with the Kaiser-Bessel window should be limited by the -31 dB sidelobe levels. The results for the measurements with an interfering tone are summarized in Table 4.1.

From the above measurements, the rectangular window appears to have superior performance compared to the Kaiser-Bessel window. This is due to the fact that for the last two measurements, the interfering tone was placed exactly on an orthogonal tone location for the rectangular window. One the reasons for using a nonrectangular window is to combat the effect of interference when the interfering signal is not orthogonal to to user's signal, as would be the case when there are frequency offsets from the ideal signaling frequencies. To investigate the effect of this sort of adjacent channel interference, two FSK signals were passed to the demodulator input, and the power level of one was lowered until errors began to occur. This was done with various frequency errors for each user. The results for both the rectangular and Kaiser-Bessel windows are summarized in Table 4.2.

| | Adjacent channel power to signal power ratio (dB) for the following frequency offsets: | | | | |
|---|---|---|---|---|---|
| | no offsets | one user offset by 10 % | both users offset by 10 % | one user offset by 20 % | both users offset by 20 % |
| Rectangular Window | 26 dB | 16 dB | 14 dB | 11 dB | 6 dB |
| Kaiser-Bessel Window, $\alpha = 1.4$ | 24 dB | 24 dB | 23 dB | 24 dB | 24 dB |

Table 4.2 : Adjacent channel interfering power required to cause bit errors.

The results clearly show the benefit of the Kaiser-Bessel window when there are frequency errors between the users and the receiver, and the relative power levels of the users differ significantly. For large frequency offsets of 20 percent the users must all have power levels within 6 dB of all the other users if a rectangular window is used, otherwise the low power users will be severely affected by the interference. For the same situation

with a Kaiser-Bessel window with $\alpha = 1.4$, the users power levels can deviate by 24 dB. Interestingly, the adjacent channel power levels which can be tolerated with the Kaiser-Bessel window is almost constant at 24 dB for frequency errors from 0 to 20 percent. With no frequency offsets, the rectangular window allows a 26 dB difference in power levels before errors occur for the low power user. Ideally this difference should approach the 44 dB limit of the dynamic range of the A/D converters since the two users signals are supposed to be orthogonal. The large difference between the measured value and the theoretical limit of 44 dB can be attributed to the combination of small phase and frequency transients resulting from the imperfect FSK generators, and the demodulator quantization. These results combined with those of section 4.5 show that the demodulation with rectangular windowing is much more sensitive to frequency misalignments.

This completes the description of the performance measurements which have been carried out with the demodulator. The results are further evaluated in the conclusions of the next chapter.

# CHAPTER 5

# CONCLUSIONS

## 5.1 PERFORMANCE SUMMARY AND EVALUATION

A digital block demodulator for M-ary NCFSK signals multiplexed in frequency has been built and tested. The measured bit error performance depends on the channel allocation within the demodulator bandwidth, due to aliased noise power resulting from the sampling process. The implementation loss thus varies from 0.2 dB for the best channels to 2.6 dB for the worst channel allocation. In the demodulator design the degradation due to aliasing was predicted and accepted in order to keep the demodulator implementation fairly simple while processing a reasonable bandwidth. With more hardware the noise aliasing could be avoided and the implementation loss could be held to about 0.2 dB for all of the channel allocations, which is a very good result.

The dynamic range of the demodulator is limited by the number of bits used in quantization by the A/D converters. With only a single user and no noise present at the input, the dynamic range was measured as 44 dB with rectangular windowing, and 42 dB with the Kaiser Bessel window ($\alpha = 1.4$). If a larger dynamic range is desired, the number of quantization bits could be increased to 10 with very little changes required in the hardware, and no changes required in the demodulator software. Increasing the number of quantization bits beyond 10 would require changes to the demodulator software to allow for software scaling to keep the magnitude of the intermediate and final DFT results below one. The increase in quantization bits would then be limited by the increase in instructions required for software scaling. As well, the increase would be limited by the 16-bit register length of the TMS32020.

With a co-channel interfering tone the demodulator performs best with the rectangular window. This is due to the fact that the rectangular window has the narrowest possible mainlobe bandwidth and there is less chance that the interfering tone will be located at a frequency with low attenuation in the DFT filter responses.

Measurements with frequency errors showed that with equal power users and moderate errors of 10 percent, the rectangular window still gives better performance than the Kaiser-Bessel window due to the inherent processing loss of the nonrectangular window. For large errors of 20 percent, the Kaiser-Bessel window is superior. With the possibility of large differences in the individual users' power levels, with frequency errors of 10 percent the Kaiser-Bessel window gives about 9 dB more protection against adjacent channel interference compared to the rectangular window. Thus for expected frequency offsets of $\geq$10% and large differences in power levels, the nonrectangular window is the better choice.

Similar measurements have been carried out, described in [4], with a SAW-based demodulator. A comparison of the two technologies is presented next.

## 5.2 COMPARISON WITH A SAW-BASED DEMODULATOR

There are a few differences in the system parameters for the SAW-based demodulator presented in [4] compared to the those used in this thesis. Most notably the SAW-based demodulator processes a greater number of signaling bins (40) and therefore can handle more users. The symbol rate is faster at 20 k symbols/s, and the effective symbol period operated on by the demodulator is shorter at 25 $\mu$s. None of these differences affect the comparison of bit error rates as a function of $E_{sr}/N_o$, and hence the measurements with system noise can be compared directly. For the SAW demodulator the FSK tone spacing is 100 kHz or 2.5 times the minimum orthogonal tone spacing of $(25\ \mu s)^{-1} = 40$ kHz. Because of this difference compared to the spacing used in this thesis, the measurements with frequency errors and interfering tones cannot be directly compared.

The measured implementation loss for the SAW-based demodulator was 0.5 dB. Since the demodulator uses analog processing, there is no problem of noise aliasing and the loss does not depend on the channel allocation. For some channel allocations, the digital demodulator performs slightly better with an implementation loss of 0.2 dB, while for the worst channel the digital implementation performs considerably worse with a loss of 2.6 dB. If the noise aliasing in the digital demodulator can be avoided, its performance would then be comparable to or in fact slightly better than the SAW-based implementation.

The dynamic range of the SAW demodulator was found to be 50 dB, which is better than the maximum 44 dB dynamic range found for the digital approach used in this thesis. As mentioned the dynamic range of the digital demodulator was sufficient for the purpose of this study, but can be improved with longer quantization register lengths. An improved SAW-based demodulator mentioned in [4] is currently being developed which should have a dynamic range greater than 50 dB.

The power requirements for the SAW-based demodulator are not discussed in [4], but the manufacturer's specifications for the particular SAW device used state a power consumption of 14.75 watts. In addition, the demodulator has some digital electronics associated with it which increases the power consumption beyond the 14.75 watts of the SAW device. Thus the SAW-based demodulator requires more power than the 9.6 watts consumed by the digital implementation presented here. In neither case, though, was the design and implementation done with the intention of minimizing power consumption.

For applications for which the demodulators would be employed in space such as onboard processing for frequency-hopping satellite communications, the effect of radiation should be minimal. Apparently surface acoustic wave devices are not affected by radiation whereas digital electronics constructed with the NMOS process used for the TMS32020 are adversely

affected by radiation and are not considered good for space applications. Texas Instruments is planning the production of a CMOS version of the TMS32020. Apparently CMOS devices have better radiation hardness than NMOS devices, but they still may not meet the radiation resistant requirements for space-borne applications, unless the CMOS process used is specifically designed to be "radiation hard".

It appears then that the SAW-based demodulator is better suited for space applications unless a suitable digital signal processor is developed which is radiation hard. Also the SAW-based demodulator is capable of processing more users. From a performance standpoint though, the digital implementation could be slightly better.

## 5.3 FUTURE STUDIES

As mentioned above, Texas Instruments has announced the development of the TMS320C25 microprocessor, an enhanced CMOS version of the TMS32020 which has an instruction time of 100 ns, and would allow TMS32020 software to run at almost twice the speed, opening up many possibilities for further study. With the configuration of two microprocessors used in this project, replacing the '32020s with '320C25s would give the designer 610 more instruction cycles for the demodulation process. This would allow the noise aliasing problem to be eliminated, and as well the bandwidth and number of users could be expanded. Alternatively the symbol period could be shortened for a faster data rate.

Several interesting experiments could be performed with the demodulator which has been built. Measurements with the simulation of the jamming signals found in frequency-hopping systems could be carried out as described in [4]. Because of the ease in changing window functions, the effect of the window with jamming signals present could be investigated. Error correction encoding and diversity combining could be implemented to experimentally study the improvements they should bring about with a jamming environment. As well, the performance measurements found in this thesis could be repeated for several different windows to aid the final choice of a window.

# REFERENCES


1. Torrieri, D.J., PRINCIPLES OF MILITARY COMMUNICATION SYSTEMS, Artech House, Dedham, Ma., 1983.

2. Simon, M.K., Omura, J.K., Scholtz, R.A., and Levitt, B.K., SPREAD SPECTRUM COMMUNICATIONS, Volume 1, Computer Science Press, Rockville, Md., 1985.

3. Dixon, R.C., SPREAD SPECTRUM SYSTEMS, Wiley-Interscience, New York, 1976.

4. Felstead, E.B., Pearce, J.L., and Selin, D.L., "A SAW-Based Demodulator For Multiple-User Dehopped M-Ary FSK Signals - Implementation And Measurements," CRC Report No. 1392, Communications Research Centre, Ottawa, Ontario, June 1985.

5. Wittke, P.H., McLane, P.J., Ma, P., "Study Of The Reception Of Frequency Dehopped M-Ary FSK," Research Report No. 83-1, Dept. of Elec. Eng., Queen's Univ., Kingston, Ontario, March 1983.

6. Proakis, J.G., DIGITAL COMMUNICATIONS, McGraw Hill, New York, 1983.

7. Urkowitz, H., SIGNAL THEORY AND RANDOM PROCESSES, Artech House, Dedham, Ma., 1983.

8. Oppenheim, A.V., and Schafer, R.W., DIGITAL SIGNAL PROCESSING, Prentice-Hall, Englewood Cliffs, N.J., 1975.

9. Metzger, L.S., Boroson, D.M., Uhran, J.J., and Kalet, I., "Receiver Windowing For FDM MFSK Signals," IEEE Trans. Comm., vol. COM-27, pp. 1519-1526, October 1979.

10. Harris, F.J., "On The Use Of Windows For Harmonic Analysis With The Discrete Fourier Transform," Proc. IEEE, vol. 66, pp. 51-83, January 1978.

11. Rabiner, L.R., and Gold, B., THEORY AND APPLICATION OF DIGITAL SIGNAL PROCESSING, Prentice-Hall, Englewood Cliffs, N.J., 1975.

12. Elliot, D.F., and Rao, K.R., FAST TRANSFORMS: ALGORITHMS, ANALYSES, APPLICATIONS, Academic Press, New York, 1982.

13. McClellan, J.H., and Rader, C.M., NUMBER THEORY IN DIGITAL SIGNAL PROCESSING, Prentice-Hall, Englewood Cliffs, N.J., 1979.

14. Winograd, S., "On Computing The Discrete Fourier Transform," Mathematics of Computation, vol. 32, number 141, pp. 175-199, January 1978.

# APPENDIX:

## TMS32020 ASSEMBLER LANGUAGE PROGRAM

## FOR THE 4-ARY NCFSK DEMODULATOR

In this appendix, the demodulator software is presented for the case M = 4. Only the decision section of code is different for the other cases of M. The first section simply assigns addresses to the various windowing constants, DFT constants, and intermediate and final DFT variables. The next section, labeled the initialization section, is where the microprocessor starts from on power-up or after the reset has been activated. In this section the DFT and window coefficients are defined and stored in the on-chip RAM, the I/O section of code is moved to on-chip RAM, and the TMS32020 is configured as required for the demodulator. The I/O section follows in which the decision results from the previous symbol are sent as output to offchip shift registers for parallel-to-serial conversion, and the complex input samples for the next symbol period are obtained as input from the FIFOs. The section that follows is the windowing and DFT section, which is an implementation of the algorithm of Figure 3.12. The window used in this example is the Kaiser-Bessel window ($\alpha$ = 1.4). The final section of code is for 4-ary decisions. Decision sections for M = 2, 8, and 16 are similar in form. The last few lines of code for the decision section are actually contained at the beginning of the I/O section.

```
*****************************************************************
*
*      TMS32020 Assembler Language Program For 4-Ary NCFSK:
*
      IDT     '4KB1'
*
*
*     This version of the program is to be run on TMS32020
* processor  1.
*
*     The following definitions assign page addresses to
* the symbols used for direct addressing throughout the
* program.
*
* The input data sequence {r(n)} is stored in memory
* locations 0 to 63:
*
RER0  EQU     0
IMR0  EQU     1
RER1  EQU     2
IMR1  EQU     3
RER2  EQU     4
IMR2  EQU     5
RER3  EQU     6
IMR3  EQU     7
RER4  EQU     8
IMR4  EQU     9
RER5  EQU     10
IMR5  EQU     11
RER6  EQU     12
IMR6  EQU     13
RER7  EQU     14
IMR7  EQU     15
RER8  EQU     16
IMR8  EQU     17
RER9  EQU     18
IMR9  EQU     19
RER10 EQU     20
IMR10 EQU     21
RER11 EQU     22
IMR11 EQU     23
RER12 EQU     24
IMR12 EQU     25
RER13 EQU     26
IMR13 EQU     27
RER14 EQU     28
IMR14 EQU     29
RER15 EQU     30
IMR15 EQU     31
RER16 EQU     32
IMR16 EQU     33
RER17 EQU     34
```

```
IMR17 EQU     35
RER18 EQU     36
IMR18 EQU     37
RER19 EQU     38
IMR19 EQU     39
RER20 EQU     40
IMR20 EQU     41
RER21 EQU     42
IMR21 EQU     43
RER22 EQU     44
IMR22 EQU     45
RER23 EQU     46
IMR23 EQU     47
RER24 EQU     48
IMR24 EQU     49
RER25 EQU     50
IMR25 EQU     51
RER26 EQU     52
IMR26 EQU     53
RER27 EQU     54
IMR27 EQU     55
RER28 EQU     56
IMR28 EQU     57
RER29 EQU     58
IMR29 EQU     59
RER30 EQU     60
IMR30 EQU     61
RER31 EQU     62
IMR31 EQU     63
*
* The window data sequence {w(n)} is stored in memory
* locations 111 to 127:
*
W0    EQU     111     The values of w(n) for n > 16
W1    EQU     112     are not stored since the
W2    EQU     113     sequence is symmetric about
W3    EQU     114     n = 16.
W4    EQU     115
W5    EQU     116
W6    EQU     117
W7    EQU     118
W8    EQU     119
W9    EQU     120
W10   EQU     121
W11   EQU     122
W12   EQU     123
W13   EQU     124
W14   EQU     125
W15   EQU     126
W16   EQU     127
*
* Constants used in calculating the DFT are stored in
```

```
* locations 108 to 110:
*
SIN2U EQU     108
SIN3U EQU     109
COS3U EQU     110
COSU  EQU     SIN3U
SINU  EQU     COS3U
*
* Memory location 107 is used as a temporary storage
* register:
*
TEMP  EQU     107
*
* Intermediate and final DFT results are stored in various
* memory locations from 0 to 37:
*
REX1  EQU     2       x(n) = r(n) + r(n+16),
IMX1  EQU     3       n = 0, 1, •••, 15
REX2  EQU     4
IMX2  EQU     5
REX4  EQU     8
IMX4  EQU     9
REX7  EQU     14
IMX7  EQU     15
REX8  EQU     16
IMX8  EQU     17
REX11 EQU     22
IMX11 EQU     23
REX13 EQU     26
IMX13 EQU     27
REX14 EQU     28
IMX14 EQU     29
*
RET1  EQU     32
IMT1  EQU     33
RET4  EQU     20
IMT4  EQU     21
RET5  EQU     12
IMT5  EQU     13
RET6  EQU     28
IMT6  EQU     29
RET8  EQU     18
IMT8  EQU     19
RET9  EQU     6
IMT9  EQU     7
RET11 EQU     10
IMT11 EQU     11
RET12 EQU     26
IMT12 EQU     27
RET14 EQU     30
IMT14 EQU     31
RET15 EQU     32
```

```
IMT15 EQU     33
RET17 EQU     32
IMT17 EQU     33
RET19 EQU     2
IMT19 EQU     3
RET20 EQU     6
IMT20 EQU     7
RET21 EQU     14
IMT21 EQU     15
RET23 EQU     30
IMT23 EQU     31
RET24 EQU     18
IMT24 EQU     19
RET25 EQU     22
IMT25 EQU     23
RET26 EQU     26
IMT26 EQU     27
*
REM2  EQU     12
IMM2  EQU     13
REM3  EQU     8
IMM3  EQU     9
REM4  EQU     16
IMM4  EQU     17
IMM5  EQU     3
REM6  EQU     22
IMM6  EQU     23
REM10 EQU     11
IMM10 EQU     10
REM11 EQU     5
IMM11 EQU     4
REM12 EQU     25
IMM12 EQU     24
REM13 EQU     15
IMM13 EQU     14
*
RES5  EQU     16
IMS5  EQU     17
RES8  EQU     14
IMS8  EQU     15
RES13 EQU     21
IMS13 EQU     20
RES15 EQU     34
IMS15 EQU     35
RES16 EQU     31
IMS16 EQU     30
RES19 EQU     25
IMS19 EQU     24
RES20 EQU     29
IMS20 EQU     28
*
REF0  EQU     32      The DFT outputs are labeled F(p)
```

```
IMF0   EQU     33      rather than X(p) to avoid
REF1   EQU     18      confusion with the sequence x(n).
IMF1   EQU     19
REF2   EQU     2
IMF2   EQU     3
REF3   EQU     29
IMF3   EQU     28
REF4   EQU     12
IMF4   EQU     13
REF5   EQU     25
IMF5   EQU     24
REF6   EQU     8
IMF6   EQU     9
REF7   EQU     21
IMF7   EQU     20
REF8   EQU     6
IMF8   EQU     7
REF9   EQU     36
IMF9   EQU     37
REF10  EQU     5
IMF10  EQU     4
REF11  EQU     31
IMF11  EQU     30
REF12  EQU     11
IMF12  EQU     10
REF13  EQU     16
IMF13  EQU     17
REF14  EQU     0
IMF14  EQU     1
REF15  EQU     26
IMF15  EQU     27
*
*****************************************************************
*
*     Initialization: On powerup or when reset is
* activated program execution starts from memory location 0.
*
       AORG    0
       B       INIT        Go to initialization routine.
*
       AORG    >0020
*
*     The following table contains the constant values
* used in windowing and computing the DFT:
*
COEF   DATA    >5A82        Sin(2U)

       DATA    >7642        Sin(3U)
       DATA    >CF04       -Cos(3U)
*
*
WINDOW         DATA    >0000       W(0)     These coefficients are
```

```
          DATA    >0D6E       W(1)     for a KAISER-BESSEL
          DATA    >1401       W(2)     window with a = 1.4.
          DATA    >1BAC       W(3)
          DATA    >2452       W(4)
          DATA    >2DCA       W(5)
          DATA    >37DE       W(6)
          DATA    >424B       W(7)
          DATA    >4CCB       W(8)
          DATA    >570D       W(9)
          DATA    >60C1       W(10)
          DATA    >6997       W(11)
          DATA    >714C       W(12)
          DATA    >778C       W(13)
          DATA    >7C30       W(14)
          DATA    >7F0A       W(15)
          DATA    >7FFF       W(16)
*
*
INIT  RPTK    9             A ten cycle delay is included
      NOP                   here in the initialization for
*                           processor  1 to ensure that
* processor  2 reaches the point of polling the BIO pin
* before processor  1.
*
*
      SOVM                  Set overflow mode.
      SPM     0             No shift on output from P reg.
      SSXM                  Set sign extension.
*
      LARP    AR1           Move DFT coefficients and window
      LRLK    AR1,876       data to on-chip memory
      RPTK    19            locations.
      BLKP    COEF,*+
*
      LDPK    6             Load data page pointer.
*
      LRLK    AR1,512       Move I/O program section to on-
      RPTK    IOL-1         chip RAM memory.
      BLKP    LOC5,*+
*
      CNFP                  Configure RAM block as program
*                           memory.
*
      RXF                   Reset the external flag to
*                           indicate that initialization
*                           is complete.
*
      B       POLL+OFFSET Proceed to I/O section.
*
*************************************************************
*
*     I/O Section: This program section is used for input
```

```
* and output. In the initialization section it is
* transfered to on-chip RAM program memory because the IN
* and OUT instructions execute in fewer machine cycles when
* executed from on-chip memory. A code representing the
* decisions as calculated in the decision section is sent
* through output port  0 to a decoding circuits. The line
* labeled LOC6 is really the last line of the decision
* section code. Then samples of the inphase and quadrature
* components of the input waveform are obtained through
* input ports  0 and  1 from the FIFOs.
*
*
IO    EQU     $           The label "IO" is assigned to
*                         the current program memory
*                         location, representing the
*                         beginning of the IO section.
*
*      The next four lines of code are the last part of
* the decision section of code. They are included here
* because this section of code will run from on-chip RAM
* memory, and the OUT instruction will then take only one
* instruction cycle to execute, whereas if these four lines
* were kept at the end of the decision section code which
* runs from external ROM, the OUT instruction would require
* two instruction cycles to execute.
*
LOC5  LARK    AR0,128
LOC6  MAR     *0+
      SAR     AR1,TEMP
      OUT     TEMP,PA0    Send the decision code to the
*                         decoding circuits.
*
*     The input section begins here.
*
POLL  BIOZ    POLL+OFFSET Test the BIO pin to see if
*                         sampled data is ready in the
*                         FIFOs. Repeat test until data
*                         is ready.
*
      IN      RER0,PA0    Get data samples from the FIFOs.
      IN      IMR0,PA1    PA0 is the port address of the
      IN      RER1,PA0    FIFO for the inphase arm of the
      IN      IMR1,PA1    demodulator, and PA1 is the
      IN      RER2,PA0    address of the FIFO for the
      IN      IMR2,PA1    quadrature arm.
      IN      RER3,PA0
      IN      IMR3,PA1
      IN      RER4,PA0
      IN      IMR4,PA1
      IN      RER5,PA0
      IN      IMR5,PA1
      IN      RER6,PA0
```

```
        IN      IMR6,PA1
        IN      RER7,PA0
        IN      IMR7,PA1
        IN      RER8,PA0
        IN      IMR8,PA1
        IN      RER9,PA0
        IN      IMR9,PA1
        IN      RER10,PA0
        IN      IMR10,PA1
        IN      RER11,PA0
        IN      IMR11,PA1
        IN      RER12,PA0
        IN      IMR12,PA1
        IN      RER13,PA0
        IN      IMR13,PA1
        IN      RER14,PA0
        IN      IMR14,PA1
        IN      RER15,PA0
        IN      IMR15,PA1
        IN      RER16,PA0
        IN      IMR16,PA1
        IN      RER17,PA0
        IN      IMR17,PA1
        IN      RER18,PA0
        IN      IMR18,PA1
        IN      RER19,PA0
        IN      IMR19,PA1
        IN      RER20,PA0
        IN      IMR20,PA1
        IN      RER21,PA0
        IN      IMR21,PA1
        IN      RER22,PA0
        IN      IMR22,PA1
        IN      RER23,PA0
        IN      IMR23,PA1
        IN      RER24,PA0
        IN      IMR24,PA1
        IN      RER25,PA0
        IN      IMR25,PA1
        IN      RER26,PA0
        IN      IMR26,PA1
        IN      RER27,PA0
        IN      IMR27,PA1
        IN      RER28,PA0
        IN      IMR28,PA1
        IN      RER29,PA0
        IN      IMR29,PA1
        IN      RER30,PA0
        IN      IMR30,PA1
        IN      RER31,PA0
        IN      IMR31,PA1
*
```

```
* and output. In the initialization section it is
* transfered to on-chip RAM program memory because the IN
* and OUT instructions execute in fewer machine cycles when
* executed from on-chip memory. A code representing the
* decisions as calculated in the decision section is sent
* through output port  0 to a decoding circuits. The line
* labeled LOC6 is really the last line of the decision
* section code. Then samples of the inphase and quadrature
* components of the input waveform are obtained through
* input ports  0 and  1 from the FIFOs.
*
*
IO    EQU     $           The label "IO" is assigned to
*                         the current program memory
*                         location, representing the
*                         beginning of the IO section.
*
*      The next four lines of code are the last part of
* the decision section of code. They are included here
* because this section of code will run from on-chip RAM
* memory, and the OUT instruction will then take only one
* instruction cycle to execute, whereas if these four lines
* were kept at the end of the decision section code which
* runs from external ROM, the OUT instruction would require
* two instruction cycles to execute.
*
LOC5  LARK    AR0,128
LOC6  MAR     *0+
      SAR     AR1,TEMP
      OUT     TEMP,PA0    Send the decision code to the
*                         decoding circuits.
*
*     The input section begins here.
*
POLL  BIOZ    POLL+OFFSET Test the BIO pin to see if
*                         sampled data is ready in the
*                         FIFOs. Repeat test until data
*                         is ready.
*
      IN      RER0,PA0    Get data samples from the FIFOs.
      IN      IMR0,PA1    PA0 is the port address of the
      IN      RER1,PA0    FIFO for the inphase arm of the
      IN      IMR1,PA1    demodulator, and PA1 is the
      IN      RER2,PA0    address of the FIFO for the
      IN      IMR2,PA1    quadrature arm.
      IN      RER3,PA0
      IN      IMR3,PA1
      IN      RER4,PA0
      IN      IMR4,PA1
      IN      RER5,PA0
      IN      IMR5,PA1
      IN      RER6,PA0
```

```
        IN      IMR6,PA1
        IN      RER7,PA0
        IN      IMR7,PA1
        IN      RER8,PA0
        IN      IMR8,PA1
        IN      RER9,PA0
        IN      IMR9,PA1
        IN      RER10,PA0
        IN      IMR10,PA1
        IN      RER11,PA0
        IN      IMR11,PA1
        IN      RER12,PA0
        IN      IMR12,PA1
        IN      RER13,PA0
        IN      IMR13,PA1
        IN      RER14,PA0
        IN      IMR14,PA1
        IN      RER15,PA0
        IN      IMR15,PA1
        IN      RER16,PA0
        IN      IMR16,PA1
        IN      RER17,PA0
        IN      IMR17,PA1
        IN      RER18,PA0
        IN      IMR18,PA1
        IN      RER19,PA0
        IN      IMR19,PA1
        IN      RER20,PA0
        IN      IMR20,PA1
        IN      RER21,PA0
        IN      IMR21,PA1
        IN      RER22,PA0
        IN      IMR22,PA1
        IN      RER23,PA0
        IN      IMR23,PA1
        IN      RER24,PA0
        IN      IMR24,PA1
        IN      RER25,PA0
        IN      IMR25,PA1
        IN      RER26,PA0
        IN      IMR26,PA1
        IN      RER27,PA0
        IN      IMR27,PA1
        IN      RER28,PA0
        IN      IMR28,PA1
        IN      RER29,PA0
        IN      IMR29,PA1
        IN      RER30,PA0
        IN      IMR30,PA1
        IN      RER31,PA0
        IN      IMR31,PA1
*
```

```
      B       DFT         Proceed to the DFT section.
*
IOEND EQU     $           The label "IOEND" is assigned to
*                         the program memory location
*                         representing the end of the IO
*                         section.
*
IOL   EQU     IOEND-IO    "IOL" is the I/O section code
*                         length.
*
OFFSET EQU    >FF00-IO    "OFFSET" is the address offset
*                         required when branching to
*                         locations within the I/O section
*                         code in on-chip RAM memory.
*
*************************************************************
*
*     DFT Section: In this section, the 32 complex input
* samples are multiplied by a window and the even ordered
* samples of the 32-point DFT are calculated.
*
*
DFT   LT      W8
      MPY     RER8
      PAC
      MPY     RER24
      LTA     W0
      SACH    REX8,1      ReX8 = 2(1/2*W8*ReR8 +
*                                 1/2*W8*ReR24)
*
      MPY     RER0
      LTA     W8
      ADD     RER16,15
      SACH    RET1,1      ReT1 = 2(1/2*ReX8 + 1/2*W0*ReR0
*                                 + 1/2*ReR16)
*
      SUBH    REX8
      SACH    REM4,1      ReM4 = 2(1/2*ReT1 - ReX8)
*
*
      MPY     IMR8
      PAC
      MPY     IMR24
      LTA     W0
      SACH    IMX8,1      ImX8 = 2(1/2*W8*ImR8 +
*                                 1/2*W8*ImR24)
*
      MPY     IMR0
      LTA     W4
      ADD     IMR16,15
      SACH    IMT1,1      ImT1 = 2(1/2*ImX8 + 1/2*W0*ImR0
*                                 + 1/2*ImR16)
```

```
*
        SUBH    IMX8
        SACH    IMM4,1          ImM4 = 2(1/2*ImT1 - ImX8)
*
*
        MPY     RER4
        LTP     W12
        MPY     RER20
        APAC
        SACH    REX4,1          ReX4 = 2(1/2*W4*ReR4 +
*                                       1/2*W12*ReR20)
*
        MPY     RER12
        LTS     W4
        MPY     RER28
        SPAC
        SACH    IMM12,1       -ImM12 = 2(1/2*ReX4 -
*                                  1/2*W12*ReR12 - 1/2*W4*ReR28)
*
        SUBH    REX4
        ADD     RET1,15
        SACH    REM3,1          ReM3 = 2(1/2*(-ImM12) - ReX4 +
*                                       1/2*ReT1)
*
        SUBH    RET1
        SACH    RET15,1       -ReT15 = 2(1/2*ReM3 - ReT1)
*
*
        MPY     IMR4
        LTP     W12
        MPY     IMR20
        APAC
        SACH    IMX4,1          ImX4 = 2(1/2*W4*ImR4 +
*                                       1/2*W12*ImR20)
*
        MPY     IMR12
        LTS     W4
        MPY     IMR28
        LTS     W14
        SACH    REM12,1       ReM12 = 2(1/2*ImX4 -
*                                  1/2*W12*ImR12 - 1/2*W4*ImR28)
*
        SUBH    IMX4
        ADD     IMT1,15
        SACH    IMM3,1          ImM3 = 2(1/2*ReM12 - ImX4 +
*                                       1/2*ImT1)
*
        SUBH    IMT1
        SACH    IMT15,1       -ImT15 = 2(1/2*ImM3 - ImT1)
*
*
        MPY     RER14
```

```
        LTP     W2
        MPY     RER30
        LTA     W6
        SACH    REX14,1     ReX14 = 2(1/2*W14*ReR14 +
*                                    1/2*W2*ReR30)
*
        MPY     RER6
        LTA     W10
        MPY     RER22
        LTA     W14
        SACH    RET5,1      ReT5 = 2(1/2*ReX14 + 1/2*W6*ReR6
*                                   + 1/2*W10*ReR22)
*
        SUBH    REX14
        SACH    RET6,1      ReT6 = 2(1/2*ReT5 - ReX14)
*
*
        MPY     IMR14
        LTP     W2
        MPY     IMR30
        LTA     W6
        SACH    IMX14,1     ImX14 = 2(1/2*W14*ImR14 +
*                                    1/2*W2*ImR*
        MPY     IMR6
        LTA     W10
        MPY     IMR22
        LTA     W2
        SACH    IMT5,1      ImT5 = 2(1/2*ImX14 + 1/2*W6*ImR6
*                                   + 1/2*W10*ImR22)
*
        SUBH    IMX14
        SACH    IMT6,1      ImT6 = 2(1/2*ImT5 - ImX14)
*
*
        MPY     RER2
        LTP     W14
        MPY     RER18
        LTA     W10
        SACH    REX2,1      ReX2 = 2(1/2*W2*ReR2 +
*                                   1/2*W14*ReR18)
*
        MPY     RER10
        LTS     W6
        MPY     RER26
        LTS     W2
        SACH    RET4,1      ReT4 = 2(1/2*ReX2 -
*                              1/2*W10*ReR10 - 1/2*W6*ReR26)
*
        SUBH    REX2
        ADD     RET5,15
        SACH    IMM11,1     ImM11 = 2(1/2*ReT4 - ReX2 +
*                                   1/2*ReT5)
```

```
*
        SUBH    RET5
        SUB     RET15,15
        SACH    REM2,1          ReM2 = 2(1/2*ImM11 - ReT5 -
*                                       1/2*(-ReT15))
*
        ADDH    RET15
        SACH    RET17,1     -ReT17 = 2(1/2*ReM2 + (-ReT15))
*
*
        MPY     IMR2
        LTP     W14
        MPY     IMR18
        LTA     W10
        SACH    IMX2,1          ImX2 = 2(1/2*W2*ImR2 +
*                                       1/2*W14*ImR18)
*
        MPY     IMR10
        LTS     W6
        MPY     IMR26
        LTS     W13
        SACH    IMT4,1          ImT4 = 2(1/2*ImX2 -
*                                  1/2*W10*ImR10 - 1/2*W6*ImR26)
*
        SUBH    IMX2
        ADD     IMT5,15
        SACH    REM11,1     -ReM11 = 2(1/2*ImT4 - ImX2 +
*                                       1/2*ImT5)
*
        SUBH    IMT5
        SUB     IMT15,15
        SACH    IMM2,1          ImM2 = 2(1/2*(-ReM11) - ImT5 -
*                                       1/2*(-ImT15))
*
        ADDH    IMT15
        SACH    IMT17,1     -ImT17 = 2(1/2*ImM2 + (-ImT15))
*
*
        MPY     RER13
        LTP     W3
        MPY     RER29
        LTA     W5
        SACH    REX13,1         ReX13 = 2(1/2*W13*ReR13 +
*                                       1/2*W3*ReR29)
*
*
        MPY     RER5
        LTA     W11
        MPY     RER21
        LTA     W13
        SACH    RET11,1         ReT11 = 2(1/2*ReX13 +
*                                  1/2*W5*ReR5 + 1/2*W11*ReR21)
*
```

```
        SUBH    REX13
        SACH    RET12,1     ReT12 = 2(1/2*ReT11 - ReX13)
*
*
        MPY     IMR13
        LTP     W3
        MPY     IMR29
        LTA     W5
        SACH    IMX13,1     ImX13 = 2(1/2*W13*ImR13 +
*                                   1/2*W3*ImR29)
*
        MPY     IMR5
        LTA     W11
        MPY     IMR21
        APAC
        SACH    IMT11,1     ImT11 = 2(1/2*ImX13 +
*                               1/2*W5*ImR5 + 1/2*W11*ImR21)
*
        SUBH    IMX13
        SACH    IMT12,1     ImT12 = 2(1/2*ImT11 - ImX13)
*
*
        MPY     RER11
        LTP     W5
        MPY     RER27
        LTA     W3
        SACH    REX11,1     ReX11 = 2(1/2*W11*ReR11 +
*                                   1/2*W5*ReR27)
*
        MPY     RER3
        LTA     W13
        MPY     RER19
        LTA     W11
        SACH    RET9,1      ReT9 = 2(1/2*ReX11 + 1/2*W3*ReR3
*                                  + 1/2*W13*ReR19)
*
        SUBH    REX11
        ADD     RET12,15
        SACH    RET25,1     ReT25 = 2(1/2*ReT9 - ReX11 +
*                                   1/2*ReT12)
*
        SUBH    RET12
        SACH    RET26,1    -ReT26 = 2(1/2*ReT25 - ReT12)
*
*
        MPY     IMR11
        LTP     W5
        MPY     IMR27
        LTA     W3
        SACH    IMX11,1     ImX11 = 2(1/2*W11*ImR11 +
*                                   1/2*W5*ImR27)
*
```

```
          MPY     IMR3
          LTA     W13
          MPY     IMR19
          LTA     W7
          SACH    IMT9,1      ImT9 = 2(1/2*ImX11 + 1/2*W3*ImR3
*                                    + 1/2*W13*ImR19)
*
          SUBH    IMX11
          ADD     IMT12,15
          SACH    IMT25,1     ImT25 = 2(1/2*ImT9 - ImX11 +
*                                     1/2*ImT12)
*
          SUBH    IMT12
          SACH    IMT26,1    -ImT26 = 2(1/2*ImT25 - ImT12)
*
*
          MPY     RER7
          LTP     W9
          MPY     RER23
          LTA     W15
          SACH    REX7,1      ReX7 = 2(1/2*W7*ReR7 +
*                                     1/2*W9*ReR23)
*
          MPY     RER15
          LTS     W1
          MPY     RER31
          LTS     W7
          SACH    RET14,1     ReT14 = 2(1/2*ReX7 -
*                                    1/2*W15*ReR15 - 1/2*W1*ReR31)
*
          SUBH    REX7
          ADD     RET9,15
          SACH    RET21,1     ReT21 = 2(1/2*ReT14 - ReX7 +
*                                     1/2*ReT9)
*
          SUBH    RET9
          SACH    RET20,1    -ReT20 = 2(1/2*ReT21 - ReT9)
*
*
          MPY     IMR7
          LTP     W9
          MPY     IMR23
          LTA     W15
          SACH    IMX7,1      ImX7 = 2(1/2*W7*ImR7 +
*                                     1/2*W9*ImR23)
*
          MPY     IMR15
          LTS     W1
          MPY     IMR31
          SPAC
          SACH    IMT14,1     ImT14 = 2(1/2*ImX7 -
*                                    1/2*W15*ImR15 - 1/2*W1*ImR31)
```

```
*
        SUBH    IMX7
        ADD     IMT9,15
        SACH    IMT21,1      ImT21 = 2(1/2*ImT14 - ImX7 +
*                                     1/2*ImT9)
*
        SUBH    IMT9
        SACH    IMT20,1     -ImT20 = 2(1/2*ImT21 - ImT9)
*
*
        MPY     RER1
        LTP     W15
        MPY     RER17
        LTA     W9
        SACH    REX1,1       ReX1 = 2(1/2*W1*ReR1 +
*                                     1/2*W15*ReR17)
*
        MPY     RER9
        LTS     W7
        MPY     RER25
        LTS     W1
        SACH    RET8,1       ReT8 = 2(1/2*ReX1 - 1/2*W9*ReR9
*                                     - 1/2*W7*ReR25)
*
        SUBH    REX1
        ADD     RET11,15
        SACH    RET19,1     -ReT19 = 2(1/2*ReT8 - ReX1 +
*                                     1/2*ReT11)
*
        SUBH    RET11
        SUB     RET20,15
        SACH    IMM10,1      ImM10 = 2(-1/2*ReT19 - ReT11 -
*                                     1/2*(-ReT20))
*
        ADDH    RET20
        SUB     RET17,15
        SACH    REF8,1       ReF(8) = 2(1/2*ImM10 + (-ReT20)
*                                      - 1/2*(-ReT17))
*
        ADDH    RET17
        SACH    REF0,1      -ReF(0) = 2(1/2*ReF(8) +
*                                      (-ReT17))
*
*
        MPY     IMR1
        LTP     W15
        MPY     IMR17
        LTA     W9
        SACH    IMX1,1       ImX1 = 2(1/2*W1*ImR1 +
*                                     1/2*W15*ImR17)
*
        MPY     IMR9
```

```
        LTS     W7
        MPY     IMR25
        LTS     SIN2U
        SACH    IMT8,1      ImT8 = 2(1/2*ImX1 - 1/2*W9*ImR9
*                                   - 1/2*W7*ImR25)
*
        SUBH    IMX1
        ADD     IMT11,15
        SACH    IMT19,1     -ImT19 = 2(1/2*ImT8 - ImX1 +
*                                     1/2*ImT11)
*
        SUBH    IMT11
        SUB     IMT20,15
        SACH    REM10,1     -ReM10 = 2(1/2*(-ImT19) - ImT11
*                                   - 1/2*(-ImT20))
*
        ADDH    IMT20
        SUB     IMT17,15
        SACH    IMF8,1      ImF(8) = 2(1/2*(-ReM10) +
*                                    (-ImT20) - 1/2*(-ImT17))
*
        ADDH    IMT17
        SACH    IMF0,1      -ImF(0) = 2(1/2*ImF(8) +
*                                     (-ImT17))
*
*
        LAC     REM2,15
        SUB     REM10,15
        SACH    REF4,1      ReF(4) = 2(1/2*ReM2 -
*                                    1/2*(-ReM10))
*
        ADDH    REM10
        SACH    REF12,1     ReF(12) = 2(1/2*ReF(4) +
*                                     (-ReM10))
*
*
        LAC     IMM2,15
        ADD     IMM10,15
        SACH    IMF4,1      ImF(4) = 2(1/2*ImM2 + 1/2*ImM10)
*
        SUBH    IMM10
        SACH    IMF12,1     ImF(12) = 2(1/2*ImF(4) - ImM10)
*
*
      MPY     IMT21
      PAC
      SACH    TEMP,1      TEMP = 2(1/2*SIN(2U)*ImT21)
      MPY     IMT19
      SPAC
      SACH    REM13,1     ReM13 = 2(1/2*SIN(2U)*ImT21 -
*                                   1/2*SIN(2U)*(-ImT19))
*
```

```
        SUBH    TEMP
        SACH    IMM5,1          ImM5 = 2(1/2*ReM13 -
*                                       SIN(2U)*ImT21)
*
*
*
        MPY     RET19
        PAC
        SACH    TEMP,1          TEMP = 2(1/2*SIN(2U)*(-ReT19))
        MPY     RET21
        LTS     SINU
        SACH    IMM13,1         ImM13 = 2(1/2*SIN(2U)*(-ReT19) -
*                                        1/2*SIN(2U)*ReT21)
*
*
        SUBH    TEMP            1/2*ReM5 = 1/2*ImM13 -
*                                           SIN(2U)*(-ReT19)
*
        ADD     REM3,15
        SUB     REM11,15
        ADD     REM13,15
        SACH    REF2,1          ReF(2) = 2(1/2*ReM5 + 1/2*ReM3 -
*                                        1/2*(-ReM11) + 1/2*ReM13)
*
        ADDH    REM11
        SUBH    REM13
        SACH    REF14,1         ReF(14) = 2(1/2*ReF(2) +
*                                          (-ReM11) - ReM13)
*
        SUBH    REM3
        SUBH    REM11
        SACH    REF6,1         -ReF(6) = 2(1/2*ReF(14) - ReM3 -
*                                         (-ReM11)
*
        ADDH    REM11
        ADDH    REM13
        SACH    REF10,1        -ReF(10) = 2(1/2*(-ReF(6)) +
*                                          (-ReM11) + ReM13)
*
*
        LAC     IMM5,15
        ADD     IMM3,15
        ADD     IMM11,15
        ADD     IMM13,15
        SACH    IMF2,1          ImF(2) = 2(1/2*ImM5 + 1/2*ImM3 +
*                                         1/2*ImM11 + 1/2*ImM13)
*
        SUBH    IMM11
        SUBH    IMM13
        SACH    IMF14,1         ImF(14) = 2(1/2*ImF(2) - ImM11 -
*                                           ImM13)
*
        SUBH    IMM3
        ADDH    IMM11
```

```
        SACH    IMF6,1      -ImF(6) = 2(1/2*ImF(14) - ImM3 +
*                                     ImM11)
*
        ADDH    IMM13
        SUBH    IMM11
        SACH    IMF10,1     -ImF(10) = 2(1/2*(-ImF(6)) + Im13
*                                      - ImM11)
*
*
        LAC     RET8,15
        SUB     RET14,15
        SACH    RET24,1     ReT24 = 2(1/2*ReT8 - 1/2*ReT14)
*
        ADDH    RET14
        SACH    RET23,1     ReT23 = 2(1/2*ReT24 + ReT14)
*
*
        LAC     IMT8,15
        SUB     IMT14,15
        SACH    IMT24,1     ImT24 = 2(1/2*ImT8 - 1/2*ImT14)
*
        ADDH    IMT14
        SACH    IMT23,1     ImT23 = 2(1/2*ImT24 + ImT14)
*
*
        MPY     IMT23
        LTP     SIN3U
        MPY     IMT25
        SPAC
        SACH    RES15,1     -ReS15 = 2(1/2*(-SIN(U))*ImT23 -
*                                      1/2*SIN(3U)*ImT25)
*
        MPY     RET25
        LTP     SINU
        MPY     RET23
        LTS     SIN3U
        SACH    IMS15,1     -ImS15 = 2(1/2*SIN(3U)*ReT25 -
*                                     1/2*(-SIN(U))*ReT23)
*
*
        MPY     IMT23
        LTP     SINU
        MPY     IMT25
        APAC
        SACH    RES16,1     ReS16 = 2(1/2*SIN(3U)*ImT23 +
*                                     1/2*(-SIN(U))*ImT25)
*
*
*
        MPY     RET25
        LTP     SIN3U
        MPY     RET23
```

```
        LTA     SIN2U
        SACH    IMS16,1     -ImS16 = 2(1/2*(-SIN(U))*ReT25
*                                    + 1/2*SIN(3U)*ReT23)
*
*
        MPY     IMT6
        PAC
        SACH    TEMP,1       TEMP = 2(1/2*SIN(2U)*ImT6)
        MPY     IMT4
        SPAC
        SACH    IMM6,1      -ImM6 = 2(1/2*SIN(2U)*ImT6 -
*                                   1/2*SIN(2U)*ImT4)
*
        SUBH    TEMP        -1/2*ReM14 = -1/2*ImM6 -
*                                        SIN(2U)*ImT6
*
        SUB     REM12,15
        SACH    RES13,1     -ReS13 = 2(-1/2*ReM14 -
*                                     1/2*ReM12)
*
        ADDH    REM12
        ADD     RES16,15
        SACH    RES19,1      ReS19 = 2(-1/2*ReS13 + ReM12 +
*                                     1/2*ReS16)
*
        SUBH    RES16
        SACH    RES20,1      ReS20 = 2(1/2*ReS19 - ReS16)
*
*
        MPY     RET6
        PAC
        SACH    TEMP,1       TEMP = 2(1/2*SIN(2U)*ReT6)
        MPY     RET4
        LTS     COS3U
        SACH    REM6,1      -ReM6 = 2(1/2*SIN(2U)*ReT6 -
*                                   1/2*SIN(2U)*ReT4)
*
        SUBH    TEMP         1/2*ImM14 = -1/2*ReM6 -
*                                        SIN(2U)*ReT6
*
        SUB     IMM12,15
        SACH    IMS13,1      ImS13 = 2(1/2*ImM14 -
*                                     1/2*(-ImM12))
*
        ADDH    IMM12
        ADD     IMS16,15
        SACH    IMS19,1     -ImS19 = 2(1/2*ImS13 + (-ImM12) +
*                                     1/2*(-ImS16))
        SUBH    IMS16
        SACH    IMS20,1     -ImS20 = 2(-1/2*ImS19 - (-ImS16))
*
*
```

```
          LAC     REM4,15
          SUB     REM6,15
          SACH    RES5,1      ReS5 = 2(1/2*ReM4 - 1/2*(-ReM6))
   *
          LAC     IMM4,15
          SUB     IMM6,15
          SACH    IMS5,1      ImS5 = 2(1/2*ImM4 - 1/2*(-ImM6))
   *
   *
          MPY     RET24
          LTP     COSU
          MPY     RET26
          APAC
          SACH    RES8,1      ReS8 = 2(1/2*(-COS(3U))*ReT24 +
   *                                  1/2*COS(U)*(-ReT26))
   *
          MPY     RET24
          LTP     COS3U
          MPY     RET26
          SPAC
   *                          1/2*ReS7 = 1/2*COS(U)*ReT24 -
   *                                  1/2*(-COS(3U))*(-ReT26)
   *
          ADD     RES5,15
          SUB     RES13,15
          SUB     RES15,15
          SACH    REF1,1      ReF(1) = 2(1/2*ReS7 + 1/2*ReS5 -
   *                              1/2*(-ReS13) - 1/2*(-ReS15))
   *
          ADDH    RES13
          ADDH    RES15
          SACH    REF15,1     ReF(15) = 2(1/2*ReF(1) +
   *                                  (-ReS13) + (-ReS15))
   *
          SUBH    RES5
          SUBH    RES15
          SACH    REF9,1     -ReF(9) = 2(1/2*ReF(15) - ReS5 -
   *                                  (-ReS15))
   *
          SUBH    RES13
          ADDH    RES15
          SACH    REF7,1     -ReF(7) = 2(-1/2*ReF(9) -
   *                                  (-ReS13) + (-ReS15))
   *
   *
   *
          MPY     IMT24
          LTP     COSU
          MPY     IMT26
          APAC
          SACH    IMS8,1      ImS8 = 2(1/2*(-COS(3U))*ImT24 +
*                                   1/2*COS(U)*(-ImT26))
```

```
*
        MPY     IMT24
        LTP     COS3U
        MPY     IMT26
        SPAC
*                               1/2*ImS7 = 1/2*COS(U)*ImT24 -
*                                          1/2*(-COS(3U))*(-ImT26)
*
        ADD     IMS5,15
        ADD     IMS13,15
        SUB     IMS15,15
        SACH    IMF1,1          ImF(1) = 2(1/2*ImS7 + 1/2*ImS5 +
*                                         1/2*ImS13 - 1/2*(-ImS15))
*
        SUBH    IMS13
        ADDH    IMS15
        SACH    IMF15,1         ImF(15) = 2(1/2*ImF(1) -ImS13 +
*                                           (-ImS15))
*
        SUBH    IMS5
        SUBH    IMS15
        SACH    IMF9,1         -ImF(9) = 2(1/2*ImF(15) - ImS5 -
*                                         (-ImS15))
*
        ADDH    IMS13
        ADDH    IMS15
        SACH    IMF7,1         -ImF(7) = 2(-1/2*ImF(9) + ImS13 +
*                                         (-ImS15))
*
*
*
        LAC     RES5,15
        ADDH    REM6
        SUB     RES8,15
        ADD     RES20,15
        SACH    REF13,1         ReF(13) = 2(1/2*ReS5 + (-ReM6) -
*                                         1/2*ReS8 + 1/2*ReS20)
*
        SUBH    RES20
        SACH    REF3,1          ReF(3) = 2(1/2*ReF(13) - ReS20)
*
        ADDH    RES8
        SUBH    RES16
        SACH    REF11,1         ReF(11) = 2(1/2*ReF(3) + ReS8 -
*                                         ReS16)
*
        ADDH    RES19
        SACH    REF5,1          ReF(5) = 2(1/2*ReF(11) + ReS19)
*
*
        LAC     IMS5,15
        ADDH    IMM6
```

```
        SUB     IMS8,15
        SUB     IMS20,15
        SACH    IMF13,1     ImF(13) = 2(1/2*ImS5 + (-ImM6) -
*                                     1/2*ImS8 - 1/2*(-ImS20))
*
        ADDH    IMS20
        SACH    IMF3,1      ImF(3) = 2(1/2*ImF(13) +
*                                     (-ImS20))
*
        ADDH    IMS8
        ADDH    IMS16
        SACH    IMF11,1     ImF(11) = 2(1/2*ImF(3) + ImS8 +
*                                      (-ImS16))
*
        SUBH    IMS19
        SACH    IMF5,1      ImF(5) = 2(1/2*ImF(11) -
*                                     (-ImS19))
*
*
*       This completes the DFT section of code.
*
*
*************************************************************
*
*       Decision Section: In this section, the square
* magnitude of the output of the DFT is computed and a
* decision is made as to which tones were present. It is
* assumed that the input is made up of four 4-ary FSK
* signals multiplexed in frequency. The DFT outputs are
* related to the complex baseband locations of the signaling
* bins as follows:
*
*       F(0) → 0 kHz            F(8) → 320 kHz
*       F(1) → 40 kHz           F(9) → -280 kHz
*       F(2) → 80 kHz           F(10) → -240 kHz
*       F(3) → 120 kHz          F(11) → -200 kHz
*       F(4) → 160 kHz          F(12) → -160 kHz
*       F(5) → 200 kHz          F(13) → -120 kHz
*       F(6) → 240 kHz          F(14) → -80 kHz
*       F(7) → 280 kHz          F(15) → -40 kHz
*
*       Auxilliary register 1 is used for the temporary
* storage of the decision results as decisions are made
* for the next channel. The least two significant bits
* represent the decoded symbol for channel 1, the next two
* represent the decoded symbol for channel 2, the next two
* represent the decoded symbol for channel 3, and the next
* two represent the decoded symbol for channel 4. Eight
* bits in total are required for the output. Axilliary
* register 0 is used as a temporary scratch pad in the
* decision making.
*
```

```
*     Decision for channel 1:
*
      SQRA    REF9        Compare |F(9)|^2 with |F(10)|^2.
      ZAC
      SQRA    IMF9
      SQRA    REF10
      SQRS    IMF10
      SPAC
      BGZ     LOC1        Go to LOC1 if |F(9)| is the
*                           larger, otherwise:
      LARK    AR1,1       Load axilliary register with data
*                           corresponding to symbol with
*                           frequency of F(10).
      ZAC
TEST11        SQRA    REF10       Compare the greater of the
      SQRA    REF11       above with |F(11)|^2.
      SQRS    IMF11
      SPAC
      BGZ     TEST12      Go to next test if |F(11)| is
*                           smaller, otherwise:
      LARK    AR1,3       Load axilliary register with data
*                           corresponding to symbol with
*                           frequency of F(11).
      ZAC
TEST12        SQRA    REF11       Compare the greater of the
      SQRA    REF12       above with |F(12)|^2.
      SQRS    IMF12
      SQRS    REF13
      BGZ     TEST13      Go to next test if |F(12)| is
*                           smaller, otherwise:
      LARK    AR1,2       Load axilliary register with data
*                           corresponding to symbol with
*                           frequency of F(12).
*
*     Decision for channel 2:
*
TEST13        ZAC                 Compare |F(13)|^2 with |F(14)|^2.
      SQRA    IMF13
      SQRA    REF14
      SQRS    IMF14
      SPAC
      BGZ     LOC2        Go to LOC2 if |F(13)| is the
*                           larger, otherwise:
      LARK    AR0,4       Load axilliary register with data
*                           corresponding to symbol with
*                           frequency of F(14).
      ZAC
TEST15        SQRA    REF14       Compare the greater of the
      SQRA    REF15       above with |F(15)|^2.
      SQRS    IMF15
      SPAC
      BGZ     TEST0       Go to next test if |F(15)| is
```

```
*                                    smaller, otherwise:
        LARK    AR0,12      Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(15).
        ZAC
TEST0   SQRA    REF15       Compare the greater of the
        SQRA    REF0        above with |F(0)|^2.
        SQRS    IMF0
        SQRS    REF1
        BGZ     TEST1       Go to next test if |F(0)| is
*                             smaller, otherwise:
        LARK    AR0,8       Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(0).
*
*     Decisions for channel 3:
*
TEST1   ZAC                 Compare |F(1)|^2 with |F(2)|^2.
        MAR     *0+
        SQRA    IMF1
        SQRA    REF2
        SQRS    IMF2
        SPAC
        BGZ     LOC3        Go to LOC3 if |F(1)| is the
*                             larger, otherwise:
        LARK    AR0,16      Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(2).
        ZAC
TEST3   SQRA    REF2        Compare the greater of the
        SQRA    REF3        above with |F(3)|^2.
        SQRS    IMF3
        SPAC
        BGZ     TEST4       Go to next test if |F(3)| is
*                             smaller, otherwise:
        LARK    AR0,48      Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(3).
        ZAC
TEST4   SQRA    REF3        Compare the greater of the
        SQRA    REF4        above with |F(4)|^2.
        SQRS    IMF4
        SQRS    REF5
        BGZ     TEST5       Go to next test if |F(4)| is
*                             smaller, otherwise:
        LARK    AR0,32      Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(4).
*
*     Decisions for channel 4:
*
TEST5   ZAC                 Compare |F(5)|^2 with |F(6)|^2.
```

```
        MAR     *0+         Store result for the previous
*                             channel.
        SQRA    IMF5
        SQRA    REF6
        SQRS    IMF6
        SPAC
        BGZ     LOC4        Go to LOC4 if |F(5)| is the
*                             larger, otherwise:
        LARK    AR0,64      Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(6).
        ZAC
TEST7   SQRA    REF6        Compare the greater of
        SQRA    REF7        the above with |F(7)|^2.
        SQRS    IMF7
        SPAC
        BGZ     TEST8       Go to next test if |F(7)| is
*                             smaller, otherwise:
        LARK    AR0,192     Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(7).
        ZAC
TEST8   SQRA    REF7        Compare the greater of
        SQRA    REF8        the above with |F(8)|^2.
        SQRS    IMF8
        SPAC
        BLZ     LOC5+OFFSET   Go to LOC5 if |F(8)| is
*                               the greater, otherwise:
        B       LOC6+OFFSET   Go to LOC6.
*
LOC1    LARK    AR1,0       Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(9).
        B       TEST11      Go to next test.
*
LOC2    LARK    AR0,0       Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(13).
        B       TEST15      Go to next test.
*
LOC3    LARK    AR0,0       Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(1).
        B       TEST3       Go to next test.
*
LOC4    LARK    AR0,0       Load axilliary register with data
*                             corresponding to symbol with
*                             frequency of F(5).
        B       TEST7       Go to next test.
*
        END
```

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section B.)

Communications Research Center
Directorate of Satellite Communications
3701 Carling Ave.
Ottawa, Ontario, K2H 8S2

2. SECURITY CLASSIFICATION (overall security classification of the document. including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title.)

A BLOCK DEMODULATOR FOR FREQUENCY DIVISION MULTIPLEXED NCFSK SIGNALS

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Shaw, M.D., Pearce, J.L., Wight, J.S.

5. DATE OF PUBLICATION (month and year of publication of document)

July 1987

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

117

6b. NO. OF REFS (total cited in document)

14

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO Technical Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DREO via CRC

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

041LG14 0835 65614

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

CRC REPORT 1427

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally corraspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Same as 11.

ABSTRACT ( a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

A block demodulator for the noncoherent demodulation of M-ary FSK signals multiplexed in frequency has been built and tested. The various users' signals are separated and demodulated with the use of the discrete Fourier transform. The demodulator is capable of handling 16/M users. The bit error performance in the presence of system noise has been measured for M = 2-, 4-, 8-, and 16-ary FSK with a symbol period of 50 μs. The measured implementation loss depends on the specific channel location within the demodulator's bandwidth, varying from 0.2 dB for the best channel allocations to 2.6 dB for the worst. The effects of frequency offsets and adjacent channel interference are characterized with both a rectangular and a Kaiser-Bessel ($\alpha$ = 1.4) window. For small frequency errors the rectangular window still performs well, but for larger offsets or large differences in power levels between the users the Kaiser-Bessel window is better.

. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible, keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

- demodulator
- M-ary FSK
- discrete Fourier transform
- digital signal processing
- on-board processing
- signal-processing satellites
- digital satellite communications